CRANFIELD UNIVERSITY

SAMUEL THOMAS WESTLAKE

The Application of Deep Learning Algorithms to Longwave Infrared Missile Seekers

SCHOOL OF DEFENCE AND SECURITY
Centre for Electronic Warfare, Information, and Cyber

Doctor of Philosophy

CRANFIELD UNIVERSITY

SCHOOL OF DEFENCE AND SECURITY
Centre for Electronic Warfare, Information, and Cyber

Doctor of Philosophy

February 2018 – December 2021

SAMUEL THOMAS WESTLAKE

The Application of Deep Learning Algorithms to Longwave Infrared Missile Seekers

Supervisor: Dr David B. James
December 2021

Abstract

Convolutional neural networks (CNNs) have already surpassed human-level performance in complex computer vision applications, and can potentially significantly advance the performance of infrared anti-ship guided missile seeker algorithms. But the performance of CNN-based algorithms is very dependent on the data used to optimise them, typically requiring large sets of fully-annotated real-world training examples.

Across four technical chapters, this thesis addresses the challenges involved with applying CNNs to longwave infrared ship detection, recognition, and identification.

The absence of suitable longwave infrared training data was addressed through the synthetic generation of a large, thermally-realistic dataset of 972,000 fully-labelled images of military ships with varying seascapes and background clutter. This dataset—IRShips—is the largest openly available repository of such images worldwide.

Configurable automated workflow pipelines significantly enhance the development of CNN-based algorithms. No such tool was available when this body of work began, so an integrated modular deep learning development environment—Deeplodocus—was created. Publicly-available, it now features among the top 50% of packages on the Python Package Index repository.

Using Deeplodocus, the fully-convolutional one-stage YOLOv3 object detection algorithm was trained to detect ships in a highly-cluttered sequence of real-world longwave infrared imagery. Further enhancement of YOLOv3 resulted in an F-score of 0.945 being achieved, representing the first time synthetic data has been used to train a CNN algorithm to successfully detect military ships in longwave infrared imagery.

Benchmarking YOLOv3's detection accuracy against two alternative CNNs—Faster R-CNN and Mask R-CNN—using visual-spectrum and near-infrared data from the Singapore Maritime dataset, showed that YOLOv3 was three times faster, but 3% less accurate than Mask R-CNN. Modifying YOLOv3 through the use of spectral domain-dependent encoding delivered state-of-the-art accuracy with respect to the near-infrared test data, while maintaining YOLOv3's considerable speed advantage.

Acknowledgements

I would like to give sincere thanks to my supervisor, Dr David James, for his guidance, support, and technical expertise. In particular, thank you for sharing your in-depth knowledge of electro-optical and infrared systems.

I'd also like to thank MBDA UK for funding this research, and in particular, Tim, James, and Andy for their considered advice and constructive feedback throughout.

I would like to thank Ioannis, who's help and support to get me up to speed with Rhinoceros 3D was invaluable.

I'd also like to take the opportunity to thank my colleagues at CDS: Akhil, Alix, Amélie, Brandon, Eddie, Gareth, James, Karthik, Leon, Lounis, and Marco, for making my time at Cranfield so enjoyable.

I'd like to thank my friend, Malcolm, for all his kind words and encouragement.

And, finally, I'd like to give special thanks to my partner, Anna, for her endless encouragement and advice. I could not have completed this thesis without her support.

Preface

Before I began this PhD in 2018, I was not well versed in putting words to paper, so I knew that writing this thesis would be the most difficult undertaking of my life so far. Now, on the final day of 2021, I feel incredibly proud to be typing these, the final words of a near four-year journey from mister to doctor.

Reflecting on this journey, I realise how fortunate I have been for this opportunity to explore my profound interest in machine learning and computer vision—an interest which began during my Master's degree a year earlier when I trained an artificial neural network to detect malaria parasites in thin-film blood smears.

Machine learning is fascinating, not only because you can write computer programs that are able to learn, but because these programs often spot patterns that even the most intelligent of humans cannot.

A particularly exciting example of this is MIT's recent deep learning approach to drug discovery, in which a neural network identified eight new compounds with antibacterial properties, despite these compounds being structurally divergent from conventional antibiotics.

I am therefore delighted to have made my own contributions to machine learning, and also to the field of electronic warfare, particularly in light of Richard Moore's recent speech on the imminent prominence of artificial intelligence in warfare.

Table of Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ATR	Automatic Target Recognition
AVI	Audio Visual Interleave
CAD	Computer-aided Design
CORD	Cranfield Online Research Data
CSV	Comma-separated Values
CNN	Convolutional Neural Network
DIRCM	Directional Infrared Countermeasures
FCN	Fully-connected Network
FIR	Far Infrared
GMOD	General Maritime Object Detection
GPS	Global Positioning System
HMNB	Her Majesty's Naval Base
IIR	Imaging Infrared
INS	Inertial Navigation System
IV	Open Inventor Scene Graph
MV	Motor Vessel
LWIR	Long-wave Infrared
MWIR	Medium-wave Infrared
NIR	Near-infrared
RoI	Region of Interest
RORO	Roll-on/roll-off
SMD	Singapore Maritime Dataset
SRBOC	Super Rapid Bloom Offboard Countermeasures
SWIR	Short-wave Infrared
USS	United States Ship
VRML	Virtual Reality Modelling Language

Chapter 1

1. General Introduction

This thesis focuses on the development of missile seekers, a guided missile sub-system tasked with manoeuvring the missile onto its target. Although all missile seekers assume this same general function, each missile seeker is unique to specific engagement conditions. From intercepting highly manoeuvrable supersonic fighters, to disabling naval vessels, missile seekers employ various guidance methods to meet the challenges of their intended role. Numerous types of guidance methods exist, such as control guidance, active and semi-active radar homing, and laser beam-riding; this thesis is concerned with the infrared guidance of anti-ship missiles, and specifically with the detection and recognition of military vessels under varying environmental conditions and engagement scenarios.

Any modern infrared anti-ship missile seeker has two key components: its infrared imager, and its target detection algorithm. The imager captures the infrared radiation emitted from the target, which can often be combined with background noise in the shape of environmental factors such as weather conditions, background clutter, and countermeasures. Complicating this is that the detection of the military vessels involves distinguishing military vessels from civilian ships such as car ferries and container ships, and must also take place in various orientations—ships steaming towards the missile, ships steaming away from the missile, and ships viewed sideways-on. Seeker algorithms are obviously central to the effectiveness and potency of anti-ship missiles, and this thesis presents a considerable body of work that has been undertaken to improve the accuracy and robustness of seeker algorithms.

Specifically, this thesis aims to improve the accuracy and robustness of infrared anti-ship guided missile seeker algorithms—how they detect ships, how they classify those ships into targets and non-targets, and how they do this in a variety of environmental conditions. Accordingly, within this chapter, the key

challenges that seeker systems must overcome are identified, and based on the current literature, Convolutional Neural Networks (CNNs) are posited as the most promising solution to these challenges. In subsequent chapters, this thesis will explore four obstacles to the application of CNNs within infrared anti-ship missile seekers, before demonstrating—through the course of four technical chapters—how each in turn can be overcome. These four technical chapters form the central contributions of this thesis to the improvement of infrared anti-ship missile seeker algorithms, with each addressing a separate aspect of this improvement.

1.1. Guided anti-ship missiles in context

Launched from a Dassault-Breguet Super Étendard flying some 15 miles away, the Exocet AM-39 missile that sunk HMS *Sheffield* in a surprise attack shortly after the outbreak of the Falklands War in 1982 underscored to the Royal Navy the devastating effectiveness of anti-ship missiles [1].

Not surprisingly, the UK government went to considerable lengths to try to destroy Argentina's remaining stock of Exocet missiles [2] and to ensure that it could acquire no further missiles [3]. UK special forces planned an operation to destroy the Exocet missiles stored at an Argentine airbase, although bad weather impeded its actual execution [4]. Additionally, MI6 agents posed as *bona fide* purchasers of military equipment on the international market in order to outbid any Argentinian delegates [5, 6], while UK diplomats dissuaded France from selling Exocet missiles to Peru, fearing that they would be passed onto Argentina [7].

Whilst a powerful 'wake up' call to western navies, the sinking of HMS *Sheffield* is not an isolated incident. Since the first ship-launched anti-ship missile attack in 1967 [8], anti-ship missiles have been used with devastating effect in conflicts around the globe, including the Indo-Pakistan war of 1971 [9], the Arab-Israeli war of 1973 [9], the Iran-Iraq war of 1980–1987 [9], and the Persian Gulf war of 1990–1991 [9]. More recently, it has been claimed that the Russian Navy fired a P-120 Malakhit anti-ship missile at the Georgian missile boat *Giorgi Torelli*,

sinking it in the Black Sea off the coast of Abkhazia during the Russo-Georgian war of 2008 [10, 11]. And in the present geopolitical arena, the potency of Chinese anti-ship missiles is cited as a constraint on America's ability to exert military control of the South China Sea in the event of hostilities [12]. Warships are a primary asset of any navy, yet history has shown time and time again how a single well-deployed anti-ship missile can completely disable them.

1.1.1. What exactly *is* a guided missile?

Anti-ship guided missiles may be thought of as falling into two main categories: short-range missiles, which lock-on to their target before launch, and medium- to long-range missiles which lock-on after launch, when they are close enough to the target for a precise lock to be established and maintained.

A typical example of a short-range missile might be an airborne missile launched from a platform such a jet aircraft or helicopter. The pilot or weapons officer would activate a missile, 'instructing' it to identify and lock-on to a target. Once target-lock is acquired, the missile is launched.

Platforms launching these weapons must venture close to their target—often within 9 miles—and remain in close proximity until the seeker locks-on and the missile is fired [13]. This can pose significant risk to the launch platform, as every moment spent in close proximity to the defending vessel increases the likelihood of the platform being detected and counter attacked. Consequently, the survivability of the launch platform depends significantly on the range and speed at which the missile can detect its intended target.

Figure 1-1: *Launch of a Sea Skua lightweight anti-ship missile from a Westland Lynx helicopter during a two-week trial period at the Aberporth Range, UK in 2006. (Image sourced from [14].)*

Medium- to long-range anti-ship missiles, on the other hand, are launched from distances which far exceed the practical detection range of a missile. The USA's Harpoon anti-ship missile, for instance, which can be seen in Figure 1-2, is a shipborne missile with a typical range of 75 miles (120 kilometres) [15]. Russia's DF-21D anti-ship missile, also shipborne, has a range estimated at 900 miles (1,500 kilometres) [16]. As locking-on before launch is not an option for these missiles, these missiles must lock-on after launch, having first navigated to the general area where a target is expected to be found.

Figure 1-2: *Test firing of a Harpoon anti-ship missile by Royal Navy warship HMS Montrose off the coast of Scotland, 2013. (Image sourced from [17].)*

As shown in Figure 1-3, anti-ship missiles consist of a number of sub-systems: the propulsion, navigation, control, and seeker systems. Together with the missile's payload—its warhead—these are all packaged within the missile's airframe.

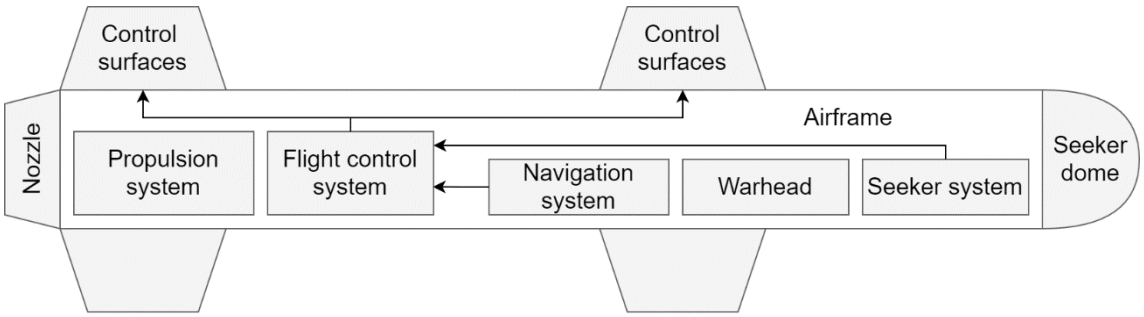


Figure 1-3: *Illustration of the key sub-systems contained within a modern guided missile.*

The **propulsion system** provides thrust to the missile, enabling it to accelerate rapidly and maintain velocity until the target is intercepted. It is usually provided by two stages; first a booster motor generates extreme thrust for a matter of seconds in order to attain the desired altitude and velocity before being jettisoned. Next, a more efficient sustainer motor ignites and maintains speed

until the target is reached. As shown in Table 1-1, propulsion can either be generated by an air-breathing engine, such as a ramjet, turbojet, or turbofan; or alternatively a rocket engine with liquid or solid propellant [18, 19].

Once launched, the missile may need to navigate tens—or possibly hundreds—of miles until it is close enough to the potential target for its seeker to lock-on and begin providing the guidance commands necessary for the missile to close in on its target. In such scenarios, initial guidance is provided by the navigation system, which is responsible for guiding the missile—typically via an Inertial Navigation System (INS) or Global Positioning System (GPS), or an integrated combination of the two—along a pre-defined route until it reaches the expected vicinity of the potential target [20, 21, 22].

Guidance commands are executed by the **flight control system**, which is responsible for manipulating the missile's flight control surfaces through a series of actuators [23, 24] in order to achieve the desired flightpath. As illustrated in Figure 1-4, the required adjustments are issued in the form of control commands, with the resultant motion being measured by onboard sensors known as Inertial Measurement Units (IMUs), which monitor the translational acceleration and angular velocity of the missile. These measurements are relayed to the autopilot which, based on the current dynamic state of the missile and the manoeuvres requested by the guidance commands, calculates the appropriate control commands. A feedback loop then detects if the control commands have delivered the desired effect, and if not, subsequent control commands are issued [23].

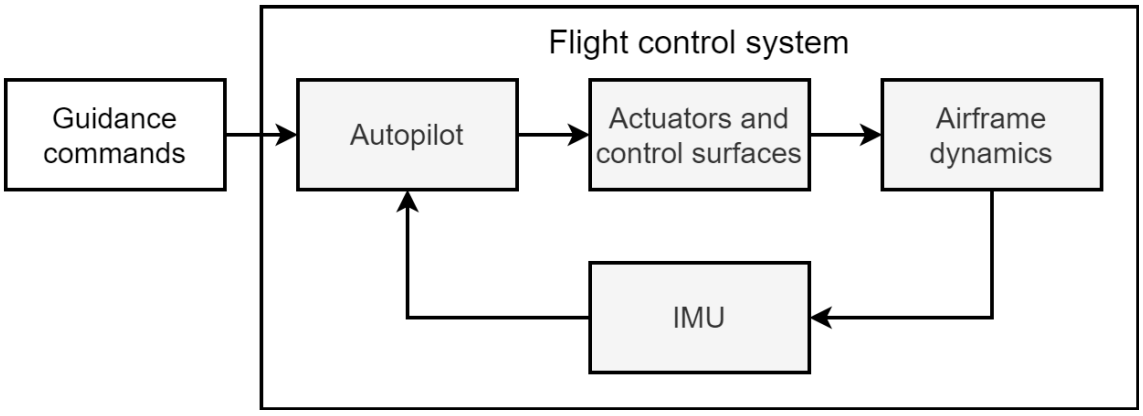


Figure 1-4: *The fundamental elements of a missile flight control system. (Diagram adapted from [23].)*

When the missile is close enough to the potential target, for terminal guidance the **seeker system** is activated, acquiring guidance-lock, and assuming control of the missile for the final approach onto the target. Modern anti-ship missiles use sophisticated seeker systems to guide missiles onto their targets, employing a target detection algorithm to pinpoint the exact location of the target from amongst the clutter of background noise. As seen in Table 1-1, the primary methods of terminal guidance provided by the seeker system are active radar, semi-active radar, and imaging infrared (IIR), which will be discussed further in Section 1.1.2.

In the final moments of the engagement, the anti-ship missile will detonate its **warhead**. This is initiated by either physical contact with the target, or via a proximity fuse which detonates once the target is within a specified distance [25]. Of the types of warhead fitted to guided missiles, blast warheads and fragmentation warheads are those most commonly employed [26]. Blast warheads comprise a high-explosive charge, which once detonated, generates a shock wave that damages the target [26]; fragmentation warheads comprise a high explosive core surrounded by a casing of high-density metal, such as iron or titanium [26].

Clearly, then, the seeker system plays a very important role in the overall performance and effectiveness of an anti-ship missile. It is the seeker system (and its algorithm) that enables the missile to home in on its target, and it is the

seeker system's algorithm that 'decides' in a time measured in milliseconds and in the absence of a human operator and human intervention what precisely constitutes a viable target (and what does not).

There is a corollary that stems from this, however. For it logically follows that incremental improvements made to seeker system algorithms hold the potential to have a return on investment that is out of all proportion to their likely cost.

A faster engine is to no avail if the missile reaches its intended target area more quickly, but fails to identify, and engage with, a suitable target ship. To choose another example, an improved navigation system may again help the missile to reach its intended target area more quickly and more accurately, but the benefit of this is again squandered if the missile fails to identify, and engage with, a suitable target ship. Improve a seeker system's algorithm, though, and the payback is undoubted.

For example, during the Falklands War, two Royal Navy Lynx helicopters engaged the patrol boat ARA *Alférez Sobral* [27]. A Sea Skua anti-ship missile was fired, but failed to recognize the vessel and instead passed harmlessly over the ship [27]. It took three further Sea Skuas to destroy the vessel: each directly impacted the superstructure of the vessel, but the fact remains that the first missile did not [27].

Continued development of seeker systems and their algorithms is therefore highly desirable, delivering relatively inexpensive improvements that directly affect the effectiveness of anti-ship missiles.

Table 1-1: *The maximum speed and range, and methods of propulsion, navigation, and terminal guidance for a selection of currently serving anti-ship missiles.*

Anti-ship Missile	Maximum Speed (Mach number)	Maximum Range (km)	Booster Motor	Sustainer Motor	Method of Navigation	Method of Terminal Guidance
ASM-2 [28]	0.9	150	None	Turbojet	INS	IIR
ASM-3 [29]	3	200	Solid rocket	Ramjet	INS / GPS	Active radar and IIR
BrahMos [30]	2	292	Solid rocket	Ramjet	INS	Active radar
DF-21D [16]	Undisclosed	1,500	Solid rocket	Solid rocket	INS / Satellite guided	Active radar and IIR (Unconfirmed)
Exocet MM40 Block 2 [31]	0.9	75	Solid rocket	Solid rocket	INS	Active radar
Exocet MM40 Block 3 [31]	0.9	200	Solid rocket	Turbojet	INS / GPS	Active radar
Harpoon [15]	0.85	124	Solid rocket	Turbojet	INS / GPS	Active radar
Hsiung Feng II [32]	0.85	150	Solid rocket	Turbojet	INS	Active radar and IIR
Hsiung Feng III [32]	2 – 2.3	200	Solid rocket	Ramjet	INS	Active radar and IIR
Naval Strike Missile [33]	0.95	200	Solid rocket	Turbojet	INS	Dual-band IIR
Penguin Mk 2 [34]	0.8	34	Solid rocket	Solid rocket	INS	IIR
Sea Eagle [35]	0.85	110	Solid rocket	Turbojet	INS	Active radar
Sea Skua [13]	0.85	20	Solid rocket	Solid rocket	None	Semi-active radar
Standard Missile-6 [36]	3	370	Solid rocket	Solid rocket	INS / GPS-aided INS	Semi-active and active radar
YJ-83 [37]	0.9	180	Solid rocket	Turbojet	INS	Active radar
Zircon [38]	8	> 500	Undisclosed	Liquid rocket	Undisclosed	Undisclosed

1.1.2. Terminal guidance methods

For terminal guidance, anti-ship missiles' seeker methodologies fall into three 'classes': active radar seeker systems, semi-active radar seeker systems, and imaging infrared missile seekers.

In active radar seeker systems, illustrated in Figure 1-5, the missile itself sends and receives radar signals that enable it to home in on targets. Generated by a transmitter onboard the missile, these energy waves propagate through the atmosphere, reflected back by objects in the environment and, eventually, a portion of the original signal is echoed back to the missile. This return signal—composed of reflections from both and targets and background clutter—is detected by a receiver and subsequently processed to determine the relative kinematic properties of the target. This mode of operation does not rely on signals from any external platforms and, consequently, active radar guided missiles provide a fire-and-forget capability.

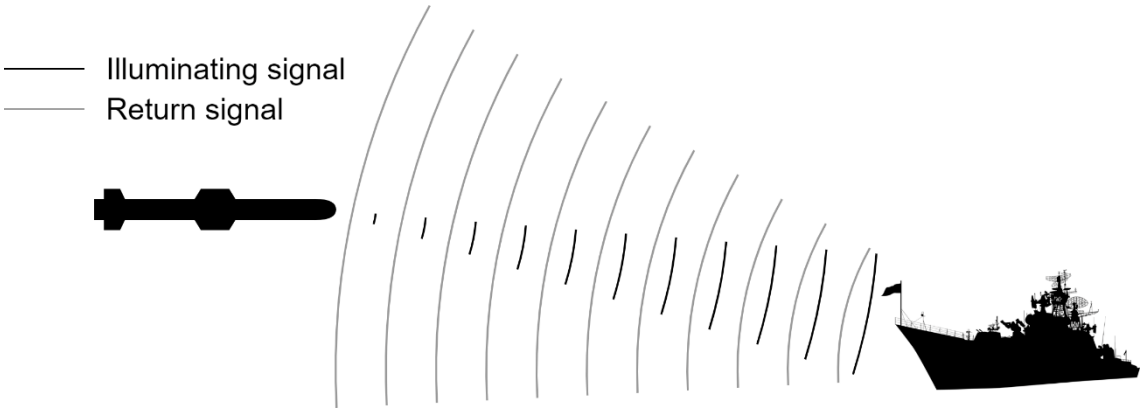


Figure 1-5: *Illustration of an active radar guidance anti-ship missile. Both the radar transmitter and receiver are packaged on-board the missile, usually in the form of a single transceiver unit.*

In semi-active radar seeker systems, on the other hand, radar signals are transmitted from an external source acting as the missile launch platform, such as a coastal battery, fixed- or rotary-wing aircraft, or ship [39], as shown in Figure 1-6. As with active radar guidance, the transmitted signal is reflected by surrounding objects and a portion of the signal is echoed back and detected by a receiver onboard the missile.

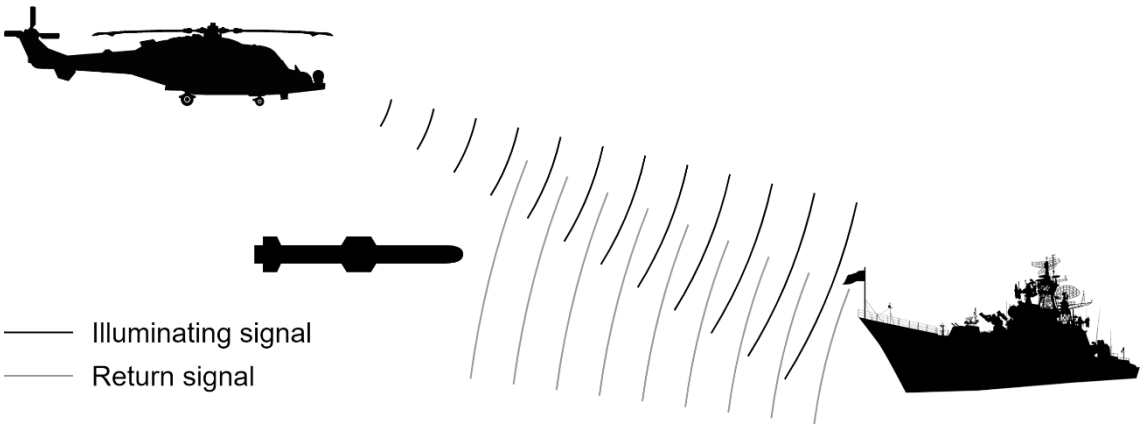


Figure 1-6: *Illustration of a semi-active radar guidance anti-ship missile. Operation of the missile seeker depends on the illumination of the target by an external radar transmitter.*

With imaging infrared missile seekers, the missile itself ‘sees’ and identifies targets, employing infrared computer vision to detect and interpret the infrared signatures of vessels in the target area, as shown in Figure 1-7. Imaging infrared guidance relies on the detection of natural infrared energy, which is radiated by all objects with temperatures above absolute zero (-273.15˚C). This energy propagates through the atmosphere and a portion enters the seeker’s optical system before being focused onto a staring array detector to produce a digital thermal image [40, 39]. The thermal image is subsequently processed by the seeker’s computer vision algorithm which exploits temperature variations within the scene to discern the relative location of any targets.

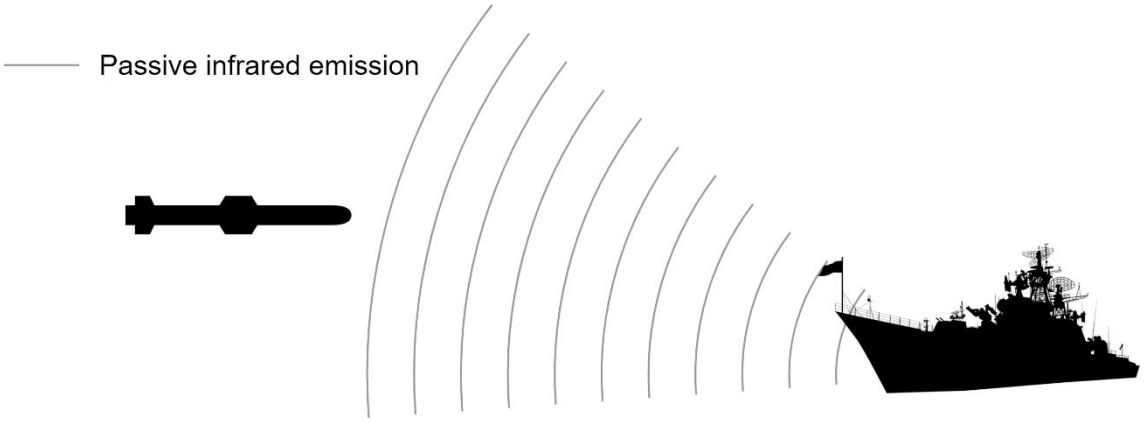


Figure 1-7: *Illustration of passive imaging infrared anti-ship missile guidance. Missile seeker operation is based on the detection of naturally occurring infrared emissions from the target.*

This thesis focuses on infrared missile seekers, and in particular the computer-vision-related challenges associated with infrared anti-ship missile seekers. Before turning to these in more detail, it is first necessary to describe the underlying principles of infrared imaging.

1.2. Fundamentals of infrared anti-ship missile seekers

In contrast to visual optical systems that detect reflected energy from a light source, such as the sun, imaging infrared systems rely primarily on the detection of radiation which is emitted by the surrounding environment. This radiation is transmitted through the atmosphere before being collected by the seeker's optical system, and then focused onto a detector array. This in turn produces a thermal image for interpretation by the seeker's image processing algorithm.

In order to understand—and subsequently improve—the operation of infrared missile seekers, this section explores the fundamental principles which underpin three topics, namely: infrared emittance, atmospheric transmission, and infrared optical systems.

1.2.1. Infrared emittance

Infrared radiation is a continuous spectrum of energy positioned between the visible and microwave bands of the electromagnetic spectrum, generally defined as being wavelengths of between 0.75 and 1,000 µm (see Figure 1-8). This radiation is emitted by all objects with a temperature above absolute zero (-273.15˚C), regardless of their state of matter. Assuming that such an object is in thermal equilibrium with its surroundings, then the rate of this emittance will always equal the simultaneous rate at which incident radiation is absorbed. Consequently, infrared optical systems operate independently of any sources of illumination and are therefore able to operate with equal effectiveness during both day and night-time conditions.

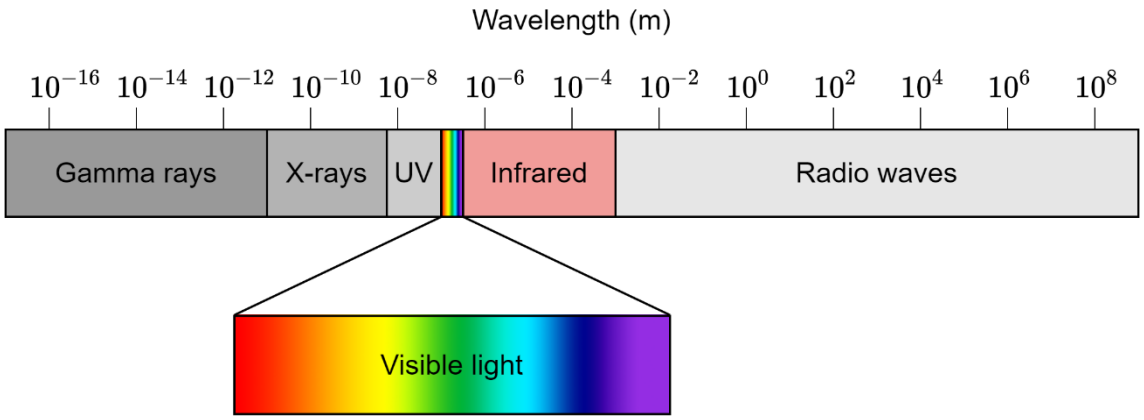


Figure 1-8: *Illustration of the electromagnetic spectrum showing the infrared waveband and its relation to the visible spectrum.*

1.2.1.1. Black-body radiation

The emittance of infrared radiation can be modelled with the aid of a ‘black-body’, a theoretical object which absorbs, and simultaneously emits, all incident electromagnetic radiation, irrespective of wavelength. Consequently, radiation emitted by a black-body—known as black-body radiation—depends only on the absolute temperature of the body.

1.2.1.2. Planck's radiation law

The spectra of radiation emitted by a black-body is closely described by Planck's law [41].

$$M_\lambda = \frac{C_1}{\lambda^5}\left(\exp\left(\frac{C_2}{\lambda T}\right) - 1\right)^{-1} \tag{1.1}$$

where:

M_λ is the spectral radiant emittance $(W \cdot m^{-2} \cdot \mu m^{-1})$,

λ is the wavelength (μm),

T is the temperature (K),

$C_1 = 2\pi c^2 h = 3.74 \times 10^{-8}\ W \cdot m^{-2} \cdot \mu m^4$ is the first radiation constant,

$C_2 = {hc}/{k} = 1.44 \times 10^4\ \mu m \cdot K$ is the second radiation constant,

and:

$c = 2.998 \times 10^8\ m \cdot s^{-1}$ (the speed of light in a vacuum),

$h = 6.626 \times 10^{-34}\ J \cdot Hz^{-1}$ (Planck's constant), and

$k = 1.381 \times 10^{-23}\ J \cdot K^{-1}$ (Boltzmann's constant).

Figure 1-9 shows black-body radiation, as described by Planck's law, for a selection of temperature values and illustrates two notable natural laws: 1) that the total radiant emittance—equal to the area under the radiance curve—increases exponentially with temperature, and 2) that the higher the temperature, the shorter the wavelength at which the spectral distribution peaks. Consequently, these laws affect the design and operation of infrared seeker systems, which must be suitably sensitive in the portion of the infrared waveband that is appropriate for their application: at too high a sensitivity, the sensor would become saturated; at too low a sensitivity, the sensor would fail to detect enough infrared energy to produce an image.

Conventionally, the infrared radiation spectrum is portioned into five distinct wavebands [42], based on the applicable wavelength and atmospheric transmission characteristics, as shown in Table 1-2 and Figure 1-9. These are the Near Infrared waveband (NIR), Short Infrared waveband (SWIR), Mid Infrared waveband (MWIR), Long Infrared waveband (LWIR), and the Far

Infrared waveband (FIR). For detecting objects at ambient temperatures, LWIR is the waveband of choice for use in thermal imaging. Clearly, in other missile applications, other wavebands may be appropriate: for anti-aircraft missiles homing-in on the exhausts of jet engines, the MWIR waveband is typically used.

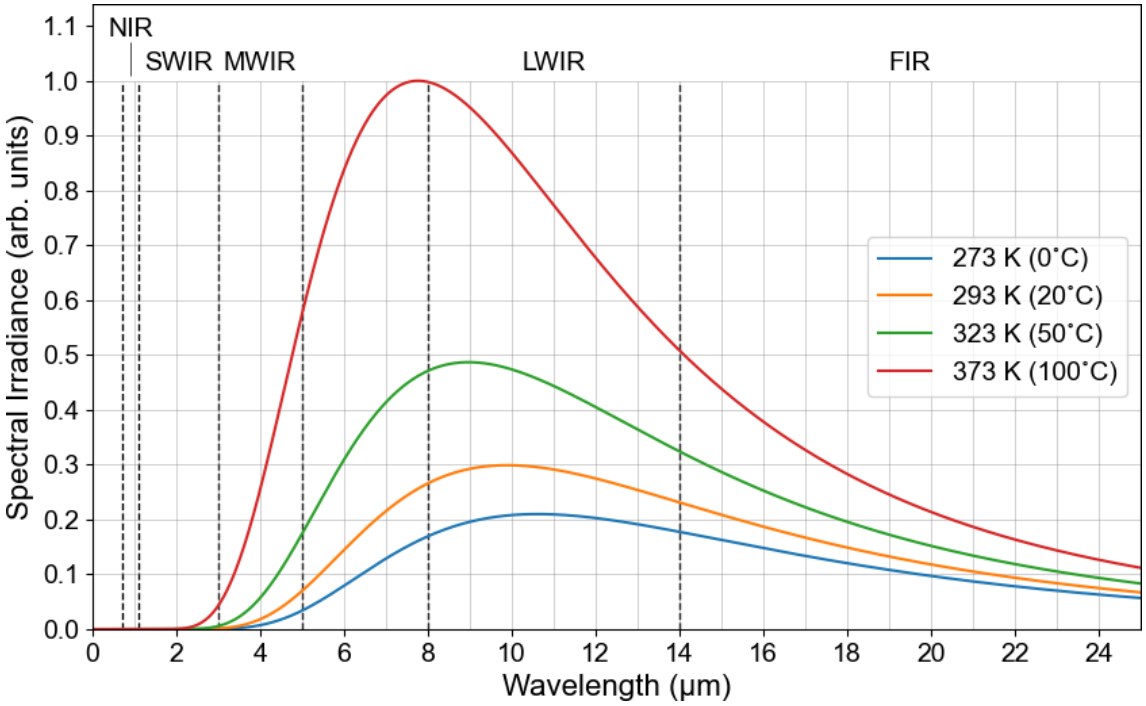


Figure 1-9: *Spectral distribution of a blackbody at different surface temperatures. As the temperature of a blackbody increases, the wavelength at which its spectral distribution peaks is reduced.*

Table 1-2: *The five infrared wavebands and the corresponding temperature at which radiation in this waveband is the primary emission [42].*

Waveband Name	Waveband (μm)	Temperature (˚C)
Near-infrared (NIR)	0.7 – 1.1	1,725 – 3,591
Short-wavelength infrared (SWIR)	1.1 – 3	693 – 1,797
Mid-wavelength infrared (MWIR)	3 – 5	306 – 689
Long-wavelength infrared (LWIR)	8 – 14	-67 – 89
Far infrared (FIR)	14 – 1,000	-270 – -67

1.2.1.3. Gray bodies and selective radiators

As they absorb only a portion of incident electromagnetic radiation, real-world objects cannot generally be modelled as black-bodies, as they not perfect emitters. To account for this, Planck's law can be modified with an efficiency term known as emissivity (ε), as shown in Equation 1.2 [43].

$$M_\lambda = \varepsilon(\lambda)\frac{C_1}{\lambda^5}\left(\exp\left(\frac{C_2}{\lambda T}\right) - 1\right)^{-1} \tag{1.2}$$

Emissivity is defined as the ratio between the real thermal radiation of an object and that of a theoretical black-body of the same temperature. For a minority of solid materials, including quartz and feldspar, emissivity is a function of wavelength and, as such, these materials are known as selective radiators [44]. However, the majority of solids can be considered as grey bodies, with an emissivity which can be approximated by a constant value across all wavelengths. For illustrative purposes, Figure 1-10 shows the spectral distribution of a range of materials: glass, steel, and PTFE—a material commonly used in the construction of radomes.

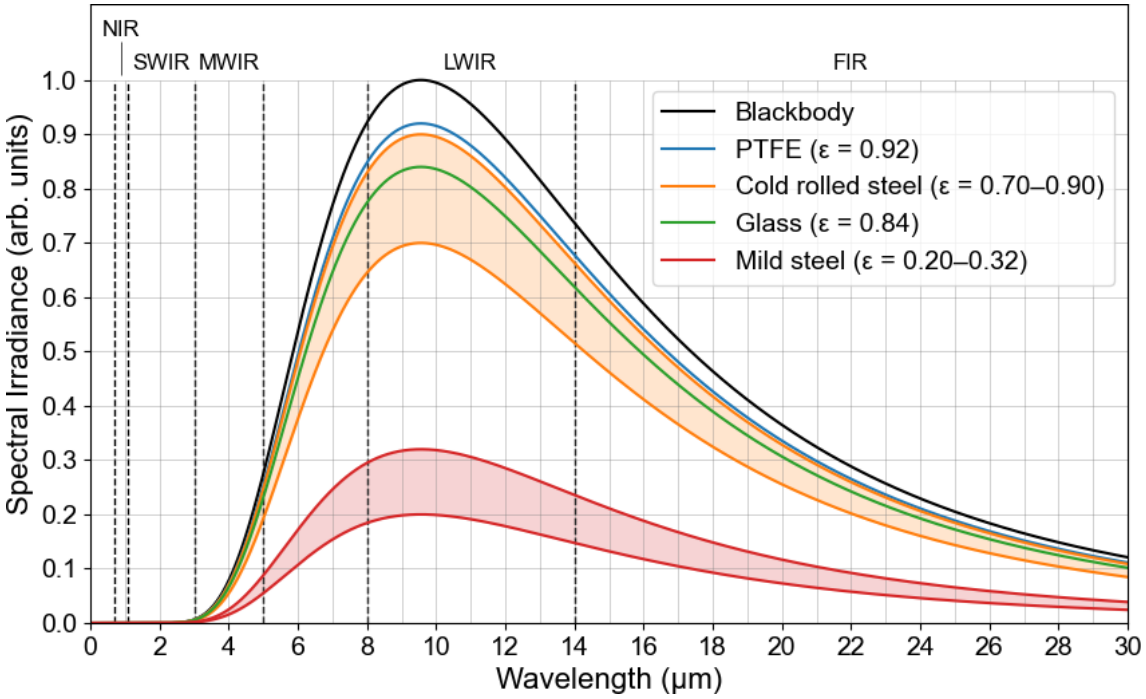


Figure 1-10: *Spectral distribution of various grey bodies at a temperature of 30˚C, based on emissivity values as stated in [45, 46, 47].*

1.2.2. Atmospheric transmission

Once emitted, infrared radiation must propagate through the atmosphere before it can be captured by an optical system in order to generate a thermal image. The efficiency with which this occurs—known as atmospheric transmission—dictates the range at which certain wavelengths of energy can be detected, and is dependent on two processes: scattering and absorption.

Absorption is a quantum process in which a molecule absorbs the energy of incident photons, causing it to change its internal state, thus increasing its energy and elevating its temperature [42]. Each molecule has several possible energy state changes within its energy structure, known as energy gaps (E_g). If an incident photon is to be absorbed by a molecule, the photon must have slightly higher energy than one of these gaps, such that $E_{photon} \geq E_g$; lower energy photons are not absorbed. The energy of a photon is given by Einstein's equation [48],

$$E_{photon} = hv \tag{1.3}$$

where E is the energy of a photon, h is Planck's constant $(6.626 \times 10^{-34}\ J.Hz^{-1})$ and v is the frequency of the incident light. Given that $v = \frac{c}{\lambda}$, where c is the speed of light in a vacuum $(2.998 \times 10^{8}\ m.s^{-1})$ and λ is the wavelength of a photon, then the possibility of a photon's absorption by a given molecule is determined by its wavelength.

Scattering, on the other hand, is a physical process, whereby photons collide with atmospheric particles and are subsequently radiated in all directions. The fraction of energy imparted from the photon—and the pattern of their re-direction—depends on the ratio of the incident wavelength to the size of the atmospheric particle.

Clearly, the processes of absorption and scattering will both be influenced by a range of environmental factors, such as humidity, temperature, pressure, haze, altitude, and the presence of smoke. Figure 1-11 illustrates how the transmission of infrared radiation changes with respect to its wavelength, as measured over a horizonal path of about 1 nautical mile (1.9 km) for a typical north-western European atmosphere at sea-level [49]. Though caused by both molecular and aerosol scattering and absorption, overall, it is the selective absorption by water vapour, carbon dioxide, and—to a lesser degree—ozone molecules which generate the largest attenuation effect. This results in several atmospheric windows: regions of high transmittance unaffected by high absorption, which lend themselves to thermal imaging. Chiefly, as shown in Figure 1-11, these are the 3–5 μm (MWIR) and 8–14 μm (LWIR) wavebands.

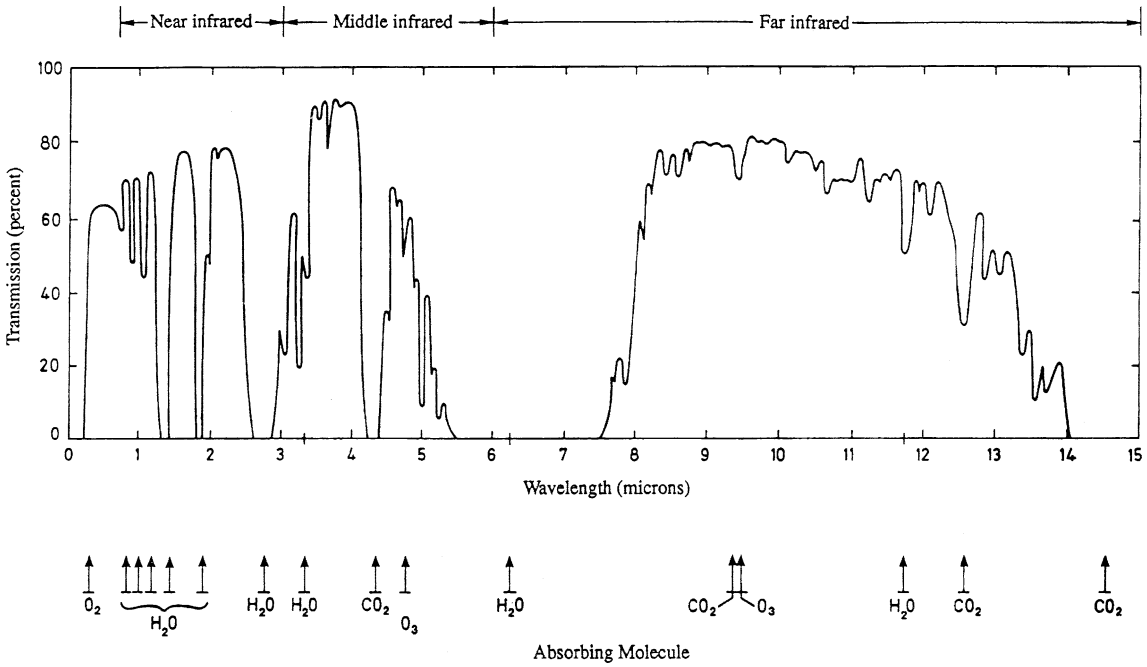


Figure 1-11: *The transmission of infrared radiation plotted with respect to wavelength, measured at sea-level over a horizontal path of circa 1 nautical mile. (Image sourced from [49].)*

When haze or smoke is present, these atmospheric windows offer superior transmission to visible light since, at *circa* 0.5 µm, the particles are of a low size relative to infrared wavelengths. However, atmospheric transmission within these windows can be impeded by other adverse weather conditions, such as fog and cloud, where constituent droplets of 5–15 µm cause high levels of scattering. In addition, rain can further reduce transmission by as much as 20% per kilometre, while simultaneously also reducing the thermal contrast of a scene by coating all surfaces with a thin layer of water.

Ultimately, although MWIR offers superior transmission for long ranges and humid conditions, transmission in the LWIR waveband is typically judged to be most efficient under most circumstances at ranges up to 10 km [49]. Consequently, LWIR is the most suitable waveband for use by an infrared anti-ship missile seeker.

1.2.3. Seeker optical system

The infrared optical system of a missile seeker is responsible for collecting incident infrared radiation from the environment in order to generate a thermal image. As illustrated in Figure 1-12, this system is dependent on four key components: an optical dome, aperture, lens, and detector array, which are in turn influenced by three key parameters: aperture diameter, focal length, and detector array size.

Infrared radiation travels through the **optical dome**, the purpose of which is to protect the rest of the system from the weather and any aerodynamic forces. It then travels through the **aperture**, that controls the amount of energy entering the system. Finally, the **lens** focuses the radiation onto the **detector array**, with the radiation interacting with the electrons in the detector material in order to produce a current which is proportional to the number of photons absorbed. The **field of view** of the overall optical system is then controlled by the configuration of the lens and the detector array.

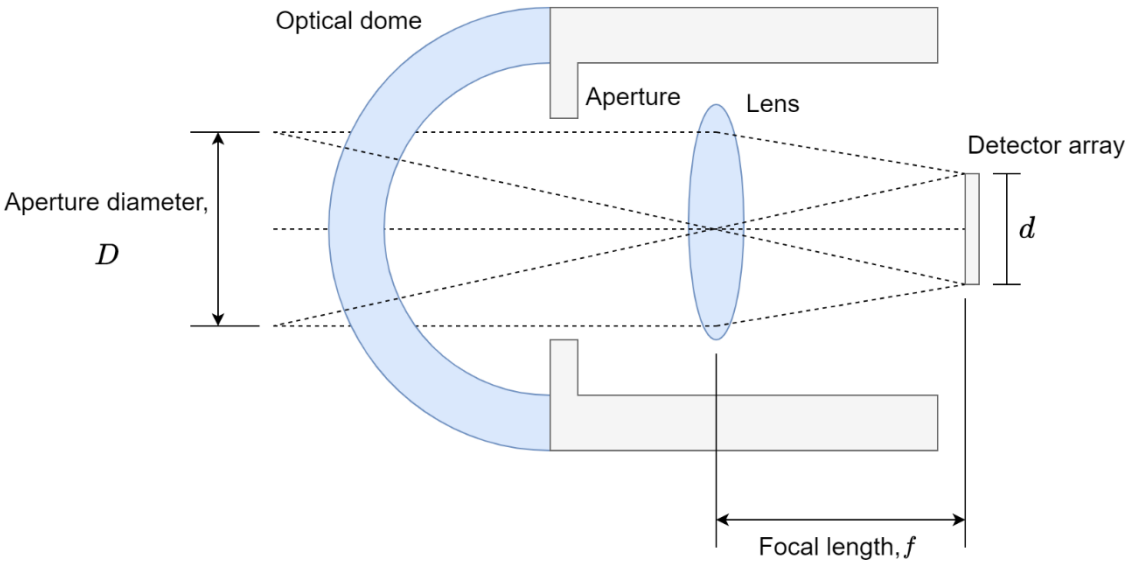


Figure 1-12: *Schematic of a simple optical system for an infrared missile seeker.*

The brightness of the detector's response depends on the focusing power, or F-number, of the optical system. This is given by the ratio of the effective focal length, f, to the aperture diameter, D, as shown in Equation 1.4, with a low F-

number resulting in more photon interactions per unit area of the detector array, as increasing the focal length causes incident radiation to be focused into a larger area, thus reducing its intensity. Increasing the aperture diameter, on the other hand, allows more energy to enter the system. Consequently, suitable aperture diameter and focal lengths must be selected, within the size constraints of the missile, in order to collect sufficient radiation as to produce a useful image.

$$F\ number = \frac{f}{D} \tag{1.4}$$

The field of view of the optical system, θ, is determined by the focal length, f, and the detector array size, d, as shown in Figure 1-13 and Equation 1.5. This array size is, in turn, dependent on its resolution and the size of its constituent pixels, which have a minimum theoretical size of 2λ (*circa* 28 µm for LWIR) [49].

Logically, the field of view dictates the extent of the missile's forward view which can be captured in any given frame. For long-distance scenarios, optical systems with a narrow field of view are best suited. As an anti-ship missile approaches its target, the target's apparent scale will increase, eventually filling the seeker's entire field of view. But if the field of view is too narrow, the seeker's field of view will be prematurely filled, resulting in loss of target-lock. It is therefore crucial that the optical system is equipped with a field of view suitable for the size of the missile's intended targets and the expected range of engagements.

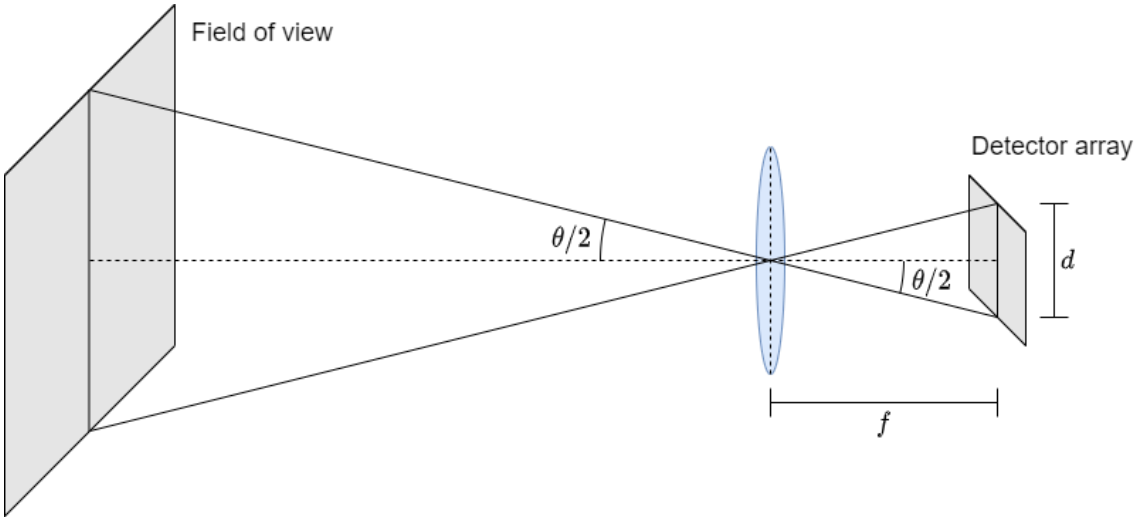


Figure 1-13: *Schematic to illustrate how focal length and detector array size affect the field of view of the overall optical system.*

$$\theta = 2\tan^{-1}\left(\frac{d}{2f}\right) \tag{1.5}$$

A missile's field of view, together with the physical construction and configuration of its optical system—the lens, aperture, and detector array—comprises only one facet of the challenge of guiding an infrared anti-ship missile onto a target. Within that field of view, suitable targets may (or may not) exist. And, determining whether this is the case calls for the infrared image built up from the photons falling on the detector array to be intelligently interpreted through the use of an appropriate computer vision algorithm.

1.3. Relevant challenges in computer vision

Computer vision is a field of study which seeks to develop techniques that provide computers with an understanding of visual data, such as images or videos [50]. In this sense, the word "understanding", amounts to automatically analysing visual data to produce higher-level abstract information or metadata such as a depth map, a summarising description, or even a 3-dimensional model [51, 52, 53]

In the case of infrared anti-ship missile seekers, the seeker algorithm must 'read' and interpret a rich infrared 'landscape' potentially containing multiple

vessels in multiple orientations, and do so in multiple weather conditions, at night and also during the day, while potentially also dealing with multiple countermeasures intended to confuse and distract the seeker from its mission of detecting and homing-in on suitable target vessels.

At first glance, computer vision tasks such as image classification and object detection can seem trivially simple. As children, each of us learned to solve these challenges very readily. In fact, leading experts in artificial intelligence supposed in the 1960s that making computers ‘see’ would be of comparable difficulty to a student’s summer project [54]. But sixty years later the field of computer vision remains rich in unsolved challenge.

A principal difficulty is the dynamic and infinitely varying nature of the physical world in which computer vision algorithms must operate. For a computer vision algorithm to successfully interpret digital image data, it must possess an understanding of the relevant objects contained with the field of vision, and also the interaction between those objects. A computer vision algorithm must also understand object ‘persistence’: if an object is temporarily obscured, for instance, the algorithm needs to understand that it has probably not in fact vanished, but is instead obscured, and is likely continuing on its last known trajectory.

Traditionally, two ways in which a general object detection algorithm can fail have been recognized. The first of these is the *false positive error*, also known as a *type I error*, where the algorithm falsely infers the existence of an object where, in actual fact, there is no object of interest. The second is the *false negative*, also known as a *type II error*, where the algorithm fails to correctly detect the presence and location of an object that is in fact actually present in the image.

More recently, richer and more granular error classification schema have been recognized. Today, for instance, the literature includes *type S errors* (errors of sign), and *type M errors* (errors of magnitude) [55], as well as the notion of the confusion matrix, which has found favour in the field of machine learning in the

context of object classification [56]. For the purposes of this thesis, we are primarily concerned with eliminating or minimising *type I* and *type II* errors: is an object of interest present, or not? An object's magnitude, or classification—destroyer or frigate, say—is of secondary interest.

A common cause for both types of failure mode is a failure to sufficiently generalise for the given scenario. First, consider the case of *false negative* errors. Upon encountering a target with an unexpected appearance—such as an aircraft carrier with *two* command bridges—the algorithm's understanding of the target must be sufficiently general as to encompass this unusual target. If the algorithm is unable to generalise, it will fail to detect the target, resulting in a failure to engage the target. Secondly, considering the case of *false positive* errors, it is vital that background clutter—including navigation buoys, drilling platforms, or even freak waves—must never fall within the algorithm's general model of the target. This would cause the regular incorrect detection of irrelevant background objects, thereby distracting the missile and causing it to veer off course.

To eliminate both these types of error will require the seeker algorithms to account for all possible likely appearances of both targets and background clutter. The challenge in this is that such appearances are hugely variable, thus providing considerable scope for errors. In the context of infrared missile seekers, there are three key sources of variation that influencing target appearance and contributing to detection failure. These are:

- The vessels themselves,
- The perspective from which the vessels are imaged,
- Any applicable environmental conditions which may impede target detection,
- Distraction and seduction caused by countermeasures.

Each of these is discussed below.

1.3.1. Vessel-related characteristics

Target vessels can vary at three distinct levels of increasing granularity: by *type*, by *class*, and by *condition*. Naval vessels are typically categorised by type based on their role: patrol boat, minesweeper, and destroyer, for example. As seen in Figure 1-14, from the 73-metre Visby corvette to the goliath 333-meter-long Nimitz class aircraft carrier, variation in ship type creates a dramatic range of potential target sizes. Moreover, with each type of vessel fulfilling different operational requirements, the result can be considerable topological differences between targets. An aircraft carrier, for instance, has a very different profile from that of a destroyer or frigate.

Figure 1-14: *A 73m-long Visby-class corvette of the Swedish Navy (left) and a 333m-long Nimitz-class aircraft carrier of the US Navy. (Images sourced from [57, 58].)*

Within each *type*, navy vessels also vary by *class*, based on the finer design details of the vessel. Consider the Royal Navy's Type 45 destroyers, the US Navy's Arleigh Burke destroyers, and Chinese Navy's Luyang III-class destroyers. Pictured in Figure 1-15, these are self-evidently all destroyers, but each hosts different radar and armament systems, placed at different locations within a uniquely designed superstructure. Additional variation can come from the individual day-to-day *condition* of each individual vessel, which will differ according to the on-deck presence (or not) of equipment such as rotary wing aircraft. And in an infrared context, the appearance of a vessel will also vary with the temperature of its surfaces, which can be affected by the use of internal equipment and other onboard conditions.

Figure 1-15: *The Type-45 destroyer of the Royal Navy (left) Arleigh Burke-class destroyer of the US Navy (right), and Luyang III-class destroyer of the Chinese Navy (centre). (Images sourced from [59, 60, 61].)*

From the point of view of a seeker algorithm, the combination of type, class, and condition introduces considerable complexity. There are currently *circa* 2,300 different classes of military vessel in operation around the world, and new classes of vessel are commissioned each year [62]. Further, the condition of each and every vessel is in a constant state of flux. Despite this, missile seekers must be capable of detecting target vessels irrespective of their type, class, and condition.

1.3.2. Orientation-related characteristics

From the initial moment of detection, the apparent scale of the target will increase rapidly as—from over the horizon—the missile approaches a target vessel. Furthermore, the target itself could be steaming towards the missile, steaming away from it, or be encountered sideways on. In addition, modern-day missiles typically perform a terminal ‘bunt’ manoeuvre, climbing steeply before diving down onto the target at a steep angle. The target can itself elect to manoeuvre throughout the engagement, again causing the perspective of the target relative to the seeker to change. This change in perspective results in

significant variations in the target appearance, yet the missile seeker must account for this and must maintain guidance-lock on the vessel.

1.3.3. Environmental conditions

Infrared missile seekers depend on the passive emission of infrared radiation by the target. This emission is captured by an imager, and the resultant image can vary considerably with atmospheric conditions. Both ambient temperature and the quantity, size, and composition of airborne particulates, such as rain, can affect the transmission of long-wave infrared radiation [63], and thus changing the apparent appearance of the target as viewed by the seeker. Also, direct sunlight can heat ship surfaces, further altering the thermal appearance of the vessel. For an infrared anti-ship missile seeker to be robust across a wide range of weather conditions, it must be capable of accounting for such environment-driven variations.

1.3.4. Problems posed by countermeasures

Targets can take countermeasures in an attempt to avoid a missile strike. Such countermeasures can be divided into two broad categories: hard-kill countermeasures and soft-kill countermeasures.

Hard-kill countermeasures aim to defend vessels from missile strikes by physically destroying the airframes of incoming anti-ship missiles. Several technologies are in use.

The Mark 15 Phalanx, shown in Figure 1-16, is a gun-based close-in weapon system, capable of firing 4,500 20 mm rounds per minute at inbound threats, directed automatically by a fire-control radar, which in addition to tracking the inbound missile, also tracks outbound rounds, enabling the Phalanx system to self-correct its aim [64].

Figure 1-16: *A Phalanx Block 1B in action (left) and the Aster-30 missile just after launch (right). (Images sourced from [65, 64].)*

The Aster 15/30, on the other hand, also pictured in Figure 1-16, is a vertically launched anti-aircraft and anti-missile missile with a two-stage solid propellant rocket motor giving it a maximum speed of 1 km per second [65]. Its active radar seeker incorporates anti-radiation capabilities, making it invulnerable to jamming, and its clutter suppression allows it to engage sea-skimming missiles.

When confronted by gun-based defences, the two key defences of an anti-ship missile are speed and agility. Should a missile alter its direction when travelling at speed, projectiles can no longer be expected to intercept it [66]. Consequently, the probability of a successful defence diminishes considerably with the manoeuvrability of the missile. It has been calculated that, for a low-agility target (capable of pulling 1g) at a distance of 500 metres, just 45 rounds from a CIWS would provide a 50% probability of scoring a hit [66]. In contrast, when considering a high-agility missile (capable of pulling 10g), this number rises considerably to 4,460 rounds [66]. With a magazine capacity of just 1,550

rounds [67], gun-based countermeasures like Phalanx are unlikely to provide effective defence against fast and manoeuvrable anti-ship missiles possessing seeker algorithms capable of maintaining target-lock while engaging in these high-speed manoeuvres.

In contrast, the ability of an anti-ship missile to perform tight manoeuvres at high speeds confers fewer advantages when confronted by anti-missile missiles, which cannot be out-manoeuvred so easily. Consequently, anti-ship missiles may instead need to rely on numerical superiority as a form of counter-countermeasure to overwhelm the missile-based ship defences. This high-volume approach is an emerging naval strategy, and it has been estimated that a naval task force within 2,000 km of China could expect to face up to 640 incoming anti-ship missiles in a single salvo [68, 69].

Beyond conventional guns and missiles, direct energy weapons are also emerging as hard-kill countermeasures. These systems use beams of electromagnetic energy to defeat incoming missiles by either dazzling their sensors, or fatally heating-up and destroying their critical components. During trials in 2019, for instance, the ground-based Demonstrator Laser Weapon System, shown in Figure 1-17, successfully engaged and shot down multiple air-launched missiles [70, 71].

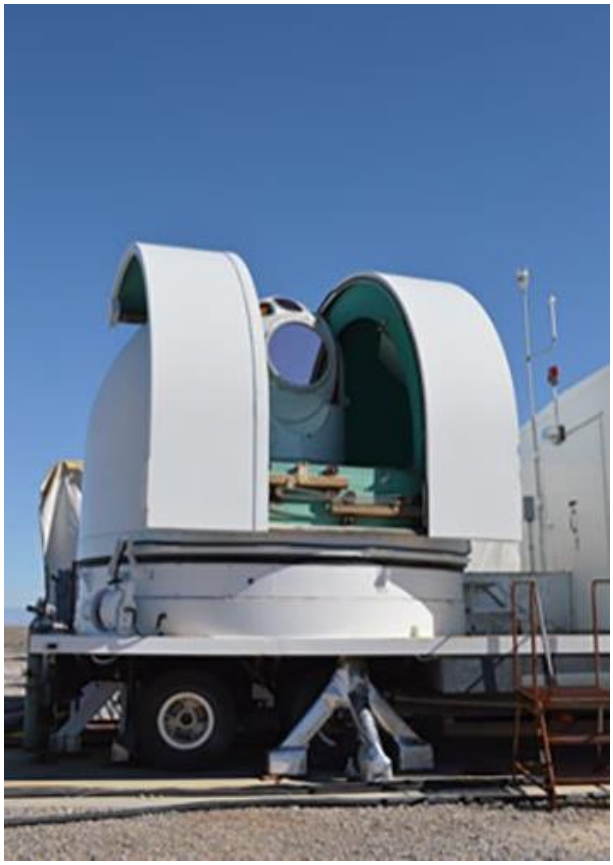


Figure 1-17: *The Demonstrator Laser Weapon System, developed by the United States Air Force Research Laboratory, which engaged and shot down several air-launched missiles in April 2019. (Image sourced from [72].)*

In contrast to hard-kill defences, **soft-kill countermeasures** aim to defeat incoming threats by confusing or seducing them to a position that is safely distant from the defending vessel. The primary soft-kill measure deployed against infrared anti-ship missiles are flares. These are expendable decoys deployed from deck mounted launchers, such as the Mk 36 SRBOC device, pictured in Figure 1-18, which use phosphorous-based materials to create persistent (30 – 60 seconds) infrared ‘blooms’ that are of comparable size to the defended vessel [40].

Figure 1-18: *A Mark 36 Super Rapid Bloom Offboard Countermeasures (SRBOC) device launching its offboard decoy from the deck of the Ticonderoga-class cruiser USS Lake Champlain. (Image sourced from [73].)*

Depending on the approach path of the inbound missile, flares can be launched at specific angles of azimuth and elevation to maximise the chances of decoying the threat. A common tactic for defending vessels is the ‘walk-off’ technique [74], whereby successive decoys are launched at increasing distances from the vessel. This causes the missile to lock-on to each newly appearing bloom in turn, until the defended vessel is safely outside of the missile seeker’s field of view.

Until the mid-90s, soft-kill countermeasures were almost entirely effective against anti-ship missiles, with successful missile strikes usually involving ships which either had no decoys or did not deploy them [9]. In response, counter-countermeasure techniques were developed and incorporated into missile seeker algorithms. Consequently, the success rate of infrared decoys—which had stood at well over 90%—fell considerably, and is currently estimated to be between 35 – 58% [40, 75].

To re-instate the effectiveness of soft-kill measures, Directional Infrared Countermeasures (DIRCMs) have been developed to dazzle and confuse infrared seekers [76, 77, 78]. Similar to direct-energy weapons, DIRCMs use a beam of electromagnetic energy—but not to heat and destroy the missile, but instead to saturate the detector array of its infrared seeker. Though not effective on their own, when used in combination with flares DIRCMs have been found to increase the probability of defeating an incoming missile to 85% [40].

From the perspective of seeker algorithms, the increased effectiveness of the current generation of soft-kill countermeasures, and the emerging threat of DIRCMs, means that the advantages of maintaining target-lock while manoeuvring at high speed—to avoid these countermeasures—remain compelling. Equally, algorithms need to be more sophisticated in target-selection terms, rather than just focusing on bright infrared sources.

1.4. Automatic target recognition

Automatic Target Recognition (ATR) is a field of research which is concerned with the detection and characterisation of specific objects of interest from sensor data for defence- and security-related applications. ATR can involve data from a wide variety of sensors, including visible, infrared, and hyper-spectral imagers, LiDAR, synthetic aperture radar, or even a fusion of data from multiple data sources and encompasses a range of applications including harbour surveillance, anti-piracy security, and, ultimately, anti-ship missile seekers [79, 80].

ATR encompasses a hierarchy of different tasks, principal of which are detection, recognition, and identification. The precise definition of these tasks can vary between ATR applications, and these terms also overlap considerably with the more general field of computer vision. Consequently, they are defined in the context of imaging infrared anti-ship missile seekers as follows:

- Detection is the task confirming the present of and localising target vessels.
- Recognition is the task of classifying target vessels by type.

- Identification is the task of classifying target vessels by class.

Accordingly, in this thesis the terms detection, recognition, and identification will refer only to these specific definitions.

Additional ATR-related tasks include target characterisation, prioritisation, aim-point refinement and tracking.

Characterisation is the task of inferring specific details or distinguishing features of a target vessel, such as its onboard weapons, sensors, and countermeasures.

Prioritisation is the task of ranking vessels, typically based on their tactical or strategic value. There are various motivations for equipping a missile seeker with target prioritisation capabilities such as: to ensure that the missile strikes the target with the highest strategic value, to ensure that multiple missiles each strike a common target, or to ensure that multiple missiles each strike different targets.

Aim-point refinement is the task of selecting a singular point on the vessel for which the missile should aim in order to maximise its chances of disabling the target. For example, this may involve the detection of a vessel's key points, such as the bridge, command and control centre, or engine room in order to facilitate a precise strike.

Finally, **tracking** is the task of preserving target identities through time by the analysis of historical characteristics, such as appearance and dynamics. This acts to improve missile performance in multi-target scenarios, especially when target speed is comparable to that of the missile, since inadvertently switching lock to a different target can cause kinetic energy to be wasted, thus reducing the chance of a successful strike.

1.4.1. Current approaches

In general, the existing IR anti-ship ATR systems are based on pattern recognition techniques and can be described by the pipeline illustrated in Figure 1-19. The task of any ATR algorithm is detection, which often consists of two

stages, Region of Interest (RoI) selection and object segmentation. This is followed by recognition and/or identification, whereby features are extracted from each target and presented to a classification algorithm which uses prior information provided by a dataset to predict the target type and/or class.

If necessary, after recognition/identification each target may be characterised further, and their attributes may be compared in order to prioritise the most appropriate target. Once the target has been selected, a process of aim-point selection may be employed to precisely identify a key point of impact. ATR systems also incorporate a form of target tracking which considers the historical attributes of targets in order to preserve target identities and improve detection in the current frame.

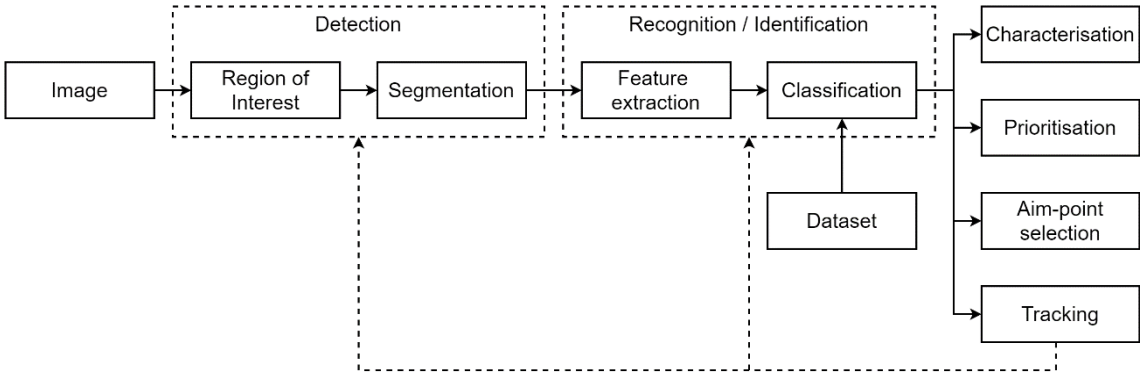


Figure 1-19: *Generalised pipeline of an automatic target recognition system.*

1.4.1.1. Detection

The aim of RoI selection is to identify the regions of an image which contains targets by bounding the extents of each target within a rectangular window. An effective approach to this is the contrast box filter [81] which determines the location of targets by assessing their relative contrast with the surrounding background. This is achieved via two concentric gates, a target gate and a background gate, as illustrated in Figure 1-20, where the background gate relates only to the pixels which are within its boarder and outside of the target gate. These gates are used to compute the contrast box metric, C at every position in the given image, where C is given by:

$$C = \frac{(\mu_T - \mu_B)^2 + \sigma_T^2}{\sigma_B} \tag{1.6}$$

where:

μ_T is the mean intensity of pixels within the target gate,

μ_B is the mean intensity of pixels within the background gate,

σ_T^2 is the variance of pixels within the target gate, and

σ_B is the standard deviation of pixels within the background gate.

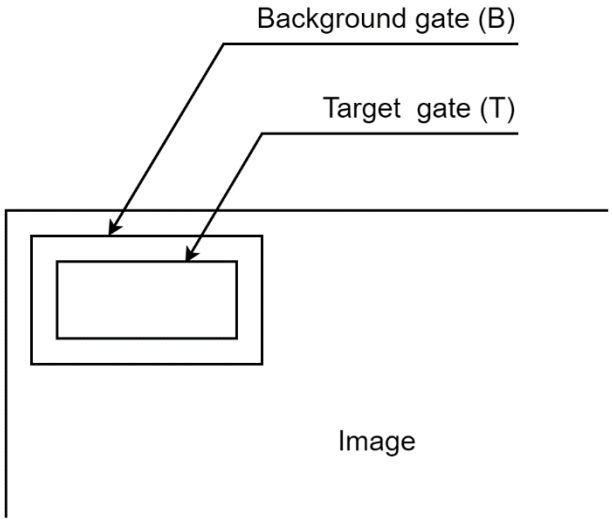


Figure 1-20: *The target gate and background gate of a contrast box filter.*

The output of the contrast box filter is a saliency map of the image where the likelihood that the target gate contains a region that is significantly different from its surroundings is provided for every position in the image. As a consequence of restricting background analysis to a localised gate, the contrast box filter is robust against large global variations in illumination. However, in order to be effective, the target gate must be of a similar size and aspect ratio to the target, which in turn, requires knowledge of the target range that is not directly available from infrared imagery [82]. Alternative approaches to RoI selection include the double-gated and spoke filters [81, 83].

After RoI selection, segmentation is performed in order to identify each of the pixels belonging to the target and isolate it from the surrounding background. As a result, the shape of the target can be extracted in the form of a silhouette, and

the target centroid can also be estimated, and these are essential precursors to classification. Segmentation is typically applied to each window identified by RoI selection, though in some systems target detection is achieved without RoI selection through a method of global segmentation [82, 75]. Effective methods of pattern recognition-based segmentation include edge detection, and intensity thresholding.

1.4.1.2. Feature Extraction

Once foreground objects have been extracted from their surrounding background, they can be classified. Depending on the desired capabilities, the classifier may perform either target recognition, target identification, or both. Classifiers may also perform binary target/non-target discrimination, in order to provide a means of countermeasure and clutter rejection. As illustrated in Figure 1-19, target recognition/identification algorithms generally consist of two stages: feature extraction and classification.

The goal of feature extraction is to convert the complex and high-dimensional target image into a low-dimensional encoding which enables a more abstract understanding of the image. This can be achieved by means of specialised shape- or intensity-based descriptors, such as those presented in Table 1-3, which were developed for differentiating flares from target ships in infrared imagery.

Table 1-3: *Shape- and intensity-based features used for infrared flare-ship discrimination [75, 84].*

Formula	Descriptor
$Z_{max}/\bar{Z}$	Maximum intensity normalised with respect to average intensity
$\mu_Z^2/\bar{Z}^2$	Intensity variance normalised with respect to squared average intensity
$\mu_Z^3/\bar{Z}^3$	Intensity third moment normalised with respect to cubed average intensity
$\sqrt{1-\frac{I_{min}}{I_{max}}}$	Eccentricity
$\bar{D}/\sqrt{A}$	Average radial distance normalised with respect to the square root area
μ_D^2/A	Variance of the radial distance normalised with respect to area

An alternative to specially designed features is image moments, such as Zernike polynomials [85], a sequence of functions that provide a global measure of how mass is distributed. Expressed in terms of polar coordinates, (ρ, ϕ), each polynomial is defined by a radial component, n and an azimuthal component, m where $0 \leq m \leq n$, and can be described as even or odd. Odd polynomials are denoted by Z_n^{-m} and defined by Equation 1.7, even polynomials are denoted by Z_n^m and are defined by Equation 1.8., and Zernike polynomials for $0 \leq n \leq 5$ are illustrated in Figure 1-21.

