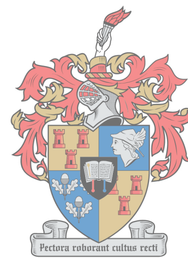
Automated elephant detection and classification from aerial infrared and colour images using deep learning

by

Jacques Charles Marais



UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

100
1918 · 2018

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Applied Mathematics in the Faculty of Science at Stellenbosch University

Supervisor: Dr WH Brink

March 2018

Stellenbosch University https://scholar.sun.ac.za

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2018 ....

Copyright © 2018 Stellenbosch University
All rights reserved.

Stellenbosch University https://scholar.sun.ac.za

Abstract

In this study we attempt to detect and classify elephants in aerial images using deep learning. This is not a trivial task even for a human since elephants naturally blend in with their surroundings, making it a challenging and meaningful problem to solve. Possible applications of this work extend into general animal conservation and search-and-rescue operations, with natural extension to satellite imagery as input source.

We create a region proposal algorithm that relies on digital image processing techniques and morphological operations on infrared images that correspond to the RGB images. The goal is to create a fast and computationally cheap algorithm that reduces the work that needs to be done by our deep learning classification models. The algorithm reaches our accuracy goal, detecting 98% of all ground truth elephants in the dataset. The resulting regions are mapped onto the corresponding RGB images using a plane-to-plane homography along with adjustment heuristics to overcome alignment issues caused by sensor vibration.

We train multiple convolutional neural network models, using various network architectures and weight initialisation techniques, including transfer learning. Two sets of models were trained, in 2015 and 2017 respectively, using different techniques, software, and hardware. The best performing model reduces the manual verification workload by 97% while missing only 1% of the elephants detected by the region proposal algorithm.

We find that convolutional neural networks, as well as the advancements in deep learning, hold significant promise in detecting elephants from aerial images for real world applications.

Stellenbosch University https://scholar.sun.ac.za

Uittreksel

In hierdie studie poog ons om olifante in lugfoto's op te spoor en te klassifiseer, deur van diepleer gebruik te maak. Selfs vir 'n mens is dit nie 'n triviale taak nie, aangesien olifante natuurlik met hul omgewing inmeng, en dit maak die probleem uitdagend en betekenisvol. Moontlike toepassings van hierdie werk strek tot algemene dierebewaring en soek-en-reddingsoperasies, ook met natuurlike uitbreiding na satellietbeelde vir insetbron.

Ons skep 'n gebiedsvoorstel-algoritme wat staatmaak op digitale beeldverwerkingstegnieke en morfologiese bewerkings op infrarooibeelde wat ooreenstem met die kleurbeelde. Die doel is om 'n vinnige en berekeningsvriendelike algoritme te skep, wat die werk van ons diepleer klassifikasiemodelle sal verminder. Die algoritme bereik ons akkuraatheidsdoelwit, en spoor 98% van alle ware olifante in die datastel op. Gevolglik word gebiede afgebeeld na die ooreenstemmende kleurbeelde, met behulp van 'n vlak-na-vlak homografie tesame met heuristiese aanpassings om inherente belyningskwessies (wat deur sensorvibrasies onstaan) aan te spreek.

Ons rig verskeie konvolusionele neurale netwerke af, met verskeie argitekture en gewigsinisialiseringstegnieke, insluitende oordragsleer. Twee stelle modelle is afgerig, in 2015 en 2017 onderskeidelik, met die gebruik van verskillende tegnieke, sagteware en hardeware. Die bespresterende model verminder die werkslading van menslike verifikasie met 97%, terwyl slegs 1% van die olifante wat deur die gebiedsvoorstel-algoritme opgespoor is, gemis word.

Ons vind dat konvolusionele neurale netwerke, sowel as verbeteringe in diepleer, baie belowend voorkom vir die opsporing van olifante uit lugfoto's vir werklike-wêreldtoepassings.

Stellenbosch University https://scholar.sun.ac.za

Acknowledgements

I am grateful to my supervisor Willie Brink for his continued support throughout this thesis, I am thankful for his patience and willingness to always offer advice and guidance.

My gratitude goes to Paul Maritz, SkyReach, and iKubu for providing the data and allowing me to work on this problem for my thesis.

A special thanks goes to the MIH Media Lab for funding this thesis and for providing a stimulating working environment that was shared with wonderful and driven individuals. Thank you to the National Research Foundation for providing additional funding during this study. Thank you to Valohai for allowing me to use their platform to run my experiments.

Finally, I must express my very profound gratitude to my parents and to my partner for providing me with unfailing support and continuous encouragement through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Stellenbosch University https://scholar.sun.ac.za

Contents

1 Introduction **1**
1.1 Creative use of technology 2
1.2 Exploratory data analysis 3
1.3 Our approach 7
1.4 Similar work 8
1.5 Thesis outline 9

2 Background and theory **10**
2.1 Giving a machine vision 10
2.2 Let it learn 12
2.3 The foundation of neural networks 13
2.4 Deep learning and convolutional neural networks 15
2.5 Practical considerations 21
2.5.1 Libraries 21
2.5.2 Hardware 22
2.5.3 Cloud, platforms, MLaaS 22

3 Region proposal **24**
3.1 Infrared blob detection 24
3.2 Image registration 29

4 Implementation and results **34**
4.1 Methodology, evaluation techniques and metrics 34
4.2 Dataset creation 37
4.3 Training from scratch 39
4.4 Transfer learning 46
4.5 System performance 50
4.6 Conclusions on the results 51

5 Conclusion **56**
5.1 Summary and conclusions 56
5.2 Future work 57
5.3 Reflections 57

Bibliography **59**

Appendix **64**

Stellenbosch University https://scholar.sun.ac.za

1 | Introduction

> *"You can't just turn on creativity like a faucet. You have to be in the right mood. What mood is that? Last-minute panic."*
>
> - Bill Watterson, *Calvin and Hobbes*

Man's relationship with nature has changed drastically over the last few centuries. For thousands of years we have lived in harmony with nature, taking only what was needed to survive. As humanity grew in numbers and as our consumption of resources steadily increased, we needed to innovate and efficiently use the scarce resources at our disposal. The rate that we consume these resources has rapidly increased since the 20th century.

Technology has allowed humanity to work less and gain more for the same amount of time and energy. Unfortunately our society no longer only takes from nature what it requires, but instead over consumes and depletes the already limited resources. The International Union for Conservation of Nature (IUCN) estimates that half of the 5,491 known mammal species populations are declining, with 1,131 classified as endangered, threatened or vulnerable [1].

One example of this is the declining elephant population in Africa, primarily due to the large demand for their tusks. As is the story with many other species of wildlife, humanity is decimating the remaining population in pursuit of profit. There is still a strong demand for ivory on the international market, especially in China, where the price of ivory peaked at $2100 per kg in 2014 [2]. The demand for ivory is fuelled by the association of wealth and status, and the ascribed medical benefits it is believed to have. Governments are stepping in and imposing strong sanctions on the open trade of ivory, but there is still a strong belief among many potential buyers that the global elephant population is stable and not at risk of extinction [3]. Contrary to that belief, the Ivory Demand In China report released by WildAid reports that at least 65% of all African forest elephants were poached between 2002 and 2013 [4].

One way to fight this ignorance and intent to buy ivory is to gather factual information that can be used to educate consumers. Awareness is spreading, but there is a lot of uncertainty regarding the size, distribution and trend of the elephant populations in Africa. A lot of the countries in Africa have outdated, inaccurate or non-existent census data, making it impossible to accurately estimate the state of the elephant population. The Great Elephant Census (GEC) [5] is a philanthropic effort by Paul G. Allen to do the first-ever pan-African survey of savanna elephants [6]. The project was started in 2013 and was mostly completed by 2016, and aimed to provide accurate data to support the growing concerns of elephant population decline across the African continent.

As we will see below, counting elephants is both time consuming and a costly exercise. The purpose of this thesis is to look at an alternative approach to the one employed by the GEC and use technology to reduce the cost and increase the frequency of such a census.

The GEC found that the overall elephant population declined by 30% between 2007 and 2014, with a current 8% per year accelerating population decline. The GEC surveyed 18 countries and counted 352,271 elephants. This number represents at least 93% of elephants in these countries. The majority of the elephants were counted in legally protected areas, though a high number

Stellenbosch University https://scholar.sun.ac.za

of poached carcasses were still found in these protected zones. There is a large cost associated with an endeavour as ambitious as this. The GEC accumulated 9,700 hours of flight time, with an average flight lasting nearly three hours. The census was conducted with a crew of 286 members using 81 airplanes, assisted by 90 scientists and six NGO partners. The survey planes flew a total of 463,000 km to sample 24% (218,238 km^2) of the total ecosystem area. The GEC had a $7 million budget donated by Paul G. Allen to sustain the initiative over its three year life [6]. Country-by-country findings attempted to estimate population trends using the carcass ratio, defined as the number of carcasses counted against the total elephants counted in the region. Due to the difficulty of spotting and identifying a carcass (above the initial difficulty of spotting a living elephant), the carcass ratio is considered an underestimation of the true carcass count. The results show an overall carcass ratio of 11.9% (1 in 10 elephants counted were found dead), which indicated overall population decline on the continent. Countries such as Cameroon and Mozambique showed the highest carcass ratios, indicating a high rate of population decline and is a cause for concern. Countries such as South Africa and Uganda showed stable and growing populations, which can be attributed to the efforts made to protect their respective elephant populations [7].

The GEC has provided invaluable information on the current state of affairs. We now have a better understanding of the overall situation, as well as country specific trends. We can identify the patterns and areas that require immediate attention, but we do not have a viable means to track the impact that these changes will have. Given the massive time and cost associated with the project, it seems infeasible to repeat the census on a regular basis. We are forced to optimise and efficiently use the limited time and money available to us. We look to a technological solution that would reduce the cost of capturing the same amount of data in the future.

The highest cost to the current method is the human component. The current method relies on multiple human spotters in the airplanes who manually count elephants that they can see from the airplane window. The reliance on humans also limits the speed, altitude and distance of the aircraft. A technological solution would ideally make use of aerial imagery, obtained from either a satellite or drone surveying the area. This would eliminate the human component, shifting responsibility to a detection and classification algorithm that would ideally detect and count elephants in real time. The ideal solution would rely on cutting edge hardware, access to specialised data and potentially a high budget.

1.1 Creative use of technology

The idea was born to create an affordable and simple capture solution that relies on the advancements in computer vision and machine learning to build an affordable detection and classification platform. The goal of this thesis is to research and test the following hypothesis.

> *Is it possible to build an affordable and usable detection and classification pipeline that relies on computer vision and machine learning, using data captured by a simple and affordable capture solution, to provide a human with a reduced set of options to verify?*

The capture solution is not in the scope of this thesis, but a brief overview of the minimum viable product used to capture the thesis dataset can be given. The solution relies on a single pilot flying a light aircraft over an area in a transact pattern while the capture rig automatically captures data at recurring intervals. The aircraft used was a two seater BushCat Light Sport Aircraft that was manufactured in Africa by SkyReach. The passenger seat was removed from the aircraft and a capture rig was installed in its place, allowing aerial images to be captured through a hole in the floor of the aircraft. The capture rig consisted of a Canon EOS D6 that served as the visual and GPS sensor, an ICI 7640 long range infrared capture sensor, and an LTI TruSense S100 laser

Stellenbosch University https://scholar.sun.ac.za

Figure 1.1: Core components used to build the prototype capture system. Pictured is the BushCat Light Sport Aircraft, followed by (from left to right) the ICI 7640 thermal sensor, the TruSense S100 altitude sensor, and lastly the Canon EOS D6 visual sensor.

altitude sensor, as shown in Figure 1.1. A capture script was run on an on-board laptop that was responsible for device synchronisation, data capture and storage.

The captured data is stored on an external hard drive, ready for processing and conversion. The RAW RGB files are converted to TIFF format, while the infrared temperature values are normalised and mapped to a greyscale $[0, 255]$ range. The converted images, GPS data and altitude data are stored together, ready for use in this thesis. Thus the scope of this thesis begins with the availability of the converted visual and IR data.

The first data captured using this setup was of cattle on a local farm, pictured in Figure 1.2. We notice some vignetting and uneven exposure on the IR images, caused by the small size of the viewing hole, and heat generated by the aircraft. These issues were addressed when the thesis dataset was captured. Once the initial tests were completed and some minor adjustments made, the first elephants were captured. Before we discuss the approach we took to build the proposed system, let us take a in-depth look at the data we will be working with in this thesis.

1.2 Exploratory data analysis

Before we start building algorithms, we need to understand the data we have at our disposal. Doing an initial exploratory data analysis is a key starting point in the data analytics pipeline, because it allows us to make initial assumptions about the data, including the overall fidelity and

Stellenbosch University https://scholar.sun.ac.za

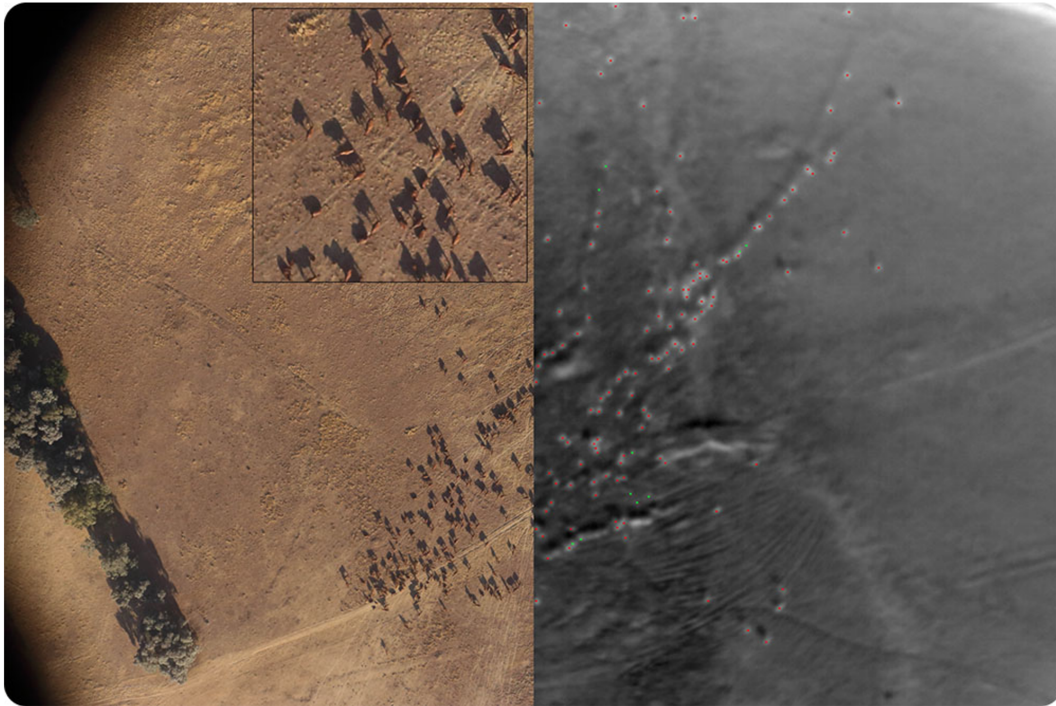


Figure 1.2: One of the first images captured with the prototype system of a herd of cattle on a local farm in South Africa. On the left is the visual band at 200m altitude, on the right is the thermal image.

completeness.

The dataset consists of 890 images (samples), including 121 images that contain at least one elephant (positive samples), 769 images that contain an animal that is not an elephant (other samples) or images that do not contain any animals at all (negative samples). Each sample consists of a 20MP RGB image with a resolution of $3,600 \times 5,400$ (Figure 1.3), an IR image with a resolution of 480×640 (Figure 1.4), the time of capture, the GPS coordinates of the aeroplane, and two altitude readings, one from the RGB camera and one from a dedicated laser altimeter.

The capture flight path can be seen in Figure 1.5. The erratic and looped path would never be used in a live capture mission, but since the main focus was to capture as many examples of elephants as possible, the pilot was instructed to do multiple flyovers when an elephant was spotted. One of the biggest challenges we face with this thesis is the relatively small dataset we have to work with, and even at this point we are using a form of data augmentation by capturing the same elephant groups at different passes. We can safely assume that the samples vary enough between captures that very little bias is introduced into the dataset at this point, since the elephants most likely moved, along with the angle and altitude of capture that would be different. The capture time of the dataset is constrained by the ambient temperature, since the difference between the ambient temperature and an elephant's core temperature needs to be at least $4°C$ for the elephant to appear on the captured thermal image. This poses an interesting challenge, since the only time to capture images would be in the early morning while the ground is still cool, but there is enough light to capture the elephants. Over the two hour flight, the sun will gradually heat up the ground, also decreasing the visibility of the elephants in our thermal images towards the end of the capture session.

The colour changes in the flight path (Figure 1.5) indicate the change in altitude as the images were captured. Using the dedicated altimeter, we can plot the altitude above ground for

Stellenbosch University https://scholar.sun.ac.za

Figure 1.3: Randomly sampled RGB images from the dataset. Most of the landscape is covered with green and grey foliage, with patches of brown soil occasionally breaking through.

each capture over the duration of the flight, as seen in Figure 1.6. The first 300 images in the dataset represent the long tail starting in the lower left corner of Figure 1.5. Once a group of elephants is spotted, the pilot starts looping over the group. Each positive sample is represented by a red marker on Figure 1.6. We notice the average altitude of the entire flight (represented by the green bar) is almost the same as the average positive capture altitude (represented by the red bar). The high variance in the altitude over the flight duration and between captures adds an interesting element to our problem. With the altitude fluctuating between 150m and 300m, the size of the elephants in the image also fluctuates significantly. Under ideal conditions we would like to keep the altitude constant, allowing us to construct a traditional computer vision based filter constrained on size and shape to extract the elephants from the images. We can see that this approach will not work here, since we need a scale invariant approach to combat the high variance in altitude and target size.

After capture and conversion, the dataset was hand annotated. Each elephant across all the infrared images was marked and hand classified as either visible (adult elephant that is fully visible in the corresponding RGB image), calf (a young elephant that is fully visible in the corresponding RGB image), partial (an adult elephant that is partially visible, and partially obscured from view in the corresponding RGB image), and lastly hidden (an elephant that is not visible at all in the corresponding RGB image). Examples of each can be see in Figure 1.7. The dataset contains 524 visible elephants, 39 calves, 312 partially visible elephants, and 119 fully hidden elephants. The true elephant count in the park is around 100 elephants and they move around in three main groups, thus we have increased the dataset size nearly tenfold by using the multiple-flyover technique.

Stellenbosch University https://scholar.sun.ac.za

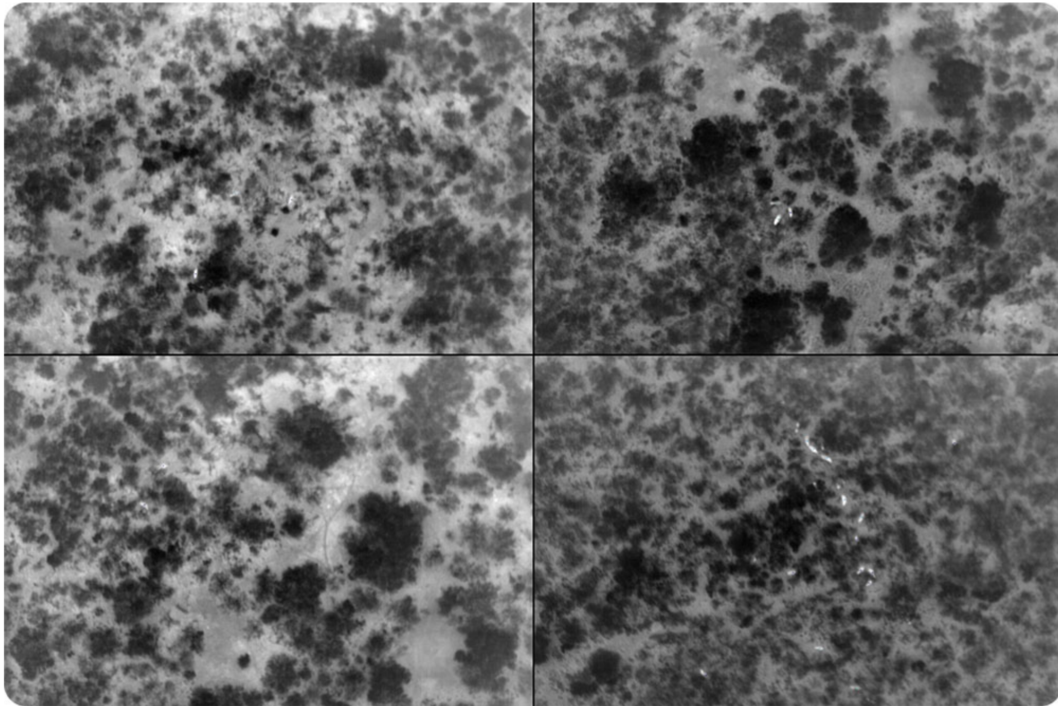


Figure 1.4: Randomly sampled thermal images from the dataset. The darker regions represent the cooler foliage, while the lighter regions represent the hotter soil. All of the images contain elephants, visible as white blobs.

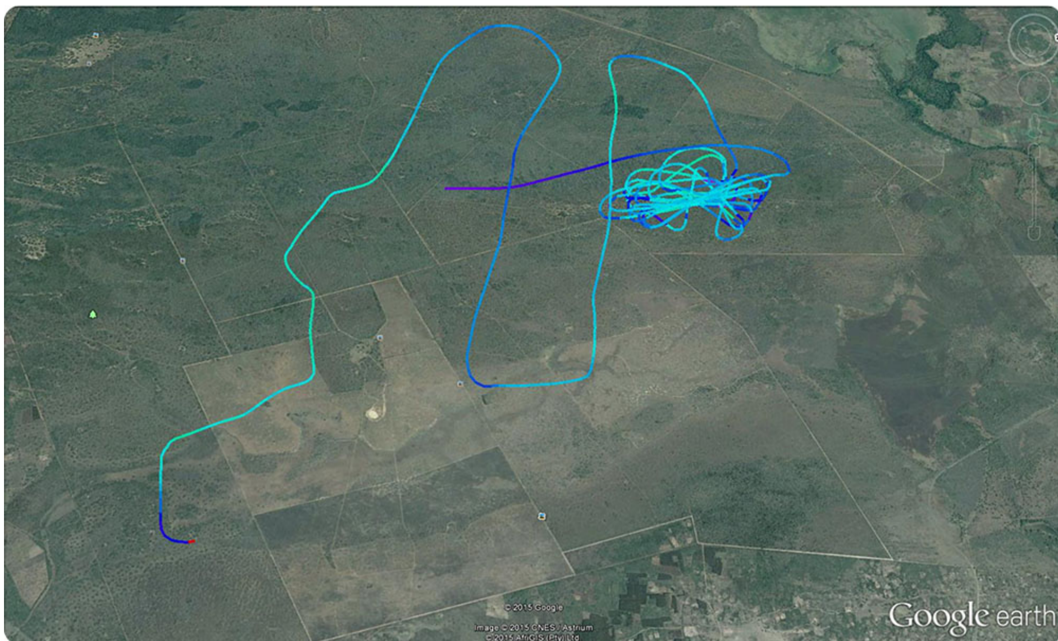


Figure 1.5: Using the GPS data, we plot the flight path used to capture the dataset. Once a group of elephants was spotted, the pilot was instructed to do multiple passes, resulting in the set of tight loops visible in the image.

Stellenbosch University https://scholar.sun.ac.za

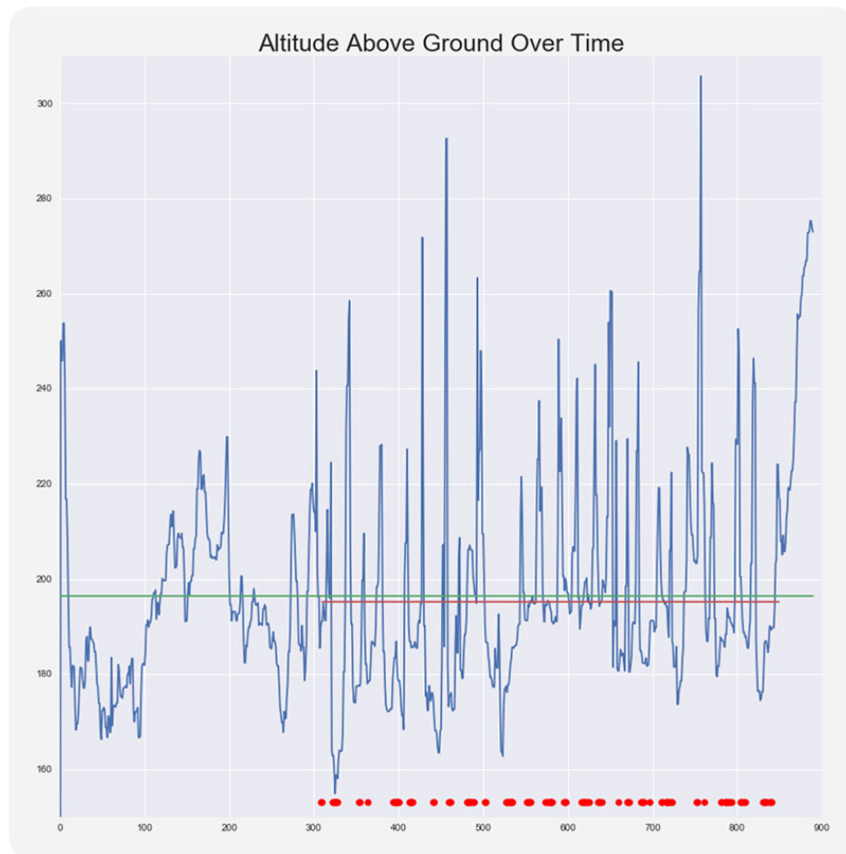


Figure 1.6: By plotting the sequential altitudes over the duration of the flight, we show the high variance present in the altitude, and as a result the high variance in target size and temperature. The red marks indicate images that contain elephants. We can see the altitude fluctuates between 150m and 300m.

1.3 Our approach

We can see from the RGB images in Figure 1.3 that manually detecting an elephant at these altitudes is challenging even for a human. With the aid of the thermal images in Figure 1.4, it becomes quite easy to find the elephant shaped blobs and search the corresponding area in the RGB image. This idea forms the foundation of our approach to the problem.

We propose a two phased approach to detecting and classifying elephants from our dataset. Instead of attempting to detect the elephants directly in the RGB image (which is computationally expensive, and comparable to the overwhelming feeling we get by searching for an elephant in the RGB image), we use the IR images to first determine which regions are most likely to contain an elephant before passing those regions onto a classifier.

Figure 1.8 shows where our solution fits into the pipeline. We take the converted and annotated data, attempt to reduce the search space of the problem through region proposal before classifying the regions of interest with a trained convolutional neural network. One of the main objectives at each step of the pipeline is to reduce the number of false negatives (without losing an elephant along the way). With this in mind, we will attempt to build a simple and fast region proposal phase with a very high true positive rate, while leaving the true decision power for the classification phase. The goal is to provide a human with a reduced set of options for manual

Stellenbosch University https://scholar.sun.ac.za

Figure 1.7: Thermal and visual example pairs of the label categories. Elephants must be clearly visible in both bands to be labelled as visible, the same holds for calves (calf). Elephants occluded in the visual band are labelled as partial, while elephants only present in the thermal band are labelled as hidden.