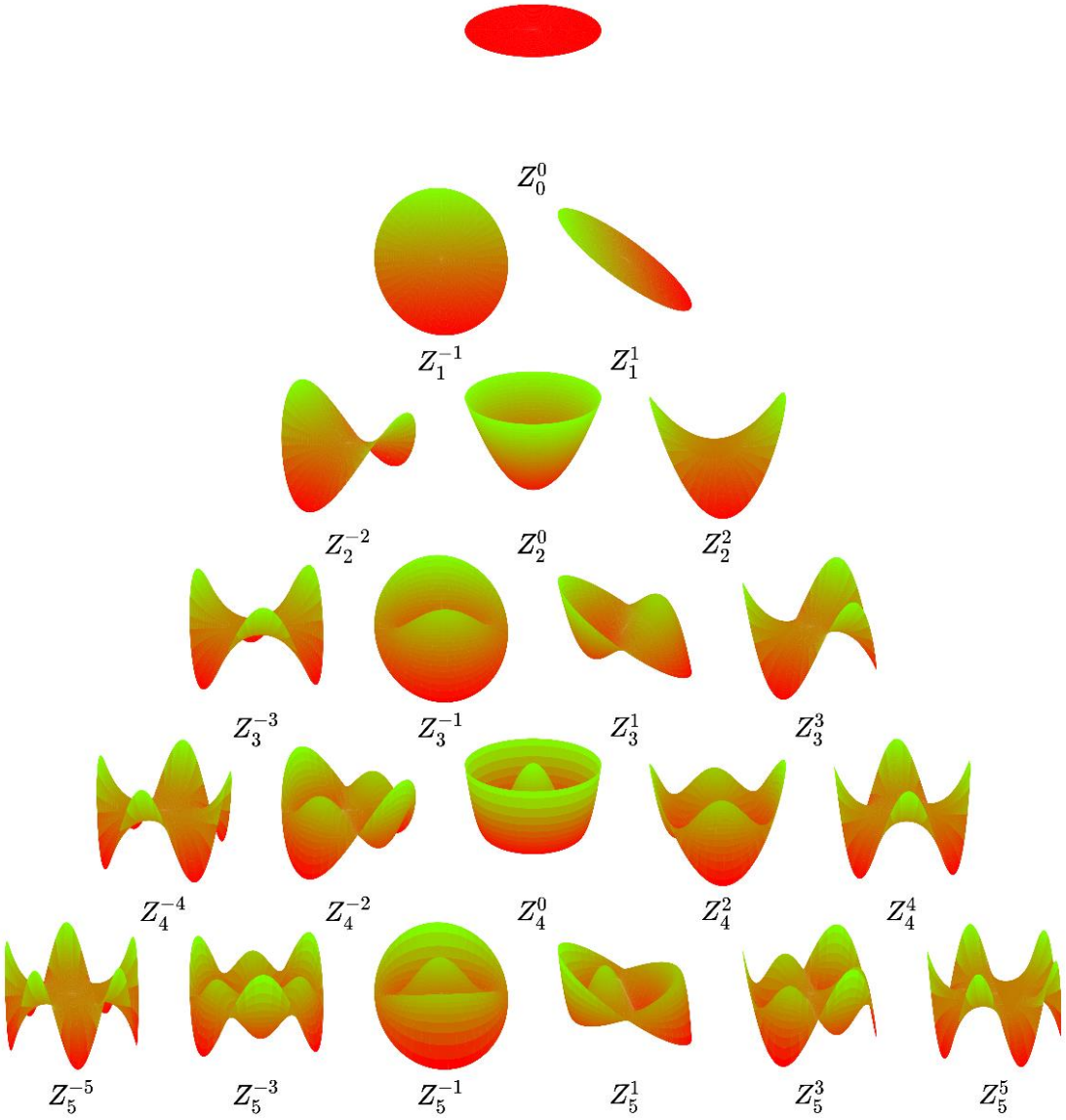


Figure 1-21: *The first 21 Zernike polynomials, ordered vertically by radial degree and horizontally by azimuthal degree.*

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi) \tag{1.7}$$

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi) \tag{1.8}$$

In Equations 1.7 and 1.8 the radial function, $R_n^m(\rho)$, is defined for n and m integers with $n \geq m \geq 0$, by:

$$R_n^m(\rho) = \begin{cases} \sum_{k=0}^{\frac{1}{2}(n-m)} \frac{(-1)^k (n-k)!}{k!\left[\frac{1}{2}(n+m)-k\right]!\left[\frac{1}{2}(n-m)-k\right]!} \rho^{n-2k} & when\ (n-m)\ is\ even \\ 0 & when\ (n-m)\ is\ odd \end{cases} \quad (1.9)$$

The consequence of this formulation is a set of descriptors which are orthogonal on the unit disc, meaning the shape of the target can be captured with no redundancy of information between moments. In addition, Zernike moments are both rotation and scale invariant, and due to their ever-increasing complexity, can provide quantitative descriptions of detailed shapes.

However, these descriptions are only useful if they can be interpreted effectively by a classification algorithm.

1.4.1.3. Classification

There is a wide variety of algorithms which can effectively classify hand-crafted features and image movements, including the generalised Hough transform [86], decision trees [87], Gaussian mixture models [88], and K-nearest neighbour [89]. However, perhaps the most effective classifier used by current ATR algorithms is the Artificial Neural Networks (ANN). Inspired by the neural networks which constitute biological brains, ANNs are mathematical models composed of multiple layers of interconnected nodes called neurons. Each neuron acts as an independent processing unit, receiving multiple signals to generate a single output which propagates through subsequent layers before eventually reaching the output layer to produce a meaningful prediction.

In general, the neural network classifiers used by IR anti-ship ATR algorithms are Fully Connected Networks (FCNs). As illustrated in Figure 1-22, these are a type of feed-forward network where all neurons in any given layer are connected to each and every neuron in the subsequent layer. FCNs can accept any number of unstructured inputs and may contain any number of hidden layers—so called because their required parameters are unknown—and may produce any number of outputs.

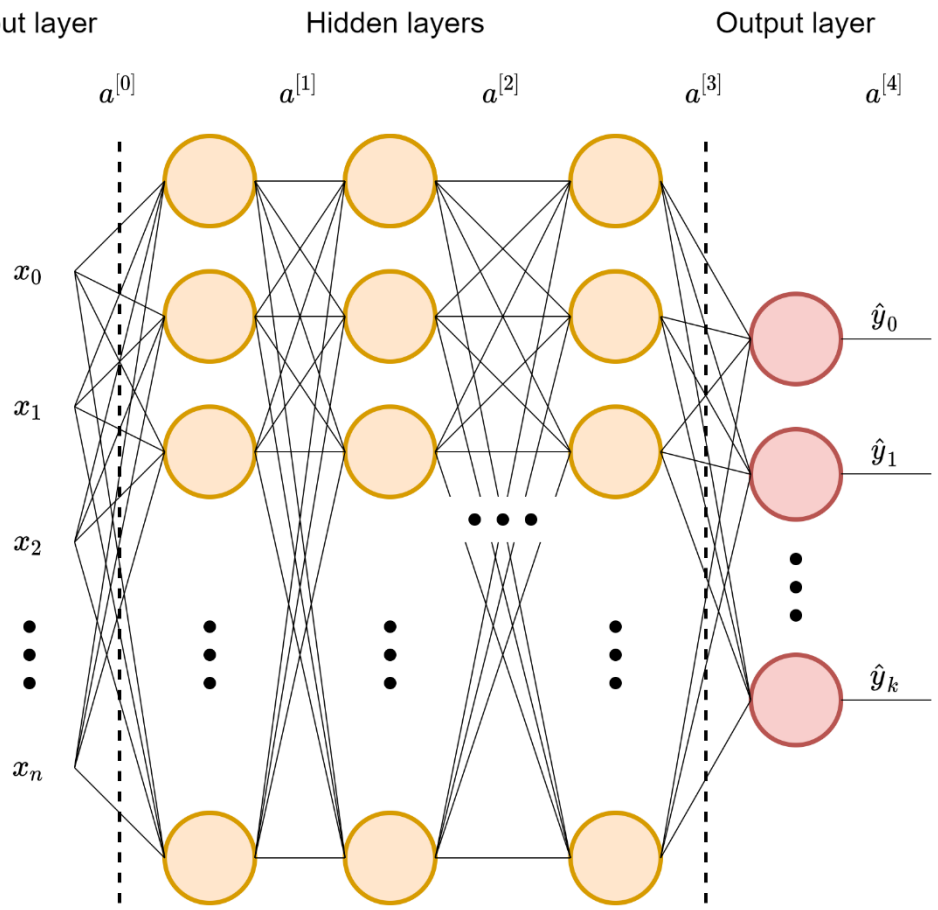


Figure 1-22: *The structure of a fully connected artificial neural network in which each node in a layer receives the output from all the nodes of the previous layer.*

As illustrated in Figure 1-23, an individual neuron accepts any number of input signals, x in the form of an $(n \times 1)$ matrix. Each input is multiplied by the internal weights of the neuron, w—also an $(n \times 1)$ matrix—and added to a bias, b to produce an intermediate variable, z. Subsequently, the output, a is generated by the application of an activation function, g. The choice of activation function depends on the desired behaviour of the output signal, though for general use, the hyperbolic tangent given in Equation 1.10 and plotted in Figure 1-24 is an effective choice.

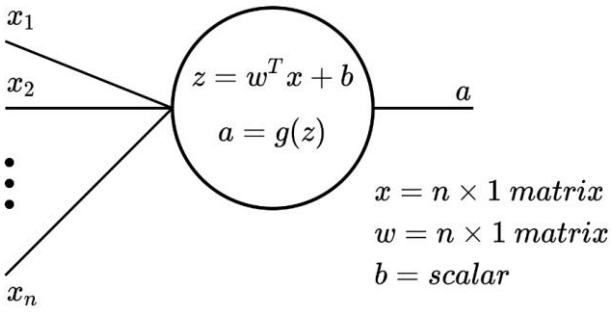


Figure 1-23: *Generalised example of an artificial neuron.*

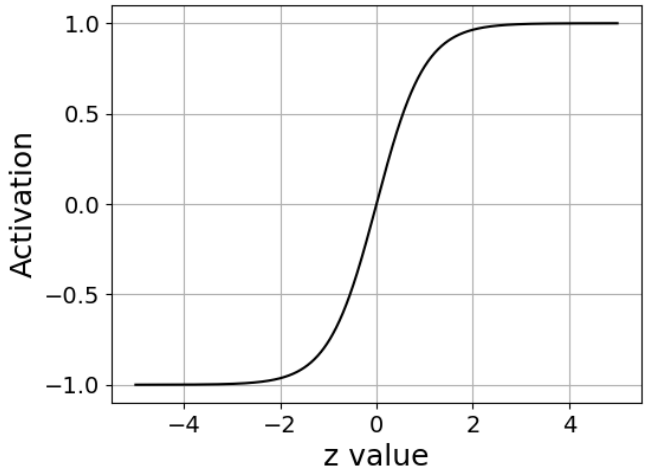


Figure 1-24: *Hyperbolic tangent activation function.*

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{1.10}$$

Each layer of an FCN contains multiple of these neurons, each executing the same process but with independent weights and biases. To compute the propagation of signals through a given layer, l, the weights and biases of each constituent neuron are combined into two-dimensional matrices $W^{[l]}$ and $b^{[l]}$. Accordingly, the forward pass through a layer which accepts n_{l-1} inputs and contains n_l neurons can be expressed as:

$$Z^{[l]} = W^{[l]T} a^{[l-1]} + b^{[l]} \tag{1.11}$$

$$a^{[l]} = g^{[l]}\left(Z^{[l]}\right) \tag{1.12}$$

where:

$W^{[l]}$ is a (n_l × n_{l-1}) matrix of weight values for all neurons in the layer,

$b^{[l]}$ is a (n_l × 1) matrix of bias values for each neuron in the layer,

$a^{[l-1]}$ is a (n_{l-1} × 1) matrix of inputs to the layer,

$a^{[l]}$ is a (n_l x 1) matrix of output values for each neuron in the layer,

$g^{[l]}$ is the activation function.

For a neural network designed for feature classification, the output layer contains as many neurons as there are classes. Each neuron is mapped to a single class and outputs the predicted probability that the given set of input features are representative of that class. Therefore, the network's output takes the form of a one-hot encoded vector, $\hat{y}$ from which a scalar prediction can be computed by taking the argmax. Instead of applying a hyperbolic tangent activation function, as done in previous layers, the output layer uses the softmax operation, given by Equation 1.13. This normalises all activations across the layer to produce desired probability predictions in a vector which sums to one.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{1.13}$$

where:

σ is the softmax function,

z is the input vector, and

k is the number of classes.

In summary, an FCN accepts a set of input features and performs a forward pass which results in a vector of class predictions. The accuracy of this production depends entirely on the internal weights and biases of each individual neuron, of which there may be thousands, or even millions. Consequently, before an ANN can be deployed, its internal parameters must be optimised to achieve the desired behaviour in a process known as training.

Training a neural network requires a set of input-output pairs, $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ and relies on the process of backpropagation

which repeatedly applies informed updates to the internal parameters. Each training input in the dataset, x_i undergoes a forward pass through the network, resulting in a series of corresponding output predictions, $\hat{y}$ and the difference between each prediction and the desired output is scored by a loss function, $\mathcal{L}(y_i, \hat{y}_i)$. A suitable loss function for multi-class classification problems is the Cross Entropy Loss [90], expressed as:

$$\mathcal{L}_i = -\sum_{c=0}^{k-1} y_{i,c} \cdot \log(\hat{y}_{i,c}) \tag{1.14}$$

where $k-1$ is the total number of classes and $\hat{y}_{i,c}$ is the inferred probability that example i is a member of class c.

The resultant set of m loss scores is then combined into a single cost score, J such that:

$$J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y_i, \hat{y}_i) \tag{1.15}$$

Subsequently, the derivative of each internal parameter is calculated with respect to the cost function in order to update each weight and bias of each layer, such that:

$$W^{[n]} := W^{[n]} - \alpha \frac{dW^{[n]}}{dJ} \tag{1.16}$$

$$b^{[n]} := b^{[n]} - \alpha \frac{db^{[n]}}{dJ} \tag{1.17}$$

where α is the learning rate, a configurable parameter which determines the magnitude of training updates. If the learning rate is too high, training may converge to a sub-optimal solution or become unstable, whereas a low learning rate will cause updates to become small and training to stall.

The iteration of this process over every input-output pair in the training dataset denotes one epoch and many epochs may be required before the cost score converges, and performance reaches an acceptable level.

1.4.2. Emergence of artificial intelligence

So how is all this to be achieved? How can ATR be built upon, employing newer technologies, approaches, and modern computer vision techniques? A significant—and growing—body of military and academic thinking [91] points to modern advances in artificial intelligence as a prominent part of the solution. In particular, attention is focusing on artificial convolutional neural networks embodying ‘deep learning’ as being a promising area of investigation and development.

It is not difficult to see why. In 2012, an artificial neural network taught itself to recognize pictures of cats [92]. Just three years later, it was declared that AI algorithms can achieve better-than-human performance at identifying general objects in visual data [93, 94]. As a result, tasks that could previously be completed only by humans, such as driving, diagnosing disease, and weather prediction, are now routinely performed by artificially intelligent systems.

These autonomous systems—for that is what they are—also have considerable scope for application in the defence industry, and militaries around the globe are currently racing to incorporate artificial intelligence into their weapon systems [95]. As the development of artificially intelligent weapons progresses, these autonomous systems will assume increasing responsibility for achieving mission objectives. They will be tasked with not only collecting information from the battlefield but also interpreting it and making tactical decisions, such as the selection and prioritisation of targets [96, 97, 98].

Both the United States and Chinese militaries are currently incorporating AI capabilities into their next generation of missile seekers [95]. The Long-Range Anti-Ship Missile (LRASM), for example, currently under development in the US, will reportedly possess automatic target detection, recognition, and identification capabilities, enabling the missile to gather and interpret its own intelligence in real-time [96]. During engagements involving multiple targets, onboard AI systems will identify and prioritise target vessels with far greater accuracy and speed than any human operator.

Robust target classification will also contribute to prevention of non-combatant deaths. During the Iran-Iraq War (1980 – 1988), the USS Stark—an Oliver Hazard Perry-class guided missile frigate—was hit by two Exocet missiles [9]. Despite being a neutral vessel and 3.2 km from the Iran-Iraq War exclusion boundary, it was mistaken for an Iranian tanker and fired at by a modified Iraqi Dassault Falcon business jet [99]. An unfortunate combination of poor military intelligence and pilot error resulted in the death of 37 neutral American sailors. The emergence of AI-driven target classification capabilities would enable intelligent missiles to identify and avoid friendly and neutral vessels.

Figure 1-25: *An image of the serious damage caused to USS Stark after being hit by an air-launched Exocet anti-ship missile during operations in the Arabian Gulf in May 1987. (Image sourced from [73].)*

Beyond LRASM, the continued advancement of AI missile seekers will enable future systems to behave analytically. Based on both real-time and long-standing intelligence, missile seekers will be capable of making rapid, abstract decisions to dynamically exploit any weaknesses the defences of their target. For example, after classification, the orientation and possible onboard countermeasures of a target vessel can be inferred. With this information, the

seeker could act autonomously to ‘outsmart’ the defending vessel by adjusting its approach to negate the effect of flares and avoid the arcs from fire of close-in-weapon-systems.

Work to develop artificially intelligent missile seekers is already underway. This next generation of missiles will offer improved detection accuracy and robustness against decoy countermeasures as well as the ability to make tactical decisions autonomously. AI guided anti-ship missiles will therefore be both more effective and much safer (at least for non-combatant vessels) than their predecessors.

1.4.3. Deep learning

AI has many branches. The branch of modern-day AI most relevant to missile seeker development is deep learning which—inspired by biological brains—uses a network of thousands of artificial neurons to model complex relationships. These Artificial Neural Networks can either learn from annotated example data in a process called supervised learning, or can even teach themselves without access to ground-truth answers, using a process known as unsupervised learning [100]. In either case, the process that enables ANNs to learn is backpropagation, whereby the performance of the ANN is repeatedly evaluated and quantified, and this quantity—known as the training loss—is used to update the relationships between the artificial neurons [101].

Deep learning algorithms are also proving particularly capable in the field of computer vision, performing tasks such as image colourisation [102], generation [103], and captioning [104], as well as human pose detection [105], facial recognition [106], and object detection [107]. Consequently, due to their powerful visual perception abilities, deep learning algorithms are perfectly suited to use by infrared anti-ship missile seekers to detect, classify, and engage target maritime vessels.

This next generation of intelligent seekers will be robust against environmental variations, impervious to confusion by soft-kill countermeasures, and will possess enhanced tactical abilities. Underpinning the development of these

three advanced capabilities re deep learning algorithms powered by Convolutional Neural Networks (CNNs).

1.4.4. Convolutional neural networks

Convolutional Neural Networks provide powerful solutions to the key challenges faced by conventional IR anti-ship ATR algorithms, equipping them with state-of-the-art target detection, recognition, and identification capabilities, while simultaneously deploying intelligent counter-countermeasure competencies.

This section provides an insight into CNNs, introducing the concept of a convolutional layer, explaining how such layers combine to form an artificial neural network, and discussing why these networks are so effective at interpreting visual data. Finally, these insights are discussed in the context of IR anti-ship ATR, outlining how and why CNNs are expected to deliver comprehensive improvements to the state of infrared anti-ship ATR.

Convolutional Neural Networks are specialised Artificial Neural Networks which assume dependency only between adjacent inputs, making them particularly useful for processing data which has a grid-like topology [108]. Examples of this include time-series data, which can be thought of as 1-dimensional grids; and images, which can be thought of as 2-dimensional grids. Instead of relying on a dedicated neuron for each input, as with the fully connected networks discussed previously in Section Classification, CNNs group multiple neurons into ‘kernels’ which are applied repeatedly over the entirety of the structured input data via convolution.

1.4.4.1. How do convolutional neural networks work?

Convolution is a mathematical operation, denoted by an asterisk and expressed in Equation 1.18, whereby a weighting function $g(-m)$ is applied at every instance of an input function, $f(m)$ [109, 108]. This causes the weighting function to emphasise different regions of the input function as it is shifted along by i.

$$z(i) = (f * g)(i) = \int_{-\infty}^{\infty} f(m)g(i-m)dm \tag{1.18}$$

When considering discrete data, such as images, this equation can be discretised to give Equation 1.19, and subsequently expanded into 2-dimensions, yielding Equation 1.20.

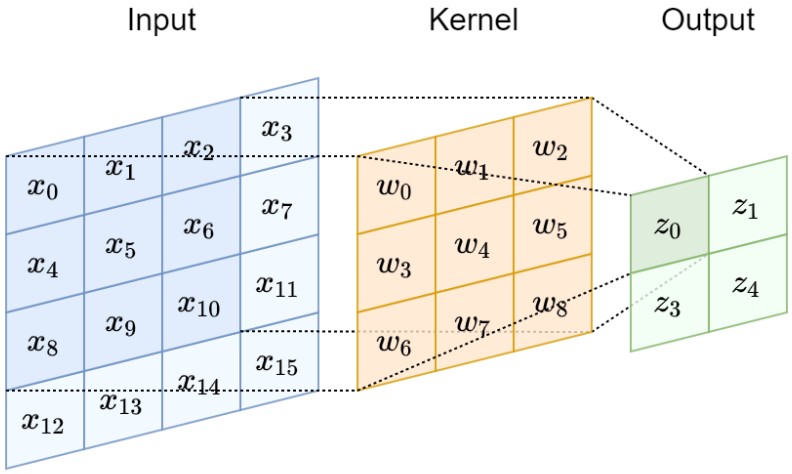
$$z(i) = (f * g)[i] = \sum_{m} f[m]g[i-m] \tag{1.19}$$

$$Z(i,j) = (f * g)(i,j) = \sum_{m}\sum_{n} f(m,n)g(i-m,j-n) \tag{1.20}$$

As m increases, the index into the input function increases, but the index into the weighting function decreases. This has the effect of reversing the weighting function which, in turn, makes the convolution operation commutative. However, while commutativity is useful for writing proofs, it is not usually important for machine learning applications, so CNN implementations must instead apply convolution without reversing the weight function, which is known as cross-correlation [108]. Consequently, if the input, f is an image, I, and the weighting function, g is an image kernel, K, which is not flipped, the operation can be expressed by Equation 1.21.

$$Z(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i+m,j+n) \tag{1.21}$$

This **kernel** is a matrix containing a series of weights, as shown in Figure 1-26, and can be thought of as a visual feature which, when convolved over an input image, produces an output that indicates where in the image the feature is present. Accordingly, this output is often referred to as a **feature map**. A typical **convolutional layer** contains multiple different image kernels, with each designed to identify a different feature, and produce its own feature map. To construct a **convolutional neural network**, multiple convolutional layers are stacked in sequence, meaning that the kernels of any given layer are tasked with identifying features from within the feature map of the previous layer. This enables the convolutional neural network to represent—and therefore interpret—complex visual data by expressing abstract features and concepts in terms of previous, simpler features.



$$z_0 = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_4 w_3 + x_5 w_4 + x_6 w_5 + x_8 w_6 + x_9 w_7 + x_{10} w_8$$

$$z_1 = x_1 w_0 + x_2 w_1 + x_3 w_2 + x_5 w_3 + x_6 w_4 + x_7 w_5 + x_9 w_6 + x_{10} w_7 + x_{11} w_8$$

$$z_2 = x_4 w_0 + x_5 w_1 + x_6 w_2 + x_8 w_3 + x_9 w_4 + x_{10} w_5 + x_{12} w_6 + x_{13} w_7 + x_{14} w_8$$

$$z_3 = x_5 w_0 + x_6 w_1 + x_7 w_2 + x_9 w_3 + x_{10} w_4 + x_{11} w_5 + x_{13} w_6 + x_{14} w_7 + x_{15} w_8$$

Figure 1-26: *2D convolution without kernel flipping (also known as cross-correlation) where the output is restricted to positions where the kernel lies entirely within the image.*

Consequently, the performance of a convolutional neural network is determined by the features which it can express which, in turn, are defined by the weights of the network's constituent kernels. Given that a CNN typically contains thousands, or even millions, of kernel weights, the network must learn appropriate values for these weights from example data by backpropagation, as described earlier in Section 1.4.1.3.

This training process enables CNNs to automatically learn kernels which are optimised towards representing the abstract concepts which are associated with the given computer vision task, often in a matter of just hours or days. It is for this reason that CNNs are the method of choice for an astonishing range of complex computer vision applications which includes general object detection, image style transfer, text-to-image translation, image colourisation, and even motion transfer [110, 103, 111, 112, 113].

1.4.4.2. How can convolutional neural networks improve anti-ship ATR?

Having already surpassed human-level performance in tasks like general object detection [93], Convolutional Neural Networks lend themselves to the similar challenge of infrared anti-ship ATR. With its ability to learn abstract visual concepts, a single well-designed and well-trained CNN has the potential to conduct all of the required ATR tasks, including detection, identification, and recognition.

Furthermore, the high-level representations learned by CNNs encode more useful information than conventional approaches based on hand-crafted features. This means that CNNs will be better at generalising to the present scenario and will therefore provide improved robustness in the presence of adverse environmental conditions, background clutter, and countermeasures. This also means that they will be capable of accomplishing more complex tasks, such as determining the optimal aim-point, or even predicting the angle of approach that will best limit the target's ability to deploy hard-kill countermeasures. As a result, it is entirely probable that intelligent Convolutional Neural Networks-based algorithms will be the brains that guide the next generation of infrared anti-ship missiles.

Chapter 2

2. Literature Review

Convolutional Neural Networks: a recommended direction for next-generation infrared anti-ship ATR algorithms

The state of the art for infrared anti-ship ATR algorithms is determined by the design and capabilities of the algorithms in question, and standard to which they are validated. This review presents a critical analysis of the current design requirements, capabilities, and validation standards that are relevant to infrared anti-ship ATR algorithms, as reported in the extant academic literature to date. Recommendations are then made for areas of algorithm development.

The two prime purposes of infrared anti-ship ATR algorithms are ship detection and ship classification, and these are typically developed as two distinct capabilities within a single system.

2.1. Ship detection

For infrared ship detection, the current state of the art is best represented by the Neural Network Tracker proposed by Gray *et al.* [114], which uses a neural network to provide advanced counter-countermeasure capabilities.

In detail, this algorithm segments potential targets from their background via adaptive global thresholding, based on the intensity values of a 10-pixel border, which results in binary white-hot silhouettes. Next, groups of contiguous pixels—which are treated as potential targets—are identified, and six scale-invariant shape features are extracted from each. These features are then inputted to single-layer fully-connected neural network which classifies each potential target as either a ‘ship’ or a ‘decoy’, enabling the system to reject any infrared countermeasures before calculating an aimpoint from the centroid of the largest ship object.

This system has been evaluated using CounterSim—an advanced missile-target engagement simulator developed by Chemring Countermeasures Ltd—and subsequently achieved 99.3% effectiveness in the presence of conventional infrared decoys. This value represents the best reported ship detection rate available and, for the purpose of this review, is considered to be the present-day state of the art [75].

2.2. Ship classification

For infrared ship classification, the current state of the art is best represented by the ANN-based classifier proposed by Kechagias-Stamatis, Aouf, and Nam [115], which can provide real-time identification of up to five types of navy vessel when trained on a very small dataset. The use of scale and rotation invariant shape descriptors and an ANN with multi-modal enabled the algorithm to be trained using just 6,884 vessel silhouettes.

Kechagias-Stamatis's algorithm operates in two stages: feature extraction, followed by feature classification. First, the ship is segmented from its surrounding background via global intensity thresholding and divided into six uniform partitions. Next, six low order Zernike moments, written in Equation 2.1, are extracted to form each partition to produce a total of six vectors. Since the magnitude of Zernike moments are dependent on their radius, each moment vector is normalised by area according to the scalar, w, which is calculated according to Equation 2.2.

$$a = [Z_2^0 \quad Z_2^2 \quad Z_0^4 \quad Z_0^6 \quad Z_1^7 \quad Z_0^8] \tag{2.1}$$

$$w = \frac{1}{5}\frac{\sum a}{\sum Z_0^0} \tag{2.2}$$

The features from each of the six partitions are subsequently combined into a single vector of length 36 and processed by an initial ANN which classifies the ship as either a merchant or naval vessel. If recognized as a merchant vessel, no further action is taken; otherwise, further analysis by a second ANN is used

to recognize the vessel as one of the five ship types that the algorithm has been trained with [115].

As with the Neural Network Tracker proposed by Gray for ship detection, Kechagias-Stamatis's ship classification algorithm was also evaluated using CounterSim, achieving an average classification accuracy of 94% at imager-ship distances of 3 – 7 km with an inference time of just 44ms. These metrics represent the best reported ship classification rates available and, for the purpose of this review, are also considered the present-day state of the art.

2.3. Current limitations

A series of limitations were identified in the ANN-based algorithms proposed by Gray and Kechagias-Stamatis, the first of which relates to the use of global intensity thresholding for target segmentation. Although both algorithms vary slightly in terms of how the threshold values are selected, each assumes that target thermal signatures are invariably more intense than the surrounding sea, sky, and coastline. Yet this assumption is frequently violated in real-world settings, for example when direct sunlight is reflected by the sea surface, or when operating within visible range of coastlines and coastal developments. Such scenarios can cause suitable threshold values to be overestimated, which may in turn cause the algorithm to reject legitimate target regions *and* falsely detect regions of background clutter.

Moreover, conventional global intensity thresholding techniques contain no mechanism for distinguishing between vessels when they overlap, which risks multiple overlapping vessels being treated as a single large ship. This limitation has been shown to severely disrupt the missile aim-point calculation of the Neural Network Tracker, causing overlapping vessels to be misclassified as decoys and falsely rejected [75]. These limitations are the principal reason why, after the optimisation of infrared decoy intensity and launch conditions, the effectiveness of Gray's Neural Network Tracker was found to drop dramatically to 42% [75].

Regarding coarse-grain vessel recognition, the Kechagias-Stamatis's algorithm is capable of distinguishing between five types of naval vessel, including patrol boat, minesweeper, and amphibious assault ship [115]. However, this accounts for fewer than half of the different military vessel types in operation around the world, meaning that, if deployed, this algorithm would be incapable of recognising the majority of the military ships at sea.

Furthermore, the recognition accuracy of Kechagias-Stamatis's algorithm was severely affected by the presence of similar vessels, for example images of a minesweeper were misclassified as a patrol ship at a rate of 19%. It is therefore important that future infrared ship classification algorithms are capable of classifying different types—and possibly even classes—of military ship with a finer level of granularity. The emergence of such fine-level classification, if robust, will have a considerable impact on the tactical capabilities of future anti-ship missiles.

Most critically, current detection and classification algorithms do not account for a wide variety of conditions at neither their design stage, nor their evaluation stage. Some approaches have been designed without mechanisms to handle target occlusion [75, 116, 117], while others only consider open ocean conditions without background clutter [118, 119, 120, 121, 122]. Moreover, the majority of studies consider only 'ideal' environmental conditions—such as clear skies and calm sea-states, with very little solar glare [123, 115]—and many more consider solely sea-skimming missiles and, consequently, assume no variation in the angle of elevation that the imager must calculate [75, 115, 124]. Given that so many possible conditions are unaccounted for, the algorithms referenced above are unlikely to be suitably robust for real-world deployment.

Consequently, attention has turned away from the use of generalist ANNs for infrared anti-ship ATR algorithms, and towards convolutional neural networks, which are better suited to processing structured data such as images but have not yet been used to construct a published state-of-the-art algorithm. As a result, any literature review of the field needs to also embrace the allied field of General Maritime Object Detection (GMOD), which is concerned with ship

detection and classification, though due to its purely civilian applications, does not need to consider the LW infrared waveband or military vessels.

In recent years, CNNs have overtaken traditional approaches for the task of GMOD, however, as shown in Figure 2-1, this trend is yet to be realised in the field of infrared anti-ship ATR, where the most effective algorithms remain reliant on hand-crafted features and conventional statistical classifiers. Consequently, GMOD is an important source of literature when considering the application and development of CNN-based approaches for infrared anti-ship ATR.

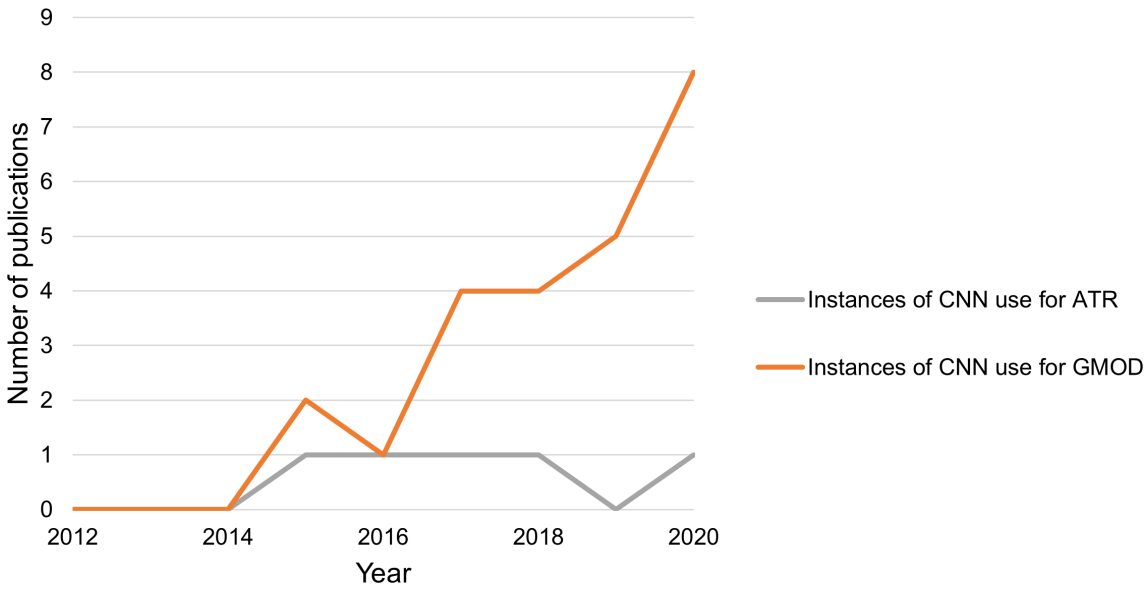


Figure 2-1: *A graph showing that the use of CNNs for GMOD have increased in recent years, but this trend is yet to be observed in the field of infrared ant-ship ATR.*

2.4. Can CNNs have an infrared anti-ship ATR application?

In 2019, Moosbauer *et al*. [125] published a comparison of two CNN-based algorithms—Faster R-CNN and Mask R-CNN [126, 127]—which were both trained for GMOD using the Singapore Maritime Dataset (SMD) [125, 128]. This study demonstrated the suitability of the Singapore Maritime Dataset for domain adaptation, and with Mask R-CNN, achieved state-of-the-art performance in both the visual and near-infrared spectrum. Example illustrations from the

evaluation of this algorithm are presented in Figure 2-2. Despite highly challenging evaluation conditions relative to those typical of infrared anti-ship ATR approaches, Mask R-CNN demonstrated consistently high performance across a broad range of conditions: it proved highly generalisable, even in instances of visual occlusion, and background and environmental clutter. Such approaches are yet to be successfully applied to infrared anti-ship ATR applications. However, once suitably trained for the task, they are expected to significantly outperform existing infrared vessel detection and classification approaches.

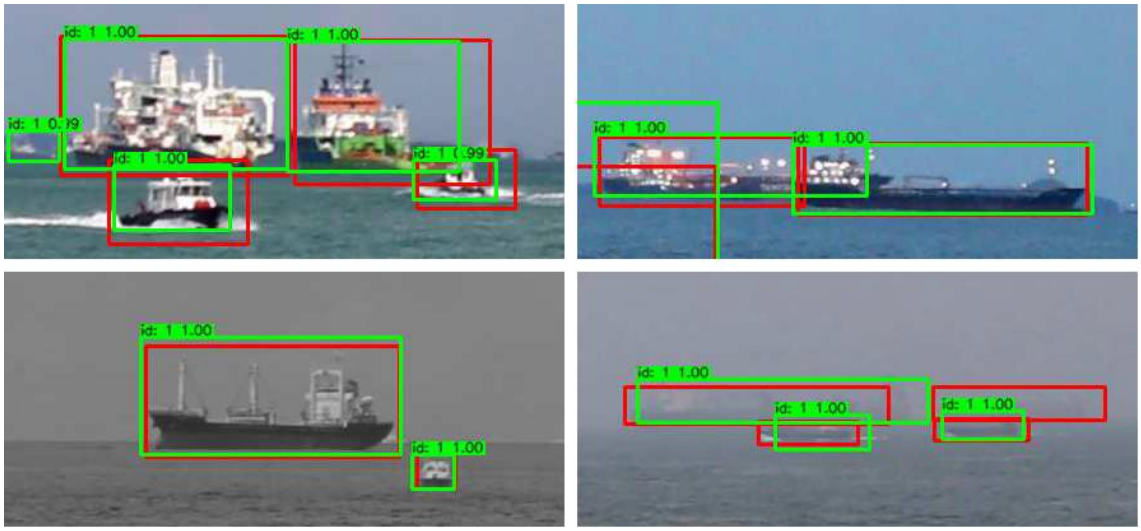


Figure 2-2: *A selection of images from the evaluation of Mask R-CNN with predicted and ground truth boxes drawn in green and red respectively. (Images sourced from [125].)*

In 2017, for example, an infrared ship recognition algorithm, consisting of convolutional layers from a pre-trained AlexNet CNN [129] combined with a Support Vector Machine classifier was proposed [115]. However, this approach failed to compete with existing conventional hand-crafted feature-based algorithms, with inadequate training data and feature corruption due to image resizing being hypothesised as the root causes of failure [115]. Consequently, at the time of writing, CNNs are yet to achieve comparable accuracy to that of traditional methods—such as [75, 118, 115, 130, 131, 132]—in the task of infrared anti-ship ATR.

There are four key challenges to the successful application of CNNs to the task of infrared anti-ship ATR. These four challenges are: 1) a lack of suitable training data, 2) inadequate standards of algorithm validation, 3) limited inter-algorithm comparison, and 4) an absence of suitable development environments. The remainder of this review analyses these challenges in depth, and provides recommendations.

2.5. What is ‘good’ training data?

The development of state-of-the-art detection, classification, and tracking algorithms is heavily reliant on large datasets, which enable deep learning-based algorithms to be trained. In recent years, the creation and publication of such datasets has promoted significant and sustained improvements to the state of the art in numerous computer vision fields. In 2015, for example, the release of ImageNet [133]—a dataset containing over 15 million labelled high-resolution images from *circa* 22,000 separate categories—facilitated comprehensive advancements in the field of general object classification. Advancements which included solutions to the vanishing gradient problem [134], the advent of new CNN architectures, such as AlexNet, GoogLeNet and VGG16 [129, 135, 136], enhanced regularisation techniques such as batch normalisation [137], and dropout [138] have all been made.

The field of general object detection has also seen rapid progress, which can be attributed, in part, to access to large datasets. A notable example is the Microsoft Common Objects in Context (COCO)—a dataset of 328,000 images with annotations for 91 everyday objects [139]—which has facilitated the development of numerous new detection algorithms, including FCOS [140], SSD [141], DSSD [142], RetinaNet [143], and variants of R-CNN [127, 144, 126] and YOLO [107, 145, 146]. The transformative effects of large datasets have also been observed in the field of human pose estimation with the MPII Human Pose dataset [147], urban scene segmentation with CityScapes and CamVid datasets [148, 149], and in multi-object tracking with the MOT16 dataset [150].

Access to large and diverse datasets is instrumental to the development of computer vision algorithms, and for that reason, this section provides a comprehensive review of relevant maritime image datasets. To be suitable for training and developing infrared anti-ship ATR algorithms, a dataset must be large, openly accessible, and provide LW infrared images of military vessels. With these requirements in mind, Table 2-1 presents a summary of all maritime datasets identified through the course of this review.

Table 2-1: *A summary of all relevant maritime datasets identified through the course of this review.*

Dataset	Military ships	Modality	No. images	Open access
Tremblay and Valin, 2002 [12]	Yes	LWIR	2,545	No
Withagen, 1999 [123]	Yes	LWIR	203	No
Wang, Bai and Zhang, 2014 [151]	Yes	LWIR	200	No
Herman, 2000 [117]	Yes	LWIR	18	No
Kechagias-Stamatis, Aouf and Nam, 2017 [7]	Yes	Synthetic LWIR	Undisclosed	No
[118]	Yes	Synthetic LWIR	11,520	No
Alves, 2001 [121]	Yes	Synthetic silhouettes	41,400	No
Zhao, Wang and Zhang, 2005 [152]	Yes	Synthetic silhouettes	3,600	No
MARVEL [153]	Yes	Visible	~2,000,000	Yes
E2S2-Vessel [154]	Yes	Visible	130,000	No
MRSP-13 [155]	Yes	Visible	37,161	No
McShips [156]	Yes	Visible	14,709	Yes
SeaGull [157]	No	LWIR, NIR, visible	~150,000	Yes
VAIS [158]	No	LWIR, visible	2,870	Yes
van den Broek, 2014 [159]	No	LWIR, MWIR	186	No
Wang, Lv and Xu, 2012 [160]	No	LWIR	180	No
Landsat-8 [161]	No	LWIR, SWIR, NIR	109	Yes
IPATCH [162]	No	LWIR, Visible	Undisclosed	No
Bouma et al., 2008 [80]	No	LWIR, MWIR, NIR, visible	Undisclosed	No
Singapore Maritime Dataset [125]	No	NIR, visible	31,653	Yes
ARGOS [163]	No	Infrared, visible	12,547	Yes
MarDCT [163]	No	Infrared, visible	Undisclosed	Yes
Annapolis [164]	No	Visible	58,365	No
SeaShips [165]	No	Visible	31,455	Yes[1]
Boat Re-ID [166]	No	Visible	5,523	Yes
Fefilatyev, Goldgof and Langebrake, 2007	No	Visible	100	No
Soloviev *et al.,* 2020 [167]	No	Visible	6,550	No
HSD [168]	No	Visible	Undisclosed	No

[1] At the time of writing, only 7,000 examples of the SeaShips dataset are openly accessible.

Of the maritime datasets surveyed, there are 12 which provide examples of military vessels through a variety of modalities: four contain LW infrared imagery, four contain visible-spectrum imagery, and four were generated synthetically. The largest of the four LW infrared datasets was compiled by Tremblay and Valin [169] and contains 2,545 examples. Containing six different military ships—two destroyers, a cruiser, a frigate, a land assault tanker, destroyer, auxiliary oil tanker—this dataset also provides the largest diversity of military vessels. Consequently, of all those surveyed, the Tremblay and Valin dataset is the most suitable for the training and development of new infrared anti-ship ATR algorithms. However, even if combined with the three alternative—and considerable smaller datasets of 203, 200, and 18 images—this dataset would not provide the quantity and diversity of examples required to train and develop modern CNN-based algorithms for infrared anti-ship ATR.

Despite its limited size, the dataset compiled by Wang, Bai, and Zhang [151] provides the greatest degree of diversity in the environmental conditions represented in its images. Images were collected in sunny, cloudy, and foggy conditions and also contain a range of complex background clutter, including coastlines and docklands. However, none of these LW infrared datasets provide comprehensive representations of military vessels at different imager-to-ship distances, angles of azimuth and elevation, and degrees of occlusion: and none have been made openly accessible. Fortunately, the complications arising from the collection of LW infrared imagery of military vessels could potentially be avoided by means of synthetic data generation.

2.6. The unmet need for synthetic training data

In response to the limited availability of LW infrared datasets which provide examples of military vessels, several attempts have been made to create new datasets computationally. Synthetic data generation provides an effective method of depicting any number of maritime vessels at many different scales and orientations, and is capable of yielding large datasets containing thousands of training images.

The first synthetic dataset for infrared anti-ship ATR was proposed by Alves, 2001, and used five computer-designed wireframe models to produce binary images of various maritime vessels, such as those shown in Figure 2-3. These models, however, were low fidelity, portraying only a simplistic representation of a few defining features for each vessel. Furthermore, being binary silhouettes, these images are high contrast, contain no background clutter, and do not account for variations in environmental conditions and ship thermal signatures that routinely occur in real-world scenarios.

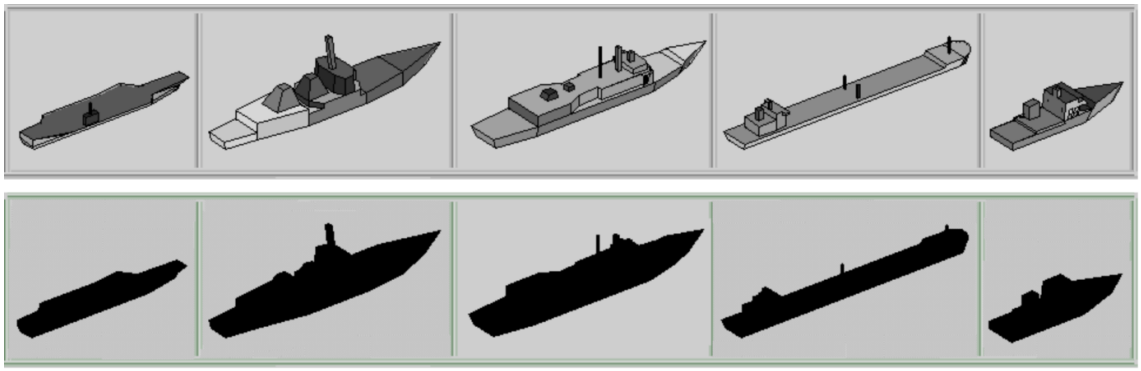


Figure 2-3: *The wireframe models and examples of corresponding silhouettes created by Alves, 2001. Models represent an aircraft carrier, destroyer, frigate, merchant ship, and research vessel respectively. (Image sourced from [121, 122].)*

Major improvements to the state of synthetic infrared data generation have been delivered through CounterSim, a virtual imager accounting for atmospheric transmission via MODTRAN4 [170], a version of the US Air Force atmospheric transmission, radiance and flux model [171]. This enabled the creation of the most advanced synthetic maritime dataset to date; a set of 11,520 LW infrared images of four military vessels (3 frigates and a corvette) generated by Gray *et al*., 2012. This dataset, illustrated in Figure 2-4, facilitated the development of novel ANN-based classifiers, capable of identifying military vessels and rejecting infrared countermeasures [75, 118, 114].

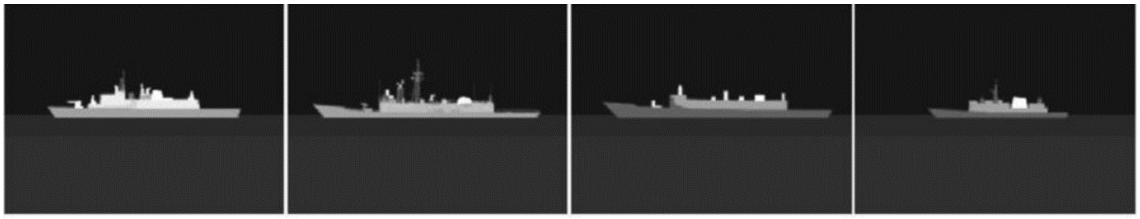


Figure 2-4: *Examples of the imager generated by Gray et al., 2012 of 3 frigates and a corvette using CounterSim. (Image sourced from [118].)*

Owing to its large size relative to real LW infrared datasets and depiction of military vessels, this is the most suitable dataset for the training and development of infrared anti-ship ATR algorithms. Nevertheless, this dataset has several limitations, for example the sea and sky are both modelled as constant, uniform surfaces and it does include any background clutter. Furthermore, like all other LW infrared datasets discussed above, it is not openly accessible and, consequently, cannot be readily applied for the purposes of the work described in this thesis.

2.7. In-depth findings from existing marine datasets

The remaining four datasets which depict military vessels are MARVEL, E2S2-Vessel, MRSP-13, and McShips [154, 153, 155, 156]. These datasets only provide visible spectrum data but are significantly larger than any LW infrared datasets and just two—MARVEL and McShips—have been made openly accessible. MARVEL, for example, contains over 2 million images of ships covering 109 types, including chemical tanker, suction dredger, and vehicle carrier. However, only a fraction of this data is representative of military vessels, which are depicted in only two of the 109 categories. Furthermore, this dataset is designed solely for ship classification, and does not provide annotations for the development of ship detection or segmentation capabilities.

The McShips dataset provides a larger range of military ship categories, as well as bounding box annotations for ship detection. Moreover, this dataset also ensures variations of scale, viewpoints, background, illumination, and atmospheric conditions. However, with *circa* 15,000 images, McShips is small

relative to other object detection datasets, which typically contain hundreds of thousands of image examples. Crucially, the absence of LW infrared imagery means that both MARVEL and McShips are not suitable for the development of infrared computer vision algorithms.

The remaining 14 datasets considered in this review do not depict any military vessels, though seven of these contain a mix of image modalities that includes LW infrared. The largest of these is SeaGull, which contains more than 150,000 LW infrared, near-infrared, and visual spectrum images. Collected by an Unmanned Aerial Vehicle of the Portuguese Air Force Research Centre, this dataset provides challenging vessel detection examples, with vessels pictured at various orientations and scales and in the presence of solar glare, wave crests, and wakes. However, these vessels are primarily small boats, such as sailing yachts, life rafts, and dinghies and, consequently, SeaGull is better suited for maritime search and rescue and surveillance applications than it is for anti-ship ATR.

Other notable mixed-modality datasets include VAIS, Landsat-8, and IPatch [162, 161, 158], each of which provide LW infrared images of maritime vessels. None, however, are suitable for military vessel detection. VAIS contains 1,000 co-located visual-LW infrared image pairs for the development of dual-mode algorithms, but only provides annotations in the form of ship type labels. IPatch depicts a limited selection of small cargo vessels, fishing boats, and Rigid Inflatable Boats, and is directed towards the development of anti-piracy systems via the detection of criminal behaviour. Wang *et al*. [161] compiled a dataset of images collected by Landsat-8, a near-polar orbit Earth observation satellite [161]. However, it contains only images showing a high-altitude perspective of vessels, and is not suitable for the development of missile seeker algorithms.

The nine remaining datasets do not contain any LW infrared imagery or military vessels, and most of these are either not openly accessible, contain only a few thousand images, or do not represent appropriate scenarios. Containing over 31,000 images complete with bounding box annotations and pre-defined training, validation, and test data partitions, the Singapore Maritime Dataset is

the most suitable dataset for use as a ship detection benchmark. However, while this dataset contains near-infrared and visual spectrum images, it does not contain and LW infrared imagery, nor depict any military ships, and is therefore not suitable for training infrared anti-ship ATR algorithms.

In summary, there are currently no datasets which are well suited for the development of CNN-based infrared anti-ship ATR algorithms. Only four LW infrared datasets have been made openly accessible, none of these provide examples of military vessels, and while other open datasets do include examples of fighting ships, each of these does so only in the visible spectrum. Numerous CNN-based algorithms have been applied to maritime applications, including surveillance, and anti-piracy [162, 172]. Yet owing to a lack of suitable training data, none have yet been trained for the detection of military vessels through LW infrared imagery.

2.8. Overcoming training data limitations

The absence of suitably large and diverse datasets is currently the primary obstacle to the application of CNNs to infrared anti-ship ATR. Three solutions to this current lack of suitable data include: (1) the collection and annotation of new, real-world datasets [163], (2) the combining of multiple pre-existing datasets [163, 173], and (3) the computational generation of new synthetic datasets [121, 115, 152, 118].

Most large image datasets which are openly available have been collected and annotated through sustained efforts by various research consortiums [139, 133]. In the field of infrared anti-ship ATR, this manual approach is yet to yield any datasets which contain more than a few thousand images [169]. This is, in part, due to the large cost and logistical challenges associated with the collection of infrared imagery in maritime environments. Equipment used for collection of the VAIS dataset, for example, cost *circa* $40,000 (US) to purchase and assemble. The main prohibitive factor, however, is the security issues which surround the collection of infrared images of military ships. As a result, the compilation of a large infrared dataset which depicts numerous different fighting ships is a

significant challenge and the permitted publication of such a dataset is undoubtedly problematic. Consequently, the creation of new and more suitable datasets will necessitate an alternative approach.

New, larger datasets could be created by combining existing maritime datasets with elements of general object detection datasets [174, 175, 163, 176, 177, 178, 173]. For instance, Spraul, Sommer and Schumann, 2020 combined extracts from COCO, ImageNet, SeaShips, Singapore Maritime Dataset, VOC, and YouTube [179, 139, 125, 133, 165] to produce a compilation of 19,380 annotated image examples for ship detection in the visual spectrum. Taking advantage of readily accessible data, this provides an efficient approach to the compilation of large, diverse, and consistently annotated training datasets.

A number of these datasets [179, 180, 139, 173], however, are compiled via keyword searches of image and video hosting sides, such as Shipspotting.com, YouTube, and Flickr and, as a result, suffer considerably from capture bias [181]. This stems from the fact that photographers tend to capture images in similar ways, for example, the subject is typically centred, well-lit, un-occluded, and clearly focused. As a result, these images are not representative of the conditions with which computer vision systems will be presented upon deployment. Moreover, this approach relies exclusively on access to existing data which is suitable for the task. Even if all existing LW infrared datasets of military vessels could be accessed and combined, the resultant dataset would provide only *circa* 3,000 examples. Alternatively, generalised object datasets, such as COCO, ImageNet, and VOC [179, 180, 139, 173], provide a much large source of data, yet contain no LW infrared imagery, few military vessels, and only provide very broad categorisation.

So far, the largest LW infrared dataset of military ships have been created via synthetic data generation [118]. This approach has yielded tens of thousands of images of fighting ships at all scales and angles of azimuth and elevation, yet even the most realistic dataset lacks diversity in terms of ship thermal signatures, background environments, and atmospheric conditions [75]. Furthermore, even if these aspects of variation are accounted for in future

synthetic dataset, it is unknown whether this would yield a dataset which can adequately train modern CNN algorithms.

2.9. The need for better algorithm validation and testing

Validation and testing are fundamental to the development and application of computer vision algorithms. Though these terms are occasionally used interchangeably, in actual fact, validation and testing refer to two distinct processes with subtly different purposes. Both involve the evaluation of an algorithm with respect to pre-defined sub-sets of example data, but in the case of validation, this is conducted solely for the purpose of algorithm development, such as hyper-parameter selection. Testing, on the other hand, is a separate task which aims to establish an unbiased estimate of the performance of a finalised algorithm, and therefore, is an essential precursor the safe and reliable deployment of computer vision algorithms. Robust algorithm validation and testing requires two distinct data sets with similar distributions which are both representative of deployment conditions.

The predominant issue with the current standard of infrared anti-ship ATR validation and testing is that the example data regularly fails to adequately represent the challenging and varying nature of real-world conditions. A large majority of test datasets, for example, depict only a narrow selection of naval vessels in calm, open water and in near-ideal atmospheric conditions [117, 123, 75, 182]. In terms of environmental conditions, the most varied and challenging dataset is that compiled by Wang, Bai, and Zhang [151] that depicts vessels in sunny, cloudy, and foggy conditions and includes a range of background clutter. Yet these conditions are not challenging in every respect, for example, ambient and sea-surface temperatures remain constant throughout the dataset and a strong contrast between targets and background is present across the majority of examples.

In terms of military vessels, most diverse datasets each depict five or six different military vessels [118, 169, 152]. However, this represents just a tiny fraction of the several thousand different classes of military ship currently in

service with navies around the world [62]. Moreover, the thermal appearance of any individual ship can vary significantly with onboard and atmospheric conditions, yet in general, image collection of each vessel has been conducted at a single location and point in time. Consequently, existing validation and test datasets provide minimal variation in ship thermal signatures between examples.

As a result, existing validation datasets cannot inform algorithm development in such a way that improved robustness against all adverse conditions can be assured, and test datasets cannot provide reliable estimates of real-world performance, and cannot be relied upon to yield reliable, unbiased estimates of real-world performance.

In addition to the limited diversity of vessels and environmental conditions, all current validation and test datasets are also unsuitably small—typically containing fewer than 200 examples. Such datasets, even if they are relatively diverse, are too small to provide the comprehensive representation of expected scenarios that is necessary to provide unbiased estimates of real-world performance. In addition, limited dataset size is known to have adverse effects on the validation and test procedures themselves. Due the lack of suitable evaluation data, a large majority of infrared anti-ship ATR studies opt to forego the use of validation sets and, instead, assign as many examples as possible to the test set [117, 119, 169, 118]. However, the development of computer vision algorithms without guidance from interim validation is both challenging and impractical and it is, therefore, likely that validation was conducted using test data in some cases. This can lead to algorithms becoming tailored to the specifics of the test data, and this is a source of bias which can invalidate the testing process [183, 184].

2.10. How can algorithm validation and testing be improved?

Cross-validation is an alternative solution which aims to minimise the bias associated with testing using small datasets. Here, all available data is divided into k groups, or folds, of equal size and the algorithm is evaluated for each fold

in turn, while the remaining $k-1$ folds serve as training data [185]. As a result, cross-validation can provide a much less biased estimate of performance than other approaches, such as a single train/test split, particularly when the amount of available example data is limited. This approach to validation was adopted by Teutsch *et al.* [182], where cross-validation enabled the thorough and relatively unbiased evaluation of a Support Vector Machine-based ship classifier using just 1,877 example images.

While cross-validation can significantly reduce the bias which stems from evaluating with small quantities of test data, it cannot make up for a lack of scenarios which are not present in the available data. Moreover, a significant portion of infrared maritime examples have been extracted from video sequences [128, 159, 186], and in such cases, cross-validation, as well as other methods of training and evaluation data allocation, are likely a further source of bias. This is because, due to being captured in quick succession, a large portion of frames are highly similar. For example, the same ship and background clutter may be present throughout an entire sequence, with only marginal variation in perspective and illumination. Splitting such a sequence using cross-validation would result in many aspects of the depicted scenario being present in both the training and evaluation sets and would likely result in a biased overestimate of model performance. Consequently, cross-validation is often not wholly appropriate, particularly when dealing with a small number of video sequences, where careful hand-selection of the training, validation, and test split may be more suitable [125].

Currently, a considerable majority of validation and test datasets contain sensitive information, and so cannot be made openly accessible to the wider research community. While endeavours to compile such datasets aid the development of new infrared anti-ship ATR algorithms, they do little to improve the standard of validation and testing within the field of infrared anti-ship ATR in general. A potential solution may be found in the emergence of Privacy Preserved Machine Learning, which enables algorithms to be tested with encrypted data which is hosted securely [187, 188]**.** Such techniques would

significantly increase the quantity and representativeness of validation and test data available for the evaluation of infrared anti-ship ATR algorithms, and thus contribute to the safe and reliable deployment of CNN-based seeker algorithms.

Crucially, synthetic data has enabled the performance of ATR algorithms to be analysed under tightly controlled and repeatable conditions which would be logistically challenging to reproduce in the real world. However, existing synthetic datasets, while representative of real infrared imagery in many respects, are unable to serve as a reliable substitute for real-world examples. The effect of this has been observed by Alves, Herman, and Rowe [122], whose infrared ship recognition algorithm achieved an f-score of 87% when evaluated using synthetically generated silhouettes, yet in subsequent trials involving real-world examples, the f-score decreased considerably to just 70%. This illustrates how synthetically generated infrared imagery is not conducive to reliable evaluation without sufficient justification and awareness of the dataset limitations, which at the time of writing, are yet to be thoroughly discussed in ATR literature.

Consequently, standards of validation and testing in infrared anti-ship ATR literature are inferior to those of similar fields of computer vision research [139, 189, 150]. This is primarily due to the limited size and quality of current validation and test datasets which, though crucial for the development and evaluation of infrared anti-ship ATR algorithms, provide only a limited representation of expected conditions, and contain too few examples to fulfil their role effectively. In addition, the validity of algorithm testing is commonly undermined by failures of test procedure, such as validating with the test data, or the duplicating similar examples across training, validation, and test sets. Such deficiencies lead to biases, which in turn cause the overestimation of algorithm performance that can impede development and hinder the safe, reliable deployment of ATR algorithms. In response, it is important that efforts to compile real-world test sequences are continued, and that such sequences are made openly accessible where possible. In addition, the application of Privacy Preserve Machine Learning should be investigated as a means of facilitating testing with datasets what contain sensitive information.

The creation and use of synthetic data is a long-established solution to the scarcity of suitable real-world examples that is currently impeding the development and deployment of new infrared anti-ship ATR algorithms. However, though synthetic data enables algorithms to be tested under tightly controlled conditions, existing datasets are not yet fully representative of real-world conditions and can lead to the overestimation of algorithm performance. Consequently, following the example set by Alves [121], it is recommended that both synthetic and real-world examples are used collectively for algorithm testing, and in addition, that any disparities between resultant performance estimates be analysed and discussed. Such discussions will serve to improve the suitability of future synthetic test data and the reliability of subsequent algorithm testing.

2.11. Comparing ATR algorithms

Direct algorithm comparison plays a significant role in the development of many computer vision research fields including general object detection and classification, yet when considering infrared anti-ship ATR approaches direct comparison is often not possible. Consequently, discussion now turns to the current obstacles preventing the direct comparison of infrared anti-ship algorithms, identifying three solutions to promote comparison in order to boost development of the field.

Direct comparison is made possible through three factors: 1) access to large and diverse training datasets, 2) the existence and availability of universally accepted benchmark test datasets, and 3) the administration of impartial testing and leader boards by research consortia. Open access to large training datasets promotes direct comparison by enabling the development of algorithms under controlled conditions. In turn, this ensures that disparities in performance result from the mechanisms of the algorithm itself, not the training data. Second, the existence of a universally accepted benchmark test promotes and enables the evaluation of algorithms under equivalent, repeatable conditions. Third, benevolent oversight serves to update test datasets and

metrics, maintain impartial test servers, and publish the subsequent valid and comparable performance scores.

However there exists no large and diverse training datasets for infrared anti-ship ATR and subsequent test procedures are typically not reproducible. Further, due to security concerns, it is unlikely that any institution will ever host a universally accepted benchmark dataset, nor publish subsequent performance scores. As a result, only a minor portion of infrared anti-ship ATR research can be directly compared. This has the effect of hindering any meaningful interpretation of reported performance metrics, thus obscuring the limitation and capabilities of the state-of-the-art and impeding the advancement of existing approaches.

Due to concerted efforts to produce directly comparable research, the state of the art in numerous fields can be clearly ascertained. In the field of general object classification, for example, Top-5 Accuracy scores have surged from 73.8% in 2011 to nearly 98.8% in 2020 [189, 190, 191] while general object detection Average Precision scores have increased by 30% since 2016 [192, 141, 193].

Since the direct comparison of infrared anti-ship ATR research through a combination of openly accessible training data, universally accepted benchmark test sets, and impartial oversight is unlikely for the foreseeable future, alternative practices must be implemented to enable direct comparison. Therefore, this review identifies the following three best practices as necessary for the direct comparison of infrared anti-ship ATR research: 1) the open-source publication of algorithms, 2) the publication of comprehensive documentation, and 3) reproducible algorithm testing.

First, ATR algorithm source code should, where appropriate, be made openly available for further testing and development by the wider research community. Openly publishing algorithm source code delivers numerous benefits to research fields, and expedites the adoption of new techniques while helping to eliminate bias and define new standards [194]. Critically, the open publication of source code also enables independent verification through subsequent studies

by impartial research groups. This can be seen in the field of general object detection, where the open publication of source code has become common practice, enabling detection algorithms to be evaluated and compared independently across a wide range of applications, including maritime object detection [195, 196, 197, 198, 173, 199]. Consequently, the open publication of source code relating to infrared anti-ship ATR will dramatically reduce the need for a universally accepted benchmark test for infrared anti-ship ATR algorithms.

Second, ATR algorithm documentation should be sufficiently comprehensive so as to permit the accurate reproduction of results. Despite active encouragement by academic publishers, such as Nature, IEEE, and Papers With Code [200, 201, 202], the open publication of new infrared anti-ship ATR algorithms remains uncommon. Consequently, in cases where source code is not published, the supporting literature is key to enabling the algorithm to be independently recreated for evaluation comparison, and further development by the wider research community. However, the accurate recreation of a detection algorithm from documentation typically necessitates considerable skill and effort, thus it is preferable that comprehensive documentation is published in combination with algorithm source code.

Finally, ATR algorithm evaluation should be conducted in a reproducible fashion and reported in full, so that alternative algorithms may be evaluated under equivalent conditions. However, infrared anti-ship ATR algorithm evaluation, as reported by the bulk of recent literature, is not typically reproducible due to insufficient reporting of test conditions, such as the performance metric definitions, training, validation, and test data partitions. Moreover, reproducible evaluation requires that the test data used is openly accessible, yet this cannot presently be the case since most datasets suitable for infrared anti-ship ATR are held privately.

2.12. The need for a generalist deep learning environment

ANNs have been a key component of infrared anti-ship ATR algorithms for decades [203, 117], however the architecture of these algorithms are yet to

progress much further than simple 2-hidden layer networks [124]. In recent years, only limited advances in deep learning-based approaches have been achieved, and none of these feature the application of CNNs. To bring together the recommendations set out above, developing the complex deep learning project structure is the initial and challenging task.

As illustrated in Figure 2-5, the training, validating, and testing deep learning algorithms necessitates the implementation and integration of multiple software components into a single, interconnected system. Moreover, training deep learning algorithms also entails differentiation of complex objective functions with respect to thousands, if not millions of different annotated examples. Consequently, the effective use of both hardware *and* software resources is essential to ensuring that experiments are developed efficiently and are reproducible, well structured, and computationally efficient.

This review highlights that infrared anti-ship ATR algorithm validation and testing is often inadvertently invalidated, uninformative, and or not repeatable. Due primarily to an absence of suitable test data, a lack of standardised test procedures, and inadequate reporting, significant barriers remain to the safe and reliable deployment of emerging ATR algorithms.

The advent and utilisation of an appropriate deep learning development environment will play a crucial role in overcoming these issues and advancing the state of deep learning-based ATR. Such an environment would overcome significant barriers in algorithm development by ensuring that research is developed efficiently and is reproducible, well structured, and computationally efficient.

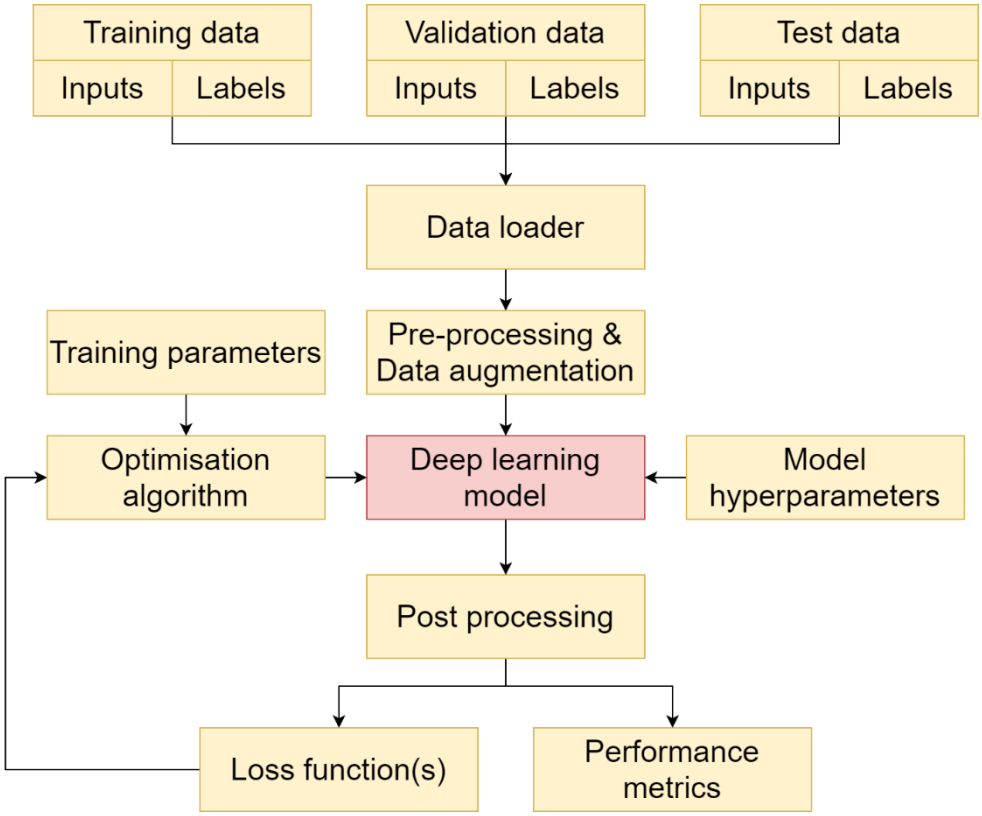


Figure 2-5: *Illustration of how multiple different processes and software components must be compiled into a single interconnected system in order to train, validate, and test a deep learning algorithm.*

2.13. Conclusion

The classical state of the art for infrared anti-ship ATR relies on generalist ANN methods such as intensity thresholding and hand-crafted features. However, these methods limit the development of detection and classification algorithms for infrared anti-ship ATR applications since the learning of optimal features that represent infrared ships is not automated. Consequently, CNN-based algorithms have the potential to overcome the limitations of current state-of-the-art algorithms that, for example, do not recognize a variety of backgrounds or environmental conditions.

In summary, this review has analysed four key reasons why CNNs are not prevalent in infrared anti-ship ATR. These are: 1) the lack of suitable training data; 2) the lack of suitable validation data; 3) the absence of algorithm cross-

comparison; and 4) the absence of a suitable open-access deep-learning development environment.

The remainder of this thesis addresses each of these issues in turn, through the course of four technical chapters. Consequently, we now turn to address the lack of suitable training data, through the development of a synthetic LW infrared dataset of military vessels, named IRShips.

Chapter 3

3. Generating an infrared dataset of military ships

This chapter describes the generation of a new synthetic LW infrared dataset of military ships for training deep learning-based ATR algorithms. Named IRShips, this dataset was made openly available on the Cranfield Online Research Data (CORD) repository in November 2020 [204] and has since been downloaded over 8,000 times across 20 different countries.

The purpose of a training dataset is to enable CNNs to 'learn' a hierarchy of features that can then be used to model abstract visual concepts. As the quantity and diversity of the training data increases, these features become more and more generalised, and this in turn enables the CNN to perform robustly across an increasingly varied range of conditions. But determining precisely how large and how diverse such a dataset should be is not an exact science, and is strongly influenced by the complexity of the task and the algorithm in question.

In terms of size, Goodfellow, Bengio and Courville state in their book Deep Learning [108] that as a general rule of thumb, supervised deep learning algorithms will achieve a performance that they regarded as generally acceptable with around 5,000 labelled examples per category, and such algorithms will match or exceed human-level performance when the number of training examples surpasses 10 million.

Diversity, though, is much more difficult to quantify, and must be considered in terms of the expected deployment conditions. This is because, in order to create features which can be generalised across a certain condition—such as object colour or scale—the training data must provide sufficient variation of this object's colour or scale. For instance, a CNN which is trained to recognize ships using a training dataset of broadside images could not be expected to recognize a ship pictured from the rear.

In addition to size and diversity, datasets must be suitably labelled so that the error of the CNNs can be quantified during training, and subsequently used to inform updates of the algorithm's internal parameters. Therefore, training a CNN to perform both object detection *and* classification requires each training image to be accompanied by ground-truth object-class labels as well as ground-truth object-localisation labels, such as bounding boxes or pixel-wise masks.

Finally, it is also crucial that the training data is from the appropriate band of the electromagnetic spectrum, be that the visual, microwave, near-infrared, or LW-infrared waveband.

This chapter outlines the development of a synthetic infrared dataset that meets these various requirements, and which—at the time of writing—is the largest extant dataset of its type, containing 972,000 fully-labelled images of 10 different maritime vessels.

3.1. Technical specifications

Based on these general requirements, this section considers the specifics of infrared missile-ship engagements to create a list of technical specifications for the new dataset. These specifications fall under four categories: optics, targets, environment, and annotations, and ensure that the resultant dataset is capable of training CNNs to effectively detect, recognize, and identify military vessels in infrared imagery.

3.1.1. Optics

For the constituent images of a synthetic infrared dataset to be generated computationally, it is crucial that the parameters controlling this computation relate to an optical system which is practicable for use in an infrared anti-ship missile seeker system.

First, the simulated images must correspond to the 8–14 μm waveband, meaning that the process of infrared emission and atmospheric transmission within this band, as discussed in Section 1.2.2, must be accounted for. Second, optical parameters such as F-number and field-of-view, discussed in Section

1.2.3, must correspond to an optical system which is compatible with the size limitations and expected performance requirements of both current and likely next-generation anti-ship missiles. Finally, the images must be of a resolution that is achievable using current sensor technology, and practical for use by a modern imaging infrared missile seeker. To date, the synthetic infrared maritime dataset with the highest resolution imagery is that of Gray [75], which contains 320 × 240 pixel images. However, modern high-definition LW infrared cameras have since become more widespread, and are likely to feature on the next generation of infrared missile seekers. Consequently, images with resolutions greater than 320 × 240 pixels are required.

3.1.2. Targets

In terms of target recognition, the development of a synthetic infrared dataset requires considering the number of images that are required, the number of different ship classes and types that are required, how best to represent these ships from a modelling perspective, the thermal characteristics of the ships in question, and their orientation. Each of these is discussed in turn.