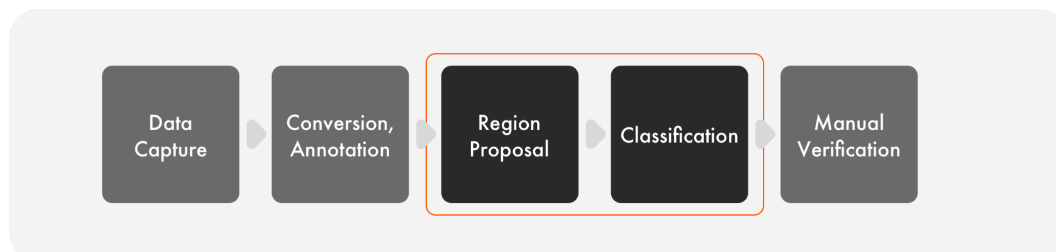


Figure 1.8: Data capture, format conversion, and manual verification are necessary steps in the pipeline, but are not in the main scope of this thesis. Our focus is on using the annotated dataset and investigating region proposal strategies that will be used along with a deep learning classification model to detect the elephants.

verification after the classification phase.

1.4 Similar work

Similar work has been done in the field of animal detection, some using deep learning and aerial drone images to detect marine mammals [8], others using motion detection cameras in the wild along with deep learning [9]. A more modern approach is to use region proposition based con-

Stellenbosch University https://scholar.sun.ac.za

volutional neural networks [10].

The current thesis contributes to this body work through its combination of traditional computer vision techniques for region proposal on infrared images and current state-of-the-art deep learning feature extraction methods for classification.

1.5 Thesis outline

The next chapter of this thesis takes a look at the history and theory of computer vision and machine learning that makes our machine vision solution feasible. We then look at ways to process the data to improve our model performance and accuracy, before moving onto the actual implementation, experimentation, and results achieved. We draw a conclusion on the outcome, propose potential future work and reflect on the entire process.

In Chapter 2 (*Background and theory*) we take a theoretical look at core computer vision and machine learning concepts that make the proposed solution possible. We start with classical computer vision and focus on its feature engineering properties. We move onto machine learning, starting with neural networks and transition into the deep learning paradigm, remaining focused on relevant machine vision components. The chapter ends with an overview of software implementations of the discussed theory.

Chapter 3 (*Region proposal*) focuses on reducing the complexity of the classification task at hand. Using the infrared data, we can reduce the problem space that needs to be searched in the visual band data. We investigate various ways to detect dim targets in the thermal data. Once the targets have been selected, we look at registering the thermal and visual images together. We end the chapter with input that is ready to be used by our deep learning models.

In Chapter 4 (*Implementation and results*) we look at various ways to create an elephant classification model. We start by training a model from scratch, with randomly initialised weights and experimenting with different model architectures and hyperparameters. We then experiment with fine-tuning an existing model, which is a widely used technique called transfer learning. After putting everything together, we evaluate the system performance of the different models.

Chapter 5 (*Conclusion*) ends off the thesis by presenting conclusions, proposing possible extensions to the work and reflecting on the process.

Stellenbosch University https://scholar.sun.ac.za

2 | Background and theory

"Do Androids Dream Of Electric Sheep?"

- Philip K. Dick

In this chapter we explore how we quantify vision and give a computer the ability to understand images, in a sense allowing a machine to see. We briefly give an overview of computer vision and why it is such an important field in today's world. We then move onto machine learning, specifically looking at neural networks and how their modern application as deep neural networks has disrupted multiple fields, including computer vision. From a theoretical viewpoint we move onto an overview of the practical aspects associated with deep learning, from the most popular libraries to the hardware choices that play a role in the training process.

2.1 Giving a machine vision

Vision is arguably considered the most important sense that we possess. A significant portion of the brain is involved in processing and reacting to visual information [11]. In the pursuit of optimisation and innovation, we would ideally like to allow a machine to also take advantage of all the visual information available in the world.

As intuitive as seeing is for a human, the same does not apply to a machine. For a machine to gain access to visual information, we need to give it a window into the world. This window would be obtained from some form of sensor that converts the analogue visual band into a digital format that it can work with. This format is typically one or more digital images in either colour or greyscale.

A digital image is represented as a grid of values called pixels. For a colour image the grid is a 3-dimensional matrix with a width, height, and depth (also called channels). The width and height correspond to the sensor that was used to capture the image, and the resolution of the sensor is usually specified in megapixels (MP). The depth of the grid depends on the colour space being used to represent the image, with red-green-blue (RGB) widely used in applications. Each pixel is a quantised representation of the analog light signal that was absorbed by the sensor at that point. Typical pixel fidelity would be 8-bits, resulting in three values ranging between 0 and 255, one for each colour channel. Sensors with higher bit rates allow for more accurate colour representation, but at the cost of higher storage requirements. The representation of visual information as a grid of pixels is illustrated in Figure 2.1.

Assuming a standard digital image with dimensions $64 \times 64 \times 3$ at 8-bit resolution, we will have 12,288 values that represent the image. From this representation, we would like the computer to understand what is happening in the image. This is an enormous task that has been a topic of research for many years.

The naive approach would be to consider how a human makes sense of visual information and manually model the behaviour. If we take an image of an elephant looking to the right, we would identify the parts that make up the whole. We would look at the distinct features that

Stellenbosch University https://scholar.sun.ac.za

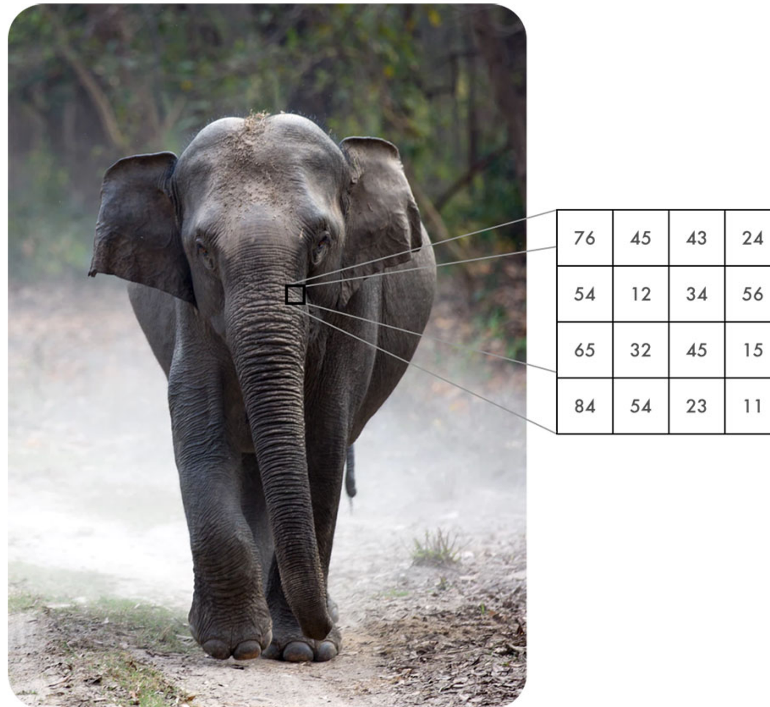


Figure 2.1: Example of the pixels that make up a digital image, visualised is a single channel of values that make up the highlighted image patch.

make up an elephant. Considering the core features of an elephant, we might list ear shape, eye form and presence of tusks as the most important identifiers. Using this information, we would attempt to describe the shape of an ear in terms of pixels and how they relate to each other. It would take many hours and multiple iterations, but we would end up with a system that could identify our elephant image as an elephant. Given a second image of another elephant that is looking slightly up, out system would no doubt fail to identify the image as an elephant. This is an example of overfitting our model to our data, resulting in a system that cannot generalise to the seemingly infinite variations of images that contain an elephant.

We just attempted to manually encode the characteristics of an elephant, a process known as feature engineering. Historically this has been the best known way to approach the problem, resulting in hours spent building models that have value in very narrow problem domains. Even for a very specific problem such as identifying an elephant in an image, we face viewpoint variations, scale variations, deformation, occlusion, illumination conditions, background clutter, and inter-class variations [12], as illustrated in Figure 2.2 through the use of cats.

For a system to be useful, it should be able to handle these and many more variations. Building such a system by manually describing every imaginable situation would be infeasible. Thus we need to rethink our approach.

A human child that sees a cat for the first time would be able to associate all of the above variations as still being a cat, without explicitly having seen a cat in each of those situations. This leads us to our second approach; what if there was a way for a machine to learn what a cat is, without explicitly defining its features. This approach can be called feature extraction, and it

Stellenbosch University https://scholar.sun.ac.za

Figure 2.2: Images from a single image class can contain many variations which make it difficult to describe the class by hand. We see cats and kittens in various positions, behind objects and under different illumination conditions.

requires that we build a model that learns a fundamental representation of visual information. To do this, we require a significant amount of data. This is one of the reasons why this approach has only become feasible during recent years, with the growth of the internet and availability of millions of images.

Now that we have established how a computer represents the visual world in the form of images and come to the conclusion that it is really difficult to explicitly explain to a computer what is going on in an image, we can investigate ways to make a computer learn meaning and representation on its own.

2.2 Let it learn

Learning is so intuitive that we rarely stop to think what it really means to learn. For a computer we would define it as an algorithm that learns from a set of data [13]. A formal definition was given by Mitchell [14]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

To continue our example, our current task is to identify elephants in a set of images. The experience consists of all the different images in the set, and the performance can be calculated as how accurately the model identifies the elephant images as elephants and the non-elephant

Stellenbosch University https://scholar.sun.ac.za

images as not elephants.

There are a lot of different tasks in machine learning. The most relevant task to this thesis, and the only one we will focus on, is the task of classification. In a classification task we ask the algorithm to assign a probability that a specific image belongs to one of a finite set of categories. In our example we have a binary classification task where the image is either designated as elephant or not elephant.

Goodfellow et al. [13] define a simple recipe for a machine learning algorithm as the combination of a dataset, a cost function, an optimisation process, and a model. We can break up each of these components to gain an understanding of how they make up the learning process. The following is a high level overview of the learning process, and the sections that follow will go into more detail.

Starting with the dataset, we first need to make the distinction between supervised and unsupervised learning. With the former our experience is a set of images with manually assigned labels. That is to say each image in the dataset has its relevant category assigned to it before the learning process starts. This allows us to calculate the performance and learn from the experience by comparing the predicted result (the task) with the known label of the image. With unsupervised learning we are given an unlabelled dataset. This means that we have no class information of the images before the learning process starts and thus we do not have a simple way to measure the performance of the classification task. The scope of this thesis is mainly on supervised learning and we will not go into further details regarding unsupervised learning.

Given a labelled dataset that consists of elephant and not-elephant images, our goal is to end up with a model that can accurately classify the images into the correct categories. We start off with the assumption that a set of rules exist that will satisfy this goal. This is a realistic assumption, simply taking into account that is seems to be a trivial task for a human. With this assumption in mind, we attempt to mathematically model a similar system in code. In its simplest form, this is represented as a mathematical function $f(x) = y$, where x is the input to our model, y is the class prediction and f is our model. Our model f can take many forms, and one of the biggest challenges in machine learning is to find the most appropriate model for a given problem. For the moment accept that f is chosen appropriately for our elephant classification challenge. For our given model, we want to find the ideal parameters for our function that will correctly classify our dataset.

The optimisation process is used to iteratively "learn" the ideal parameters. As mentioned earlier, our system learns from an experience when the performance on a task increases. Thus we need a way to measure the performance of our model (our task) on the dataset (our experience). We can define a cost function that will act as the performance measure. This cost function is usually defined as a distance metric that can quantify how well a model can predict the correct classes for the images in the dataset.

With our recipe in hand, we move to a class of models that hold the most promise for image classification: the neural network.

2.3 The foundation of neural networks

The universal approximation theorem states that a neural network with a single hidden layer of finite neurons and any continuous sigmoidal nonlinearity can approximate any continuous function of n real variables [15]. This means that structured high-dimensional data such as languages, sounds, and images can be approximated and represented by a neural network model.

To understand neural networks, we must first discuss what inspired their design and what potential they might hold. The idea of artificial intelligence has been with humanity since ancient times, such as the golem made from mud, in search of intelligence and the divine [16]. In the spirit of biomimicry, we look to our own minds in search of a solution to intelligence. The idea of modelling the human brain was first attempted in modern times in 1943 by McCulloch and Pitts

Stellenbosch University https://scholar.sun.ac.za

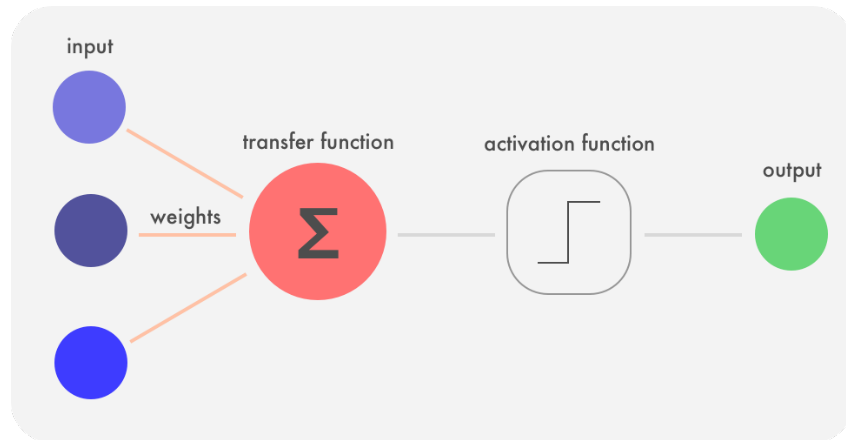


Figure 2.3: A model of a neuron used in a perceptron. The blue input values are combined with the connection weights in the red transfer function, and the real valued output from the transfer function is either activated (1) or not (0) in the activation function using a set threshold value. The activation value is passed along in the network as the green output value.

[17], when they modelled a simple neural network using electrical circuits. This led Rosenblatt [18] to create a mathematical model (a perceptron) that attempts to explain how the neurons in our brain works. A perceptron consists of multiple neurons (Figure 2.3), each taking a set of binary inputs (either model input or from other neurons), multiplying these input values by continuous learned weights, and finally thresholding the sum of these weighted values either to activate the neuron (with a value of 1), or to keep it dormant (with a value of 0). It can be shown that a perceptron can model basic boolean gates, which seems to imply that it is capable of logical reasoning and ultimately artificial intelligence. With a simple model of the brain at hand, a natural question arises: how do we get our artificial brain to learn? Backpropagation was proposed in the early 1960s and has been of interest in academia since, but it was in 1989 that LeCun et al. [19] demonstrated the real world feasibility of backpropagation by training a neural network to recognize handwritten digits. With the continuous advances in technology along with popular acceptance of backpropagation, we rapidly advanced towards more complex models and techniques, soon arriving at what we today know as deep learning.

Before we move to deep learning concepts, we take a brief look at the most important neural network components that we will be using to solve our classification problem.

The feedforward neural network (also called a multilayer perceptron) is a fundamental deep learning model, taking an input x, passing it through the model f to get output y. This is done in a single pass without any feedback back into the function, earning it the feedforward name [13]. Feedforward neural networks are widely used in machine learning and are a foundation component of deep learning and convolutional neural networks.

The feedforward neural network (Figure 2.4) consists of one or many hidden layers that contain multiple neurons. Each neuron contains a weight vector that is initialised with random values [20], but is tuned to the classification problem at hand through training. Every neuron has an activation function that determines its output value, given all input values and its internal weight vector. Popular choices for the activation function include the sigmoidal nonlinearity [15] and the rectified linear unit (ReLU) [21]. To train these weights we used an optimisation process such as stochastic gradient descent (SGD) [22] along with backpropagation [21] to iteratively minimise our cost function. During training the cost function measures the error between the classified output label and the actual output label, and the cross entropy loss [13] is a popular choice.

Training a feedforward neural network is a supervised learning task, requiring a labelled training set to optimise against, as well as a validation set to estimate the model performance and to

Stellenbosch University https://scholar.sun.ac.za

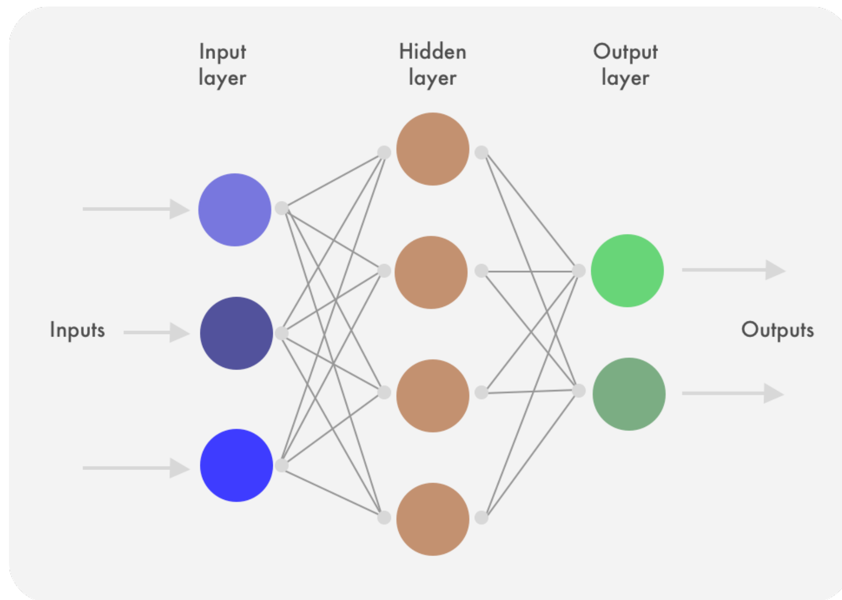


Figure 2.4: Simple graph representation of a feedforward neural network. The blue input nodes in the input layer are connected to all of the brown hidden nodes in the hidden layer, while all the brown hidden nodes are in turn connected to all of the green output nodes in the output layer. The information flows from one layer to the next, never flowing back into a previous layer.

guide hyperparameter selection during training [13]. To prevent the model from overfitting we use techniques such as L_2 regularisation [13].

With our foundation set, we dive into deep learning and more specifically the convolutional neural network models that we will be using to train our classifier.

2.4 Deep learning and convolutional neural networks

Deep learning is a direct extension of neural networks, since the concept of a deep neural network comes from a neural network with multiple hidden layers. Even the arguably shallow two layered neural network could be considered deep. Deep learning is not a new concept, but has been limited by input data volume and computation power, both of which are no longer a restrictive issue. The recent growth in deep learning interest started with the seminal paper by Krizhevsky et al. [21] in 2012. They showed how the hierarchical structure of a deep neural network could be used to learn feature representations from visual images. The model performance was significantly better than any of the previous feature engineered methods employed in earlier years. An in-depth review of deep learning is given by LeCun et al. [23]. This was also the starting point for us when we started researching the problem of image classification using deep learning. Convolutional neural networks introduce the convolutional layer into our existing feedforward neural network toolbox, and allows us to work with visual data and ultimately train our elephant classifier.

We will look at the three most important layers in a traditional convolutional neural network used for classification: the convolutional layer, the pooling layer, and the fully connected layer.

Convolution is a simple mathematical operation that is used to combine (or convolve) information. In computer vision we use convolution in two dimensions by moving a small convolution kernel over the input image to produce a new output image, as shown in Figure 2.5. The convolutional layer consists of multiple convolutional kernels (also known as filters), which are

Stellenbosch University https://scholar.sun.ac.za

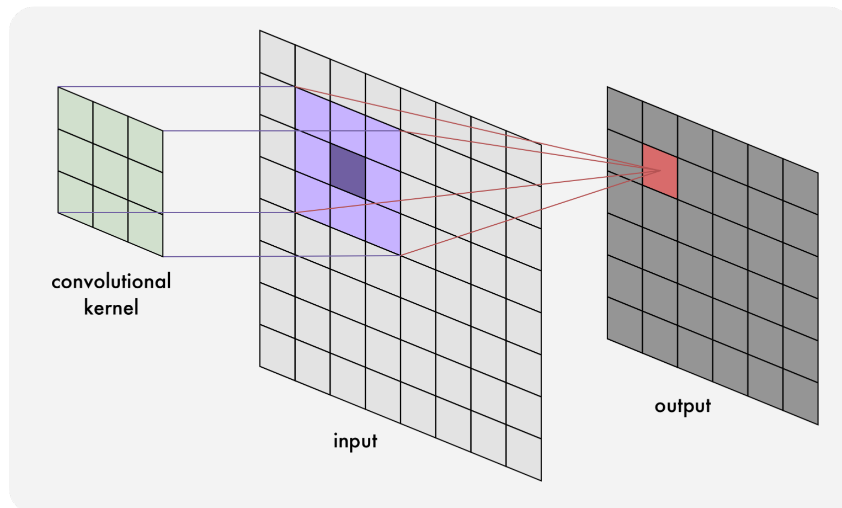


Figure 2.5: An example of how an 8×8 input image is convolved into a 6×6 output image when a 3×3 convolutional kernel is applied using a stride length of 1 across the entire input image.

the layer weights that need to be tuned during training to extract and activate on relevant features in the input image. Each convolutional layer in a network can flow into another convolutional layer, which allows the network to learn hierarchical features [13]. Starting with simple image parts such as edges and gradients in the first layers, we end up with complex object parts such as elephant body shapes. An example of a single filter applied to our elephant images is shown in Figure 2.6. It seems to successfully extract the shape information of an elephant (amongst other shapes). The size of a convolutional kernel is called the receptive field. By learning filter weights that move across an input image we are locally connecting our filter weights to the input. This reduces the number of parameters we need to train (called parameter sharing), and allows us to train features that are translation invariant. We specify the depth of a convolutional layer to determine the number of unique features we would like the layer to learn, and this depth is a hyperparameter of the layer. Another hyperparameter is the stride length, which specifies by how many pixels the filter will move between every convolution operation. A larger stride size will result in a smaller output (as well as more information loss between layers). Since a convolution operation results in information reduction we typically end up with a smaller output compared to our input. By zero-padding the output we can control its size as it moves through our network. Each convolutional filter output is called a feature map. All feature maps created by a convolutional layer pass through an activation function to produce an activation map [12], sometimes called the detector stage [13]. This is the same nonlinear activation function discussed earlier, with ReLU being a popular option due to its positive impact on training times. ReLU can be implemented as a simple threshold operation, compared to the more computationally expensive activations, such as the sigmoidal function.

The pooling layer (also called the subsampling layer) is used to reduce the spatial size between different layers [24]. There are different pooling strategies such as average pooling, L_2-norm pooling, and the most popular max pooling (shown in Figure 2.7). All of them result in information reduction, reducing the number of network parameters and computations required by subsequent layers. The need for pooling layers is debated, since they remove a lot of information from the network. It can be considered beneficial, since it acts as a form of regularisation to reduce model overfitting as well as makes our network invariant to small translations in the input [13]. Due to the increase in feature maps created as we go deeper in our network we need to pool between convolutional layers to prevent parameter combinatorial explosion.

Stellenbosch University https://scholar.sun.ac.za

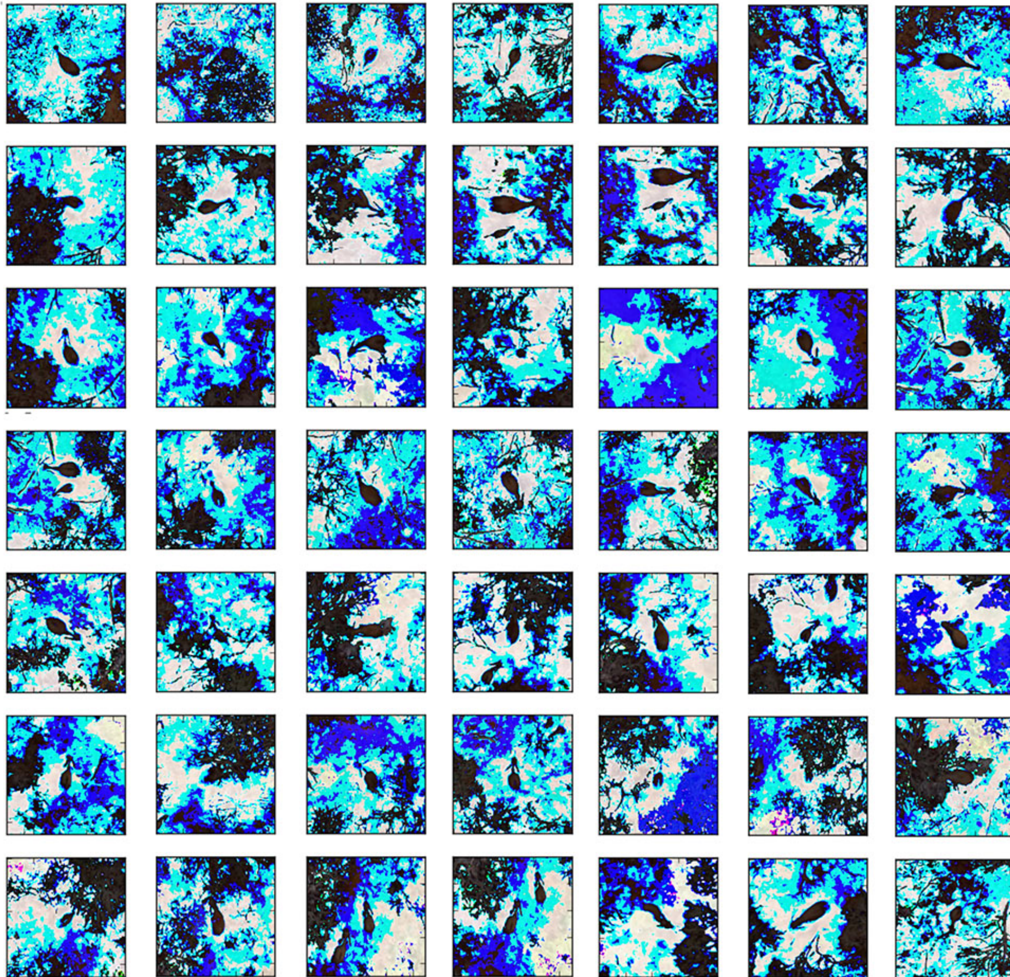


Figure 2.6: We selected a single convolution filter that looked promising from one of the first neural networks that we trained. We convolved a few of the training images with this filter and obtained this result. It is interesting to note how a single 3×3 filter can extract so much shape information from the images. This result motivated us to continue investigating deep convolutional neural networks as a solution.

Stellenbosch University https://scholar.sun.ac.za

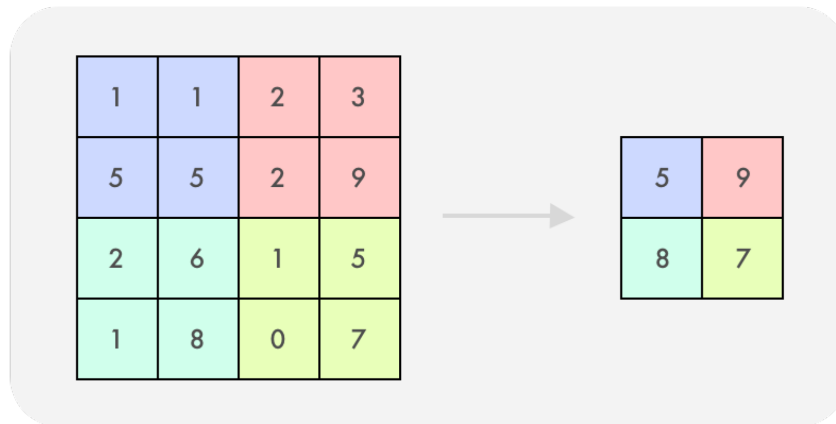


Figure 2.7: We illustrate a 2×2 max pooling operation with a stride length of 2 pixels. Similar to the convolution filter, we move the pooling window across the input and use the maximum value found inside the pooling window as the output. By doing this we introduce translation invariance into the model, as well as reduce the number of parameters in the network by reducing the activation map sizes being used as input by subsequent convolutional layers.

A fully connected layer (also known as a dense layer) is the same hidden layer discussed in the previous section under feedforward neural networks. Compared to the local connectivity of the convolutional layer, a fully connected layer connects every one of its neurons with every input from the previous layer. Each neuron has a weight vector that needs to be trained and an activation function that determines its state after multiplying the neuron input with its weights. Given the fully connected nature of the layer, we see a growth in parameters in the network (since each neuron in the layer needs to be connected with every activation map). A fully connected layer is usually used as the final layer in a convolutional neural network, and their activations correspond to the higher level features detected in the activation map from the previous layer. These activations are connected to a softmax classifier, which applies logistic regression to the activations to calculate class probabilities for the input image. These class probabilities correspond to normalised real values that represent the confidence of the input belonging to the given class [25].

Given the three layers just discussed, we can build the simplest architecture that takes an input image, passes it through a convolutional layer that creates multiple activation maps, reduces the map size though pooling, and passes the activations through a fully connected layer that classifies the input into classes using a softmax classifier (as seen in Figure 2.8).

We can use these building blocks to create increasingly complex and deep networks. One of the earliest successful applications of a convolutional neural network was the LeNet network [26] in 1998 that consisted of seven layers, a small extension to the basic model shown in Figure 2.8 by adding a second convolution and pooling pair. In 2012 Krizhevsky et al. created the eight layer deep AlexNet [21], winning the ImageNet ILSVRC 2012 challenge and starting the age of deep learning. AlexNet was similar to LeNet, except with more parameters, a slightly deeper architecture, and stacked convolutional layers. VGGNet [27] was the runner-up in the ImageNet ILSVRC 2014 competition and showed that increasing the depth of the net had a positive impact on the model performance. Two variations were trained, namely VGG16 with 16 layers and VGG19 with 19 layers. All of these models are visualised in Figure 2.9. In 2015 Google created InceptionV3 [28], which was an improvement on the original GoogleLeNet [29] that won the ImageNet ILSVRC challenge in 2014 with a 22 layer deep architecture. The GoogleLeNet and Inception architectures strayed from the accepted format of stacking layers sequentially by splitting and joining (concatenating) the network at various depths, as seen in Figure 2.10.

Stellenbosch University https://scholar.sun.ac.za

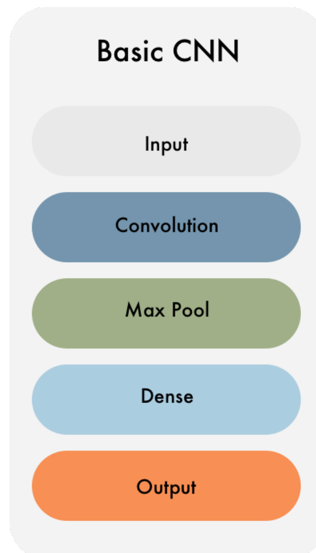


Figure 2.8: The simplest convolutional neural network architecture we can build that includes our foundation convolution, pooling, and fully connected (or dense) layers. The convolution layer includes an activation function. The output layer represents the softmax classifier that predicts the class labels using the activations in the previous dense layer.

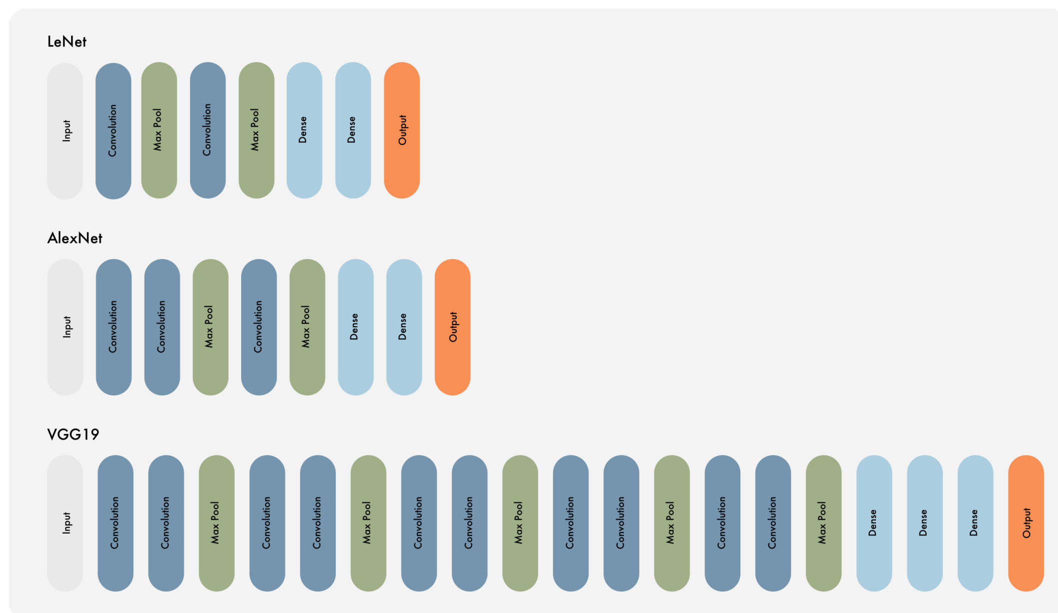


Figure 2.9: An illustration of popular architectures that have influenced the direction of deep learning research. LeNet was the first to show the value of convolutional neural networks in a image classification task. By winning the ImageNet ILSVRC 2012 challenge, AlexNet showed the potential that deep learning holds. VGG19 continued the tradition of building deeper and more complex models, showing that the depth of an architecture had a notable impact on performance.

Stellenbosch University https://scholar.sun.ac.za

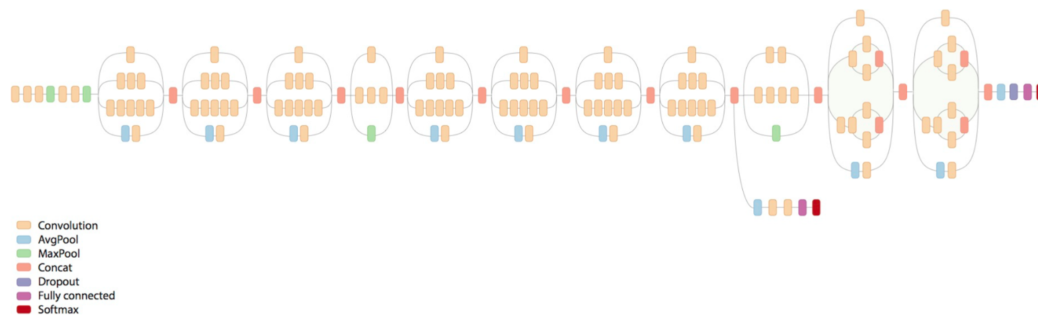


Figure 2.10: InceptionV3 schematic diagram showing the split and join operations throughout the network architecture. Image from [30].

Before an architecture can be used it needs to be trained. Once a network is built all its weights and biased need to be given initial values. There are multiple strategies that include setting all weights to zero (all neurons initially calculate the same result, which makes this a bad choice), selecting random values (which will allow the neurons to calculate unique results from the start), uniform distribution initialisation (a common heuristic which uniformly generates and distributes the weights), along with more advanced methods such as Xavier initialisation [31] and ReLU initialisation [32]. These strategies of weight initialisation are called training from scratch. Another strategy is to use weights from pre-trained networks, such as AlexNet or InceptionV3. This is called transfer learning and usually provides the fastest route to a well performing model [33]. Transfer learning can be used in various ways, the simplest being to train a classifier on the different layer activations generated by predicting the values of a training set. The existing weights can also be modified, or fine-tuned, to another problem domain.

Both training from scratch and fine-tuning require a training strategy. This strategy iteratively attempts to improve the weights on the model in order to minimise a set cost function [13]. This cost function is the error between the predicted label and the actual value, and a common choice is to use the softmax prediction values and to calculate the cross entropy loss [13] to measure the intensity of the prediction, in other words how correct or incorrect the prediction was. The strategy consists of one or more epochs, where an epoch is completed when the model has seen all of the training images once [13]. The model sees the training images in batches, which limits the number of images seen per iteration, thus an epoch consists of multiple batch iterations before it completes. The rate at which the model adjusts the weights is known as the learning rate [13], and is used by the optimisation strategy to find the best direction to move the weight values in. Popular optimisation strategies rely on gradient descent, including stochastic gradient descent (SGD) [22], AdaGrad [34], and Nadam [35]. Some of these methods rely on momentum [36], which allows the learning rate to increase (and the search range in the search space) if the direction of change remains consistent between iterations. A popular momentum strategy is Nesterov momentum [36] (employed by Nadam). Another option is to vary the learning rate over the training period, allowing it to decay over time (either linearly, quadratically, or exponentially), as used by AdaGrad [37].

To prevent overfitting to the training data (and hopefully generalise to the problem at hand), there are a few common regularisation strategies that we can employ. By adding dropout [38] during training we ignore a neuron (and its connections) with a set probability p, removing it from consideration during some iterations and preventing fixed activation paths from forming around certain training images (which is overfitting). Another way is to generate pseudo-unique images from our training data, a strategy called data augmentation [21], and allows relatively

Stellenbosch University https://scholar.sun.ac.za

small datasets to be used as input to deep learning models. Typical augmentation types include translation (horizontal and vertical), rotation, scaling, shearing and flipping. Batch normalisation normalises each training batch, and also allows for higher learning rates to be used [39].

One would think the next step in the process is to grab your favourite editor and start writing clear, concise and working code. This might have been the feeling at the start of the thesis, but there are some practicalities to consider.

2.5 Practical considerations

In the early days of deep learning it was difficult to find clear documentation and robust implementations of the algorithms we just discussed, since most of the available resources came from research oriented sources that were still figuring out the best way to implement them. As the popularity of deep learning grew through commercial application, more funding was allocated to both academic and commercial research and development. This, along with the rapid improvement in hardware performance and efforts by Nvidia to support the growing interest in deep learning resulted in stable libraries, powerful hardware, and cloud infrastructure that took advantage of both to offer powerful solutions to academia and industry alike. We take a look at some of the libraries that we used, along with the improvements between the years 2015 and 2017.

2.5.1 Libraries

Through the power of open source communities it is possible to create and support massive libraries that contain all of the core algorithms and processes of deep learning. Without these libraries we would not have the advancements in industry that we see today, since it would just be the large corporations that would have access the resources and manpower to create these systems. Luckily the mindset has shifted to an open source mentality, allowing anyone access to the same tools and resources used by these corporations. There is no need to reinvent the wheel, only to adapt it to the specific needs of your problem.

In 2015 we used two popular deep learning libraries at the time, Theano [40] and Caffe [41]. Theano is not specifically aimed at deep learning, but rather a numerical computation library written in Python by LISA labs at the Université de Montréal. It was designed to evaluate mathematical expressions involving multi-dimensional arrays in an efficient manner [40]. Its wide use within deep learning research has made it synonymous with more specialised deep learning frameworks. Give the more abstract nature of Theano, we used the Python libraries Lasagna [42] and NoLearn [43] to build and train our models. These libraries implemented the core elements (such as backpropagation) and allowed us to easily write the training and evaluation routines we required. Caffe was designed with deep learning in mind, by Berkeley AI Research (BAIR) and written in C++ (along with a Python interface that was used in this thesis). We used Caffe to load pretrained model weights and attempt our transfer learning experiments.

In the years since the improvements in theory, best practises and community support have led to new and more powerful libraries. The most popular of these is TensorFlow by Google. TensorFlow is similar to Theano, in that it is also a numerical computation library designed around nodes and dataflow graphs. "Nodes in the graph represent mathematical operations, while edges represent the multidimensional data arrays (tensors) communicated between them" [44]. TensorFlow is used internally by Google for research as well as systems running in production, and the availability of such a powerful framework as an open source project is truly amazing. Similar to Lasagna and NoLearn, we now have the Keras library [45] that builds on top of both TensorFlow and Theano. Keras is a simple and powerful high-level neural network API written in Python. It allows fast prototyping and abstracts away the technicalities that are not related to the problem one is trying to solve. We used Keras to define, train, and evaluate all our 2017 models.

Stellenbosch University https://scholar.sun.ac.za

2.5.2 Hardware

Other than the availability of data, computational performance has been a major factor that led to the popularity and success of deep learning. With the advances in graphical processor unit (GPU) architecture in recent years we have seen model training times go from weeks down to hours. Naturally this has allowed us to attempt larger challenges that still take weeks to train, such as Google's AlphaGO [46] which was initially trained on 176 GPUs and later on proprietary tensor processing units (TPUs) developed by Google.

In 2015 we trained our models on a desktop computer with the best GPU on the market, the Nvidia Titan Z. This GPU has 12GB of memory and 5,760 CUDA cores. Most of our models trained for about 48 hours. There were few alternatives available, with the Amazon AWS cloud platform being the best option. It was a costly option without much support, and the available GPU instances were not as powerful as our Titan Z GPU.

In 2017 the landscape has changed and the cloud options provide the best value for most research and commercial applications. The GPU instances have grown and the economies of scale have allowed the prices to reasonable and fair levels. We trained all our models using the Amazon p2.xlarge instance, with an Nvidia K80 GPU (24GB RAM, 4,992 CUDA cores). On paper it might seem that the Titan Z is more powerful, but using the cloud allowed us to run nine instances simultaneously, allowing us to do hyperparameter optimisation on a 180 model batch run.

2.5.3 Cloud, platforms, MLaaS

As of writing this thesis there are are numerous cloud based options available to do machine learning and deep learning work on. The most popular options are Amazon AWS, Google Cloud, and Microsoft Azure. While they started by only providing the infrastructure as a service (IaaS), they now provide tools and services that cover the entire machine learning pipeline, offering machine learning as a service (MLaaS). This includes Azure Internet of Things (IoT) by Microsoft, Alexa Machine Learning by Amazon, and Cloud Vision by Google. The cost of using these services is usually determined on a pay-per-use basis, such as the Amazon p2.xlarge instance that costs $0.9 per hour of usage.

A new form of machine learning service that has emerged is one that offers a full commercial pipeline solution from data collection all the way to deployment and versioning. One of the leaders in this market is Valohai [47], a company situated in Turku, Finland. One of their goals is to allow academic research to be easily reproducible and verifiable using their platform. The team at Valohai has given me access to their beta platform, and all of the 2017 experiments were run on this.

Valohai relies on the Amazon GPU infrastructure, but adds value by relying on version control to setup experiments, by automatically generating graphs from training metadata, and automatically caching and storing model training data and results. Some of the features are illustrated in Figure 2.11. An experiment is uploaded to a GIT repository along with a Valohai configuration file that specifies the experiment parameters, such as dataset location and hyperparameters. We can perform hyperparameter optimisation as a task by setting possible values for these hyperparameters, which will create an execution from each combination of hyperparameter options. An active execution will log and graph all output specified by us in the experiment. We can run experiments in parallel, setting how many instances we would like to use for a specific task. We used nine active p2.xlarge instances on all our experiments. All logs and output files are archived in the cloud for easy evaluation. The value that Valohai adds are things that are (or should be) done during research and experimentation. It is just another example of how our development and research workflows can be optimised and automated for more efficient work and record keeping.

Continuing on, we start implementing our solution in Chapter 3 by looking at the infrared images and how they can be used to propose regions of interest for our deep learning classifier in Chapter 4.

Stellenbosch University https://scholar.sun.ac.za

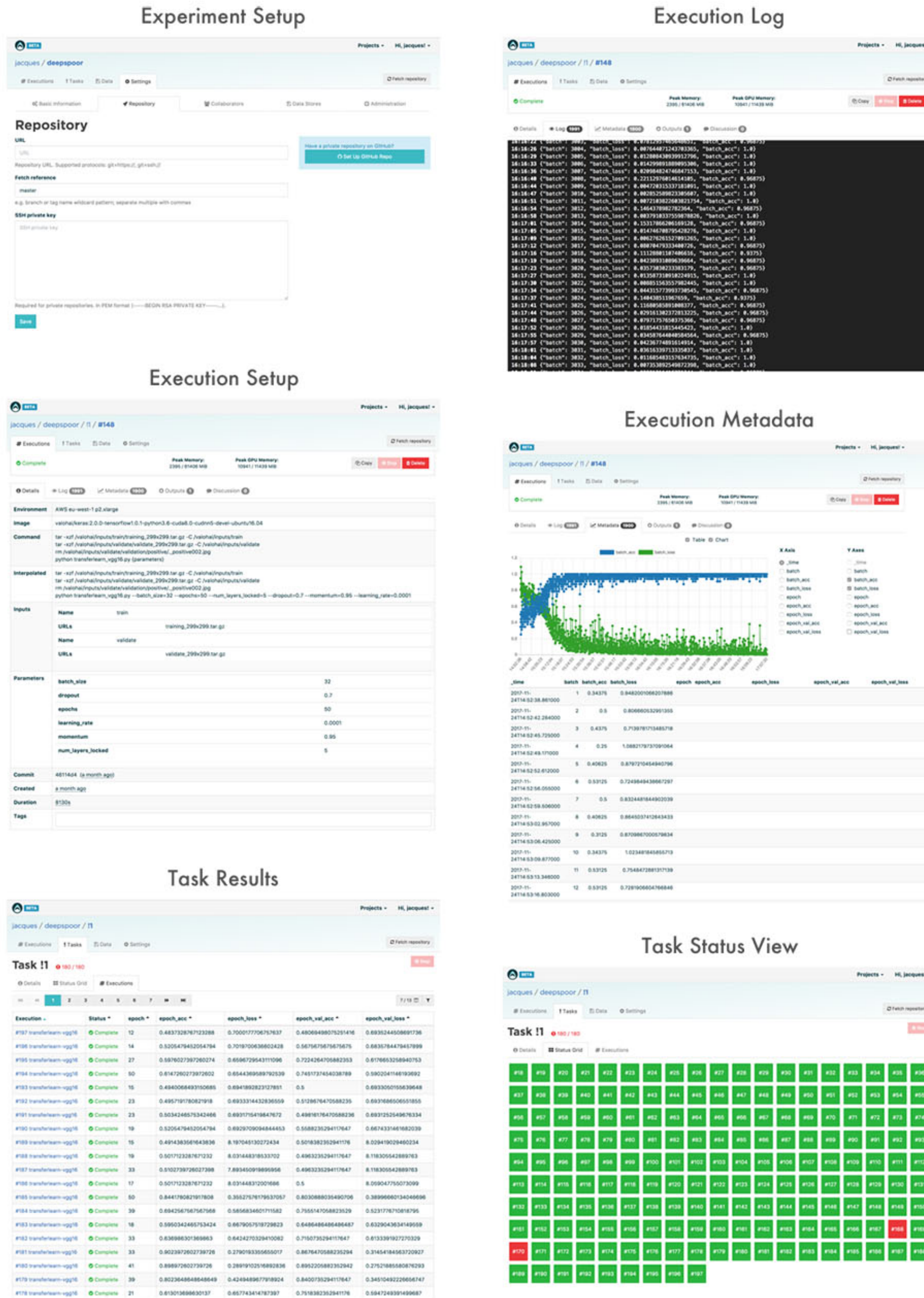


Figure 2.11: Screenshots of the Valohai platform that was used to run the 2017 experiments. Tasks were used to run hyperparameter optimisation batches, with each model execution having its own setup, log, and metadata views.

Stellenbosch University https://scholar.sun.ac.za

3 | Region proposal

"Complexity is your enemy. Any fool can make something complicated. It is hard to keep things simple."

- Richard Branson

The purpose of region proposal is to reduce the number of queries that we need to submit to our classifier in Chapter 4. By ranking areas by likelihood, we can eliminate a lot of work and save our most important resource, time. The classification phase is the most resource intensive, so the more pre-work we can do using cheap and fast methods, the better. Thus this chapter will focus on using traditional image processing techniques on the infrared data to propose regions of interest that will ultimately be classified by our deep learning models.

Our goal is to build a cheap and fast region proposal algorithm that attempts to minimise false negatives, while keeping the number of non-elephant regions detected as elephant regions as low as possible. This means that we do not want to lose any elephants during this phase, even at the cost of a large number of false positive detections.

3.1 Infrared blob detection

Infrared (IR) imagery is used in many domains. Most of the early work focused on military application for offence and defence, while more recent work has focused on more constructive uses such as nature conservation. An example of one of our IR images can be seen in Figure 3.1. A long range infrared sensor was used to capture the image, and each pixel value is actually a real valued temperature measurement. The image was created by scaling all of the temperature values between 0 and 255, with the hottest measurement at 255 and the coldest at 0. All of the images were captured in the early morning, just after dawn when the earth is still cool before the African sun heats it up.

One of the key requirements of IR to be useful is that the target object has an internal temperature that is different from the ambient temperature by at least $4°C$. If this requirement is not met, the sensor will not be able to distinguish between the object and the background around it. This allows for a very small window when the data can be captured, since multiple factors such as the current and previous day's weather could drastically influence the quality and usability of the captured data.

Since our data is captured every four seconds from a moving aircraft, we have a single snapshot to work with, and fall into the single image-based detection group. An example of detecting pedestrians in infrared images [48] allows a stationary camera to capture multiple images over time, for sequential image-based detection methods to be applied. These methods take advantage of the constant background data between sequential images, thus allowing for background removal and ultimately isolating the moving targets in the image. A defence application of this is the sea-based searching, detection and tracking of a moving missile through its heat signature against a static background or cluttered background [49].

Stellenbosch University https://scholar.sun.ac.za

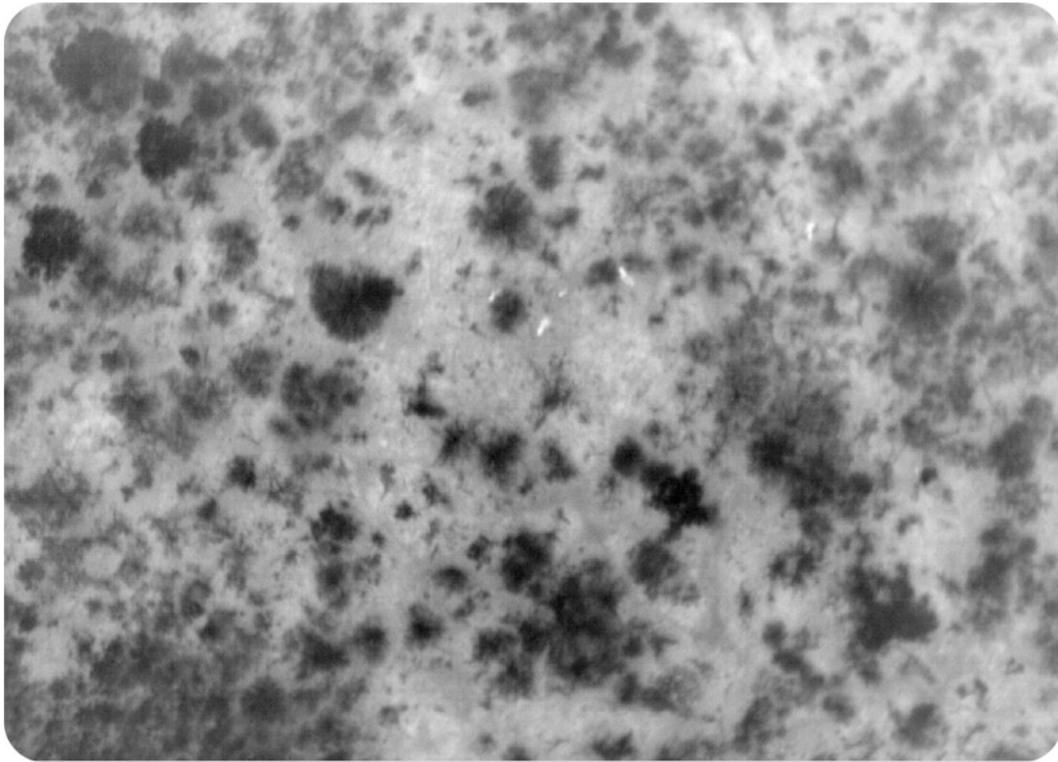


Figure 3.1: Infrared image showing four targets. The colder trees are visible as darker patches, in contrast to the warmer ground which is a lighter grey.

A similarity shared between all of these applications is the need to isolate and detect a dim target against a complex and cluttered background. In search of a solution, we first look at blob detection.

A blob is a mostly homogeneous region in an image that differs in shape and intensity from its surrounding area. Blob detection has traditionally been used to propose regions of interest, but has fallen out of favour with the advancements in computer vision and machine learning.

We are interested in this technique, because its simplicity is exactly what we need for our computationally cheap and fast region proposal algorithm. We will rely on some feature engineering to build our solution, making assumptions about the shape and form of our target. These are reasonable assumptions to make, since we know the general shape of an elephant in the IR images is approximately oval with a varying degree of rotation.

Some of the challenges we face are uneven exposure, orientation changes, large altitude fluctuations, and occlusion. We investigated some of the most popular techniques using the OpenCV [50] Python library. These included the Laplacian of Gaussian (LoG) and Lindeberg's watershed-based grey-level blob detection algorithm [51]. The most successful of these was the top-hat transformation.

Mathematical morphology theory naturally applies to digital image processing, relying on the assumption that an image consists of structures that can be explained by set theory [52]. It is mainly used with binary (black and white) images, but is easily extended to greyscale images. We use these morphological operations to enhance and ultimately detect dim infrared targets embedded in a cluttered and heterogeneous background. We chose the top-hat transform for its ability to increase the contrast between our targets and the background, given the prior knowledge about the target shape and size. As Glasbey [52] explains, "It does this by extracting small or

Stellenbosch University https://scholar.sun.ac.za

narrow, bright or dark features in an image. It is useful when variations in the background mean that this cannot be achieved by a simple threshold".

To understand how the top-hat transformation works, we must first understand the fundamental morphological operation that is erosion. To erode an image, we need to define a structuring element S. We define the shape, size, and the reference point of S, which is typically a basic shape such as a square or ellipse. The reference point is a chosen pixel position that will be used by the different morphological operations. We illustrate these concepts in Figure 3.2, with the original image A in *(a)* showing the foreground as blue and the background as grey. The structuring element S in *(b)* is shown as a simple 3×3 square with a reference point in the centre of the structure. Intuitively we can define erosion as removing (or eroding) away pixels from the edges of our foreground image. We define the erosion of A by S as

$$A \ominus S = \{(i, j) : S_{(i,j)} \subset A\}. \tag{3.1}$$

By placing our structuring element S on A at different positions, we look at the reference position pixel in A. If the reference pixel is foreground, as well as the remaining 8 structuring element pixels, we keep the reference pixel as foreground. On the other hand, if the reference pixel is foreground, but at least one of the remaining 8 structuring element pixels is not, we change the reference pixel to background. If the reference pixel is background, we keep it background. Intuitively this can be compared to image smoothing, except that the background is never changed into foreground. An example of erosion is shown in Figure 3.2 *(c)*.

A complement to erosion is dilation, which has the opposite result on A. We define dilation as

$$A \oplus S = (A^c \ominus S)^c. \tag{3.2}$$

Similar to erosion, we place our structuring element S on A at different positions, again looking at the reference position pixel in A. If the reference pixel is background with at least one of the remaining 8 structuring element pixels as foreground, we turn the reference pixel into foreground. If the reference pixel is background and all the remaining structuring element pixels are background, we keep it unchanged as background. All foreground reference pixels remain as foreground. The intuitive understanding of the dilation effect on A is that the foreground edges are padded and increased. An example of dilation is shown in Figure 3.2 *(d)*.