3.1.2.1. Number of images

To date, the largest LW infrared dataset of military vessels is that of Gray *et al*. [118], which contained a total of 11,520 synthetically-generated images of four different military vessels. However, neither this dataset, nor any others, has been found to be sufficiently large for training CNN-based algorithms to successfully detect and classify maritime vessels in real-world LW infrared images. Consequently, any new dataset must provide considerably more training examples than all existing alternatives: at least 5,000 images per ship class, as suggested by Goodfellow, Bengio, and Courville [108].

3.1.2.2. Number of different ship categories and classes

In modern naval parlance, some ambiguity surrounds the words ‘class’ and ‘type’, with the two sometimes being used interchangeably, and sometimes not. From a taxonomy point of view, this can lead to obvious confusion. Accordingly, the word ‘type’ has been used to designate a higher-order taxonomical

classification, and ‘class’ as the corresponding lower-order taxonomical classification.

To date, the largest variation in ship taxonomy provided by an infrared ATR-oriented dataset is that of Tremblay and Valin [169], which depicts eight different ship categories: six military categories (consisting of frigate, cruiser, land assault tanker, destroyer, auxiliary oil tanker, and destroyer with guided missile); and two civilian categories (container vessel and freighter). It is therefore desirable that any new dataset contains no fewer than eight ship categories, and ideally more. Consequently, a decision was taken that the dataset on which this body of research is based should include three different classes of military vessel within each of three broad category designations, giving nine different fighting ships in total, and one civilian category. It was further decided that the ship classes in question must be suitably representative of the variation within each broad category designation, and must be modern vessels which are in active service.

3.1.2.3. Ship modelling

Each class of ship depicted in a synthetic dataset must be represented by a Computer-aided Design (CAD) model which is suitably detailed, and which accurately depicts the key distinctive features of the chosen vessels, including masts, radomes, flight decks, and main weapon systems.

It is also important that this high level of fidelity is consistent between CAD models, since any significant disparity in the quality of the models could be identified by a CNN during training and subsequently used by the algorithm as a spurious discriminatory feature. Nor is this a purely theoretical risk, as one recent study [118] makes clear. The infrared ship classification algorithm in question was trained to identify Adelaide-class, Oliver Hazard Perry-class, and Meko 200-class frigates, and successfully did so—but the distinguishing characteristics for the Adelaide-class turned out not be its real-life distinguishing characteristics, but the poor quality of its CAD representation.

3.1.2.4. Thermal signatures

To date, all existing synthetic infrared anti-ship ATR datasets feature ships with unvarying thermal properties. In reality, however, the surface temperatures of maritime vessels will vary continuously according to the prevailing onboard and environmental conditions, giving rise to differing thermal signatures. In the glaring noonday sun of the Mediterranean, a ship will have one thermal signature; in the near-perpetual darkness of the North Atlantic winter, quite another.

Consequently, it is important that anti-ship missiles must be capable of detecting—and possibly classifying—military vessels irrespective of their thermal appearance, calling for synthetic infrared datasets to contain a suitable degree of variation in the thermal signatures of target vessels.

3.1.2.5. Orientations

An anti-ship missile can approach its target from over the horizon, and may view the target from any bearing. Furthermore, through the course of the engagement, the vessel may manoeuvre in an attempt to evade the threat. To account for this, it is important for synthetic infrared datasets to contain ships depicted from all bearings from 0–359°, and from a suitable selection of different imager-to-ship distances.

In addition, ships must also be depicted from a variety of different elevations. A significant proportion of existing ATR-oriented maritime datasets assume a sea-skimming missile and, accordingly, only depict ships from a near-sea-level perspective [114, 115, 128, 159]. Such an assumption may have been appropriate in previous decades, but as discussed in Section **Error! Reference s ource not found.**, the trajectory of modern anti-ship missiles can be highly variable. Though the precise profile of the resulting trajectories is classified, modern anti-ship missiles are likely to perform a terminal bunt. Again, it is important that any synthetic infrared dataset adequately captures these differing perspectives.

3.1.3. Environmental conditions

Variations in environmental conditions—such as sea-state or weather—can have a significant impact on the thermal appearance of an engagement scenario. With sea states ranging from completely calm to waves in excess of 14 metres, reflected solar glare, and temperatures ranging from sub-zero to over 30 °C [205], the thermal appearance of the sea surface can vary considerably. The same is true for atmospheric conditions, where factors such as ambient temperature, humidity, and cloud cover also have a significant effect on thermal appearances. In previous work, for instance, clouds have been identified as a major impediment to the performance of seeker algorithms in simulated infrared missile-ship engagements [75].

However, recent synthetic infrared maritime datasets have generally modelled the sea as a flat surface of uniform and constant temperature, while others have assumed no variation in ambient temperature [118, 124, 115]. Since anti-ship missiles are expected to perform under a wide range of adverse environmental conditions, it is critical that any synthetic infrared dataset captures this variety of sea-state and atmospheric conditions.

It is also important to consider background clutter as a source of variation. Existing missiles, such as the Penguin and NSM, are designed for use in coastal environments, where coastlines, harbours and even navigation buoys have the potential to distract a missile's infrared seeker. Therefore, in addition to sea-state and weather conditions, it is also important that any synthetic dataset includes a wide variety of feasible background clutter for the sake of developing robustness in such conditions.

3.1.4. Annotations

To be suitable for the development of supervised deep learning algorithms, each image must be annotated with the location, type, and class of the ships it contains.

There are two methods for annotating the location of objects within an image: bounding boxes, and pixel-wise labelling. Under the pixel-wise approach, each

and every pixel in an image is assigned to either its corresponding object or the background in order to provide precise description object locations. Bounding boxes, on the other hand, are parameterised by the cartesian coordinate of the object centroid along with a width and height value and, therefore, are simpler to create manually but offer a reduced level of precision.

Given the computational generation of a synthetic infrared dataset, labels should be generated automatically, and—ideally—both bounding box and pixel-wise annotations should be created.

3.1.4.1. Metadata

In addition to the location, type, and class labels which are necessary for training, each image should also be annotated with comprehensive metadata, such as imager-to-ship distance, ship orientation, sea surface temperature, and more. Such information is important for characterising the performance of detection algorithms and identifying systematic errors.

3.1.5. Summary

Taken together, it is clear that a synthetic infrared dataset should possess a diverse set of characteristics, and these are all presented in Table 3-1.

Table 3-1: *The proposed specifications for an effective synthetic infrared dataset.*

Optics	
1.1.	Images must correspond to the LW infrared (8–14 μm) domain and, as such, must accurately account for emission and atmospheric transmission in this waveband.
1.2.	Images should have a resolution which is greater than 320 × 240 pixels.
1.3.	All optical parameters, including focal length, field-of-view, and detector array size should be compatible with the size limitations and expected performance requirements of current anti-ship missiles.
Targets	
2.1.	The dataset should contain at least three different types of military vessel.
2.2.	The dataset at least three classes of ship for each type of military vessel.
2.3.	The dataset should contain at least 5,000 examples of each class of ship.
2.4.	All military vessels should currently be in commission.
2.5.	Together, selected classes should be representative of the typical variation that exists within their type designation.
2.6.	All CAD models should be of a consistent and high fidelity.
2.7.	The dataset should present various thermal signatures within each class of vessel.
2.8.	Each different ship thermal signatures should be suitably realistic.
2.9.	Imager-ship distances should vary in range up to at least 10 km.
2.10.	Ship bearings should vary in the range 0–360°.
2.11.	Imager elevations should vary in the range 0–20°.
Environment	
3.1	Ambient temperature should vary.
3.2	Sea surface temperature should vary.
3.3	Sea state should vary.
3.4	Sky state conditions should vary.
3.5	A variety of feasible background clutter should be included.
Annotations and Metadata	
4.1	Class and type labels must be provided for every ship.
4.2	Bounding box and or pixel-wise labels must be provided for every ship.
4.3	Comprehensive metadata should be available for every image.

3.2. Dataset generation

The process of developing a synthetic infrared dataset of military vessels that complies with these requirements consisted of five key stages:

1. Create CAD models of military ships,
2. Apply thermal properties to the surfaces of each model,
3. Generate synthetic LW infrared imager of each model,
4. Compile the resultant images into a structured dataset, and
5. Implement a data augmentation pipeline to further increase image complexity.

First, to serve as the basis of the dataset, a representative range of various military vessels was selected, and a series of high-fidelity CAD models were created of them. Second, nine distinct sets of thermal properties were generated, and then applied to the surfaces of each model. Next, a total of 972,000 synthetic infrared images of these models were generated using the advanced missile-target engagement simulator, CounterSim. Subsequently, a number of post-processing steps were applied to produce a structured dataset with comprehensive metadata. Finally, a data augmentation pipeline was developed to enhance the complexity and diversity of the images, and achieve greater representation of expected real-world conditions.

3.2.1. Creation of ship models

Creating the synthetic infrared images in the dataset involves two processes: selecting a representative range of ships, then creating suitable high-fidelity CAD models of these vessels.

3.2.1.1. Ship selection

In accordance with technical requirements 2.1 and 2.2 from Table 3-1, three classes of corvette, frigate, and destroyer were selected. In addition, roll-on/roll-off (RORO) vehicle ferry—the MV *Armorique*—was included to provide a representation of a civilian vessel.

The three corvettes selected were the Ada-, Independence-, and Visby-classes of the Turkish, United States, and Swedish navies respectively, pictured in Figure 3-1. Though there is no universal definition of a modern corvette, the term is generally understood to describe the smallest class of vessel considered be rated a warship, typically displacing between 500 and 3,000 tonnes. Smaller corvettes are intended as small, fast, missile-armed attack craft optimised for anti-surface warfare, such as the Visby; whereas larger variants, such as Ada and Independence, operate as small frigates with moderate speed, designed to escort fleets of larger ships. As per specification 2.5, this selection of vessels is representative of the two emerging doctrines for the type and, as per specification 2.4, all the vessels selected are in commission at the time of writing.

It should be noted that the official type designation of the Independence-class warships is actually ‘littoral combat ship’. However, this term is used exclusively by the United States Navy, and therefore, the Independence-class of warships are variously referred to as either corvettes or frigates by navies which do not use this term. Though Janes Fighting Ships [62] refer to Independence-class warships as frigates, they are significantly smaller than conventional frigates with their 3,188 tonnes displacement being more comparable to corvettes. Moreover, the Independence-class vessels are designed for multi-purpose operations in coastal zones, a role closer to that of corvette than the fleet defence role assumed by frigates. Consequently, for the purposes of this thesis, Independence-class vessels are considered to be corvettes.

Figure 3-1: *Examples of Ada- (top left), Independence- (top right), and Visby- (bottom) class corvettes. (Images sourced from [206, 207, 57].)*

Of the frigates, two classes were selected—the Alvaro de Bazán- and Oliver Hazard Perry-classes, pictured in Figure 3-2—because high-fidelity CAD models of these vessels already exist as part of the CounterSim engagement simulator. The Alvaro de Bazán-class contains five air-defence frigates, the first of which was commissioned into service with the Spanish Navy in September 2002. The Oliver Hazard Perry, on the other hand, is comparatively older class of vessel which was first commissioned into service with the United States Navy in December 1977. Despite its age, however, it remains a ubiquitous ship, with 71 being built between 1975 and 2004 and, though no longer in service with the United States Navy, 34 of the class remain in service with the navies of Taiwan (10), Turkey (8), Spain (6), Egypt (4), Poland (2), Chile (2), Pakistan (1), and Bahrain (1) [208, 209, 210, 211, 212, 213, 214, 215].

The third frigate selected was the Type 054A Chinese multi-role frigate (NATO codename Jiangkai II), also pictured in Figure 3-2. First commissioned into the People's Liberation Army Navy in January 2008, the Type 054A is thought to be the most advanced in the Chinese Navy, and is one of the most numerous fighting ships in the world, with 30 of the class currently in active service and a further six under construction—two for the People's Liberation Army Navy and four under order by the Pakistan Navy as the Type 054AP [216].

Figure 3-2: *The Alvaro de Bazán (top right), Oliver Hazard Perry (top left), and Jiangkai-II (bottom) frigates. (Images sourced from [217, 218, 216].)*

The three destroyers, pictured in Figure 3-3, were the Akizuki-class of Japan's Maritime Self-Defence Force, Sejong Daewang-class of the Republic of Korea Navy, and the Zumwalt-class of the United States Navy. As per requirement 2.5, these vessels represent the large variation in size that can be found within the destroyer designation—at 6,800 tonnes, the Akizuki-class is considered a conventional destroyer [219], while, at 10,455 tonnes, the Sejong Daewang-class is an example of a large destroyer [220]. At over 15,000 tonnes, the Zumwalt-class far exceeds typical expectations of a destroyer [221]. As per specification 2.4, each of these ship classes are in commission at the time of writing.

Figure 3-3: *The Akizuki (top left), Sejong Daewang (top right), and Zumwalt (bottom) destroyer. (Images sourced from [219, 220, 221].)*

Finally, to fulfil the requirement that the dataset represents civilian vessels, and to enable the development of capabilities to autonomously recognize non-combatant vessels, a modern roll-on-roll-off (RORO) vehicle ferry—comparable in size to a large warship—was selected for the role. Crucially, this ship, the MV *Armorique*, pictured in Figure 3-4, operates from the Plymouth ferry terminal, where it can be viewed clearly from public land, thus making it feasible that real-world infrared imagery of this vessel could be collected for the purpose of algorithm testing.

Figure 3-4: *The MV Armorique RORO vehicle ferry. (Image sourced from [222].)*

Table 3-2: *A summary of the vessels selected as CAD models for the synthetic infrared dataset, where length = overall length; displacement = full load displacement, and country of origin = the country in which the class was initially commissioned; and ship types abbreviations are as follows:*

G denotes a vessel with a force guided missile system, including SAM, USM, and SUM, usually with a range exceeding 20 miles;

H denotes a vessel equipped with a helicopter, or a platform for operating one;

M denotes a combatant vessel with a close-range guided missile system.

Class	Type	Length (m)	Displacement (tonnes)	Country of origin	First commissioned	No. ships in service
Ada [206]	Corvette GHM	99.0	2,400	Turkey	2011	4
Independence [207]	Corvette GHM	128.5	3,188	United States	2010	11 (Plus 1 in reserve, 7 to be built)
Visby [57]	Corvette GH	72.7	640	Sweden	2012	5
Alvaro de Bazán [217]	Frigate GHM	146.4	6,250	Spain	2002	5
Oliver Hazard Perry [218]	Frigate GHM	138.1	4,166	United States	1977	34
Jiangkai II [216]	Frigate GHM	134.0	4,050	China	2008	30 (Plus 6 to be build)
Akizuki [219]	Destroyer GHM	151.0	6,800	Japan	2012	4
Sejong Daewang [220]	Destroyer GHM	165.9	10,455	South Korea	2008	3 (Plus 3 to be built)
Zumwalt [221]	Destroyer GH	186.0	> 15,995	United States	2016	2 (Plus 1 to be built)
Armorique [222]	Passenger and RORO cargo ferry	167.0	29,500	France	2009	1

3.2.1.2. Model sourcing, adaption, and creation

3D CAD models already existed of nine of the selected vessels, either as part of the CounterSim software package, or as openly accessible contributions to online CAD libraries. Therefore, where the accuracy of available models was deemed appropriate, these sources provided the basis for the majority of CAD models in the dataset. In cases where the existing CAD models were neither available, nor of suitable quality, 3D models were constructed from scratch using the Rhinoceros 3D modelling software and insight gained from photographs.

3D CAD models of the Oliver Hazard Perry- and Alvaro de Bazan-class frigates were available within the CounterSim software package, and these models, which can be seen in Figure 3-5, were considered suitably accurate for use in the dataset generation process.

Models of the Ada-, Independence-, Visby-, Jiangkai II-, Akizuki-, and Sejong Daewang-class ships were sourced from two other online CAD libraries, CadNav and GrabCAD [223, 224]. However, these models were available in formats that are not compatible with CounterSim, which only accepts Virtual Reality Modelling Language (VRML) and Open Inventor Scene Graph (IV) formats. Therefore, each of these models were converted into VRML format using IVCon [225].

Once converted, each model was inspected and then edited using the Rhinoceros 3D CAD application [226] to ensure that the required level of consistency and fidelity. For example, the conversion process could give rise to inconsistencies such as gaps in the mesh, and surfaces that were overlapping, misaligned, or reversed.

No available models of a Zumwalt-class destroyer were considered suitably accurate for use in the data generation process and, therefore, a 3D model was constructed from scratch, based on photographs of the vessel. Side- and front-view images of a Zumwalt-class destroyer were imported into the Rhinoceros

3D modelling environment and scaled to the appropriate size to enable a faithful representation of the vessel. The finished model is pictured in Figure 3-5.

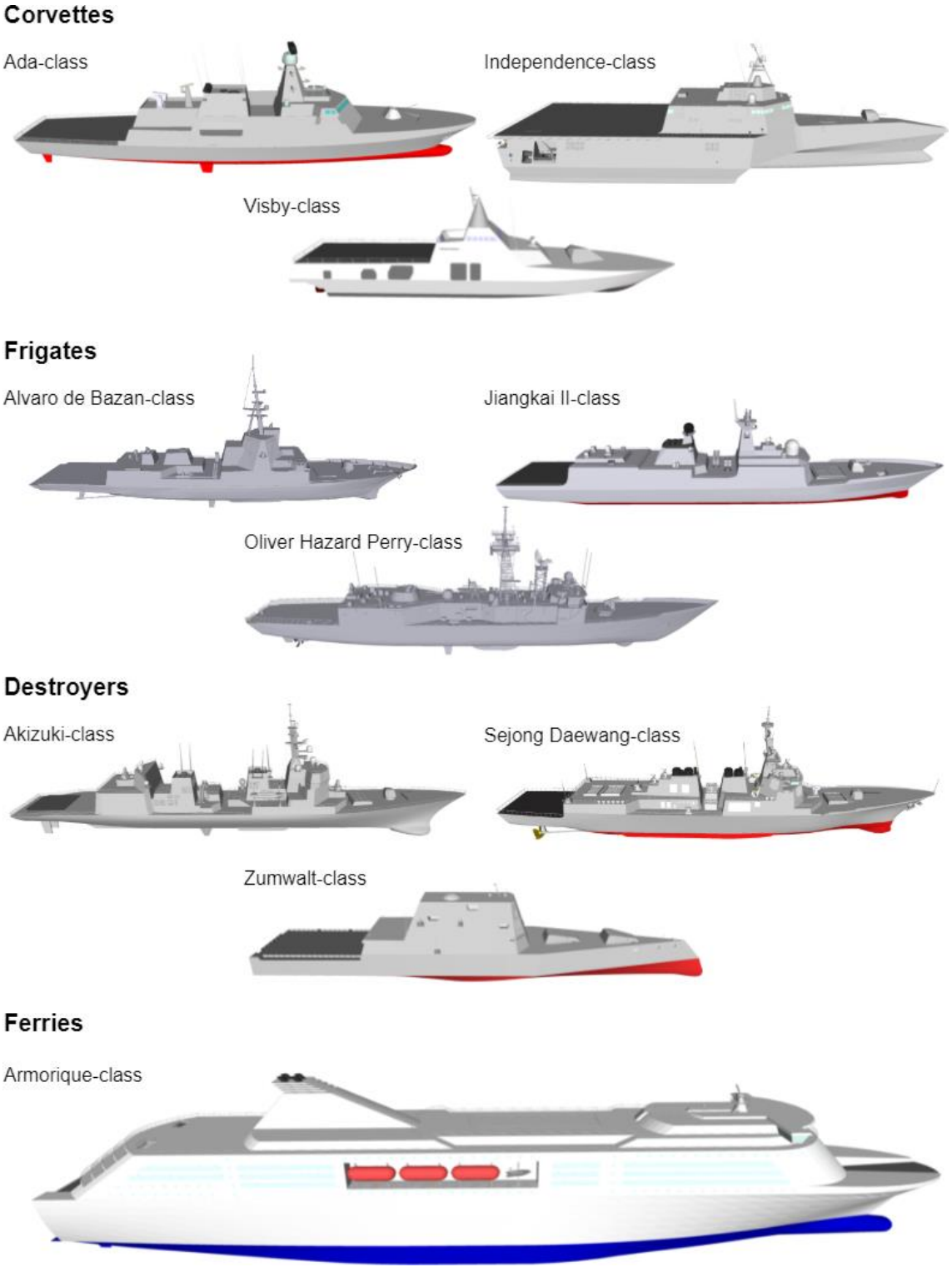


Figure 3-5: *Each of the 10 CAD models which served as the basis of the synthetic infrared dataset. (Not to scale.)*

In accordance with technical specification 2.6, it was ensured that all key physical objects, including main weapon systems, and radomes were represented in the CAD models, along with all major elements of the ships' superstructure.

Once the structure of each model was completed, surfaces were then grouped by their corresponding object and annotated with descriptive human-readable labels. To ensure that surface properties for any given object could be manipulated independently, each was given unique names, such as "Main-Radome', 'Rear-Goalkeeper-Radome', 'Right-Goalkeeper-Radome'. All glass surfaces were labelled with the word 'Window'. As shown in Table 3-3, and this resulted in an average of 93 distinct named object groups per vessel.

Finally, each model was orientated parallel to the X-axis, with height in the Z-direction, and with the origin situated in the centre of the vessel at the waterline.

Table 3-3: *The number of named object groups defined for the 10 CAD models.*

Model	Number of named object groups
Ada	62
Independence	124
Visby	49
Alvaro de Bazan	87
Oliver Hazard Perry	45
Jiangkai-II	118
Akizuki	134
Sejong Daewang	177
Zumwalt	64
Armorique	63
Average:	**92.3**

3.2.2. Design and application of thermal properties

At the CAD stage, the surfaces of the models still lacked the necessary thermal properties to be rendered in the infrared domain. In order to do this, properties such as emissivity and temperature were applied to each surface of the models.

3.2.2.1. Open Inventor extensions for infrared properties

To enable the application of thermal properties to model surfaces, Chemring Countermeasures Ltd has proposed a series of extension nodes to the Open Inventor file format [227], one of which—the `SoIRMaterial` node—is suitable for the task in question. The additional material properties available through the node are described in Table 3-4 below.

Table 3-4: *Summary of the additional material properties which are prescribed by the SoIRMaterial node extension to the Open Inventor file format.*

Field Name	Default Value
`spectralType`	`TEMPERATURE`
`temperature`	`20.0`
`emissivity`	`0.97`
`spectrumFileName`	`""`
`spectrumScale`	`100.0`
`IRCaching`	`ON`

The `spectralType` field specifies the method of infrared processing which will be applied to the material and can be one of three possible values: `TEMPERATURE`, `SPECTRUM`, and `NONE`. If set to `TEMPERATURE`, the material is considered as a grey body, with radiance calculated from values specified by the `temperature` and `emissivity` fields.

Alternatively, if set to `SPECTRUM`, data from a spectrum file, specified by the `spectrumFileName` field, is integrated over the waveband in which the virtual camera is sensitive, and then multiplied by the value of the `spectrumScale` field. As such, the `spectrumFieldName` and `spectrumScale` fields are only used when `spectralType` is set to `SPECTRUM`. Otherwise, if set to `NONE`, no infrared processing is applied, and the values specified by the `emissiveColor` field of the `SoMaterial` node will be used to represent the radiance of the material.

Since the calculation of radiance values is considered to be computationally expensive, setting the `IRCaching` field to `ON` enables values calculated from previous infrared property fields to be cached into memory.

Overall, the properties which must be added to the surfaces of each vessel model to render it in the infrared domain are `spectralType`, `temperature`, and `emissivity`. Figure 3-6 illustrates how these are used to define infrared properties to a surface within an Open Inventor file.

independence.iv

```
#Inventor V2.1 ascii

DEF Bow-Vent Separator {
  SoIRMaterial {
    spectralType TEMPERATURE
    temperature 5.29
    emissivity 0.97
  }
  Coordinate3 {
    point [
    23.6398868560791 4.473113536834717 8.12590217590332, # coord point    0
    23.55459213256836 4.416445255279541 8.245899200439453,
    23.59564971923828 4.43012809753418 8.181756973266602,
```

Figure 3-6: *Example of an SoIRMaterial node specified for the 'Bow-Vent' surface of the Independence-class CAD model.*

3.2.2.2. A software application to apply thermal properties

As each of the 10 CAD models contained, on average, 93 named objects, each with multiple surfaces, then each CAD model contained hundreds, if not thousands, of individual surfaces. The manual application of thermal properties in the form of SoIRMaterial nodes to the Open Inventor files was deemed to be prohibitively time-consuming, and an alternative method was required.

Accordingly, a software application, named IVTools, was developed using in Python3 in order to convert VRML CAD models into Open Inventor format and apply new nodes to the named surfaces programmatically. This software has been published on GitHub at [228]. The detail usage of IVTools is outlined in Appendix A-1.

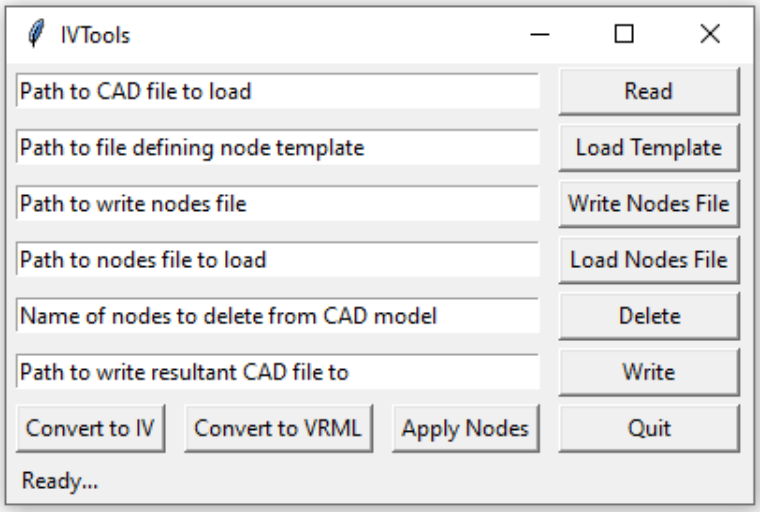


Figure 3-7: *The Graphical User Interface of IVTools.*

3.2.2.3. A stochastic rules-based approach to thermal signature generation

Technical requirements 2.7 and 2.8 call for the dataset to depict multiple realistic thermal signatures for each class of ship. But with a total of 932 named surfaces across the 10 CAD models, the careful hand-selection of appropriate thermal properties for every such surface would be prohibitively time-consuming.

A programmatic approach was therefore devised in order to generate realistic ship thermal signatures by randomly applying, following a stochastic rules-based approach, surface-by-surface temperature variations to a generic model where all surfaces had a starting temperature of 20 °C.

In all, nine unique sets of thermal properties were generated, and applied to each CAD model in a process illustrated in Figure 3-8. Pseudocode, to describe the internal logic of the process, is contained within Appendix A-2. Finally, this process was repeated for each of the 10 CAD models, resulting in a total of 90 unique CAD files, each with a distinct and realistic thermal signature. For illustrative purposes, Figure 3-9 shows the nine unique thermal signatures which were applied to the Akizuki-class model.

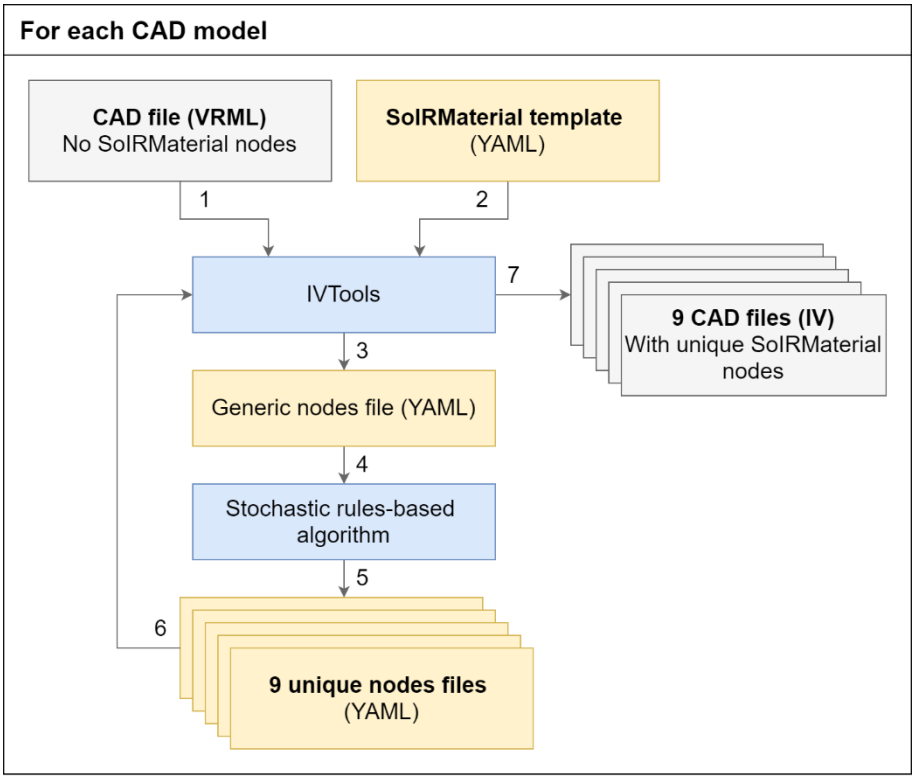


Figure 3-8: *An overview of the process through which nine unique thermal signatures were generated for a given CAD model and subsequently applied to create nine new models with varied surface thermal properties.*

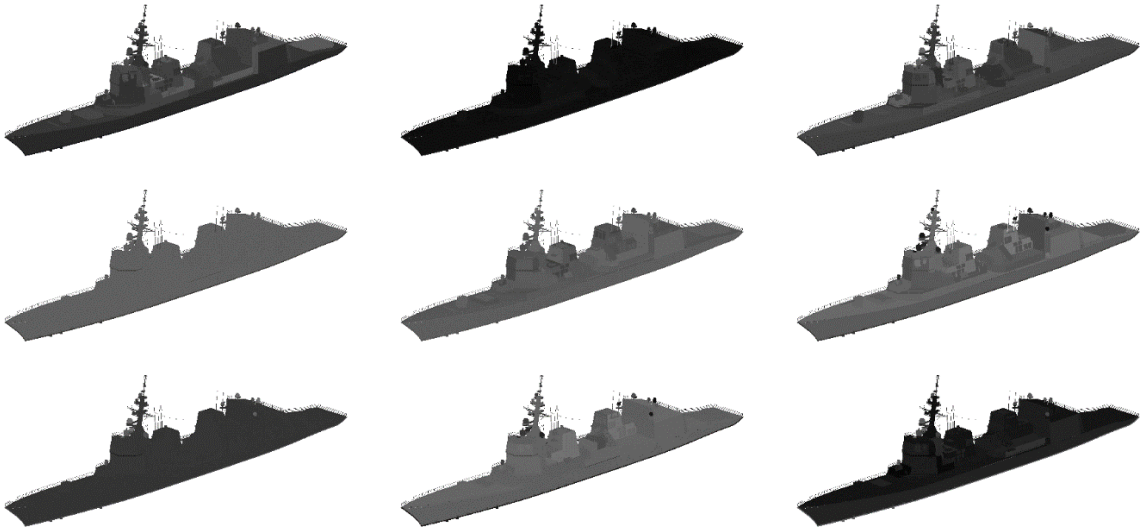


Figure 3-9: *The nine CAD models of the Akizuki destroyer, each with a unique thermal signature and rendered in the infrared domain using CounterSim.*

3.2.3. Synthetic infrared image generation

With thermal properties applied to the surfaces of each CAD model, it was possible to generate synthetic infrared imagery of the 10 ships. The images were generated by using CounterSim, an advanced simulation environment developed by Chemring Countermeasures Ltd for modelling the engagement of radio frequency- and infrared-guided missiles with ships, vehicles, and aircraft. By calculating the radiance of surfaces across the relevant waveband, and modelling the subsequent atmospheric transmission, CounterSim is capable of generating realistically rendered virtual scenes in the infrared domain.

3.2.3.1. The CounterSim scenario

The CounterSim scenario which was used to generate infrared imagery of the 10 CAD models, shown in Figure 3-10, was constructed as a hierarchy containing five virtual components: an Engagement item, Scene Generator item, a Ship item, CM System item, and a Thermal Imager item.

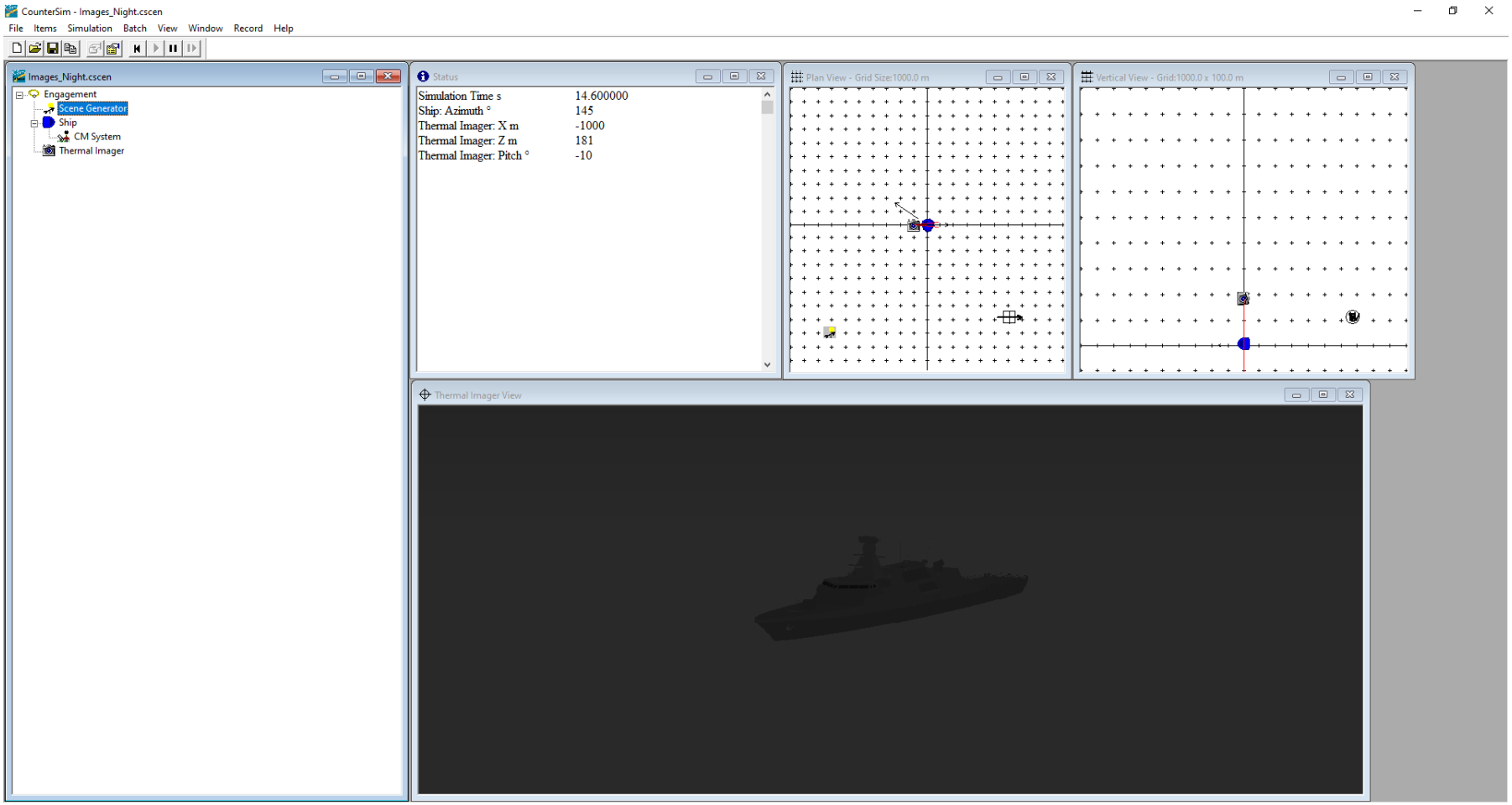


Figure 3-10: *The CounterSim scenario which was used to generate infrared imagery of each military ship.*

Positioned at the root of the hierarchy, the **Engagement** item, does not represent a physical object, but is instead responsible for controlling the execution of the simulation and logging engagement parameters. Using this item, the ‘maximum run time’ of the simulation was set to 36.3 seconds, in order to give the Thermal Imager item enough time to capture a single frame at each ship bearing in 0 – 359° at a frame rate of 10 Hz.

Positioned within the engagement item, the **Scene Generator** item is responsible for generating the infrared scene, which is subsequently imaged by the Thermal Imager item. This scene was defined as ‘Slight Daylight’ with a ‘Smooth Sea’ at a latitude of 45° and, as per technical requirement 1.1, MODTRAN4—a version of the US Air Force atmospheric transmission, radiance and flux model [171]—was used to simulate infrared atmospheric transmission. Then, to satisfy technical requirements 3.1 and 3.2, the Scene Generator was used to vary the sea surface temperature and ambient temperature between simulations by drawing values for these parameters from $U(5, 25)$ and $U(5, 30)$ respectively.

Also placed in the Engagement, the **Ship item** was used to manage each of the CAD models, which were selected by the user-definable parameter, ‘IR Graphics File’. This Ship item was located at the origin of the scene (0, 0, 0) and, as per technical requirement 2.10, was rotated by 360° in the azimuthal plane during the simulation while being imaged by the Thermal Imager item. This rotation manoeuvre was dictated by a Comma-separated Values (CSV) ‘Track File’ which incrementally increased the Ship item’s bearing by 1° every 0.1 seconds.

A **CM System** item was assigned to the Ship item, since this was required to initiate the Ship’s rotation, as defined by the Track File.

Finally, a **Thermal Imager** item, sensitive in the 8–14μm waveband and the 0–100 °C temperature range, was inserted into the engagement, as per technical requirement 1.1. In order to account for the wide aspect ratio of ships, the imager had a resolution of 1024 × 512—which satisfied technical requirement 1.2—and a horizontal and vertical field of view of 12° and 6° respectively.

Assuming a pixel size of 2λ (28μm), this corresponds to a detector array size of 2.9 × 1.4 cm and, using Equation 1.5, the requisite focal length is *circa* 7 cm. Therefore, each of the optical parameters is compatible with the size limitations of an anti-ship missile, as detailed in technical requirement 1.3.

In order to correspond with the Ship's rotation manoeuvre, the Thermal Imager captured images at a rate of 10 Hz, and the result of each simulation was stored as a single Audio Video Interleave (AVI) file. Over multiple simulations, the thermal imager was positioned at various distances from the Ship item and at various angles of pitch.

To satisfy technical requirement 2.9, 10 different ship-imager distances between 1km and 10km were selected for use during the simulations. These distances increased non-linearly according to the set $D = \{1.0, 1.1, 1.3, 1.4\ 1.7, 2.0, 2.5, 3.3, 5.0, 10\}$, as defined by Equation 3.1, to ensure that the apparent size of each target reduced linearly between distance increments.

Other synthetic datasets [118, 75, 115] contained in the literature have commonly incremented imager-target distances linearly, causing the change in the apparent size of the target to diminish with each increment. For example, increments of $\{1, 2, 3, \dots, 10\}$ kilometres could cause the apparent scale of the target to halve as the imager is moved from 1 km to 2 km away, whereas in images taken from 9 km and 10 km the size of the target would be only marginally different, as illustrated in Figure 3-11a. This is undesirable, as it causes a wide range of different target scales to be omitted between the first few increments. Instead, by incrementing ship to imager distances non-linearly, a consistent reduction in the apparent size of the target can be achieved, as illustrated in Figure 3-11b.

$$D = \frac{10\ km}{i}\ for\ i\ in\ \{10, 9 \dots, 1\} \tag{3.1}$$

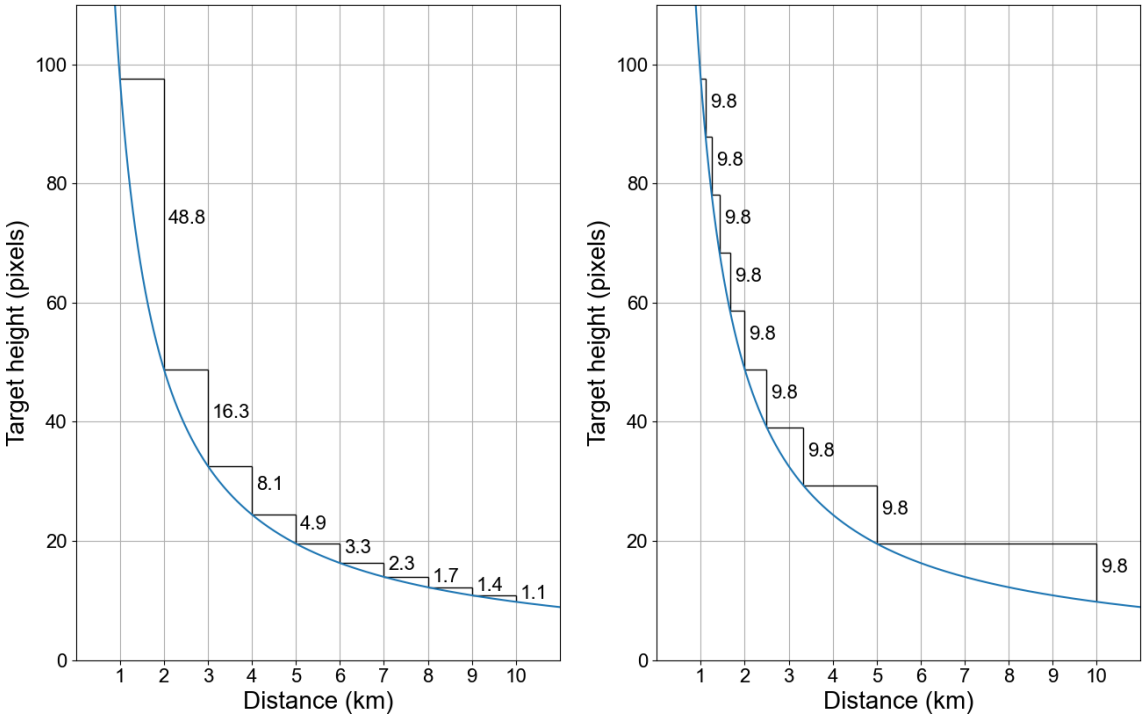


Figure 3-11: *Both graphs show the effect of distance on the pixel height of a 20-metre-tall target, assuming an imager with a field of view of 6° and a vertical resolution of 512. Annotations on graph a) illustrate how changes in the pixel-height of the target diminishes if range increases incrementally by 1 km. Annotations on graph b) show that the imager-to-ship distances selected above provide a consistent change in the pixel-height of a target.*

To satisfy technical requirement 2.11, three different pitch angles—the angle between the missile's thermal imager and the horizontal plane—were selected as $P = \{0, -10, -20\}°$, where negative values correspond to scenarios where the seeker is directed down from above the target. In order to ensure that the target remained in frame for each increment of pitch, the Thermal Imager's position in the Z-direction was calculated as $d \times \tan(-p) + 5$ where d is the imager-to-ship distance, and p is the imager angle of pitch.

3.2.3.2. Executing the simulations

Using CounterSim's 'Batch Simulations' feature, multiple simulations were executed, each depicting a single vessel rotating through 360° in the azimuthal plane. With a total of 10 CAD models, 9 different thermal signatures, 10 different imager-to-ship ranges, and 3 different angles of pitch, this resulted in

2,700 simulations (10 × 9 × 10 × 3). Each simulation was outputted to video file in AVI format, which was named using the scenario conditions, including the vessel class, the ID number of its thermal signature, the ambient and sea surface temperature, the angle of pitch, and the imager-ship distance. Example frames from these simulations can be seen in Figure 3-12.

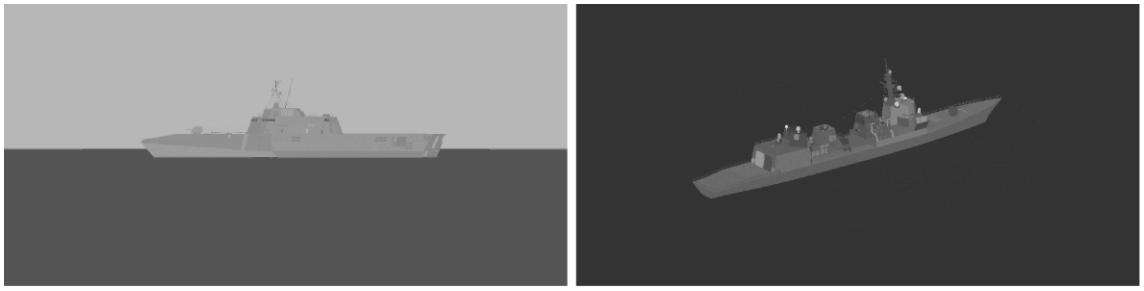


Figure 3-12: *Examples of synthetic LW infrared images which were generated using CounterSim. (Image contrasts have been enhanced for illustrative purposes.)*

To enable the creation of bounding box and pixel-wise labels, as per technical requirement 4.2, further simulations were run, similar to those described above, except that in these the Scene Generator did not simulate any background, and ship thermal signatures did not vary. Instead, ships' surface temperatures were each set to 100 °C, ensuring that the ship contrasted highly against the black background, as show in Figure 3-12. This resulted in a further 300 videos (10 × 1 × 10 × 3) which could be used to create the necessary ground truth localisation annotations.

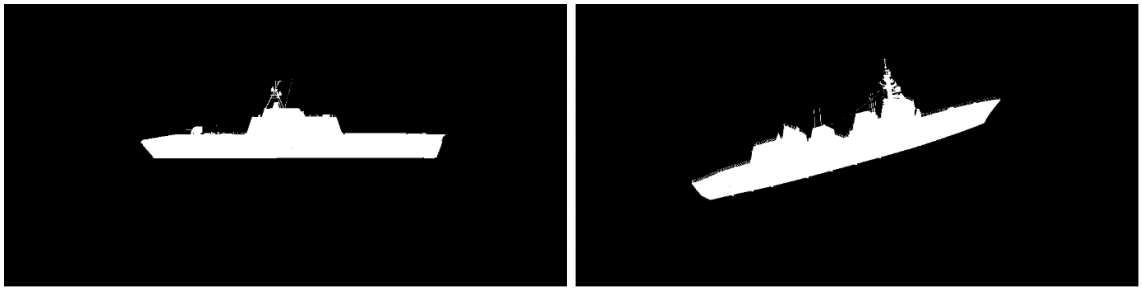


Figure 3-13*: Examples of the binary label images which were also generated using CounterSim.*

3.2.4. Post-processing

With the generation of 2,700 ‘data videos’ and 300 ‘label videos’ complete, a computer program was then written using Python to process this data into a structured image dataset, containing 972,000 different synthetic infrared images—the product of multiplying 2,700 by 360, or 2,700 data videos for each of the 360 degrees from which an image seeker might view a ship. The program ensures compliance with technical requirements 4.1, 4.2, and 4.3 by assigning ship-type and ship-class labels to each image, calculating bounding box labels for each ship, and adding applicable metadata, such as the ambient temperature, imager-to-ship distance, and ship bearing. Encodings for ship type and ship class, shown in Table 3-5, were then appended. Intensity values for the label videos were applied by setting pixels with a value of less than 15 to zero, with all remaining pixels were set to a value of 255, and bounding box extents calculated using an algorithm presented in Appendix A-3. Finally, all data and label images were saved to sperate directories and metadata for each data image was written to a ‘summary’ CSV file. The logic followed by this program is outlined in Figure 3-14.

Table 3-5: *The encodings which were used by this dataset to denote ship type and ship class.*

Type name	Type encoding	Class name	Class encoding
Corvette	0	Ada	0
		Independence	1
		Visby	2
Frigate	1	Akizukia	3
		Jiangkai-II	4
		Oliver Hazard Perry	5
Destroyer	2	Akizuki	6
		Sejong Daewang	7
		Zumwalt	8
Ferry	3	Armorique	9

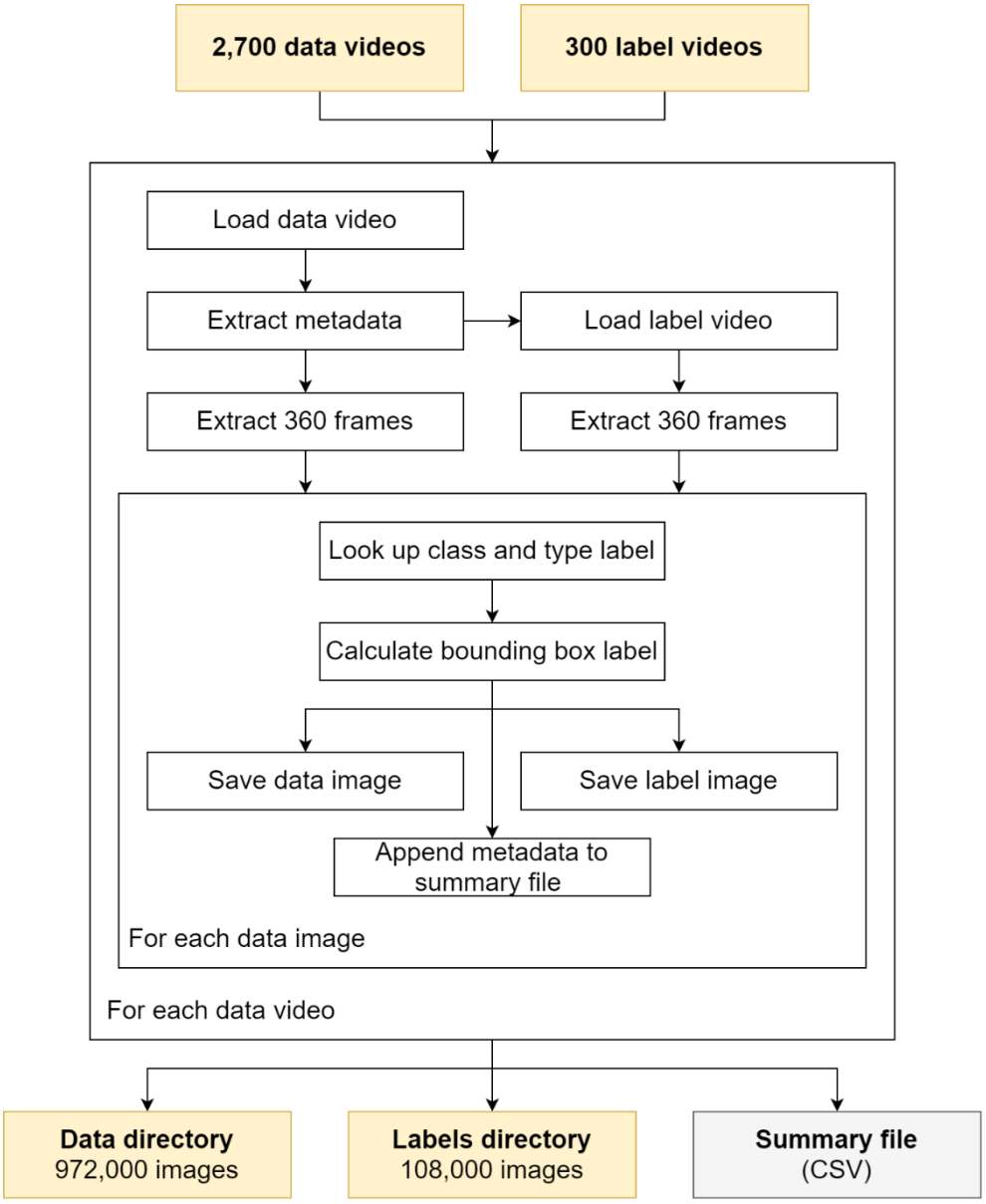


Figure 3-14: *Overview of the data post-processing algorithm. Further details of the three outputs can be seen in Figure 3-15.*

The result of this post-processing was a structured image dataset containing two directories—the first containing the 972,000 synthetic infrared image files and a second containing 108,000 label image files—and a single summary file containing relevant metadata for each example. This structure is illustrated in Figure 3-15, which shows how key annotations are formatted within the summary file.

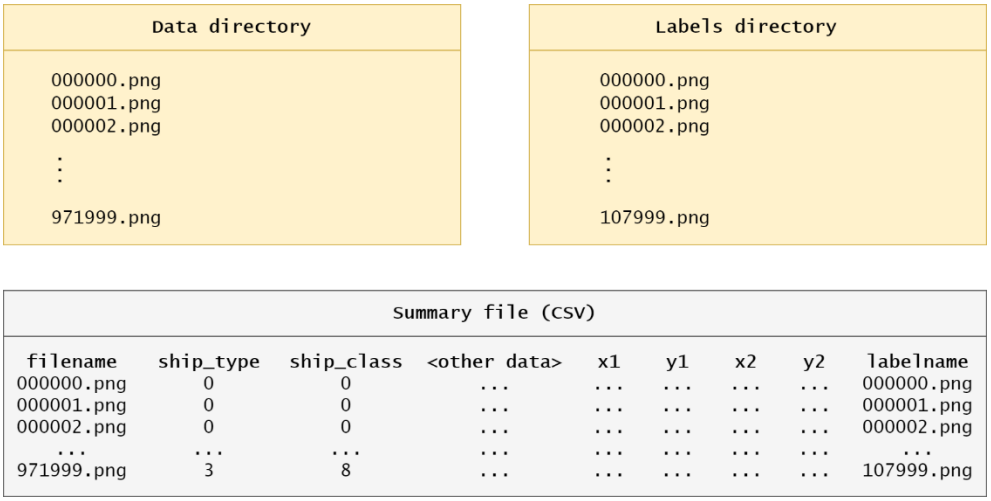


Figure 3-15: *The structure of the resultant image dataset. The data directory contains 972,000 data images and the labels directory contains 108,000 label images. The summary file contains a single row of metadata for each data image, where: filename is the name of the example image, ship_type and ship_class are the type and class designations of the depicted vessel, {x1, y1, x2, y2} are the normalised bounding box coordinates for the vessel, and labelname is the name of the corresponding label image. Additional metadata, such as ambient temperature, surface temperature, and ship bearing is not shown but is represented by <other data>.*

3.2.5. Data augmentation pipeline

Technical requirements 3.3, 3.4, and 3.5 call for an adequate degree of variation in sea-state, sky-state, and background clutter to be reflected in the dataset. Satisfying these requirements was conducted in three stages: 1) sea-state augmentation, 2) sky-state augmentation, and 3) background clutter augmentation, an overview of which is illustrated in Figure 3-16.

As a design principle, it was decided that the data augmentation process should be executed at runtime, when the images are first loaded into memory by the deep learning training algorithm. This provides three key benefits. First, this approach enables any number of different sea, sky, and background states to be depicted within the dataset. Second, the variety and complexity of these states can be independently extended in future work. And third, by augmenting images stochastically, images will differ each time they are loaded, further increasing the diversity of the dataset.

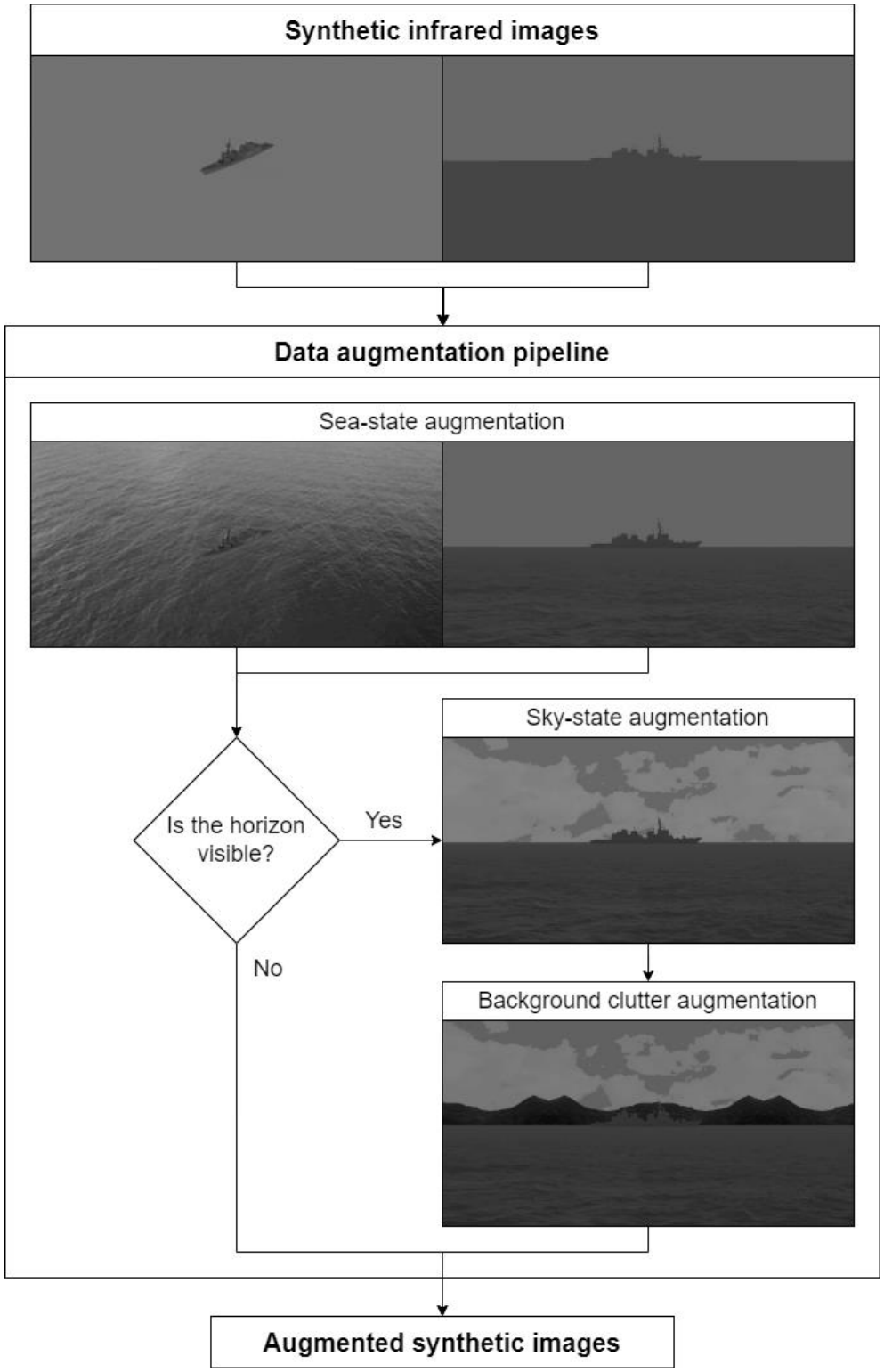


Figure 3-16: *An overview of the data augmentation pipeline, developed to increase the complexity and diversity of the infrared dataset. (Image brightness and contrasts have been enhanced for illustrative purposes.)*

3.2.5.1. Sea-state augmentation

The sea-state augmentation process began with two collections of real-world images of ocean surfaces. The first of these collections contained 15 images of the sea captured from a horizontal perspective, with the sea-sky horizon being visible. The second collection contained 10 images of sea surfaces taken from an elevated perspective. These images were taken from the online image hosting sites Flickr and Wikimedia, each licenced under CC BY 2.0 or 3.0 [229, 230], which permits image sharing and editing, provided that appropriate credit is given.

When an image is first loaded, its angle of pitch is ascertained from the metadata file, in order to determine which set of real-world images should be used. If the pitch angle is zero, an image from the first collection is selected at random, and its pixel values scaled according to Equation 3.2.

$$I_{i,j} = \tilde{I}_{i,j} \times U(a, b) + c \tag{3.2}$$

where $\tilde{I}$ is the normalised real-world image and c is the average pixel intensity of the sea region of the synthetic image. Values of 5 and 30 were selected for the bounds a and b respectively to provide a feasible variety of pixel, and therefore temperature, ranges. 50% of the time, the image is flipped horizontally. Images are then cropped to size, using randomly selected parameters denoting scale and position. The processed image is then superimposed below the sea-sky horizon line of the synthetic image, using the synthetic image's binary mask in order to preserve any pixels relating to the ship. The results of this data augmentation process is shown in Figure 3-17.

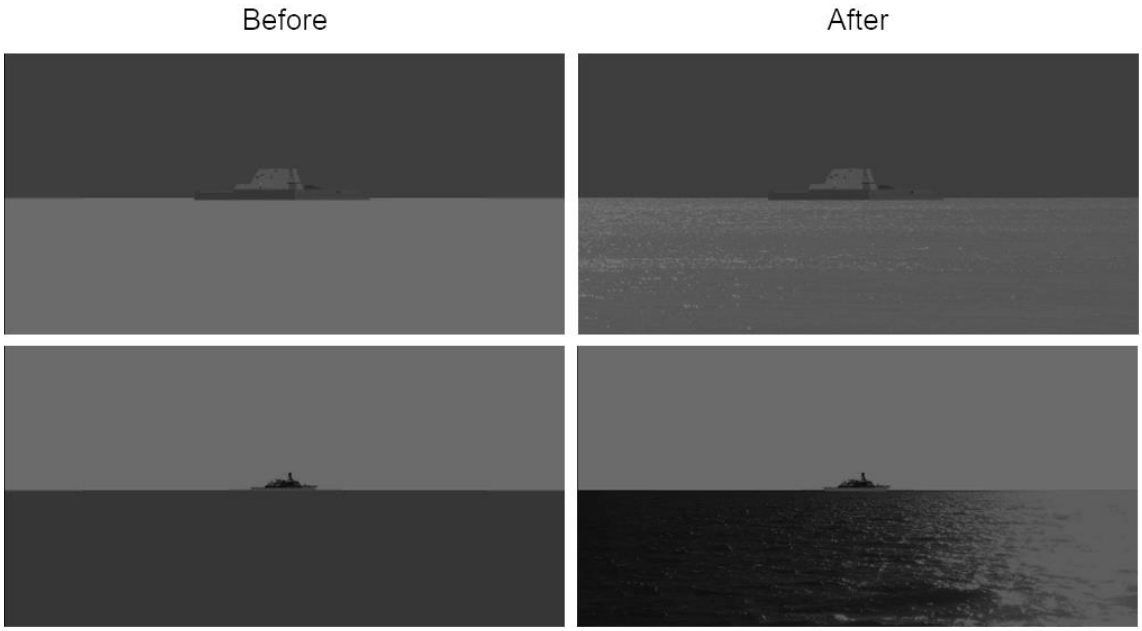


Figure 3-17: *Sea-state augmentation of synthetic infrared images where the sea-sky horizon line is visible. (Image brightness and contrasts have been enhanced for illustrative purposes.)*

For synthetic images with an angle of pitch of -10 or -20 degrees, a real-world seascape was selected at random from the second collection. This seascape was then normalised following Equation 3.2, using the same bounds a and b, and flipped (or not) and cropped as described above. The processed seascape was then superimposed into the entirety of the scene, as can be seen in Figure 3-18. Finally, as (by definition) no sea-sky horizon line is present in these images, further image augmentation for sky state and background clutter are not relevant, and image augmentation can be considered as complete.

Figure 3-18: *Sea-state augmentation of synthetic infrared images where the sea-sky horizon line is not visible. (Image brightness and contrasts have been enhanced for illustrative purposes.)*

3.2.5.2. Sky-state augmentation

The sky-state augmentation process began with 10 suitably licenced real-world images of clouded skies collected from Flickr and Wikimedia. Each of these real-world images was processed, by first setting the value of any pixels which corresponded to blue sky to zero, and then converting the image to grayscale following the ITU-R Recommendation BT.601 image encoding standard [231], described in Equation 3.3, and the effect of this process is visualised in Figure 3-19.

$$V = 0.299R + 0.587G + 0.114B \tag{3.3}$$

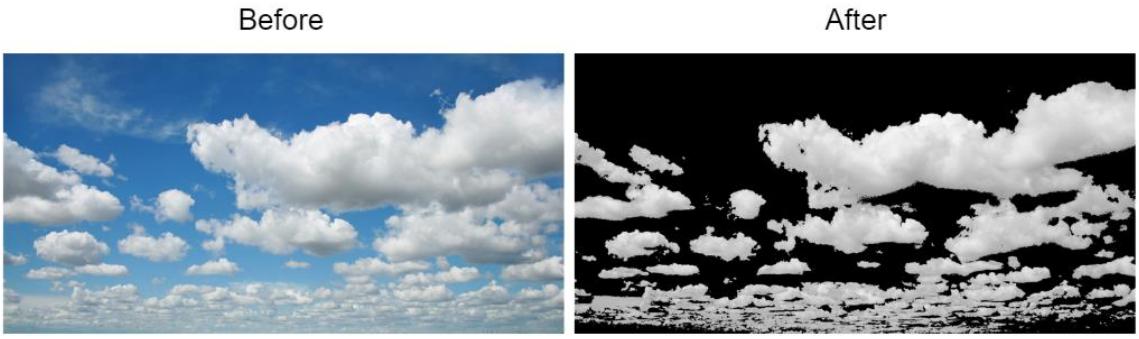


Figure 3-19: *An example of the sky-state processing step.*

The sky-state image augmentation process that follows is broadly analogous to the sea-state augmentation process. When an image is first loaded, a real-world image is randomly selected from the now-encoded collection of clouded skies. This image is then normalised, as per Equation 3.2, with, in this case, c being the average pixel intensity of the sky region in the synthetic image. Values of 2 and 50 were chosen for the bounds a and b respectively, in order to provide a feasible variety of pixel ranges. Finally, the image is flipped (or not) and cropped as for sea-state augmentation, and superimposed into the region above the sea-sky horizon line of the synthetic image, preserving the values of all pixels relating to the ship. Examples of the final outcome of this process are shown in Figure 3-20.

Figure 3-20: *Sky-state augmentation of synthetic infrared images. (Image brightness and contrasts have been enhanced for illustrative purposes.)*

3.2.5.3. Background clutter augmentation

As with sea-state augmentation and sky-state augmentation, a selection of suitably licenced real-world images was first collected from Flickr and Wikimedia. This collection totalled 30 images, containing images of icebergs, natural coastlines, and man-made maritime structures respectively, where man-made structures included built-up coastlines, offshore wind turbines, and oil platforms. These images were subsequently pre-processed by cropping to the area of interest and setting the value of background pixels to zero, as illustrated in Figure 3-21.

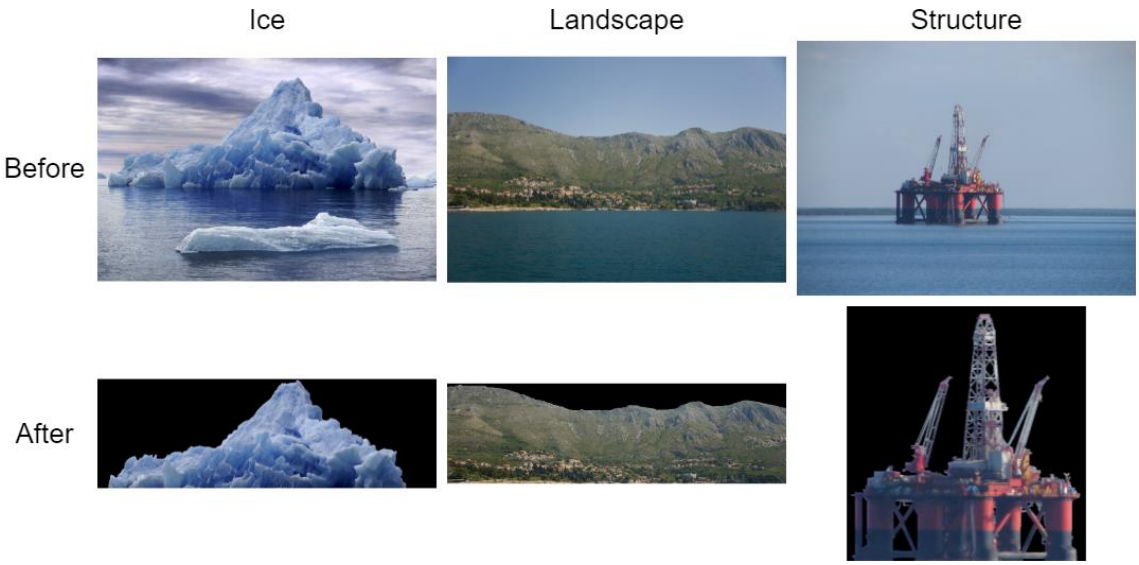


Figure 3-21: *Examples of the background clutter processing step.*

Again, the background clutter image augmentation process that follows is broadly analogous to the sea-state and sky-state augmentation processes. When an image is first loaded, a real-world image of background clutter is randomly selected and normalised as per equation 3.2. In this case, however, c is set to zero, with bounds a and b dependent on the nature of the real-world image. For images containing icebergs, values of 0 and 2 were used. For images containing natural coastlines, values of 15 and 60 were used, while for images containing man-made structures, values of 20 and 80 were used. These pairs of values were selected to roughly correspond with the expected temperature ranges of background clutter. Finally, the image is again flipped (or not) and cropped as for sea-state and sky-state augmentation and superimposed above the sea-sky horizon line of the synthetic image, preserving the values of all pixels relating to the ship. Examples of the final outcome of this process are shown in Figure 3-22.

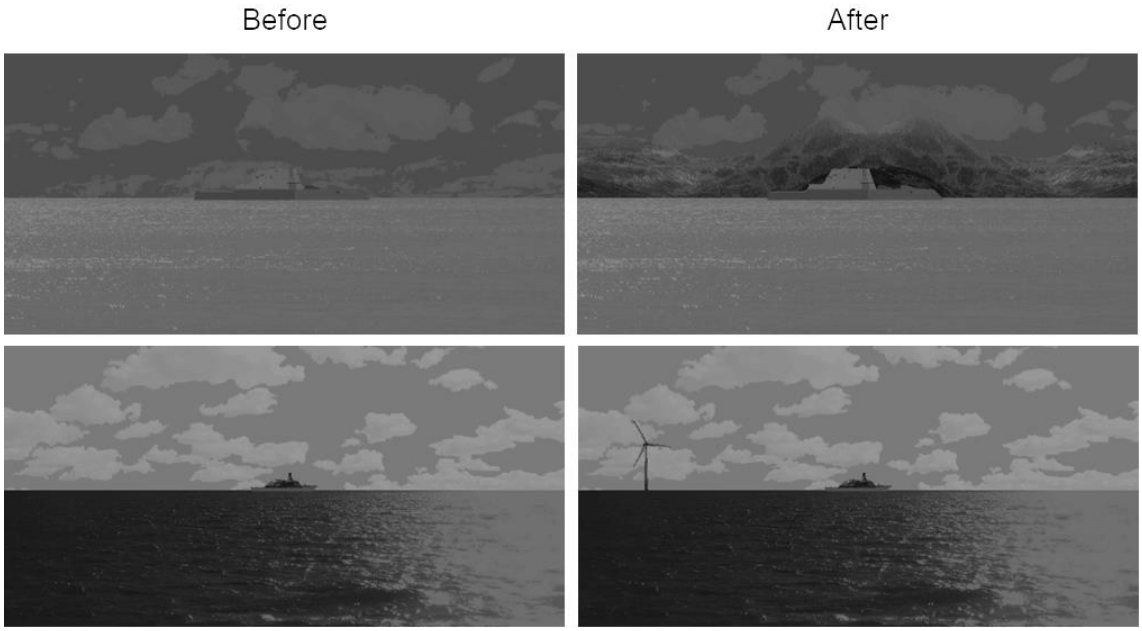


Figure 3-22: *Background clutter augmentation of synthetic infrared images. Images on the left-hand side show the superimposition of a background landscape, while images on the right depict the insertion of an offshore wind turbine. (Image brightness and contrasts have been enhanced for illustrative purposes.)*

3.2.5.4. Data-loader

For actual use within the training routine of a deep learning object detection algorithm, the images resulting from these three image augmentation processes require a data loader program. This program is tasked with controlling and sequentially applying each of the data augmentation processes, enabling the bounds a and b for each process to be amended as required, iterating through the required number of images, shuffling the order in which images are presented to the training algorithm, and bypassing processes as necessary—eliminating the background clutter process if the assumption has been made that a given simulated engagement will not take place near a coastline. In addition, it can also specify the range of possible height values for background clutter, as well as the frequency at which the background clutter augmentation processes is applied.