Using erosion and dilation, we can define two composite operators. We define the opening of A by S as

$$\psi_S(A) = (A \ominus S) \oplus S, \tag{3.3}$$

and the closing of A by S as

$$\phi_S(A) = (A \oplus S) \ominus S. \tag{3.4}$$

This means that opening of A by S is accomplished by first eroding A by S, followed by a dilation of the resulting eroded image by S, as seen in Figure 3.2 *(e)*. Similarly, closing A by S is accomplished by first dilating A by S, followed by an erosion of the resulting dilated image by S, as seen in Figure 3.2 *(f)*.

Stellenbosch University https://scholar.sun.ac.za

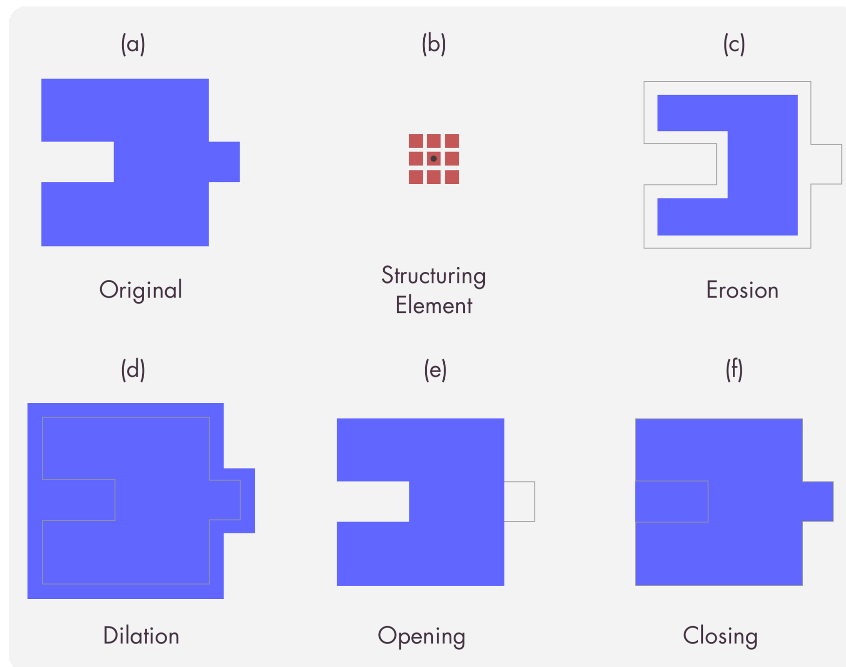


Figure 3.2: Illustration of the morphological operations that build up to the top-hat transform, with the foreground shown in blue and the background in grey. The original image is shown in *(a)*, with the example structuring element shown in *(b)*. Applying the erosion operation on *(a)* results in *(c)*, while applying the dilation operation on *(a)* results in *(d)*. The composite opening and closing operations on *(a)* result in *(e)* and *(f)* respectively. Example from [53].

We can now define the top-hat transform as another composite operation:

$$A - \psi_S(A). \tag{3.5}$$

By subtracting the opened image from the original image, we end up with a peak detector that highlights the brightest areas of our image.

Since we are working with greyscale images, we simply define a binary threshold value that will turn our problem into a binary variant, with everything below the threshold considered background and everything above it considered foreground.

Our region proposal algorithm starts with the top-hat transformation stage. Given our input image A (shown in Figure 3.3), we use an elliptical structuring element of size 6×6 with a reference point in the centre, and apply the transformation.

The resulting image is shown in Figure 3.4. For the erosion and dilation operations, we used a threshold value of 200. When a pixel is dilated or eroded, the average values of the remaining structuring element pixels was used as the new value. This ensured that we increased the contrast between the foreground and the background, without losing intensity information from the true targets as would be the case with a binary operation.

After the top-hat transformation, we take the resulting image B and apply a Gaussian blur with a kernel size of 3×3 to smooth out the image. We then apply histogram equalisation to further enhance the contrast of the image. Lastly we threshold and binarise the image, and the result C for our example is shown in Figure 3.5.

Stellenbosch University https://scholar.sun.ac.za

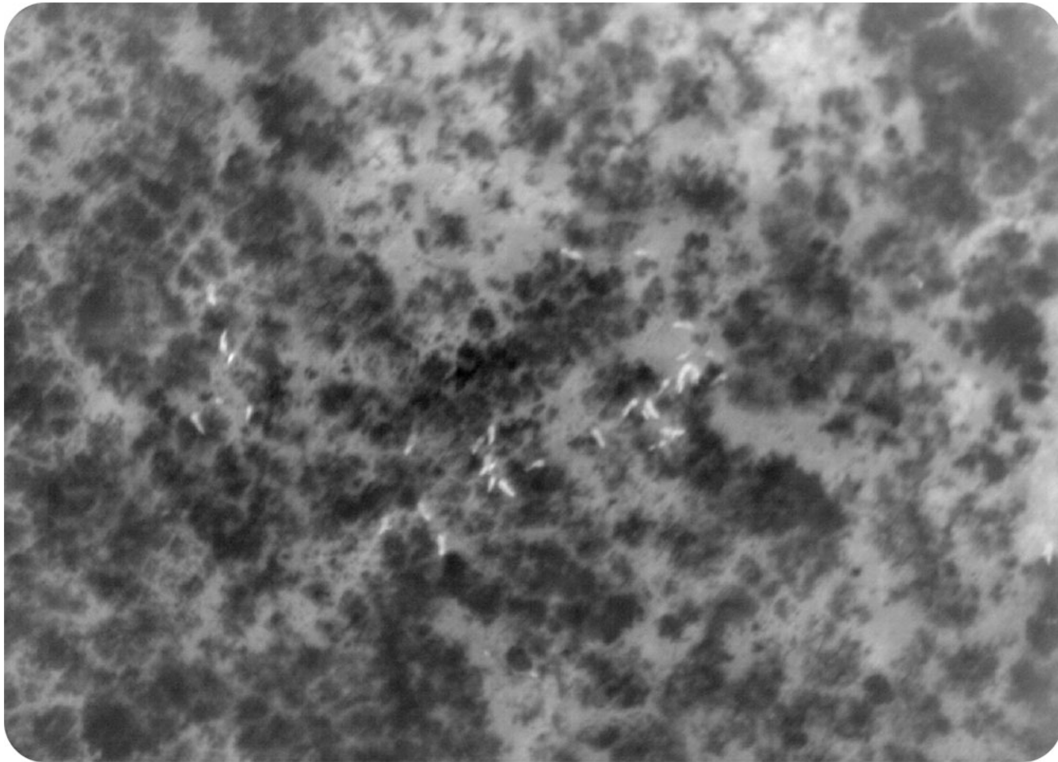


Figure 3.3: An original infrared image showing multiple elephant targets across the image.

Detected regions	Detected elephants	Accuracy	Overdetection Rate
33,608	986	98.80	8202.19%
9,215	961	97.58	2507.71%
5,812	919	93.26	1414.65%

Table 3.1: Performance of three versions of the region proposal algorithm on the 121 images that contain at least one elephant. By experimenting with different hyperparameter values we see how quickly the overdetection rate grows with only a small improvement in detection accuracy.

We detect the contours of the remaining blobs and apply a merging algorithm to join smaller blobs with larger ones. We used the prior knowledge of the elephant size and shape to create realistic blobs, shown in Figure 3.6.

The detected contour shapes can be mapped back to the original image, as seen in Figure 3.7. We considered using the contour shape information in the classification phase, but instead decided to experiment with pure colour image-based classification first. This has potential for future research.

We strived to obtain at least 98% true positive accuracy, and to minimise the number of false positives. We experimented with different threshold values, structuring element shapes and sizes, and the various other image processing hyperparameters to achieve this goal. There is a correlation between the true positive accuracy obtained and the overdetection rate, which can be seen in Table 3.1. To obtain our 98% accuracy we have to lower the threshold value slightly, resulting in a large number of false positives. If we accept 97% accuracy, we reduce the number of false positives by over two thirds. To reduce the overdetection rate further we see a larger drop in accuracy.

Stellenbosch University https://scholar.sun.ac.za

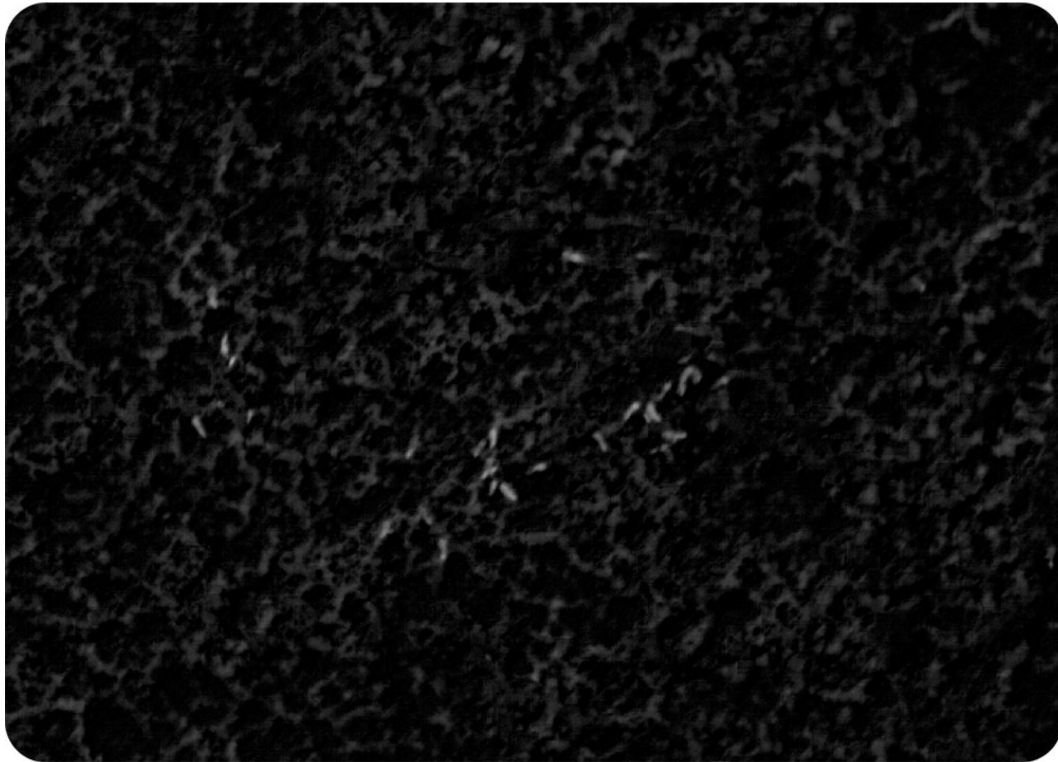


Figure 3.4: The resulting image after the top-hat transform was applied to the original infrared image in Figure 3.3.

The final phase of the region proposal algorithm is to determine the coordinates of the detected targets in the corresponding colour image. Using the contour shapes, we calculate the minimal rectangle that will contain the entire contour shape, and use the centre of the rectangle as our detected IR coordinates.

3.2 Image registration

We have the coordinates on the infrared image, but how do we get the corresponding coordinates on the RGB image? Image registration allows us to transform one set of coordinates into another, allowing us to map from the smaller IR image onto the larger RGB image.

During experimentation we noticed that there were alignment issues between corresponding pairs of IR and RGB images. We concluded that the capture rig did not dampen the aircraft vibration enough, which resulted in varying and unpredictable alignment errors throughout the dataset.

We investigated using a single dataset homography to map between the IR and RGB images, and also experimented with different ideas to solve the alignment problem.

Given our IR and RGB images, we have two different representations of the same physical space. The differences between the representation are in the capture spectrum (infrared and visible light), the image size, and of course the extrinsic parameters (location and orientation) of the sensor. Both of the images contain the same physical elements, thus have corresponding points. A homography in computer vision is a matrix that captures the relationship between two planes. Once we calculate the correct homography for a given pair, we can perform plane-to-plane transformations, allowing us to map coordinates from one plane to another.

Stellenbosch University https://scholar.sun.ac.za

Figure 3.5: The resulting image after the top-hat transform result from Figure 3.4 was smoothed, equalised and binarised.

We calculate a homography between two images using at least four point correspondences between the two images to solve a system of equations that will give us our homography [54]. Point correspondences are usually collected using a feature detection algorithm such as SURF [55]. Since we deal with two different spectra, we manually selected the corresponding points between the IR and RGB images used to calculate our homography. We used 21 points across multiple images in an attempt to reduce the effect of the misalignment added during the capture phase, and used the OpenCV library to perform our calculations. The best performing homography (called the dataset homography) can be seen in Figure 3.8.

The dataset homography performed well, but there were still inaccuracies in the projection that could not be removed, even after multiple attempts and approaches. We refer to using just the dataset homography as the unadjusted homography method.

We came up with two solutions to the unadjusted homography alignment issues. The first was to manually determine an additional translational component to compensate for x- and y-offsets in each image, resulting in a list with each image number and its corresponding offset values. This is called the manual homography adjustment method. The second was to manually map the positive ground truth coordinates by hand onto the RGB image. This is called the manual method. Results from all three methods can be seen in Figure 3.9.

We successfully implemented a cheap and fast region proposal algorithm that drastically reduces the number of regions that need to be classified. We investigated various ways to map the detected regions onto our RGB images given the alignment issues introduced during the capture phase. We defined three methods to map the infrared coordinates onto the corresponding RGB images. These will be used in Chapter 4 to train and evaluate our classification models. We are now ready to move onto the final phase of the project and implement our deep learning solution

Stellenbosch University https://scholar.sun.ac.za

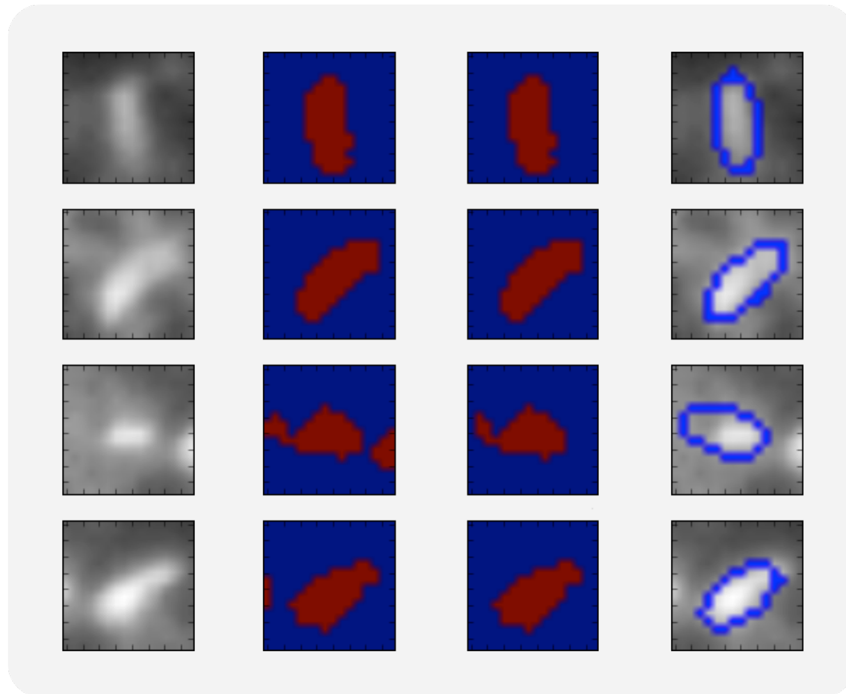


Figure 3.6: Contour detection process showing the original infrared image patch of an elephant, the blob after our top-hat process, the blob after our merging operation, and finally the best fitting ellipse on the merged blob mapped onto the original infrared image patch.

to classify every proposed region in the RGB image, as containing an elephant or not.

Stellenbosch University https://scholar.sun.ac.za

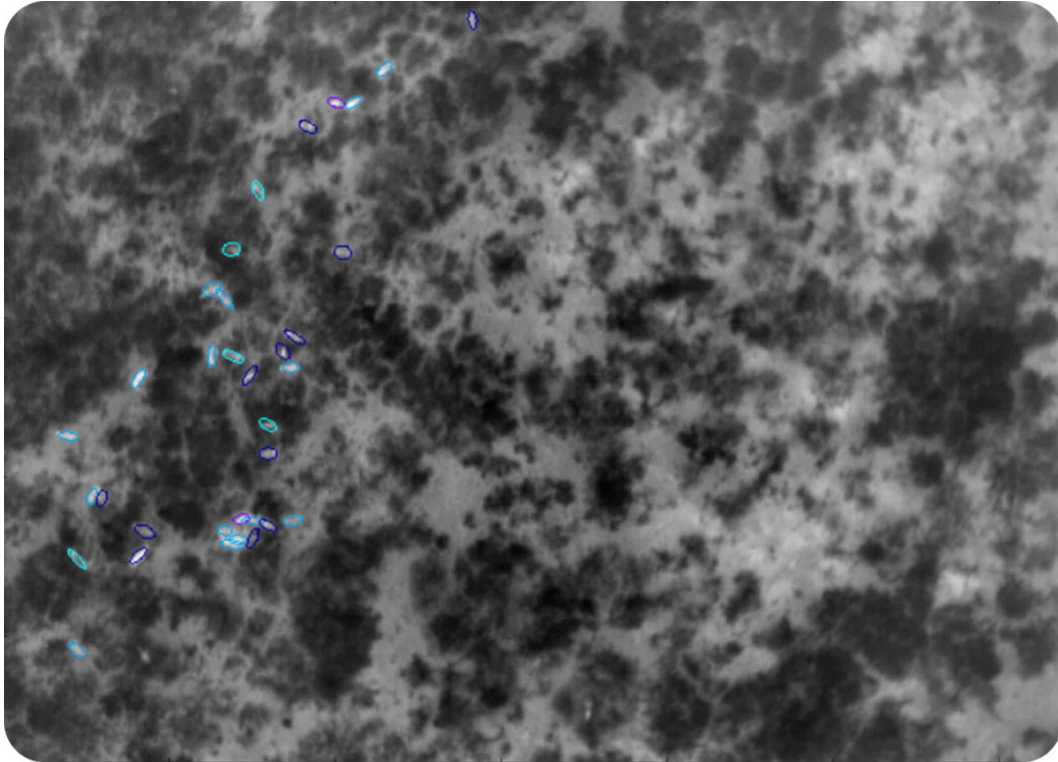


Figure 3.7: Best fitting ellipses from the detected and merged blobs, drawn on the original infrared image using the class labels for the colours: turquoise for visible elephants, dark blue for partially visible elephants, purple for calf, and aqua for not visible.

```
homography = np.array(  [[0.006652157582043, -0.000304488417064, 0.937449122580456],
                        [0.000068349461114, 0.006411369859734, 0.347997848951433],
                        [0.000000041108610, -0.000000093134481, 0.001088783486020]])
```

Figure 3.8: Dataset homography calculated using 21 point correspondence between corresponding IR and RGB image pairs.

Stellenbosch University https://scholar.sun.ac.za

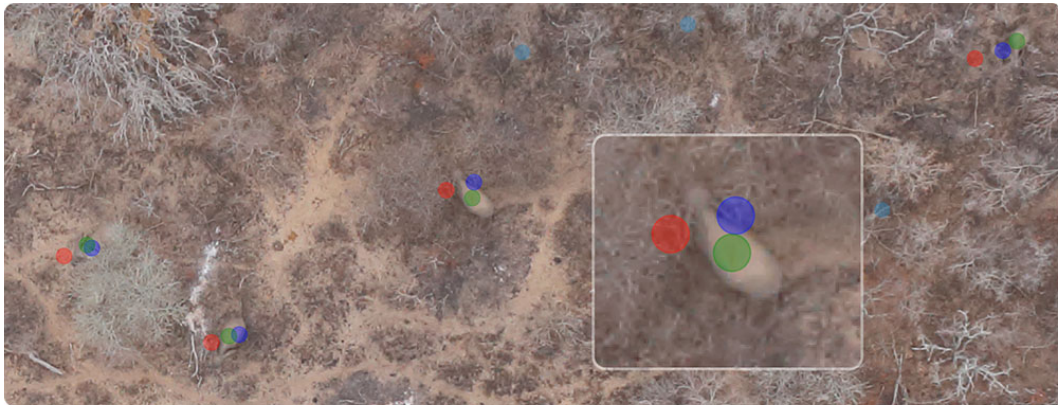


Figure 3.9: Image registration example showing IR detections mapped to the corresponding RGB image, using our three different methods. The red markers are from the unadjusted homography method, the blue markers are from the manual adjustment homography method, and the green markers are from the manual method.

Stellenbosch University https://scholar.sun.ac.za

4 | Implementation and results

"Essentially, all models are wrong, but some are useful."

- George E. P. Box, *Empirical Model-Building and Response Surfaces, p. 424*

After an overview of the theory in Chapter 2, and the groundwork in Chapter 3, we finally arrive at the core of the project. The implementation of the classifier will determine the validity of our hypothesis.

We continue from Chapter 3 by using the proposed RGB region coordinates to build our model training and evaluation datasets. By the end of this chapter we will have trained various models, evaluated their performance on sample images, as well as on the entire image dataset. We will be able to conclude the feasibility of our system and be able to say which factors are most important to building a successful pipeline.

The original implementation and evaluation was done at the end of 2015, the performance was evaluated and a conclusion was drawn. With all the improvements over the last two years, the experiments were re-run at the end of 2017, using new libraries and infrastructure. We will present the results from both years as a way to highlight and reflect on the changes in workflow and performance. Before we begin training our models, we need to discuss how the experiments were constructed and evaluated.

4.1 Methodology, evaluation techniques and metrics

The focus of our training and evaluation methodology is to keep the system as unbiased as possible, given the uncontrolled bias that was already introduced into the data during the capture phase. By keeping the training and evaluation dataset design strict, we ensure that the models are comparable, that the experiments are repeatable, and that the results remain verifiable.

First we will discuss how the different datasets were generated in Section 4.2. The 2015 and 2017 sets differ in a few ways, each with their own motivations. Arguably the most important set is the training set (along with the validation set), since all the models will be trained on it. Once we have our training and validation sets created, we move onto the different model creation strategies used. We start by training a few models from scratch in Section 4.3, then we take existing models in Section 4.4 and attempt to transfer the knowledge onto our own problem. Once we have our models trained and ranked, we take the strongest candidates from each strategy and start evaluating their performance in Section 4.5. We start with a single image and ultimately evaluate the strongest candidates on the entire image dataset. All of the datasets and evaluation methods rely on the regions proposed and mapped from Chapter 3.

The training methodology differs drastically between the first experiments in 2015 and the latest ones in 2017. All of the 2015 experiments were run on a local machine with an Intel i5-3570K CPU, 16GB of RAM and a 12GB Nvidia GeForce GTX Titan Z reference card. The machine was running Windows 7 and the majority of the code was written in Python using Theano and Caffe. Supporting libraries such as Lasagna and NoLearn were used to implement the model

Stellenbosch University https://scholar.sun.ac.za

architecture, execute training and to do model evaluation. Back then the libraries and GPU optimisation were still in their infancy, leading to very long training times. A single model typically trained for just over a day, at about 30 hours. This was also the time when South Africa was experiencing loadshedding (scheduled rolling power outages per region), which meant that a single model could take up to a week to train during the worst loadshedding peaks. This meant that hyperparameter optimisation was not much of an option, and the models had to be trained on estimates and best practices found in the literature and online.

The world changed a lot between 2015 and 2017, with deep learning becoming a buzz word that is heard everywhere. With the increase in popularity, the funding allocated to open research and development by private companies such as Google and Baidu allowed the industry to flourish. This gave birth to TensorFlow from Google, and its support by the deep learning framework Keras. This combination trivialised everything from data cleaning, augmentation, training and evaluation. Our 2017 models were trained using TensorFlow and Keras, through the Valohai platform (Section 2.5.3) running concurrently on nine Amazon AWS p2.xlarge instances, with a 24GB Nvidia K80 GPU and 61GB RAM each. This meant that a hyperparameter optimisation task with 180 models took just over two days to complete, at a cost of roughly $150.

While training the models we keep track of a few metrics, the most important being the training accuracy, training loss, validation accuracy, and validation loss. The training and validation accuracy is the number of correct (negative and positive) predictions per epoch. The validation accuracy is used to rank the models after training, while the validation loss is used to determine if the model has converged and allows us to stop training early (and avoid overfitting).

Once we have our models trained, we evaluate their performance on various sets, ranging from a single image to the entire image dataset (which includes all 890 images captured). To accurately evaluate and compare the model performance, we run all the images in the evaluation set through a model and get the prediction values (2D vector with real confidence values for each class) for each region of interest image. We record the true positive, true negative, false positive, and false negative count for each model and evaluation set pair. From these we can calculate the accuracy, true negative rate, precision rate, recall rate, F_1 score, and Matthews correlation coefficient. Here follows a brief overview of each of the mentioned metrics.

We are interested in when a model correctly classifies a cropped image region, hereafter referred to as an image. Since we are dealing with a binary classification problem we have a positive image when the image contains at least one visible (or partially visible) elephant, and we have a negative image when the image does not. From this we define the true positive (T_p) rate as the fraction of positive images that are correctly classified as positive, true negative (T_n) rate as the fraction of negative images correctly classified as negative, false positive (F_p) rate as the fraction of negative images incorrectly classified as positive, and false negative (F_n) rate as the the fraction of positive images incorrectly classified as negative. From our hypothesis, we would like to reduce the number of false negative classifications (to ensure we do not lose any elephants), while keeping the false positive rate relatively low and manageable for a human to verify all the results. We can combine these numbers to get a more succinct perspective of a model's performance.

The first of these metrics is accuracy, which gives us a percentage of correctly classified images. We do not distinguish between positive and negative classification accuracy, which means this is not a good measure for unbalanced datasets (when there are different numbers of images per class), as is our case. We still use accuracy during training where we have a balanced dataset of positive and negative images. We define accuracy as

$$\frac{T_p + T_n}{T_p + T_n + F_p + F_n}. \tag{4.1}$$

Taking the unbalanced nature of our problem into account, we look to the Matthews corre-

Stellenbosch University https://scholar.sun.ac.za

lation coefficient (MCC). The MCC is a balanced measure for binary classification models, and takes the number of elements in each class into account, producing a real valued coefficient score between -1 and 1. An MCC of $+1$ represents a perfect prediction rate and is the ideal we strive for. An MCC of 0 means no correlation and practically random prediction, while an MCC of -1 represents an inverse relation between prediction and observation (which is still valuable information). We define the MCC as

$$\frac{T_p T_n - F_p F_n}{\sqrt{(T_p + F_p)(T_p + F_n)(T_n + F_p)(T_n + F_n)}}. \tag{4.2}$$

Looking specifically at the negative predictions, we have specificity, also known as the true negative rate (TNR), which is the percentage of negative images correctly classified as negative:

$$\frac{T_n}{T_n + F_p}. \tag{4.3}$$

Similarly, we can look at the positive predictions, then we have the sensitivity, also known as the true positive rate, or recall. This is the percentage of positive images correctly classified as positive:

$$\frac{T_p}{T_p + F_n}. \tag{4.4}$$

On a broader view, we can calculate the precision of a model, which is the likelihood that an image is classified as positive. This means that a high number of overdetections (a high false positive rate) will result in a lower precision score. We define it as

$$\frac{T_p}{T_p + F_p}. \tag{4.5}$$

By combining the precision and recall scores, we get the F_1 score. This is a balanced measure of both precision and recall, and acts as a good indicator for a model's ability to correctly classify images given large differences in the class counts. We define F_1 as

$$2 \times \frac{precision \times recall}{precision + recall}. \tag{4.6}$$

Now that we have our evaluation strategy ready and a way to measure and compare model performance, we can start generating our training data.

Stellenbosch University https://scholar.sun.ac.za

4.2 Dataset creation

Using the IR images as input to the region proposal algorithm from Chapter 3, we have the proposed regions of interest mapped onto our RGB images. This is the starting point for us to create our datasets, for both 2015 and 2017. There are a few key differences in the datasets for each year, namely the crop size, number of images, augmentation strategy, and the image coordinates used to generate them. We will look at each year in turn and motivate the choices made.

Starting with 2015, we have the first major difference, the image crop size. All dataset images are either $168 \times 168 \times 3$ (inspired by best practices and appropriateness to the problem), and $224 \times 224 \times 3$ (inspired by AlexNet). This resulted in two similar datasets, at two different crop sizes. The sets contain 5,250 unique negative images, randomly sampled from the proposed regions that do not contain elephants, mapped from IR using the manually adjusted homography method. The base of the positive images is from the 875 positive crops (visible, partially visible, and calf) from the ground truth set. The RGB coordinates were created using the manually adjusted homography method. These 875 positive images were augmented five times, resulting in 5,250 positive images (including the originals). The choice to only augment five times was based on memory usage and training time considerations. To augment the data we used a maximum translation range of 32px (20% of 168px) vertically and horizontally, a maximum rotation range of 360 degrees, a scaling range of -20% to 20%, and a Bernoulli trial to horizontally or vertically flip the image. The augmentations were made before the images were cropped, with the intention to introduce new information into the system through additional background data, and to ensure the two datasets were as similar as possible. Once the 5,250 positive and 5,250 negative images were assembled, we split them into training and validation sets. We allocated 80% of the images to the training set (4,200 positive, 4,200 negative), and 20% of the images to the validation set (1,050 positive, 1,050 negative). We ensured that an original positive image and all its augmentation were in the same set to reduce bias. All of the augmentation code was written from scratch and performed using standard image processing libraries in Python. Examples of the augmented positive data can be seen in Figure 4.1, with the original marked in blue followed by its augmentations.

Arriving at 2017, we have a single set of images at a crop size of $299 \times 299 \times 3$ (inspired by Google Inception). These images were scaled to a different resolution if required by a specific model. There are 875 unique negative images, randomly sampled from the proposed negative regions, mapped from IR using the unadjusted homography method. There are 875 positive ground truth crops (visible, partially visible, and calf) from the ground truth set, created using the manual method. We split 70% of these into the training set (600 positive, 600 negative), and the remaining 30% into the validation set (275 positive, 275 negative). In contrast to the 2015 method of generating the entire augmented set before training, in 2017 we use real-time augmentation per epoch, ensuring that the model never sees the same image twice during the training process. This also ensures that the model never sees the original image either, allowing for more accurate model performance evaluation. Augmentation was performed on both the positive and negative training images with an augmentation ratio of $21:1$, resulting in 12,600 positive and 12,600 negative images every epoch. To augment the data we used a maximum translation range of 60px (20% of 299px) vertically and horizontally, a maximum rotation range of 40 degrees, a scaling range of -20% to 20%, a maximum shear range of 20%, and a Bernoulli trial to horizontally or vertically flip the image. Since the images were augmented in real-time from pre-cropped images, we used a nearest neighbour padding technique to fill in the missing information. The validation images were not augmented. An example of the augmentation can be seen in Figure 4.2, with the original image in blue followed by its augmentations.

In 2015 a simple experiment was run to motivate the use of augmentation, and the results can be seen in Table 4.1. In each case we took 80% of the available data to train a model and reserve 20% for testing (holdout set). The table shows how each model performed on the holdout

Stellenbosch University https://scholar.sun.ac.za

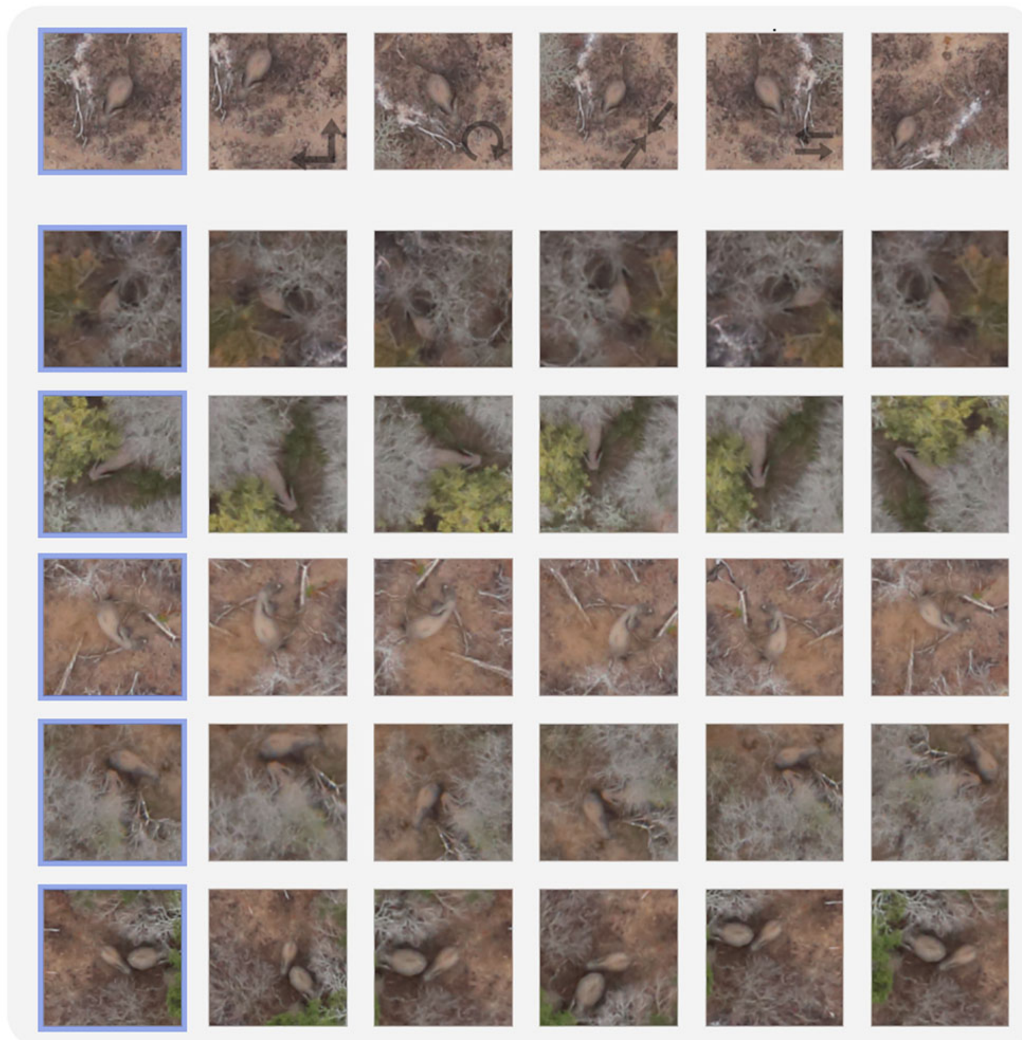


Figure 4.1: We show an example of 2015 augmentation results. In the first row the individual augmentation operations are visualised, we also show the result when the different operations are combined in the final column. The remaining rows illustrate how the original crops (seen in the first column, highlighted by blue) can be augmented to generate more data (as seen in the last five columns).

Stellenbosch University https://scholar.sun.ac.za

Figure 4.2: We show an example of 2017 augmentation results. The original positive crop is highlighted in blue, with 19 example augmentations following it.

set, and also how it performed on the original unaugmented set of ground truth positive samples (grouped into three types of elephant images). We see that applying no augmentation (875 positive and 875 negative images), resulted in decent performance, but at the price of overfitting to the data. Adding various forms of augmentation increased the ability of a model to generalise, resulting in higher recall and classification accuracy on the ground truth set. After applying all permutations of translation, rotation, scaling and flipping, it is interesting to note that translation had the greatest impact on performance. We believe this allows the convolutional filters to generalise better. It is also plausible that rotation and scale are represented in the existing data, and that rotation and scale invariance would be present in the model even without augmentation.

Even though the core set from 2017 is a lot smaller than the one from 2015, the number and quality of augmentations generated over the entire training phase makes it more likely to yield a more generalised model. To verify this, we need to first train our models.

4.3 Training from scratch

The simplest place to start training a model is to configure the desired architecture and initialise the model with random weights. We used four different architectures in 2015, and three different ones in 2017 following exactly this approach. The goal is to get these randomly chosen weights to the point where the model specialises to our classification task.

In 2015 we looked at four model architectures, some chosen for their simplicity and others

Stellenbosch University https://scholar.sun.ac.za

	Holdout set				Ground truth set			
	Precision	Recall	F_1	Accuracy	Recall	Visible	Calf	Partial
No augmentation	0.9193	0.8457	0.8810	0.8857	0.7303	0.7457	0.6316	0.7161
Translation	0.7220	0.9200	0.8090	0.7829	0.8800	0.8975	0.9474	0.8419
Transl., rotation	0.6452	0.9143	0.7565	0.7057	0.8914	0.9089	0.9211	0.8581
Transl., rotation, scaling	0.6864	0.9257	0.7883	0.7514	0.8846	0.8956	0.9211	0.8613

Table 4.1: We show that increasing the dataset size with augmentation allows us to generate better performing models. All four configurations were trained on their respective training sets. The first column shows the model performance on their respective holdout sets (20%). The improvement in recall shows the models with augmentation are generalising more, compared to the model without augmentation that is overfitting to the training data. We also generated a set containing only ground truth positive samples (excluding hidden) and classified it using the different models. The result of this is shown in the second column. Again we see improved performance when augmentation is used.

inspired by related work. All the models trained from scratch used the $168 \times 168 \times 3$ dataset, a learning rate of 0.0001, and momentum of 0.5 (using Nesterov momentum [36]). The first model we trained was as simple it gets, and we called it the simple neural network model (Simple NN). Our second model (Simple CNN) improved on our first model by adding a convolution and max pooling layer before the fully connected layer, we also added 50% dropout to the fully connected layer. Our third model was inspired by a similar work detecting dugong in the ocean [8]. The use of a stacked convolutional layer was worth exploring, resulting in our Stacked CNN. Our fourth and final model is the Deeper CNN, which consists of three layers of convolution and max pooling layers, followed by two dense layers with a dropout rate of 50%. All of these models are visualised in Figure 4.3.

All of the models were trained over 500 epochs with a batch size of 32 images per iteration, and all images were normalised (mean subtraction and divided by the standard deviation). PCA, whitening and scaling were also used on some of the earlier experiments, with normalisation giving the most consistent results. Figure 4.4 shows the validation accuracy (red) for each model in the first four graphs, with the training (red) and validation (blue) loss in the last four graphs. We can see that the validation accuracy for the convolution based networks grows over time, while the simple neural network converges very fast. This shows that the simple neural network does not have the ability to capture the core information that describes an elephant, an expected result. Looking at the loss graphs, we see that the training loss decreases for all the models, while the validation loss marginally decreases over time for the convolution based networks. This is a strong indication that the models are overfitting to the training data and will not generalise well. We suspect the small number of images available is the cause and that it will not be enough to train a robust model.

The models will be evaluated in Section 4.5, but some preliminary evaluations were performed on the validation set in 2015, shown in Table 4.2. The results show that the Simple CNN performed the best. Due to the limited number of training samples, the more complex models were not able to train strong enough weights, while the simpler network performed slightly better.

By combining the different models into ensembles we exploit the fact that different architectures and training runs make different classification mistakes. We create a set of ensemble models using a simple threshold system. We classify the test set using each of the four models mentioned above and include a fifth model that is similar to the Stacked CNN except for its hyperparameters (training of 1000 epochs occurred using a more aggressive learning rate). We take the classification results and aggregate them for each image into a vote set, effectively giving each model an equal vote on every image. We then threshold the vote set for every integer in the range $[1, 5]$ and calculate the performance metrics for each of these threshold models. The results of this can be

Stellenbosch University https://scholar.sun.ac.za

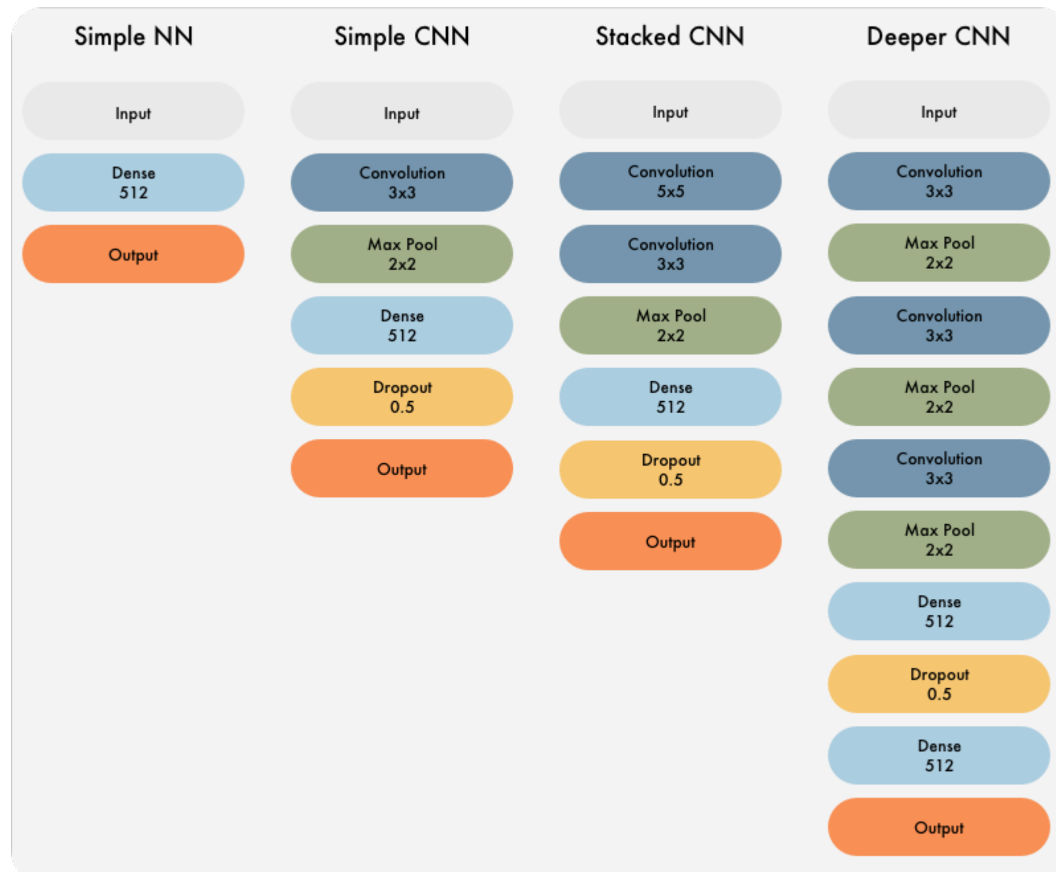


Figure 4.3: We visualise the different architectures used to train our models from scratch in 2015. We visualise the depth of the networks as well as some of the layer parameters.

Model	T_n	F_p	F_n	T_p	Precision	Recall	F_1	MCC
Simple NN	907	143	246	804	0.8490	0.7657	0.8052	0.6326
Simple CNN	946	104	185	865	0.8927	0.8238	0.8569	0.7269
Stacked CNN	948	102	225	825	0.8900	0.7857	0.8346	0.6933
Deeper CNN	930	120	233	817	0.8719	0.7781	0.8223	0.6677

Table 4.2: We show the raw classification results for each 2015 model, along with the calculated performance metrics. We see that the simple neural network performs the worst. The simple convolutional neural network outperforms all the other models, showing the limitations imposed by our limited dataset. We do not have enough unique data to train a more complex architecture without it overfitting.

Stellenbosch University https://scholar.sun.ac.za

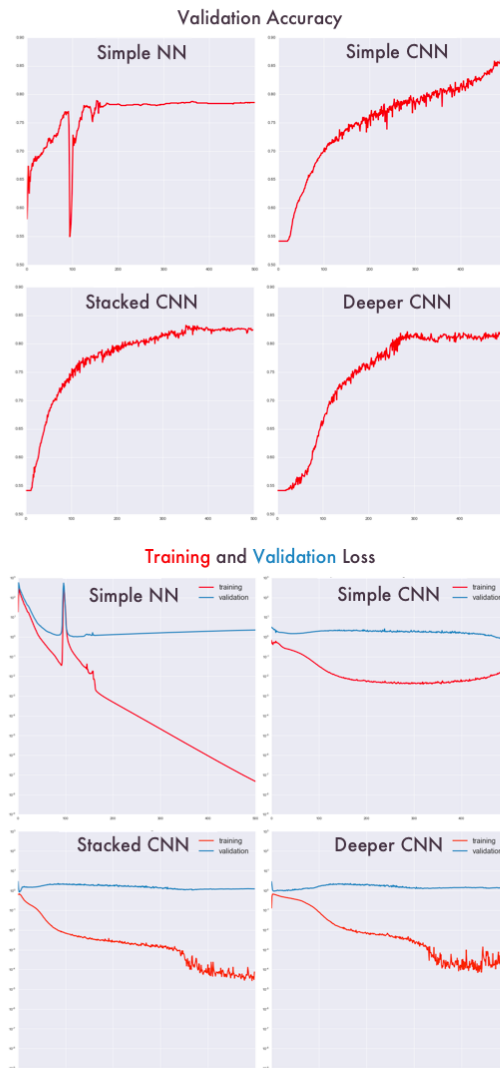


Figure 4.4: Here we see the validation accuracy, and training and validation loss graphs for all the 2015 models we trained, at the same scale. The simple neural network shows the validation loss diverging while the training loss decreases. This indicates that the model is overfitting to the training data at the cost of validation performance. Both the stacked convolutional neural network and the deeper convolutional neural network reach a constant validation loss while the training loss fluctuates wildly as training continues.

Stellenbosch University https://scholar.sun.ac.za

Ensemble	T_n	F_p	F_n	T_p	Precision	Recall	F_1	MCC
Threshold 1	773	277	88	962	0.7764	0.9162	0.8405	0.6632
Threshold 2	905	145	160	890	0.8599	0.8476	0.8537	0.7096
Threshold 3	960	90	216	834	0.9026	0.7943	0.8450	0.7137
Threshold 4	995	55	274	776	0.9338	0.7390	0.8251	0.7021
Threshold 5	1025	25	378	672	0.9641	0.6400	0.7693	0.6543

Table 4.3: We show the ensemble performance at different threshold levels. By increasing the threshold, we increase the confidence the ensemble has when it classifies an image. A high threshold indicates agreement between the models. Increasing the threshold results in an increase in precision and a decrease in recall.

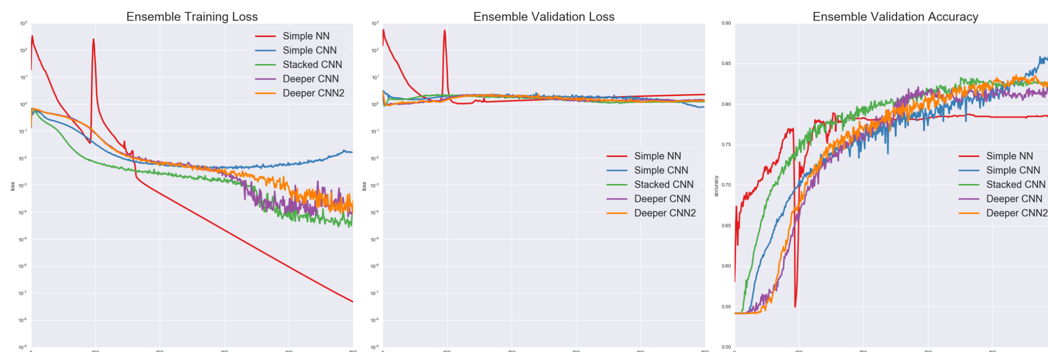


Figure 4.5: Here we show the training loss, validation loss and validation accuracy of all the models included in the ensemble. We see that each model follows a different path, in effect making different mistakes when classifying an image.

seen in Table 4.3. As we increase the threshold, we increase the precision and decrease the recall. This makes sense since we are increasing the certainty that an image is positive, thus reducing the number of false positives, while also creating a stricter requirement to be classified as positive, thus reducing the number of true positives. On average, when compared to the separate models, the ensembles do not perform better than the simple convolutional neural network. However when keeping in mind that our focus is primarily on recall and secondly on precision, we see that an ensemble can offer notable improvement.

By combining the training loss, validation loss and validation accuracy of the individual models in Figure 4.5, we can imagine how the different paths taken by the individual models each contribute to the generalisation of the ensemble and create a more robust prediction.

In 2017 we took a simple approach to training comparable models to 2015. We trained three convolutional neural networks, the first with a single convolution and max pooling layer, the second with two, and the third with three. The model architectures are visualised in Figure 4.6.

With the all the improvements between 2015 and 2017, we could be a bit more adventurous when training the models. We did hyperparameter optimisation on each of the models, with dropout options of $[0.3, 0.5, 0.9]$, learning rates at $[0.01, 0.001, 0.0001]$, and momentum at $[0.5, 0.9, 0.95, 0.99]$. All the images were scaled and augmented in real-time during training. This resulted in 36 different trained models. We also trained the models with 500 epochs, as well as another set with early stopping (converging validation accuracy) enabled. The end result was 216 different models that allowed us to get a better understanding of how training from scratch on our dataset worked. All of the model training results are available in the Appendix. We visualise the accuracy and loss at each epoch during training for the top performing models in Figure 4.7.

Looking at the results in Table 4.4, we see that the more complex network performed the best

Stellenbosch University https://scholar.sun.ac.za

Figure 4.6: We visualise the different architectures used to train our models from scratch in 2017. Each models is a convolutional neural network, with increasing depth. The dropout options are used during hyperparameter optimisation.

Model	Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss
Scratch 2	486	500	0.7089	0.5479	0.7261	0.5398
Scratch 3	590	500	0.9495	0.1419	0.9633	0.1037
Scratch 4	560	500	0.9632	0.1122	0.9846	0.0803

Table 4.4: Results from the best performing 2017 models for each architecture, trained for 500 epochs each. Scratch 4 performs the best across the board.

across all metrics. We will evaluate the performance of the models in Section 4.5 using these three models selected from the 216 models trained. It was observed that enabling early stopping during training resulted in models with lower accuracy. This could indicate that our 500 epoch models have overfit to the data and should have been stopped earlier. We will evaluate both in Section 4.5 to check this.

Comparing 2015 with 2017, we note the differences visible in the training graphs. While in 2015 the models struggled to learn weights that would generalise to the problem, we do not have the same problem in 2017. This can almost certainly be attributed to the difference in data augmentation and availability of unique (or pseudo-unique) data, along with the ability to do hyperparameter optimisation. The results also show that the more complex model is capable of performing better, given enough data to train all its parameters. The rate at which we could train

Stellenbosch University https://scholar.sun.ac.za

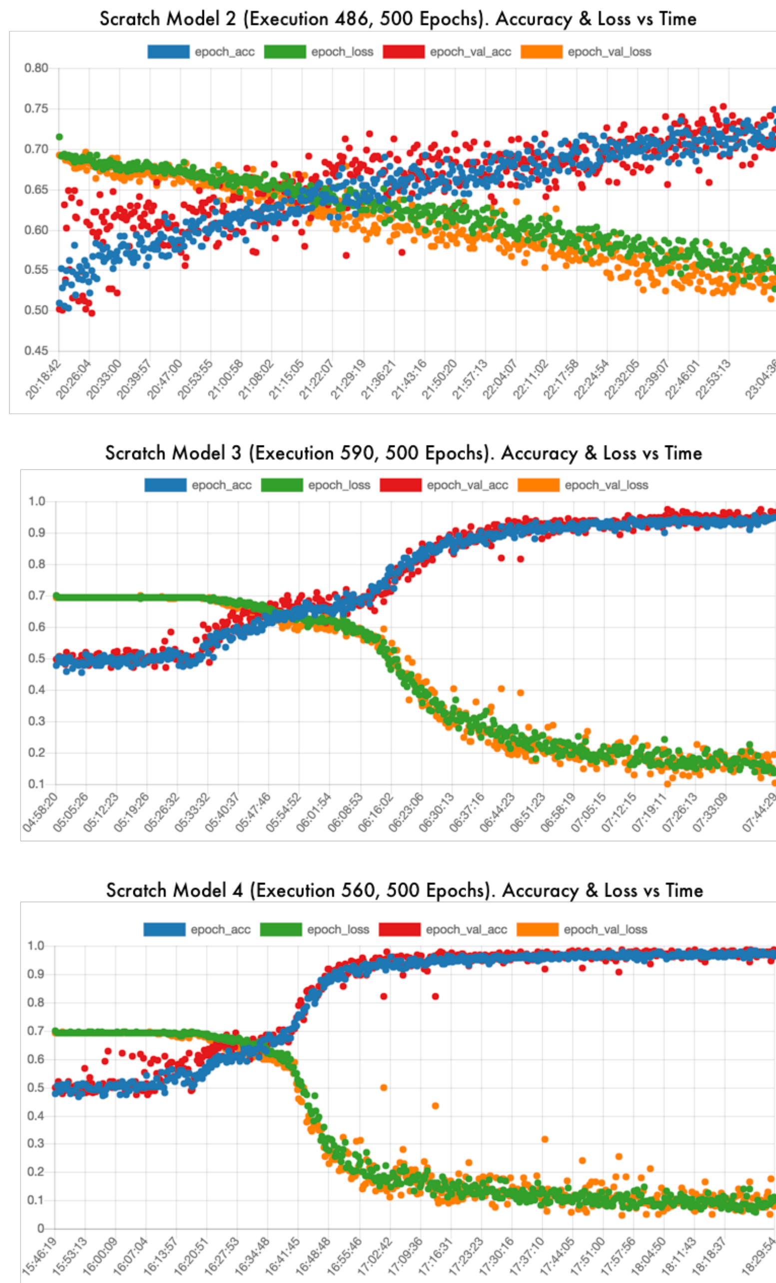


Figure 4.7: Visualising the accuracy and loss (training and validation) of the best performing models at each training epoch. Accuracy and loss cross at roughly the same time period for all the models, with Model 4 rapidly reaching a high accuracy and low loss afterwards. Model 2 is slower and more linear in its improvement.

Stellenbosch University https://scholar.sun.ac.za

and tweak models in 2017 made it much easier to test assumptions, compared to the struggles and long training times faced in 2015.

Before we evaluate the models and see which ones perform the best in our real world simulation, we look at a creative way to use existing models to solve our problem.

4.4 Transfer learning

Training a network from scratch to solve a complex classification problem such as ours proved to be tricky; we just do not have enough data to train the model as well as we would like. Another approach is to take existing models that were trained on larger sets of data, and to modify the weights to fit our smaller dataset. We discussed the idea of transfer learning in Chapter 2, and now we take a practical look at the potential benefits of this technique. We start with a simpler model in 2015 and use the different layers as feature extractors. In 2017 we fine-tune more complex models to meet our requirements.

In 2015 we chose to use the AlexNet [21] architecture discussed in Section 2.4 to investigate transfer learning. We wanted a relatively simple model architecture, but still something that was out of our reach to train from scratch. Before modifying any of the weights, we wanted to determine the predictive power of the different layers inside a pre-trained AlexNet model. We used a Caffe Model Zoo AlexNet model that was trained on ILSVRC 2012. Using the $224\times224\times3$ dataset, we ran each image in the training set through the model and recorded the activations on each layer. We then trained a linear classified for each of the layers using the activation values and the known training labels. We did not train on the first and second convolutional layers due to the large number of output nodes. Table 4.5 shows the results on the validation set for each layer's linear classifier.