Such a data-loader was written using Python. An example of how this data-loader can be used to initialise, shuffle, and iterate through the images that

make up the synthetic infrared dataset is shown in Figure 3-23, while further examples of augmented data, showing images where all three image augmentation processes have been sequentially applied, are provided in Figure 3-24.

load_data.py

```python
from irships.utils import Dataloader       # Import the dataloader

dataset = Dataloader(root="IRShips")       # Initialise dataloader
dataset.shuffle()                          # Shuffle the dataset

for inp, label in dataset:                 # Iterate the dataset

    pass  # Do something with input and label
```

Figure 3-23: *A snippet of Python code to illustrate how the data loader can be initialised and used to shuffle and iterate through this new synthetic dataset.*

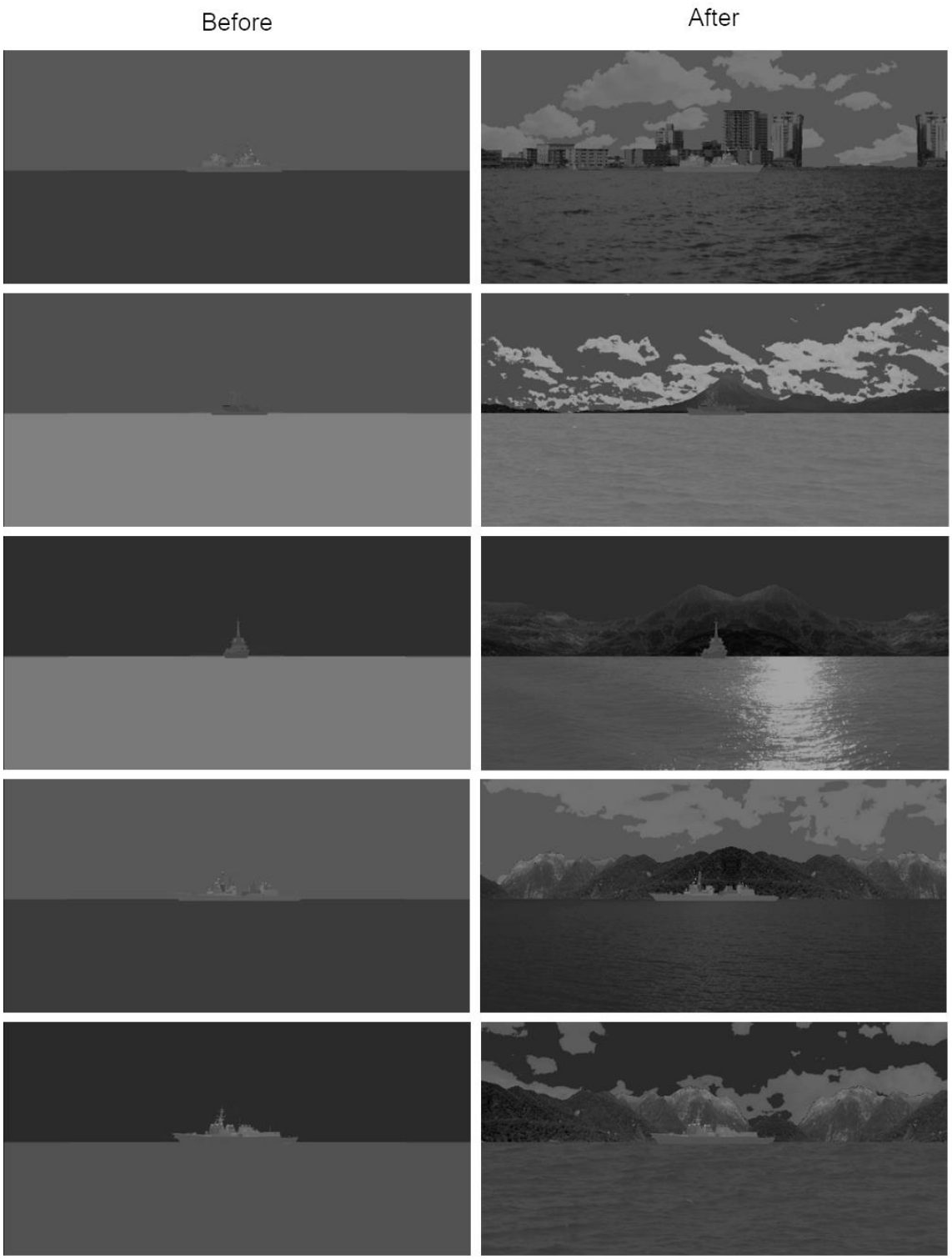


Figure 3-24: *Examples of synthetic images before and after application of the sea, sky, and background clutter augmentation processes. (Image brightness and contrasts have been enhanced for illustrative purposes.)*

3.3. Conclusion

The synthetic dataset resulting from this process of generating and augmenting infrared images of ships, contains 972,000 fully annotated examples for training of CNN-based anti-ship ATR algorithms.

Depicting 10 different classes of vessel, each with nine unique thermal signatures, this dataset—now named IRShips—is a considerable advance on previous infrared ship datasets. Most notably, as shown in Table 3-6, IRShips is two orders of magnitude larger than the next largest real-world dataset while also containing nearly twice as many different classes of military vessel. Made openly available on the Cranfield Online Research Data (CORD) repository [204] it has, at the time of writing, been downloaded over 8,000 times across 20 different countries, as shown in Figure 3-25.

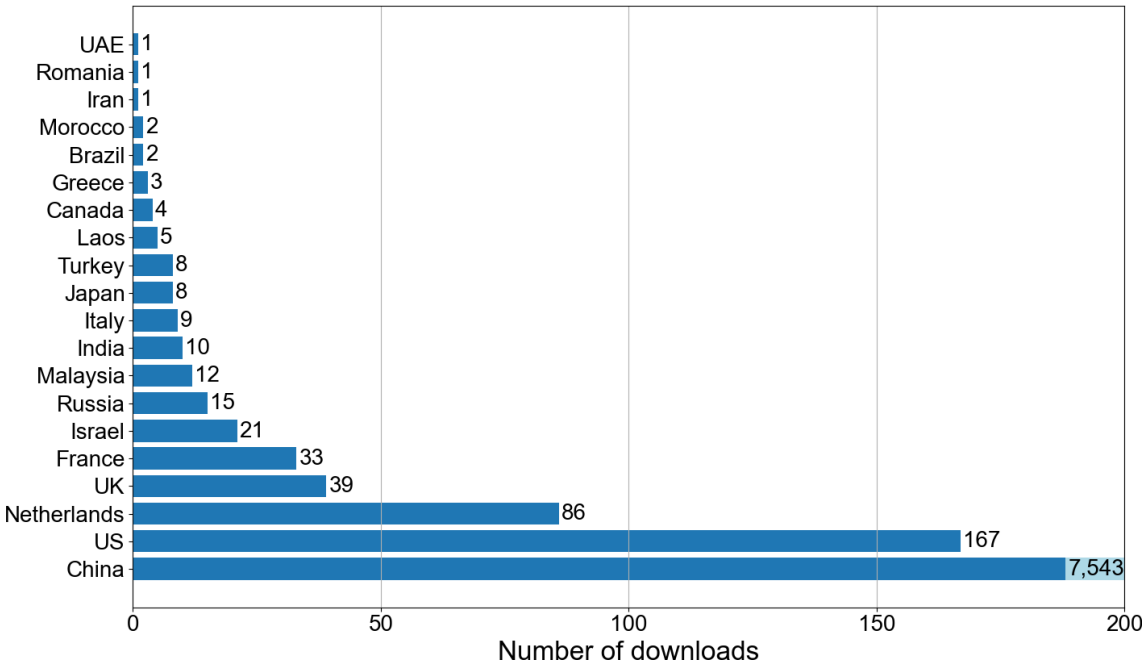


Figure 3-25: *The number of downloads of IRShips to date, sorted by country.*

Furthermore, as shown in

Table *3-7*, IRShips also provides more examples than any previous synthetic datasets—over 23 times more than the next largest—while simultaneously offering roughly seven times greater image resolution. In addition, IRShips contains more classes of currently serving military vessels than any of the other synthetic dataset, which in contrast to other datasets, are represented by CAD models all of a comparably high fidelity.

Ultimately, IRShips is the first synthetic infrared dataset of military ships to provide a variety of thermal signatures for each of its constituent targets. It is also the first to include significant variations in sea-surface and background temperatures, the first to include scenes with complex background clutter, and crucially, the first to have been made openly accessible. Consequently, as the largest, most diverse, and the only openly available infrared dataset of military vessels, IRShips is a significant contribution towards the development of future deep learning-based infrared anti-ship ATR algorithms.

Table 3-6: *Comparison of IRShips with all real-world infrared datasets of military maritime vessels, where No. military types and No. military classes refer only to ships that are in active service at the time of writing.*

Dataset	No. images	No. military types	No. military classes	Localisation annotations	Image resolution	Open access
IRShips	972,000	3	9	✓	1024 × 512	✓
Tremblay and Valin, 2002 [12]	2,545	4	5	×	Not stated	×
Li, 2008 [232]	270	2	2	×	Not stated	×
Wang, Bai and Zhang, 2014 [151]	200	Not stated	Not stated	✓	Up to 720 × 576	×
Withagen, 1999 [123]	136	1	4	×	720 × 576	×
Herman, 2000 [117]	18	3	4	×	Not stated	×

Table 3-7: *Comparison of IRShips with all other synthetically generated infrared anti-ship ATR-focused datasets, where No. military types and No. military classes refer only to ships that are in active service at the time of writing.*

Dataset	No. images	No. military types	No. military classes	High-fidelity CAD models	Contains a variety of different:			Image resolution	Open access
					Sea states	Background clutter	Ship thermal signatures		
IRShips [204]	972,000	3	9	✓	×	×	×	1024 × 512	✓
Kechagias-Stamatis, Aouf and Nam, 2017 [7]	Not stated	2	3	✓	×	×	×	Not stated	×
Gray et al., 2013 [75]	Not stated	2 (3)	6 (8)2	✓ (5 of 9)	×	×	×	320 × 240	×
Gray et al., 2012 [118]	11,520	2	4	✓ (3 of 4)	×	×	×	320 × 240	×
Alves, 2001 [121]	41,400	3	3	×	×	×	×	Not stated	×
Zhao, Wang and Zhang, 2005 [152]	3,600	2	3	×	×	×	×	Not stated	×

[2] One of the eight ships in this dataset—the Niels Juel-class corvette—is no longer in service, with the last ship of this class being decommissioned in 2009 [274]. Another, the Adelaide-class frigate, is approximately the same as the Oliver Hazard Perry-class [275], which is also features in the dataset. Therefore, this dataset effectively contains six different classes of military ship.

Chapter 4

4. Deeplodocus: a modular deep learning development environment

This chapter describes Deeplodocus, software developed in collaboration with a former Cranfield University PhD student, Alix Leroy, enrolled at the university between 05/02/2018 and 05/09/2019. Alix's research also required a modular deep learning environment, albeit for a different purpose—monocular distance, surface normal, and optical flow estimation—and in the absence of similarly functional commercial software in 2018, Deeplodocus was designed and implemented. Deeplodocus now features among the top 50% of packages on the Python Package Index (PyPI) repository [233]. It should be noted that both collaborators each contributed approximately equal shares towards the completion of this project.

In order to progress to develop, train and test convolutional neural network algorithms using the IRShips dataset, it is necessary to test and analyse a large number of architectures and hyper-parameters before the 'right' model can be selected. Environments to accelerate this process are vitally important: critically, as illustrated in Figure 4-1, only a small proportion of any machine learning system is made up of the machine learning algorithm itself, with the remainder comprising an extensive and complex supporting infrastructure [234].

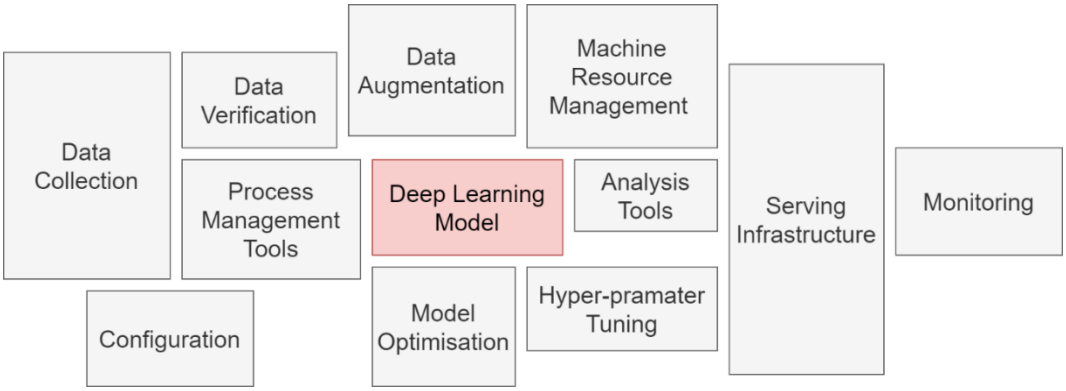


Figure 4-1: *Only a small fraction of a deep learning systems consists of the deep learning model itself. To train, validated, test, and deploy a deep learning algorithm necessitates vast and complex supporting infrastructure. (Image adapted from [234].)*

Developing a modern deep learning algorithm is therefore a complex undertaking, demanding a substantial investment of time, skill, and effort. One complexity is the challenge of implementing the algorithm itself: a complex network of thousands—if not millions—of interconnected artificial neurons. An additional complexity is the need for numerous auxiliary and low-level software components, termed modules. Implementation of the modules initiates software sequences—called inference pipelines—in which the deep learning model performs inference. To enable this, modules must be precisely interconnected to achieve the desired outputs from the algorithm's specific training, validation, and testing pipelines.

Training deep learning algorithms calls for effective gradient-based optimisation algorithms, specialist objective (or loss) functions, and the efficient loading and transformation of hundreds of thousands of instances of annotated training data. Yet building these inference pipelines is an intensive task, consuming valuable time and resources that could otherwise be dedicated to further develop the algorithm to improve its performance. Clearly, the use of such a modular environment can result in models being trained more quickly, models that are more robust, make better use of costly GPU resources, and more readily provide reproducible conclusions.

Deeplodocus is a modular deep learning environment specifically created to facilitate the rapid development, prototyping, evaluation, and selection of ship detection algorithms within this thesis. Importantly, Deeplodocus is a powerful and generalisable tool, and is not intended to replace or extend any existing machine learning libraries, but to instead provide a flexible, modular, and structured environment that augments existing required software. Hitherto, deep learning software libraries, such as TensorFlow, PyTorch, and Keras [235, 236, 237], have reduced the time required to implement deep learning solutions by reducing the amount of code that needs to be written. Yet these libraries have not eliminated the need to manually assemble multiple low-level modules within effective inference pipelines, nor the need for an environment such as Deeplodocus. Clearly, therefore, a modular deep learning development environment which can automatically assemble any number of distinct, low-level modules into effective inference pipelines delivers a significant advance in both researcher productivity and model quality.

As a single researcher, it is all too easy for the time scheduled for developing and improving deep learning algorithms to be spent on managing infrastructure and jobs instead—Deeplodocus was built to avoid this. Specifically, in 2018, in the absence of commercial solutions, such as Kubeflow and DeterminedAI [238, 239], standard tasks in this body of work would have taken months instead of days. For example, if Deeplodocus had not been developed in 2018, this body of work would not have:

- been able to invest so much time in building models, rather than infrastructure,
- had the time to experiment with many different algorithm training conditions, or
- had efficiently made use of the required custom-built modules for the experiments.

Developed in 2018, Deeplodocus is open-source and currently features among the top 50% of packages on the Python Package Index (PyPI) repository [233],

having had over 1.1k downloads in last 90 days at the time of writing. It is also hosted on GitHub [240] and featured in the Global PyTorch Hackathon 2019.

Although Deeplodocus has been instrumental to the implementation, testing, and development of the multiple CNN-based models explored within the body of work described in this thesis, the open-source nature of its publication has led to its adoption elsewhere. Within the Cranfield-centric universe known to this researcher, Deeplodocus has contributed towards the development of a bespoke CNN for infrared footprint segmentation; estimating distance, surface orientation, and optical flow for the purposes of space navigation using a monocular camera; and has been chosen to facilitate the development of ATR algorithms on behalf of multinational defence company Leonardo. In addition, Google searches suggest its use within the mobile cloud architecture of Chinese telecommunications firm Huawei; various cryptocurrency mining operations; and by a Japanese technical consultancy [241, 242]

4.1. The overarching design principles of Deeplodocus

Deeplodocus was built with four design principles in mind. These were:

Modularity: Deeplodocus should enable any number of third-party or custom-built low-level modules to be incorporated into its inference pipelines.

Flexibility: Deeplodocus should be able to accommodate a wide range of supervised deep learning applications, and be able to facilitate the creation of new deep learning techniques and algorithms.

Speed and efficiency: Deeplodocus should facilitate the rapid initialisation of new deep learning projects, and make efficient use of available CPU and GPU hardware.

High-capacity and scalable: Deeplodocus should scale to accommodate any number of low-level software components, any quantity of data, and any size of neural network.

4.2. An overview of Deeplodocus

Figure 4-2 illustrates how Deeplodocus works. It shows that when implementing a deep learning project with Deeplodocus, users have control of three key elements: the dataset(s), a number of low-level modules, and a collection of high-level configuration files.

Configuration files hold the settings that control the modules. User modifications to these files inform Deeplodocus of the required modules and data, and instruct Deeplodocus on how they should be used. At run-time, Deeplodocus adheres to these configurations to load the chosen modules into multiple 'wrapper' objects, each with a specific role, which act to house modules and manage their operations, and ensure a smooth flow of data between them.

For example, the Dataset wrapper (see Figure 4-2, Box A) accommodates any number of data-loader modules, of which can have one of three purposes: loading input examples, loading ground truth data, or loading any additional data. Once the configured modules are in place, the Dataset wrapper acts as a single collection of data, which in turn, acts as a single point of control that utilises multiple CPU workers to locate and load data as quickly as possible. As with all Deeplodocus wrapper objects, these modules may be either custom-built, third-party, or one of the default modules native to Deeplodocus.

The Input Transform Manager (see Figure 4-2, Box B) is another wrapper, and accommodates any number of data-transform modules, in order to augment and or pre-process the input examples, labels, and additional data. When constructing an inference pipeline, Deeplodocus embeds an Input Transform Manager within each Dataset to create a single object capable of seamlessly loading and transforming any quantity of data, via any number of custom-built or third-party data-loaders and data-transforms.

Users of Deeplodocus must also specify one deep learning model and—in order to train the model—one optimisation algorithm. The deep learning model is initialised into a Model wrapper (see Figure 4-2, Box C), which automatically enables multi-GPU training and inference, while the optimiser is initialised into

an Optimiser wrapper (see Figure 4-2, Box D), which controls the optimiser module, and ensures that user configuration instructions are met.

Deeplodocus can accommodate any number of modules for calculating loss functions and evaluation metrics, which are managed by a Losses wrapper and a Metrics wrapper respectively (see Figure 4-2, Boxes E and F). A key task of these specific wrappers is to accumulate the resultant loss and metric scores across the dataset in order to provide the essential summary statistic for each function and metric respectively.

The final wrapper of note is the Output Transform Manager (see Figure 4-2, Box G), which controls any post-processing transforms that are to be applied to each batch data outputted by the deep learning model, before the model's outputs are evaluated via the given metrics.

Once all the wrappers have been constructed, they are then assembled within Deeplodocus to form the various inference pipelines (see Figure 4-2, Boxes H, I, J and K), which may then be executed by the user as required.

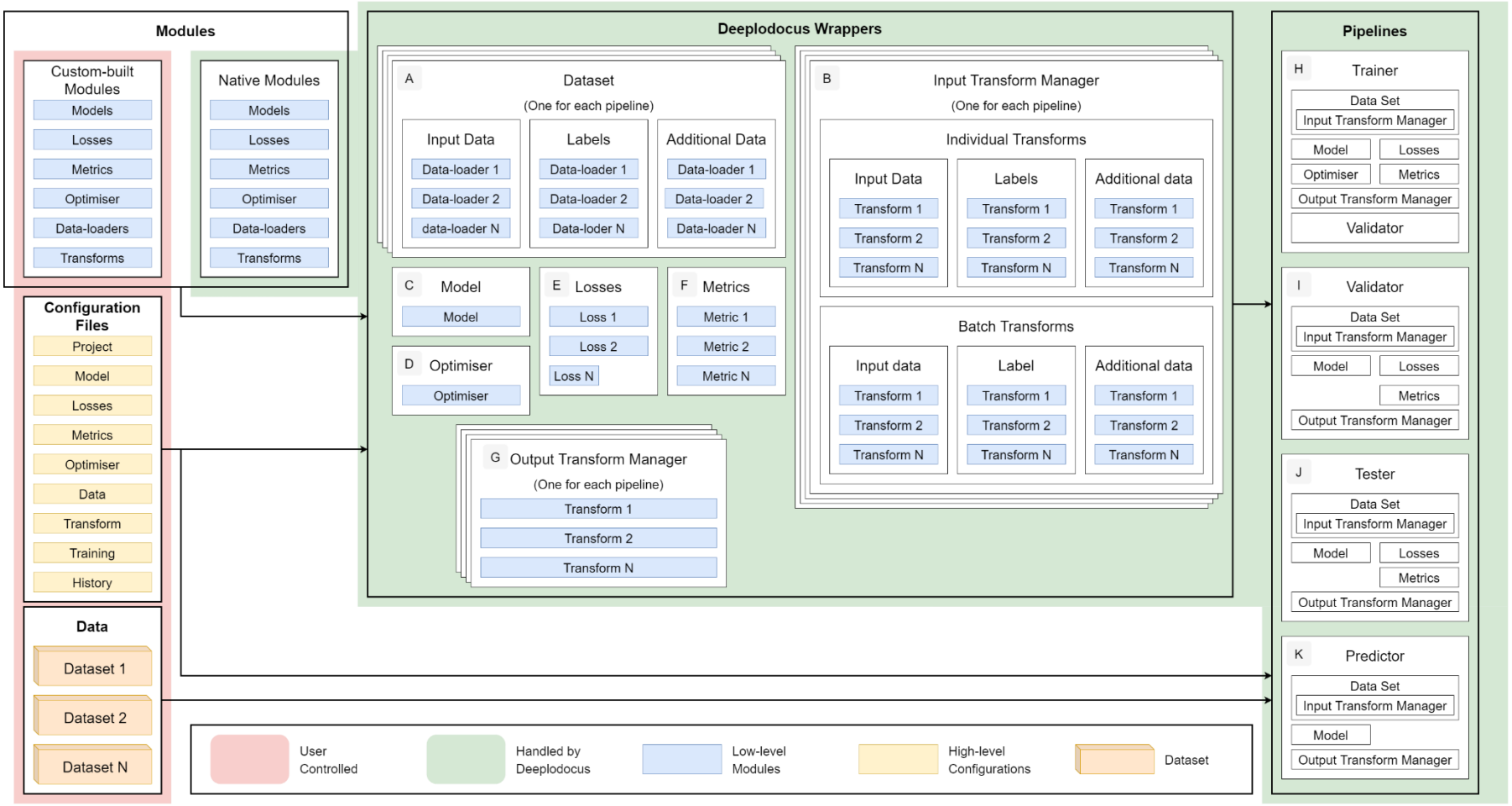


Figure 4-2: *Overview of the Deeplodocus deep learning development environment, showing the three user-controlled elements, key Deeplodocus wrappers, and the four inference pipelines.*

4.2.1. Inference pipelines

There are four inference pipelines within Deeplodocus, which are used for the tasks of training, validation, testing, and inference. These pipelines are respectively defined by Deeplodocus' `Trainer`, `Validator`, `Tester`, and `Predictor` objects.

To train a selected deep learning model, Deeplodocus first assembles a training pipeline to enable the required data flow between the specified modules. In the context of inference pipelines, 'data' can be one of three main types:

- **Input data:** used by the model to perform inference.
- **Label data:** the response desired from the model for a given input.
- **Additional data:** data pertinent to the calculation of loss functions and or evaluation metrics.

In the field of computer vision and object detection, for instance, if the input data is an image, then the label data could be an array of ground truth bounding boxes denoting the position and size of foreground objects, and the additional data could be comprised of metadata relating to the input image, which inform the calculation of losses and evaluation metrics.

4.2.1.1. The training pipeline

Expanding on Box H of Figure 4-2, Figure 4-3 shows how these different types of data flow between modules within the training pipeline. Appendix A-4 outlines the types of data that are accepted and returned by each module.

For each batch of training data, the workflow is as follows:

1. The Dataset object locates and loads a batch of transformed inputs, ground truth labels, and additional data.
2. The Model performs a forward pass on the input data and returns a batch of outputs.
3. The Loss Functions evaluate the outputs with respect to the ground truth labels.

4. The resultant loss is used by the Optimiser to perform a backward pass through the model.
5. Outputs from the model are post-processed by the Output Transform Manager.
6. The Evaluation Metrics then evaluate the processed outputs with respect to the ground truth labels.

Once all the training batches have been completed, the workflow continues as follows:

7. Losses and Metrics are averaged across all training batches.
8. Optionally, the Model is passed to the validator and its performance is evaluated with respect to the validation dataset.

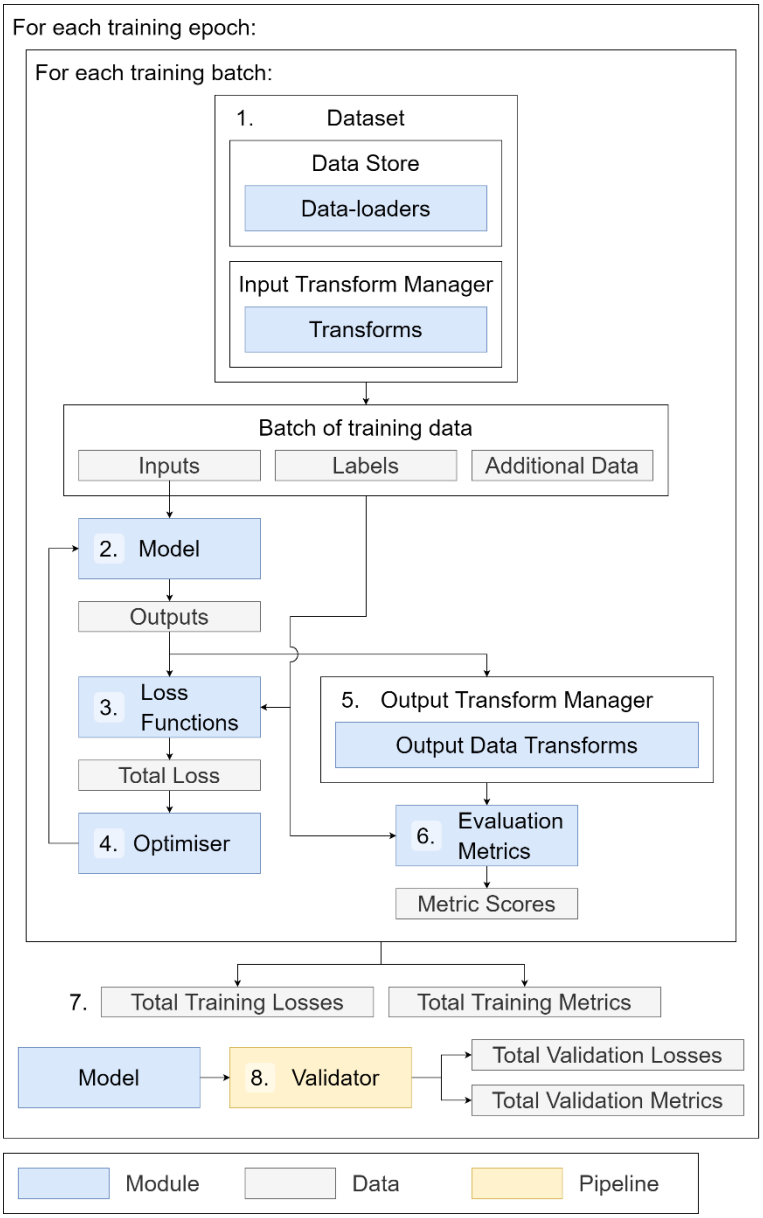


Figure 4-3: *Overview of the Deeplodocus training pipeline showing the main flow of data.*

While this overview has described the flow of data typically required to train a model, other routes through the training pipeline are possible. One example of an alternative data route would be to pass input and label data to the output transform module in order to provide visual feedback by illustrating the model's predictions.

4.2.1.2. The validation and test pipelines

To validate and test the deep learning model, Deeplodocus constructs inference pipelines as shown in Figure 4-4. Both the validation and test pipelines follow the same operating procedure, but depend on different sets of data, and may be comprised of different modules. In contrast to the training pipeline, the validation and test pipelines iterate only once through the applicable dataset, do not include the optimiser module, and do not perform a backward pass of the model.

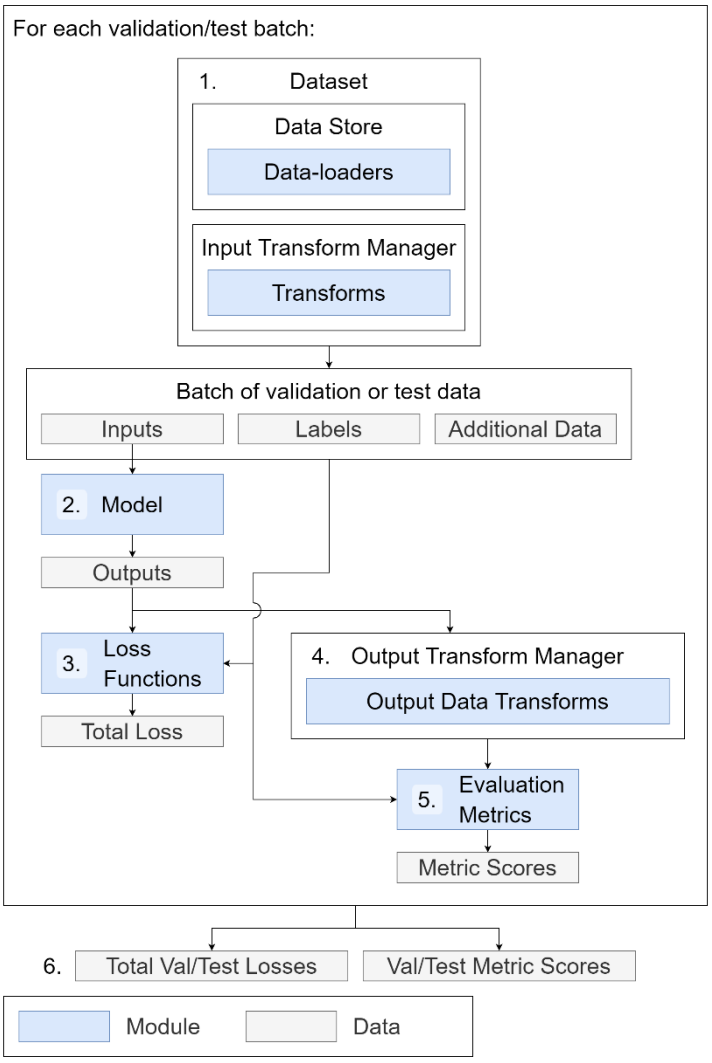


Figure 4-4: *Overview of the Deeplodocus validation and test pipelines.*

4.2.1.3. The prediction pipeline

Deeplodocus constructs a prediction pipeline, shown in Figure 4-5, to infer predictions on unlabelled data. This inference pipeline terminates with the Output Transform Manager, and enables post-processing of the model output before moving on to the next batch of input data. In contrast to the training, validation, and testing pipelines, the prediction pipeline does not evaluate the model with respect to any loss functions or evaluation metrics, and so the provision of ground truth labels is optional.

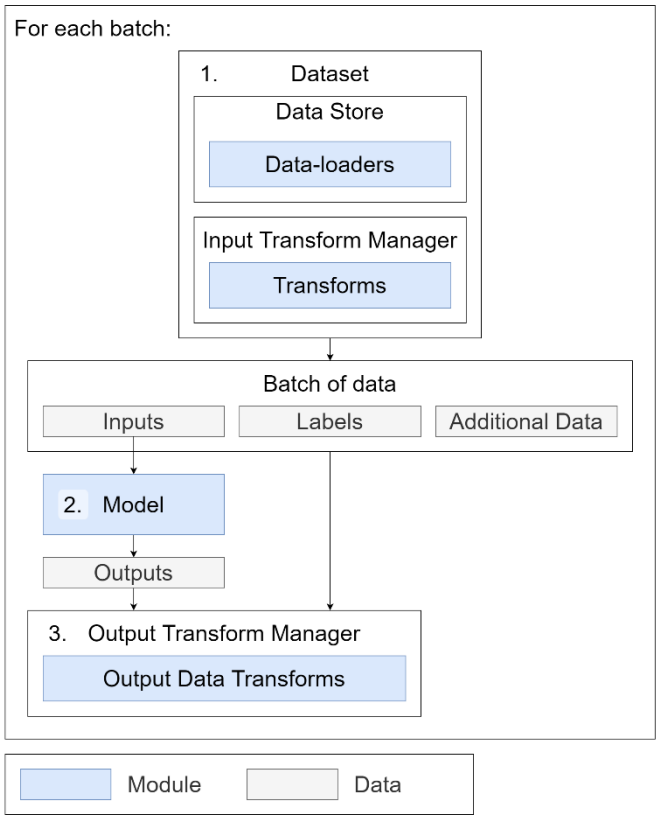


Figure 4-5: *Overview of the Deeplodocus prediction pipeline.*

4.3. How to use Deeplodocus

There are four basic steps to training a deep learning algorithm using Deeplodocus:

1. Install Deeplodocus.
2. Initialise a new Deeplodocus Project.

3. Configure the project to include the required dataset(s) and modules.
4. Execute Deeplodocus.

4.3.1. Installation of Deeplodocus

Deeplodocus is hosted on both GitHub and the PyPI repository, and can be installed using the python package-management system, pip, with the command:

```
pip install deeplodocus
```

This command will install all Deeplodocus dependencies, with the exception of PyTorch [243] which must be installed independently.

4.3.2. Initialising a new project

New deep learning algorithms are constructed within a Deeplodocus Project, which can be initialised using the command:

```
deeplodocus newproject <project_name>
```

As shown in Figure 4-6, a newly initialised Project contains four items at its top level: a main Python script—used to execute the Project—a directory for housing any custom-built modules, a directory for housing sets of example data, and a directory containing the high-level configuration files.

Configuration files are initialised with default values that represent a simple working example which utilises the MNIST dataset of handwritten digits [244] to train a LeNet-5 image classification network [245] with a Cross Entropy loss function [90], and an Adam optimiser [246].

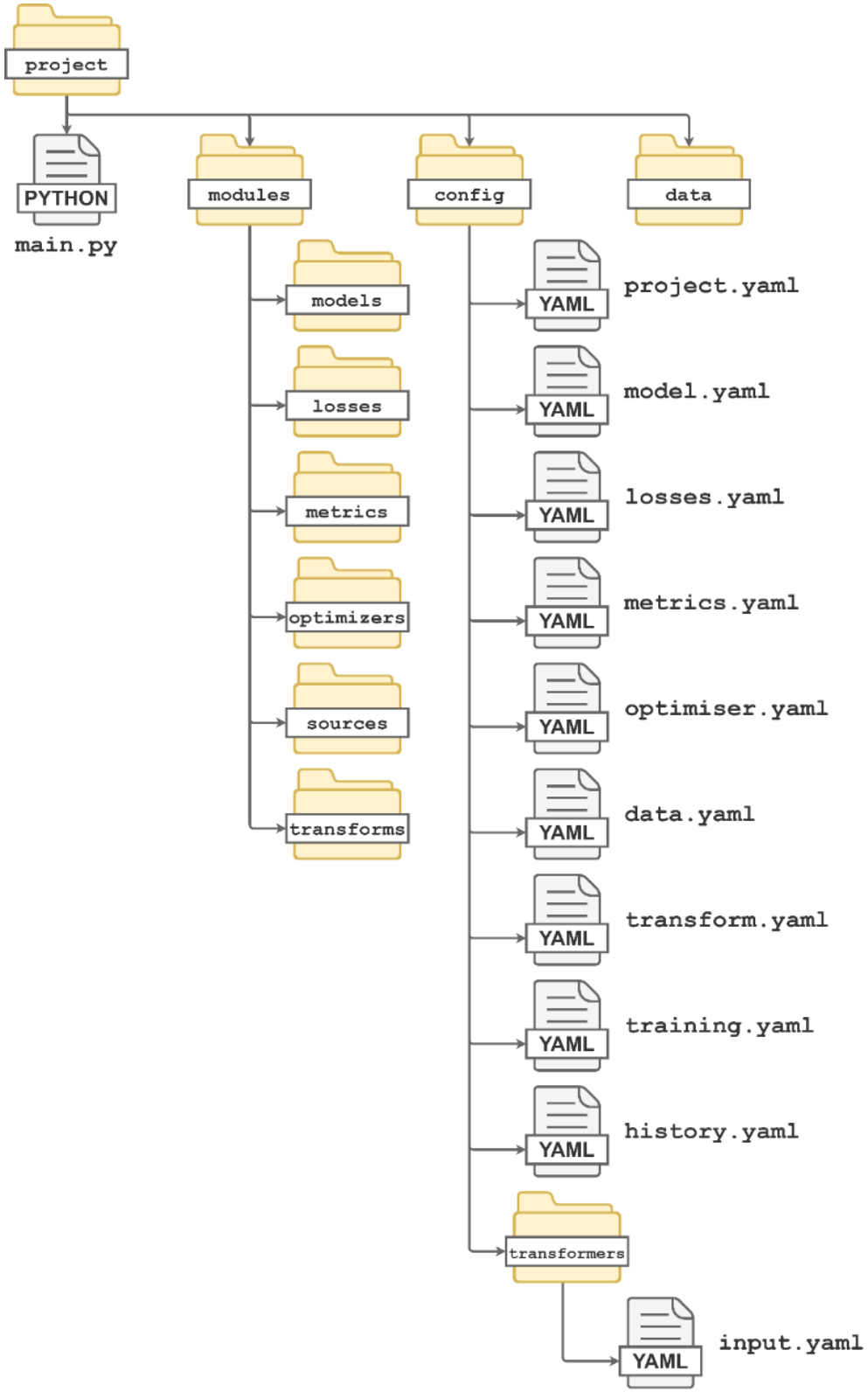


Figure 4-6: *The structure of a newly-initialised Deeplodocus Project. At the top level, there are three directories for containing datasets, custom-built modules, and configuration files respectively, and a Python script for execute the project.*

4.3.3. Configuring the project

The Deeplodocus Project can be configured for a set purpose by modifying its nine configuration files. The general purpose of each such file is outlined in Table 4-1.

Table 4-1: *A summary describing the general purpose of each Deeplodocus configuration file.*

Configuration file	Function
`project.yaml`	Controls the overarching configurations encompassing the entire project, such as: • where to save resultant files, • which GPUs to use, and • which commands to run on start-up.
`model.yaml`	Controls all settings relating to the deep learning model, such as: • the path to the module containing the model, • whether to initialise the model with any pre-trained weights, • the path to the pre-trained weights (if required), and • any user-defined hyper-parameters.
`losses.yaml`	Specifies the loss function(s) to be used, and the user-defined parameters for each loss function.
`metrics.yaml`	Specifies the evaluation metric(s) to be used and the user-defined parameters for each evaluation metric.
`optimiser.yaml`	Specifies the optimisation algorithm to be used, sets any user-defined parameters—such as learning rate and weight decay—defines any parameter groups, and controls whether any pre-existing internal parameters should be loaded.
`data.yaml`	Controls all aspects of data loading, including: • which data should be used for training, validation, testing, and prediction respectively, • which modules should be used to load this data, • the number of CPU workers, and • the batch sizes.
`transform.yaml`	Controls the data transforms to be applied to the input data, labels, additional data, and output data during each of training, validation, testing, and prediction pipelines.
`training.yaml`	Controls aspects of the training pipeline, such as: • the number of training epochs, • the learning rate scheduler,

	• the criteria for saving model checkpoints, and • the verbosity of training notifications.
`history.yaml`	Controls which loss and metric scores are outputted into CSV files.

4.3.3.1. Global configurations

The global configurations for a Deeplodocus project can be specified through the `project.yaml`. An example of this is shown in Figure 4-7, with explanatory test as follows.

project.yaml

```yaml
session: version-01       # The directory to which any weight/log/result files should be written.
enable_log: True          # Whether to write a log of all notifications (True / False).
device: auto              # Which hardware device to use (cpu, gpu, or auto).
device_ids: auto          # Which GPU devices to use (auto or [0, 1, ...]).
on_wake:                  # List of commands to execute on start-up, for example:
  - config.snapshot()       # Save a snapshot of all current configurations.
  - load()                  # Load all modules.
  - train()                 # Execute the training pipeline.
  - test()                  # Execute the test pipeline.
  - sleep()                 # Terminate.
imports:                  # Import any third-party pacakges.
  - numpy as np
```

Figure 4-7: *Example of a project.yaml configuration file.*

The `session` field in Figure 4-7 specifies the directory to which its files will be written, and this facilitates multiple training runs under various conditions.

The `device` field specifies the hardware device on which the model will operate. Valid options include `cpu`, `gpu`, and `auto`, where auto will set device to GPU if one is available, otherwise it will set the device to CPU.

Deeplodocus supports the use of multiple GPUs during model training, validation, and testing, and the precise GPU devices to be used can be specified by using the `device_id` field. GPU devices can be specified individually using a list of corresponding device ID numbers, or users may use the `auto` option, which automatically selects all available GPU devices.

Another notable field is `on_wake`, which enables users to specify a list of commands to execute on start-up. The on-wake commands shown in Figure

4-7, for example, will first write a snapshot of all current configurations to `version-01/config_yyyy-MM-dd-HH-mm-ss.yaml`, load all modules, train the model, and then evaluate the model with respect to the test examples before terminating the program.

Finally, lambda functions can be inputted instead of parameter values within Deeplodocus configuration files such as `optimizer.yaml`, in order to perform specific tasks. Such lambda functions are one-line functions that may require third-party packages such as NumPy to be imported via the imports field as shown in Figure 4-7.

4.3.3.2. Configuring the deep learning model

The deep learning model is configured by modifying the `model.yaml` file. As shown in Figure 4-8, this file specifies the name of the required model, which Python module to load it from, which pre-trained weights (if any) should be loaded into the model, and any user-defined keyword arguments.

model.yaml

```yaml
name: LeNet                          # Name of the deep learning model.
module: deeplodocus.app.models.lenet # Path to model.
from_file: False                     # Whether to load internal parameters from a file.
file: Null                           # Path to internal parameter file.
input_size:                          # The shape of each input to the model.
  - [1, 28, 28]
kwargs:                              # Keyword arguments to pass to the model
  num_classes: 10
```

Figure 4-8: *A model.yaml configuration file that points to the LeNet image classification network from Deeplodocus.*

Users can choose to load either a custom-built model, a model supplied by a third-party package, or one of the native implementations of, for example, LeNet, AlexNet, and VGG [245, 135, 129] provided by Deeplodocus. Each of these in-house models is located in separate modules within `deeplodocus.app.models`. In Figure 4-8, for example, the selected model is `LeNet` from the `deeplodocus.app.models.lenet` module.

To include a custom-built model, or a model supplied by a third-party package, they must first be implemented using the PyTorch framework within a Python

module, located either within the `modules/models` directory of the project, or the applicable package. The model must be implemented as a Python Class that inherits from `pytorch.nn.Module`, and defines an `__init__` constructor, which initialises the attributes of the model. This constructor may accept any number of keyword arguments, and its values can be specified via the `kwargs` field. As is standard practice when using PyTorch, the operation of the model's forward pass must be defined within `forward`, a method which may accept any number of positional arguments and returns the result of the forward pass.

As an example, Figure 4-9 shows how the LeNet image classification network can be implemented in this way.

Once a custom-build model has been implemented it can then be specified by its name and module which, in the case of Figure 4-9, would be `LeNet` and `modules.models.lenet` respectively.

lenet.py

```python
import torch.nn as nn
import torch.nn.functional as F


class LeNet(nn.Module):
    """
    Implementation of LeNet as defined in http://yann.lecun.com/exdb/lenet/
    Modified to use ReLU activation instead of sigmoid activation,
    and to use max pooling instead of average pooling.
    """
    def __init__(self, in_channels=1, num_classes=10):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=6, kernel_size=(5, 5))
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=(5, 5))
        self.fc1 = nn.Linear(in_features=16 * 5 * 5, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84, out_features=num_classes)

    def forward(self, x):           # DEFINE THE FORWARD PASS
        x = self.conv1(x)           # 1st convolutional layer
        x = F.relu(x)               # ReLU activation
        x = F.max_pool2d(x, 2)      # Max pooling
        x = self.conv2(x)           # 2nd convolutional layer
        x = F.relu(x)               # ReLU activation
        x = F.max_pool2d(x, 2)      # Max pooling
        x = x.view(x.shape[0], -1)  # Flatten
        x = F.relu(self.fc1(x))     # 1st fully-connected layer
        x = F.relu(self.fc2(x))     # 2nd fully-connected layer
        x = self.fc3(x)             # 3rd fully-connected layer
        return x
```

Figure 4-9: *Example implementation of the LeNet-5 image classification network using the PyTorch framework.*

To initialise the model with a set of pre-trained weights and biases, the `from_file` option should be set to `True`, and the requisite file of PyTorch parameters can be specified using the `file` option.

4.3.3.3. Configuring loss functions

Loss functions are configured by modifying the `losses.yaml` file, an example of which is shown below in Figure 4-10. Each loss function requires a unique name and can be specified via four fields—`name`, `module`, `weight`, and `kwargs`—where `name` and `module` reference the loss function itself, `kwargs` specifies any keyword arguments to be passed to the module, and `weight` specifies a multiplicative scale factor, which will be applied to the calculation of the loss function.

losses.yaml

```yaml
CELoss:                             # Unique human-readable name.
  name: CrossEntropyLoss            # Name of loss object.
  module: torch.nn.modules.loss     # Module to import from.
  weight: 1
  kwargs: {}
```

Figure 4-10: *A losses.yaml configuration file that points to the CrossEntropyLoss from PyTorch.*

The example shown in Figure 4-10 specifies a single loss function—an implementation of Cross Entropy loss, sourced from PyTorch's `torch.nn.modules.loss` module. Deeplodocus can accommodate any number of loss functions specified in this manner, and can also accept custom-built loss functions.

Similar to custom-built models, custom-built loss functions must also be implemented as a Python Class which inherits from `nn.module` and must contain an `__init__` constructor to initialise its attributes, and a `forward` method that defines the loss calculation. Further, the loss function module should be located within the `modules/losses` directory.

The `forward` method accepts up to five positional arguments: inputs, labels, additional data, outputs, and the deep learning model. The order of these positional arguments is not important, since each loss function is inspected on start-up by the Losses wrapper object, which then ensures that the expected arguments are provided correctly. However, each argument must be named using the following convention:

Input: `x`, `input`[3], `inputs`, `inp`

Labels: `y`, `label`, `labels`, `lab`, `target`, `targets`

Additional data: `additional_data`, `additonaldata`

Outputs: `y_hat`, `outputs`, `output`

[3] Note that Python contains a built-in function with this name, so the use of '`input`' as a variable name should be discouraged.

Model: `model`, `network`

Figure 4-11 gives an example of how a custom-built loss function can be implemented in this way, and shows the definition of the weighted Cross Entropy Loss between the model outputs, `y_hat`, and the ground truth labels, `y`, with Logistic SoftMax activation.

crossentropy.py

```python
from torch import nn


class CrossEntropyLoss(nn.Module):

    def __init__(self, weights=None):
        super().__init__()
        # If any class weights are given, initialise them as a torch tensor
        self.w = None if weights is None else torch.tensor(weights, dtype=torch.float32)

    def forward(self, y_hat, y):
        # Apply log softmax activation to the model outputs
        y_hat = self.log_softmax(y_hat)

        # Create a mask which ignores predictions that do not correspond to a target
        i = torch.eye(y_hat.shape[1])[y].to(dtype=torch.bool)

        # If there are no class weights
        if self.w is None:
            # Calculate the loss value
            return torch.sum(-y_hat[i]) / len(y_hat)

        # If class weights are given
        else:
            # Calculate the weighted loss value
            return torch.sum(-y_hat[i] * self.w[y]) / torch.sum(self.w[y])

    @staticmethod
    def log_softmax(x):
        x = torch.exp(x)
        return torch.log(x / torch.sum(x, dim=1).view(-1, 1))
```

Figure 4-11*: Example implementation of the Cross Entropy loss function as a Python Class with PyTorch.*

4.3.3.4. Configuring the evaluation metrics

Evaluation metrics are configured into the project via the `metrics.yaml` file, an example of which is shown below in Figure 4-12. Each metric requires a unique name, and can be specified via six fields—`name`, `module`, `weight`, `kwargs`, reduce, and `ignore_value`—where `name` and `module` reference the

loss function itself, and `kwargs` specifies any keyword arguments to be passed to the metric.

metrics.yaml

```yaml
Accuracy:                                  # A unique human-readable name for the metric.
  name: accuracy                           # Name of the metric object/function.
  module: deeplodocus.app.metrics.accuracy # Module to import from.
  kwargs: {}                               # Any keyword arguments.
  reduce: mean                             # How to reduce the metric scores).
  ignore_value: Null                       # Which value (if any) to ignore when reducing.
```

Figure 4-12: *A metrics.yaml configuration file that points to the accuracy metric from Deeplodocus.*

The `reduce` field is used to specify the method of summarising the resultant metric scores across all batches to a single value, and can be one of `mean`, `sum`, or `latest`. If `mean` is specified, metric scores will be summarised via the arithmetic mean; if `sum` is specified, the sum of all metric scores will be calculated; and if `latest` is specified, the metric score of the last batch will be used. The latter two options are useful when the metric is a running tally of, for example, the number of true positives.

If an `ignore_value` is specified, any instances of this value will be removed from the list of metric scores before they are reduced.

Custom-built evaluations can be implemented as either Python functions or as Python classes, and should be located within the `modules/metrics` directory of the Project. In contrast to models and losses, evaluation metrics do not require `torch.autograd`, and do not need to inherit from `nn.module`. An example implementation of a metric for evaluating image classification accuracy is shown below in Figure 4-13.

accuracy.py

```python
import torch


def accuracy(y_hat, y):
    _, y_hat = y_hat.max(dim=1)
    return torch.mean((y_hat == y).float())
```

accuracy.py

```python
import torch


class Accuracy(object):

    def __init__(self):
        pass

    @staticmethod
    def forward(y_hat, y):
        _, y_hat = y_hat.max(dim=1)
        return torch.mean((y_hat == y).float())
```

Figure 4-13: *Example implementations of a metric for evaluating classification accuracy, both as a Python Class (left), and as a Python Function (right).*

4.3.3.5. Configuring an optimiser

An optimiser is configured by modifying the `optimizer.yaml` file, as shown in Figure 4-14. The name of the optimisation algorithm, and the Python module from which it should be imported, are specified with the fields `name` and `module` respectively. Any keyword arguments which should be passed to the optimiser upon its initialisation can be specified using the `kwargs` field. The example in Figure 4-14 configures the Adam optimiser [246] from the `torch.optim` module [247], with a learning rate of 0.001.

optimizer.yaml

```yaml
name: Adam                # Name of the optimizer object to use.
module: torch.optim       # Module to import from.
load_state_dict: False    # Whether to load parameters from a previous optimizer state.
kwargs:                   # Any keyword arguments:
  lr: 1e-3                  # Learning rate.
param_groups:             # Define parameter groups (optional).
  - condition: "lambda x: x.endswith('.weight')" # All parameters with names ending in '.weight'.
    kwargs:                                      # Keyword arguments:
      weight_decay: 0.0005                         # Set weight decay.
  - condition: "lambda x: x.endswith('.bias')"   # All parameters with names ending in '.bias'.
    kwargs:                                      # Keyword arguments:
      weight_decay: 0.0                            # Set weight decay.
```

Figure 4-14: *An optimizer.yaml file which points to the Adam optimiser from PyTorch.*

When loading pre-trained weights into the deep learning model, the file containing the weights may optionally hold a copy of the optimiser's internal state from the time the weights were recorded. If this internal state is present,

and the `load_state_dict` field is set to `True`, the internal state will be loaded into the optimiser on start-up. Otherwise, if `load_state_dict` is set to `False`, the copy of the optimiser's internal state will be ignored.

Control over when a prior internal state is loaded into the optimiser is important for two particular scenarios:

1) When resuming training of a partially-trained model the optimiser's internal state must first be re-instated in order to ensure continuity of the training process.

2) Alternatively, when a model is being re-trained for a new purpose, the optimiser's prior internal state should be ignored before training is resumed.

The `optimizer.yaml` file may also be used to specify distinct parameter groups within the model. This can be achieved by using the `param_groups` field, which, as shown in Figure 4-14, expects a list of conditions and keyword arguments. Each condition must be specified within quotation marks, using the syntax of a Python lambda function declaration, which when given the name of a parameter, must return either `True` or `False`.

Upon initialisation of the optimiser, if any parameter groups are specified, Deeplodocus will cycle through the internal parameters of the model and assign each to the appropriate parameter group. In the example presented in Figure 4-14 for instance, all internal parameters with names ending in `.weight` will be optimised using a weight decay of 0.0005, whereas no weight decay will be applied to the internal parameters with names ending in `.bias`.

As shown in Figure 4-15, Deeplodocus also notifies the user of the grouping of model parameters, and the optimisation settings that have been assigned to each such group.

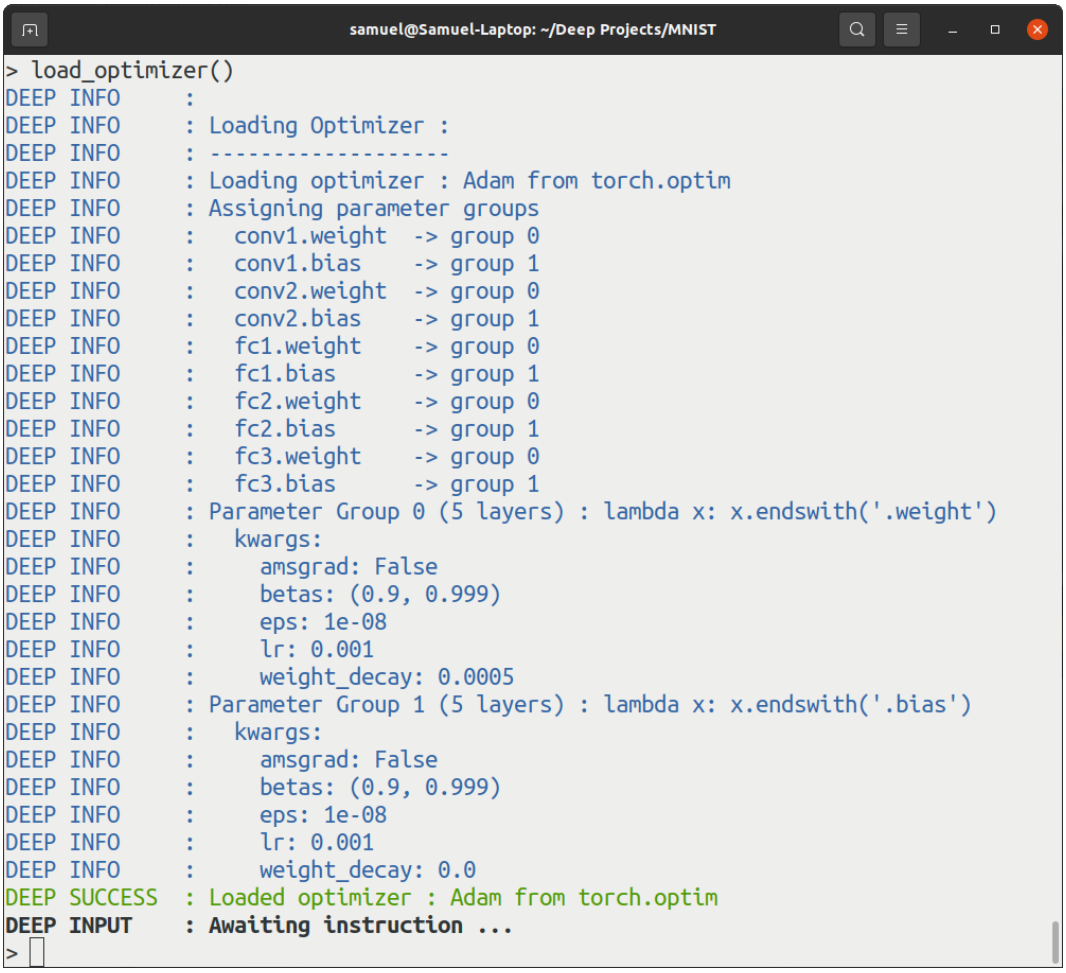


Figure 4-15: *An example of how Deeplodocus reports the composition of the different parameter groups and the optimisation settings for each group.*

4.3.3.6. Configuring a Dataset

Datasets are incorporated into the project by modifying the `datasets` field within the `data.yaml` configuration file, an example of which is shown in Figure 4-16.

data.yaml

```yaml
dataloader:
  num_workers: 4                # Use 4 CPU workers to load examples.

enabled:                       # Enable the required pipelines.
  train: True                  # Enable the trainer.
  validation: True             # Enable the validator.
  test: False                  # Do not initialise the tester.
  predict: False               # Do not initiaise the predictor.

datasets:                      # Specify the datasets.
  # TRAINING DATA
  - name: MNIST-train            # A human-readable name for the dataset.
    type: train                    # One of: 'train', 'validation', 'test', or 'prediction'.
    batch_size: 64                 # Load examples in batches of 64.
    num_instances: Null            # Use all examples.
    entries:                       # Define the dataset entries:
      # INPUT DATA
      - name: MNIST data             # A human-readable name for the input entry.
        type: input                  # One of: 'input', 'label', 'additional_data'.
        load_as: image               # Treat the data as an image.
        convert_to: float32          # Convert the images to 32-bit floats.
        move_axis: [2, 0, 1]         # Transpose the image (h x w x ch) -> (ch x h x w).
        enable_cache: True             # Required when using a source pointer
        sources:                       # Specify the 'source' of the MNIST examples.
          - name: MNIST                  # The MNIST dataset.
            module: torchvision.datasets  # Module to import from.
            kwargs:                       # Any keyword arguments:
              root: ./data/MNIST            # Root directory of dataset.
              train: True                   # Wheter to use the training portion.
              download: True                # Whether to download the dataset if required.
      # LABEL DATA
      - name: MNIST labels           # A human-readable name for the label entry.
        type: label                  # One of: 'input', 'label', 'additional_data'.
        load_as: integer             # Treat the data as an integer.
        convert_to: int64            # Convert labels to 64-bit integers.
        move_axis: Null              # No need to transpose the data.
        enable_cache: False          # No need to cache the data.
        sources:                     # Specify the 'source' of the label data.
          - name: SourcePointer        # Use a SourcePointer.
            module: Null               # Load it from the Deeplodocus modules.
            kwargs:                    # Any keyword arguments:
              entry_id: 0                # Select from the 1st entry.
              source_id: 0               # Select from the 1st source.
              instance_id: 1             # Select the 2nd item.
    # VALIDATION DATA
    - name: MNIST-val
      type: val
      ...
```

Figure 4-16: *An example data.yaml configuration file which specifies a the training portion of the MNIST dataset as training data.*

The `datasets` field is used to define a list where each item corresponds to a set of data for either training, validation, testing, or prediction. Each of these is, in turn, defined using a further five fields: `name`, `type`, `batch_size`, `num_instances`, and `entries`.

The `name` field allows users to give each set a human-readable name.

The `type` field is used to specify the purpose of the data, and must be one of `train`, `val`, `test`, or `predict`.

The `batch_size` field is used to define the number of instances that each batch yielded by the Dataset wrapper should contain.

The `num_instances` field offers control over the amount of data from the set which should be used. If `num_instances` is specified with an integer, the size of the dataset will be truncated to this quantity, otherwise if `num_instances` is `Null`, the Dataset wrapper will iterate through all instances of the dataset.

Finally, the `entries` field is used to define a list of items, where each item corresponds to either input data, label data, or additional data. Each of these items are defined through a further seven fields: `name`, `type`, `load_as`, `convert_to`, `move_axis`, `enable_cache`, and `sources`.

For each entry, the `type` field specifies whether the entry should be treated as input data, label data, or additional data.

The `source` field is used to specify a list of data-loader modules which will load the data.

In this example, shown in Figure 4-16, the source of training input data is the `MNIST` data-loader from PyTorch's `torchvision.datasets` module [248]. Furthermore, the `kwargs` field has been used to specify the `root`, `train`, and `download` keyword arguments, directing the data-loader to load training data from the `data/MNIST` directory of the project, and to download the data automatically if required.

The `MNIST` data-loader does not return input data only, but rather image-label pairs. Therefore, the source of training label data does not need to be another `MNIST` data-loader, it can instead be a `SourcePointer`, which simply points to the second item of the first `source` in the first `entry`.

As a result of this flexible approach to configuring different data-loader modules, users of Deeplodocus can easily combine any number of different datasets simultaneously during algorithm training, validation, testing, and prediction.

4.3.3.7. Configuring data transforms

Depending on the application in question, data may or may not require pre-processing. Where pre-processing is required—such as in the case of images, for example—this is accomplished through the use of data-transform modules. In complex pre-processing scenarios, sequences of data-transform modules may be required.

For example, images could be normalised and reshaped in sequence by specifying multiple modules, as shown in Figure 4-17.

input.yaml

```yaml
method: sequential                                  # Apply tansforms sequentially
name: Transformer for MNIST input

mandatory_transforms_start:
  - normalize:                                      # Human-readable name
      name: normalize_image                           # Name of the transform function
      module: deeplodocus.app.transforms.image        # Module to import from
      kwargs:                                         # Keyword arguments for the transform
        standard_deviation: 1.0
        mean: 0.0
  - reshape:                                        # Human-readable name
      name: reshape                                   # Name of the transform function
      module: deeplodocus.app.transforms.generic      # Module to import from
      kwargs:                                         # Keyword arguments
        shape: [28, 28, 1]
transforms: Null
mandatory_transforms_end: Null
```

Figure 4-17: *Example of a YAML file, which specifies a sequence of data-transform modules to normalise and reshape the input data.*

4.3.3.8. Configuring the training pipeline

The training pipeline can be configured using the `training.yaml` file, shown in Figure 4-18, which controls the main settings describing:

- the number of training epochs,
- data shuffling,
- the learning rate schedule, and
- how and when to save the model.

The `scheduler` field is used to define an object that will incrementally alter the learning rate throughout the training process. The configuration file shown in Figure 4-18, for example, specifies PyTorch's `ExponentialLR` scheduler in order to reduce the learning rate after each training epoch via exponential decay with a decay constant of 0.95.

The `saver` field is used to specify how regularly a snapshot of model's internal parameters should be saved to a file. In the example shown in Figure 4-18, the `save_signal` field is set to `auto`, meaning the model parameters will be saved only when the condition specified by the `overwatch` field are met. This condition, in this instance, is that the model's total validation loss must be less than all previous total validation loss values achieved by the model.

training.yaml

```yaml
num_epochs: 20                      # Train for 20 epochs.
verbose: batch                      # Print feedback info after each batch.
shuffle: all                        # Shuffle the training data after each epoch.
accumulate: 1                       # Number of mini-batches for gradient accumulation.
enable_metrics: True                # Calculate metrics during training.

scheduler:                          # Specify a learning rate scheduler.
  name: ExponentialLR                 # Use the ExponentialLR scheduler.
  module: torch.optim.lr_scheduler    # Import from a PyTorch module.
  enabled: True                       # Enable the scheduler.
  kwargs:                             # Any keyword arguments.
    gamma: 0.95

saver:                              # When and how to save a model checkpoint.
  save_signal: auto                   # Only save if overwatch conditions are met.
  method: pytorch                     # Save with PyTorch format.
  overwrite: False                    # Do not overwrite pre-existing checkpoints.

overwatch:                          # Futher conditions for saving a checkpoint.
  dataset: validation                 # Which dataset to consider.
  metric: Total Loss                  # Which 'metric' to compare.
  condition: less                     # Save when current metric is less than the previous best.
```

Figure 4-18: *A training.yaml configuration file.*

4.3.4. Executing Deeplodocus

Deeplodocus can be executed by running the default main script with:

```
python3 main.py
```

On start-up, Deeplodocus reads each configuration file from the `config` directory and stores all contents within a single `Config` object.

Deeplodocus performs a validation of all configuration entries on start-up. This check is conducted with the aid of a template, which specifies permissible data types and default values for every field in the configuration files.

If the value given for any field conforms to any of the permissible data types, Deeplodocus will accept the value and continue to the next field. However, if a given value does not conform to any permissible data types, Deeplodocus will attempt to convert the given value to each of the permissible data types in turn. If the value is subsequently converted to an acceptable data type in this manner, Deeplodocus replaces the given value with this converted value, and raises a warning message to the user, to inform them of this change.

4.4. Conclusion

Developing a modern deep learning algorithm is a complex undertaking, demanding a substantial investment of time, skill, and effort. In order to progress to develop, train and test convolutional neural network algorithms, it is necessary to test and analyse a large number of architectures and hyper-parameters before the ‘right’ model can be selected. Only a small proportion of any machine learning system is actually made up of the machine learning algorithm itself, with the remainder comprising an extensive and complex supporting infrastructure.

Building and operating this infrastructure is an intensive task, consuming valuable time and resources that could otherwise be dedicated to further develop the algorithm to improve its performance.

For research teams, the result is that it is all too easy for the time scheduled for developing and improving deep learning algorithms to be spent on managing infrastructure and jobs instead. Automating this infrastructure, and embedding it within a workflow pipeline, is therefore vital.

Deeplodocus was built to achieve this. Had Deeplodocus not been developed in 2018, this body of work would not have:

- been able to invest so much time in building models, rather than infrastructure,
- had the time to experiment with many different algorithm training conditions, or
- had efficiently made use of the required custom-built modules for the experiments.

Additionally, too, the use of such a pipeline can result in models being trained more quickly, models that are more robust, make better use of costly GPU resources, and more readily provide reproducible conclusions.

This chapter has described the design, construction, configuration, and use of Deeplodocus. It is now possible to use it to train a CNN to detect, recognize,

and identify ships contained in complex LW infrared imagery—specifically, IRShips, the largest and most diverse dataset of military ships currently contained in the academic literature.

Chapter 5

5. Using IRShips to train a CNN

This chapter details how IRShips was used to train the YOLOv3 general object detection algorithm to detect military ships in real-world LW infrared imagery. This body of work led to a peer-reviewed publication, which was presented at the SPIE Security and Defence conference 2020, and is believed to represent the first time that synthetic data was used to train a convolutional neural network to successfully detect military ships in real-world LW infrared imagery.

5.1. Introduction

IRShips is the largest and most diverse dataset of military ships currently contained in the academic literature and was designed specifically for training CNN-based object detection algorithms. This chapter details experiments which were undertaken to assess its performance when deployed to train a CNN to detect, recognize, and identify ships in complex real-world LW infrared imagery.

To achieve this, LW infrared images of a military ship were collected and manually labelled in order to serve as a real-world validation dataset. Next, the fully-convolutional YOLOv3 object detection algorithm was trained using IRShips, and its performance was validated using this real-world data. This training process was repeated several times under different conditions in order to answer the following three questions:

1. How effective is the synthetic IRShips dataset in training a CNN to detect ships in real-world images?
2. What are its limitations?
3. How can its effectiveness be improved?

5.2. Collecting LW Infrared validation data

HMNB Devonport is the largest naval base in Western Europe, and the home base of half the Royal Navy's frigates. Immediately to the south of the base, the approach channel narrows to a 500-metre strip of water between two headlands, from which ships entering and leaving the base can be clearly observed (50.358870849412675°N, -4.1718390316861935°E). A sequence of real-world validation data was duly captured from this point, using a Tau® 2 infrared camera sensitive in the 8 – 14 µm waveband, resulting in 940 frames, each sized 640 × 384 pixels, of a Karel Doorman-class frigate of the Belgian Navy as it navigated through the harbour and out to open water.

Significantly, this sequence of images is believed to so far represent the most complex and challenging test data reported in the academic literature for the evaluation of an infrared anti-ship seeker algorithm. In terms of size and complexity, the next largest and most complex test dataset is that of Bai *et al*. [249], which provides just 200 LW infrared images, none of which, as published, contain objects of background clutter contiguous to the pixels of the target vessel.

Pictured in Figure 5-1, this sequence of frames would present particularly challenging conditions to an infrared missile seeker, due to the large variety and high density of clutter it contained. This included a rocky shoreline, a navigation buoy, sea-surface reflections in the foreground, a developed coastline, and clouded sky in the background. Furthermore, the IRShips training data does not contain Karel Doorman-class frigates, preventing the trained algorithm from being able to identify this ship. However, with three different examples of frigates depicted in the training data, the trained algorithm could still be expected to recognize the Karel Dorman-class as a military vessel.

Figure 5-1: *An LW infrared image from the sequence of validation data depicting the Karel Dorman-class frigate in a densely cluttered scene. (Image brightness and contrast have been edited for illustrative purposes).*

5.2.1.1. Annotation

Before this sequence could be used to evaluate algorithm performance, it first needed to be annotated with a ship bounding box, ship type, and ship class label for each frame. This was achieved by manually labelling every 10th frame and, since the vessel's progression through the scene was slow and approximately linear, values for all remaining labels were calculated using linear interpolation. The ship type was labelled with a '1', as per the encodings defined in Table 3-5, and ship class was labelled with a '-1', to denote the class is not present in the training data. This resulted in a CSV file with 6 columns—for bounding box, ship type, and ship class labels—and 940 rows—one for each image.

5.2.1.2. Calibration

In addition to being annotated, this validation sequence also needed to be calibrated to approximately match the camera sensitivity—the relationship between incident infrared radiation intensity and pixel intensity—of the training data. The pixel values of images in the IRShips dataset are stored as 8-bit integers—meaning they range in value from zero to 255—and map to a temperature range of 0 to 100 °C. However, pixel values in the real-world validation sequence were recorded as 16-bit integers—which range from 0 to 65,535—and, though camera sensitivity was stated in the product specifications, the true relationship between its infrared intensity and pixel intensity was unknown.

Consequently, the sensitivity of the infrared camera needed to be calculated empirically using a black-body. In a controlled environment the black-body was heated to a temperature of 26 °C and its LW infrared emission recorded by the Tau® 2 camera for a period of 120 seconds. This process was then repeated for temperatures of 30, 60, and 80 °C, and the mean pixel response to the black-body at each temperature calculated. Finally, camera sensitivity was parameterised using linear regression to give a gradient of 57.429 and an intercept of 1960.9.

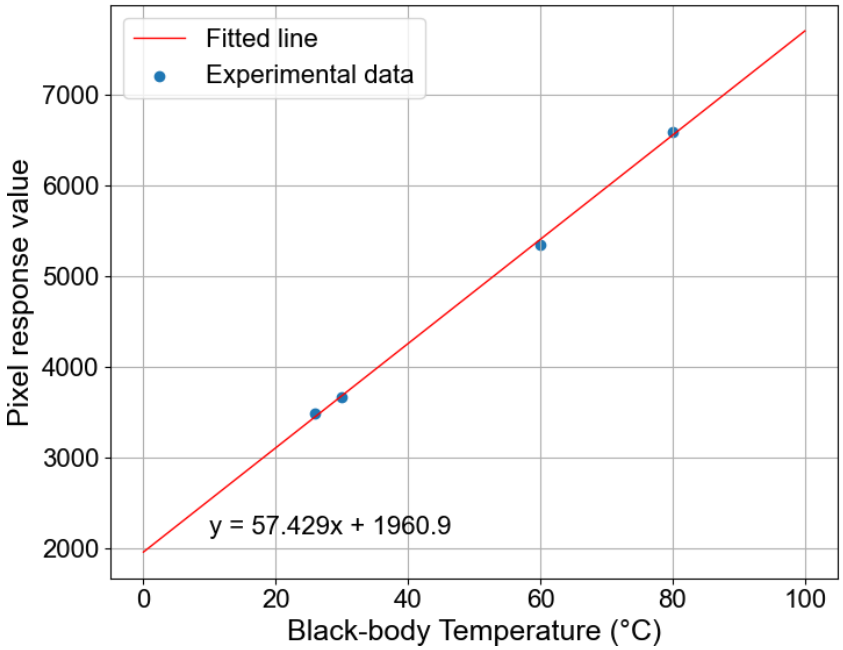


Figure 5-2: *A graph of the camera's measured response to different temperatures so the validation sequence could be calibrated to have the same sensitivity as the training data.*

5.3. Experimental setup

With the collection, annotation, and calibration of the validation sequence now complete, a software pipeline was developed in order to train a CNN-based general object detection algorithm using the IRShips dataset. Illustrated in Figure 5-3, this pipeline consisted of multiple software modules, each configured to the required experimental setup using the Deeplodocus deep learning development environment. This experimental setup operated as follows.

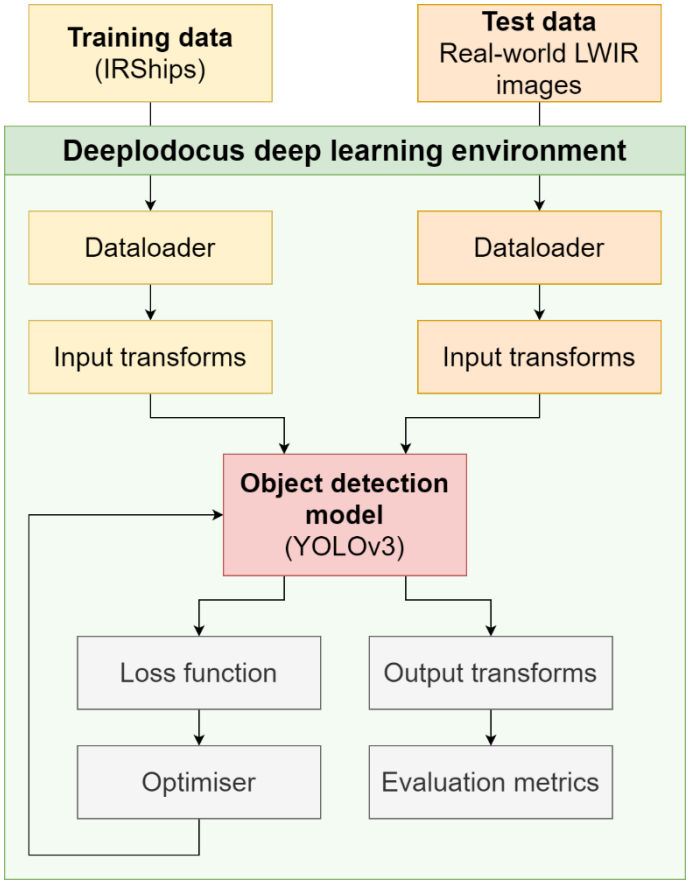


Figure 5-3: *Overview of the software pipeline which was used to train and test the YOLOv3 algorithm using the IRShips dataset and a sequence of real-world LWIR imagery.*

During training, batches of example images were loaded from the IRShips dataset by a **data-loader**, augmented by a sequence of **input transforms**, and then inputted to the **object detection algorithm**. After performing a forward pass on the batch of images, this algorithm outputted a series of object bounding box, type, and class predictions for each image, which were then compared with the corresponding ground truth labels by a **loss function**. This loss function subsequently informed the **optimiser** which, in a backward pass, then adjusted the internal weights of the object detector accordingly. The algorithm outputs were also processed by a series of **output transforms** before being evaluated by a selection of **metrics** which provided recognizable measures of algorithm performance.