Layer	T_n	F_p	F_n	T_p	Precision	Recall	F_1	MCC
Pool 1	984	66	69	981	0.9370	0.9343	0.9356	0.8714
Pool 2	1037	13	21	1029	0.9875	0.9800	0.9837	0.9676
Conv 3	1034	16	26	1024	0.9846	0.9752	0.9799	0.9600
Conv 4	1030	20	30	1020	0.9808	0.9714	0.9761	0.9524
Conv 5	1018	32	26	1024	0.9697	0.9752	0.9725	0.9448
Pool 5	1015	34	38	1012	0.9666	0.9638	0.9652	0.9305
FC 6	1048	2	155	895	0.9978	0.8524	0.9194	0.8597
FC 7	1048	2	244	806	0.9975	0.7676	0.8676	0.7869
FC 8	1050	0	296	754	1.000	0.7181	0.8359	0.7485

Table 4.5: Softmax classifier results for the different layers in the reference model on the validation set. Performance reaches its peak early on at the second pooling layer and slowly decreases as we go deeper into the network. This tells us the lower extracted features represent our data the best, which is to be expected from a model trained on data that is different from our own.

As we move through the model layers, we reach peak performance early on. This indicates that our data is best expressed using the lower level features of AlexNet. Since the reference model was training on data much different from our own, we might get better performance if we fine-tune the model using our data. This leads us to our next idea, partial fine-tuning. In 2015 we attempted to fine-tune the model, but with limited success. Table 4.6 show the marginal improvements gained by locking the earlier layers and allowing the last fully connected layers to train. We used a very aggressive learning strategy, which caused the fine-tuned model to overfit.

We also attempted to leave all the layers unlocked and fine-tune the entire model, but this did not work.

Stellenbosch University https://scholar.sun.ac.za

Layer	T_n	F_p	F_n	T_p	Precision	Recall	F_1	MCC
FC 6	1047	3	137	913	0.9967	0.8695	0.9288	0.8739
FC 7	1048	2	218	832	0.9976	0.7924	0.8832	0.8078
FC 8	818	232	419	631	0.7312	0.6010	0.6597	0.3862
Test prediction	1017	33	925	125	0.7911	0.1190	0.2070	0.1661

Table 4.6: Softmax classifier results for the fully connected layers in the partially fine-tuned model along with classification results on the validation set. The first two layers improve in performance compared to the reference model, while the final classification layer performs much worse. This is also apparent in the classification performance of the model, with a very low recall score, indicating that the final layer (and consequently the model) overfit to the training data.

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Dropout	Learning Rate	Momentum	Locked Layers
148	50	0.994	**0.0157**	**0.9903**	0.044	0.7	0.0001	0.95	5
29	43	**0.9949**	0.0162	0.9853	0.0587	0.3	0.0001	0.99	5
185	50	0.8442	0.3553	0.8031	0.39	0.7	0.0001	0.99	20
63	49	0.7474	0.4967	0.761	0.4931	0.3	0.0001	0.9	20
117	18	0.53	0.6806	**0.473**	0.7035	0.5	0.01	0.99	20
131	22	0.5351	0.691	0.4749	0.6934	0.5	0.001	0.9	25

Table 4.7: Six VGG19 models out of the 180 model hyperparameter batch, with the best performing models on top, randomly selected average models in the middle, and the worst performing models at the bottom.

In 2017 we used two models in our transfer learning experiments, the first being VGG19 [27]. Back in 2014 this was a strong contender in the ImageNet challenge, promoting the idea that deep model architectures are key in building strong visual classifiers. We feel more confident now using such a complex model than in 2015 given the improved deep learning libraries and support from the academic community. Using the Valohai platform[1] we did hyperparameter optimisation to fine-tune the model, with dropout options on the new top layer of $[0.3, 0.5, 0.9]$, learning rates at $[0.01, 0.001, 0.0001]$, momentum at $[0.5, 0.9, 0.95, 0.99]$, and locking different numbers of layers from 5 to 25 (in increments of 5). All the images were scaled and augmented in real-time during training with early-stop enabled. This resulted in 180 different fine-tuned models.

Table 4.7 shows the results of six different models: the two best performing, two average performing, and the two worst performing ones. We show the training metrics, as well as the hyperparameters used on those specific models. The models are ranked by epoch validation accuracy. It is promising to see accuracy values in the 99% range, something we have not seen in previous models. All the model results can be seen in the Appendix.

In Figure 4.8 we visualise and compare the accuracy and loss for some of the models in Table 4.7. There are clear differences in the routes taken by each of the models during training. With the luxury of running 180 different configurations of the same experiment, we can investigate what impact the different hyperparameters have.

Table 4.8 shows the different hyperparameters used, along with the average epoch validation accuracy obtained by models with that parameter configuration. Changes in dropout did not have a major impact on training, while the largest impact can be seen from the learning rate. The more aggressive the learning rate, the worst the model performs. Combining this with a slower momentum seems to be the best combination. We assumed that changing the number of locked layers would result in more overfitting and allow lower locked models to perform better, but the results show that keeping most of the layers locked is a better strategy for this particular problem.

Figure 4.9 shows the final validation accuracy distribution over all 180 models. We note that a large majority of the models did not adapt well at all, remaining in the 50% accuracy range. About 15% of the models performed better than 95% accuracy. This shows us how important the correct

[1] The 2017 experiments can be repeated on Valohai using `https://bitbucket.org/marais/msc-deepspoor`

Stellenbosch University https://scholar.sun.ac.za

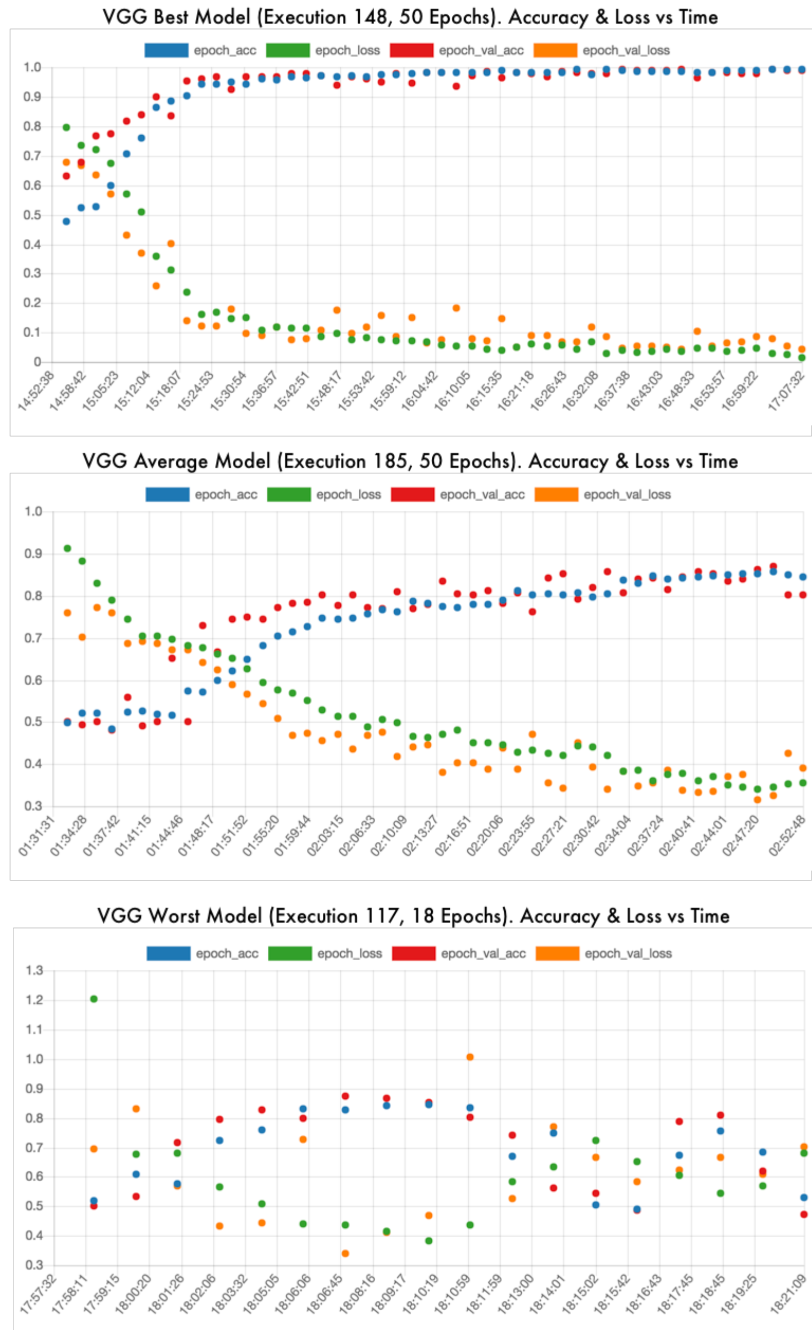


Figure 4.8: Accuracy and loss comparison of the best, average, and worst performing models in the hyperparameter batch.

Dropout	0.3	0.5	0.7		
	0.6964	0.6856	0.6558		
Learning rate	0.01	0.001	0.0001		
	0.5507	0.6402	**0.8468**		
Momentum	0.5	0.9	0.95	0.99	
	0.7411	0.6755	0.6732	0.6272	
Locked layers	5	10	15	20	25
	0.6683	0.665	0.7035	**0.7902**	0.5692

Table 4.8: Looking at the hyperparameter combinations with the average validation accuracy obtained by VGG19 models using that configuration. We not that learning rate has the highest impact on training performance, while dropout has the lowest.

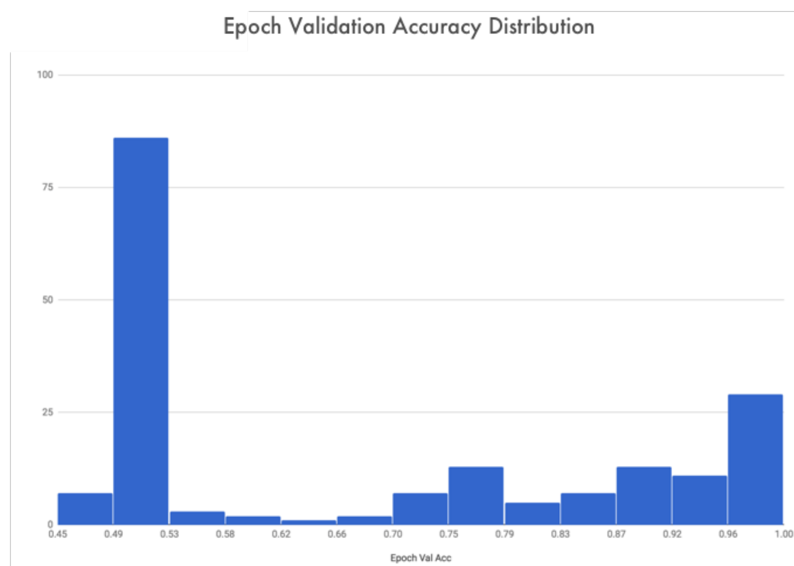


Figure 4.9: Final epoch validation accuracy for all 180 VGG19 models. Nearly half of the models did not adapt at all, and above 15% of the models reached 95% accuracy of more.

hyperparameter choices are in training the best model.

Our final experiment was to fine-tune a Google InceptionV3 [28] model. With the results obtained on the VGG19 model, we wanted to see if a more recent and complex model would perform better. We trained 12 models, using hyperparameter optimisation on learning rate and momentum.

Table 4.9 contains the results of the best and worst performing models. All of the results can be seen in the Appendix. We note that there are no average models, only models that adapted above 90% validation accuracy, and models that did not adapt at all. We obtain the same best validation accuracy here as with the VGG19 model.

Compared to the hyperparameter impact observed with the VGG19 models, the InceptionV3 models performed better with a more aggressive learning rate and a higher momentum, as shown in Table 4.10. This can be accredited to the increased complexity of the network architecture and its ability to represent the data in more unique ways.

We have models trained from scratch, models used as feature extractors, and pre-trained

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Learning Rate	Momentum
359	31	**0.9991**	**0.0025**	**0.9903**	0.0954	0.01	0.95
355	20	0.4932	8.1694	0.4982	8.0887	0.0001	0.5

Table 4.9: Best (top) and worst (bottom) performing InceptionV3 models.

Learning rate	0.01	0.001	0.0001	
	0.7444	0.7385	0.6204	
Momentum	0.5	0.9	0.95	0.99
	0.6605	0.6605	**0.8234**	0.6599

Table 4.10: Hyperparameter impact on InceptionV3 models, showing that a more aggressive learning rate and momentum resulted in the highest average validation accuracy.

models that are fine-tuned to our task. To determine which of these is the strongest, and to test our hypothesis, we need to evaluate their performance as part of the complete elephant detection and classification system.

4.5 System performance

The main goal of this thesis is to investigate the feasibility of a system that uses a fast detector and a sophisticated classifier to detect elephants from aerial imagery of difficult environments.

Our systems starts with the original IR and RGB images. Using the techniques from Chapter 3 we process the IR images and determine which regions might contain elephants, we then map the locations to the RGB images and crop patches out. In this chapter we trained our models, and now we are ready to evaluate their performance through different scenarios. We start with a single IR and RGB image pair, use the IR image to determine the regions of interest, map those regions to the RGB image, crop them, and finally classify them. Using the ground truth labels we can determine the performance of each model on the same data. The most important test is how well the models perform on the entire dataset of 890 images, 121 of which contain elephants. This final test is the closest we can simulate a real-world scenario, and will give us a good indication of how feasible our approach would be out in the field. This test does include data similar to that used during training, which is usually considered a taboo. Unfortunately we were not able to capture more data, and we decided to include this test just as a proof-of-concept that the trained models can be applied to full images. It should be noted that these images include large numbers of negative regions not seen during training, as well as unaugmented positive samples badly centred (via the unadjusted homograph method, as opposed to the manual method used for training), and we therefore believe the results are relevant and useful.

There are a few differences between the 2015 and 2017 results. The differences are due to slight changes in the region proposal algorithm, crop size and image registration. The IR proposal algorithm implementation of 2015 could not be reproduced exactly in 2017, resulting in a slightly different detection count (53,867 in 2015 vs 55,507 in 2017).

We attempted to do an RGB only evaluation on the 2015 models using a sliding window, but the number of image crops was so large that it was deemed impractical at the time.

Starting with a single image test, we have 60 proposed regions: 56 of the regions are negative and 4 are positive. This is visualised in Figure 4.10, with the positive regions in blue, and the negative regions in black. This test was performed by cropping the detected regions at the crop size required by each model and recording the prediction performance of all the 2015 and 2017 models in Table 4.11.

The best performing model in 2015 was an ensemble of the models trained from scratch, while all of the fine-tuned models in 2017 take the top spots. We notice that the models in 2015

Stellenbosch University https://scholar.sun.ac.za

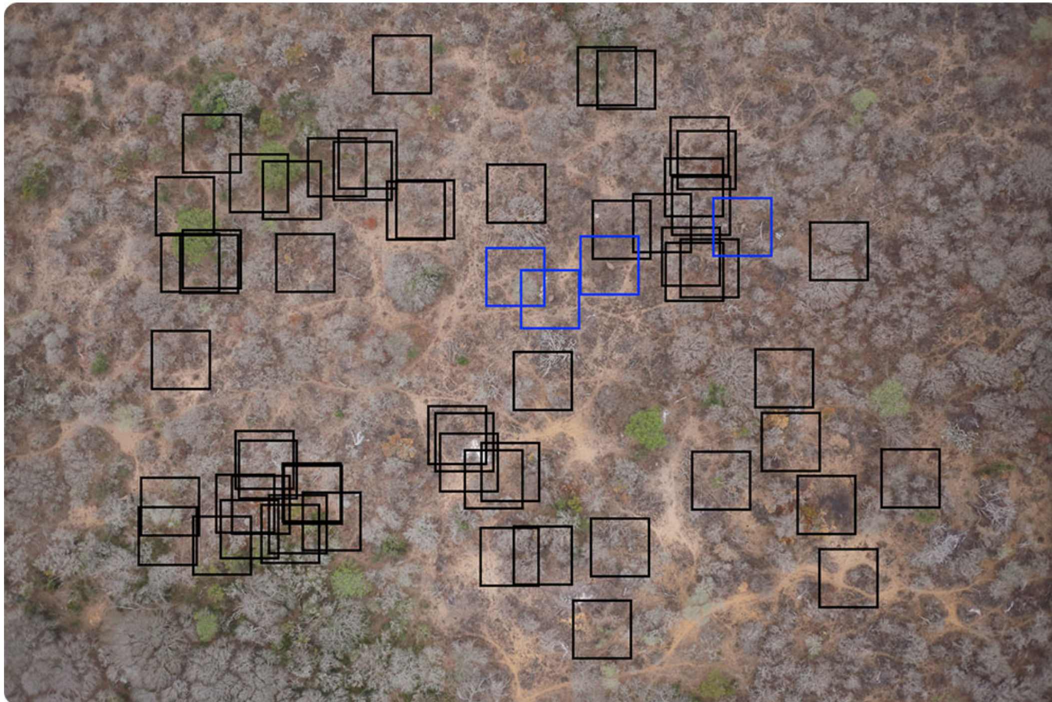


Figure 4.10: Marked RGB image used to evaluate the models' performance on a single image. The positive regions are marked in blue, and the negative regions marked in black.

have a higher recall rate, but fail to compete with precision.

For the 2017 models we added an intermediate experiment with only the 121 images that contained elephants. We detected 5,624 regions: 4,749 negative and 875 positive. The results of this experiment are listed in Table 4.12. The best performing are the fine-tuned VGG19 and InceptionV3 models, obtaining the highest precision and recall. We use these two models in our final experiment.

For our final experiment we used our region proposal algorithm on all 890 images, mapping them to RGB using the adjusted homography in 2015, and the unadjusted homography in 2017. There are more than 50,000 detected regions (53,867 in 2015, and 55,507 in 2017), which would prove frustrating for a human to verify considering the small number of positive images in that batch (778 in 2015, 875 in 2017). The results are listed in Table 4.13. The best performer is the fine-tuned InceptionV3 model, with the highest precision and competing recall.

Let us discuss the results and visualise the winning model's performance on a few images.

4.6 Conclusions on the results

Starting with the single image test, we see that the 2015 models have better recall, but suffer from weaker precision. This as an indication that these models have overfit to the training data, and will not generalise well. This is confirmed in the final test, where the models overdetect and generate a lot of false positive results, making them impractical for real world use.

The 2017 models that were trained from scratch do not do much better on the positive image set test. This confirms our assumption that we do not have enough data (no matter the augmentation strategy) to train a useful model from scratch.

The best performer as given by the MCC is the InceptionV3 fine-tuned model, with the highest precision and a competing recall rate. The goal was never to train the perfect model, which

Stellenbosch University https://scholar.sun.ac.za

Single Image (2015)									
Model	T_p	T_n	F_p	F_n	Precision	Recall	F_1	MCC	TNR
Trained Simple NN	4	36	20	0	0.1667	1	0.2857	0.3273	0.6429
Trained Simple CNN	4	42	14	0	0.2222	1	0.3636	0.4082	0.75
Trained Stacked CNN	4	39	17	0	0.1905	1	0.32	0.3642	0.6964
Trained Deeper CNN	4	43	13	0	0.2353	1	0.381	0.425	0.7679
Ensemble Threshold 1	4	27	29	0	0.1212	1	0.2162	0.2417	0.4821
Ensemble Threshold 2	4	34	22	0	0.1538	1	0.2667	0.3056	0.6071
Ensemble Threshold 3	4	42	14	0	0.2222	1	0.3636	0.4082	0.75
Ensemble Threshold 4	4	46	10	0	0.2857	1	0.4444	0.4845	0.8214
Ensemble Threshold 5	4	51	5	0	**0.4444**	**1**	**0.6154**	**0.6362**	**0.9107**
Reference Model	1	51	5	3	0.1667	0.25	0.2	0.1336	0.9107
Single Image (2017)									
Model	T_p	T_n	F_p	F_n	Precision	Recall	F_1	MCC	TNR
Scratch2 - 219	1	33	23	3	0.0417	0.25	0.0714	-0.0818	0.5893
Scratch3 - 330	2	30	26	2	0.0714	0.5	0.125	0.0179	0.5357
Scratch4 - 290	2	28	28	2	0.0667	0.5	0.1176	0	0.5
Scratch2 (500) - 486	3	23	33	1	0.0833	0.75	0.15	0.0818	0.4107
Scratch3 (500) - 590	4	3	53	0	0.0702	1	0.1311	0.0613	0.0536
Scratch4 (500) - 560	4	3	53	0	0.0702	1	0.1311	0.0613	0.0536
VGG19 - 28	3	56	0	1	**1**	**0.75**	**0.8571**	**0.8584**	**1**
VGG19 - 29	3	56	0	1	**1**	**0.75**	**0.8571**	**0.8584**	**1**
VGG19 - 89	3	56	0	1	**1**	**0.75**	**0.8571**	**0.8584**	**1**
VGG19 - 148	3	56	0	1	**1**	**0.75**	**0.8571**	**0.8584**	**1**
InceptionV3 - 359	3	56	0	1	**1**	**0.75**	**0.8571**	**0.8584**	**1**

Table 4.11: Single image performance results for all the models trained.

Positive Images (2017)									
Model	T_p	T_n	F_p	F_n	Precision	Recall	F_1	MCC	TNR
Scratch2 - 219	388	2761	1988	487	0.1633	0.4434	0.2387	0.0182	0.5814
Scratch3 - 330	526	1908	2841	349	0.1562	0.6011	0.248	0.0022	0.4018
Scratch4 - 290	363	2909	1840	512	0.1648	0.4149	0.2359	0.0204	0.6126
Scratch2 (500) - 486	629	1528	3221	246	0.1634	0.7189	0.2662	0.0317	0.3218
Scratch3 (500) - 590	720	809	3940	155	0.1545	0.8229	0.2602	-0.0065	0.1704
Scratch4 (500) - 560	715	832	3917	160	0.1544	0.8171	0.2597	-0.0073	0.1752
VGG19 - 28	863	4662	87	12	0.9084	0.9863	0.9458	0.9364	0.9817
VGG19 - 29	874	4630	119	1	0.8802	**0.9989**	0.9358	0.9257	0.9749
VGG19 - 89	873	4669	80	2	0.9161	0.9977	0.9551	0.9477	0.9832
VGG19 - 148	867	4700	49	8	0.9465	0.9909	0.9682	0.9625	0.9897
InceptionV3 - 359	865	4727	22	10	**0.9752**	0.9886	**0.9818**	**0.9785**	**0.9954**

Table 4.12: Model performance on the 121 images in the dataset that do contain elephants.

Stellenbosch University https://scholar.sun.ac.za

All Images (2015)									
Model	T_p	T_n	F_p	F_n	Precision	Recall	F_1	MCC	TNR
Trained Simple NN	743	45489	7590	35	0.0892	0.955	0.1631	0.2697	0.857
Trained Simple CNN	754	47188	5891	24	0.1135	0.9692	0.2032	0.3113	0.889
Trained Stacked CNN	748	47154	5925	30	0.1121	0.9614	0.2008	0.3078	0.8884
Trained Deeper CNN	746	46064	7015	32	0.0961	0.9589	0.1747	0.2809	0.8678
Ensemble Threshold 1	769	37756	15323	9	0.0478	**0.9884**	0.0912	0.1824	0.7113
Ensemble Threshold 2	763	44619	8460	15	0.0827	0.9807	0.1526	0.2601	0.8406
Ensemble Threshold 3	750	47927	5152	28	0.1271	0.964	0.2246	0.3312	0.9029
Ensemble Threshold 4	738	49977	3102	40	0.1922	0.9486	0.2196	0.4128	0.9416
Ensemble Threshold 5	722	51694	1385	56	**0.3427**	0.928	**0.5005**	**0.5551**	**0.9739**
Reference Model	437	47584	5495	341	0.0737	0.6617	0.1303	0.1746	0.8965
Locked Fine-tuned Model	596	17646	35433	182	0.0165	0.7661	0.0324	0.025	0.3324
All Images (2017)									
Model	T_p	T_n	F_p	F_n	Precision	Recall	F_1	MCC	TNR
VGG19 - 29	873	53139	1493	2	0.369	**0.9977**	0.5387	0.5983	0.9727
InceptionV3 - 359	865	53622	1010	10	**0.4613**	0.9886	**0.6291**	**0.6688**	**0.9815**

Table 4.13: Model performance on all 890 images in the dataset.

is why we did not run every single model through the tests. Through our test and model selection choices we show that a fine-tuned model is the way to go, with more recent and advanced architectures providing better results.

To get a visual perspective of what we have achieved, we illustrate the full system in Figure 4.11 and Figure 4.12.

In Figure 4.11 we have the same image used in the single image test, showing the original IR image along with the stages in the region proposal algorithm. The image contains three visible elephants and one partially visible elephant. Our algorithm proposed 60 regions, and our classifier marked all 56 negative images as negative (white), three of the four elephants as positive (green), with the partially visible elephant marked incorrectly (in red).

Figure 4.12 shows the image with the most elephants in our dataset, 35 in total (22 visible, 9 partially visible, 2 calfs, and 2 hidden). Our region proposal algorithm detected 25 regions, 24 of which are positive and one which is negative. Our classifier correctly classified 23 of the positive regions correctly (green), along with the negative region as negative (white), and one positive region as negative (red).

Stellenbosch University https://scholar.sun.ac.za

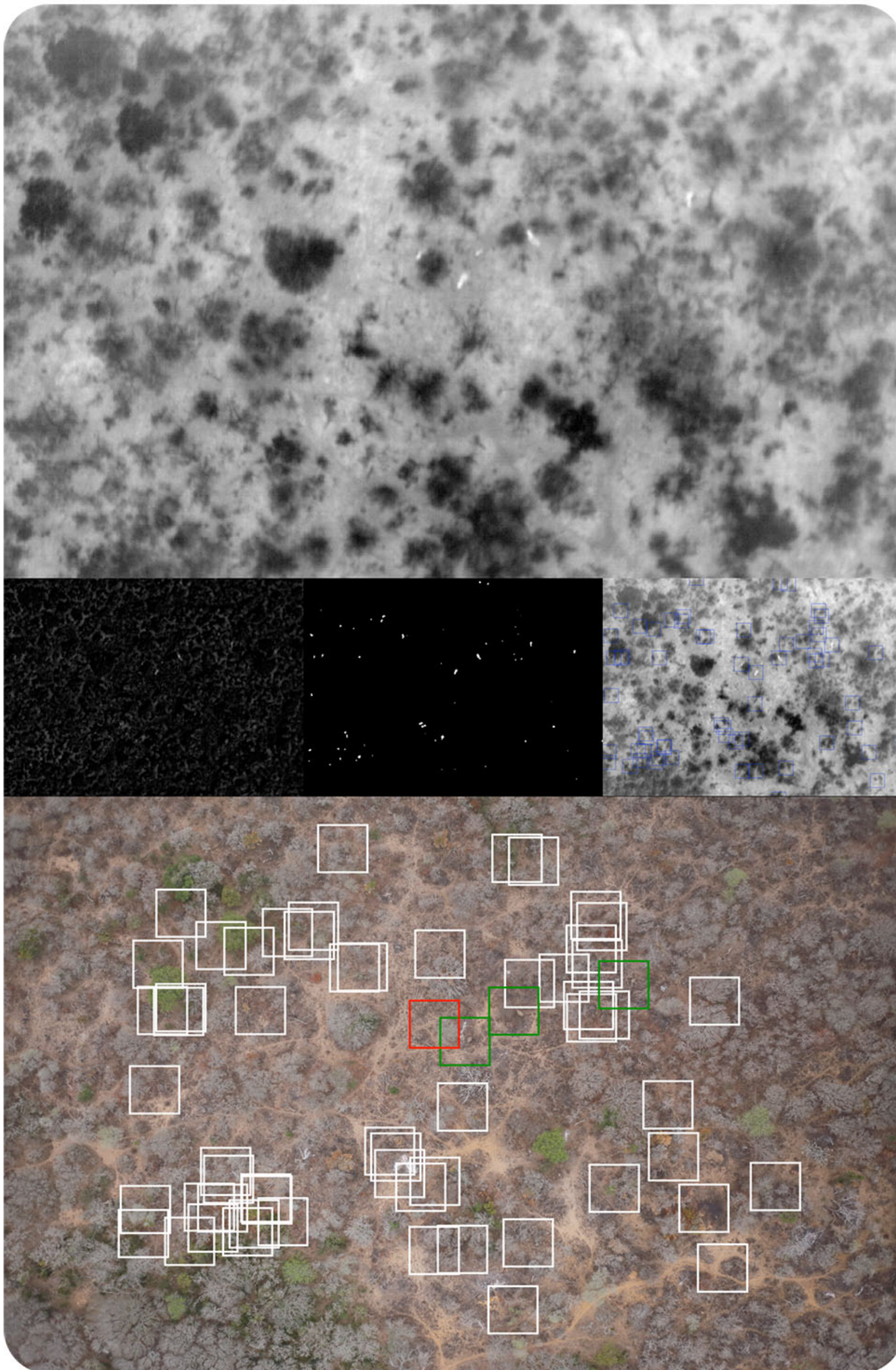


Figure 4.11: Full system overview 1: 60 detected regions, 56 negative regions classified as negative (white), three positive regions classified as positive (green), and one positive regions classified as negative (red).

Stellenbosch University https://scholar.sun.ac.za

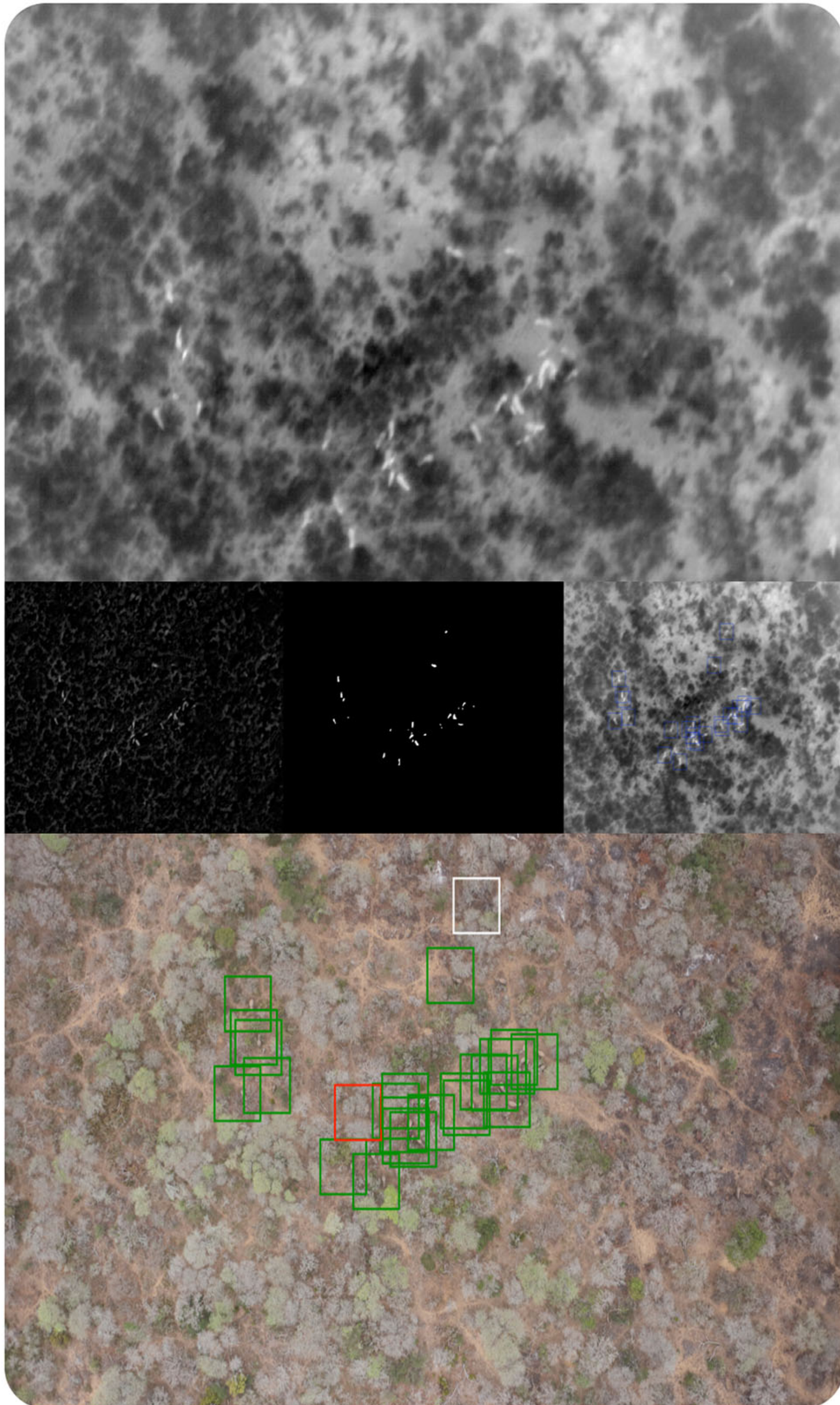


Figure 4.12: Full system overview 2: 25 detected regions, 2 negative regions classified as negative (white), 23 positive regions classified as positive (green), and one positive regions classified as negative (red).

Stellenbosch University https://scholar.sun.ac.za

5 | Conclusion

> *"The race is now on between the technoscientific and scientific forces that are destroying the living environment and those that can be harnessed to save it... If the race is won, humanity can emerge in far better condition than when it entered, and with most of the diversity of life still intact."*
>
> \- Edward O. Wilson, *The Future of Life*

The goal of this thesis was to determine the feasibility of using computer vision and deep learning to build an affordable and usable system that could detect elephants from aerial infrared and RGB images, providing a reduced set of options to a human for verification.

The inspiration for this goal was the adverse impact humanity is having on the environment around us, and how we can use technology as a tool to minimise or even reverse this effect. The Great Elephant Census and the related SkyReach project was my first encounter with what I now know as PeaceTech, which aims to use technology in order to promote peace in all forms.

We have the knowledge and resources to make a significant difference in the lives of so many people, and preserve the natural world world around us. Planet Labs Inc. [56] had a vision to capture and index every visible piece of earth from space, every single day. They have accomplished this goal using more than 175 satellites in orbit. The opportunities this presents in conservation, disaster response, humanitarian aid, as well as numerous other areas is truly motivating. The thought of applying machine learning techniques (such as the ones employed in this thesis) on their data gives credibility to our study and the need for similar work.

5.1 Summary and conclusions

In the spirit of creating a minimal viable product, we did not use satellite imagery in this thesis. Instead we used the most cost effective means available, relying on consumer grade hardware on an enthusiast level light aircraft to capture data that was flawed in some ways, as discussed in Chapter 1. If we could make it work (and we did) with this data, it would be a strong indicator for future work using higher fidelity data.

In Chapter 2 we dived into the theory of machine vision, machine learning, and specifically deep learning. It is an exciting time to be involved in both the academic and commercial side of this field. The theory is evolving at a rapid pace, with hardware and software improvements following closely behind. We are at a point where our algorithms detect cancer better than a human specialist [57] and drive cars on public roads. Hopefully the focus will shift more towards sustainable solutions such as the smart grid or process optimisation inside large production companies in an effort to meet important, yet largely ignored, climate goals [58].

We started our investigation and implementation on the infrared images we had available in Chapter 3. Given the complexity of the RGB images, we wanted to develop a region proposal algorithm that would reduce the number of regions our deep learning classifier would have to process. We succeeded in this goal, creating an algorithm that could achieve 98% true positive

Stellenbosch University https://scholar.sun.ac.za

accuracy. There was a high overdetection cost associated with this though, and we found that being slightly less restrictive on our parameters reduced the overdetections by a large amount while only having a slight impact on accuracy. We dealt with the misaligned infrared and RGB images using a global homograpy along with some heuristics. This allowed us to proceed to the classification phase of the work with confidence.

The most important and telling part of the thesis is Chapter 4, where we showed that it was indeed possible to build a system that could detect and classify elephants from our data. With a recall and true negative rate above 98% from our best performing model, we reduced the number of proposed regions from 55,507 to 1,875. This removed almost 97% of the manual verification work, while only missing 1% of the elephants detected during the region proposal phase. By revisiting the problem in 2017 we could compare the improvements in the field since the beginning of 2015. We can conclude that we do not have enough data available to train a model from scratch, as these models do not generalise well, even though the neural networks are capable of learning core features that distinguish elephants from their environment. Transfer learning is a powerful tool that allowed us to take advantage of the resources available to large companies such as Google and apply it to our problem. Combining modern architectures such as InceptionV3, cloud infrastructure from Amazon, an experimentation platform like Valohai and open source frameworks like TensorFlow and Keras we were able to create an affordable and realistic solution to our problem.

5.2 Future work

To continue this work we would look at combining the 180 VGG19 models that resulted from the hyperparameter optimisation task into an ensemble network. We attempted a naive ensemble of the 2015 models with surprisingly good results, allowing weaker models to perform better combined than individually. This is definitely an area worth investigating further with more sophisticated methods to calculate model consensus.

Further work can be done on the infrared data, either by improving the region proposal algorithm performance or by utilising the infrared information such as the shape and intensity information during the classification phase.

Any future data captured using similar means would ideally not have the misalignment issues that we faced, and would make the task of image registration much simpler. Further work using our dataset could benefit from research on mutual information between similar regions in images [59].

An extension of this work could focus on using only the available RGB data, removing the infrared data from the process entirely. Research on using deep convolutional neural networks for object detection, localisation and segmentation has gained popularity. We suggest starting the investigation with R-CNN [60] from 2014, Faster R-CNN [61] from 2015, and YOLO [62] from 2016.

Convolutional neural networks have been considered the state-of-the-art since 2012 for image recognition tasks, but in 2017 Geoffrey Hinton and his team introduced a new form of neural network based on capsules [63]. This work promises to improve on CNN performance and be less susceptible to the same flaws and tricks that fool the network into predicting unrecognisable images into a class with high certainty [64]. This area of research holds a lot of potential for our problem, as well as all other image recognition tasks.

5.3 Reflections

The journey for this thesis is similar to what many students have faced. The early days were full of energy and excitement, most of the work was done within the first two years and everything was

Stellenbosch University https://scholar.sun.ac.za

heading towards completion. A casual conversation in the research lab in November 2015 turned into a job opportunity that extended this journey from one more month to a few additional years.

The opportunities that presented themselves along the way were life altering. I am grateful for every person that shared in my adventures and escapades over the last four years, for every person that was excited about detecting elephants from images in the sky, and with whom I could share my passion for technology and the positive impact it can have on our world.

For once I was not the ideal student that finished on time, and I am glad I did not. The conversations about this thesis still lead to interesting opportunities that I cannot wait to explore in the future.

I end this thesis with one of the first images that was accidentally created while experimenting with convolutions and deep learning. While tinkering with messy code, I generated the image in Figure 5.1. My first thought was of art and how closely connected technology and creativity could be. I hope to take what I have learned over the last few years, to combine technology and creativity and to create beautiful and impactful works that will change the world for the better.

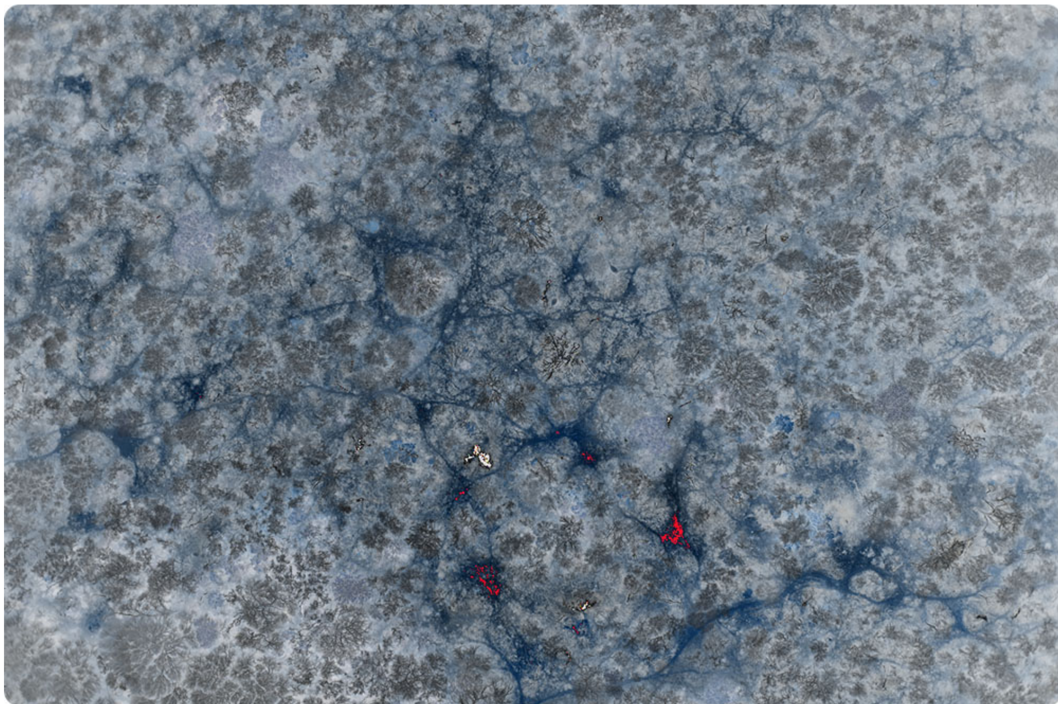


Figure 5.1: Late night in the lab, tinkering with messy cody, I ended up with an RGB image that was filtered by a random convolutional kernel and displayed with its colour channels in the wrong order. There is something beautiful about this image that made me keep it from the beginning until the end.

Stellenbosch University https://scholar.sun.ac.za

Bibliography

[1] Center for Biological Diversity. The extinction crisis. http://www.biologicaldiversity.org/programs/biodiversity/elements_of_biodiversity/extinction_crisis/, 2017. [Online; accessed 10 December 2017].

[2] Save The Elephants. Annual report 2016. http://www.savetheelephants.org/wp-content/uploads/2017/05/2016STE_AnnualReport_web.pdf, 2016. [Online; accessed 10 December 2017].

[3] National Geographic. Reducing demand for ivory: An international study. http://press.nationalgeographic.com/files/2015/09/NGS2015_Final-August-11-RGB.pdf, 2015. [Online; accessed 10 December 2017].

[4] WildAid. Ivory demand in china 2012-2014. https://www.wildaid.org/sites/default/files/resources/Print_Ivory%20Report_Final_v3.pdf, 2014. [Online; accessed 10 December 2017].

[5] Michael J Chase, Scott Schlossberg, Curtice R Griffin, Philippe JC Bouché, Sintayehu W Djene, Paul W Elkan, Sam Ferreira, Falk Grossman, Edward Mtarima Kohi, Kelly Landen, et al. Continent-wide survey reveals massive decline in African savannah elephants. *PeerJ*, 4:e2354, 2016.

[6] The Great Elephant Census. 2016. http://www.greatelephantcensus.com/s/GEC-Results-Summary-Fact-Sheet-FINAL_8-26-2016.pdf, Census Results Summary. [Online; accessed 10 December 2017].

[7] The Great Elephant Census. Country-by-country findings. http://www.greatelephantcensus.com/s/GEC-Results-Country-by-Country-Findings-Fact-Sheet_FINAL_8-26-2016.pdf, 2016. [Online; accessed 10 December 2017].

[8] Frederic Maire, Luis Mejias Alvarez, and Amanda Hodgson. Automating marine mammal detection in aerial images captured during wildlife surveys: A deep learning approach. In *Australasian Joint Conference on Artificial Intelligence*, pages 379–385. Springer, 2015.

[9] Mohammed Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Ali Swanson, Craig Packer, and Jeff Clune. Automatically identifying wild animals in camera trap images with deep learning. *arXiv preprint arXiv:1703.05830*, 2017.

[10] Chien-Hung Chen and Keng-Hao Liu. Stingray detection of aerial images with region-based convolution neural network. In *Consumer Electronics-Taiwan (ICCE-TW), 2017 IEEE International Conference on*, pages 175–176. IEEE, 2017.

[11] Kalanit Grill-Spector and Rafael Malach. The human visual cortex. *Annu. Rev. Neurosci.*, 27:649–677, 2004.

Stellenbosch University https://scholar.sun.ac.za

[12] Andrej Karpathy. CS231n convolutional neural networks for visual recognition. `http://cs231n.github.io/`, 2017. [Online; accessed 10 December 2017].

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Tom M Mitchell. Machine learning. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[16] Moshe Idel. Golem: Jewish magical and mystical traditions on the artificial anthropoid. 1990.

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[18] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[19] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[20] Raja Giryes, Guillermo Sapiro, and Alexander M Bronstein. Deep neural networks with random Gaussian weights: a universal classification strategy? *IEEE Trans. Signal Processing*, 64(13):3444–3457, 2016.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[24] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.

[25] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Stellenbosch University https://scholar.sun.ac.za

[30] Alex Alemi. Improving inception and image classification in TensorFlow. `https://research.googleblog.com/2016/08/improving-inception-and-image.html`, 2016. [Online; accessed 30 December 2017].

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[33] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[34] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[35] Timothy Dozat. Incorporating Nesterov momentum into Adam. *Technical Report, Stanford University*, 2015. `http://cs229.stanford.edu/proj2015/054_report.pdf`.

[36] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[37] Andrew Senior, Georg Heigold, Ke Yang, et al. An empirical study of learning rates in deep neural networks for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6724–6728. IEEE, 2013.

[38] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[39] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[40] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[41] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[42] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, et al. Lasagne: First release., August 2015.

[43] Daniel Nouri. NoLearn: scikit-learn compatible neural network library. `https://github.com/dnouri/nolearn`, 2014.

[44] TensorFlow. `https://www.tensorflow.org/`. [Online; accessed 10 December 2017].

[45] Keras: The Python deep learning library. `https://keras.io/`. [Online; accessed 10 December 2017].

Stellenbosch University https://scholar.sun.ac.za

[46] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[47] Valohai. `https://valohai.ai/`. [Online; accessed 10 December 2017].

[48] Jiabao Wang, Yafei Zhang, Jianjiang Lu, and Yang Li. Target detection and pedestrian recognition in infrared images. *JCP*, 8(4):1050–1057, 2013.

[49] Sungho Kim. Sea-based infrared scene interpretation by background type classification and coastal region detection for small target detection. *Sensors*, 15(9):24487–24513, 2015.

[50] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015. [Online; accessed 10 December 2017].

[51] Tony Lindeberg. *Discrete scale-space theory and the scale-space primal sketch*. PhD thesis, KTH Royal Institute of Technology, 1991.

[52] C.A. Glasbey and G.W. Horgan. *Image analysis for the biological sciences*. Statistics in practice. J. Wiley, 1995.

[53] P. Peterlin. Morphological operations: An overview. `http://www.inf.u-szeged.hu/ssip/1996/morpho/morphology.html`, 1996. [Online; accessed 10 December 2017].

[54] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[55] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.

[56] Planet Labs Inc. Planet. `https://www.planet.com`, 2017. [Online; accessed 31 December 2017].

[57] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*, 2017.

[58] Joeri Rogelj, Michel Den Elzen, Niklas Höhne, Taryn Fransen, Hanna Fekete, Harald Winkler, Roberto Schaeffer, Fu Sha, Keywan Riahi, and Malte Meinshausen. Paris agreement climate proposals need a boost to keep warming well below 2 c. *Nature*, 534(7609):631–639, 2016.

[59] Daniel B Russakoff, Carlo Tomasi, Torsten Rohlfing, and Calvin R Maurer Jr. Image similarity using mutual information of regions. In *European Conference on Computer Vision*, pages 596–607. Springer, 2004.

[60] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[61] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[62] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

Stellenbosch University https://scholar.sun.ac.za

[63] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

[64] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*, 2017.

Stellenbosch University https://scholar.sun.ac.za

Appendix

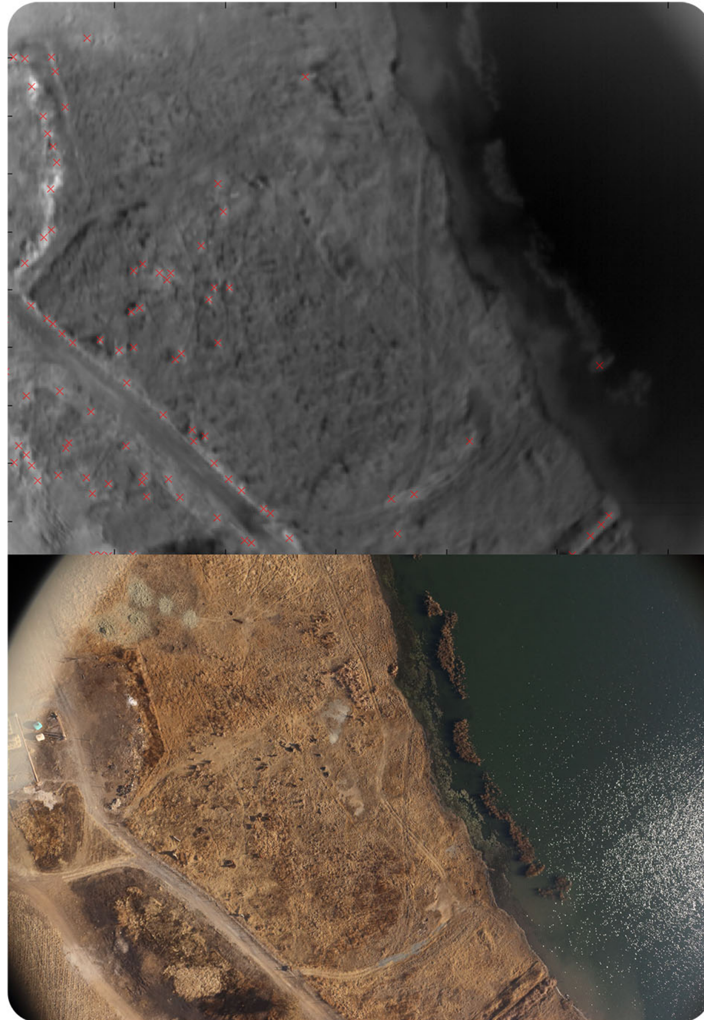


Figure 1: Cow data

Stellenbosch University https://scholar.sun.ac.za

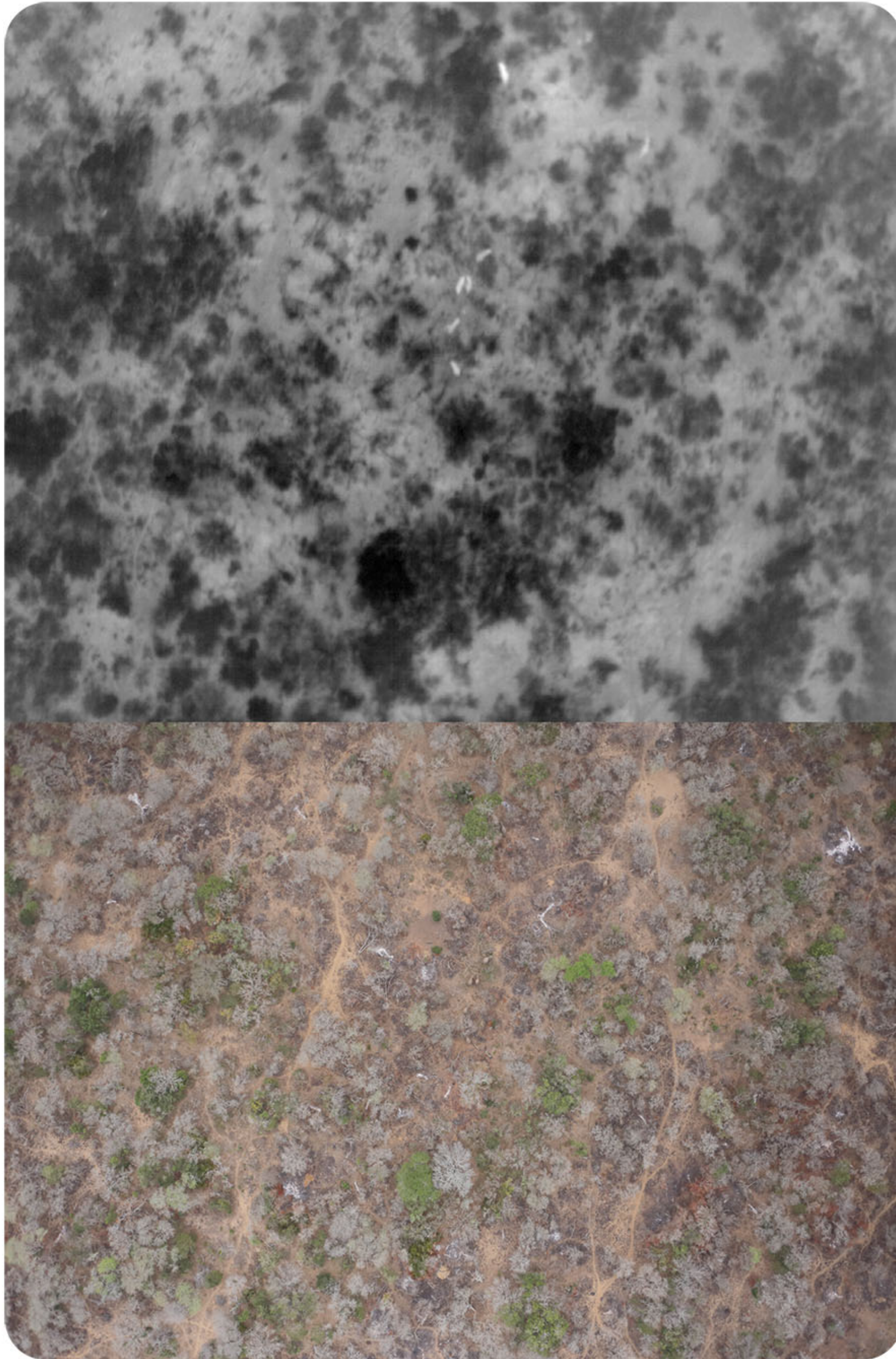


Figure 2: Image 55 IR and RGB

Stellenbosch University https://scholar.sun.ac.za

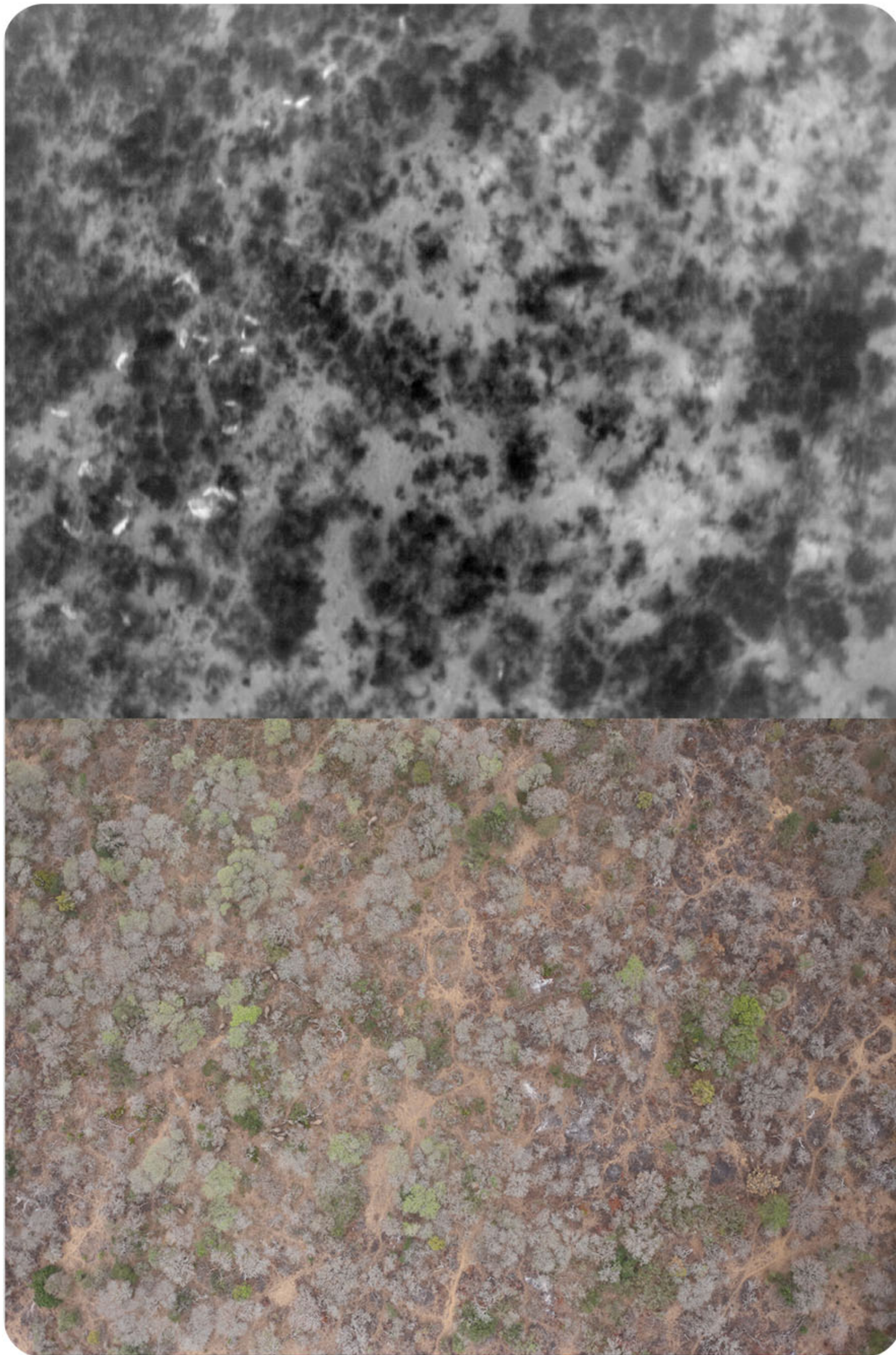


Figure 3: Image 89 IR and RGB

Stellenbosch University https://scholar.sun.ac.za

Figure 4: Positive images (sampled from training set)