During validation, on the other hand, batches of images from the real-world infrared sequence were loaded by an alternative **data-loader** and subsequently calibrated using a different set of **input transforms**. These images were then inputted into the **object detection algorithm**, and **losses**, **output transforms**, and **metrics**, were calculated from the outputs as before. In this case, however, the resultant loss values were not passed to the optimiser and a backward pass was not performed.

5.3.1. Algorithm selection

Instead of designing a CNN from scratch, it was decided that a pre-existing general object detector should be selected and adapted, as a large number of CNNs already exist, including the Faster R-CNN, RetinaNet, SSD, DSSD, and R-FCN [142, 250, 143, 141, 127] algorithms, each of which is capable of achieving high accuracy scores with respect to the COCO general object detection benchmark [139].

In addition to detection accuracy, it is also important to consider the speed of the algorithm, since real-time inference is a critical requirement of a missile seeker algorithm. The architecture of modern object detection algorithms can be categorised as either one-stage, or a two-stage. One-stage algorithms regress the object bounding box and object class predictions directly, in a single forward pass, whereas two-stage algorithms use a Region Proposal Network to extract multiple regions of interest, each of which is then refined by a second network. Consequently, two-stage algorithms typically deliver superior accuracy, but can be significantly slower relative to their single-stage counterparts.

The speed and detection accuracy—as measured using mean average precision—for a selection of high-performance general object detection algorithms was compared, and is shown in Figure 5-4. This shows that, at the time of writing, YOLOv3 [146] is the fastest object detection algorithm that is capable of also delivering a competitive level of accuracy.

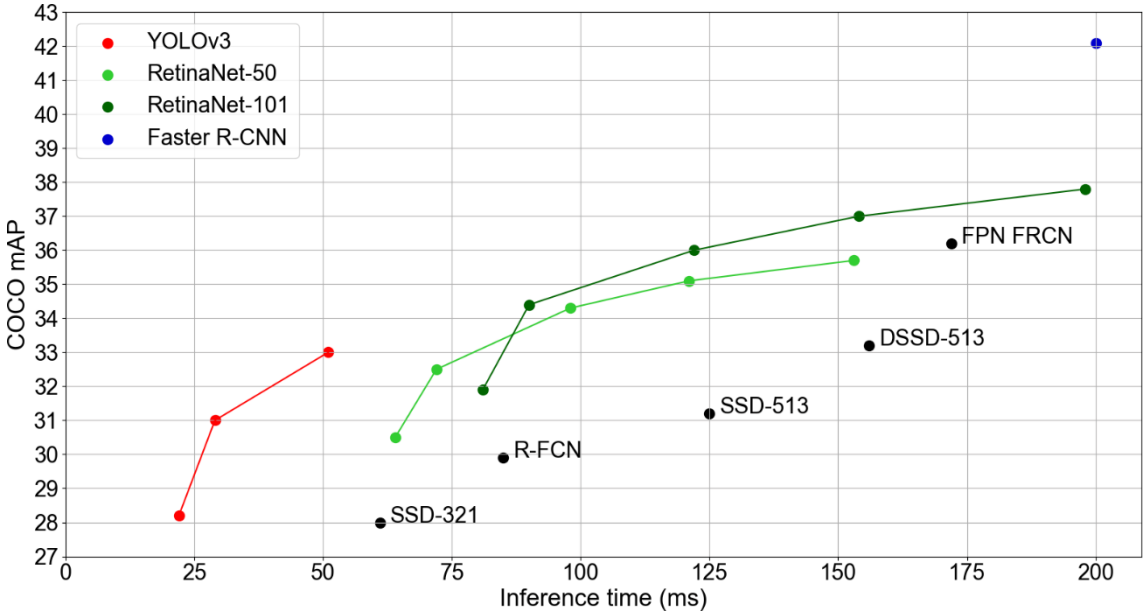


Figure 5-4: *The speed and detection performance of current CNN-based object detection algorithms, as evaluated with respect to the COCO benchmark dataset using mean average precision. (Data sourced from [142, 250, 143, 141, 146, 127].)*

In addition to its speed and accuracy, YOLOv3 also possessed a demonstrated ability to generalise into the real-world domain from non-real world training data. This was because YOLO [107], a predecessor on which YOLOv3 is based, had previously demonstrated a superior performance in the task of recognising artistic representations of people contained in abstract art [251, 252], despite being trained using the real-world VOC 2007 general object detection dataset [179], as shown in Table 5-1. Due to the similarity of their architectures, it was decided that this ability to generalise across domains was likely to persist in the YOLOv3 algorithm, and could help to ensure good performance in the real-world domain, despite training with only synthetic examples.

Due to this combination of speed, accuracy, and generalisability, YOLOv3 was therefore selected as the detection algorithm of choice throughout the experiments in this chapter.

Table 5-1: *Algorithm test performance on the VOC 2007, Picasso, and People-Art datasets. (Data sourced from [107].)*

Algorithm	VOC 2007 [179]	Picasso [252]		People-Art [251]
	AP	AP	Best F1	AP
YOLO [107]	0.592	0.533	0.590	0.45
R-CNN [112]	0.542	0.104	0.226	0.26
DPM [253]	0.432	0.378	0.458	0.32
Poselets [254]	0.365	0.178	0.271	-
D&T [255]	-	1.019	0.051	-

5.3.2. The YOLOv3 object detection algorithm

The YOLOv3 algorithm has a one-stage architecture, meaning that the object bounding box and object class predictions are regressed directly in a single forward pass. Illustrated in Figure 5-5, YOLOv3 is also fully-convolutional network—meaning that all its 'learned layers' are convolutional—which consists of an encoder, or 'backbone' feature detector, followed by a Pyramid Feature Network-style [250] decoder which generates predictions at three different scales.

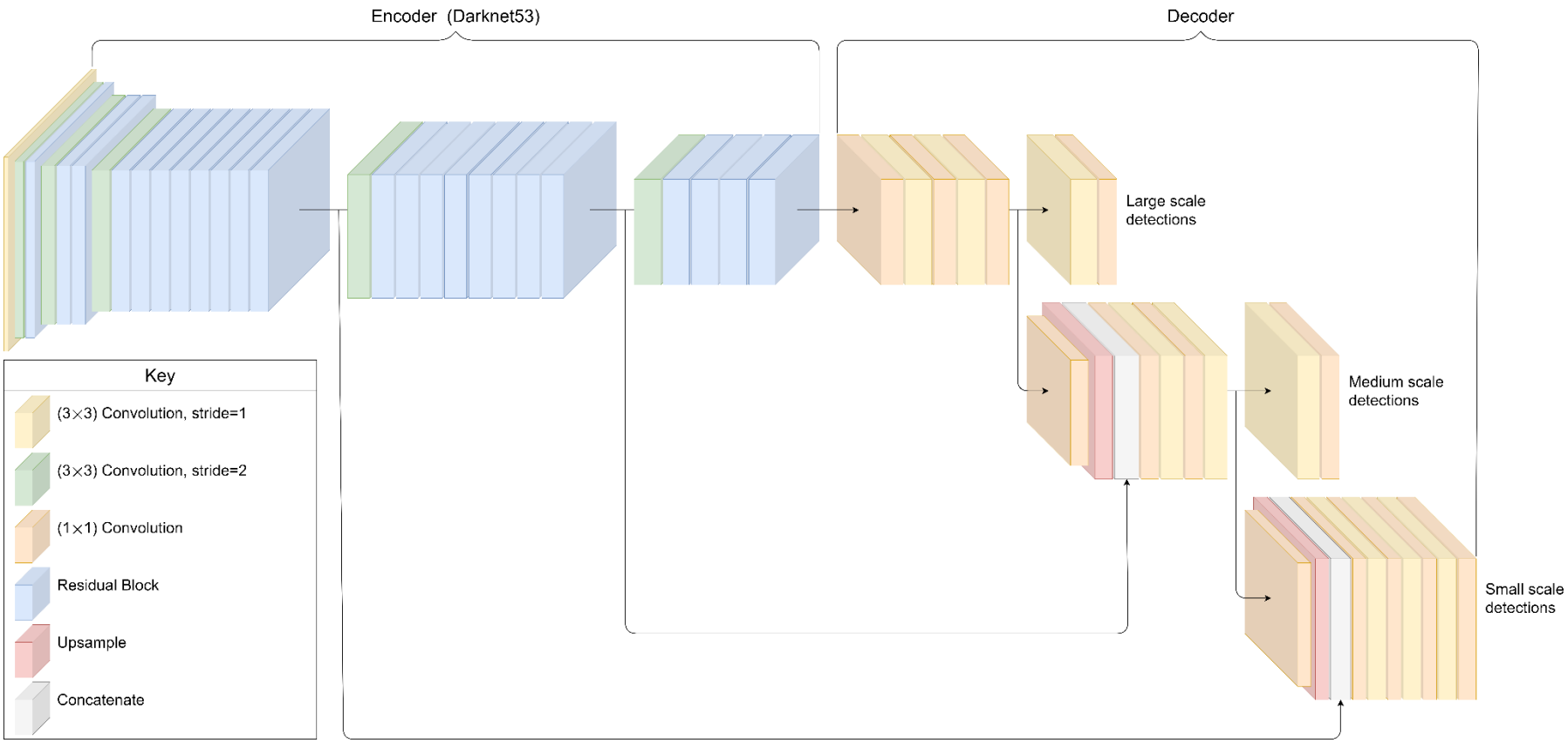


Figure 5-5: *Illustration of YOLOv3, a fully-convolutional neural network, which uses a feature pyramid network-style architecture [250] to perform object detection across three different scales.*

The YOLOv3 **encoder**, known as Darknet-53, is a form of Residual Network, meaning that it is constructed from a series of Residual Blocks [134]. As illustrated in Figure 5-6, a residual block is a sequence of convolutional layers where the input of the first layer is added to the output of the final layer by means of a 'skip-connection'. As a result, if the sequence of layers is to perform some overall function $H(x)$, which is equal to $x + F(x)$, then the constituent convolutional layers are only required to learn the residual $F(x)$, hence the name 'residual blocks'.

These blocks possess the important property of being able to easily perform the identity function. Achieved by simply setting $F(x)$ to zero, this is beneficial to the performance of CNNs, since it enables the block to refine features from the previous block without forcing the algorithm to generate a new level of abstraction. While constructing a hierarchy of abstract concepts is crucial to the operation of CNNs, just a few levels of abstraction are often sufficient, and extracting hundreds of levels is often excessive. Consequently, perfecting an existing hierarchy can be just as important as adding a new level of abstraction.

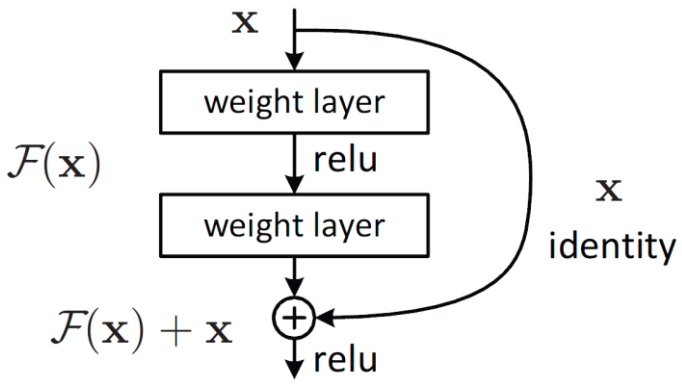


Figure 5-6: *Illustration of a residual block, where relu—which stands for Rectified Linear Unit [129]—is the activation function. (Image sourced from [134].)*

Within each residual block, YOLOv3 incorporates a convolutional layer of 1 × 1 sized kernels. These layers—which are sometimes called bottleneck layers—'learn' the relationships between feature maps in order to encode them into a reduced parameter space while preserving as much useful information as possible. YOLOv3 uses these layers to halve the number of feature maps

inputted to the residual block, before processing by a subsequent convolutional layer with 3 × 3 kernels, in order to reduce the number of matrix multiplications required and maximise inference speed.

In addition, batch normalisation layers [137] are applied between convolutional layers, to standardise the activation values in each batch of data according to their mean and standard deviation. This process is described by Equation 5.1, where Z is an activation vector, μ and σ are the mean and standard deviation of the activation values across the batch, and ε is a constant used for numerical stability. The output of the batch normalisation layer is calculated, as per Equation 5.2, where γ and β are trainable parameters learned through gradient descent that allow the algorithm to choose optimal distributions for the activation values.

$$Z_{norm} = \frac{Z - \mu}{\sqrt{\sigma^2 - \varepsilon}} \tag{5.1}$$

$$\check{Z} = \gamma \times Z_{norm} + \beta \tag{5.2}$$

Batch normalisation improves algorithm performance by reducing the interdependency between layers, and also algorithm training by re-parametrising the underlying optimisation problem, making it smoother and more numerically stable [256]. Moreover, since the resultant activation values are dependent on the statistics of each batch, batch normalisation also introduces additional noise into the training process which has a regularisation effect that helps to avoid overfitting.

Between sequences of residual blocks, Darknet-53 uses convolutional layers of 3 × 3 kernels which are applied with a stride of 2—i.e., at every second position in the given input—in order to down-sample the data. In total, YOLOv3 contains five of these down-sampling layers, each of which add to the hierarchy of abstraction by initiating features at a new and increased scale.

After passing through Darknet-53, the data is subsequently processed by the **decoder**, which consists of further convolutional layers, and two layers of nearest neighbour upscaling. This decoder terminates at three output layers,

which due to the different levels of upscaling, correspond to three different scales. As can be seen in Figure 5-5, the medium- and small-scale portions of the decoder use two skip-connections from the encoder to carry forward useful, partially-encoded information from pervious layers, which is then incorporated using a bottleneck layer.

The three terminal convolutional layers each contain 255 1 × 1 kernels, meaning that they produce 255 outputs for each x, y position in the given data. These 255 outputs correspond to 3 sets of object predictions which are each parameterised by 85 values—namely, 80 class predictions, 4 bounding box parameters, and 1 'objectness' score—as illustrated in Figure 5-7. Each of these sets correspond to a different anchor, $(p_w^a,\ p_h^a)$, which serves as a prior box, whose position, height, and width is refined by the box parameters, $\hat{b}_x^a$, $\hat{b}_y^a$, $\hat{b}_w^a$, and $\hat{b}_h^a$.

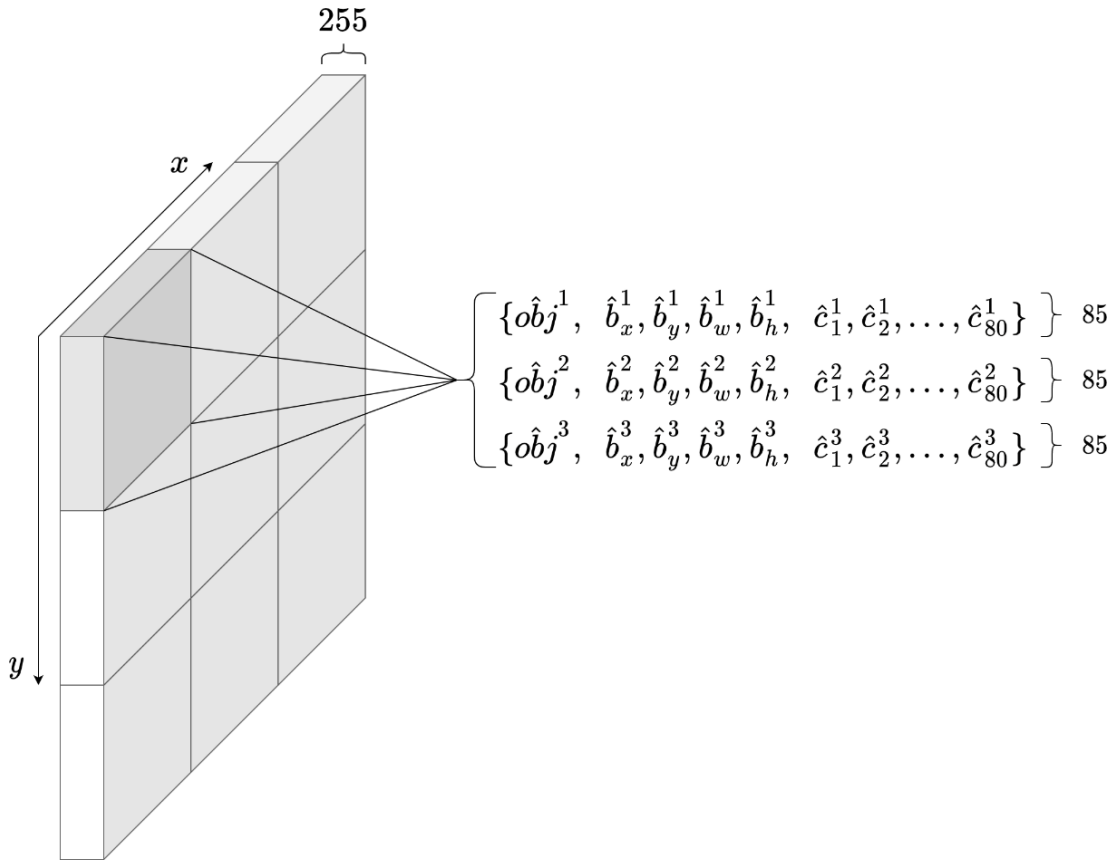


Figure 5-7: *Illustration of a final convolutional layer from the YOLOv3 architecture, which outputs 3 sets of 85 predictions at every x, y position in the given data.*

As illustrated in Figure 5-8, the position of an anchor originates at the centre of each cell in the given data. If, for any cell, there is sufficient overlap between an anchor and a foreground object, the position of the anchor within the cell will be refined as per Equations 5.1 and 5.2, its shape will be refined according to Equations 5.3 and 5.4, and the object's presence will be indicated in the objectness score, $\widehat{obj}^a$. This objectness score is bounded between 0 and 1 by a sigmoid activation function, as are class predictions, $\{\hat{c}_1^a . \hat{c}_2^a, \dots, \hat{c}_{80}^a\}$, which correspond to the 80 labelled objects in the COCO object detection dataset. Subsequently, the predicted class of the foreground is taken as simply the maximum argument of all class predictions.

$$\hat{b}_x^a := \sigma\left(\hat{b}_x^a\right) + C_x \tag{5.1}$$

$$\hat{b}_y^a := \sigma\left(\hat{b}_y^a\right) + C_y \tag{5.2}$$

$$\hat{b}_w^a := p_w^a \cdot \exp\left(\hat{b}_w^a\right) \tag{5.3}$$

$$\hat{b}_h^a := p_h^a \cdot \exp\left(\hat{b}_h^a\right) \tag{5.4}$$

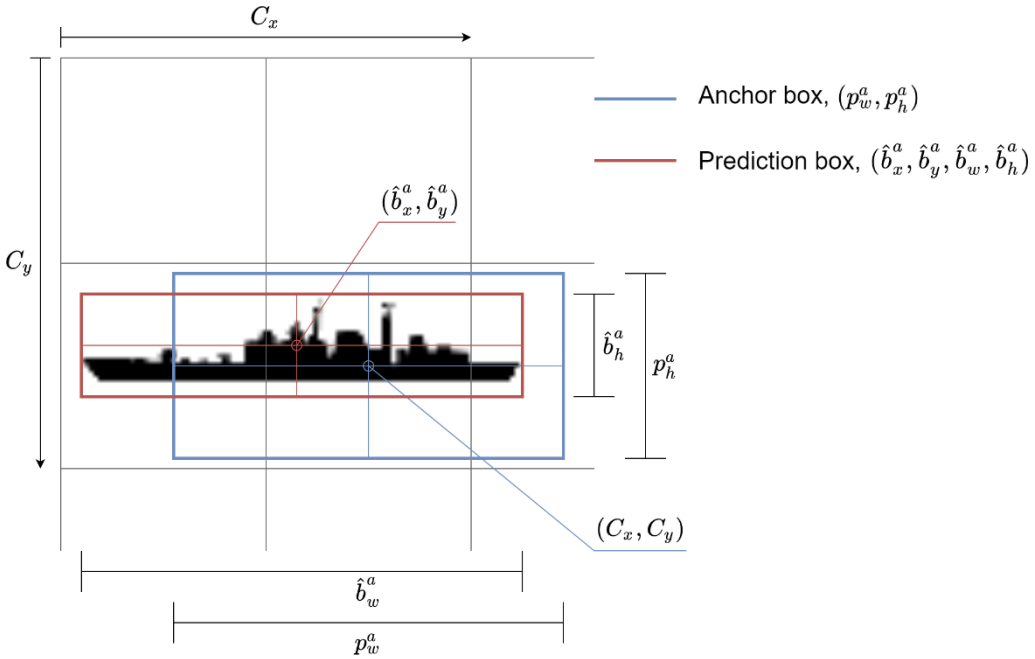


Figure 5-8: *In the final layer of YOLOv3, an anchor (also called a prior box) is transformed via Equations 5.1–5.4 and the activations $\hat{b}_x$, $\hat{b}_y$, $\hat{b}_w$, and $\hat{b}_h$, in order to generate bounding box predictions.*

5.3.2.1. Implementing YOLOv3

The YOLOv3 architecture was implemented as a Python3 Class object constructor, using the PyTorch open-source machine learning library which enables the use of GPU hardware. All aspects of the algorithm were kept true to the original documentation, [107, 145, 146], except for the number of possible class predictions, which were reduced to 14 to correspond with the 4 types and 10 classes of vessel represented in the IRShips dataset.

This implementation of YOLOv3 was included as a module within a new Deeplodocus project using the `model.yaml` configuration file, shown in Figure 5-9. This file specified the name and source module of the YOLOv3 implementation, as well as three keyword arguments to set the number of possible type predictions, the number of possible class predictions, and the backbone encoder. Darknet53 was specified as the encoding architecture, although this implementation of YOLOv3 is modular and could therefore be initialised with any compatible encoder, such as Darknet19 or a ResNet variant. Further keyword arguments for the encoder were used to specify the number of input channels and specify that its full-connected layer—which performs image classification—should be omitted.

model.yaml

```yaml
name: YOLOv3                    # Name of the Python class
module: modules.models.yolo     # Path to Python module
epoch: 0                        # Current training epoch
from_file: False                # Whether to load internal parameters from a file
file: Null                      # Path to model internal parameters file
input_size: Null                # Model input size
kwargs:                         # Keyword arguments to pass to YOLOv3
  num_types: 4                    # Number of possible object type predictions
  num_classes: 10                 # Number of possible object class predictions
  backbone:                       # Specify the backbone encoder
    name: Darknet53                 # Use Darknet53
    module: modules.models.darknet  # Path to the Darknet53 module
    kwargs:                         # Keyword arguments to pass to Darknet53
      include_top: False              # Whether to include to fully-connected classification layer
      num_channels: 3                 # Number of input channels
```

Figure 5-9: *Configuration of the YOLOv3 implementation into a Deeplodocus project.*

5.3.3. Training data

During the training of YOLOv3, batches of training examples needed to be continuously loaded, transformed, and inputted to the YOLOv3 algorithm. As shown in Figure 5-10, this process was achieved via two data-loaders and two sequences of input transforms, and to maximise computational efficiency, it was executed across multiple CPU cores using multi-processing.

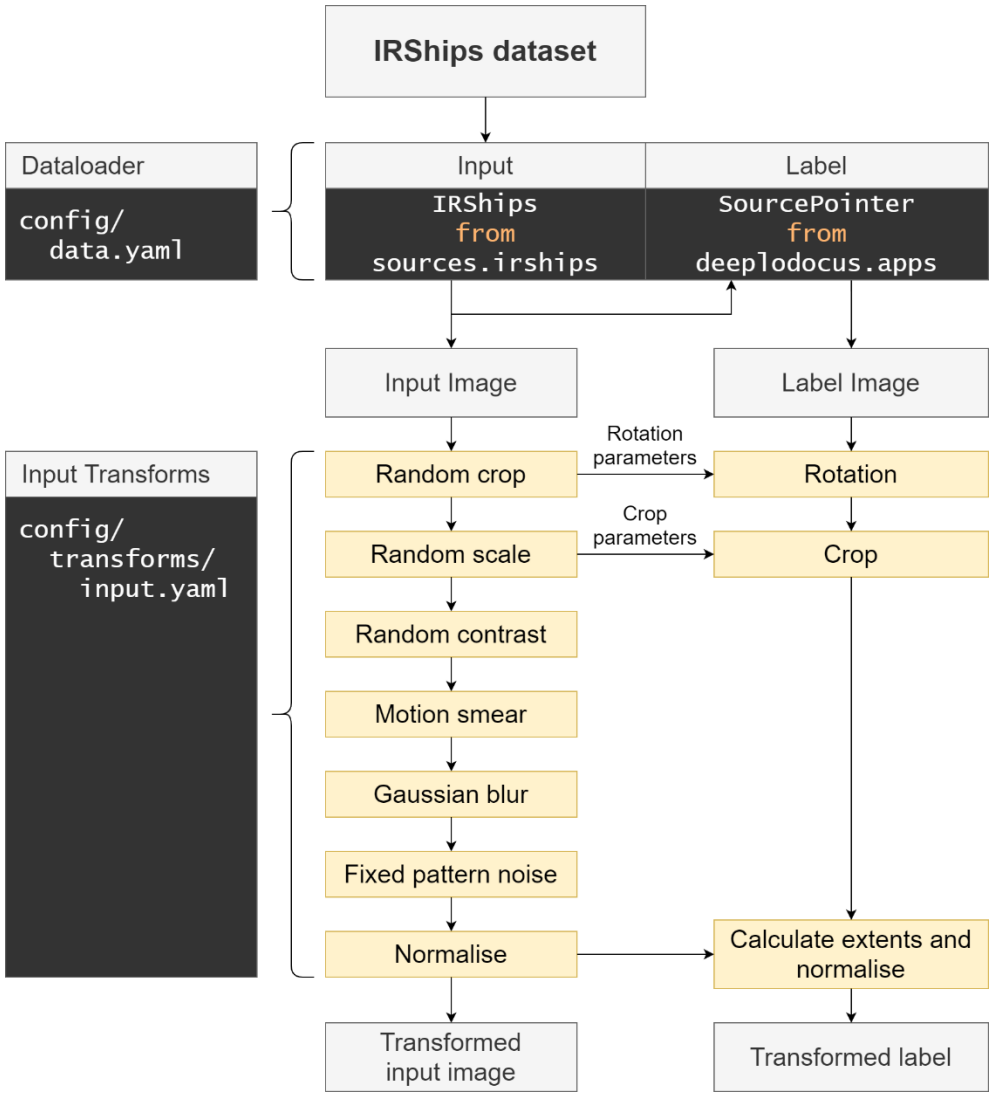


Figure 5-10: *Overview of the data loading and processing pipeline used during algorithm training.*

5.3.3.1. Data-loaders

To load training data from IRShips, Deeplodocus required two data-loaders—one loader for the input images, and a second loader for the corresponding labels. For the first of these, IRShips's native data-loader module was configured into the deep learning pipeline using the `data.yaml` configuration file. This module, which was presented in Section 3.2.5.4, continuously outputted synthetic LW infrared images—which were augmented with various different sea-states, sky-states, and background clutter—and their corresponding labels in the form of input-label pairs. Each paired input and label were formatted as an `IRShipsInstance` with the input instance containing the synthetic example image, and the label instance containing the corresponding binary label image, as illustrated in Figure 5-11. Both the input and label also contained relevant metadata.

Figure 5-11: *An example of an augmented synthetic input image (left) and its corresponding binary label image (right).*

Next, in order for the second instance in each input-label pair to be treated as the label, rather than an input, a `SourcePointer` was configured as the second data-loader. Native to Deeplodocus, the `SourcePointer` is capable of 'pointing' to data which has been loaded by other data-loaders, and as such, was configured to point to the second item of the first data-loader.

5.3.3.2. Geometric transforms

With suitable data-loaders in place for both the input images and their corresponding labels, a series of geometric transforms were then incorporated

into the pipeline to increase variation within the training examples by means of **random cropping**, **random rotations**, and **random scaling**.

To ensure that each ground truth label remained accurate, each transform was applied with the same randomly selected parameters to both the input image and the label image. For the random rotation transform, for example, a rotation angle was selected from the distribution $U(-5, 5)$ and applied to the input image, which was then returned along with this randomly selected rotation parameter. This parameter was subsequently passed to a deterministic rotation transform, which applied the same degree of rotation to the label image, thus ensuring that it remained accurate.

This mirroring of input and label transforms was similarly applied to the subsequent random crop and random scaling transforms. The random crop function randomly selected a patch of between 56% and 100% of the original image size, and the random scaling function resizing each image to a resolution selected from the sequence $\{(a_1 \times 64, a_1 \times 32), (a_2 \times 64, a_2 \times 32), \ldots, (a_{13} \times 64, a_{13} \times 32)\}$ where $a = \{9, 10, \ldots, 16\}$.

The resizing transform was seeded such that resolution changes occurred with every fourth batch of images. The effect of these geometric transforms is illustrated in Figure 5-12.

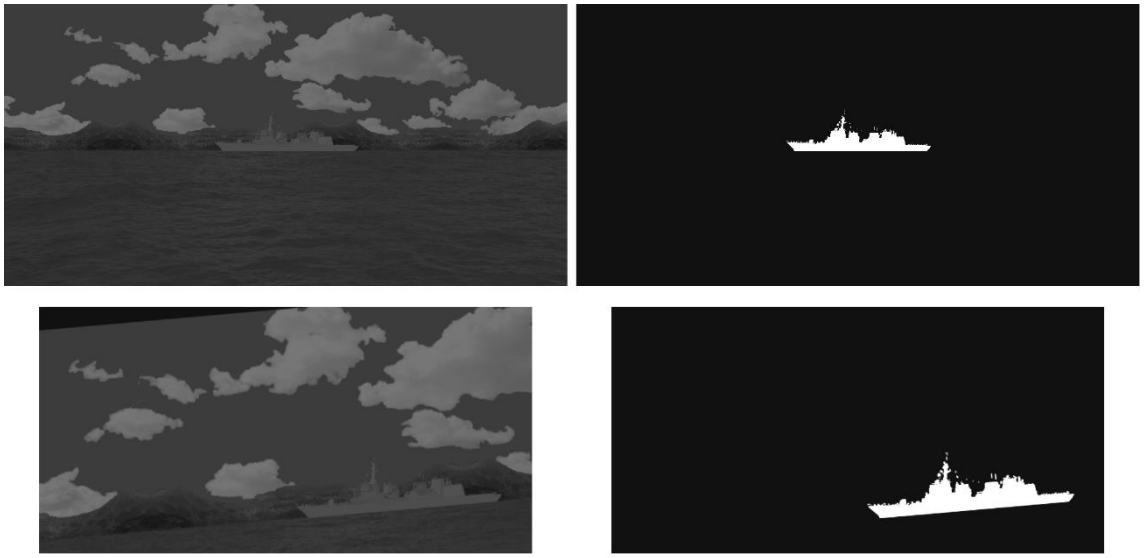


Figure 5-12: *An example of an image-label pair before (top) and after (bottom) the application of geometric transforms.*

5.3.3.3. Sensor noise

After all the geometric transforms were complete, three forms of sensor noise were applied to each input image. These were **motion blur**, **Gaussian blur**, and **fixed-pattern noise**.

Motion blur is the apparent streaking which can be observed in the photography of moving objects, and was simulated in the synthetic images using an algorithm supplied by MBDA, an implementation of which is provided in Appendix A-5.

Next, **Gaussian blur**, also known as image smoothing, was applied using the process specified in Equation 5.6, where x and y are the location indices and σ was selected at random from the set $\{1, 3, 5, \dots, 13\}$.

$$G_{2D}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5.6}$$

Finally, vertical **fixed pattern noise** was also applied to each input image, using a further algorithm supplied by MBDA, and provided in Appendix A-6.

The combined effect of motion blur, Gaussian blur, and fixed pattern noise is shown in Figure 5-13.

Figure 5-13: *Example of an input image before (left) and after (right), the application of simulated sensor noise.*

5.3.3.4. Formatting

After the application of geometric transforms and sensor noise, each input image was normalised between 0 and 1 by dividing its pixel values by 255, and the label formatted into a (1×6) array containing the four bounding box parameters, ship class label, and ship type label. These bounding box parameters were calculated from the target extents in the binary label image, and were then normalised with respect to the height and width of the input image.

5.3.4. Loss function

Optimisation of the internal parameters of YOLOv3 was controlled by the loss function, defined in Equation 5.7. Though a function for optimising a YOLO architecture was first presented by Redmon *et al*. [107], the loss function implemented here is an updated version which incorporates an additional level of object classification as well as a range of performance-enhancing alterations that have been identified in recent literature [257, 146].

$$\mathcal{L} = \sum_{s}^{S} \sum_{i}^{X_s} \sum_{j}^{Y_s} \sum_{a}^{anchors} \left(\lambda_{obj} BCE\left(\mathbb{1}_{sija}^{obj}, \widehat{OBJ}_{sija}\right) + \lambda_{box} \mathbb{1}_{sija}^{obj} \left(1 - GIoU\left(B_{sija}, \hat{B}_{sija}\right)\right) + \lambda_{cls} \sum_{t}^{types} \mathbb{1}_{sij}^{obj} BCE\left(T_{sij}^{t}, \hat{T}_{sij}^{t}\right) + \lambda_{cls} \sum_{k}^{classes} \mathbb{1}_{sij}^{obj} BCE\left(C_{sij}^{k}, \hat{C}_{sij}^{k}\right) \right) \tag{5.7}$$

where:

s is an index used to denote the output scale of the current prediction,

i and j are indexes to denote cell position in the x and y direction,

a denotes the index of the prior anchor box,

λ_{obj} is a weighting given to the objectness loss,

λ_{box} is a weighting given to the bounding box loss,

λ_{cls} is a weighting given to the recognition and identification losses,

$\mathbb{1}_{sij}^{obj}$ denotes whether the centre of the target object is within cell i , j at scale s,

$\mathbb{1}_{sija}^{obj}$ is denotes whether the target object matches the a^{th} anchor box within cell i , j at scale s,

$\widehat{OBJ}_{sija}$ is the predicted probability that there is a foreground object which corresponds to the a^{th} anchor box within cell i , j at scale s,

BCE is the binary cross-entropy function, see equation 5.10,

B_{sija} is the ground truth which corresponds to the a^{th} prior box at cell i,j and scale s,

$\hat{B}_{sija}$ is the bounding box prediction which corresponds to the a^{th} prior box at cell i,j and scale s,

$GIoU$ is the Generalise Intersection over Union, see Equation 5.9,

T_{sij} is the ground truth for the foreground object type at cell i,j and scale s,

$\hat{T}_{sij}$ is the predicted type for the foreground object type at cell i,j and scale s,

C_{sij} is the ground truth for the foreground object class at cell i,j and scale s, and

$\hat{C}_{sij}$ is the predicted class for the foreground object at cell i,j and scale s.

This function was designed to quantify the error of the YOLOv3 algorithm outputs for each batch of labelled example images and therefore enables the optimiser to effectively update the algorithm's internal parameters. To do this, the function consists of four terms respectively relating to objectness scores, bounding box localisations, object type predictions, and object class predictions.

Since the YOLOv3 algorithm outputs three grids of predictions—which each correspond to a different scale—and each cell within a grid contains prediction data for three different anchor boxes, this means that each term of the loss function is accumulated over each anchor box, at each grid position, and at each of the three different scales.

The first term of the loss function is responsible for optimising the YOLOv3 algorithm to correctly determine the presence, or absence, of a foreground object within each anchor box of every grid cell.

The second term is responsible for optimising the algorithm's bounding box predictions, which was achieved by using the Generalised Intersection over Union (GIoU) metric. Originally, YOLO bounding box predictions were optimised according to $(x-\hat{x})^2+(y-\hat{y})^2+\left(\sqrt{w}-\sqrt{\hat{w}}\right)^2+\left(\sqrt{h}-\sqrt{\hat{h}}\right)^2$ where x, y, w, and h denote the position and size of the box. However, this approach is not scale-invariant, which meant that bounding box predictions for small ground truth boxes generated only a small loss signal relative to larger box predictions, even if the larger prediction boxes matched well with their respective ground truths. This in turn acted to reduce the YOLO algorithm's ability to accurately determine the location of small objects.

Consequently, it is important that loss relating to bounding box predictions are calculated using a scale-invariant method, such as the Intersection over Union (IoU) metric. Also known as the Jaccard Index, this is illustrated in Figure 5-14 and expressed in Equation 5.8. However, the IoU value plateaus at zero when there is no overlap between the prediction and ground truth, meaning that in such cases, it will not provide any information which can be used to improve the algorithm's internal parameters.

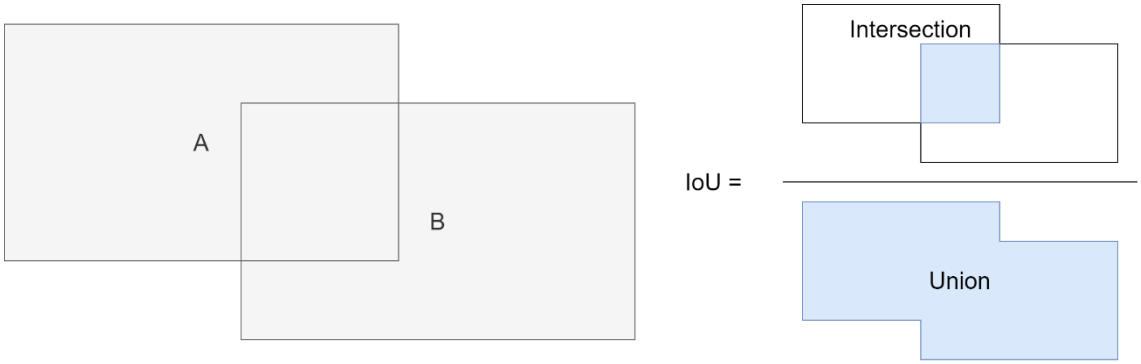


Figure 5-14: *Illustration of the Intersection over Union statistic (also known as the Jaccard Index) for evaluating the similarity between two bounding boxes.*

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.8}$$

Therefore, a modified IoU metric known as the Generalised Intersection over Union (GIoU) [258] was used instead. Expressed in Equation 5.9, where C is the smallest rectangle which enclose both A and B, GIoU provides useful feedback even when prediction and ground truth boxes do not overlap and, as a result, is more effective than IoU for evaluating bounding box-related loss.

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|} \tag{5.9}$$

The final two terms of the loss function are responsible for optimising the algorithm's type and class predictions respectively. This was done using the Binary Cross Entropy loss function, as shown in Equation 5.10.

$$BCE(\mathrm{y}, \hat{y}) = \begin{cases} -\mathrm{y} \cdot \log(\hat{y}) & if\ y = 1 \\ (\mathrm{y}-1) \cdot \log(1-\hat{y}) & if\ y = 0 \end{cases} \tag{5.10}$$

This loss function was again implemented as a Python Class—named YOLOv3Loss—and included in the Deeplodocus project using the `losses.yaml` configuration file, as shown in Figure 5-15.

losses.yaml

```yaml
DetectionLoss:              # Arbitary name
  name: YOLOv3Loss            # Name of the Python Class
  module: modules.losses      # Path to the Python module
  weight: 1                   # Loss weight
  kwargs:                     # Loss keyword arguments
    obj: 64.3                   # Weighting for objectness score loss
    box: 3.54                   # Weighting for bounding box loss
    cls: 0.2                    # Weighting for recognition and identification
```

Figure 5-15: *Configuration of the YOLOv3Loss class within the experimental setup.*

5.3.5. Optimiser

After the evaluation of loss, an optimiser was used to update the internal parameters of the YOLOv3 algorithm in a backward pass. Originally, neural networks were optimised by means of Stochastic Gradient Descent (SGD), which maintains a constant learning rate for all of the model's internal parameters throughout the training process. However, several extensions to this algorithm have since been developed which are more effective, more stable, and more computationally efficient.

Foremost of these is the adaptive moment 'Adam' optimisation algorithm, which combines together the advantages of two other SGD extensions: Adaptive Gradient Algorithm (AdaGrad) [259] and Root Mean Square Propagation (RMSProp) [260]. AdaGrad maintains independent learning rates for each parameter of the neural networks, which enables an optimised state to be reached more quickly, enabling training to remain effective at smaller derivatives of the loss function. RMSProp also maintains per-parameter learning rates, but adapts them based on the average magnitudes of recent gradients, improving stability in non-stationary problems such as computer vision. Adam incorporates concepts from both methods, and uses an exponential moving average of the gradient and squared gradient to update the learning rate for each parameter. As a result, Adam is generally considered a more effective optimiser than alternatives such as SGD, AdaGrad, RMSProp, and AdaDelta [261].

An implementation of the Adam optimiser is available in the PyTorch machine learning library's `torch.optim` module. This module was incorporated into the Deeplodocus project using the `config/optimizer.yaml` file, as shown in Figure 5-16.

It should be noted that weight decay—an optimiser parameter which drives the neural network's internal parameters towards zero in order to reduce overfitting—is conventionally often applied to all learnable parameters in the network. However, it has been recommended by Jia *et al*. [262] that weight decay is applied only to the weights of convolutional and fully-connected layers, not the biases. Consequently, a weight decay of 5×10^{-4} was applied to all convolutional layer weights and a weight decay of zero was applied to all other learned parameter groups, as can be seen in Figure 5-16.

optimizer.yaml

```yaml
name: Adam                          # Name of the Python Class
module: torch.optim                 # Path to the module location
load_state_dict: False              # If any optimiser state dict in the model file should be loaded
kwargs:                             # Keyword arguments for the optimiser
  lr: 1e-6                            # Optimiser learning rate
param_groups:                       # L2 regularisation for the weights of covnolutional layers
  - condition: "lambda x: 'norm' in x and x.endswith('.weight')"
    kwargs:
      weight_decay: 0
  - condition: "lambda x: 'conv' in x and x.endswith('.weight')"
    kwargs:
      weight_decay: 0.0005
  - condition: "lambda x: x.endswith('.bias')"
    kwargs:
      weight_decay: 0
```

Figure 5-16: *Configuration of the Adam optimised from the PyTorch machine learning library with different parameter groups.*

5.3.6. Output data transforms

The YOLOv3 loss function is calculated directly from the outputs of the YOLOv3 model. However, in order to calculate metrics—which monitor algorithm performance—the YOLOv3 outputs must first be processed.

5.3.6.1. Concatenating and filtering

First, the YOLOv3 outputs at each of the three scales were collapsed from a 5D tensor of shape (b × h × w × a × 19)—denoting 19 prediction values for each

batch, grid cell, and anchor box—to a 2D tensor of shape (n × 19). These were then concatenated to form a single tensor of all algorithm predictions, where the first five positions in the second dimension corresponded to $\hat{b}_x$, $\hat{b}_y$, $\hat{b}_w$, $\hat{b}_h$, and $\widehat{obj}$ respectively; the following four positions corresponded to object ship-type predictions; and the final 10 positions corresponded to ship-class predictions. Next, algorithm predictions were filtered, and all predictions possessing an objectness score of less than $\tau_{obj} = 0.5$ were removed.

5.3.6.2. Non-maximum suppression

Next, any duplicate detections were removed. Most object detection algorithms, including YOLOv3, sometimes produce multiple bounding box predictions for a single object, as illustrated in Figure 5-17, which reduces the precision of the algorithm. As a result, duplicate bounding boxes must be removed, and this was achieved by using Non-Maximum Suppression (NMS), which operates as follows:

1. If P is a list of all output predictions, select the prediction, S, which has the greatest objectness score, and add it to the list of predictions to keep, K.
2. Calculate the IoU overlap of S with all other predictions in P.
3. Remove all predictions from P which overlap with S by an IoU which exceeds some threshold, τ_{iou}.
4. Repeat steps 1–3 until P is empty.

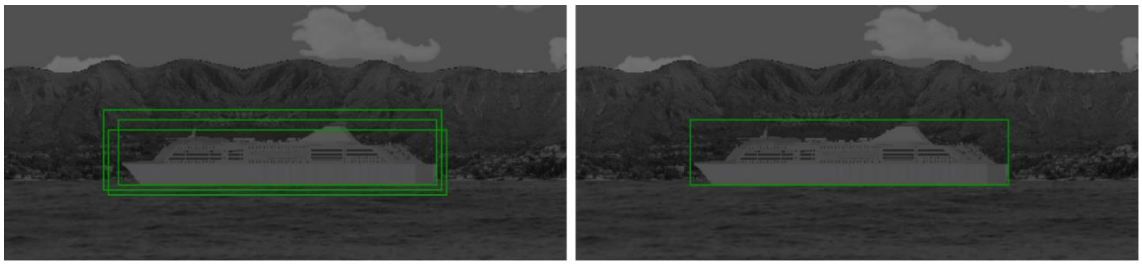


Figure 5-17: *Multiple detections for a single ship (left), and the effect of applying NMS (right).*

5.3.6.3. Visualisation

After NMS, all remaining output predictions were passed to a third output transform. This did not alter the algorithm's predictions, but instead visualised the predictions and the ground truth values, in order to provide visual feedback during the training and validation processes.

5.3.7. Metrics

In order to measure and compare the performance of object detection algorithms, six performance metrics—recall, precision, f-score, average IoU score, recognition accuracy, and identification accuracy—were incorporated into the deep learning pipeline.

5.3.7.1. Recall

Recall was used to measure the algorithm's ability to correctly detect vessels which were present in given imagery, and is calculated according to Equation 5.11. Throughout all the experiments, bounding box predictions were considered as correct if they overlapped a ground truth box with an IoU score of 0.5 of greater.

$$Recall = \frac{TP}{TP + FN} = \frac{No.correct\ detections}{No.targets} \tag{5.11}$$

In Equation 5.11, TP is the number of true positives—instances where a ship was correctly detected—and FN is the number of false negatives—instances where a ship was not correctly detected.

5.3.7.2. Precision

Precision is the proportion of the algorithm's predictions that are correct, as calculated according to Equation 5.12.

$$Precision = \frac{TP}{TP + FP} = \frac{No.correct\ detections}{No.predictions} \tag{5.12}$$

In Equation 5.12, TP is the number of true positives—instances where a ship was correctly detected—and FP is the number of false positives—algorithm predictions which do not match any actual ships.

5.3.7.3. F-score

The overall performance of a detection algorithm is dependent on its recall and its precision. However, there is an inverse relationship between these two metrics—if the objectness threshold is reduced, then recall will increase, but precision will fall. As a result, recall and precision must be considered together, and this can be done by calculating the F-score—the harmonic mean of recall and precision—as written in Equation 5.13.

$$F\text{-}score = 2 \times \frac{Recall \times Precision}{Recall + Precicion} \tag{5.13}$$

5.3.7.4. Average IoU

The average IoU between predicted boxes and actual ships was calculated to measure the algorithm's ability to determine the extent of the bounding boxes of the ships. In order to ensure that this localisation metric was not dependent on the algorithm's recall, it was decided that the average IoU calculation should only include bounding box predictions that were considered as correct detections.

5.3.7.5. Recognition and identification accuracy

Finally, the algorithm's ability to correctly recognize and identify ships was measured using recognition and identification accuracy respectively, defined as the ratio of correct ship classifications to correct detections.

5.1.1. Test data

During algorithm testing, images from the real-world infrared sequence and its corresponding label were loaded in turn by two data-loaders, and were then processed by two input transforms in order to calibrate and normalise them. This procedure is outlined below in Figure 5-18.

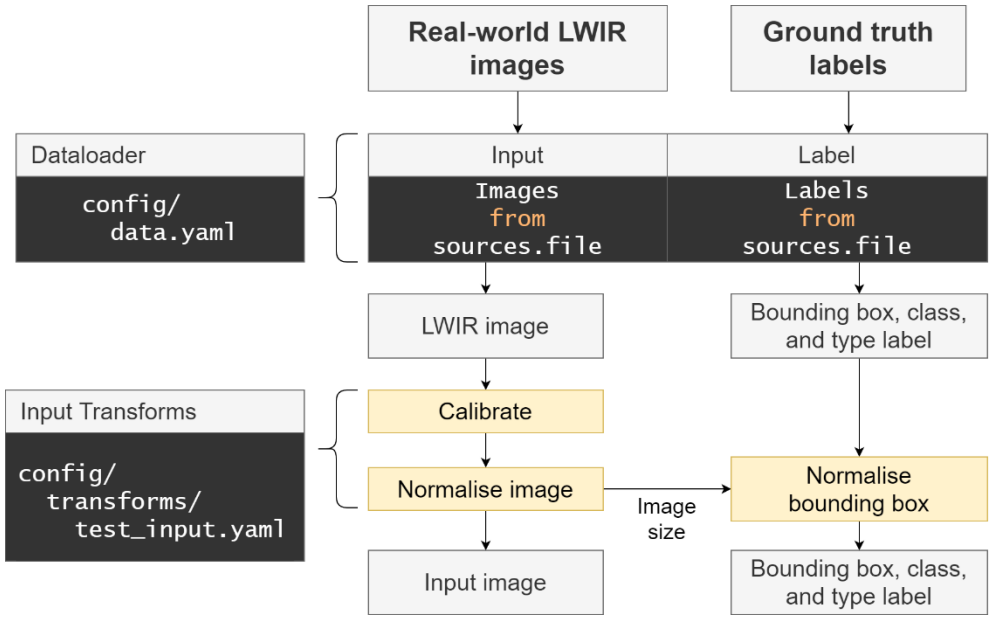


Figure 5-18: *Overview of the data loading and pre-processing pipeline used during algorithm testing.*

5.1.1.1. Data-loader

Input images were loaded by an `Images` dataloader which, on initialisation, receives two arguments: `filepath` and `root`. The `filepath` variable directed the data-loader to a text file which specifies the location of all requisite images with respect to some location, given by `root`. Subsequently, the data-loader can be iterated, whereby all referenced images are loaded sequentially and returned as arrays of unsigned 16-bit integers.

The `Labels` data-loader, on the other hand, received just a single argument, `filepath`, which specified the location of a text file containing the corresponding ground truth labels for each image.

5.1.1.2. Data transforms

Two data transforms were implemented in order to first calibrate the real-world infrared images to a brightness and contrast that was approximately equivalent to the synthetic training data, and subsequently normalise all pixel values between zero and one.

The `calibrate` transform receives five arguments: an infrared image, I in addition to m, c, t_0, and t_1, where m and c correspond to the gradient and intercept of the linear regression line calculated in Section 5.2.1.2, and t_0 and t_1 are 0 and 100 respectively, as per the temperature range of the synthetic imagery, as detailed in Section 3.2.3.1. As per Equation 5.14, the pixel values of a given image were first transformed to the temperature domain, before being converted, via Equation 5.15, to the same intensity scale as the synthetic infrared imagery. Finally, all negative values were set to zero, any values greater 255 were set to equal 255, and the image was converted to unsigned 8-bit integers.

$$I \coloneqq \frac{(I - c)}{m} \tag{5.14}$$

$$I \coloneqq \frac{I \cdot 255}{(t_1 - t_0)} - t_0 \tag{5.15}$$

The `normalise` function simply converts the given image to floating-point 32-bit numbers and then divides all pixel values by 255 to return an image bounded between 0 and 1.

5.1.2. Pre-training with COCO

Finally, before the network can be trained with IRShips, its internal parameters must be initialised. Conventionally, if the available training data are fully representative of the expected deployment conditions, the most effective approach is to train the network from scratch using parameters selected at random following a method such as Xavier or Kaiming initialisation [93, 263].

However, when the available training data have size or diversity limitations, or are from a distribution which differs significantly from the test data, transfer learning is generally considered to be more suitable. Since the IRShips training dataset was generated synthetically, it was assumed to be from a different distribution to that of the real-world test sequence, so transfer learning was therefore used to optimise the YOLOv3 algorithm.

Transfer learning is a machine learning technique in which algorithm parameters which have been optimised for one task are reused as the starting point when training for a different task [108]. As a result, features that are already effectively recognized are simply adapted to the new data, rather than learned from scratch—which may not be possible if only limited data for this new task are available [264]. As a result, transfer learning usually accelerates the training process, and often delivers a superior algorithm performance.

To leverage the benefits of transfer learning, the YOLOv3 algorithm was first pre-trained for general object detection using the COCO benchmark dataset [139]. This dataset, examples of which are shown in Figure 5-19, provides approximately 2½ million labelled objects from 80 different categories across 118,287 training images, 5,000 validation images, and 40,670 test images, and as result, is ideal for creating general and transferable features. In order to train the YOLOv3 algorithm using the COCO dataset, a software pipeline, similar to that described in Figure 5-3, was implemented using the Deeplodocus deep learning development environment.

Figure 5-19: *Ground truth annotations for a selection of COCO training examples [139].*

First, the YOLOv3 algorithm, its corresponding loss function, optimiser, and output transforms were configured into the project, as described previously in Section 0, 5.3.4, 5.3.5, and 5.3.6 respectively.

To load image-label pairs during training and validation, the COCODetection data-loader from PyTorch's `torchvision.datasets` module and a `SourcePointer` were configured into the pipeline via the `data.yaml` file.

Metrics, such as average precision and average recall, were calculated using the `COCOeval` interface from the `pycocotools.cocoeval` module of the COCO API [265], which was then incorporated into the project using the `metrics.yaml` file.

Finally, data augmentation functions were implemented and incorporated into the project, in order to randomly rotate, crop, and scale each of the training examples and also apply colour space transforms.

With the training and validation pipeline configured, the YOLOv3 algorithm was trained for a total of 50 epochs, where an epoch is defined as one complete iteration of the training data, starting with an initial learning rate of 1×10^{-6}, which gradually reduced using a cosine decay [266], as described in Equation 5.16.

$$\eta_i = \eta_0 \frac{1 + \cos\left(\frac{n\pi}{N}\right)}{2}(1 - \delta) + \delta \tag{5.16}$$

where η_0 is the initial learning rate, N is the total number of training epochs, n is the index of the current epoch, and δ is the magnitude of decay.

This resulted in a validation average precision of 31%—roughly equivalent to the average precision of 33% originally reported with respect to the test set [146]. A tutorial for re-creating this project has been published on the Deeplodocus documentation website [267].

The internal parameters which resulted from this pre-training stage were used to initialise the YOLOv3 algorithm in subsequent experiments.

5.2. Training with IRShips

With the YOLOv3 algorithm now initialised with these pre-trained internal parameters, it was then trained using synthetic examples from the IRShips dataset. First, YOLOv3 was trained without any sea-state, sky-state, or background clutter augmentation, as this was considered to be the control state. In order to explore the effect of augmenting the synthetic IRShips images in this way, the network was then trained a further three times with each of the three forms of augmentation were separately applied. Finally, the algorithm was trained for a fifth time where all three forms of augmentation were applied together.

To ensure that each training run was directly comparable, all random processes—including dataset shuffling, image augmentation, geometric transforms, and the application of sensor noise—were seeded. Each training run lasted for 20 epochs, with each epoch comprising 32,000 randomly-selected training examples drawn from IRShips, grouped into a thousand batches of 32 images, using gradient accumulation [268] to fit them into the 11GB of GPU RAM available.

Visualisations from each of the five training runs are presented below in Figure 5-20, with their respective training and validation loss and metric values shown in Appendix A-7.

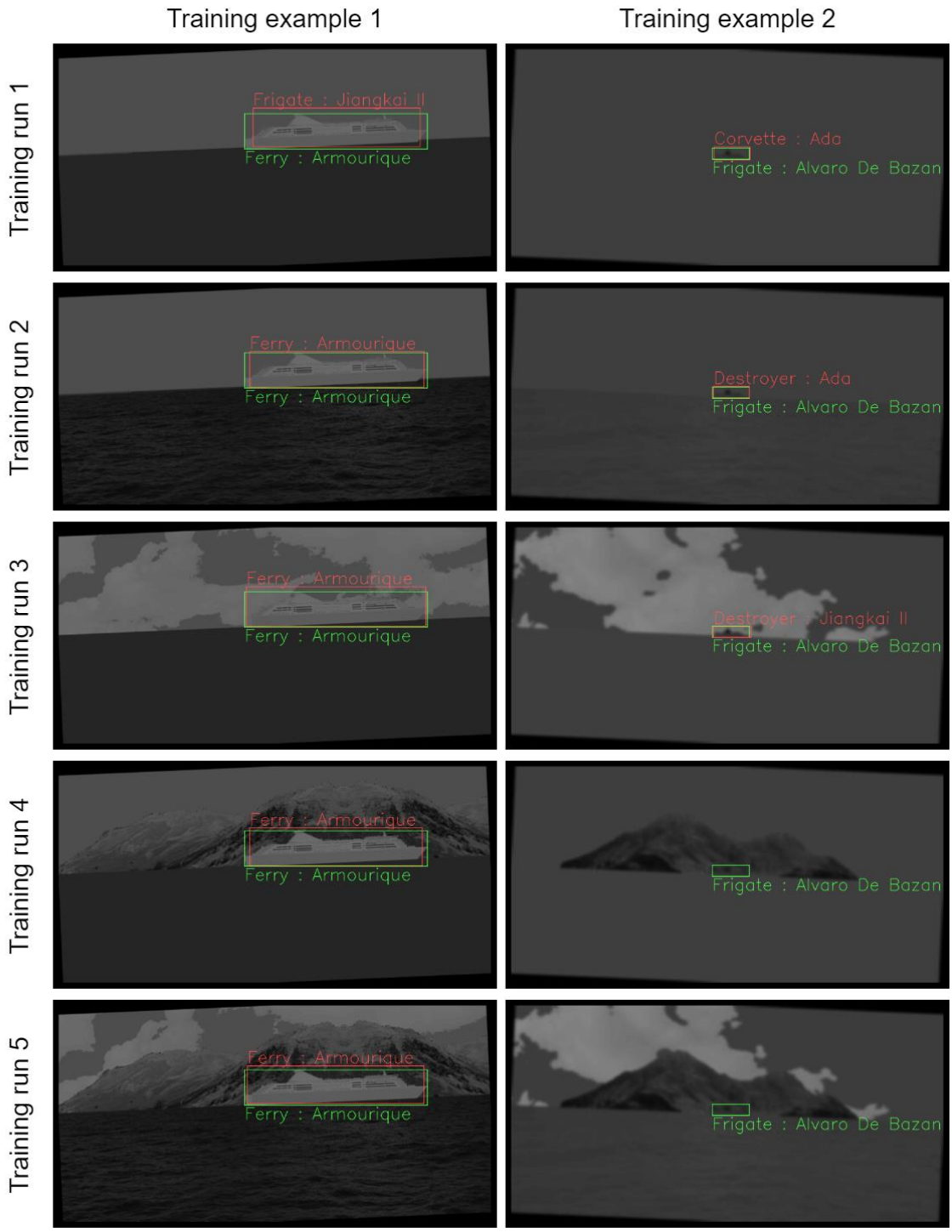


Figure 5-20: *Visualisations of two image examples from each of the five training runs, with ground truth labels in green, and predictions from the YOLOv3 algorithm in red.*

These images show how each image augmentation process increases the complexity of the ship detection task.

5.2.1. Results

Training loss and metric scores are plotted in Figure 5-21. As expected, this shows that YOLOv3 consistently fitted to the non-augmented data more accurately than to the augmented data, confirming that each of the three augmentation processes achieved their objective of significantly increasing the complexity of the synthetic infrared images for algorithm training.

Sea-state augmentation yielded the greatest effect, causing the final training F-score to drop by 3.9 percentage points, relative to the non-augmented control. This was followed by sky-state augmentation, which caused a reduction of 3.2 percentage points, and then background clutter augmentation with a 1.5 percentage point reduction. When applied together, these three augmentation processes caused the training F-score to fall by 5.5 percentage points.

Consequently, in respect of training loss and metric scores, it is logical to expect that—despite the negative effect of image augmentation on the network's training performance—the more realistic and more challenging examples resulting from image augmentation will deliver improved performance during validation with real-world images.

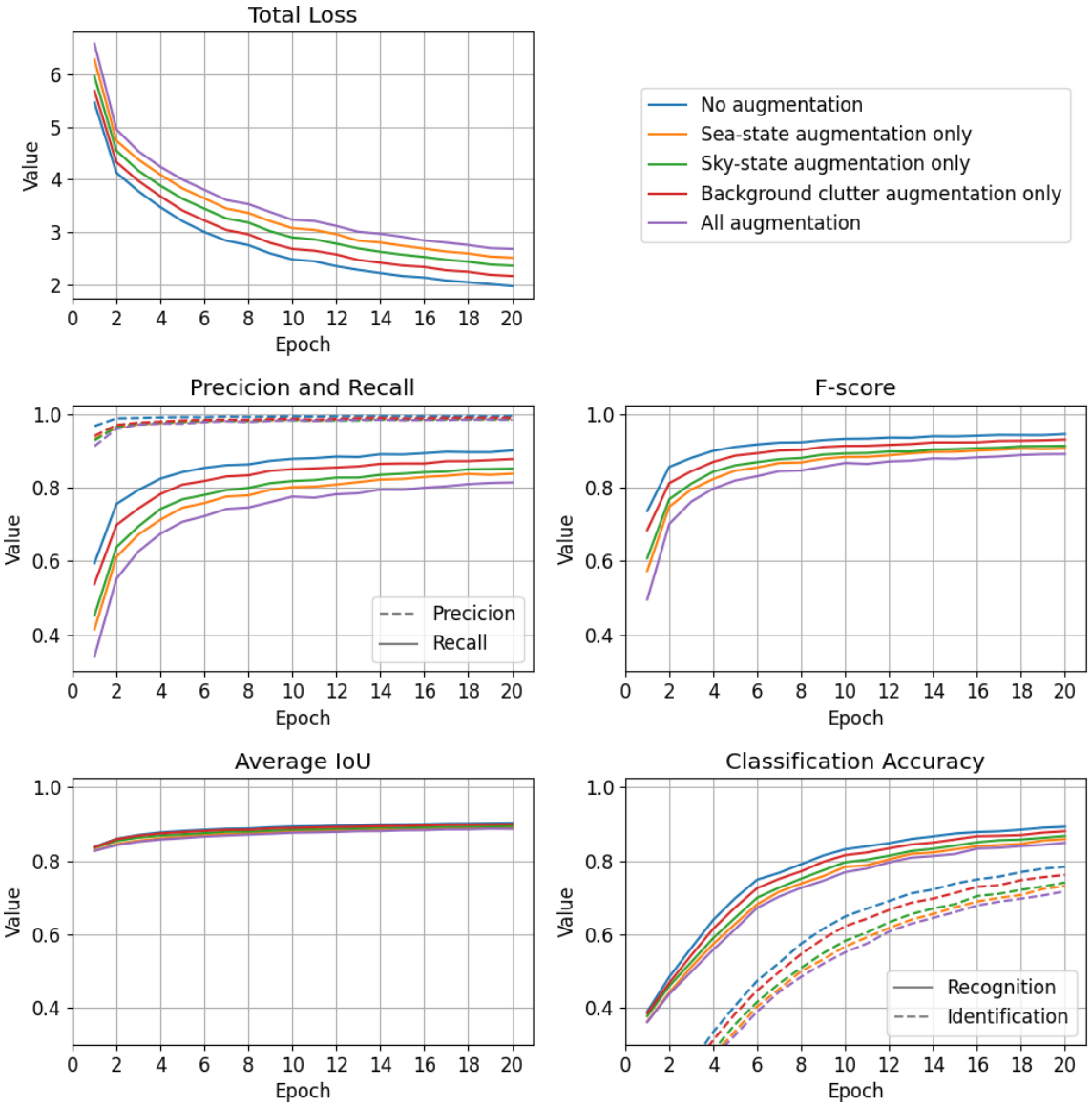


Figure 5-21: *The total training loss and metric scores after training with and without sea-state, sky-state, and background clutter augmentation.*

Validation loss and metric scores with respect to the real-world LW infrared images contained in the Karel Doorman-class frigate validation sequence are plotted in Figure 5-22. When trained with non-augmented images (the blue line in the charts contained within Figure 5-22), the network's validation loss did not converge. Instead, the algorithm began to overfit before the end of the first epoch, indicating that these training examples were not adequately representative of real-world conditions. Consequently, the validation F-score

peaked by the first epoch at just 0.198 (precision = 1.00 and recall = 0.110) and, as training continued, this performance only deteriorated.

In contrast, as shown by the non-blue lines in the charts contained within Figure 5-22, when any of the image augmentation processes were applied during training, the network's validation loss did not increase significantly, demonstrating that augmentation provided synthetic images which were suitable for training YOLOv3 to detect ships in real-world infrared imagery. As a result, the four YOLOv3 models trained with augmented data each achieved F-scores and average IoU scores than the control model.

As expected, the more realistic and more challenging examples resulting from image augmentation delivered improved performance during validation.

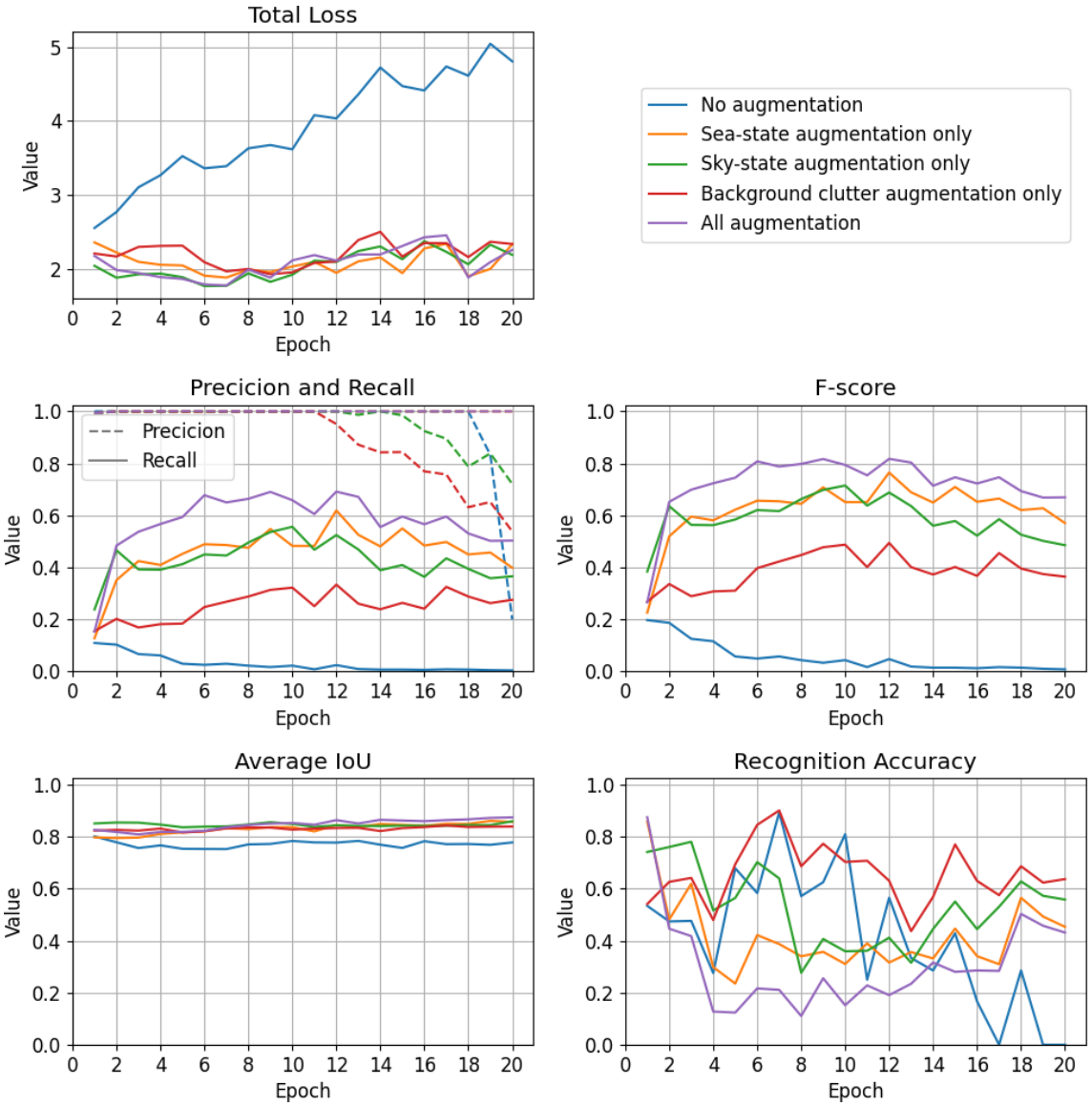


Figure 5-22: *The total validation loss and metric scores after training with and without sea-state, sky-state, and background clutter augmentation.*

Figure 5-23: *Detection of the Karel Doorman-class frigate by the YOLOv3 algorithm.*

Ultimately, when all three data augmentation processes were applied, YOLOv3 achieved a peak F-score of 0.818 (precision=1.00, recall=0.692) at epoch 12, at which point its average IoU score was 0.864. Figure 5-23 shows the network's predictions for a selection of frames taken from the Karel Doorman-class frigate validation sequence. These images show how the algorithm accurately inferred the bounding box of the ship, despite the high density of buildings in the background, serving to occlude the outline of the vessel. In addition, the algorithm continued to detect the ship as it left the scene and became only partially visible. Furthermore, despite the high density of buildings in the background and the presence of a navigation buoy and a rocky shore in the foreground, no false positives resulted.

In terms of the algorithm's apparently inconsistent performance in respect of recognition accuracy, as shown in Figure 5-22, then Figure 5-24—which plots the algorithm's ship-type and ship-class predictions with respect to each frame of the validation sequence—provides an explanation.

In fact, the low recognition accuracy is due to the Karel Doorman-class frigate being regularly mis-classified as a destroyer, a type of ship which can often

appear similar to frigates. Importantly, the frigate was never mis-classified as a corvette or—even more importantly—as a ferry. Since the Karel Doorman-class frigate did not feature in the training data, it was not possible for the algorithm to correctly identify the ship, which was most-commonly identified as an Akizuki-class vessel in 62% of detections, and as a Jiangkai II-class in 30%.

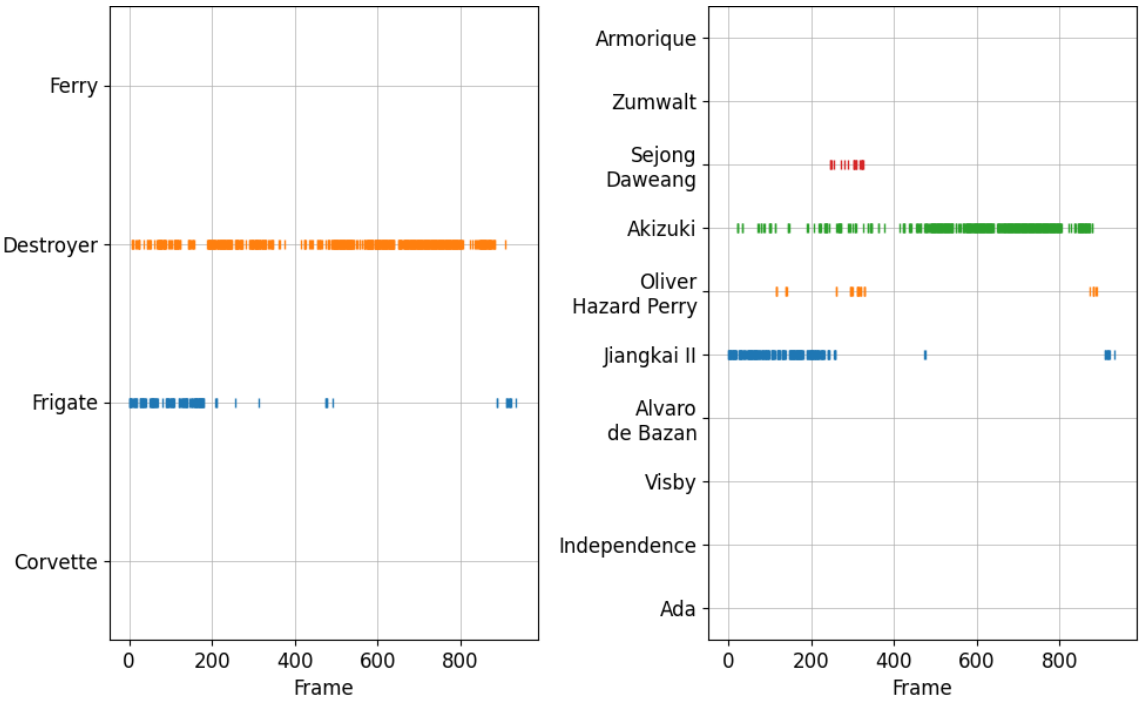


Figure 5-24: *How the Karel Doorman-class frigate in the validation sequence was recognized and identified by the YOLOv3 model (epoch 12) at each frame.*

Finally, in terms of speed, the algorithm achieved an average inference time of 16.8 milliseconds per validation image—which equates to approximately 60 frames per second—when using a Nvidia GeForce GTX 1080 Ti GPU.

5.3. Improving performance with multi-task training

As shown in Figure 5-22, the algorithm's validation F-score peaks at epoch 12, with an F-score of 0.818 deteriorating steadily thereafter, falling to 0.667 by epoch 20. Techniques to overcome this overfitting are therefore required, in order to enable the algorithm's performance to benefit from further training.

Overfitting can often be negated during the training process by applying regularisation techniques, such as batch normalisation, data augmentation, and L2 regularisation/weight decay. However, each of these techniques work best when the training and validation data originate from similar distributions, which in this case is not true. Consequently, despite all three of these techniques having been applied during training, as mentioned in sections 0, 5.3.3, and 5.3.5 respectively, the network nevertheless overfitted. As a result, an alternative method is required if the effects of overfitting are to be overcome.

Overfitting is typically indicative of an algorithm ‘learning’ spurious features that relate to the individual specifics of the training dataset, in this case comprised of synthetic data. It was therefore proposed to mitigate against this by incorporating some real-world images into the training process. It was reasoned that this would promote the algorithm’s learning performance, as the real-world images would provide a level of diversity which is not present in IRShips. Moreover, it was decided that the examples did not necessarily need to be from the infrared domain, and that more easily sourced visual-spectrum images may be sufficient instead.

Consequently, a web-scraping program was written using Python3, which used a selection of keywords in order to automatically download from Google Images approximately 10,000 images of the ten vessels present in IRShips. After the manual removal of unsuitable images, this resulted in a collection of 8,343 images, which were near-evenly distributed across the 10 class categories. Examples of these images are shown in Figure 5-25. Finally, each image was converted to grayscale using Equation 3.3.

Figure 5-25: *A selection of the real-world visual-spectrum images that were collected from Google Images.*

These images were not labelled with bounding box coordinates, since it was reasoned that simply training the network's encoder to classify these images would enable it to learn sufficiently general encoding features so as to improve the detection performance of the whole network. Therefore, manual bounding box annotation—which is a cumbersome and time-consuming task—would not be necessary. Consequently, these real-world example images were labelled only with ship-type and ship-class annotations, and for that reason are referred to as semi-labelled.

To make use of these semi-labelled examples during training, the YOLOv3 algorithm was modified with a new fully-connected layer at the end of its encoder, as shown in Figure 5-26. This layer performs image classification in order to predict both the type and class of the ship or ships present in a given image.

In addition, the pre-existing loss function was modified in order to ignore any examples which were not fully-labelled, and a second loss function included. This second loss function computed the cross-entropy loss for both the ship-type and ship-class predictions of the new full-connected layer, and was applied to semi-labelled examples only.

Finally, the now-modified YOLOv3 algorithm was then initialised with the internal parameters that had been learned during pre-training with the COCO dataset in Section 5.1.2, and the new fully-connected classification layer optimised. Optimisation was achieved by ‘freezing’ the parameters of all other learned layers, and training the network for 20 epochs with only the semi-labelled examples. This was done to ensure that each layer was at least partially optimised before training with the mix of IRShips and semi-labelled data, for the sake of stability.

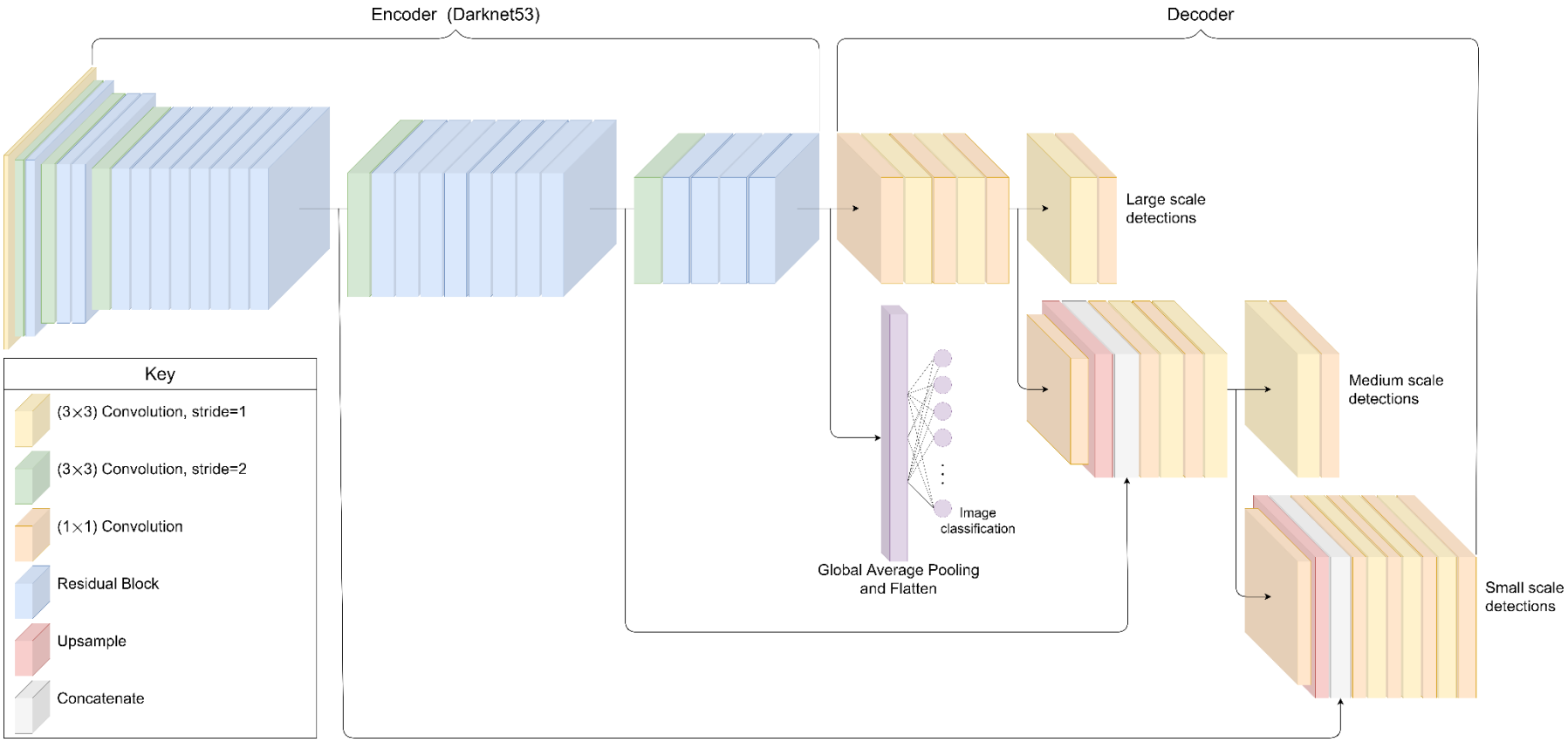


Figure 5-26: *The YOLOv3 algorithm with the addition fully-connected layer which performs image classification based on the outputs of the backbone encoder.*

5.3.1. Training with IRShips and semi-labelled visual-spectrum data

To train YOLOv3, all internal parameters were ‘unfrozen’, and the model was then trained using both IRShips and the semi-labelled visual-spectrum examples for a further 20 epochs. During training, the semi-labelled examples were subjected to random rotations, and re-scaling. The training and validation loss and metric values from this training run are shown in Appendix A-8.

5.3.2. Re-distributing the semi-labelled data

Next, it was proposed YOLOv3 might learn more useful features from the visual-spectrum examples if the pixel intensity distribution of this data correlated more closely with that of the infrared validation sequence.

Figure 5-27 shows that the pixel intensity distribution of the visual-spectrum images differs significantly from that of the infrared validation sequence. Therefore, each visual-spectrum image was adjusted using Equation 5.17, in order to ensure a greater similarity to pixel intensities of LW infrared imagery. The effect of this process on the pixel intensity distribution of the visual images is plotted in Figure 5-27 and shown in Figure 5-28.

$$I \coloneqq \frac{(I - \tilde{I}) \times \sigma}{std(I)} + \mu \tag{5.17}$$

where I is a visual-spectrum image, $\tilde{I}$ and $std(I)$ are the median and standard deviation the image respectively, and μ and σ are the mean and standard deviation for the LW infrared validation sequence respectively. Median pixel intensity, as opposed to mean, was used to mitigate against the relatively large frequency of pixels with values approaching 255 in the visual images.

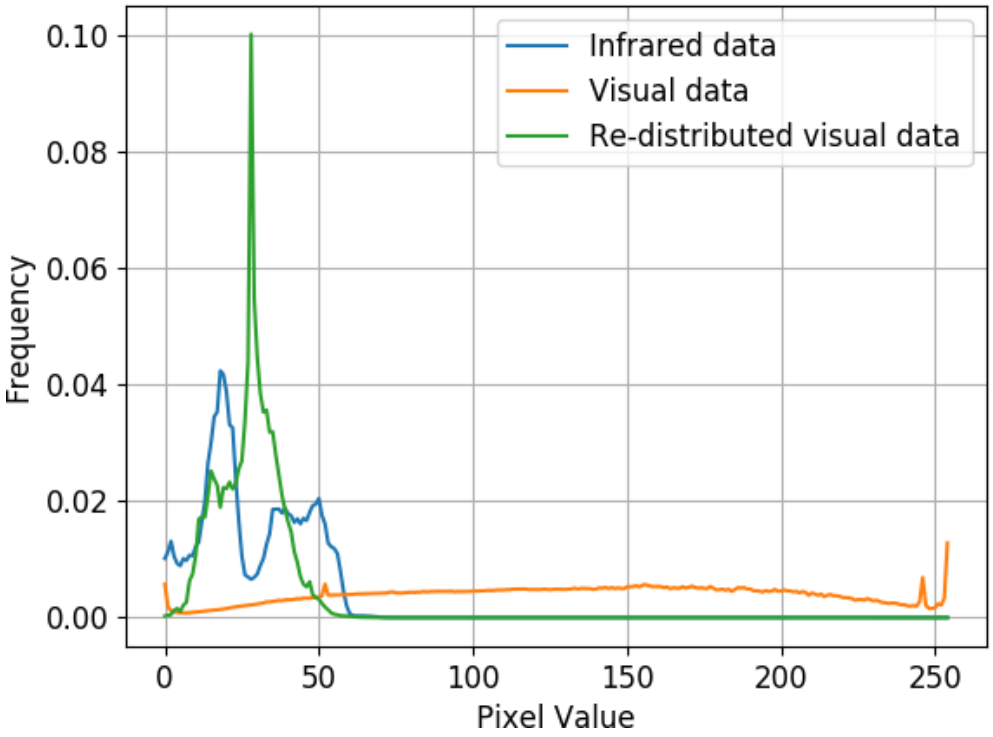


Figure 5-27: *Pixel intensity distributions of the infrared validation data, the visual-spectrum data, and the re-distributed visual-spectrum data.*

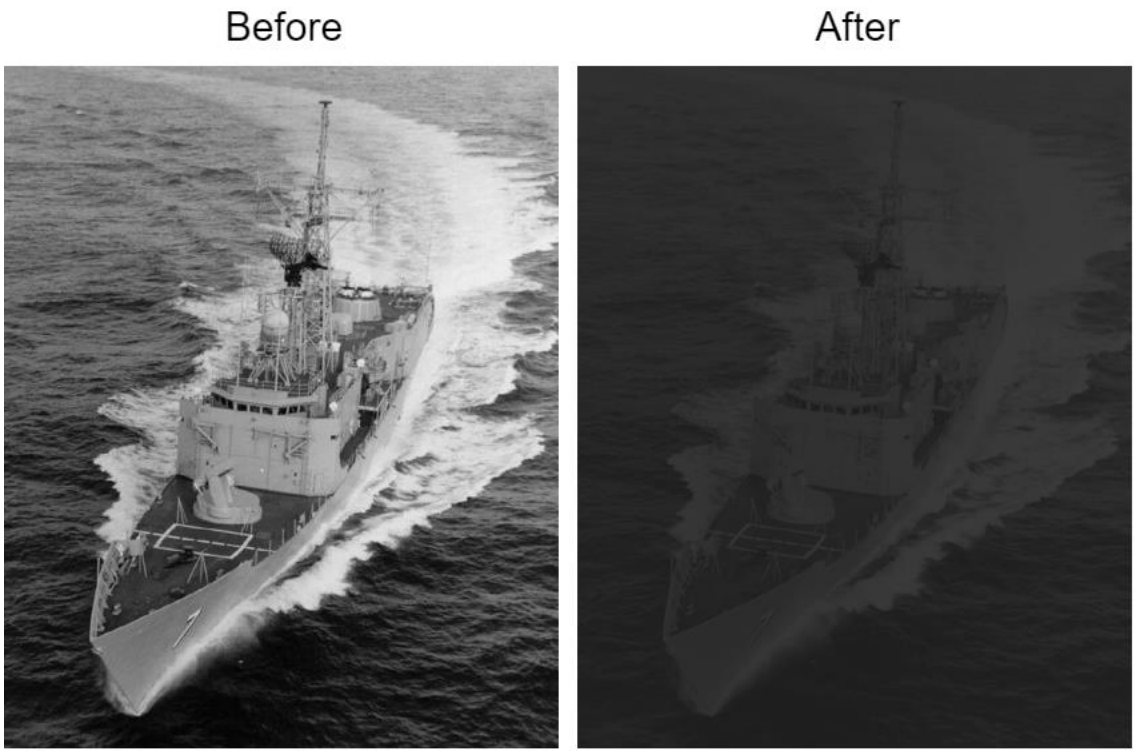


Figure 5-28: *The effect of contrast re-distribution on a visual-spectrum image.*

The algorithm was then initialised with the pre-trained parameters generated in Section 5.3, and subsequently trained again, using both examples from IRShips and these ‘re-distributed’ visual-spectrum images. All training and validation loss and metric values from this training run are shown in Appendix A-9.

5.3.3. Results

As shown in Figure 5-29, the inclusion of semi-labelled real-world images in the training process significantly reduced the effect of overfitting, despite these images not being from the infrared domain. Originally, when the network was trained only with data from IRShips, the network’s F-score peaked at epoch 12 and then began to reduce—falling by 18% by epoch 20. In contrast, once the semi-labelled real-world examples had been introduced into the training dataset, the network’s F-score peaked at epoch 12, and subsequently remained at a similar level, falling just 3–5% by epoch 20. However, while reducing the effect of overfitting, the inclusion of the semi-labelled visual-spectrum images in the training dataset did not significantly improve the algorithm’s peak F-score.

Once the pixel intensities of the visual-spectrum images were re-distributed via Equation 5.17, the network peak F-score increased by 16% from 0.818 to 0.945 (precision = 0.986, recall = 0.907). This considerable increase in algorithm detection performance required the addition of just 8,343 easily-sourced images and did not necessitate any manual annotation.

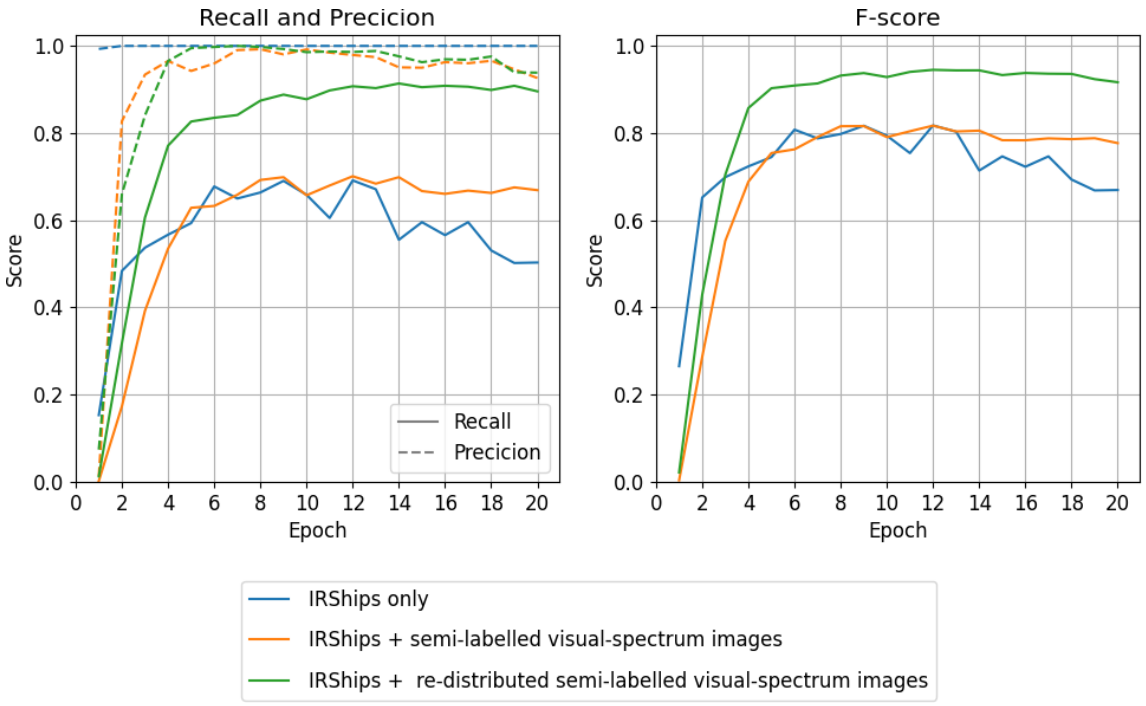


Figure 5-29: *Performance comparison between YOLOv3 trained with 1) IRShips only, 2) IRShips and the semi-labelled visual-spectrum images, and 3) IRShips and the re-distributed semi-labelled visual-spectrum images.*

Since these real-world images do not contain bounding box annotations, the network's performance with respect to these images cannot be quantified, but can be inspected visually. Predictions made by the best performing network for a selection of the re-distributed semi-labelled visual-spectrum training images are shown in Figure 5-30. Despite not being trained with ground truth bounding boxes, the network still learned to consistently localise the ships in these images. This indicates that the features which the network learned in order to classify these images, were also used effectively by the network's decoder to infer the bounding box, type, and class of individual ships.

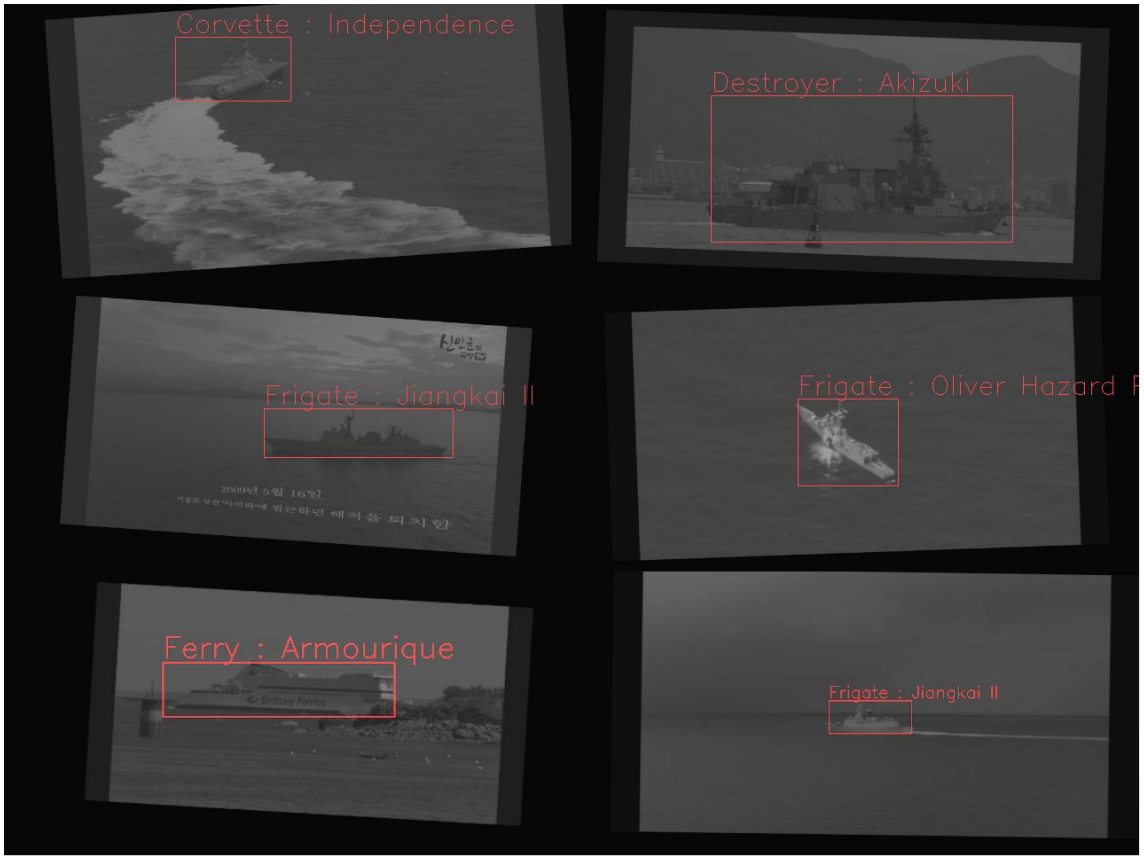


Figure 5-30: *Ship detections by YOLOv3 on the re-distributed semi-labelled visual-spectrum images. (Image contrasts have been enhanced for illustrative purposes.)*

Since some of the semi-labelled images contain multiple vessels, this enables a demonstration of the algorithm's ability to detect multiple ships. Despite the fully-labelled training data containing just a single ship per image, Figure 5-31 shows that the network is able to successfully detect multiple vessels. Since YOLOv3 performs equivalent computation for each 32 × 32, 16 × 16, and 8 × 8 region of the given image, the network is able to detect any number of vessels, though it cannot be expected to perform well when vessels occlude each other, since this condition was not included in the training data.

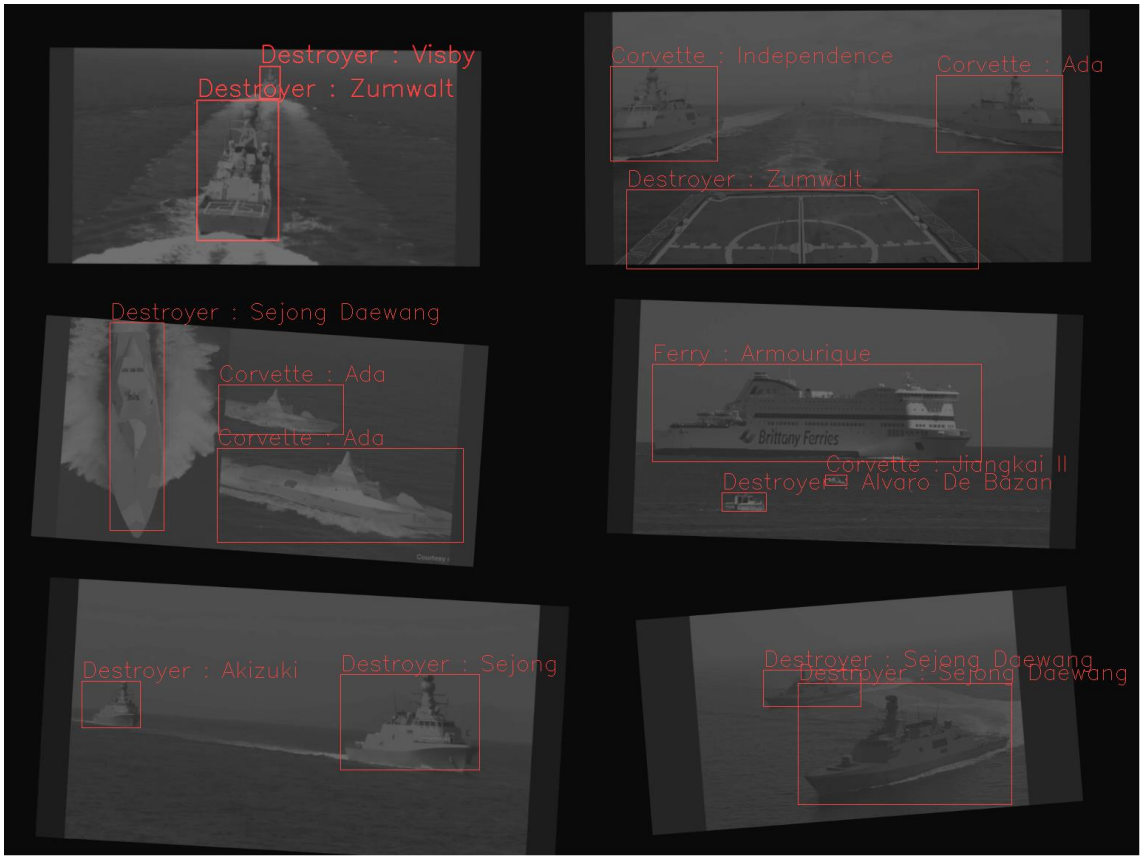


Figure 5-31: *Multiple ship detections by YOLOv3 on the re-distributed semi-labelled visual-spectrum images. (Image contrasts have been enhanced for illustrative purposes).*

5.4. Conclusion

In Section 5.2 the YOLOv3 general object detection algorithm was trained to detect, recognize, and identify military ships in infrared images, using the IRShips dataset. This is believed to represent the first time that a fully-convolutional algorithm has been trained for this application using synthetic infrared data, with the resultant network achieving an F-score of 0.818 with respect to a complex real-world infrared validation sequence.

Overfitting is a perennial danger in computer vision, especially when using either limited or synthetic training data. Overfitting was mitigated against by incorporating real-world visual-spectrum images, corrected for pixel-intensity, into the training process. The inclusion of this data considerably reduced the

effect of overfitting, and raised the network's peak F-score by 12.7 percentage points from 81.8% to 94.5%. This considerable increase in algorithm detection performance required the addition of just 8,343 easily-sourced images and did not necessitate any manual annotation.

Chapter 6

6. Benchmarking and developing YOLOv3

This chapter describes how the detection accuracy of YOLOv3 was benchmarked against that of Faster R-CNN and Mask R-CNN using visual and near-infrared test data from the Singapore Maritime Dataset. It also describes a method of spectral domain-dependent encoding, which enabled YOLOv3 to achieve state-of-the-art performance with respect to the near-infrared test data while maintaining its three-factor speed advantage.

6.1. Introduction

When developing missile seeker algorithms, inference speed and detection accuracy are generally in contention: greater inference speed comes at the expense of lower levels of detection accuracy, and vice versa. Typically, single-stage algorithms prioritise inference speed; two-stage algorithms, on the other hand, generally offer superior detection accuracy, but at a speed that is rarely considered adequate for real-time applications. Consequently, single-stage CNNs, such as YOLOv3, are generally considered to be better suited to missile seeker applications than their two-stage counterparts.

Consequently, this chapter compares the speed-accuracy trade-off of YOLOv3 with two alternative CNN-based detection algorithms, Faster R-CNN [127] and Mask R-CNN [126], in order to explore how it might be possible to improve the detection accuracy of the YOLOv3 algorithm while maintaining its superior inference speed.

6.2. The Singapore Maritime Dataset

There are no publicly available benchmark datasets containing LW infrared images of military ships. Consequently, an alternative dataset must be used.

Shown in Figure 6-1, the Singapore Maritime Dataset [125, 128] contains a total of 62 fully-annotated video sequences of civilian ships and other maritime

objects in both the visible and near-infrared spectrum. These 62 sequences contain a total of 31,653 labelled frames, 240,842 labelled objects, and, as detailed in Table 6-1, have been grouped into pre-defined training, validation, and test sets. This makes the Singapore Maritime Dataset the largest publicly available benchmark dataset for maritime object detection.

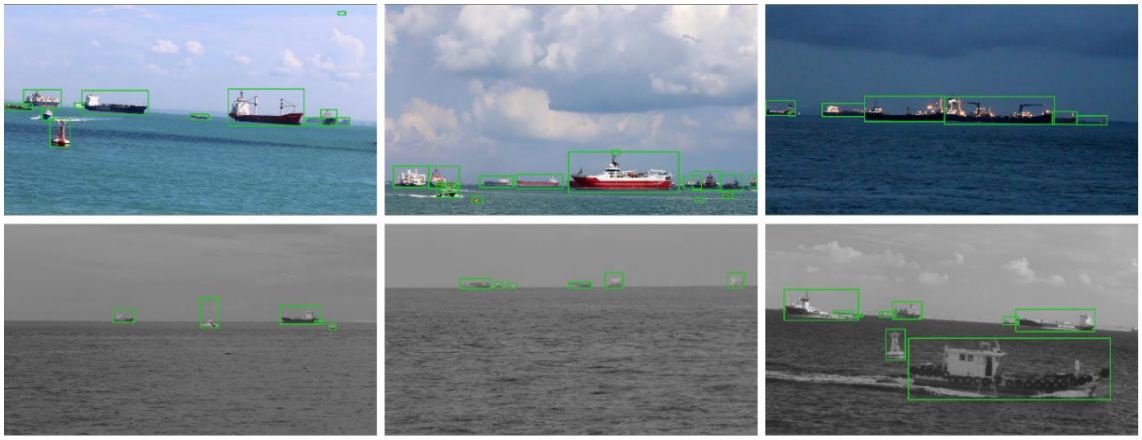


Figure 6-1: *Examples of visual-spectrum (top row) and near-infrared (bottom row) images from the Singapore Maritime Dataset.*

Table 6-1: *Summary of the Singapore Maritime dataset training, validation, and test data split.*

Waveband	Frames (Videos)			Total
	Training	Validation	Test	
Visual	12,830 (26)	1,705 (3)	5,232 (10)	19,767 (39)
NIR	8,953 (17)	0 (0)	2,333 (6)	11,286 (23)
Total	21,783 (43)	1,705 (3)	7,565 (16)	31,053 (62)

That said, the Singapore Maritime Dataset does not contain any LW infrared imagery, nor any examples of military vessels. However, its 10 visual-spectrum and 6 near-infrared test sequences contain approximately 60,000 examples of ships under a range of illumination conditions, including hazy, daylight, and dark/twilight. Consequently, despite its lack of LW infrared images and military ships, the Singapore Maritime Dataset is well suited for characterising the performance of ship detection algorithms and has previously been used to

benchmark and compare the performance of the Faster R-CNN and Mask R-CNN algorithms [125].

The Singapore Maritime Dataset provides both ship bounding box and object class labels for 10 categories: ferry, buoy, vessel/ship, speed boat, boat, kayak, sailboat, swimming person, flying bird/plan, and other [128]. However, these class labels are inconsistent and have been found to change at least for about 9% of objects in the dataset [125]. Furthermore, there is a strong imbalance between classes, with fewer than 5,000 examples for each of buoy, boat, kayak, sailboat, and swimming person, but over 50,000 examples for vessel/ship. Consequently, for the purposes of algorithm comparison, the Singapore Maritime Dataset was reformulated from a ten-class problem to a two-class problem (object vs background) by ignoring the class labels.

6.3. Experimental setup

A deep learning pipeline for training, validating, and testing YOLOv3 with the Singapore Maritime Dataset was implemented using the Deeplodocus deep learning environment. Shown in Figure 6-2, this system incorporated the same detection algorithm, optimiser, and output transforms that were described in sections 5.3.2, 5.3.5, and 5.3.6 respectively, together with a sub-set of the performance metrics described in Section 5.3.7.

However, different data-loaders and input data transformers were required in order to load and process image and label data from the Singapore Maritime Dataset. In addition, the loss function, described previously in Section 5.3.4 was adapted in response to the absence of ground truth object-class labels.

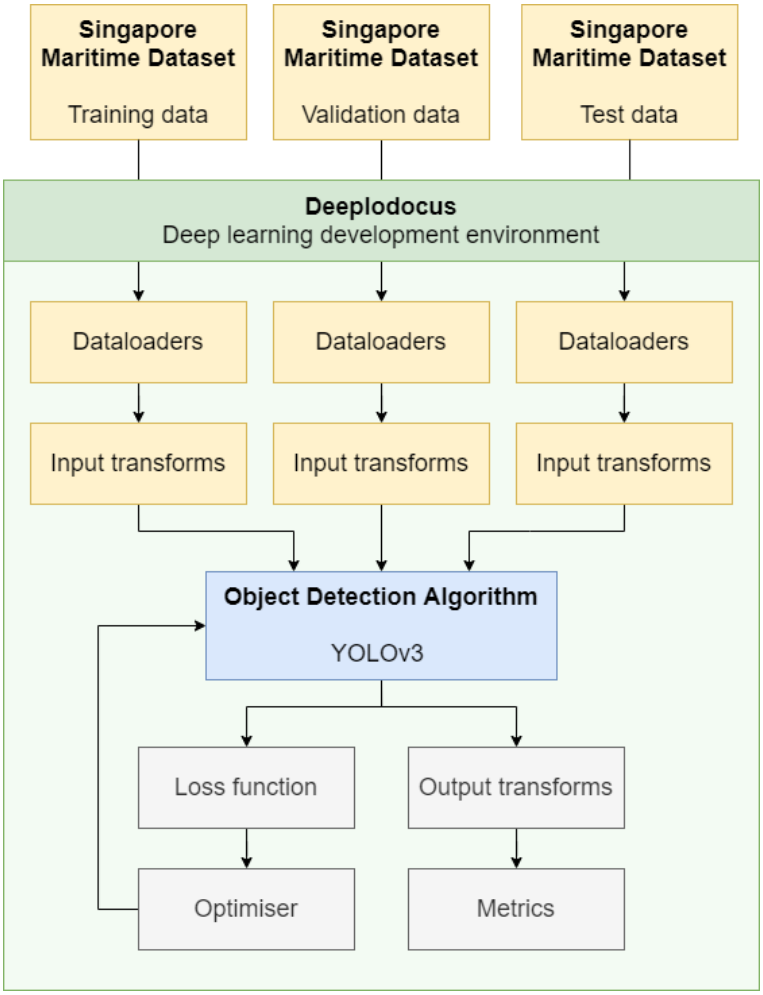


Figure 6-2: *Overview of the software pipeline which used the Singapore Maritime Dataset to train, validate, and test the YOLOv3 object detection algorithm.*

6.3.1. Training, validation, and test data

The visual-spectrum and near-infrared video sequences contained within the Singapore Maritime Dataset were categorised into training, validation, and test sub-sets using the data categorisation proposed by Moosbauer *et al.* [125], which is described in Appendix A-10. When categorising data into training, validation, and test subsets it is crucial to minimise any potential bias, as this can prevent a reliable indication of algorithm accuracy from being ascertained. The categorisation designed by Moosbauer *et al.* [125], minimises bias by ensuring that no single object (which may appear in multiple videos) appears in more than one sub-set at a time, and by ensuring similar distributions of object sizes and aspect ratios across the three sub-sets.

6.3.1.1. Data-loaders

Since each video sequence of the Singapore Maritime Dataset is stored as an AVI file, it was necessary to create a new `Videos` data-loader to extract and iterate the constituent frames from each video. The data-loader receives two positional arguments: `filepath` and `root`, where `filepath` directs the data-loader to each of the video files with respect to the location given by `root`.

Upon receipt of an index value, the data-loader must fetch the corresponding frame from the relevant video file. However, the process of opening the relevant a video file, selecting the required frame, and then closing the video file, for every image required would be prohibitively time-consuming.

A considerably faster solution would be to load the constituent frames from all the required videos in advance, store each frame in RAM, and then fetch each frame from memory as required. However, storing all 12,830 visible-spectrum training images in this way would require at least 80 GB of RAM—which is more storage than that provided by the available hardware.

Instead, when initialised for the first time, the `Videos` data-loader reads each referenced video file, extracts all constituent frames, and then saves each frame to separate PNG files. Subsequently, when a frame is requested, it can be loaded from its corresponding image file without the need to open a much larger video file and also without requiring prohibitively large quantities of RAM.

A new data-loader was also required in order to load the Singapore Maritime Dataset's ground truth labels, which are stored in MATLAB-formatted files, known as MAT files. The `Labels` data loader receives a single positional argument, `filepaths`, which specifies the locations of the ground truth MAT file. On initialisation, each MAT file—which contains the bounding box coordinates for each object in each frame—is loaded into RAM by the `scipy.io` [269] module's `loadmat` function, and the ground truth for each frame is extracted. Subsequently, when a label is requested by its index value, the corresponding ground truth bounding boxes are returned as an $n \times 4$ array, where n is the number of ground truth objects.

6.3.1.2. Input transforms

A series of input transformers were included in order to normalise and format the data, as well as to increase the variation between training examples. Figure 6-3 shows the pipeline of input transforms used during training.

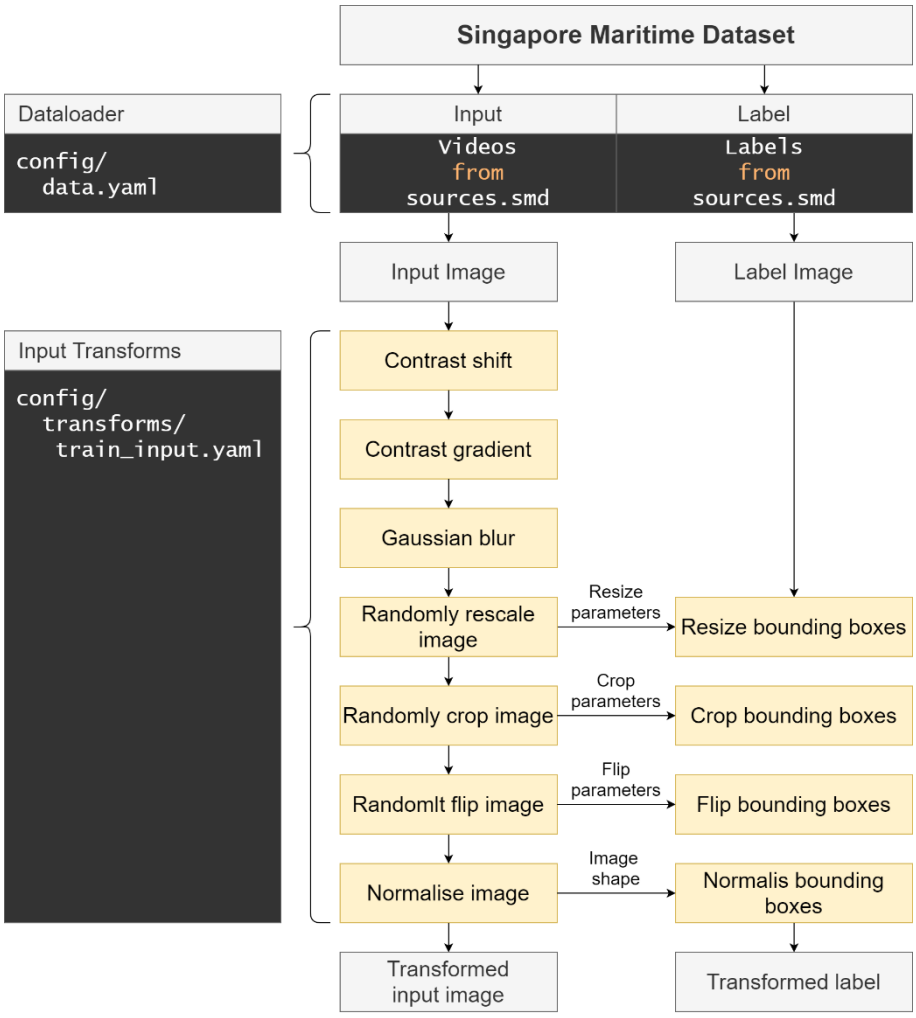


Figure 6-3: *Overview of the transformation process for training images and their corresponding labels.*

The first three input transforms executed pixel-wise transformations that altered the appearance of images without affecting the size, position, or orientation of target objects within the image. Shown in Appendix A-11, the first two of these transforms randomly adjusted the contrast of given images, while the third transform applied Gaussian blur using a kernel size selected at random from the set $\{1, 3, 5\}$.

Next, variation was further increased through three geometric transforms: random scaling, random cropping, and random flipping. The random scaling function resized each image to a resolution selected at random from the sequence $\{(a_1 \times 64, a_1 \times 32), (a_2 \times 64, a_2 \times 32), \ldots, (a_{13} \times 64, a_{13} \times 32)\}$ where $a = \{7, 10, \ldots, 19\}$. This function was seeded such that the resolution was only re-selected after every fourth batch of images. The random cropping function selected a patch of between 80% and 100% of the image size, before rescaling this cropped section to the size of the original image. The final geometric transform applied a horizontal flip to 50% of the input images.

For each geometric transform applied to an image, an equivalent transform was similarly applied to the image's ground truth annotations. Each geometric transformation applied to an example image was defined by a series of randomly-selected parameters. After each image was transformed, these parameters were passed to a second transformation which subsequently applied an equivalent transformation process to the coordinates of the corresponding ground truth bounding boxes.

Finally, the image was normalised by dividing all pixel values by 255, and the corresponding bounding box label was then normalised by dividing its x and y coordinates by the width and height of the image respectively.

During algorithm validation and testing, the only input transforms that were applied were the rescaling function that resized all images to a resolution 1024 × 576 and the normalisation function.

6.3.2. Loss function and metrics

Due to the absence of object class labels in the Singapore Maritime Dataset, the YOLOv3 loss function described in Equation 5.7 required adjustment, with the final two terms being omitted.

Furthermore, the Recognition Accuracy and Identification Accuracy metrics were disregarded due to the absence of object class labels, and the Average IoU metric was also disregarded, since this metric was not evaluated by Moosebauer *et al*. [125]. This left precision, recall, and F-score metrics as the

three remaining metrics, which were each calculated with respect to two IoU thresholds, τ_{IoU} of 0.3 and 0.5.

Conventionally, an IoU of 0.5 is used to determine true positive detections. However, it has been observed that the bounding boxes of the Singapore Maritime Dataset are labelled inconsistently when objects are partially occluded: sometimes only the visible part is annotated, and sometimes the estimated entire object is annotated [125]. As a result, it was proposed by Moosbauer *et al.* that a reduced IoU threshold of 0.3 should also be considered.

6.4. Benchmarking YOLOv3 with the Singapore Maritime Dataset

With the deep learning pipeline complete, the YOLOv3 algorithm was then trained twice. First, using the 12,830 visual-spectrum training examples, and then again using the 8,953 near-infrared training examples. In both instances, the algorithm was trained for a total of 50 epochs while applying a constant learning rate of 1×10^{-6}.

The two resultant models were subsequently evaluated using the 5,232 visual-spectrum and 2,333 near-infrared test examples respectively, and their precision-recall curves were calculated by incrementing the algorithm's objectness threshold, τ_{obj} between 0 and 1 by 0.05.

6.4.1. Results

Figure 6-4 shows the resultant precision-recall curves for YOLOv3 with respect to the visual-spectrum test examples, along with those for Mask R-CNN and Faster R-CNN, as reported by Moosbauer *et al.*

Of the three object detection algorithms evaluated, Mask R-CNN achieved the highest accuracy, achieving maximum F-scores of 0.875 at the IoU thresholds of 0.3 and 0.5 respectively. At an IoU threshold of 0.3, YOLOv3 was the worst performing algorithm—achieving a maximum F-score two percentage points below that of Faster R-CNN. However, YOLOv3 outperformed Faster R-CNN

when considering an IoU of 0.5, outperforming it by one percentage point, indicating that YOLOv3 may perform better than Faster R-CNN when tightly-defined ship bounding boxes are required.

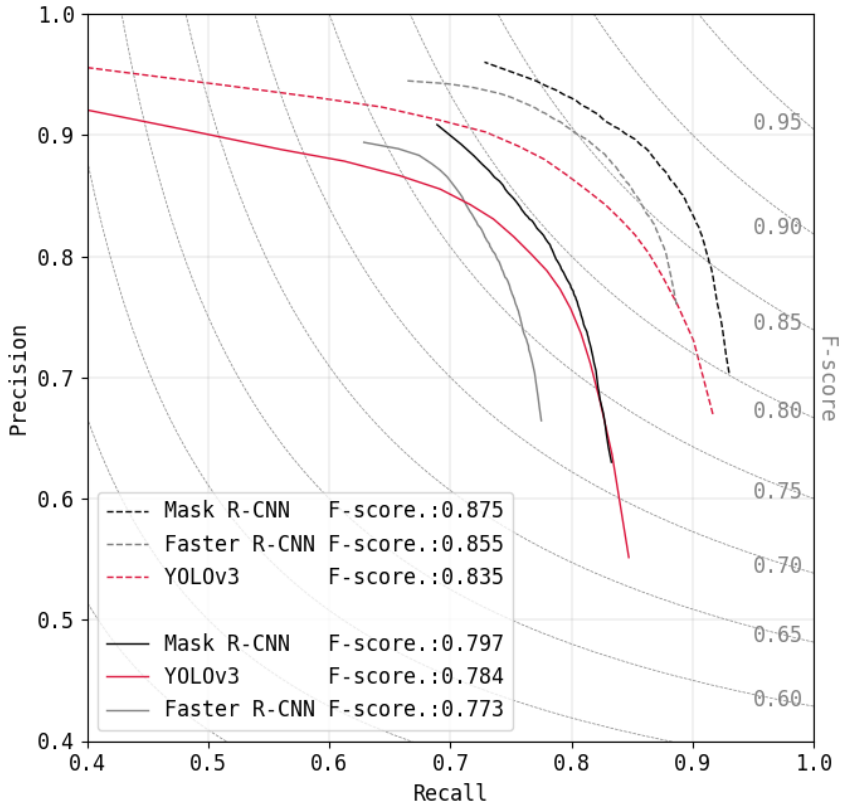


Figure 6-4: *Precision-recall curves for YOLOv3, Faster R-CNN, and Mask R-CNN with respect to the visual-spectrum test data. Solid lines correspond to an IoU threshold of 0.5, and dashed lines to an IoU threshold of 0.3. (Values for Mask R-CNN and Faster R-CNN are sourced from [125].)*

Figure 6-5 shows the precision-recall curves for YOLOv3 and Mask R-CNN with respect to the near-infrared test examples.

YOLOv3 failed to outperform Mask R-CNN at either IoU threshold. At a threshold of 0.3, it achieved a maximum F-Score of 0.850 (precision = 0.853 and recall = 0.846), 2.7 percentage points lower than that of Mask R-CNN. However, the detection accuracy of YOLOv3 was particularly poor at a threshold of 0.5, where its maximum F-score was 7.4 percentage points lower than that of Mask R-CNN.

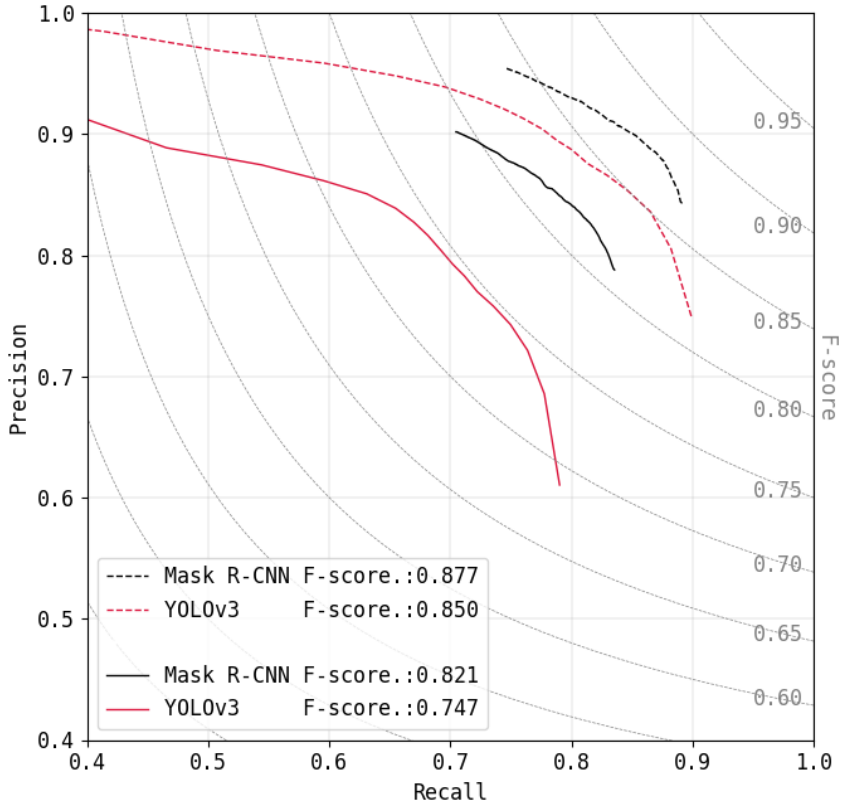


Figure 6-5: *Precision-recall curves for YOLOv3 and Mask R-CNN with respect to the near-infrared test data. Solid lines correspond to an IoU threshold of 0.5, and dashed lines to an IoU threshold of 0.3. (Values for Mask R-CNN are sourced from [125].)*

These results for YOLOv3 with respect to both the visual-spectrum and near-infrared test data are also tabulated in Appendix A-12.

Figure 6-6 shows the inference time and inference speed of YOLOv3 and Mask R-CNN for a range of different image sizes.

At the image resolution used for testing, 1024 × 576, YOLOv3 achieved an average inference speed of 24 frames per second, more than three times faster than Mask R-CNN's average inference speed of 7.5 frames per second. Consequently, despite its reduced detection accuracy, YOLOv3 is considerably better suited to use onboard remote real-time systems, such as infrared anti-ship missiles, where computational hardware is limited by the size and electrical power constraints of the missile.

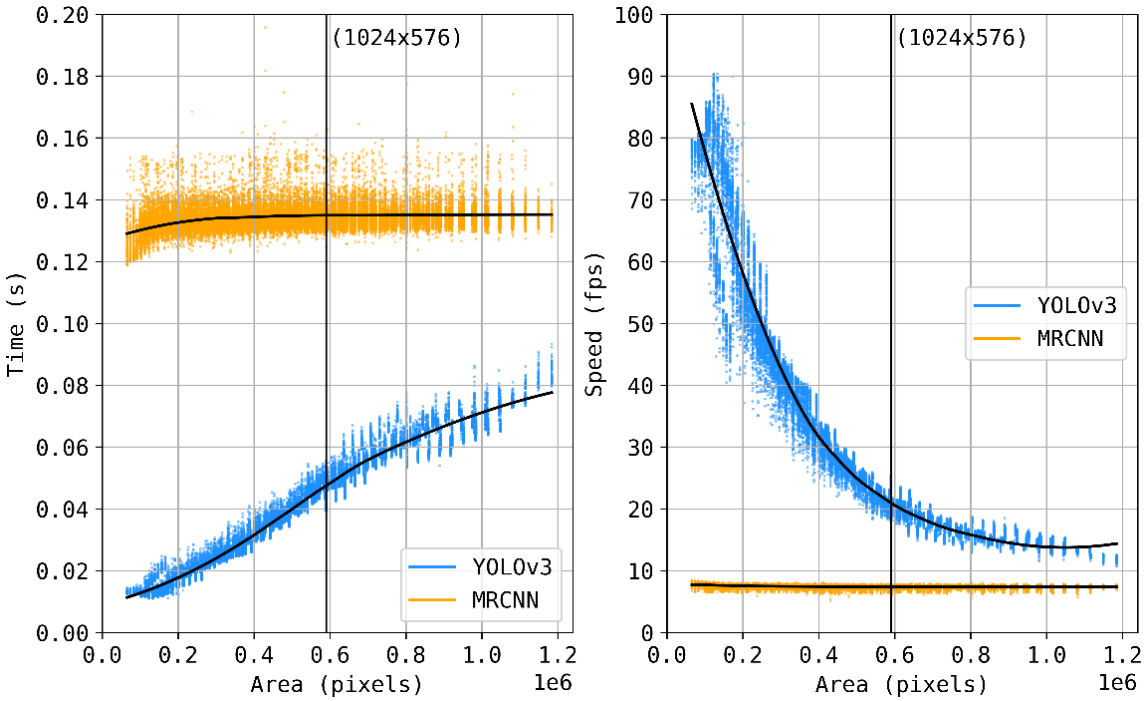


Figure 6-6: *Comparison of object detector inference speeds with image size. The vertical line at 5.9×106 corresponds to the image resolution used during algorithm testing in this chapter.*

Of the two two-stage algorithms, Mask R-CNN is the most accurate, consistently outperforming YOLOv3 at both of the IoU thresholds, as well as across both the visual-spectrum and the near-infrared test sequences. But while Mask R-CNN demonstrates the level of detection accuracy that two-stage CNN-based algorithms—thereby setting the bar for what YOLOv3 could possibly achieve in terms of accuracy—it does so at a speed that is three times slower than YOLOv3.

This therefore raises the question: might it be possible to improve the detection accuracy of the YOLOv3 algorithm while maintaining its superior inference speed?

6.5. Improving YOLOv3 with spectral domain-dependent encoding

Recalling Chapter 3, the general rule of thumb proposed by Goodfellow, Bengio, and Courville was that deep learning algorithms generally achieve an acceptable performance with around 5,000 labelled examples per category, but that surpassing human-level performance might require over 10 million examples [108] (page 20).

In terms of this heuristic, although the Singapore Maritime Dataset contains an adequate quantity of training examples to deliver an 'acceptable' level of performance—12,830 visual-spectrum images and 8,953 near-infrared images—it does so by only a margin of roughly 1.5 in the case of visual-spectrum images and 0.8 in the case of near-infrared images. Consequently, the provision of more training data is an obvious consideration when seeking to improve detection accuracy.

Clearly, though, it might be argued that including further training data from outside the Singapore Maritime Dataset could invalidate its use as a benchmark. Ideally, then, any performance gains should be sought by relying solely on training data contained within the Singapore Maritime Dataset. Given that the dataset contains images from two spectral domains, the visual-spectrum and near-infrared spectrum, this raises an interesting possibility: could the training data from one spectral domain be used to improve algorithm performance in a second domain? And if so, how could this best be achieved?

6.5.1. Method

First, the YOLOv3 algorithm was simply re-trained using a combined training dataset containing both the visual-spectrum and the near-infrared training examples. This YOLOv3 model—which is subsequently referred to as YOLOv3-all—was trained for a total of 50 epochs at a constant learning rate of 1×10^{-6}. It was subsequently evaluated with respect to both the visual-spectrum test data and then the near-infrared test data.

Training the algorithm with both visual-spectrum and near-infrared data in this way forces its encoding layers to learn features which can be generalised across the two different spectral domains. it is therefore reasonable to expect these more general features to prove more effective than those learned by the previous two YOLOv3 models, which were trained with either visual-spectrum or near-infrared data only.

However, it can also be reasoned that forcing the algorithm to generalise across multiple spectral domains will prevent it from learning useful domain-specific features. But, while effective across both spectral domains, this might deliver sub-optimal performance when applied in any single domain. Therefore, it is important that the algorithm is capable of learning both domain-specific features and general cross-domain features.

Finally, it was reasoned that the learning of both domain-specific and general cross-domain features could best be facilitated by including separate encoding convolutional layers for each different spectral domain, the premise being that separate encoding layers would first extract any low-level spectral-specific features, and a subsequent series of common encoding layers would then extract more general, higher-level features.

In order to test this proposed approach, a further four versions of YOLOv3 were created, each with varying degrees of spectral domain-dependent encoding, as shown in Figure 6-7.

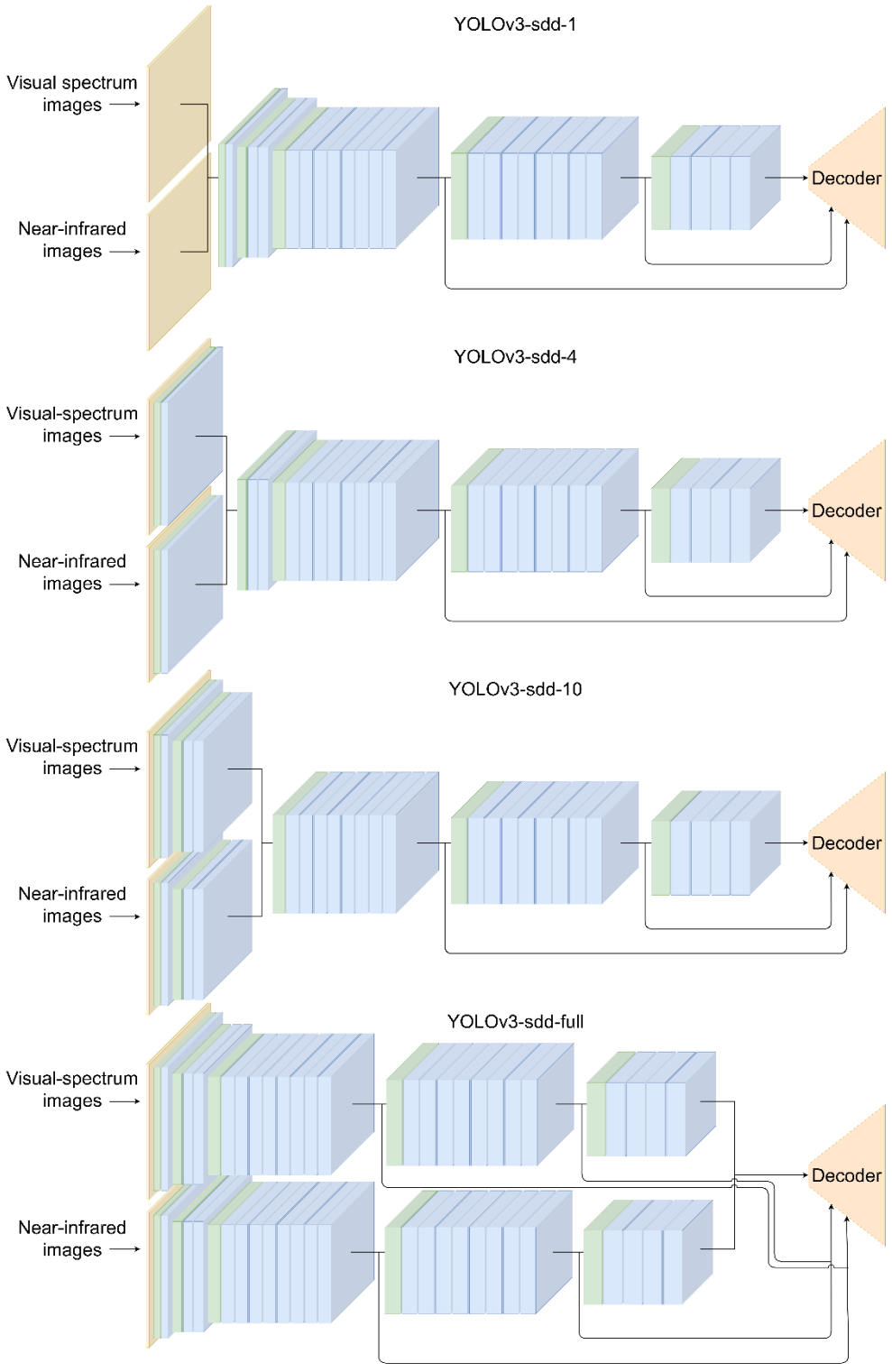


Figure 6-7: *The four different ways in which spectral domain-dependant encoding was applied to the YOLOv3 algorithm. Spectral domain-dependant encoding was applied to the first convolutional layer, the first four convolutional layers, the first 10 convolutional layers, and finally to all of the encoder's convolutional layers.*

For the first version, the algorithm was given just a single convolutional layer with which to encode visual-spectrum and near-infrared images separately, after which all examples were processed by a common encoding and decoding layers. This model is subsequently referred to as YOLOv3-sdd-1.

In the second version, the extent of spectral domain-dependent encoding was increased to include the first down-scaling convolutional layer and the first residual block. This equates to a total of four separated convolutional layers, so this model is subsequently referred to as YOLOv3-sdd-4.

The third version extended this further by including the second down-scaling convolutional layer and the subsequent two residual blocks. This equates to a total of 10 separated convolutional layers, so this model is subsequently referred to as YOLOv3-sdd-10.

Finally, in the fourth version, the concept of spectral domain-dependent encodingwas extended to its logical conclusion by fully separating all of the encoding layers. This model is subsequently referred to as YOLOv3-sdd-full.

Each of these four versions was trained for a total of 50 epochs with a constant learning rate of 1×10^{-6}. During training, the algorithms received batches of b images, some of which were from the visual-spectrum, with the rest being from the near-infrared domain. The algorithms also received positional vectors of length b that indicated the spectral domain of each image. Visual-spectrum images and near-infrared images were then grouped into two separate tensors and inputted into their respective encoding layers. The resulting feature maps were then re-combined into a single tensor—while preserving their original order within the batch—before being processed by the remainder of the network.

6.5.2. Visual-spectrum test results

Figure 6-8 shows the precision-recall curves for each model with respect to the visual-spectrum test data, and Table 6-2 summarises the improvement of each

model with respect to the detection accuracy reported previously in Section 6.4.1.

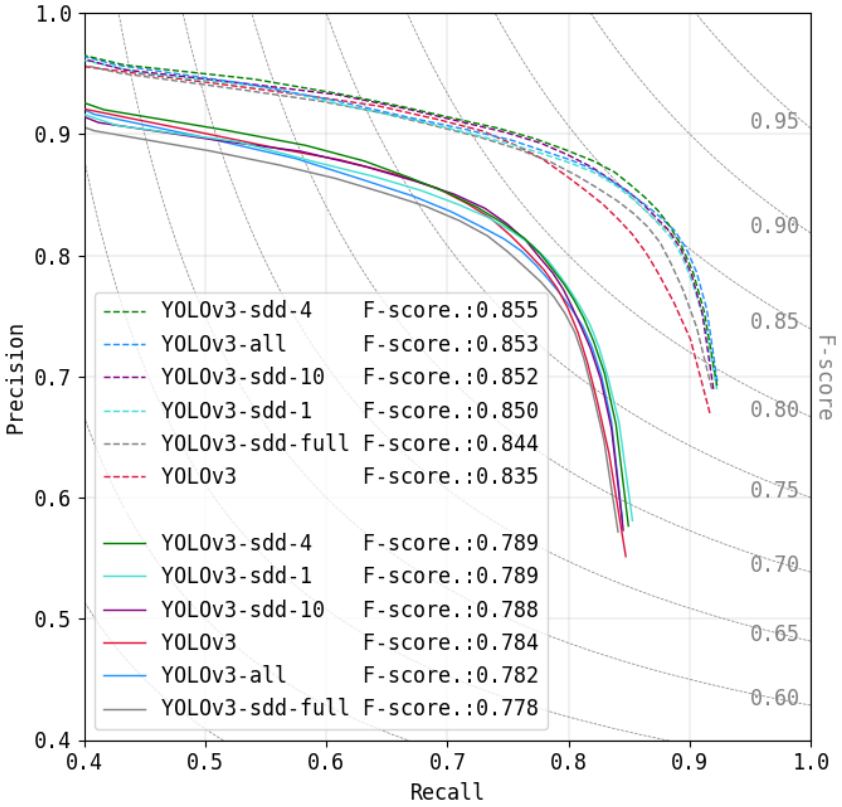


Figure 6-8: *Precision-recall curves for each YOLOv3 model with respect to the visual-spectrum test data. Solid lines correspond to an IoU threshold of 0.5, and dashed lines to an IoU threshold of 0.3.*

Table 6-2: *The improvement in maximum F-score for each of the YOLOv3 models which were trained using both visual-spectrum and near-infrared data, relative to YOLOv3 model which was trained using visual-spectrum examples only.*

Model	**Improvement (percentage points)**	
	IoU threshold = 0.3	**IoU threshold = 0.5**
YOLOv3-all	1.8	-0.2
YOLOv3-sdd-1	1.5	0.5
YOLOv3-sdd-4	**2.0**	**0.5**
YOLOv3-sdd-10	1.7	0.4
YOLOv3-sdd-full	0.9	-0.6

As can be seen, at an IoU threshold of 0.3, the addition of near-infrared examples to the training data consistently improved detection accuracy in the visual domain. Even with no architectural changes, the algorithm's maximum F-score increased by 1.8 percentage points. However, an even greater performance increase of 2.0 percentage points was achieved when four layers of domain-specific encoding were included.

In the case of an IoU threshold of 0.5, detection accuracy did not always improve. In fact, the algorithm's maximum F-score decreased by 0.2 percentage points when no architectural changes were applied, and decreased by 0.6 points when domain-specific encoding was extended to all encoding layers. However, when domain-specific encoding was applied to the first 1, 4, or 10 encoding layers, performance consistently increased by between 0.4 and 0.5 percentage points.

Figure 6-9 shows bounding box predictions from YOLOv3 and YOLOv3-sdd-4 with respect to a frame of visual-spectrum test data containing 12 ships. In this example, both models correctly detected the severely occluded vessel to the left-hand-side of the scene, and neither proposed any false positive predictions. However, by failing to detect the two overlapping ships also on the left-hand-side, YOLOv3 reported two false negative detections. This is in contrast to YOLOv3-sdd-4, which was able to correctly detect one of these overlapping ships, and therefore reported just a single false negative detection.

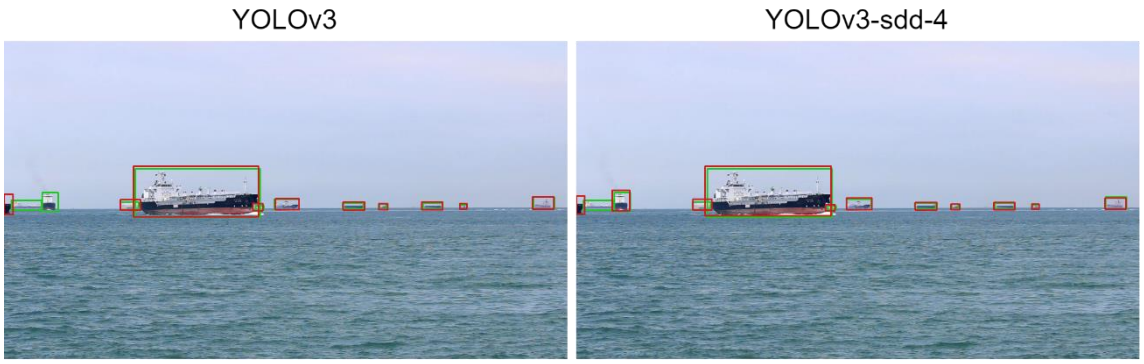


Figure 6-9: *Bounding box predictions by YOLOv3 and YOLOv3-sdd-4 (drawn in red) and ground truth labels (drawn in green) for a frame of visual-spectrum test data.*

Overall, the greatest improvement in detection accuracy was achieved by YOLOv3-sdd-4, which achieved maximum F-scores of 0.855 (precision = 0.847 and recall = 0.863) and 0.789 (precision = 0.801 and recall = 0.777) at IoU thresholds of 0.3 and 0.5 respectively. Figure 6-10 compares the detection accuracy of this improved version of YOLOv3 with Faster R-CNN and Mask R-CNN, showing that YOLOv3-sdd-4 now matches or exceeds the detection accuracy of Faster R-CNN at both IoU thresholds considered.

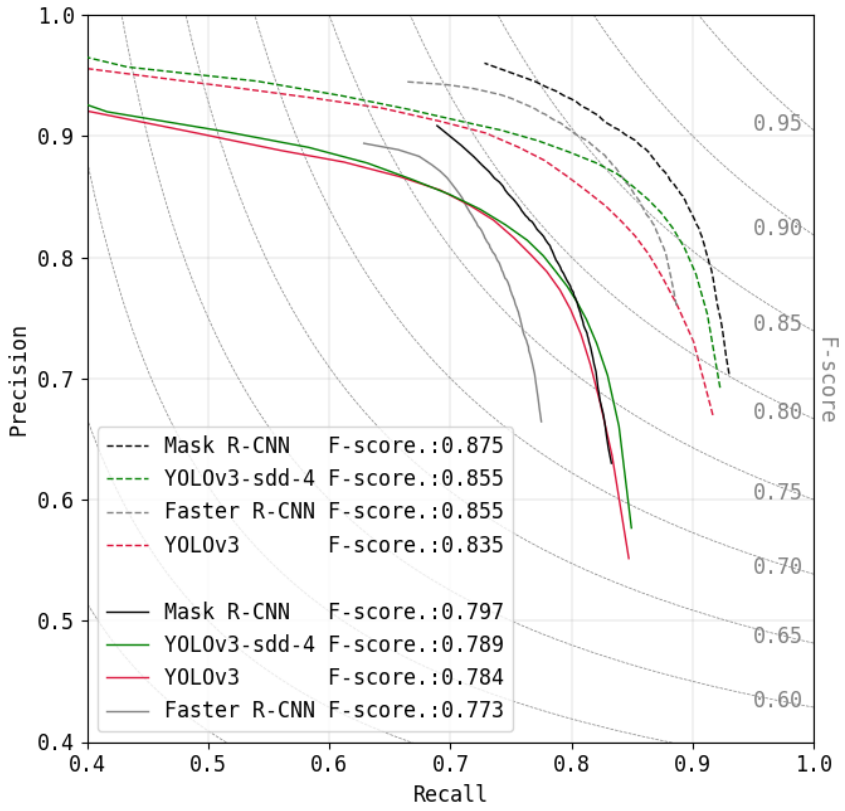


Figure 6-10: *Precision-recall curves for the best representatives for each approach with respect to the visual-spectrum portion of the SMD test data. Dashed curves represent results for an IoU threshold of 0.3 and solid curves for 0.5. (Results for MRCNN and FRCNN are sourced from [125].)*

These results for YOLOv3-all and the four modified versions of YOLOv3 are also tabulated in Appendix A-13.

6.5.3. Near-infrared test results

Figure 6-11 shows the precision-recall curves for each model with respect to the near-infrared test data, and Table 6-3 summarises the improvement of each model with respect to the detection accuracy observed in Section 6.4.1.

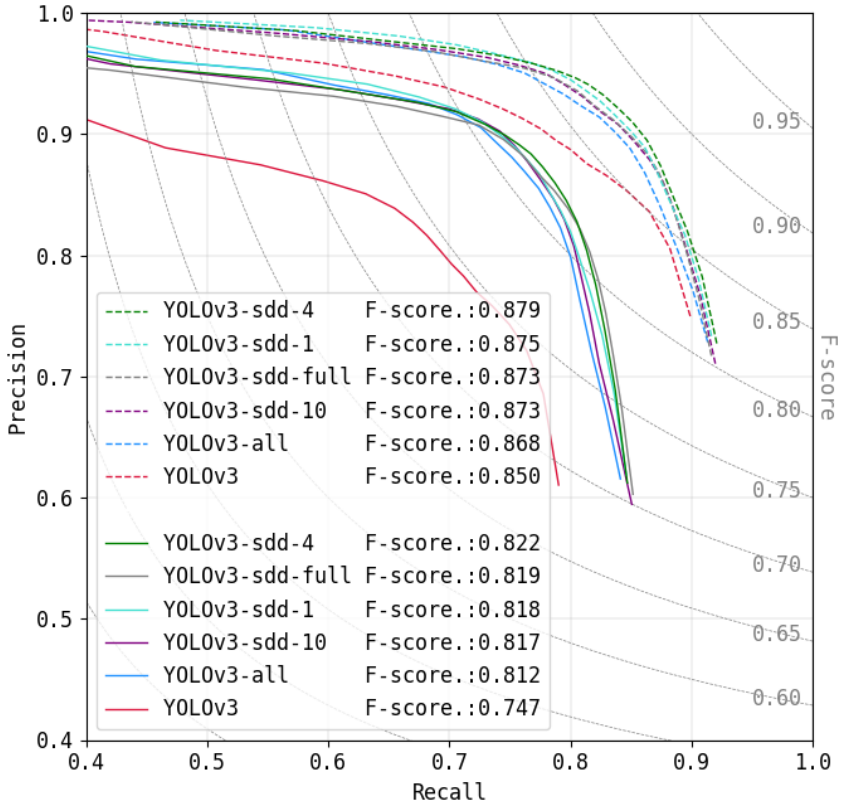


Figure 6-11: *Precision-recall curves for each YOLOv3 model with respect to the near-infrared test data. Solid lines correspond to an IoU threshold of 0.5, and dashed lines to an IoU threshold of 0.3.*

Table 6-3: *The improvement in maximum F-score for each of the YOLOv3 models which were trained using both visual-spectrum and near-infrared data, relative to YOLOv3 model which was trained using near-infrared examples only.*

Model	**Improvement (percentage points)**	
	IoU threshold = 0.3	**IoU threshold = 0.5**
YOLOv3-all	1.8	6.5
YOLOv3-sdd-1	2.5	7.1
YOLOv3-sdd-4	**2.9**	**7.5**
YOLOv3-sdd-10	2.3	7.0
YOLOv3-sdd-full	2.3	7.2

The addition of visual-spectrum images to the training data invariably improved the algorithm's detection accuracy with respect to the near-infrared test data. For YOLOv3-all, in which the algorithm's architecture was left unchanged, the

maximum F-score increased by 1.8 percentage points at an IoU threshold of 0.3, and by 6.5 percentage points at an IoU threshold of 0.5.

However, the increase in detection accuracy achieved by YOLOv3-all was surpassed by all other versions of YOLOv3 which used spectral domain-specific encoding. Overall, the most accurate version was YOLOv3-sdd-4, which achieved a maximum F-score of 0.879 (precision = 0.910 and recall = 0.851) at an IoU threshold of 0.3, and 0.822 (precision = 0.873 and recall = 0.777) at an IoU threshold of 0.5. This equates to an improvement of 2.9 percentage points at an IoU threshold of 0.3, and of 7.5 percentage points at an IoU threshold of 0.5.

This increase in detection performance is visualised in Figure 6-12, which shows bounding box predictions made by the original YOLOv3 and YOLOv3-sdd-4 with respect to a frame of near-infrared test data. As can be seen, YOLOv3 failed to infer an accurate bounding box for the sailboat at the centre of the frame and proposed a false-positive prediction to the left of the image. In contrast, YOLOv3-sdd-4, was able to accurately infer the sailboat's bounding box, and did not propose any false-positive predictions.

Figure 6-12: *Bounding boxes predictions (drawn in red) from YOLOv3 and YOLOv3-sdd-4 and ground truth labels (drawn in green) for a frame of near-infrared test data.*

Figure 6-13 compares the precision-recall curve YOLOv3-sdd-4 with that of Mask R-CNN and the original YOLOv3 model trained with near-infrared data only. Due to the application of spectral domain-specific encoding, this version of YOLOv3 achieved a detection accuracy which exceeds that of Mask R-CNN at

both IoU thresholds considered. Furthermore, this considerable performance increase was delivered while preserving the YOLOv3 algorithm's significant advantage in inference speed.

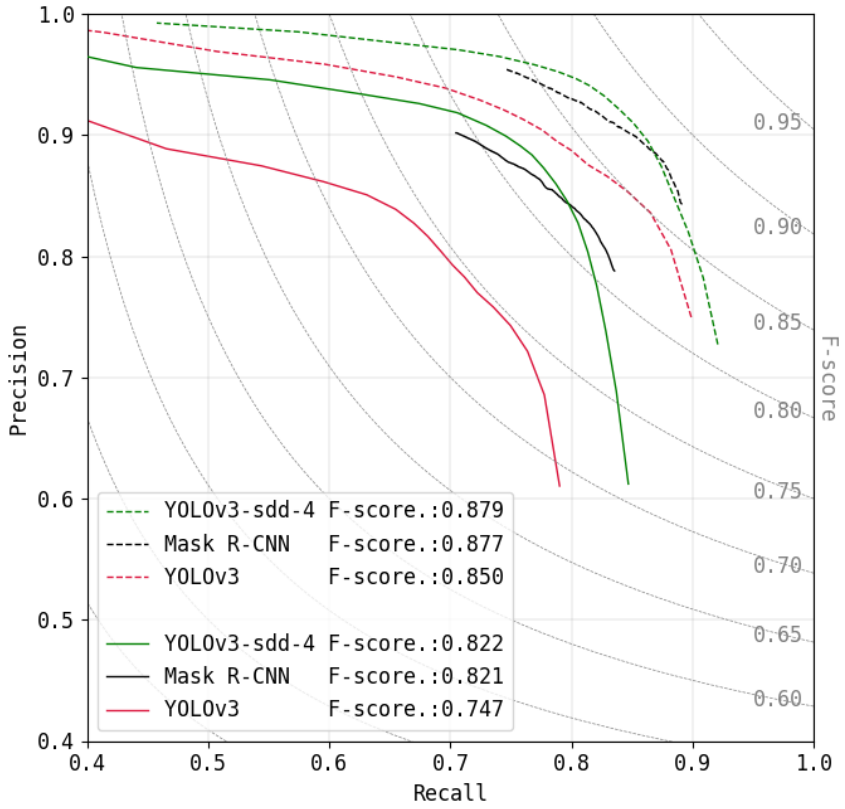


Figure 6-13: *precision-recall curves for the best representatives for each approach with respect to the visual-spectrum portion of the SMD test data. Dashed curves represent results for an IoU threshold of 0.3 and solid curves for 0.5. (Results for MRCNN are sourced from [125].)*

These results for YOLOv3-all and the four modified versions of YOLOv3 are also tabulated in Appendix A-14.

6.6. Conclusion

When developing missile seeker algorithms, inference speed and detection accuracy are generally in contention: greater inference speed comes at the expense of lower levels of detection accuracy, and vice versa.

Typically, single-stage algorithms prioritise inference speed; two-stage algorithms, on the other hand, generally offer superior detection accuracy, but

at a speed that is rarely considered adequate for real-time applications. Consequently, single-stage CNNs, such as YOLOv3, have conventionally been regarded as being better suited for missile seeker applications than their two-stage counterparts.

Benchmarking the single-stage YOLOv3 algorithm against the Faster R-CNN and Mask R-CNN algorithms using the Singapore Maritime Dataset supported this contention. Although much faster, the YOLOv3 algorithm could not match the superior detection accuracy of Mask R-CNN, the better of these two two-stage algorithms. In the visual domain, its maximum F-score averaged 2.7 percentage points lower than that of Mask R-CNN, and in the near-infrared domain, it averaged 5.1 percentage points lower. It was, however, fast: in a comparison of inference speeds, YOLOv3 achieved an average speed of 24 frames per second, outperforming Mask R-CNN by a factor of three.

Supplementing the YOLOv3 algorithm's performance through spectral domain-dependent encoding transformed this paradigm, improving the algorithm's detection accuracy in the visual domain by an average of 1.3 percentage points, and improving its accuracy in the near-infrared domain by an average of 5.2 percentage points.

Importantly, this resulted in a maximum F-score of 0.879 with respect to the near-infrared test data, surpassing that of Mask R-CNN and representing a new state-of-the-art for near-infrared ship detection: a single-stage algorithm delivering two-stage performance, entailing no significant reduction in algorithm inference speed, and showing how a single detection algorithm can switch between spectral domains without compromising detection accuracy.

Chapter 7

7. General Discussion and Conclusion

The four technical chapters presented in this thesis contribute a number of improvements to the state of the art of infrared anti-ship missile seeker algorithm. They do so by addressing five key challenges:

1. Developing a large, diverse, and thermally-representative LW infrared training dataset.
2. Developing a flexible, high-capacity, and high-speed modular deep learning development environment.
3. Selecting, implementing, and training an appropriate detection algorithm.
4. Controlling against overfitting: learning better features for greater accuracy.
5. Two-stage accuracy at one-stage speed: breaking the speed versus accuracy paradigm.

The following sections contextualise the findings of this thesis in terms of their real-world potential, addressing each challenge in turn.

7.1. Developing a thermally-representative LW infrared training dataset

Modern convolutional neural networks represent the state of the art for image-based object detection. Yet a lack of suitable training data has prevented algorithm developers from fully exploiting the advanced capabilities of these algorithms. Although numerous publicly-available datasets contain ships, none of these contain sufficient LW infrared imagery of military vessels so as to effectively train a modern CNN-based object detector. Therefore, the first challenge addressed by the body of work described in this thesis was the development of a suitably large, diverse, and thermally-representative LW infrared training dataset.

This challenge was overcome with the creation of IRShips, a synthetically-generated LW infrared training dataset containing 972,000 images of 10 ships—9 military and 1 civilian—depicted from an extensive range of viewpoints. Believed to be the largest and most diverse infrared dataset of military ships in existence at the present time, IRShips is also the first publicly-available dataset designed specifically for training CNNs to detect military ships in LW infrared imagery. Moreover, with nine unique thermal signatures for each vessel and an online augmentation pipeline for superimposing different sea-states, sky-states, and background clutter into the images, IRShips is also believed to be the most complex and thermally-realistic dataset of military ships to be generated synthetically.

7.1.1. Impacts of IRShips

1. IRShips is contributing to the development of LW infrared seeker algorithms.

Since its open-access publication in November 2020, IRShips has generated significant interest, and as of December 2021 had been downloaded over 8,000 times across 20 different countries. This indicates that there is considerable demand for a LW infrared dataset of military ships, and that IRShips fulfils this demand as intended. Moreover, it also indicates that IRShips is currently facilitating multiple independent efforts around the globe to improve the effectiveness of infrared anti-ship missile seekers. It can therefore be assumed that IRShips' three key dataset properties of size, diversity, and thermal representativeness—as extensively described in Chapter 3—are making a useful contribution to the development of infrared anti-ship missile seekers algorithms.

To expand, the thermal representativeness delivered by IRShips ensures that its constituent images contain many of the abstract features that are present in real-world imagery of engagement scenarios. Such abstract features encode information about the depicted scene, and when recognized by a seeker algorithm, improve the accuracy with which target vessels can be detected. As a result, IRShips enables the development of algorithms which can successfully exploit these thermal features in order to maximise detection accuracy.

IRShips further improves algorithms' performance by meeting the rule of thumb suggested by Goodfellow, Bengio and Courville in respect of what constitutes an adequate size for an image dataset. This, as far as is known, has never so far been met for LW data of military ships, and so by using IRShips developers can be confident that their algorithms are trained on 'enough' data. Moreover, the image diversity provided by IRShips ensures that algorithms that are trained on its data are "useful" in real-world settings.

Altogether, the improvements delivered by IRShips result in missile seeker algorithms that:

1. can be trained to recognize more types of ships,
2. can be trained to identify more classes of ships,
3. can prioritise multiple targets based on their type and class,
4. use label data from IRShips to learn to perform pixel-wise target segmentation,
5. use detailed thermal information from IRShips to learn aim-point refinement strategies, and
6. use metadata from IRShips, such as ship range, bearing, and the angle of elevation in order to further refine confidence in target accuracy.

2. IRShips should influence the development of future synthetic datasets.

Together with the body of work described in Chapter 3 of this thesis, IRShips advances the state of the art of infrared image development and augmentation as it applies to LW infrared datasets of military—as well as civilian—vessels. This enables several fundamental limitations to the current state of LW infrared data generation to be overcome.

In terms of the development of LW infrared datasets of maritime vessels, IRShips represents advances in:

- *the sourcing and adaptation of suitably realistic ship CAD models,*
- *the procedural generation of suitably representative ship thermal properties,*
- *the automatic application of thermal properties to ship CAD models, and*
- *the depiction of diverse sea-states, sky-states, and background clutter.*

It is therefore anticipated that the creation of IRShips together with its open-access publication will facilitate the development of future LW infrared maritime datasets.

7.1.2. Future work

Six important directions for future work suggest themselves. These include improvements in image diversity, the number of identifiable thermal signatures, the diversity of recognizable environmental conditions, the order in which image data is implemented, image augmentation through the addition of decoys, and even the software used to generate the data.

IRShips contains more military ships than any other publicly-available LW infrared dataset, enabling trained seeker algorithms to recognize and identify up to 4 different ship types and 10 different ship classes. But this represents only a fraction of the various types and classes of military vessels currently in active service around the globe, and therefore omits many of the vessels that an anti-ship missile could potentially encounter. The addition of a greater diversity of ship types and classes would therefore allow future seeker algorithms to not only recognize and identify an increased range of military ships, but also improve their general detection accuracy in the presence of unfamiliar vessels.

IRShips also represents the first publicly-available synthetically-generated infrared maritime dataset to contain multiple different thermal signatures, with nine different thermal representations for each ship it contains. Enhancing IRShips—or any other future LW infrared maritime dataset—through the addition of further thermal representations should contribute significantly to its effectiveness as a training dataset.

There is also considerable scope for the future development of IRShips' data augmentation pipeline. At present, this pipeline relies on 66 real-world visual spectrum images of various sea-states, sky-states, and background clutter, which are superimposed onto synthetically-generated infrared images. It is unlikely that these 66 images constitute a fully-representative sample of the conditions that a missile might encounter. Consequently, incorporating

additional real-world images would increase the diversity of IRShips, and in turn increase its representativeness of real-world situations.

A further improvement, of course, would be to either replace these real-world visual spectrum images of various sea-states, sky-states, and background clutter with authentic LW infrared imagery, or augment these real-world visual spectrum images through the addition of LW infrared imagery of various sea-states, sky-states, and background clutter. This would increase both the diversity and the thermal-representativeness of IRShips and similar maritime datasets, further enhancing missile seeker algorithms' training and subsequent performance.

In addition, the data augmentation process could be extended to generate images containing multiple ships, ships occluded by foreground objects, and even infrared countermeasures. This again could be expected to enhance seeker algorithms' performance.

Finally, given the provision of enough appropriate infrared image data, and given that the state of the art of Generative Adversarial Networks is reaching an appropriate level of maturity, the potential of Generative Adversarial Networks could be investigated, either as a method for enhancing the complexity and thermal-representativeness of IRShips data, or as an alternative method of synthetic data generation. Such algorithms have the potential to generate unending quantities of diverse and thermally-representative LW infrared training data which, once again, would enhance the performance of seeker algorithms.

7.2. Developing a deep learning development environment

Convolutional Neural Networks addressing complex tasks such as object detection typically contain tens of millions of internal parameters, all of which must successfully interoperate together. Optimising these parameters necessitates the development of multiple and precisely-designed supporting components, assembled together into an efficient and high-capacity training pipeline.

Consequently, implementing a CNN-based solution from scratch can call for a considerable investment of skill, time, and effort. Consequently, the second challenge addressed by the body of work described in this thesis was development of a flexible, high-capacity, and high-speed modular deep learning development environment.

This challenge was overcome with the implementation of Deeplodocus, a research-oriented software development environment that accelerates the initialisation of CNN-based projects and promotes the rapid prototyping of potential solutions. With a modular and flexible design, Deeplodocus enables both bespoke and third-party software modules to be incorporated into distinct and individual training, validation, and test pipelines. With an efficient design that permits fast and effective training, Deeplodocus' structured approach to project organisation promotes collaboration and reproducibility.

Throughout the body of work described in this thesis, Deeplodocus was instrumental in the implementation, testing, and development of multiple CNN-based models. Deeplodocus has also contributed towards several external research projects, including the development of a bespoke CNN for infrared footprint segmentation, the implementation of SegNet for the semantic segmentation of urban scenes; and the development of a CNN for distance, surface normal, and optical flow estimation. Most recently, Deeplodocus has been chosen to facilitate the development of ATR algorithms on behalf of multinational defence company Leonardo.

7.2.1. Impacts of Deeplodocus

1. *Deeplodocus prompts the sharing and collaboration of open-access deep learning research.*

Compartmentalised and modular in nature, Deeplodocus forces users to structure their projects in a standardised format. Outputting in a clear and consistent manner means that projects can be more easily published, understood, and run by other researchers, which in turn aids reproducibility.

2. *Deeplodocus permits more time to be allocated to project design and solution extension.*

An additional benefit of the modular structure is that new projects can be quickly initialised and implemented, with pre-existing modules re-used, and new modules easily integrated into the existing structure. The time saved can then be allocated to project design and solution extension without adversely impacting overall project duration.

3. *Deeplodocus enables the rapid testing of solutions.*

The computational efficiency of Deeplodocus allows algorithms to be trained faster, thus enabling more solutions, hypotheses, and training conditions to be tested in the time available.

4. *Deeplodocus' automatic retention of project environments and parameters aids model repeatability and extension.*

The automated retention of the configurations, inputs, conditions, processes, and results of each experiment facilitates repeatability, accelerates model validation, and aids solution communication and analysis. The fact that this is automatic, rather than manual, aids researcher productivity.

7.2.2. Future work

Usability, modularity, and flexibility are the most important features of Deeplodocus, and any subsequent development should therefore focus on extending these features.

1. *Enabling the use of bespoke trainer modules.*

Deeplodocus enables users to easily assemble deep learning solutions using pre-existing or bespoke modules. However, as presently configured, the user has a limited ability to vary the exact order of module computation during algorithm training, thus limiting the potential to implement and investigate new training techniques. Consequently, any future work to improve Deeplodocus should consider extending its modularity to support the inclusion of bespoke "trainer" modules, in order to increase the level of flexibility that Deeplodocus provides.

2. *Providing support for multi-model algorithms.*

Deeplodocus supports the use of multiple loss functions, multiple evaluation metrics, multiple datasets, and even multiple GPUs, yet can train just one deep learning model at a time. This is adequate for many deep learning applications, but not, however, for the development of Generative Adversarial Networks, which consist of two deep learning models: a generator, and a discriminator [270]. These networks increasingly represent the state of the art in many computer vision applications, and it is therefore important that future versions of Deeplodocus support the training and development of these algorithms.

3. *Improving usability with an intuitive Graphical User Interface.*

Currently, users of Deeplodocus can construct complex deep learning pipelines via its high-level configuration files. However, completing all of the necessary fields within these files can be time consuming, and potentially subject to error. Therefore, future development of Deeplodocus could consider the provision of an intuitive Graphical User Interface (GUI). By providing real-time visual feedback, drag-and-drop placement of component modules, and multiple-choice dialogue, such a GUI would significantly improve the usability of Deeplodocus.

7.3. Selecting, implementing, and training a detection algorithm

The third challenge addressed by the body of work described in this thesis was to select, implement, and train an appropriate object detection algorithm using the IRShips training dataset.

Due to its superior speed and competitive detection accuracy, YOLOv3 was identified as the most appropriate object detection algorithm. It was subsequently incorporated into an efficient training and validation pipeline using Deeplodocus, trained using the IRShips dataset, and then evaluated using a sequence of complex LW infrared imagery. Despite being trained solely with computer-generated imagery, the YOLOv3 algorithm was able to accurately

detect a Karel Doorman-class frigate in a densely-cluttered sequence of real-world LW infrared imagery, achieving a maximum F-score of 0.818.

This is believed to represent the first time that a synthetically-generated dataset was used to train a CNN-based object detection algorithm to detect military vessels in LW infrared imagery.

7.3.1. The impact of training with IRShips

1. *The proven impact of augmented synthetic data on training effectiveness should lead to its re-use in similar object detection problems.*

Data augmentation was shown to be crucial to the effective development of IRShips. This finding identifies data augmentation as a primary focus for similar future work involving object detection in complex and demanding settings.

2. *The proven success of Convolutional Neural Networks in a missile seeker application should lead to the further exploration of Convolutional Neural Networks in similar future work.*

Prior to the body of work described in this thesis, Convolutional Neural Networks had not been extensively applied to LW infrared military ship detection. This thesis shows that Convolutional Neural Networks not only have application in this context, but also the promise to significantly improve the detection accuracy performance of missile seekers.

3. *The training experiments described here provide a glimpse into the future intelligence of missile seekers.*

At present, anti-ship missile seekers are only thought to have been trained on low-complexity at-sea data, with relatively little background clutter. In the real world, missile seekers need to be able to hunt targets in coastal, littoral, or harbour locations, or in situations where other non-target vessels provide noise. The body of work described in this thesis demonstrates that using Convolutional Neural Networks to detect target vessels in these complex scenarios is possible.

7.3.2. Future work

Future work to train, develop, and evaluate CNN-based seeker algorithms using IRShips could include:

1. *The training and evaluation of new cutting-edge object detection algorithms.*

General object detection is a fast-paced field of research, and a number of improved network architectures, such as YOLOv4 [271] and MRFPN [272] and SWIN Transformer v2 [273], have emerged since the experiments described in this thesis were concluded. This is not to suggest that the work described in this thesis is outdated, merely that new general object-detection algorithms have emerged which may be shown to have applicability to missile seeker applications. If such corroboration is sought, then the body of work described in this thesis—Deeplodocus, data augmentation, IRShips, the demonstration of successful strategies for mitigating against overfitting, and the identification of a means of spectral domain-dependent encoding—should significantly reduce the time taken to demonstrate such corroboration.

2. *Developing object-tracking capabilities.*

This thesis has focused on the object detection, object recognition, and object identification capabilities of anti-ship seeker algorithms. However, effective seeker algorithms should provide object-tracking capabilities, as well. By preserving object identities and continuing to predict the location of objects in the event of target occlusion or detection algorithm failure, object-tracking capabilities could significantly improve missile seeker performance, particularly in adverse conditions. Therefore, future work could consider extending the CNN-based detection algorithms developed within this thesis by adding object-tracking capabilities.

3. *Evaluating with a larger and more diverse set of real-world LW infrared data.*

This thesis used a sequence of 940 densely-cluttered LW infrared images of a Karel Doorman-class frigate to evaluate the detection accuracy of the YOLOv3 algorithm. Significantly, this sequence of images is believed to so far represent

the most complex and challenging test data reported in the academic literature for the evaluation of an infrared anti-ship seeker algorithm.

However, there remains considerable scope to increase the quantity and diversity of this test data further. Additional test data covering a wider range of environmental conditions, viewpoints, and target ships will be crucial for developing CNN-based seeker algorithms that can be expected to operate safely in real-world conditions.

7.4. Controlling against overfitting

After training with the IRShips dataset, YOLOv3 was able to accurately detect military ships in real-world LW infrared imagery. However, analysis of the algorithm's training and validation results suggested that it was overfitting to the synthetic training data, and that controlling against this overfitting could significantly improve the algorithm's detection accuracy. Therefore, the next challenge addressed by the body of work described in this thesis was to develop techniques that mitigate against overfitting in order to deliver greater detection accuracy.

This was achieved through the development of a multi-task training technique. With modifications to the algorithm's architecture and the addition of just 8,343 easily-sourced semi-labelled visual-spectrum images to the training dataset, YOLOv3 was trained for both object detection and image classification simultaneously. This caused the algorithm's encoder to learn more general—and therefore more effective—image features, thereby reducing the effect of overfitting, and significantly increasing the algorithm's peak F-score from 0.818 to 0.945.

7.4.1. Impacts of mitigating against overfitting

1. *It has been demonstrated that synthetic data can be readily augmented through the addition of easily-sourced real-world data to reduce overfitting.*

The fact that synthetic data can be so effectively augmented through the use of easily-sourced real-world data, can be expected to strengthen the case among

researchers that the use of synthetic data is a valid and viable means of developing training data. Given that objection detection problems may often include images where real-world photography and its subsequent labelling is difficult, expensive, or time-consuming, then this demonstration in the context of military ships provides a valuable endorsement of the value of synthetic data.

7.4.2. Future work

Overfitting presents an acute challenge when training CNNs with synthetic data. Future work to build on the contribution of this thesis could include:

2. *Extension of the multi-task learning approach to make full use of all available data.*

This thesis has demonstrated that, through a multi-task learning approach, the overfitting of YOLOv3 to IRShips can be significantly reduced with just 8,343 easily-sourced real-world visual-spectrum images of ships. However, there are several open-access maritime datasets—such as MARVEL [153] and SeaShips [165]—that provide considerably more examples of similar images. Therefore, future work could consider extending the multi-task learning approach presented in this thesis in order to capitalise on this data.

3. Further analysis to identify important features which are not present in IRShips.

The fact that an additional 8,343 images proved useful in improving algorithm detection accuracy indicates that these real-world images contain many useful features that are not present in the synthetically-generated IRShips data. Therefore, future work should consider analysing the features learned by YOLOv3 in order to determine what these missing features are, and how they could be included in future synthetic datasets.

7.5. Breaking the speed versus accuracy paradigm

YOLOv3 has an efficient single-stage architecture, which delivers higher inference speeds relative to two-stage algorithms, such as Mask R-CNN. Despite its superior speed, however, YOLOv3 could not match the detection

accuracy of Mask R-CNN, which outperformed it by up to 7.4 percentage points with respect to the Singapore Maritime Dataset. This represented an unfortunate trade-off between two characteristics—speed and accuracy—which are essential to any effective missile seeker algorithm. Therefore, the final challenge addressed by the body of work described in this thesis was to break this speed versus accuracy paradigm in order to deliver an algorithm that achieved two-stage accuracy at single-stage speeds.

This was delivered through the development of a spectral domain-dependent encoding technique, which significantly increased the detection accuracy of YOLOv3 without affecting its inference speed. Available data from two distinct spectral domains were combined into single training dataset, and separate domain-dependent layers were introduced into YOLOv3's encoder. As a result, YOLOv3 more efficiently leveraged data from an alternative spectral domain during training, thereby increasing its detection accuracy by up to 2.0 percentage points in the visual domain, and up to 7.5 percentage points in the near-infrared domain.

As a result, the modified YOLOv3 algorithm achieved a detection accuracy surpassing that of Mask R-CNN in the near-infrared domain. This represents a new state of the art with respect to the Singapore Maritime Dataset, and constitutes a significant contribution to the field of maritime object detection.

7.5.1. The impacts of breaking the speed versus accuracy paradigm

1. *With the demonstration that the speed versus accuracy paradigm can be broken, future research need not be constrained by artificial ceilings on either.*

Spectral domain-dependent encoding enabled a fast single-stage object detector to achieve a level of detection accuracy that is comparable to a two-stage algorithm. This demonstrated that detection algorithms for missile seekers should not have to compromise between accuracy and speed.

2. It has been demonstrated that a single detection algorithm can operate effectively over multiple spectral domains.

Object detection algorithms—especially in autonomous systems—may need to synthesise data across images that have been collected from different cameras sensitive to different spectral domains. With spectral domain-dependent encoding, a single detection algorithm is able to switch between spectral domains without impacting the performance in either domain, often attaining increased detection accuracy. For researchers working in the field, this is an important finding.

3. It is likely that spectral domain-dependent encoding can be applied to LW infrared data.

This thesis has demonstrated that spectral domain-dependent encoding enabled YOLOv3 to learn effectively from visual-spectrum and near-infrared images simultaneously in order to deliver improved detection accuracy across both of these domains. However, this approach could be applied to a combination of two different spectral domains, such as visual and LW infrared, or even extended to accommodate visual-spectrum, near-infrared, and LW infrared training data. Since spectral domain-dependent encoding therefore has the potential to improve detection accuracy in the LW infrared domain, it may have an impact on the development of future LW infrared anti-ship missile seekers.

7.5.2. Future work

The future work stemming from this use of spectral domain-dependent encoding could include:

1. Extending this technique across further spectral domains, including LW infrared.

Spectral domain-dependent encoding has the potential to improve detection accuracy in the LW infrared domain, and future work could investigate this further.

2. *Applying this technique to other object detection algorithms, including Mask R-CNN.*

The spectral domain-dependent encoding approach developed in this thesis considerably increased the detection accuracy of YOLOv3, enabling it to outperform Mask R-CNN in the near infrared domain. However, this technique is not specific to YOLOv3 and can be applied to the encoder of any CNN-based object detector. Therefore, it may be appropriate for future work to investigate how spectral domain-dependent encoding affects the performance of other object detectors, and specifically, whether it yields similar improvements in the context of two-stage algorithms.

7.6. Conclusion

This thesis has shown that synthetic data can be used to train CNN-based object detection algorithms in infrared anti-ship missile seekers. In achieving this, a number of advances have been made:

1. The generation and open-access publication of a large, diverse, and thermally-realistic LW infrared dataset of military and civilian ships,
2. A modular, flexible, and high-capacity environment for the development of deep learning algorithms has been built,
3. Using synthetically-generated training data, a modern CNN-based object detection algorithm has been trained to detect, recognize, and identify military ships in real-world LW infrared imagery,
4. Methods to correct for the overfitting of object detection algorithms have been developed,
5. Through the use of spectral domain-dependent encoding, it has been demonstrated that a single detection algorithm can operate effectively over multiple spectral domains, and
6. It has been shown that the speed vs accuracy paradigm need not apply in the context of CNN-based object detection algorithms.

For researchers working on the next generation of intelligent missile seekers, and in the area of object detection generally, it is suggested that these advances will be of interest.

8. References

[1] White R. Harrier 809: Britain's Legendary Jump Jet and the Untold Story of the Falklands War. Penguin Random House UK; 2020.

[2] Southby-Tailyour E. Exocet Falklands: The untold story of special forces operations. Pen and Sword; 2014.

[3] Goldblat J, Millán V. In the Wake of the Falklands/Malvinas War: A New Cycle of the Arms Race. Bulletin of Peace Proposals. 1983;14(3):253-61.

[4] Finlan A. British special forces in the Falklands War of 1982. Small Wars and Insurgencies. 2002;13(3):75-96.

[5] Freedman L. The official history of the Falklands Campaign, Volume 2: War and diplomacy. Routledge; 2005.

[6] Nott J. Here Today, Gone Tomorrow: Recollections of an Errant Politician. Politicos Pub; 2002.

[7] Kuchler H. Thatcher gave Paris Falklands ultimatum. 2012 December.

[8] Grant RG. Battle at sea: 3,000 years of naval warfare. Penguin; 2011.

[9] Schulte JC. An analysis of the historical effectiveness of anti-ship cruise missiles in littoral warfare. Naval Postgraduate School, Monterey, CA; 1994.

[10] Shachtman N. Inside the Battle for the Black Sea | WIRED; 2008. (Accessed on 22/08/2021). https://www.wired.com/2008/08/while-the-media/.

[11] Russian navy sinks Georgian boat: Defence ministry | Reuters; 2008. (Accessed on 22/08/2021). https://www.reuters.com/article/newsOne/-idUSLA56070520080810.

[12] Bluth C. Hypersonic missiles are fuelling fears of a new superpower arms race; 2021. (Accessed on 30/12/2021). https://theconversation.com/hypersonic-missiles-are-fuelling-fears-of-a-new-superpower-arms-race-172716.

[13] Weapons: Naval - Sea Skua; 2020. (Accessed on 03/05/2021). https://-customer.janes.com/Janes/Display/JNWS0159-JNW_.

[14] Armes G. Fire At Will. Royal Navy; 2006. (Accessed on 23/07/2021). https://imagery.royalnavy.mod.uk/fotoweb/archives/5003-General%20Public%20Archive/Portsmouth_Archive/2015/March/04/-04122474.jpg.

[15] Weapons: Naval - RHM-84/UGM-84 Harpoon (GWS 60); 2020. (Accessed on 03/05/2021). https://customer.janes.com/Janes/Display/-JNWS0161-JNW_.

[16] Weapons: Strategic - DF-21; 2021. (Accessed on 10/05/2021). https://-customer.janes.com/Janes/Display/JSWS0411-JSWS.

[17] Swain D. MQ130002038. Royal Navy; 2013. (Accessed on 23/07/2021). https://imagery.royalnavy.mod.uk/fotoweb/archives/5003-General%20Public%20Archive/Portsmouth_Archive/2015/March/05/-MQ130002038.jpg.

[18] Alemayehu AY, Solomon LG. Design of a Solid Rocket Propulsion System. International Journal of Aeronautical Science & Aerospace Research. 2020;7(2):224-9.

[19] Satyaprasad P, Pandu Ranga Sharma M, Richarya A, Rolex Ranjit A, Subhash Chandran BS. In: Prasad NE, Wanhill RJH, editors. Missile Propulsion Systems. Singapore: Springer Singapore; 2017. p. 305-30. Available from: https://doi.org/10.1007/978-981-10-2143-5_15.

[20] Bezick SM, Pue AJ, Patzelt CM. Inertial navigation for guided missile systems. Johns Hopkins APL technical digest. 2010;28(4):331-42.

[21] Lechner W, Baumann S. Global navigation satellite systems. Computers and Electronics in Agriculture. 2000;25(1-2):67-85.

[22] Titterton D, Weston JL, Weston J. Strapdown inertial navigation technology. vol. 17. IET; 2004.

[23] Jackson PB. Overview of missile flight control systems. Johns Hopkins APL technical digest. 2010;29(1):9-24.

[24] Mracek CP. In: Baillieul J, Samad T, editors. Tactical Missile Autopilots. London: Springer London; 2015. p. 1459-65. Available from: https://doi.org/-10.1007/978-1-4471-5058-9_19.

[25] Singh J. Sea Skimmer Anti-Ship Missiles. Strategic Analysis. 1983;7(9):751-65.

[26] Pike J, Aftergood S. Naval Ships' Technical Manuals Fundamentals of Naval Weapons Systems - Chapter 13 - Warheads;. (Accessed on 03/06/2021). https://fas.org/man/dod-101/navy/docs/fun/part13.htm.

[27] Hampshire E. The Falklands Naval Campaign 1982: War in the South Atlantic. Bloomsbury Publishing; 2021.

[28] Weapons: Air Launched - ASM-2 (Type 93); 2020. (Accessed on 12/05/2021). https://customer.janes.com/Janes/Display/JALW2911-JALW.

[29] Weapons: Air Launched - ASM-3 (XASM-3)-series missiles; 2021. (Accessed on 13/05/2021). https://customer.janes.com/Janes/Display/-JALW3704-JALW.

[30] Weapons: Naval - BrahMos (PJ-10); 2021. (Accessed on 13/05/2021). https://customer.janes.com/Janes/Display/JNWSA010-JNW_.

[31] Weapons: Naval - Exocet MM38/SM39/MM40 (GWS 50); 2020. (Accessed on 12/05/2021). https://customer.janes.com/Janes/Display/-JNWS0137-JNW_.

[32] Weapons: Naval - Hsiung Feng I/II/IIB/IIE/IIER/III (HF-I, -II, -IIB, -IIE, -IIER, -III); 2021. (Accessed on 13/05/2021). https://customer.janes.com/Janes/-Display/JNWS0158-JNW_.

[33] Weapons: Naval - Naval Strike Missile (NSM); 2021. (Accessed on 12/05/2021). https://customer.janes.com/Janes/Display/JNWS0911-JNW_.

[34] Weapons: Naval - Penguin; 2021. (Accessed on 11/05/2021). https://-customer.janes.com/Janes/Display/JNWS0148-JNW_.

[35] Weapons: Air Launched - Sea Eagle; 2020. (Accessed on 11/05/2021). https://customer.janes.com/Janes/Display/JALW3052-JALW.

[36] Weapons: Naval - Standard Missile-6 (SM-6)/Extended Range Active Missile (ERAM); 2020. (Accessed on 03/05/2021). https://customer.janes.com/-Janes/Display/JNW_0076-JNW_.

[37] Weapons: Naval - CSS-N-4 'Sardine' (YJ-8/YJ-8A/C-801); CSS-N-8 'Saccade' (YJ-82/YJ-83/C-802/C-802A/Noor/Ghader); 2021. (Accessed on 13/05/2021). https://customer.janes.com/Janes/Display/JNWS0132-JNW_.

[38] Russia tests Zircon against naval target; 2020. (Accessed on 5/05/2021). https://customer.janes.com/Janes/Display/FG_3758013-JNI.

[39] Siouris GM. Missile guidance and control systems. Springer Science & Business Media; 2004.

[40] De Martino A. Introduction to modern EW systems, second edition. 2nd ed. Artech House; 2018.

[41] Planck M. The theory of heat radiation. Blakiston; 1914.

[42] Driggers RG, Friedman MH, Nichols J. Introduction to infrared and electro-optical systems. Artech House; 2012.

[43] Barducci A, Pippi I. Temperature and emissivity retrieval from remotely sensed images using the "Grey body emissivity" method. IEEE Transactions on Geoscience and Remote Sensing. 1996;34(3):681-95.

[44] Emissivity;. (Accessed on 01/06/2021). http://gsp.humboldt.edu/OLM/-Courses/GSP_216_Online/lesson8-1/emissivity.html.

[45] Emissitivity of Common Materials;. (Accessed on 01/06/2021). https://-www.omega.co.uk/literature/transactions/volume1/emissivitya.html.

[46] Emissivity Coefficient Materials;. (Accessed on 01/06/2021). https://-www.engineeringtoolbox.com/emissivity-coefficients-d_447.html.

[47] Emissivity Table;. (Accessed on 01/06/2021). https://-www.optotherm.com/emiss-table.htm.

[48] Pais A. Einstein and the quantum theory. Reviews of modern physics. 1979;51(4):863.

[49] Richardson MA, Luckraft IC, Picton RS, Rodgers AL, Powell RF. Surveillance and Target Acquisition Systems. vol. 4 of 3. 2nd ed. Brassey's (UK) Ltd; 1997.

[50] Voulodimos A, Doulamis N, Doulamis A, Protopapadakis E. Deep learning for computer vision: A brief review. Computational intelligence and neuroscience. 2018;2018.

[51] Garg R, Bg VK, Carneiro G, Reid I. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In: European conference on computer vision. Springer; 2016. p. 740-56.

[52] Jackson AS, Bulat A, Argyriou V, Tzimiropoulos G. Large pose 3D face reconstruction from a single image via direct volumetric CNN regression. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 1031-9.

[53] Vinyals O, Toshev A, Bengio S, Erhan D. Show and tell: A neural image caption generator. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2015. p. 3156-64.

[54] Hartley R, Zisserman A. Multiple View Geometry in Computer Vision. 2nd ed. Cambridge University Press; 2004.

[55] Gelman A, Carlin J. Beyond power calculations: Assessing type S (sign) and type M (magnitude) errors. Perspectives on Psychological Science. 2014;9(6):641-51.

[56] Visa S, Ramsay B, Ralescu AL, Van Der Knaap E. Confusion matrix-based feature selection. MAICS. 2011;710:120-7.

[57] Janes: Visby class; 2021. (Accessed on 01/09/2021). https://-customer.janes.com/Janes/Display/jfs_2970-jfs_.

[58] Janes: Nimitz (CVN 68) class; 2021. (Accessed on 20/09/2021). https://-customer.janes.com/FightingShips/Display/jfs_3526-jfs_.

[59] Janes: Arleigh Burke (DDG 51 Flight IIA) class; 2021. (Accessed on 20/09/2021). https://customer.janes.com/FightingShips/Display/jfs_3533-jfs_.

[60] Janes: Daring class (Type 45); 2020. (Accessed on 20/09/2021). https://-customer.janes.com/FightingShips/Display/jfs_4494-jfs_.

[61] Janes: Luyang III (Type 052D) class (DDGHM); 2021. (Accessed on 20/09/2021). https://customer.janes.com/FightingShips/Display/JFS_B684-JFS_.

[62] Janes fighting ships 2020-2021. 120th ed. Janes Information Group; 2020.

[63] Morozhenko V. Infrared Radiation. BoD–Books on Demand; 2012.

[64] Janes: Weapons: Naval - Mk 15 close-in weapon system (Phalanx); 2020. (Accessed on 20/09/2021). https://customer.janes.com/Janes/Display/-JNWS0269-JNW_.

[65] Aster 15/30; 2013. (Accessed on 10/05/2021). https://-customer.janes.com/Janes/Display/jsws0163-jsws.

[66] Wachsberger C, Lucas M, Krstic A. Limitations of Guns as a Defence against Manoeuvring Air Weapons. Defence Science and Technology Organisation, Australian Government Department of Defence; 2004.

[67] Janes Weapons: Naval 2019-2020. 60th ed. IHS Janes; 2019.

[68] Aircraft-carriers are under threat from modern missiles. The Economist; 2019. (Accessed on 15/04/2021). https://www.economist.com/leaders/2019/11/-14/aircraft-carriers-are-under-threat-from-modern-missiles?giftId=af8f93b8-fee0-4a63-8b78-0718d4a382da.

[69] Channeling Streetfighter? The PLAN's Houbei FAC. U.S. Naval Institute Blog; 2011. (Accessed on 15/04/2021). https://blog.usni.org/posts/2011/08/02/-channeling-streetfighter-plans-houbei-fac.

[70] Hughes R. Jane's Missiles and Rockets. IHS Janes; 2019.

[71] Sampson B. US Air Force successfully tests anti-missile laser defence system | Aerospace Testing International. Aerospace Testing International; 2019. (Accessed on 13/04/2021). https://-www.aerospacetestinginternational.com/news/defense/us-air-force-successfully-tests-anti-missile-laser-defence-system.html.

[72] Hughes R. Janes: Missile shootdown marks programme milestone for AFRL's SHiELD; 2019. (Accessed on 20/09/2021). https://customer.janes.com/-Janes/Display/FG_1952447-JMR.

[73] Scott R, Rosamond J. Janes: An ever-evolving game of deception and protection; 2010. (Accessed on 20/09/2021). https://customer.janes.com/-Janes/Display/jni74153-jni-2010.

[74] Gray GJ, Aouf N, Richardson MA, Butters B, Walmsley R, Nicholls E. Feature-based tracking algorithms for imaging infrared anti-ship missiles. In: Technologies for Optical Countermeasures VIII. vol. 8187. International Society for Optics and Photonics; 2011. p. 81870T.

[75] Gray GJ. Advanced Algorithms and Countermeasures for Imaging Infrared Anti-Ship Missiles. Cranfield University; 2013.

[76] Veerman HET, Schleijpen HMA, Brettschneider A, Amram R. Ship signature management to increase infrared countermeasure effectiveness. In: Technologies for Optical Countermeasures XVII; and High-Power Lasers: Technology and Systems, Platforms, Effects IV. vol. 11539. International Society for Optics and Photonics. SPIE; 2020. p. 5 20. Available from: https://-doi.org/10.1117/12.2573953.

[77] Euronaval 2018: Elettronica launches new DIRCM self-protection for naval applications. Navy Recognition; 2018. (Accessed on 12/04/2021). http://-

www.navyrecognition.com/index.php/naval-news/naval-exhibitions/2018/-euronaval-2018/.

[78] SEA: New Elettronica Group DIRCM. Armada International; 2018. (Accessed on 12/05/2021). https://armadainternational.com/2018/10/sea-new-elettronica-group-dircm/.

[79] Andersson M, Johansson R, Stenborg KG, Forsgren R, Cane T, Taberski G, et al. The ipatch system for maritime surveillance and piracy threat classification. In: 2016 European Intelligence and Security Informatics Conference (EISIC). IEEE; 2016. p. 200-0.

[80] Bouma H, de Lange DJJ, van den Broek SP, Kemp RA, Schwering PB. Automatic detection of small surface targets with electro-optical sensors in a harbor environment. In: Electro-Optical Remote Sensing, Photonic Technologies, and Applications II. vol. 7114. International Society for Optics and Photonics; 2008. p. 711402.

[81] Burton M, Benning C. Comparison of imaging infrared detection algorithms. In: Infrared Technology for Target Detection and Classification. vol. 302. International Society for Optics and Photonics; 1982. p. 26-32.

[82] Bhanu B. Automatic target recognition: State of the art survey. IEEE transactions on aerospace and electronic systems. 1986;(4):364-79.

[83] Wang G, Zhang T, Wei L, Sang N. Efficient method for multiscale small target detection from a natural scene. Optical Engineering. 1996;35(3):761-8.

[84] Labonté G, Deck W. Infrared target-flare discrimination using a ZISC hardware neural network. Journal of real-time image processing. 2010;5(1):11-32.

[85] Zernike F. Diffraction theory of the knife-edge test and its improved form, the phase-contrast method. Monthly Notices of the Royal Astronomical Society. 1934;94:377-84.

[86] Maji S, Malik J. Object detection using a max-margin Hough transform. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. IEEE; 2009. p. 1038-45.

[87] Ali J, Khan R, Ahmad N, Maqsood I. Random forests and decision trees. International Journal of Computer Science Issues (IJCSI). 2012;9(5):272.

[88] Permuter H, Francos J, Jermyn I. A study of Gaussian mixture models of color and texture features for image classification and segmentation. Pattern recognition. 2006;39(4):695-706.

[89] Ma L, Crawford MM, Tian J. Local manifold learning-based k-nearest-neighbor for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing. 2010;48(11):4099-109.

[90] Hinton GE, Dayan P, Frey BJ, Neal RM. The "wake-sleep" algorithm for unsupervised neural networks. Science. 1995;268(5214):1158-61.

[91] Majumder UK, Blasch EP, Garren DA. Deep Learning for Radar and Communications Automatic Target Recognition. Artech House; 2020.

[92] Le QV, Ranzato M, Monga R, Devin M, Chen K, Corrado GS, et al. Building high-level features using large scale unsupervised learning. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE; 2013. p. 8595-8.

[93] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision; 2015. p. 1026-34.

[94] Wu R, Yan S, Shan Y, Dang Q, Sun G. Deep image: Scaling up image recognition. arXiv preprint arXiv:150102876. 2015;7(8).

[95] China and the US are racing to develop AI weapons. The Conversation; 2018. (Accessed on 13/04/2021). https://theconversation.com/china-and-the-us-are-racing-to-develop-ai-weapons-97427.

[96] Long range anti-ship missile (LRASM). Naval Technology;. (Accessed on 14/07/2020). https://www.darpa.mil/about-us/long-range-anti-ship-missile.

[97] Kott A, Stump E. Intelligent autonomous things on the battlefield. In: Artificial intelligence for the internet of everything. Elsevier; 2019. p. 47-65.

[98] Johnson J. Artificial intelligence & future warfare: implications for international security. Defense & Security Analysis. 2019;35(2):147-69.

[99] Simon S. How A Business Jet Attacked A US Warship.

[100] Barlow HB. Unsupervised learning. Neural computation. 1989;1(3):295-311.

[101] Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural networks for perception. Elsevier; 1992. p. 65-93.

[102] Zhang R, Isola P, Efros AA. Colorful image colorization. In: European Conference on Computer Vision. Springer; 2016. p. 649-66.

[103] Chan C, Ginosar S, Zhou T, Efros AA. Everybody dance now. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2019. p. 5933-42.

[104] Mitchell M, Dodge J, Goyal A, Yamaguchi K, Stratos K, Han X, et al. Midge: Generating image descriptions from computer vision detections. In: Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics; 2012. p. 747-56.

[105] Murphy-Chutorian E, Trivedi MM. Head pose estimation in computer vision: A survey. IEEE transactions on pattern analysis and machine intelligence. 2008;31(4):607-26.

[106] Emami S, Suciu VP. Facial recognition using OpenCV. Journal of Mobile, Embedded and Distributed Systems. 2012;4(1):38-43.

[107] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2016. p. 779-88.

[108] Goodfellow I, Bengio Y, Aaron C. Deep Learning. MIT Press; 2016.

[109] Hirschman II, Widder DV. The convolution transform. Courier Corporation; 2012.

[110] Blanch MG, Mrak M, Smeaton AF, O'Connor NE. End-to-end conditional gan-based architectures for image colourisation. In: 2019 IEEE 21st International Workshop on Multimedia Signal Processing. IEEE; 2019. p. 1-6.

[111] Gatys LA, Ecker AS, Bethge M. Image style transfer using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2016. p. 2414-23.

[112] Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2014. p. 580-7.

[113] Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, et al. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In: Proceedings of the IEEE international conference on computer vision; 2017. p. 5907-15.

[114] Gray GJ, Aouf N, Richardson MA, Butters B, Walmsley RH. Countermeasure effectiveness against an intelligent imaging infrared anti-ship missile. Optical Engineering. 2013;52(2):026401.

[115] Kechagias-Stamatis O, Aouf N, Nam D. Multi-Modal Automatic Target Recognition for Anti-Ship Missiles with Imaging Infrared Capabilities. In: 2017 Sensor Signal Processing for Defence Conference (SSPD); 2017. p. 1-5.

[116] Tao W, Jin H, Liu J. Unified mean shift segmentation and graph region merging algorithm for infrared ship target segmentation. Optical Engineering. 2007;46(12):127002.

[117] Herman J. Target identification algorithm for the AN/AAS-44V Forward Looking Infrared (FLIR). Naval Postgraduate School, Monterey, CA; 2000.

[118] Gray GJ, Aouf N, Richardson M, Butters B, Walmsley R, Nicholls E. Feature-based recognition approaches for infrared anti-ship missile seekers. The Imaging Science Journal. 2012;60(6):305-20.

[119] Fernandez H, de Seixas J, Neves S, Souza Filho J. Combining morphological mapping and principal curves for ship classification. In: International Symposium on Signals, Circuits and Systems, 2005. ISSCS 2005.. vol. 2. IEEE; 2005. p. 605-8.

[120] Fefilatyev S, Goldgof DB, Langebrake L. Toward detection of marine vehicles on horizon from buoy camera. In: Unmanned/Unattended Sensors and Sensor Networks IV. vol. 6736. International Society for Optics and Photonics; 2007. p. 67360O.

[121] Alves JA. Recognition of ship types from an infrared image using moment invariants and neural networks. Naval Postgraduate School, Monterey, CA; 2001.

[122] Alves J, Herman J, Rowe NC. Robust recognition of ship types from an infrared silhouette. Naval Postgraduate School, Monterey, CA; 2004.

[123] Withagen PJ, Schutte K, Vossepoel AM, Breuers MG. Automatic classification of ships from infrared (FLIR) images. In: Signal Processing, Sensor Fusion, and Target Recognition VIII. vol. 3720. International Society for Optics and Photonics; 1999. p. 180-7.

[124] Gray GJ, Aouf N, Richardson MA, Butters B, Walmsley R. An intelligent tracking algorithm for an imaging infrared anti-ship missile. In: Technologies for Optical Countermeasures IX. vol. 8543. International Society for Optics and Photonics; 2012. p. 85430L.

[125] Moosbauer S, Konig D, Jakel J, Teutsch M. A Benchmark for Deep Learning Based Object Detection in Maritime Environments. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops; 2019. p. 0-0.

[126] He K, Gkioxari G, Dollár P, Girshick R. Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision; 2017. p. 2961-9.

[127] Ren S, He K, Girshick R, Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems; 2015. p. 91-9.

[128] Prasad DK, Prasath CK, Rajan D, Rachmawati L, Rajabaly E, Quek C. Challenges in video based object detection in maritime scenario using computer vision. arXiv preprint arXiv:160801079. 2016.

[129] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems; 2012. p. 1097-105.

[130] Gray GJ, Aouf N, Richardson MA, Butters B, Walmsley R, Nicholls E. Feature-based tracking algorithms for imaging infrared anti-ship missiles. In: Technologies for Optical Countermeasures VIII. vol. 8187. International Society for Optics and Photonics; 2011. p. 81870T.

[131] Liu Z, Bai X, Sun C, Zhou F, Li Y. Infrared ship target segmentation through integration of multiple feature maps. Image and Vision Computing. 2016;48:14-25.

[132] Jin D, Bai X. Distribution information based intuitionistic fuzzy clustering for infrared ship segmentation. IEEE Transactions on Fuzzy Systems. 2019.

[133] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. Imagenet large scale visual recognition challenge. International journal of computer vision. 2015;115(3):211-52.

[134] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 770-8.

[135] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556. 2014.

[136] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2015. p. 1-9.

[137] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:150203167. 2015.

[138] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research. 2014;15(1):1929-58.

[139] Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, et al. Microsoft coco: Common objects in context. In: European conference on computer vision. Springer; 2014. p. 740-55.

[140] Tian Z, Shen C, Chen H, He T. FCOS: Fully Convolutional One-Stage Object Detection. arXiv preprint arXiv:190401355. 2019.

[141] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: Single shot multibox detector. In: European conference on computer vision. Springer; 2016. p. 21-37.

[142] Fu CY, Liu W, Ranga A, Tyagi A, Berg AC. Dssd: Deconvolutional single shot detector. arXiv preprint arXiv:170106659. 2017.

[143] Lin TY, Goyal P, Girshick R, He K, Dollár P. Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision; 2017. p. 2980-8.

[144] Girshick R. Fast R-CNN. In: Proceedings of the IEEE international conference on computer vision; 2015. p. 1440-8.

[145] Redmon J, Farhadi A. YOLO9000: better, faster, stronger. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017. p. 7263-71.

[146] Redmon J, Farhadi A. Yolov3: An incremental improvement. arXiv preprint arXiv:180402767. 2018 April.

[147] Andriluka M, Pishchulin L, Gehler P, Schiele B. 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2014. p. 3686-93.

[148] Brostow GJ, Fauqueur J, Cipolla R. Semantic object classes in video: A high-definition ground truth database. Pattern Recognition Letters. 2009;30(2):88-97.

[149] Cordts M, Omran M, Ramos S, Rehfeld T, Enzweiler M, Benenson R, et al. The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2016. p. 3213-23.

[150] Milan A, Leal-Taixé L, Reid I, Roth S, Schindler K. MOT16: A benchmark for multi-object tracking. arXiv preprint arXiv:160300831. 2016.

[151] Wang T, Bai X, Zhang Y. Multiple features based low-contrast infrared ship image segmentation using fuzzy inference system. In: 2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA). IEEE; 2014. p. 1-6.

[152] Zhao G, Wang F, Zhang T. A new method for ship classification and recognition. In: MIPPR 2005: Image Analysis Techniques. vol. 6044. International Society for Optics and Photonics; 2005. p. 604416.

[153] Gundogdu E, Solmaz B, Yücesoy V, Koc A. Marvel: A large-scale image dataset for maritime vessels. In: Asian Conference on Computer Vision. Springer; 2016. p. 165-80.

[154] Dao-Duc C, Xiaohui H, Morère O. Maritime vessel images classification using deep convolutional neural networks. In: Proceedings of the Sixth International Symposium on Information and Communication Technology; 2015. p. 276-81.

[155] Liu Z, Waqas M, Yang J, Rashid A, Han Z. A Multi-Task CNN for Maritime Target Detection. IEEE Signal Processing Letters. 2021;28:434-8.

[156] Zheng Y, Zhang S. Mcships: A Large-Scale Ship Dataset For Detection And Fine-Grained Categorization In The Wild. In: 2020 IEEE International Conference on Multimedia and Expo (ICME); 2020. p. 1-6.

[157] Ribeiro R, Cruz G, Matos J, Bernardino A. A data set for airborne maritime surveillance environments. IEEE Transactions on Circuits and Systems for Video Technology. 2017;29(9):2720-32.

[158] Zhang MM, Choi J, Daniilidis K, Wolf MT, Kanan C. VAIS: A dataset for recognizing maritime imagery in the visible and infrared spectrums. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops; 2015. p. 10-6.

[159] van den Broek SP, Bouma H, Veerman HE, Benoist KW, den Hollander RJ, Schwering PB. Recognition of ships for long-term tracking. In: Signal Processing, Sensor/Information Fusion, and Target Recognition XXIII. vol. 9091. International Society for Optics and Photonics; 2014. p. 909107.

[160] Wang X, Lv G, Xu L. Infrared dim target detection based on visual attention. Infrared Physics & Technology. 2012;55(6):513-21.

[161] Wang H, Zou Z, Shi Z, Li B. Detecting ship targets in spaceborne infrared image based on modeling radiation anomalies. Infrared Physics & Technology. 2017;85:141-6.

[162] Patino L, Cane T, Vallee A, Ferryman J. Pets 2016: Dataset and challenge. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops; 2016. p. 1-8.

[163] Bloisi DD, Iocchi L, Pennisi A, Tombolini L. ARGOS-Venice boat classification. In: 2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance. IEEE; 2015. p. 1-6.

[164] Morris B, Aha DW, Auslander B, Gupta K. Learning and leveraging context for maritime threat analysis: Vessel classification using Exemplar-SVM. Naval Research Laboratory, Washington, DC; 2012.

[165] Shao Z, Wu W, Wang Z, Du W, Li C. Seaships: A large-scale precisely annotated dataset for ship detection. IEEE transactions on multimedia. 2018;20(10):2593-604.

[166] Spagnolo P, Filieri F, Distante C, Mazzeo PL, D'Ambrosio P. A new annotated dataset for boat detection and re-identification. In: 2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). IEEE; 2019. p. 1-7.

[167] Soloviev V, Farahnakian F, Zelioli L, Iancu B, Lilius J, Heikkonen J. Comparing CNN-Based Object Detectors on Two Novel Maritime Datasets. In: 2020 IEEE International Conference on Multimedia & Expo Workshops (ICMEW). IEEE; 2020. p. 1-6.

[168] Zhang W, He X, Li W, Zhang Z, Luo Y, Su L, et al. A Robust Deep Affinity Network for Multiple Ship Tracking. IEEE Transactions on Instrumentation and Measurement. 2021;70:1-20.

[169] Tremblay C, Valin P. Experiments on individual classifiers and on fusion of a set of classifiers. In: Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002. (IEEE Cat. No. 02EX5997). vol. 1. IEEE; 2002. p. 272-7.

[170] Berk A, Anderson GP, Bernstein LS, Acharya PK, Dothe H, Matthew MW, et al. MODTRAN4 radiative transfer modeling for atmospheric correction. In: Optical spectroscopic techniques and instrumentation for atmospheric and space research III. vol. 3756. International Society for Optics and Photonics; 1999. p. 348-53.

[171] Berk A, Bernstein L, Anderson G, Acharya P, Robertson D, Chetwynd J, et al. MODTRAN cloud and multiple scattering upgrades with application to AVIRIS. Remote sensing of Environment. 1998;65(3):367-75.

[172] Cane T, Ferryman J. Saliency-based detection for maritime object tracking. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops; 2016. p. 18-25.

[173] Spraul R, Sommer L, Schumann A. A comprehensive analysis of modern object detection methods for maritime vessel detection. In: Artificial Intelligence and Machine Learning in Defense Applications II. vol. 11543. International Society for Optics and Photonics. SPIE; 2020. p. 13 24. Available from: https://-doi.org/10.1117/12.2574097.

[174] Atalar O, Bartan B. Ship Classification Using an Image Dataset. Image (r, c). 2018;100:1.

[175] Betti A, Michelozzi B, Bracci A, Masini A. Real-Time target detection in maritime scenarios based on YOLOv3 model. arXiv preprint arXiv:200300800. 2020.

[176] Huang Z, Sui B, Wen J, Jiang G. An intelligent ship image/video detection and classification method with improved regressive deep convolutional neural network. Complexity. 2020;2020.

[177] Mirghasemi S, Yazdi HS, Lotfizad M. A target-based color space for sea target detection. Applied Intelligence. 2012;36(4):960-78.

[178] Nalamati M, Sharma N, Saqib M, Blumenstein M. Automated Monitoring in Maritime Video Surveillance System. In: 2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ). IEEE; 2020. p. 1-6.

[179] Everingham M, Van Gool L, Williams CK, Winn J, Zisserman A. The PASCAL visual object classes challenge 2007 (VOC2007) results. 2007.

[180] Everingham M, Van Gool L, Williams CK, Winn J, Zisserman A. The pascal visual object classes (voc) challenge. International journal of computer vision. 2010;88(2):303-38.

[181] Torralba A, Efros AA. Unbiased look at dataset bias. In: CVPR 2011. IEEE; 2011. p. 1521-8.

[182] Teutsch M, Krüger W. Classification of small boats in infrared images for maritime surveillance. In: 2010 International WaterSide Security Conference. IEEE; 2010. p. 1-7.

[183] Kuhn M, Johnson K, et al. Applied predictive modeling. vol. 26. Springer; 2013.

[184] Russell S, Norvig P. Artificial intelligence: a modern approach. 2002.

[185] James G, Witten D, Hastie T, Tibshirani R. An introduction to statistical learning. vol. 112. Springer; 2013.

[186] Brauchle J, Bayer S, Berger R. Automatic ship detection on multispectral and thermal infrared aerial images using MACS-Mar remote sensing platform. In: Pacific-Rim Symposium on Image and Video Technology. Springer; 2017. p. 382-95.

[187] Hesamifard E, Takabi H, Ghasemi M, Wright RN. Privacy-preserving Machine Learning as a Service. Proc Priv Enhancing Technol. 2018;2018(3):123-42.

[188] Patra A, Suresh A. BLAZE: blazing fast privacy-preserving machine learning. arXiv preprint arXiv:200509042. 2020.

[189] ImageNet Benchmark (Image Classification). Papers With Code;. (Accessed on 24/08/2021). https://paperswithcode.com/sota/image-classification-on-imagenet.

[190] Pham H, Dai Z, Xie Q, Le QV. Meta pseudo labels. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2021. p. 11557-68.

[191] Sánchez J, Perronnin F, Mensink T, Verbeek J. Image classification with the fisher vector: Theory and practice. International journal of computer vision. 2013;105(3):222-45.

[192] Dai X, Chen Y, Xiao B, Chen D, Liu M, Yuan L, et al. Dynamic Head: Unifying Object Detection Heads with Attentions. In: Proceedings of the

IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2021. p. 7373-82.

[193] COCO test-dev Benchmark (Object Detection). Papers With Code;. (Accessed on 24/08/2021). https://paperswithcode.com/sota/object-detection-on-coco.

[194] How open-source software shapes AI policy; 2021. (Accessed on 23/08/2021). https://www.brookings.edu/research/how-open-source-software-shapes-ai-policy/.

[195] Ale L, Zhang N, Li L. Road damage detection using retinanet. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE; 2018. p. 5197-200.

[196] AlMansoori AA, Swamidoss I, Sayadi S, Almarzooqi A. Analysis of different tracking algorithms applied on thermal infrared imagery for maritime surveillance systems. In: Artificial Intelligence and Machine Learning in Defense Applications II. vol. 11543. International Society for Optics and Photonics; 2020. p. 1154308.

[197] Hu Y, Wu X, Zheng G, Liu X. Object detection of UAV for anti-UAV based on improved YOLO v3. In: 2019 Chinese Control Conference (CCC). IEEE; 2019. p. 8386-90.

[198] Johnson JW. Automatic nucleus segmentation with mask-RCNN. In: Science and Information Conference. Springer; 2019. p. 399-407.

[199] Zhou Z, Sanders J, Johnson J, Guha-Thakurta N, Chen M, Briere T, et al. Machine Learning Based Detection of Brain Cancer Metastases in MR Images for Stereotactic Radiosurgery Using Single-Shot Detectors. International Journal of Radiation Oncology, Biology, Physics. 2019;105(1):S116-7.

[200] Share Your Data and Code – IEEE Author Center Conferences. IEEE;. (Accessed on 24/08/2021). https://conferences.ieeeauthorcenter.ieee.org/get-published/share-your-data-and-code/.

[201] Does your code stand up to scrutiny?; 2018. (Accessed on 23/08/2021). https://www.nature.com/articles/d41586-018-02741-4.

[202] The latest in Machine Learning. Papers With Code;. (Accessed on 24/08/2021). https://paperswithcode.com/.

[203] Rogers SK, Ruck DW, Kabrisky M, Tarr GL. Artificial neural networks for automatic target recognition. In: Applications of Artificial Neural Networks. vol. 1294. International Society for Optics and Photonics; 1990. p. 2-12.

[204] Westlake S. IRShips. Cranfield Online Research Data (CORD); 2020. Available from: https://cord.cranfield.ac.uk/articles/dataset/IRShips/12800324.

[205] Sea Surface Temperature. NASA Earth Observatory;. (Accessed on 03/09/2021). https://earthobservatory.nasa.gov/global-maps/MYD28M.

[206] Janes: Ada (MILGEM) class; 2021. (Accessed on 01/09/2021). https://-customer.janes.com/Janes/Display/jfs_a507-jfs_.

[207] Janes: Independence (LCS 2) class (Littoral Combat Ship Flight 0); 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/-Display/jfs_6042-jfs_.

[208] Janes: Cheng Kung (Modified Oliver Hazard Perry) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/-jfs_3072-jfs_.

[209] Janes: Gabya (Oliver Hazard Perry) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_3252-jfs_.

[210] Janes: Santa María (Oliver Hazard Perry) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_2861-jfs_.

[211] Janes: Oliver Hazard Perry class; 2020. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_0904-jfs_.

[212] Janes: Oliver Hazard Perry class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_4329-jfs_.

[213] Janes: Adelaide (Oliver Hazard Perry) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_c983-jfs_.

[214] Janes: Oliver Hazard Perry class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_b307-jfs_.

[215] Janes: Oliver Hazard Perry class; 2020. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_0193-jfs_.

[216] Janes: Jiangkai II (Type 054A) class (FFGHM); 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/JFS_A723-JFS_.

[217] Janes: Alvaro de Bazán (F-100) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_4369-jfs_.

[218] Janes: Oliver Hazard Perry class; 2016. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_3535-jfs_.

[219] Janes: Akizuki class; 2021. (Accessed on 01/09/2021). https://-customer.janes.com/FightingShips/Display/jfs_a797-jfs_.

[220] Janes: Sejong Daewang (KDX-III) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/Janes/Display/jfs_5836-jfs_.

[221] Janes: Zumwalt (DDG 1000) class; 2021. (Accessed on 01/09/2021). https://customer.janes.com/FightingShips/Display/jfs_5873-jfs_.

[222] Armorique | Armorique Ship Guide | Brittany Ferries;. (Accessed on 06/09/2021). https://www.brittany-ferries.co.uk/ships/cruise-ferries/armorique.

[223] CadNaV: Free 3D Models, CAD Models And Textures Download;. (Accessed on 07/09/2021). https://www.cadnav.com/.

[224] GrabCAD: Design Community, CAD Library, 3D Printing Software;. (Accessed on 07/09/2021). https://grabcad.com/.

[225] Burkardt J, Flanagan JF, Saleeba Z, van Velsen M, van der Spoel G, Guglielmetti P, et al.. IVCon project home;. (Accessed on 25/10/2021). http://-ivcon-tl.sourceforge.net/.

[226] Rhino - Rhinoceros 3D;. (Accessed on 09/07/2021). https://-www.rhino3d.com/.

[227] Ltd CC. Open Inventor Extension Notes. 2012 August.

[228] Westlake S. samuelwestlake/IVTools: Tool for editing Open Inventor nodes and converting files to VRML format.; 2021. (Accessed on 20/10/2021). https://github.com/samuelwestlake/IVTools.

[229] Creative Commons – Attribution 2.0 Generic – CC BY 2.0;. (Accessed on 14/09/2021). https://creativecommons.org/licenses/by/2.0/.

[230] Creative Commons – Attribution 3.0 Unported – CC BY 3.0;. (Accessed on 14/09/2021). https://creativecommons.org/licenses/by/3.0/.

[231] BT RIR, et al. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. 2011.

[232] Li H, Wang X. Automatic recognition of ship types from infrared images using support vector machines. In: 2008 International Conference on Computer Science and Software Engineering. vol. 6. IEEE; 2008. p. 483-6.

[233] Deeplodocus | Python Package Wiki;. (Accessed on 27/12/2021). https://package.wiki/Deeplodocus.

[234] Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, et al. Hidden technical debt in machine learning systems. Advances in neural information processing systems. 2015;28:2503-11.

[235] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:160304467. 2016.

[236] Chollet F, et al.. Keras; 2015. https://keras.io.

[237] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems. 2019;32:8026-37.

[238] GitHub - determined-ai/determined: Determined: Deep Learning Training Platform;. (Accessed on 20/12/2021). https://github.com/determined-ai/-determined.

[239] Bisong E. Kubeflow and kubeflow pipelines. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Springer; 2019. p. 671-85.

[240] Westlake S, Leroy A. GitHub - Deeplodocus/deeplodocus: The deep platform keeping your head above water;. (Accessed on 20/12/2021). https://-github.com/Deeplodocus/deeplodocus.

[241] Mobile App Huawei Cloud;. (Accessed on 27/12/2021). https://-www.huaweicloud.com/en-us/solution/app/.

[242] Seiwa Co., Ltd.;. (Accessed on 27/12/2021). https://www.seiwa-st.com/.

[243] PyTorch;. (Accessed on 12/12/2021). https://pytorch.org/.

[244] LeCun Y, Cortes C, Burges CJC. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges;. (Accessed on 09/12/2021). http://yann.lecun.com/exdb/mnist/.

[245] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278-324.

[246] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014.

[247] torch.optim – PyTorch 1.10.0 documentation;. (Accessed on 20/12/2021). https://pytorch.org/docs/stable/optim.html.

[248] torchvision.datasets – Torchvision 0.11.0 documentation;. (Accessed on 12/12/2021). https://pytorch.org/vision/stable/datasets.html.

[249] Bai X, Chen Z, Zhang Y, Liu Z, Lu Y. Infrared ship target segmentation based on spatial information improved FCM. IEEE transactions on cybernetics. 2015;46(12):3259-71.

[250] Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S. Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017. p. 2117-25.

[251] Cai H, Wu Q, Corradi T, Hall P. The cross-depiction problem: Computer vision algorithms for recognising objects in artwork and in photographs. arXiv preprint arXiv:150500110. 2015.

[252] Ginosar S, Haas D, Brown T, Malik J. Detecting people in cubist art. In: European Conference on Computer Vision. Springer; 2014. p. 101-16.

[253] Sadeghi MA, Forsyth D. 30hz object detection with dpm v5. In: European Conference on Computer Vision. Springer; 2014. p. 65-79.

[254] Bourdev L, Malik J. Poselets: Body part detectors trained using 3d human pose annotations. In: 2009 IEEE 12th International Conference on Computer Vision. IEEE; 2009. p. 1365-72.

[255] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. vol. 1. IEEE; 2005. p. 886-93.

[256] Santurkar S, Tsipras D, Ilyas A, Madry A. How does batch normalization help optimization? In: Proceedings of the 32nd international conference on neural information processing systems; 2018. p. 2488-98.

[257] He T, Zhang Z, Zhang H, Zhang Z, Xie J, Li M. Bag of tricks for image classification with convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019. p. 558-67.

[258] Rezatofighi H, Tsoi N, Gwak J, Sadeghian A, Reid I, Savarese S. Generalized Intersection over Union. 2019 June.

[259] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research. 2011;12(7).

[260] Hinton G, Srivastava N, Swersky K. Neural Networks for Machine Learning;. (Accessed on 21/10/2021). https://www.cs.toronto.edu/~tijmen/-csc321/slides/lecture_slides_lec6.pdf.

[261] Zeiler MD. Adadelta: an adaptive learning rate method. arXiv preprint arXiv:12125701. 2012.

[262] Jia X, Song S, He W, Wang Y, Rong H, Zhou F, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. arXiv preprint arXiv:180711205. 2018.

[263] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings; 2010. p. 249-56.

[264] Yosinski J, Clune J, Bengio Y, Lipson H. How transferable are features in deep neural networks? arXiv preprint arXiv:14111792. 2014.

[265] cocodataset/cocoapi: COCO API - Dataset;. (Accessed on 18/10/2021). https://github.com/cocodataset/cocoapi.

[266] Loshchilov I, Hutter F. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:160803983. 2016.

[267] Westlake S. COCO Object Detection - Deeplodocus Documentation; 2019. (Accessed on 18/10/2021). https://deeplodocus.readthedocs.io/en/-master/tutorials/coco/.

[268] Jungo M, Cardona JE. python - Understanding accumulated gradients in PyTorch - Stack Overflow; May. (Accessed on 20/10/2021). https://-stackoverflow.com/questions/62067400/understanding-accumulated-gradients-in-pytorch.

[269] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods. 2020;17:261-72.

[270] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. Advances in neural information processing systems. 2014;27.

[271] Bochkovskiy A, Wang CY, Liao HYM. Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:200410934. 2020.

[272] Aziz L, FC MSbHS, Sheikh UU, Khan S, Ayub H, Ayub S. Multi-level Refinement Feature Pyramid Network for scale imbalance object detection. IEEE Access. 2021:1-1.

[273] Liu Z, Hu H, Lin Y, Yao Z, Xie Z, Wei Y, et al. Swin Transformer V2: Scaling Up Capacity and Resolution. arXiv preprint arXiv:211109883. 2021.

[274] Janes: Danish navy retires Niels Juel-class corvettes; 2009. (Accessed on 07/09/2021). https://customer.janes.com/Janes/Display/jni73114-jni-2009.

[275] Janes: Adelaide (Oliver Hazard Perry) class; 2021. (Accessed on 02/09/2021). https://customer.janes.com/FightingShips/Display/jfs_c983-jfs_.

8. Appendix

A-1

This section outlines how, in eight steps, IVTools can be used to convert a conventional VRML-formatted CAD model into an IV file with an SoIRMaterial node at each named surface. These steps are as follows:

Step 1: Load the CAD file.
Write the path to a VRML or IV CAD file into the first text entry box of the GUI and press the ‘Read’ button to load the CAD file into IVTools.

Step 2: Convert CAD file to IV format.
If a VRML file was loaded, press ‘Convert to IV’ to convert it to the Open Inventor format.

Step 3: Load SoIRMaterial template.
Load a template of the node to be added to each surface of the CAD model. This template must be defined using the YAML Ain’t Markup Language (YAML) format and, for an SoIRMaterial node, should appear similar to the file seen in Figure 8-1.

SoIRMaterial.yaml

```yaml
SoIRMaterial:
    spectralType: TEMPERATURE
    temperature: 20.0
    emisivity: 0.97
```

Figure 8-1: *Template for an SoIRMaterial node, written in YAML format.*

Step 4: Write a ‘nodes file’.
Next, use the template loaded in Step 3 to create a second YAML file which specifies an SoIRMaterial node for each surface of the CAD model by entering a suitable destination path into the third text entry box and pressing the ‘Write Nodes File’ Button. An example of a resultant nodes file can be seen in Figure 8-2. Note that each of the top field names correspond to the object names previously specified in Section 3.2.1.2.

nodes_file.yaml

```yaml
RearCIWSSensor:
  SoIRMaterial:
    spectralType: TEMPERATURE
    temperature: 20.0
    emissivity: 0.97
RearCIWSRadome:
  SoIRMaterial:
    spectralType: TEMPERATURE
    temperature: 20.0
    emissivity: 0.97
RearCIWSBarrel:
  SoIRMaterial:
    spectralType: TEMPERATURE
    temperature: 20.0
    emissivity: 0.97
RearCIWSFrame:
  SoIRMaterial:
    spectralType: TEMPERATURE
    temperature: 20.0
    emissivity: 0.97
RearCIWSBase:
  SoIRMaterial:
    spectralType: TEMPERATURE
    temperature: 20.0
    emissivity: 0.97
```

Figure 8-2: *Example of a nodes YAML file. In total, this file is 670 lines long and specifies SoIRMaterial nodes for each surface of the Akizuki CAD model.*

Step 5: Edit the fields within the nodes file.

Using a text editor, open the newly created nodes file and edit the fields of each node as required.

Step 6: Load the edited nodes file.

Load the edited nodes file into IVTools by writing its path in the fourth text entry box and pressing ‘Load Nodes File’ button.

Step 7: Apply nodes to the CAD model.

Press the ‘Apply Nodes’ button to write each of the SoIRMaterial nodes into the corresponding surfaces of the loaded CAD model.

Step 8: Write CAD resultant file.

Finally, enter the desired destination path for the finished CAD model into the final text entry box and press the ‘Write’ button.

In summary, the IVTools application allows SoIRMaterial properties to be applied quickly and simply to any VRML CAD model, thus enabling such models to be rendered in the infrared domain with the CounterSim missile-target engagement simulator.

A-2

Thermal signatures for each vessel were generated programmatically using a stochastic rules-based approach, pseudocode for which is outlined below:

For each nodes file:
  $\mu_g = U(5, 25)$
  $\sigma_g = U(0, 3)$
  $\mu_r = U(3, 10)$
  $\sigma_r = U(0, 4)$
  $If\ U(0,1)\ >\ 0.5$:
    $A = True$
  $Else$:
    $A = False$
  $For\ each\ named\ surface\ in\ the\ nodes\ file$:
    If "$window$" $in\ surface\ name$:
      $T = N(\mu g, \sigma g)$
      $\varepsilon = N(0.86, 0.01)$
    $Else\ if$ "$chimney$" $in\ surface\ name$:
      $T = N(50, 10)$
      $\varepsilon = N(0.97, 0.97)$
    $Else$:
      $T = N(\mu_g, \sigma_g)$
      $\varepsilon = 0.97$
    $If\ each\ of$ ("$platform$","$support$","$base$") $not\ in\ surface\ name$:
      $condition_1 = True$
    $Else$:
      $condition_1 = True$
    If radome $in\ surface\ name$:
      $condition = True$
    $Else$:
      $condition = False$
    $If (A\ is\ True)\ and\ condition_1\ and\ condition_2$:
      $T \coloneqq T + max(0, N(\mu_r, \sigma_r))$
    $Update\ the\ surface\ in\ the\ nodes\ file\ with\ new\ values\ for\ T\ and\ \varepsilon$

A-3

Ground truth bounding box coordinates each ship were calculated from its binary label image using the Python function shown in Figure 8-3.

calculate_extents.py

```python
import numpy as np


def calculate_extents(image, threshold=50):

    # Threshold pixel values
    image[image < threshold] = 0
    image[image > 0] = 255

    # Get index of all rows which contain a pixel > 0
    x = np.argwhere(image.any(axis=0))[:, 0]

    # Get index of all columns which contain a pixel > 0
    y = np.argwhere(image.any(axis=1))[:, 0]

    if not len(x) or not len(y):
        return None
    else:
        np.array((x[0], y[0], x[-1], y[-1]))
```

Figure 8-3: *A Python function that calculates the extents of objects (represented by non-zero pixels) in an image.*

A-4

Deeplodocus is able to construct deep learning inference pipelines from different software modules, such as input transforms, deep learning models and loss functions. The type of data that each such module accepts are outlined below in Table 8-1.

Table 8-1: *A summary describing which types of data are accepted and returned by each module.*

Module	**Accepts**	**Returns**
Input transform	A single instance of: • input data, • label data, or • additional data.	A single instance of: • input data, • label data, or • additional data.
Model	A batch of inputs	A batch of outputs
Loss	A batch of: • inputs, • labels, • additional data, • outputs, and or • the model.	A single: • loss value, or • dictionary of loss values.
Output transform	A batch of: • inputs, • outputs, • labels, and or • the model.	A batch of outputs.
Metric	A batch of: • inputs, • labels, • additional data, • outputs, and or • the model	A single: • Metric score, or • Dictionary of metric scores

A-5

Figure 8-4 shows the Python program which was used to simulate motion smear in images.

motion_smear.py

```python
import cv2
import numpy as np


def sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return -1
    else:
        return 0


def motion_smear(image, kernel_size, angle, eps=2.22e-16):
    if kernel_size > 1:
        half = int((kernel_size - 1) / 2)
        phi = (angle % 180) / 180 * np.pi
        sx = int(half * np.cos(phi) + sign(np.cos(phi)) - kernel_size * eps)
        sy = int(half * np.sin(phi) + 1 - kernel_size * eps)
        x, y = np.meshgrid(
            np.linspace(0, sign(sx), abs(sx) + 1),
            np.linspace(0, sy, sy + 1)
        )
        h = y.shape[0]
        w = x.shape[1]
        dist2line = (y * np.cos(phi) - x * np.sin(phi))
        rad = np.sqrt(x**2 + y**2)

        # Points beyond the line's end-point but within the line width
        lastpix = np.bitwise_and(rad >= half, abs(dist2line) <= 1).astype(np.float32)

        # Distance to the line's end-point parallel to the line
        x2lastpix = half - np.abs((x + dist2line * np.sin(phi)) / np.cos(phi))
        dist2line = np.sqrt(dist2line ** 2 + x2lastpix ** 2) * lastpix + dist2line * (1 - lastpix)
        dist2line = 1 - np.abs(dist2line)
        dist2line[dist2line < 0] = 0
        kernel = np.zeros((h * 2 - 1, w * 2 - 1), dtype=np.float32)
        kernel[:h, :w] = np.rot90(dist2line, 2)
        kernel[h - 1:, w - 1:] = dist2line
        kernel /= np.sum(kernel)

        if np.cos(phi) > 0:
            kernel = np.flipud(kernel)

        image = cv2.filter2D(image, -1, kernel)
    return image
```

Figure 8-4: *A function written using Python to simulate motion smear.*

A-6

Figure 8-5 shows the Python3 program which was used to simulate fixed pattern noise in images.

fixed_pattern_noise.py

```python
import numpy as np


def fixed_pattern_noise(image, p, magnitude):
    image = image.astype(int)
    h, w = image.shape[0:2]
    noise = np.zeros(w, dtype=int)
    last_i = 0
    for i, value in enumerate(np.random.random(w)):
        if value < p:
            noise[last_i:i] = np.random.standard_normal() * magnitude
            last_i = i + 1
    noise = noise[np.newaxis, :]
    if image.ndim == 3:
        noise = noise[:, :, np.newaxis]
    image += noise
    image[image < 0] = 0
    image[image > 255] = 255
    return image
```

Figure 8-5: *A function written using Python 3 to simulate fixed pattern noise.*

A-7

Training and validation loss and metric scores from the five training runs of YOLOv3 in Section 5.2 are shown below.

Table 8-2: *Training losses and metrics for the YOLOv3 model trained using the non-augmented IRShips data.*

Epoch	Losses					Metrics					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	5.4688	1.9897	0.4877	1.8164	1.1750	0.9675	0.5942	0.7362	0.3904	0.1264	0.8379
2	4.1312	1.5858	0.3670	1.2953	0.8831	0.9882	0.7555	0.8563	0.4842	0.1775	0.8601
3	3.7778	1.4610	0.3232	1.1718	0.8218	0.9891	0.7936	0.8806	0.5630	0.2484	0.8704
4	3.4736	1.3583	0.3033	1.0449	0.7672	0.9908	0.8244	0.9000	0.6395	0.3348	0.8773
5	3.2086	1.2857	0.2944	0.9237	0.7048	0.9916	0.8421	0.9108	0.6986	0.4064	0.8812
6	3.0017	1.2462	0.2839	0.8207	0.6509	0.9913	0.8538	0.9174	0.7490	0.4740	0.8843
7	2.8357	1.2034	0.2704	0.7564	0.6056	0.9929	0.8611	0.9223	0.7675	0.5227	0.8872
8	2.7505	1.2002	0.2705	0.7090	0.5709	0.9920	0.8632	0.9231	0.7910	0.5749	0.8880
9	2.5928	1.1550	0.2579	0.6451	0.5348	0.9929	0.8729	0.9290	0.8144	0.6153	0.8912
10	2.4793	1.1322	0.2571	0.5911	0.4988	0.9939	0.8781	0.9324	0.8315	0.6488	0.8933
11	2.4435	1.1287	0.2522	0.5816	0.4811	0.9929	0.8800	0.9330	0.8397	0.6705	0.8943
12	2.3500	1.1044	0.2476	0.5446	0.4533	0.9939	0.8843	0.9359	0.8482	0.6912	0.8961
13	2.2800	1.0887	0.2494	0.5128	0.4291	0.9941	0.8833	0.9354	0.8595	0.7115	0.8966
14	2.2201	1.0742	0.2398	0.4920	0.4142	0.9943	0.8907	0.9396	0.8666	0.7221	0.8983
15	2.1637	1.0594	0.2384	0.4679	0.3981	0.9941	0.8901	0.9392	0.8742	0.7384	0.8989
16	2.1347	1.0654	0.2352	0.4518	0.3823	0.9938	0.8935	0.9410	0.8783	0.7496	0.9000
17	2.0779	1.0341	0.2361	0.4392	0.3685	0.9938	0.8975	0.9432	0.8802	0.7572	0.9016
18	2.0445	1.0310	0.2296	0.4266	0.3574	0.9943	0.8964	0.9428	0.8846	0.7695	0.9020
19	2.0084	1.0289	0.2264	0.4078	0.3452	0.9939	0.8963	0.9426	0.8900	0.7787	0.9027
20	1.9708	1.0113	0.2268	0.3959	0.3368	0.9948	0.9012	0.9457	0.8926	0.7838	0.9033
					Maximum	**0.9948**	**0.9012**	**0.9457**	**0.8926**	**0.7838**	**0.9033**

Table 8-3: *Validation losses and metrics values for the YOLOv3 model trained using the non-augmented IRShips data.*

Epoch	Metrics					Losses					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	2.5499	1.1355	0.8800	0.5343	0.0000	1.0000	0.1096	0.1975	0.5340	NaN	0.8011
2	2.7668	1.2141	1.0049	0.5479	0.0000	1.0000	0.1032	0.1871	0.4742	NaN	0.7786
3	3.1013	1.2950	1.1632	0.6431	0.0000	1.0000	0.0670	0.1256	0.4762	NaN	0.7563
4	3.2647	1.2744	1.1869	0.8035	0.0000	1.0000	0.0617	0.1162	0.2759	NaN	0.7666
5	3.5233	1.4065	1.3418	0.7750	0.0000	1.0000	0.0298	0.0579	0.6786	NaN	0.7534
6	3.3592	1.3470	1.2890	0.7232	0.0000	1.0000	0.0255	0.0498	0.5833	NaN	0.7527
7	3.3874	1.4014	1.3152	0.6708	0.0000	1.0000	0.0298	0.0579	0.8929	NaN	0.7523
8	3.6293	1.4216	1.3808	0.8269	0.0000	1.0000	0.0223	0.0437	0.5714	NaN	0.7704
9	3.6735	1.4245	1.4001	0.8489	0.0000	1.0000	0.0170	0.0335	0.6250	NaN	0.7721
10	3.6156	1.4056	1.3500	0.8601	0.0000	1.0000	0.0223	0.0437	0.8095	NaN	0.7835
11	4.0780	1.4891	1.5740	1.0149	0.0000	1.0000	0.0085	0.0169	0.2500	NaN	0.7780
12	4.0340	1.4647	1.4352	1.1340	0.0000	1.0000	0.0245	0.0478	0.5652	NaN	0.7772
13	4.3594	1.5210	1.6030	1.2354	0.0000	1.0000	0.0096	0.0190	0.3333	NaN	0.7838
14	4.7226	1.6593	1.7688	1.2945	0.0000	1.0000	0.0074	0.0148	0.2857	NaN	0.7692
15	4.4715	1.6252	1.7799	1.0664	0.0000	1.0000	0.0074	0.0148	0.4286	NaN	0.7568
16	4.4126	1.5743	1.7499	1.0884	0.0000	1.0000	0.0064	0.0127	0.1667	NaN	0.7827
17	4.7371	1.5731	1.7591	1.4050	0.0000	1.0000	0.0085	0.0169	0.0000	NaN	0.7713
18	4.6116	1.6743	1.8822	1.0550	0.0000	1.0000	0.0074	0.0148	0.2857	NaN	0.7720
19	5.0448	1.6631	1.9232	1.4586	0.0000	0.8333	0.0053	0.0106	0.0000	NaN	0.7688
20	4.8063	1.6448	1.8447	1.3168	0.0000	0.2000	0.0043	0.0083	0.0000	NaN	0.7778
					Maximum	**1.0000**	**0.1096**	**0.1975**	**0.8929**	**NaN**	**0.8011**

Table 8-4: *Training losses and metrics for the YOLOv3 model trained using the IRShips data augmented with sea-state.*

Epoch	Losses					Metrics					
	Total	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	6.2877	2.4025	0.7143	1.9467	1.2242	0.9325	0.4142	0.5736	0.3632	0.1232	0.8279
2	4.7396	1.9486	0.5170	1.3670	0.9070	0.9629	0.6115	0.7480	0.4419	0.1696	0.8458
3	4.3796	1.8119	0.4511	1.2675	0.8491	0.9716	0.6723	0.7947	0.5113	0.2067	0.8549
4	4.0941	1.6992	0.4133	1.1735	0.8081	0.9747	0.7127	0.8233	0.5739	0.2722	0.8613
5	3.8326	1.6027	0.3926	1.0768	0.7605	0.9783	0.7453	0.8461	0.6287	0.3370	0.8658
6	3.6423	1.5587	0.3791	0.9876	0.7170	0.9796	0.7581	0.8547	0.6823	0.4041	0.8704
7	3.4465	1.4956	0.3584	0.9170	0.6755	0.9825	0.7757	0.8669	0.7159	0.4516	0.8731
8	3.3649	1.4959	0.3539	0.8691	0.6459	0.9814	0.7789	0.8685	0.7386	0.4997	0.8750
9	3.2067	1.4358	0.3360	0.8195	0.6155	0.9831	0.7940	0.8785	0.7583	0.5323	0.8769
10	3.0749	1.3980	0.3299	0.7636	0.5834	0.9841	0.8013	0.8834	0.7838	0.5670	0.8800
11	3.0412	1.3959	0.3255	0.7523	0.5674	0.9825	0.8028	0.8836	0.7882	0.5925	0.8814
12	2.9598	1.3825	0.3225	0.7124	0.5424	0.9844	0.8084	0.8878	0.8041	0.6176	0.8821
13	2.8361	1.3364	0.3108	0.6713	0.5177	0.9879	0.8149	0.8931	0.8190	0.6400	0.8841
14	2.8006	1.3331	0.3089	0.6531	0.5055	0.9873	0.8218	0.8970	0.8228	0.6557	0.8850
15	2.7400	1.3134	0.3047	0.6323	0.4897	0.9862	0.8234	0.8974	0.8314	0.6736	0.8868
16	2.6843	1.3087	0.3004	0.6042	0.4710	0.9862	0.8289	0.9007	0.8398	0.6896	0.8865
17	2.6292	1.2833	0.2980	0.5920	0.4558	0.9870	0.8322	0.9030	0.8430	0.6995	0.8886
18	2.5935	1.2769	0.2927	0.5774	0.4464	0.9878	0.8371	0.9062	0.8464	0.7074	0.8893
19	2.5337	1.2606	0.2876	0.5537	0.4318	0.9878	0.8348	0.9049	0.8555	0.7236	0.8914
20	2.5132	1.2573	0.2851	0.5465	0.4242	0.9877	0.8378	0.9066	0.8593	0.7315	0.8909
					Maximum	**0.9879**	**0.8378**	**0.9066**	**0.8593**	**0.7315**	**0.8914**

Table 8-5: *Validation losses and metrics for the YOLOv3 model trained using the IRShips data augmented with sea-state.*

Epoch	Losses					Metrics					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	2.3551	1.1520	0.7244	0.4787	0.0000	0.9917	0.1277	0.2262	0.8583	NaN	0.7975
2	2.2190	1.1096	0.5670	0.5423	0.0000	1.0000	0.3511	0.5197	0.4818	NaN	0.7947
3	2.0927	1.0568	0.5347	0.5011	0.0000	0.9975	0.4245	0.5955	0.6190	NaN	0.7966
4	2.0524	0.9941	0.5134	0.5449	0.0000	1.0000	0.4096	0.5811	0.2987	NaN	0.8105
5	2.0434	0.9700	0.4925	0.5809	0.0000	0.9977	0.4521	0.6223	0.2353	NaN	0.8162
6	1.9054	0.9247	0.4488	0.5319	0.0000	0.9978	0.4894	0.6567	0.4217	NaN	0.8205
7	1.8792	0.8630	0.4477	0.5685	0.0000	1.0000	0.4862	0.6543	0.3873	NaN	0.8324
8	1.9866	0.8772	0.4634	0.6460	0.0000	1.0000	0.4755	0.6446	0.3400	NaN	0.8291
9	1.9542	0.8276	0.4128	0.7138	0.0000	1.0000	0.5479	0.7079	0.3573	NaN	0.8364
10	2.0288	0.8323	0.4432	0.7533	0.0000	1.0000	0.4830	0.6514	0.3106	NaN	0.8365
11	2.0921	0.8763	0.4733	0.7424	0.0000	1.0000	0.4830	0.6514	0.3899	NaN	0.8202
12	1.9433	0.7720	0.3696	0.8016	0.0000	1.0000	0.6191	0.7648	0.3162	NaN	0.8448
13	2.0984	0.8039	0.4391	0.8554	0.0000	1.0000	0.5255	0.6890	0.3563	NaN	0.8371
14	2.1523	0.7744	0.4827	0.8951	0.0000	1.0000	0.4809	0.6494	0.3319	NaN	0.8486
15	1.9392	0.7602	0.4132	0.7659	0.0000	1.0000	0.5500	0.7097	0.4468	NaN	0.8458
16	2.2726	0.7903	0.4713	1.0110	0.0000	1.0000	0.4840	0.6523	0.3407	NaN	0.8419
17	2.3370	0.7837	0.4666	1.0866	0.0000	1.0000	0.4979	0.6648	0.3098	NaN	0.8495
18	1.8964	0.7694	0.4934	0.6336	0.0000	1.0000	0.4500	0.6207	0.5650	NaN	0.8492
19	1.9958	0.7419	0.4962	0.7577	0.0000	1.0000	0.4574	0.6277	0.4930	NaN	0.8607
20	2.3295	0.8030	0.5750	0.9515	0.0000	1.0000	0.3989	0.5703	0.4533	NaN	0.8568
					Maximum	**1.0000**	**0.6191**	**0.7648**	**0.8583**	**NaN**	**0.8607**

Table 8-6: *Training losses and metrics for the YOLOv3 model trained using the IRShips data augmented with sky-state.*

Epoch	Losses					Metrics					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	5.9642	2.2496	0.6752	1.8537	1.1857	0.9285	0.4523	0.6083	0.3779	0.1190	0.8355
2	4.5511	1.8044	0.5006	1.3461	0.9001	0.9653	0.6380	0.7683	0.4596	0.1634	0.8542
3	4.1692	1.6607	0.4269	1.2402	0.8414	0.9737	0.6946	0.8108	0.5248	0.2172	0.8634
4	3.8842	1.5524	0.3927	1.1402	0.7989	0.9764	0.7419	0.8431	0.5899	0.2847	0.8689
5	3.6357	1.4697	0.3770	1.0396	0.7494	0.9785	0.7681	0.8606	0.6453	0.3558	0.8720
6	3.4456	1.4320	0.3610	0.9479	0.7047	0.9814	0.7801	0.8692	0.6998	0.4164	0.8754
7	3.2594	1.3797	0.3410	0.8766	0.6620	0.9809	0.7933	0.8771	0.7273	0.4664	0.8792
8	3.1804	1.3784	0.3403	0.8314	0.6303	0.9801	0.7992	0.8805	0.7516	0.5091	0.8795
9	3.0153	1.3233	0.3195	0.7733	0.5991	0.9837	0.8121	0.8897	0.7743	0.5489	0.8822
10	2.8974	1.2901	0.3219	0.7195	0.5660	0.9833	0.8178	0.8930	0.7964	0.5828	0.8850
11	2.8612	1.2888	0.3119	0.7095	0.5510	0.9820	0.8205	0.8940	0.8034	0.6056	0.8859
12	2.7780	1.2720	0.3109	0.6709	0.5242	0.9825	0.8271	0.8981	0.8140	0.6333	0.8871
13	2.6886	1.2434	0.3096	0.6356	0.4999	0.9824	0.8272	0.8982	0.8266	0.6551	0.8881
14	2.6242	1.2278	0.2958	0.6136	0.4869	0.9847	0.8348	0.9035	0.8330	0.6705	0.8900
15	2.5699	1.2110	0.2960	0.5925	0.4705	0.9839	0.8379	0.9050	0.8416	0.6825	0.8902
16	2.5254	1.2144	0.2895	0.5695	0.4519	0.9839	0.8417	0.9073	0.8508	0.7044	0.8913
17	2.4731	1.1869	0.2925	0.5559	0.4378	0.9850	0.8437	0.9089	0.8560	0.7106	0.8928
18	2.4352	1.1843	0.2828	0.5399	0.4282	0.9853	0.8498	0.9126	0.8580	0.7213	0.8929
19	2.3811	1.1688	0.2809	0.5175	0.4139	0.9850	0.8506	0.9129	0.8629	0.7308	0.8936
20	2.3591	1.1618	0.2793	0.5121	0.4059	0.9852	0.8517	0.9136	0.8678	0.7411	0.8946
					Maximum	**0.9853**	**0.8517**	**0.9136**	**0.8678**	**0.7411**	**0.8946**

Table 8-7: *Validation losses and metrics for the YOLOv3 model trained using the IRShips data augmented with sky-state.*

Epoch	Losses					Metrics					
	Total	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	2.0386	0.9153	0.6166	0.5067	0.0000	0.9912	0.2383	0.3842	0.7411	NaN	0.8508
2	1.8774	0.8717	0.5128	0.4929	0.0000	1.0000	0.4660	0.6357	0.7603	NaN	0.8545
3	1.9231	0.8853	0.5477	0.4902	0.0000	1.0000	0.3926	0.5638	0.7805	NaN	0.8539
4	1.9325	0.8766	0.5263	0.5296	0.0000	1.0000	0.3915	0.5627	0.5163	NaN	0.8464
5	1.8850	0.8828	0.5025	0.4996	0.0000	1.0000	0.4128	0.5843	0.5644	NaN	0.8360
6	1.7635	0.8582	0.4609	0.4444	0.0000	1.0000	0.4500	0.6207	0.7021	NaN	0.8384
7	1.7670	0.8587	0.4631	0.4452	0.0000	1.0000	0.4457	0.6166	0.6396	NaN	0.8396
8	1.9363	0.7914	0.4236	0.7214	0.0000	1.0000	0.4957	0.6629	0.2768	NaN	0.8468
9	1.8216	0.7569	0.4025	0.6621	0.0000	1.0000	0.5362	0.6981	0.4067	NaN	0.8558
10	1.9189	0.7527	0.3844	0.7818	0.0000	1.0000	0.5564	0.7150	0.3595	NaN	0.8491
11	2.1081	0.8173	0.4416	0.8492	0.0000	1.0000	0.4681	0.6377	0.3614	NaN	0.8376
12	2.0915	0.7913	0.4055	0.8947	0.0000	1.0000	0.5245	0.6881	0.4118	NaN	0.8435
13	2.2375	0.7918	0.4456	1.0001	0.0000	0.9866	0.4691	0.6359	0.3152	NaN	0.8416
14	2.3006	0.8221	0.5065	0.9720	0.0000	1.0000	0.3894	0.5605	0.4454	NaN	0.8409
15	2.1274	0.8269	0.4871	0.8134	0.0000	0.9847	0.4096	0.5785	0.5506	NaN	0.8424
16	2.3718	0.8377	0.5188	1.0152	0.0000	0.9243	0.3638	0.5221	0.4444	NaN	0.8418
17	2.2276	0.8035	0.4751	0.9490	0.0000	0.8950	0.4351	0.5855	0.5306	NaN	0.8422
18	2.0589	0.8261	0.5237	0.7090	0.0000	0.7877	0.3947	0.5259	0.6280	NaN	0.8450
19	2.3249	0.8326	0.5705	0.9218	0.0000	0.8383	0.3585	0.5022	0.5727	NaN	0.8445
20	2.1866	0.7905	0.5615	0.8346	0.0000	0.7212	0.3660	0.4855	0.5581	NaN	0.8591
					Maximum	**1.0000**	**0.5564**	**0.7150**	**0.7805**	**NaN**	**0.8591**

Table 8-8: *Training losses and metrics for the YOLOv3 model trained using the IRShips data augmented with background clutter.*

Epoch	Losses					Metrics					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	5.6905	2.1123	0.5601	1.8375	1.1806	0.9406	0.5379	0.6844	0.3856	0.1247	0.8365
2	4.3324	1.6928	0.4280	1.3191	0.8924	0.9705	0.6982	0.8121	0.4685	0.1706	0.8592
3	3.9743	1.5622	0.3735	1.2069	0.8318	0.9766	0.7434	0.8442	0.5430	0.2359	0.8685
4	3.6810	1.4562	0.3467	1.0943	0.7838	0.9795	0.7825	0.8700	0.6153	0.3126	0.8746
5	3.4122	1.3713	0.3315	0.9825	0.7270	0.9826	0.8082	0.8869	0.6735	0.3844	0.8782
6	3.2208	1.3378	0.3208	0.8857	0.6765	0.9840	0.8181	0.8934	0.7260	0.4477	0.8814
7	3.0397	1.2884	0.3035	0.8154	0.6324	0.9845	0.8303	0.9009	0.7511	0.4978	0.8843
8	2.9575	1.2874	0.3022	0.7691	0.5988	0.9841	0.8334	0.9025	0.7715	0.5467	0.8845
9	2.7921	1.2322	0.2868	0.7082	0.5649	0.9865	0.8457	0.9107	0.7975	0.5882	0.8882
10	2.6792	1.2080	0.2861	0.6545	0.5306	0.9873	0.8501	0.9136	0.8156	0.6225	0.8895
11	2.6449	1.2039	0.2810	0.6459	0.5141	0.9848	0.8524	0.9138	0.8228	0.6431	0.8910
12	2.5739	1.1930	0.2801	0.6122	0.4885	0.9869	0.8550	0.9162	0.8340	0.6665	0.8920
13	2.4691	1.1604	0.2753	0.5722	0.4613	0.9884	0.8580	0.9186	0.8445	0.6863	0.8932
14	2.4159	1.1484	0.2665	0.5524	0.4486	0.9892	0.8648	0.9228	0.8499	0.6975	0.8946
15	2.3627	1.1318	0.2650	0.5328	0.4330	0.9878	0.8658	0.9228	0.8584	0.7128	0.8952
16	2.3355	1.1432	0.2621	0.5140	0.4162	0.9885	0.8657	0.9230	0.8670	0.7296	0.8966
17	2.2719	1.1097	0.2613	0.4987	0.4023	0.9891	0.8719	0.9268	0.8681	0.7340	0.8980
18	2.2437	1.1083	0.2545	0.4876	0.3934	0.9900	0.8722	0.9273	0.8701	0.7477	0.8981
19	2.1873	1.0970	0.2510	0.4614	0.3780	0.9893	0.8748	0.9285	0.8766	0.7556	0.8988
20	2.1643	1.0855	0.2522	0.4548	0.3718	0.9900	0.8777	0.9304	0.8805	0.7620	0.8996
					Maximum	**0.9900**	**0.8777**	**0.9304**	**0.8805**	**0.7620**	**0.8996**

Table 8-9: *Validation losses and metrics for the YOLOv3 model trained using the IRShips data augmented with background clutter.*

Epoch	Losses					Metrics					
	Total	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	2.2051	0.9973	0.7092	0.4986	0.0000	0.9932	0.1553	0.2686	0.5411	NaN	0.8234
2	2.1647	0.9922	0.6894	0.4832	0.0000	1.0000	0.2021	0.3363	0.6263	NaN	0.8260
3	2.2937	1.0150	0.7608	0.5179	0.0000	1.0000	0.1691	0.2894	0.6415	NaN	0.8233
4	2.3076	0.9735	0.7331	0.6010	0.0000	1.0000	0.1819	0.3078	0.4795	NaN	0.8310
5	2.3110	1.0011	0.7512	0.5586	0.0000	1.0000	0.1840	0.3109	0.6936	NaN	0.8162
6	2.0875	0.9765	0.6659	0.4452	0.0000	1.0000	0.2479	0.3973	0.8455	NaN	0.8208
7	1.9629	0.9304	0.6599	0.3725	0.0000	1.0000	0.2681	0.4228	0.9008	NaN	0.8321
8	1.9956	0.8784	0.6221	0.4951	0.0000	1.0000	0.2883	0.4476	0.6863	NaN	0.8351
9	1.9251	0.8808	0.5895	0.4548	0.0000	1.0000	0.3138	0.4777	0.7729	NaN	0.8348
10	1.9501	0.8753	0.5666	0.5082	0.0000	1.0000	0.3223	0.4875	0.7030	NaN	0.8272
11	2.0773	0.8947	0.6642	0.5184	0.0000	1.0000	0.2511	0.4014	0.7076	NaN	0.8309
12	2.0984	0.8770	0.5668	0.6546	0.0000	0.9515	0.3340	0.4945	0.6306	NaN	0.8332
13	2.3870	0.8967	0.6421	0.8483	0.0000	0.8719	0.2606	0.4013	0.4367	NaN	0.8339
14	2.4984	0.9563	0.7473	0.7948	0.0000	0.8427	0.2394	0.3728	0.5689	NaN	0.8215
15	2.1631	0.9173	0.7051	0.5407	0.0000	0.8435	0.2638	0.4019	0.7702	NaN	0.8329
16	2.3469	0.9083	0.7046	0.7339	0.0000	0.7695	0.2415	0.3676	0.6300	NaN	0.8368
17	2.3447	0.8443	0.6222	0.8782	0.0000	0.7574	0.3255	0.4554	0.5752	NaN	0.8432
18	2.1568	0.9080	0.6963	0.5525	0.0000	0.6317	0.2883	0.3959	0.6863	NaN	0.8372
19	2.3639	0.9036	0.7263	0.7340	0.0000	0.6517	0.2628	0.3745	0.6235	NaN	0.8384
20	2.3333	0.9052	0.7047	0.7234	0.0000	0.5385	0.2755	0.3645	0.6371	NaN	0.8390
					Maximum	**1.0000**	**0.3340**	**0.4945**	**0.9008**	**NaN**	**0.8432**

Table 8-10: *Training losses and metrics for the YOLOv3 model trained using the IRShips data augmented with sea-state, sky-state, and background clutter.*

Epoch	Losses					Metrics					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	6.5868	2.5714	0.8292	1.9638	1.2224	0.9129	0.3403	0.4957	0.3616	0.1227	0.8271
2	4.9577	2.0601	0.5917	1.3906	0.9152	0.9595	0.5521	0.7009	0.4379	0.1643	0.8427
3	4.5391	1.8969	0.5019	1.2856	0.8547	0.9711	0.6267	0.7618	0.4983	0.2011	0.8521
4	4.2440	1.7753	0.4548	1.1993	0.8147	0.9741	0.6745	0.7971	0.5584	0.2653	0.8582
5	4.0004	1.6876	0.4352	1.1072	0.7704	0.9740	0.7070	0.8193	0.6144	0.3261	0.8618
6	3.8048	1.6372	0.4148	1.0236	0.7292	0.9780	0.7223	0.8309	0.6724	0.3903	0.8661
7	3.6095	1.5761	0.3913	0.9529	0.6891	0.9803	0.7420	0.8447	0.7029	0.4437	0.8689
8	3.5327	1.5748	0.3870	0.9098	0.6611	0.9784	0.7458	0.8464	0.7267	0.4848	0.8712
9	3.3806	1.5149	0.3669	0.8653	0.6335	0.9810	0.7606	0.8568	0.7459	0.5203	0.8736
10	3.2350	1.4683	0.3605	0.8052	0.6009	0.9832	0.7755	0.8671	0.7693	0.5514	0.8765
11	3.2098	1.4710	0.3543	0.7976	0.5869	0.9814	0.7725	0.8645	0.7794	0.5756	0.8773
12	3.1166	1.4539	0.3505	0.7521	0.5602	0.9832	0.7820	0.8711	0.7960	0.6080	0.8786
13	3.0055	1.4116	0.3413	0.7143	0.5383	0.9843	0.7847	0.8733	0.8088	0.6292	0.8806
14	2.9684	1.4056	0.3372	0.6985	0.5271	0.9841	0.7946	0.8793	0.8132	0.6451	0.8809
15	2.9106	1.3860	0.3330	0.6797	0.5119	0.9829	0.7941	0.8785	0.8187	0.6607	0.8830
16	2.8393	1.3744	0.3250	0.6474	0.4925	0.9836	0.7998	0.8822	0.8334	0.6788	0.8834
17	2.7991	1.3549	0.3284	0.6379	0.4779	0.9836	0.8037	0.8846	0.8355	0.6892	0.8854
18	2.7527	1.3452	0.3203	0.6196	0.4675	0.9849	0.8094	0.8886	0.8402	0.6973	0.8855
19	2.6940	1.3260	0.3119	0.6007	0.4554	0.9854	0.8127	0.8907	0.8436	0.7061	0.8872
20	2.6798	1.3290	0.3127	0.5916	0.4465	0.9844	0.8140	0.8911	0.8494	0.7172	0.8867
					Maximum	**0.9854**	**0.8140**	**0.8911**	**0.8494**	**0.7172**	**0.8872**

Table 8-11: *Validation losses and metrics for the YOLOv3 model trained using the IRShips data augmented with sea-state, sky-state, and background clutter.*

Epoch	Losses					Metrics					
	Total	Box	Objectness	Recognition	Identification	Precision	Recall	F-score	Recognition	Identification	Average IoU
1	2.1730	1.0338	0.6630	0.4762	0.0000	0.9931	0.1532	0.2654	0.8750	NaN	0.8260
2	1.9814	0.9579	0.4791	0.5444	0.0000	1.0000	0.4840	0.6523	0.4462	NaN	0.8180
3	1.9400	0.9429	0.4553	0.5418	0.0000	1.0000	0.5372	0.6990	0.4178	NaN	0.8087
4	1.8854	0.8725	0.4143	0.5986	0.0000	1.0000	0.5670	0.7237	0.1276	NaN	0.8180
5	1.8612	0.8618	0.3726	0.6269	0.0000	1.0000	0.5936	0.7450	0.1237	NaN	0.8186
6	1.7864	0.8253	0.3393	0.6217	0.0000	1.0000	0.6777	0.8079	0.2166	NaN	0.8225
7	1.7717	0.7714	0.3382	0.6622	0.0000	1.0000	0.6500	0.7879	0.2111	NaN	0.8365
8	1.9876	0.7469	0.3441	0.8967	0.0000	1.0000	0.6638	0.7980	0.1106	NaN	0.8449
9	1.8781	0.7220	0.3319	0.8242	0.0000	1.0000	0.6904	0.8169	0.2558	NaN	0.8503
10	2.1119	0.7073	0.3431	1.0615	0.0000	1.0000	0.6585	0.7941	0.1519	NaN	0.8528
11	2.1837	0.7480	0.3761	1.0596	0.0000	1.0000	0.6053	0.7541	0.2285	NaN	0.8459
12	2.1091	0.6773	0.3194	1.1123	0.0000	1.0000	0.6915	0.8176	0.1908	NaN	0.8636
13	2.1915	0.7263	0.3376	1.1275	0.0000	1.0000	0.6713	0.8033	0.2345	NaN	0.8507
14	2.1923	0.6899	0.3889	1.1135	0.0000	1.0000	0.5553	0.7141	0.3161	NaN	0.8648
15	2.3063	0.6945	0.3676	1.2442	0.0000	1.0000	0.5957	0.7467	0.2804	NaN	0.8620
16	2.4243	0.7060	0.3797	1.3386	0.0000	1.0000	0.5660	0.7228	0.2857	NaN	0.8595
17	2.4514	0.6880	0.3704	1.3930	0.0000	1.0000	0.5957	0.7467	0.2839	NaN	0.8638
18	1.8793	0.6759	0.4048	0.7986	0.0000	1.0000	0.5309	0.6935	0.5030	NaN	0.8667
19	2.0882	0.6797	0.4348	0.9737	0.0000	1.0000	0.5021	0.6686	0.4576	NaN	0.8723
20	2.2542	0.6855	0.4585	1.1101	0.0000	1.0000	0.5032	0.6695	0.4313	NaN	0.8744
						1.0000	**0.6915**	**0.8176**	**0.8750**	**NaN**	**0.8744**

A-8

Training and validation loss and metric scores that resulted from training YOLOv3 using IRShips and re-distributed semi-labelled visual-spectrum images, as described in Section 5.3.1, are presented below.

Table 8-12: *Training losses and metrics for the YOLOv3 model which was trained using both IRShips and the semi-labelled visual-spectrum data.*

Epoch	Losses						Metrics					
	Total	**Backbone**	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	5.9105	0.6872	2.0391	0.6575	1.5573	0.9693	0.9128	0.3403	0.9402	0.3617	0.1228	0.8270
2	4.5898	0.6865	1.6218	0.4637	1.0962	0.7216	0.9596	0.5603	0.8237	0.4423	0.1676	0.8433
3	4.2385	0.6849	1.4851	0.3901	1.0069	0.6715	0.9725	0.6367	0.7944	0.5111	0.2103	0.8534
4	3.9875	0.6832	1.3857	0.3528	0.9296	0.6362	0.9742	0.6853	0.7697	0.5779	0.2862	0.8597
5	3.7810	0.6813	1.3160	0.3373	0.8498	0.5966	0.9749	0.7170	0.7403	0.6360	0.3524	0.8634
6	3.6196	0.6794	1.2767	0.3218	0.7808	0.5610	0.9791	0.7323	0.7133	0.6930	0.4206	0.8674
7	3.4613	0.6774	1.2282	0.3034	0.7244	0.5279	0.9809	0.7519	0.6864	0.7229	0.4722	0.8704
8	3.3999	0.6755	1.2279	0.3002	0.6907	0.5056	0.9783	0.7541	0.6667	0.7451	0.5143	0.8729
9	3.2798	0.6736	1.1814	0.2849	0.6562	0.4838	0.9818	0.7691	0.6482	0.7632	0.5481	0.8750
10	3.1629	0.6716	1.1454	0.2798	0.6086	0.4575	0.9844	0.7822	0.6246	0.7856	0.5826	0.8783
11	3.1441	0.6698	1.1478	0.2750	0.6044	0.4470	0.9824	0.7809	0.6145	0.7924	0.6042	0.8787
12	3.0694	0.6679	1.1338	0.2720	0.5697	0.4259	0.9834	0.7887	0.5944	0.8074	0.6341	0.8802
13	2.9819	0.6660	1.1013	0.2650	0.5408	0.4087	0.9846	0.7908	0.5777	0.8210	0.6518	0.8824
14	2.9511	0.6640	1.0962	0.2619	0.5288	0.4003	0.9847	0.8007	0.5692	0.8234	0.6665	0.8826
15	2.9061	0.6621	1.0817	0.2587	0.5150	0.3885	0.9838	0.8007	0.5571	0.8303	0.6803	0.8844
16	2.8492	0.6602	1.0721	0.2528	0.4907	0.3734	0.9838	0.8061	0.5414	0.8432	0.6983	0.8851
17	2.8157	0.6581	1.0570	0.2550	0.4831	0.3624	0.9845	0.8094	0.5298	0.8450	0.7072	0.8870
18	2.7781	0.6561	1.0494	0.2488	0.4695	0.3543	0.9849	0.8161	0.5211	0.8475	0.7151	0.8872
19	2.7304	0.6540	1.0350	0.2424	0.4538	0.3451	0.9858	0.8167	0.5113	0.8532	0.7231	0.8889
20	2.7174	0.6517	1.0368	0.2428	0.4475	0.3385	0.9847	0.8188	0.5038	0.8593	0.7346	0.8882
						Maximum	**0.9858**	**0.8188**	**0.9402**	**0.8593**	**0.7346**	**0.8889**

Table 8-13: *Validation losses and metrics for the YOLOv3 model which was trained using both IRShips and the semi-labelled visual-spectrum data.*

Epoch	Losses						Metrics					
	Total	**Backbone**	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	3.6451	0.0000	1.7880	1.2673	0.5898	0.0000	0.0135	0.0021	0.0037	0.5000	NaN	0.6795
2	2.8301	0.0000	1.4075	0.9506	0.4720	0.0000	0.8274	0.1734	0.2867	0.4356	NaN	0.8006
3	2.6451	0.0000	1.3736	0.8147	0.4568	0.0000	0.9342	0.3926	0.5528	0.3631	NaN	0.7676
4	2.5830	0.0000	1.3737	0.7063	0.5030	0.0000	0.9655	0.5351	0.6886	0.3539	NaN	0.7635
5	2.4301	0.0000	1.3536	0.6279	0.4487	0.0000	0.9426	0.6287	0.7543	0.5821	NaN	0.7608
6	2.4909	0.0000	1.3859	0.6283	0.4767	0.0000	0.9597	0.6330	0.7628	0.5126	NaN	0.7560
7	2.4312	0.0000	1.3233	0.6152	0.4926	0.0000	0.9904	0.6585	0.7911	0.4895	NaN	0.7679
8	2.5790	0.0000	1.3520	0.6063	0.6206	0.0000	0.9924	0.6926	0.8158	0.3118	NaN	0.7528
9	2.6199	0.0000	1.3421	0.6098	0.6680	0.0000	0.9806	0.6989	0.8161	0.3120	NaN	0.7563
10	2.5737	0.0000	1.3565	0.6110	0.6061	0.0000	0.9920	0.6574	0.7908	0.3883	NaN	0.7507
11	2.6838	0.0000	1.3396	0.5987	0.7455	0.0000	0.9846	0.6798	0.8043	0.2473	NaN	0.7493
12	2.5727	0.0000	1.3824	0.6110	0.5792	0.0000	0.9792	0.7011	0.8171	0.4643	NaN	0.7341
13	2.6714	0.0000	1.4058	0.6031	0.6625	0.0000	0.9742	0.6840	0.8037	0.4028	NaN	0.7297
14	2.5627	0.0000	1.3625	0.6232	0.5770	0.0000	0.9508	0.6989	0.8056	0.4901	NaN	0.7331
15	2.4268	0.0000	1.3372	0.6272	0.4624	0.0000	0.9500	0.6670	0.7837	0.6555	NaN	0.7382
16	2.5833	0.0000	1.3600	0.6222	0.6011	0.0000	0.9628	0.6606	0.7836	0.4605	NaN	0.7319
17	2.6232	0.0000	1.3654	0.6244	0.6334	0.0000	0.9602	0.6681	0.7880	0.4729	NaN	0.7296
18	2.3071	0.0000	1.3144	0.5998	0.3929	0.0000	0.9659	0.6628	0.7861	0.7560	NaN	0.7319
19	2.4533	0.0000	1.3705	0.6335	0.4492	0.0000	0.9463	0.6755	0.7883	0.6677	NaN	0.7170
20	2.4530	0.0000	1.3177	0.6127	0.5225	0.0000	0.9264	0.6691	0.7770	0.5787	NaN	0.7327
						Maximum	**0.9924**	**0.7011**	**0.8171**	**0.7560**	**NaN**	**0.8006**

A-9

Training and validation loss and metric scores that resulted from training YOLOv3 using IRShips and re-distributed semi-labelled visual-spectrum images, as described in Section 5.3.2, are presented below.

Table 8-14: *Training losses and metrics for the YOLOv3 model which was trained using both IRShips and the re-distributed semi-labelled visual-spectrum data.*

Epoch	Losses						Metrics					
	Total	**Backbone**	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	5.9108	0.6874	2.0392	0.6576	1.5573	0.9693	0.9128	0.3404	0.4958	0.3615	0.1224	0.8271
2	4.5901	0.6867	1.6219	0.4637	1.0962	0.7216	0.9593	0.5603	0.7075	0.4420	0.1675	0.8432
3	4.2387	0.6852	1.4851	0.3901	1.0069	0.6715	0.9722	0.6368	0.7695	0.5117	0.2106	0.8534
4	3.9874	0.6834	1.3856	0.3527	0.9295	0.6361	0.9741	0.6850	0.8044	0.5788	0.2863	0.8597
5	3.7809	0.6816	1.3159	0.3372	0.8496	0.5965	0.9747	0.7172	0.8263	0.6368	0.3529	0.8633
6	3.6194	0.6797	1.2765	0.3218	0.7805	0.5609	0.9790	0.7320	0.8376	0.6933	0.4207	0.8675
7	3.4615	0.6779	1.2282	0.3033	0.7242	0.5278	0.9808	0.7518	0.8512	0.7230	0.4722	0.8704
8	3.4001	0.6759	1.2279	0.3002	0.6905	0.5055	0.9783	0.7544	0.8519	0.7453	0.5152	0.8729
9	3.2797	0.6741	1.1814	0.2848	0.6559	0.4836	0.9820	0.7693	0.8627	0.7630	0.5485	0.8751
10	3.1630	0.6722	1.1454	0.2797	0.6084	0.4572	0.9844	0.7819	0.8715	0.7857	0.5824	0.8782
11	3.1444	0.6704	1.1478	0.2751	0.6043	0.4468	0.9823	0.7807	0.8700	0.7927	0.6045	0.8788
12	3.0695	0.6685	1.1339	0.2720	0.5694	0.4257	0.9838	0.7886	0.8755	0.8081	0.6346	0.8802
13	2.9823	0.6667	1.1015	0.2651	0.5405	0.4085	0.9846	0.7909	0.8772	0.8209	0.6514	0.8825
14	2.9515	0.6648	1.0962	0.2619	0.5286	0.4001	0.9845	0.8009	0.8833	0.8231	0.6666	0.8825
15	2.9065	0.6629	1.0816	0.2587	0.5148	0.3884	0.9839	0.8015	0.8834	0.8307	0.6801	0.8843
16	2.8497	0.6611	1.0721	0.2527	0.4904	0.3733	0.9837	0.8066	0.8864	0.8428	0.6986	0.8851
17	2.8163	0.6591	1.0570	0.2551	0.4829	0.3622	0.9845	0.8095	0.8885	0.8450	0.7075	0.8869
18	2.7789	0.6572	1.0494	0.2488	0.4693	0.3542	0.9850	0.8157	0.8924	0.8468	0.7150	0.8873
19	2.7312	0.6551	1.0351	0.2424	0.4536	0.3451	0.9857	0.8170	0.8935	0.8539	0.7229	0.8890
20	2.7184	0.6529	1.0369	0.2428	0.4474	0.3384	0.9844	0.8190	0.8941	0.8590	0.7349	0.8882
						Maximum	0.9857	0.8190	0.8941	0.8590	0.7349	0.8890

Table 8-15: *Validation losses and metrics for the YOLOv3 model which was trained using both IRShips and the re-distributed semi-labelled visual-spectrum data.*

Epoch	Losses						Metrics					
	Total	**Backbone**	**Box**	**Objectness**	**Recognition**	**Identification**	**Precision**	**Recall**	**F-score**	**Recognition**	**Identification**	**Average IoU**
1	3.3968	0.0000	1.4802	1.3475	0.5692	0.0000	0.0741	0.0128	0.0218	0.2500	NaN	0.8635
2	2.5777	0.0000	1.1563	0.9109	0.5105	0.0000	0.6600	0.3181	0.4293	0.1572	NaN	0.8150
3	2.3840	0.0000	1.0787	0.7460	0.5593	0.0000	0.8407	0.6064	0.7046	0.1000	NaN	0.8225
4	2.3493	0.0000	1.0560	0.6097	0.6835	0.0000	0.9654	0.7713	0.8575	0.0497	NaN	0.8289
5	2.2158	0.0000	1.0228	0.5085	0.6845	0.0000	0.9949	0.8266	0.9030	0.0644	NaN	0.8247
6	2.3667	0.0000	1.0656	0.4840	0.8171	0.0000	0.9975	0.8351	0.9091	0.0280	NaN	0.8120
7	2.3819	0.0000	1.0221	0.4654	0.8944	0.0000	1.0000	0.8415	0.9139	0.0291	NaN	0.8184
8	2.6495	0.0000	1.0350	0.4484	1.1661	0.0000	0.9976	0.8745	0.9320	0.0109	NaN	0.8091
9	2.7941	0.0000	1.0296	0.4427	1.3218	0.0000	0.9929	0.8883	0.9377	0.0144	NaN	0.8095
10	2.7704	0.0000	1.0422	0.4389	1.2893	0.0000	0.9857	0.8777	0.9285	0.0218	NaN	0.8022
11	2.9800	0.0000	0.9956	0.4211	1.5633	0.0000	0.9871	0.8979	0.9404	0.0201	NaN	0.8063
12	2.7976	0.0000	1.0163	0.4214	1.3599	0.0000	0.9861	0.9074	0.9452	0.0410	NaN	0.8037
13	3.0086	0.0000	1.0503	0.4118	1.5465	0.0000	0.9884	0.9032	0.9439	0.0318	NaN	0.7867
14	2.8570	0.0000	1.0086	0.4318	1.4166	0.0000	0.9761	0.9138	0.9440	0.0780	NaN	0.7954
15	2.6689	0.0000	0.9831	0.4328	1.2530	0.0000	0.9627	0.9053	0.9331	0.0952	NaN	0.7868
16	3.0162	0.0000	0.9949	0.4309	1.5904	0.0000	0.9694	0.9085	0.9379	0.0539	NaN	0.7810
17	3.0947	0.0000	1.0036	0.4246	1.6665	0.0000	0.9682	0.9064	0.9363	0.0481	NaN	0.7851
18	2.4794	0.0000	0.9610	0.4048	1.1137	0.0000	0.9758	0.8989	0.9358	0.1669	NaN	0.7912
19	2.5982	0.0000	0.9821	0.4412	1.1749	0.0000	0.9395	0.9085	0.9237	0.1604	NaN	0.7838
20	2.7574	0.0000	0.9592	0.4244	1.3738	0.0000	0.9387	0.8957	0.9167	0.1140	NaN	0.7917
						Maximum	**1.0000**	**0.9138**	**0.9452**	**0.2500**	**NaN**	**0.8635**

A-10

The visual-spectrum and near-infrared video sequences contained within the Singapore Maritime Dataset were categorised into training, validation, and test sub-sets using the data categorisation proposed by Moosbauer et al. [125].

The visual-spectrum training, validation, and test sets are presented below in Tables 8-16, 8-17, and 8-18 respectively.

Table 8-16: *The visual-spectrum video sequences of the Singapore Maritime Dataset that, as per [3], were categorised as training data.*

Video Name	Condition	Frames	Labels	Objects
MVI_1451	hazy	439	3,270	8
MVI_1609	dark/twilight	505	9,072	20
MVI_1452	hazy	340	1,700	5
MVI_1610	daylight	543	3,166	6
MVI_1478	daylight	477	2,901	7
MVI_1479	daylight	206	1,271	7
MVI_1481	daylight	409	3,095	9
MVI_1482	daylight	454	2,460	6
MVI_1584	dark/twilight	550	7,320	14
MVI_1613	daylight	626	6,574	12
MVI_1614	daylight	582	6,957	13
MVI_1615	daylight	566	3,843	8
MVI_1617	daylight	600	5,940	14
MVI_1619	daylight	473	2,838	6
MVI_1620	daylight	502	3,012	6
MVI_1583	dark/twilight	251	3,186	13
MVI_1622	daylight	309	1,103	4
MVI_1623	daylight	522	3,094	6
MVI_1587	dark/twilight	600	8,858	15
MVI_1624	daylight	494	1,976	4
MVI_1625	daylight	994	8,111	11
MVI_1592	dark/twilight	491	3,629	8
MVI_1644	daylight	252	1,764	7
MVI_1645	daylight	535	3,210	6
MVI_1646	daylight	520	4,533	9
MVI_0801	daylight	600	919	2
Total		**12,840**	**103,802**	

Table 8-17: *The visual-spectrum video sequences of the Singapore Maritime Dataset that, as per [3], were categorised as validation data.*

Video Name	Condition	Frames	Labels	Objects
MVI_1469	daylight	600	5,947	11
MVI_1578	dark/twilight	505	3,535	7
MVI_0790	daylight	1,010	597	1
Total		**2,115**	**10,079**	

Table 8-18: *The visual-spectrum video sequences of the Singapore Maritime Dataset that, as per [3], were categorised as test data.*

Video Name	Condition	Frames	Labels	Objects
MVI_1448	hazy	604	5,443	10
MVI_1474	daylight	445	6,674	15
MVI_1484	daylight	687	2,748	4
MVI_1486	daylight	629	6,713	11
MVI_1582	dark/twilight	540	6,480	12
MVI_1612	daylight	261	2,514	10
MVI_1626	daylight	556	5,329	12
MVI_1627	daylight	600	4,200	7
MVI_1640	daylight	310	2,183	9
MVI_0797	daylight	600	1,258	3
Total		**5,232**	**43,542**	

The near-infrared training, and test sets are documented below in Table 8-19, Table 8-20 respectively.

Table 8-19: *The near-infrared video sequences of the Singapore Maritime Dataset that, as per [3], were categorised as training data.*

Video Name	Frames	Labels	Objects
MVI_1523	600	5,960	11
MVI_1524	579	6,028	28
MVI_1525	566	3,562	7
MVI_1526	600	2,154	4
MVI_1463	317	6,324	20
MVI_1527	602	5,864	14
MVI_1528	600	2,207	7
MVI_1532	295	852	3
MVI_1529	478	2,868	6
MVI_1530	497	2,485	5
MVI_0895	440	3,201	9
MVI_1538	417	1,599	4
MVI_1539	601	3,606	6
MVI_1541	508	7,789	16
MVI_1552	799	2,584	4
MVI_1550	534	3,738	7
MVI_1551	520	4,680	9
Total	**8,953**	**65,501**	

Table 8-20: *The near-infrared video sequences of the Singapore Maritime Dataset that, as per [3], were categorised as test data.*

Video Name	Frames	Labels	Objects
MVI_1468	349	3,032	9
MVI_1520	541	2,573	5
MVI_1521	600	3,600	6
MVI_1522	262	2,519	10
MVI_1545	307	3,483	14
MVI_1548	274	2,466	9
Total	**2,333**	**17,673**	

A-11

Figure 8-6 and Figure 8-7 and shows the two Python3 functions used to randomly vary the contrast of images during algorithm training in Chapter 6.

contrast_shift.py

```python
import contextlib
import numpy as np
import random


def contrast_shift(image,  sy=(0.8, 1.1)):
    image = image.astype(np.float32)

    with contextlib.suppress(TypeError):
        sy = sy[0] + np.random.rand() * (sy[1] - sy[0])
    shift = np.random.rand() * 255 * (1 - sy)

    if random.choice((True, False)):
        image = -sy * 255 / 2 * np.cos(np.pi * image / 255) + sy * 255 / 2 + shift
    else:
        image = image * sy + shift

    image[image > 255] = 255
    image[image < 0] = 0
    image = image.astype(np.uint8)
    return image
```

Figure 8-6: *The first function used to alter image contrast during algorithm training.*

contrast_gradient.py

```python
import contextlib
import numpy as np


def gradient(image, intensity=(-50, 50)):
    with contextlib.suppress(TypeError):
        intensity = intensity[0] + np.random.rand() * (intensity[1] - intensity[0])
    image = image.astype(np.float32)

    gx, gy = np.meshgrid(np.linspace(0, 1, image.shape[1]), np.linspace(0, 1, image.shape[0]))
    gx *= np.pi / 2
    gy *= np.pi / 2
    gx = np.sin(gx + np.radians(np.random.rand() * 90))
    gy = np.sin(gy + np.radians(np.random.rand() * 90))
    image += gx[..., None] * gy[..., None] * intensity

    image[image > 255] = 255
    image[image < 0] = 0
    image = image.astype(np.uint8)
    return image
```

Figure 8-7: *The second function used to alter image contrast during algorithm training.*

A-12

The detection performance of YOLOv3 with respect to the SMD visual-spectrum and near-infrared test data, are presented below in Table 8-21 and Table 8-22 respectively.

Table 8-21: *Results for YOLOv3 when trained with the visual-spectrum training data and tested with the visual-spectrum test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.670	0.917	0.774	0.551	0.847	0.668
0.10	0.730	0.901	0.807	0.636	0.833	0.722
0.15	0.761	0.887	0.819	0.680	0.824	0.745
0.20	0.784	0.875	0.827	0.712	0.815	0.760
0.25	0.802	0.864	0.832	0.736	0.808	0.770
0.30	0.818	0.852	0.835	0.757	0.799	0.777
0.35	0.829	0.841	0.835	0.773	0.791	0.782
0.40	**0.842**	**0.829**	**0.835**	0.788	0.780	0.784
0.45	0.854	0.813	0.833	**0.802**	**0.767**	**0.784**
0.50	0.866	0.797	0.830	0.816	0.752	0.783
0.55	0.880	0.778	0.826	0.831	0.736	0.780
0.60	0.891	0.756	0.818	0.843	0.716	0.774
0.65	0.903	0.729	0.806	0.855	0.692	0.765
0.70	0.912	0.692	0.787	0.867	0.658	0.748
0.75	0.923	0.643	0.758	0.878	0.612	0.722
0.80	0.931	0.587	0.720	0.888	0.560	0.687
0.85	0.942	0.509	0.661	0.903	0.488	0.633
0.90	0.956	0.394	0.558	0.924	0.381	0.540
0.95	0.991	0.245	0.393	0.980	0.243	0.389
Maximum			**0.835**			**0.784**

Table 8-22: *Results for YOLOv3 when trained with the near-infrared training data and tested with the near-infrared test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.750	0.899	0.817	0.610	0.790	0.689
0.10	0.805	0.882	0.842	0.686	0.778	0.729
0.15	0.836	0.865	0.850	0.721	0.764	0.742
0.20	**0.853**	**0.846**	**0.850**	0.743	0.750	0.746
0.25	0.866	0.829	0.847	**0.758**	**0.735**	**0.747**
0.30	0.876	0.813	0.843	0.770	0.722	0.745
0.35	0.887	0.800	0.841	0.783	0.712	0.746
0.40	0.895	0.788	0.838	0.793	0.702	0.744
0.45	0.904	0.776	0.835	0.804	0.692	0.744
0.50	0.913	0.760	0.829	0.816	0.681	0.743
0.55	0.921	0.744	0.823	0.827	0.670	0.740
0.60	0.929	0.724	0.814	0.839	0.655	0.735
0.65	0.939	0.696	0.799	0.851	0.631	0.725
0.70	0.948	0.655	0.775	0.861	0.595	0.704
0.75	0.958	0.596	0.735	0.875	0.544	0.671
0.80	0.969	0.508	0.666	0.889	0.466	0.611
0.85	0.985	0.414	0.583	0.917	0.385	0.543
0.90	0.998	0.302	0.463	0.957	0.289	0.444
0.95	1.000	0.150	0.260	0.997	0.149	0.259
Maximum			**0.850**			**0.747**

A-13

The detection performance of YOLOv3-all and the four modified versions of YOLOv3 with respect to the SMD visual-spectrum test data, are presented below in Tables 8-23, 8-24, 8-25, 8-26, and 8-27.

Table 8-23: *Results for YOLOv3-all (YOLOv3 trained using both the visual-spectrum and the near-infrared training data) with respect to the visual-spectrum test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.696	0.923	0.793	0.581	0.845	0.688
0.10	0.755	0.914	0.827	0.663	0.835	0.739
0.15	0.786	0.906	0.842	0.700	0.827	0.758
0.20	0.806	0.897	0.849	0.724	0.820	0.769
0.25	0.821	0.887	0.853	0.741	0.812	0.775
0.30	**0.831**	**0.876**	**0.853**	0.755	0.803	0.778
0.35	0.841	0.864	0.852	0.768	0.794	0.780
0.40	0.851	0.851	0.851	0.780	0.784	0.782
0.45	0.860	0.837	0.849	**0.792**	**0.773**	**0.782**
0.50	0.870	0.822	0.845	0.803	0.761	0.782
0.55	0.878	0.804	0.839	0.814	0.747	0.779
0.60	0.886	0.780	0.829	0.824	0.727	0.772
0.65	0.895	0.752	0.818	0.836	0.703	0.764
0.70	0.904	0.715	0.799	0.849	0.672	0.750
0.75	0.914	0.667	0.771	0.862	0.629	0.728
0.80	0.928	0.606	0.733	0.880	0.575	0.695
0.85	0.942	0.530	0.678	0.896	0.504	0.645
0.90	0.957	0.426	0.590	0.916	0.408	0.565
0.95	0.993	0.291	0.450	0.975	0.285	0.441
Maximum			**0.853**			**0.782**

Table 8-24: *Results for YOLOv3-sdd-1 with respect to the visual-spectrum test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.689	0.923	0.789	0.581	0.853	0.691
0.10	0.748	0.912	0.822	0.661	0.842	0.741
0.15	0.778	0.902	0.835	0.700	0.832	0.760
0.20	0.799	0.894	0.844	0.724	0.825	0.771
0.25	0.814	0.885	0.848	0.742	0.818	0.778
0.30	0.824	0.876	0.849	0.756	0.811	0.782
0.35	**0.835**	**0.866**	**0.850**	0.769	0.803	0.785
0.40	0.846	0.854	0.850	0.782	0.793	0.788
0.45	0.857	0.840	0.848	**0.795**	**0.783**	**0.789**
0.50	0.867	0.823	0.845	0.808	0.770	0.789
0.55	0.876	0.803	0.838	0.819	0.754	0.785
0.60	0.884	0.779	0.828	0.831	0.734	0.779
0.65	0.892	0.751	0.815	0.842	0.710	0.770
0.70	0.901	0.716	0.798	0.853	0.679	0.756
0.75	0.912	0.673	0.775	0.865	0.639	0.735
0.80	0.923	0.620	0.742	0.877	0.590	0.705
0.85	0.938	0.554	0.696	0.894	0.527	0.663
0.90	0.950	0.448	0.609	0.907	0.428	0.581
0.95	0.986	0.287	0.445	0.956	0.279	0.431
Maximum			**0.850**			**0.789**

Table 8-25: *Results for YOLOv3-sdd-4 with respect to the visual-spectrum test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.692	0.923	0.791	0.577	0.849	0.687
0.10	0.753	0.912	0.825	0.660	0.839	0.739
0.15	0.786	0.903	0.840	0.702	0.830	0.761
0.20	0.808	0.893	0.848	0.729	0.821	0.772
0.25	0.824	0.883	0.853	0.749	0.812	0.779
0.30	0.837	0.874	0.855	0.763	0.805	0.783
0.35	**0.847**	**0.863**	**0.855**	0.776	0.796	0.786
0.40	0.858	0.851	0.855	0.789	0.787	0.788
0.45	0.868	0.838	0.853	**0.801**	**0.777**	**0.789**
0.50	0.878	0.821	0.848	0.814	0.764	0.788
0.55	0.887	0.798	0.840	0.827	0.746	0.784
0.60	0.896	0.772	0.829	0.840	0.725	0.778
0.65	0.905	0.742	0.815	0.852	0.699	0.768
0.70	0.913	0.705	0.796	0.864	0.668	0.754
0.75	0.922	0.663	0.772	0.878	0.631	0.734
0.80	0.933	0.611	0.738	0.891	0.583	0.705
0.85	0.945	0.540	0.688	0.903	0.516	0.657
0.90	0.957	0.433	0.597	0.920	0.417	0.574
0.95	0.993	0.284	0.441	0.968	0.277	0.430
Maximum			**0.855**			**0.789**

Table 8-26: *Results for YOLOv3-sdd-10 with respect to the visual-spectrum test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.690	0.920	0.788	0.573	0.845	0.683
0.10	0.752	0.910	0.824	0.657	0.835	0.735
0.15	0.784	0.902	0.839	0.698	0.827	0.757
0.20	0.806	0.893	0.847	0.723	0.818	0.768
0.25	0.821	0.882	0.851	0.743	0.810	0.775
0.30	0.832	0.871	0.851	0.758	0.803	0.780
0.35	**0.842**	**0.861**	**0.852**	0.772	0.796	0.784
0.40	0.852	0.850	0.851	0.785	0.787	0.786
0.45	0.862	0.836	0.849	0.798	0.777	0.787
0.50	0.873	0.819	0.845	**0.812**	**0.765**	**0.788**
0.55	0.883	0.800	0.839	0.825	0.750	0.786
0.60	0.893	0.777	0.831	0.839	0.731	0.781
0.65	0.901	0.747	0.817	0.850	0.706	0.771
0.70	0.910	0.712	0.799	0.861	0.674	0.756
0.75	0.920	0.668	0.774	0.873	0.633	0.734
0.80	0.932	0.609	0.737	0.886	0.579	0.700
0.85	0.942	0.534	0.682	0.895	0.508	0.648
0.90	0.953	0.432	0.594	0.909	0.412	0.567
0.95	0.991	0.291	0.450	0.961	0.282	0.436
Maximum			**0.852**			**0.788**

Table 8-27: *Results for YOLOv3-sdd-full with respect to the visual-spectrum test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.689	0.918	0.788	0.572	0.841	0.681
0.10	0.741	0.906	0.815	0.647	0.829	0.727
0.15	0.772	0.894	0.829	0.687	0.820	0.748
0.20	0.794	0.884	0.837	0.715	0.812	0.761
0.25	0.811	0.876	0.842	0.736	0.805	0.769
0.30	0.824	0.865	0.844	0.752	0.797	0.774
0.35	**0.835**	**0.853**	**0.844**	0.766	0.788	0.777
0.40	0.845	0.840	0.843	**0.778**	**0.777**	**0.778**
0.45	0.855	0.824	0.839	0.790	0.764	0.777
0.50	0.866	0.807	0.835	0.803	0.750	0.775
0.55	0.877	0.786	0.829	0.816	0.733	0.772
0.60	0.887	0.759	0.818	0.828	0.710	0.765
0.65	0.897	0.727	0.803	0.841	0.681	0.753
0.70	0.907	0.689	0.783	0.851	0.647	0.736
0.75	0.917	0.645	0.758	0.863	0.608	0.713
0.80	0.927	0.596	0.725	0.874	0.562	0.684
0.85	0.936	0.532	0.678	0.886	0.503	0.642
0.90	0.950	0.430	0.592	0.903	0.408	0.562
0.95	0.983	0.284	0.440	0.949	0.274	0.425
Maximum			**0.844**			**0.778**

A-14

The detection performance of YOLOv3-all and the four modified versions of YOLOv3 with respect to the SMD visual-spectrum test data, are presented below in Tables 8-28, 8-29, 8-30, 8-31, and 8-32.

Table 8-28: *Results for YOLOv3-all (YOLOv3 trained using both the visual-spectrum and the near-infrared training data) with respect to the near-infrared test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.728	0.913	0.810	0.615	0.841	0.711
0.10	0.770	0.901	0.831	0.675	0.829	0.744
0.15	0.805	0.888	0.845	0.720	0.817	0.766
0.20	0.840	0.874	0.857	0.763	0.808	0.785
0.25	0.868	0.861	0.865	0.798	0.800	0.799
0.30	**0.888**	**0.849**	**0.868**	0.822	0.792	0.807
0.35	0.902	0.837	0.868	0.839	0.783	0.810
0.40	0.914	0.824	0.867	0.855	0.774	0.812
0.45	0.924	0.808	0.862	**0.869**	**0.762**	**0.812**
0.50	0.933	0.794	0.858	0.881	0.751	0.811
0.55	0.941	0.779	0.852	0.892	0.739	0.809
0.60	0.950	0.762	0.846	0.905	0.726	0.806
0.65	0.958	0.739	0.834	0.916	0.707	0.798
0.70	0.964	0.711	0.818	0.925	0.683	0.786
0.75	0.970	0.674	0.795	0.932	0.648	0.765
0.80	0.976	0.628	0.765	0.940	0.605	0.736
0.85	0.986	0.565	0.718	0.953	0.546	0.695
0.90	0.992	0.456	0.625	0.962	0.442	0.605
0.95	1.000	0.295	0.456	0.985	0.291	0.449
Maximum			**0.868**			**0.812**

Table 8-29: *Results for YOLOv3-sdd-1 with respect to the near-infrared test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.733	0.916	0.815	0.619	0.846	0.715
0.10	0.780	0.906	0.838	0.686	0.836	0.754
0.15	0.817	0.894	0.854	0.734	0.826	0.777
0.20	0.845	0.882	0.863	0.769	0.817	0.792
0.25	0.870	0.871	0.870	0.797	0.808	0.802
0.30	0.889	0.861	0.875	0.819	0.800	0.810
0.35	**0.903**	**0.849**	**0.875**	0.837	0.792	0.814
0.40	0.916	0.836	0.874	0.853	0.782	0.816
0.45	0.927	0.824	0.873	**0.867**	**0.774**	**0.818**
0.50	0.936	0.812	0.870	0.879	0.764	0.817
0.55	0.945	0.800	0.866	0.889	0.755	0.816
0.60	0.952	0.785	0.861	0.899	0.743	0.813
0.65	0.959	0.765	0.851	0.910	0.727	0.808
0.70	0.966	0.738	0.837	0.921	0.705	0.798
0.75	0.974	0.706	0.818	0.930	0.675	0.782
0.80	0.981	0.659	0.788	0.941	0.632	0.756
0.85	0.988	0.590	0.739	0.950	0.568	0.711
0.90	0.994	0.478	0.645	0.961	0.462	0.624
0.95	1.000	0.292	0.452	0.993	0.290	0.449
Maximum			**0.875**			**0.818**

Table 8-30: *Results for YOLOv3-sdd-4 with respect to the near-infrared test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.727	0.921	0.813	0.612	0.847	0.711
0.10	0.782	0.909	0.841	0.688	0.837	0.755
0.15	0.820	0.897	0.856	0.738	0.828	0.781
0.20	0.849	0.884	0.866	0.775	0.821	0.798
0.25	0.874	0.874	0.874	0.804	0.813	0.809
0.30	0.895	0.863	0.879	0.828	0.805	0.816
0.35	**0.910**	**0.851**	**0.879**	0.845	0.796	0.820
0.40	0.921	0.839	0.878	0.860	0.787	0.822
0.45	0.932	0.827	0.876	**0.873**	**0.777**	**0.822**
0.50	0.941	0.815	0.873	0.884	0.767	0.821
0.55	0.946	0.804	0.869	0.891	0.758	0.819
0.60	0.952	0.789	0.863	0.899	0.746	0.815
0.65	0.958	0.769	0.853	0.908	0.730	0.809
0.70	0.965	0.741	0.838	0.918	0.706	0.798
0.75	0.970	0.707	0.818	0.926	0.675	0.780
0.80	0.976	0.656	0.784	0.934	0.627	0.750
0.85	0.985	0.574	0.725	0.946	0.551	0.696
0.90	0.992	0.458	0.626	0.956	0.441	0.603
0.95	1.000	0.286	0.445	0.990	0.284	0.441
Maximum			**0.879**			**0.822**

Table 8-31: *Results for YOLOv3-sdd-10 with respect to the near-infrared test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.711	0.920	0.802	0.594	0.851	0.700
0.10	0.766	0.907	0.831	0.664	0.835	0.740
0.15	0.804	0.896	0.848	0.711	0.824	0.763
0.20	0.837	0.886	0.861	0.753	0.816	0.783
0.25	0.860	0.876	0.868	0.783	0.809	0.795
0.30	0.879	0.865	0.872	0.808	0.803	0.805
0.35	**0.894**	**0.852**	**0.873**	0.830	0.795	0.812
0.40	0.908	0.839	0.872	0.847	0.786	0.815
0.45	0.919	0.823	0.869	0.863	0.775	0.816
0.50	0.931	0.810	0.866	**0.877**	**0.764**	**0.817**
0.55	0.940	0.796	0.862	0.890	0.754	0.816
0.60	0.950	0.780	0.856	0.902	0.742	0.814
0.65	0.957	0.758	0.846	0.912	0.724	0.807
0.70	0.963	0.731	0.831	0.921	0.700	0.795
0.75	0.969	0.692	0.807	0.928	0.662	0.773
0.80	0.976	0.636	0.770	0.936	0.611	0.739
0.85	0.983	0.559	0.712	0.945	0.537	0.685
0.90	0.992	0.435	0.605	0.958	0.420	0.584
0.95	1.000	0.274	0.430	0.989	0.271	0.426
Maximum			**0.873**			**0.817**

Table 8-32: *Results for YOLOv3-sdd-full with respect to the near-infrared test data.*

τ_{obj}	IoU Threshold = 0.3			IoU Threshold = 0.5		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
0.05	0.717	0.916	0.804	0.603	0.852	0.706
0.10	0.765	0.906	0.829	0.674	0.842	0.749
0.15	0.798	0.897	0.845	0.718	0.835	0.772
0.20	0.824	0.889	0.856	0.752	0.829	0.789
0.25	0.848	0.882	0.864	0.780	0.823	0.801
0.30	0.868	0.873	0.870	0.803	0.816	0.809
0.35	**0.884**	**0.863**	**0.873**	0.821	0.808	0.815
0.40	0.898	0.850	0.873	0.839	0.798	0.818
0.45	0.911	0.836	0.872	**0.854**	**0.786**	**0.819**
0.50	0.923	0.820	0.869	0.868	0.773	0.818
0.55	0.935	0.806	0.866	0.882	0.761	0.817
0.60	0.946	0.786	0.859	0.895	0.745	0.813
0.65	0.954	0.762	0.847	0.907	0.724	0.806
0.70	0.960	0.729	0.829	0.914	0.695	0.790
0.75	0.968	0.688	0.804	0.923	0.656	0.767
0.80	0.975	0.632	0.767	0.931	0.604	0.733
0.85	0.981	0.555	0.709	0.939	0.531	0.679
0.90	0.993	0.437	0.607	0.953	0.420	0.583
0.95	1.000	0.295	0.456	0.965	0.285	0.440
Maximum			**0.873**			**0.819**