Stellenbosch University https://scholar.sun.ac.za

Figure 5: Negative images (sampled from training set)

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum
201	16	0.5334	0.6923	0.4981	0.6915	1537	0.7188	0.6854	0.3	0.01	0.5
202	12	0.4546	0.694	0.4961	0.6932	1137	0.5313	0.6927	0.3	0.01	0.9
203	14	0.4974	0.6933	0.5193	0.6929	1337	0.5625	0.6922	0.3	0.01	0.95
204	14	0.4812	0.6966	0.4923	0.6951	1337	0.625	0.6835	0.3	0.01	0.99
205	21	0.5034	0.6931	0.5077	0.6927	2037	0.5313	0.6923	0.3	0.001	0.5
206	18	0.4786	0.6932	0.4961	0.6931	1737	0.4063	0.6932	0.3	0.001	0.9
207	22	0.5471	0.6861	0.5946	0.6853	2137	0.4688	0.6996	0.3	0.001	0.95
208	23	0.4632	0.6934	0.4903	0.6932	2237	0.4688	0.6938	0.3	0.001	0.99
209	20	0.4812	0.6942	0.5	0.6926	1937	0.4688	0.6916	0.3	0.0001	0.5
210	24	0.4991	0.6928	0.5116	0.6926	2337	0.5313	0.6947	0.3	0.0001	0.9
211	15	0.5111	0.6931	0.5039	0.6931	1437	0.5	0.6931	0.3	0.0001	0.95
212	36	0.5051	0.6929	0.4826	0.6924	3537	0.5313	0.6915	0.3	0.0001	0.99
213	28	0.5077	0.6935	0.5347	0.6915	2737	0.375	0.7056	0.5	0.01	0.5
214	12	0.4709	0.6936	0.4846	0.6935	1137	0.5	0.6944	0.5	0.01	0.9
215	18	0.5094	0.6948	0.4981	0.6932	1737	0.5	0.693	0.5	0.01	0.95
216	18	0.4863	0.6936	0.5154	0.6931	1737	0.4688	0.6931	0.5	0.01	0.99
217	15	0.4966	0.6931	0.4691	0.6932	1437	0.625	0.6926	0.5	0.001	0.5
218	17	0.4991	0.6932	0.5058	0.693	1637	0.4375	0.6934	0.5	0.001	0.9
219	22	0.5505	0.687	0.6448	0.6787	2137	0.5938	0.6839	0.5	0.001	0.95
220	35	0.5017	0.6933	0.5154	0.6929	3437	0.4375	0.6956	0.5	0.001	0.99
221	16	0.4812	0.6941	0.5058	0.693	1537	0.5	0.6942	0.5	0.0001	0.5
222	30	0.5334	0.693	0.4942	0.6931	2937	0.5625	0.6925	0.5	0.0001	0.9
223	32	0.5146	0.6931	0.529	0.6931	3137	0.5625	0.6929	0.5	0.0001	0.95
224	17	0.4829	0.6932	0.529	0.6931	1637	0.6563	0.6925	0.5	0.0001	0.99
225	25	0.5291	0.6923	0.5019	0.6925	2437	0.5938	0.6895	0.7	0.01	0.5
226	25	0.5128	0.693	0.5097	0.693	2437	0.4063	0.6987	0.7	0.01	0.9
227	13	0.4872	0.6934	0.5	0.6932	1237	0.625	0.6928	0.7	0.01	0.95
228	18	0.4983	0.6944	0.5116	0.6929	1737	0.4688	0.697	0.7	0.01	0.99
229	17	0.5163	0.6931	0.4807	0.6932	1637	0.625	0.6921	0.7	0.001	0.5
230	14	0.4914	0.6932	0.5039	0.6931	1337	0.375	0.6934	0.7	0.001	0.9
231	36	0.5077	0.6933	0.5193	0.692	3537	0.4688	0.6931	0.7	0.001	0.95
232	16	0.5068	0.6933	0.5	0.6932	1537	0.6563	0.6926	0.7	0.001	0.99
233	21	0.5034	0.6938	0.5	0.6933	2037	0.375	0.6952	0.7	0.0001	0.5
234	21	0.4717	0.6935	0.5037	0.6931	2037	0.375	0.695	0.7	0.0001	0.9
235	24	0.518	0.693	0.5039	0.6931	2337	0.5	0.6928	0.7	0.0001	0.95
236	17	0.5009	0.6931	0.5097	0.6931	1637	0.375	0.6936	0.7	0.0001	0.99

Table 1: Model 2 (2017): Training From Scratch - Early Stop, 36 Runs

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum
473	500	0.7175	0.5632	0.6911	0.5448	49937	0.625	0.6432	0.3	0.01	0.5
474	500	0.542	0.6851	0.6409	0.6631	49937	0.5	0.6683	0.3	0.001	0.5
475	500	0.5548	0.6839	0.6274	0.6795	49937	0.6875	0.6712	0.3	0.0001	0.5
476	500	0.6926	0.5919	0.6757	0.568	49937	0.7188	0.5407	0.5	0.01	0.5
477	500	0.5557	0.6777	0.6062	0.6656	49937	0.5313	0.678	0.5	0.001	0.5
478	500	0.5873	0.6801	0.6255	0.6707	49937	0.6563	0.6816	0.5	0.0001	0.5
479	500	0.7123	0.5874	0.6988	0.548	49937	0.7188	0.6243	0.7	0.01	0.5
480	500	0.5351	0.6897	0.6158	0.6885	49937	0.5	0.7069	0.7	0.001	0.5
481	500	0.5188	0.692	0.4961	0.6914	49937	0.6875	0.6776	0.7	0.0001	0.5
482	500	0.4872	0.6934	0.5077	0.6931	49937	0.5	0.6933	0.3	0.01	0.9
483	500	0.5077	0.6931	0.4903	0.6932	49937	0.5313	0.693	0.3	0.001	0.9
484	500	0.5394	0.6894	0.5502	0.6882	49937	0.375	0.6985	0.3	0.0001	0.9
485	500	0.5	0.6933	0.5077	0.693	49937	0.5	0.6932	0.5	0.01	0.9
486	500	0.7089	0.5479	0.7261	0.5398	49937	0.6875	0.5475	0.5	0.001	0.9
487	500	0.5171	0.6917	0.5811	0.6906	49937	0.5	0.6911	0.5	0.0001	0.9
488	500	0.4812	0.6933	0.5174	0.6931	49937	0.4063	0.6929	0.7	0.01	0.9
489	500	0.5026	0.6932	0.5039	0.6931	49937	0.4688	0.6932	0.7	0.001	0.9
490	500	0.5248	0.6917	0.5656	0.691	49937	0.4375	0.6994	0.7	0.0001	0.9
491	500	0.506	7.9624	0.4923	8.1835	49937	0.5625	7.0517	0.3	0.01	0.95
492	500	0.5094	0.6931	0.4807	0.6933	49937	0.4375	0.6937	0.3	0.001	0.95
493	500	0.4889	0.6932	0.5116	0.6931	49937	0.5625	0.6931	0.3	0.0001	0.95
494	500	0.4872	0.6939	0.473	0.6942	49937	0.4688	0.6945	0.5	0.01	0.95
495	500	0.6096	0.6607	0.5985	0.6663	49937	0.5625	0.6806	0.5	0.001	0.95
496	500	0.5462	0.6868	0.6042	0.6808	49937	0.5	0.684	0.5	0.0001	0.95
497	500	0.4889	0.6935	0.4807	0.6935	49937	0.5625	0.6923	0.7	0.01	0.95
498	500	0.6378	0.6455	0.6853	0.6131	49937	0.7188	0.6129	0.7	0.001	0.95
499	500	0.4966	0.6932	0.4961	0.6932	49937	0.4063	0.6933	0.7	0.0001	0.95
500	500	0.4974	0.6949	0.5077	0.6937	49937	0.4063	0.7049	0.3	0.01	0.99
501	500	0.5163	0.693	0.4672	0.6934	49937	0.3438	0.6947	0.3	0.001	0.99
502	500	0.6644	0.5956	0.6757	0.5879	49937	0.6563	0.609	0.3	0.0001	0.99
503	500	0.4897	0.6954	0.5039	0.6934	49937	0.4375	0.6982	0.5	0.01	0.99
504	500	0.506	0.6931	0.4923	0.6932	49937	0.5625	0.6927	0.5	0.001	0.99
505	500	0.5009	0.6931	0.4903	0.6932	49937	0.5625	0.6931	0.5	0.0001	0.99
506	500	0.5111	0.6932	0.5251	0.6927	49937	0.6875	0.6908	0.7	0.01	0.99
507	500	0.5103	0.6931	0.5	0.6931	49937	0.3438	0.6943	0.7	0.001	0.99
508	500	0.6558	0.6308	0.6486	0.6321	49937	0.5938	0.6574	0.7	0.0001	0.99

Table 2: Model 2 (2017): Training From Scratch - 500 Epochs, 36 Runs

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum
311	29	0.5068	0.6931	0.4807	0.693	2837	0.5625	0.693	0.3	0.01	0.5
312	12	0.4889	0.694	0.5077	0.693	1137	0.6563	0.691	0.3	0.01	0.9
313	30	0.4957	0.6934	0.4961	0.6933	2937	0.4375	0.6942	0.3	0.01	0.95
314	19	0.5068	0.6942	0.5097	0.6933	1837	0.5	0.694	0.3	0.01	0.99
315	40	0.5428	0.6912	0.5309	0.689	3937	0.5313	0.6884	0.3	0.001	0.5
316	23	0.542	0.6899	0.5058	0.6894	2237	0.5	0.6919	0.3	0.001	0.9
317	20	0.494	0.6931	0.4923	0.6929	1937	0.4688	0.6963	0.3	0.001	0.95
318	25	0.4837	0.6933	0.5463	0.6929	2437	0.5313	0.693	0.3	0.001	0.99
319	33	0.5205	0.6919	0.5212	0.6904	3237	0.5313	0.6879	0.3	0.0001	0.5
320	14	0.5068	0.6931	0.5174	0.6928	1337	0.5	0.6943	0.3	0.0001	0.9
321	14	0.482	0.6942	0.4846	0.6945	1337	0.5313	0.6907	0.3	0.0001	0.95
322	29	0.5248	0.6893	0.4884	0.6912	2837	0.5313	0.6845	0.3	0.0001	0.99
323	27	0.5719	0.6832	0.4961	0.7032	2637	0.625	0.6549	0.5	0.01	0.5
324	17	0.4752	0.6934	0.5135	0.6925	1637	0.6563	0.6902	0.5	0.01	0.9
325	19	0.4957	0.6932	0.4903	0.6935	1837	0.5938	0.6903	0.5	0.01	0.95
326	19	0.4854	0.694	0.5	0.6936	1837	0.5	0.6946	0.5	0.01	0.99
327	25	0.5171	0.693	0.5019	0.693	2437	0.5625	0.6909	0.5	0.001	0.5
328	27	0.4683	0.6933	0.4961	0.6931	2637	0.3125	0.6953	0.5	0.001	0.9
329	19	0.5531	0.6908	0.5058	0.6895	1837	0.5938	0.6856	0.5	0.001	0.95
330	25	0.5137	0.6928	0.6042	0.6921	2437	0.5625	0.6926	0.5	0.001	0.99
331	18	0.4983	0.6945	0.4961	0.6932	1737	0.5313	0.6938	0.5	0.0001	0.5
332	15	0.512	0.693	0.5077	0.6918	1437	0.4375	0.6933	0.5	0.0001	0.9
333	26	0.524	0.6929	0.4982	0.6931	2537	0.5313	0.6922	0.5	0.0001	0.95
334	28	0.5205	0.6928	0.5521	0.6904	2737	0.5938	0.6878	0.5	0.0001	0.99
335	23	0.5317	0.6905	0.5463	0.6889	2237	0.6875	0.6845	0.7	0.01	0.5
336	19	0.5094	0.6931	0.5097	0.693	1837	0.3125	0.7074	0.7	0.01	0.9
337	24	0.4863	0.6933	0.5116	0.6931	2337	0.5938	0.6922	0.7	0.01	0.95
338	20	0.5111	0.6931	0.5174	0.6925	1937	0.5313	0.6902	0.7	0.01	0.99
339	25	0.4932	0.6932	0.529	0.6931	2437	0.5313	0.6931	0.7	0.001	0.5
340	32	0.5034	0.693	0.4903	0.693	3137	0.4688	0.6933	0.7	0.001	0.9
341	41	0.5214	0.6924	0.5753	0.6911	4037	0.5	0.6902	0.7	0.001	0.95
342	32	0.5077	0.6929	0.4942	0.6933	3137	0.5313	0.6912	0.7	0.001	0.99
343	17	0.4949	0.6942	0.5077	0.6929	1637	0.5313	0.6938	0.7	0.0001	0.5
344	20	0.5248	0.6923	0.4556	0.6934	1937	0.375	0.695	0.7	0.0001	0.9
345	26	0.4846	0.6932	0.4926	0.6931	2537	0.375	0.6942	0.7	0.0001	0.95
346	38	0.5462	0.6907	0.5039	0.6904	3737	0.6875	0.6877	0.7	0.0001	0.99

Table 3: Model 3 (2017): Training From Scratch - Early Stop, 36 Runs

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum
581	500	0.8356	0.3484	0.8205	0.3725	49937	0.9688	0.2258	0.3	0.01	0.5
582	500	0.6062	0.658	0.61	0.6482	49937	0.8125	0.5831	0.3	0.001	0.5
583	500	0.6027	0.6715	0.6178	0.6644	49937	0.5938	0.6822	0.3	0.0001	0.5
584	500	0.9007	0.2476	0.8958	0.243	49937	0.875	0.2737	0.5	0.01	0.5
585	500	0.5822	0.6677	0.6197	0.6543	49937	0.5625	0.6638	0.5	0.001	0.5
586	500	0.5557	0.6877	0.5753	0.6825	49937	0.5938	0.6793	0.5	0.0001	0.5
587	500	0.9084	0.2522	0.9266	0.236	49937	0.9375	0.1418	0.7	0.01	0.5
588	500	0.5086	0.6931	0.5695	0.693	49937	0.5313	0.6935	0.7	0.001	0.5
589	500	0.5753	0.6779	0.6525	0.673	49937	0.5313	0.7106	0.7	0.0001	0.5
590	500	0.9495	0.1419	0.9633	0.1037	49937	0.875	0.1864	0.3	0.01	0.9
591	500	0.738	0.5374	0.7143	0.5438	49937	0.8125	0.4863	0.3	0.001	0.9
592	500	0.6062	0.6599	0.6409	0.6493	49937	0.625	0.6778	0.3	0.0001	0.9
593	500	0.726	0.548	0.7085	0.5582	49937	0.6563	0.5386	0.5	0.01	0.9
594	500	0.774	0.4974	0.7452	0.4881	49937	0.5938	0.6294	0.5	0.001	0.9
595	500	0.6002	0.6785	0.6178	0.6607	49937	0.6563	0.6821	0.5	0.0001	0.9
596	500	0.4854	0.6933	0.4923	0.6932	49937	0.5938	0.6925	0.7	0.01	0.9
597	500	0.5933	0.6732	0.6236	0.6605	49937	0.5	0.6888	0.7	0.001	0.9
598	500	0.5308	0.6884	0.6274	0.6859	49937	0.5938	0.6884	0.7	0.0001	0.9
599	500	0.4752	0.6934	0.4981	0.6932	49937	0.6563	0.6928	0.3	0.01	0.95
600	500	0.8861	0.2694	0.8803	0.2519	49937	0.875	0.2924	0.3	0.001	0.95
601	500	0.5848	0.6755	0.6699	0.6476	49937	0.4688	0.7266	0.3	0.0001	0.95
602	500	0.4923	0.6937	0.4923	0.6932	49937	0.5625	0.6926	0.5	0.01	0.95
603	500	0.8801	0.2821	0.9208	0.2337	49937	0.875	0.2249	0.5	0.001	0.95
604	500	0.4872	0.6928	0.5907	0.6907	49937	0.5313	0.6936	0.5	0.0001	0.95
605	500	0.4735	0.6933	0.4884	0.6932	49937	0.5313	0.693	0.7	0.01	0.95
606	500	0.8716	0.3184	0.8822	0.301	49937	0.7813	0.3402	0.7	0.001	0.95
607	500	0.5719	0.6758	0.5772	0.6827	49937	0.625	0.6771	0.7	0.0001	0.95
608	500	0.494	0.6942	0.4961	0.694	49937	0.625	0.6853	0.3	0.01	0.99
609	500	0.4606	0.6934	0.5097	0.6931	49937	0.5625	0.6935	0.3	0.001	0.99
610	500	0.7979	0.4311	0.832	0.4003	49937	0.8125	0.4333	0.3	0.0001	0.99
611	500	0.4889	0.695	0.529	0.6922	49937	0.5	0.6933	0.5	0.01	0.99
612	500	0.6173	0.6479	0.5946	0.6452	49937	0.5938	0.641	0.5	0.001	0.99
613	500	0.756	0.4974	0.7722	0.4723	49937	0.6563	0.4962	0.5	0.0001	0.99
614	500	0.5034	0.6932	0.5135	0.6928	49937	0.4688	0.6942	0.7	0.01	0.99
615	500	0.5043	0.6933	0.4903	0.6932	49937	0.4375	0.6935	0.7	0.001	0.99
616	500	0.5043	0.6931	0.5154	0.6931	49937	0.5625	0.6929	0.7	0.0001	0.99

Table 4: Model 3 (2017): Training From Scratch - 500 Epochs, 36 Runs

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum
273	14	0.506	0.6931	0.5135	0.6929	1337	0.4375	0.693	0.3	0.01	0.5
274	29	0.488	0.6932	0.5154	0.6929	2837	0.5625	0.6918	0.3	0.01	0.9
275	25	0.5017	0.6945	0.5135	0.6928	2437	0.2813	0.7041	0.3	0.01	0.95
276	25	0.4932	0.6945	0.5135	0.6937	2437	0.5	0.6954	0.3	0.01	0.99
277	24	0.4932	0.6935	0.5193	0.6929	2337	0.6875	0.6899	0.3	0.001	0.5
278	14	0.4914	0.6942	0.5019	0.693	1337	0.5313	0.6905	0.3	0.001	0.9
279	24	0.524	0.6922	0.5309	0.6899	2337	0.5313	0.6942	0.3	0.001	0.95
280	15	0.5325	0.6925	0.5116	0.6915	1437	0.4375	0.6958	0.3	0.001	0.99
281	43	0.5034	0.6935	0.4768	0.693	4237	0.6563	0.6899	0.3	0.0001	0.5
282	17	0.4872	0.6944	0.4961	0.6932	1637	0.5625	0.6986	0.3	0.0001	0.9
283	23	0.518	0.692	0.5869	0.6917	2237	0.5938	0.6899	0.3	0.0001	0.95
284	30	0.5009	0.6943	0.5077	0.6911	2937	0.5313	0.6838	0.3	0.0001	0.99
285	23	0.5223	0.6915	0.4903	0.6919	2237	0.5625	0.6862	0.5	0.01	0.5
286	36	0.5402	0.6889	0.5502	0.6811	3537	0.5625	0.6722	0.5	0.01	0.9
287	18	0.5	0.6935	0.4942	0.6934	1737	0.7188	0.6898	0.5	0.01	0.95
288	17	0.5034	0.6925	0.4865	0.6946	1637	0.5313	0.6938	0.5	0.01	0.99
289	17	0.5171	0.6925	0.5328	0.6924	1637	0.625	0.6907	0.5	0.001	0.5
290	40	0.5634	0.6905	0.6313	0.6892	3937	0.6875	0.6838	0.5	0.001	0.9
291	33	0.5291	0.6898	0.5521	0.6878	3237	0.5938	0.6888	0.5	0.001	0.95
292	22	0.4957	0.6933	0.5039	0.693	2137	0.4688	0.6936	0.5	0.001	0.99
293	19	0.4777	0.6935	0.4884	0.6937	1837	0.4063	0.6933	0.5	0.0001	0.5
294	31	0.5137	0.6927	0.5174	0.6927	3037	0.5625	0.6887	0.5	0.0001	0.9
295	34	0.5171	0.6927	0.4903	0.6931	3337	0.4063	0.6959	0.5	0.0001	0.95
296	24	0.5274	0.6921	0.5154	0.6922	2337	0.625	0.6893	0.5	0.0001	0.99
297	35	0.5111	0.6913	0.5135	0.6893	3437	0.375	0.6981	0.7	0.01	0.5
298	30	0.5068	0.6935	0.5154	0.6926	2937	0.5	0.6955	0.7	0.01	0.9
299	29	0.4846	0.6935	0.4884	0.6933	2837	0.4063	0.6937	0.7	0.01	0.95
300	15	0.5026	0.6936	0.4826	0.6944	1437	0.5	0.6927	0.7	0.01	0.99
301	18	0.5043	0.6932	0.529	0.693	1737	0.625	0.6922	0.7	0.001	0.5
302	30	0.4949	0.6937	0.4961	0.6929	2937	0.4688	0.6935	0.7	0.001	0.9
303	17	0.5043	0.693	0.5077	0.6917	1637	0.5625	0.6897	0.7	0.001	0.95
304	44	0.4991	0.6932	0.5116	0.6929	4337	0.4375	0.6968	0.7	0.001	0.99
305	27	0.5231	0.6911	0.5232	0.6921	2637	0.5	0.6925	0.7	0.0001	0.5
306	30	0.5	0.6931	0.5058	0.6927	2937	0.5938	0.6928	0.7	0.0001	0.9
307	18	0.4966	0.6931	0.473	0.6931	1737	0.5938	0.6884	0.7	0.0001	0.95
308	14	0.5137	0.6929	0.5618	0.6917	1337	0.5625	0.6901	0.7	0.0001	0.99

Table 5: Model 4 (2017): Training From Scratch - Early Stop, 36 Runs

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum
545	500	0.97	0.0754	0.973	0.1029	49937	1	0.0245	0.3	0.01	0.5
546	500	0.6447	0.6309	0.6526	0.628	49937	0.6875	0.5883	0.3	0.001	0.5
547	500	0.5599	0.6852	0.5541	0.6843	49937	0.5313	0.689	0.3	0.0001	0.5
548	500	0.9615	0.1049	0.9826	0.071	49937	0.9688	0.0923	0.5	0.01	0.5
549	500	0.5736	0.6768	0.6139	0.6635	49937	0.5313	0.7015	0.5	0.001	0.5
550	500	0.5548	0.6858	0.6486	0.6814	49937	0.4688	0.6853	0.5	0.0001	0.5
551	500	0.97	0.0784	0.973	0.1105	49937	1	0.0432	0.7	0.01	0.5
552	500	0.5719	0.6782	0.6197	0.6589	49937	0.625	0.6507	0.7	0.001	0.5
553	500	0.5231	0.6911	0.5212	0.6904	49937	0.5313	0.6886	0.7	0.0001	0.5
554	500	0.9743	0.06	0.9633	0.1316	49937	0.9688	0.0561	0.3	0.01	0.9
555	500	0.8527	0.34	0.8687	0.2946	49937	0.875	0.2591	0.3	0.001	0.9
556	500	0.6019	0.6671	0.6525	0.656	49937	0.6875	0.6079	0.3	0.0001	0.9
557	500	0.9777	0.0689	0.9768	0.0956	49937	0.9688	0.1067	0.5	0.01	0.9
558	500	0.7483	0.5149	0.749	0.5303	49937	0.75	0.4822	0.5	0.001	0.9
559	500	0.589	0.6724	0.6293	0.6654	49937	0.5313	0.7333	0.5	0.0001	0.9
560	500	0.9632	0.1122	0.9846	0.0803	49937	1	0.0199	0.7	0.01	0.9
561	500	0.7166	0.5573	0.7104	0.5646	49937	0.8125	0.4498	0.7	0.001	0.9
562	500	0.5848	0.6691	0.612	0.67	49937	0.5938	0.6624	0.7	0.0001	0.9
563	500	0.5068	0.6933	0.5058	0.6931	49937	0.5	0.6932	0.3	0.01	0.95
564	500	0.9812	0.0517	0.9688	0.0862	49937	0.9688	0.0661	0.3	0.001	0.95
565	500	0.6481	0.6344	0.6448	0.629	49937	0.7188	0.5761	0.3	0.0001	0.95
566	500	0.5017	0.6934	0.4981	0.6933	49937	0.4375	0.6954	0.5	0.01	0.95
567	500	0.9144	0.2021	0.9575	0.1222	49937	0.9063	0.1715	0.5	0.001	0.95
568	500	0.5796	0.6677	0.6564	0.6453	49937	0.4375	0.6622	0.5	0.0001	0.95
569	500	0.5094	0.6935	0.4865	0.6943	49937	0.5313	0.6917	0.7	0.01	0.95
570	500	0.9478	0.1596	0.8571	0.3058	49937	0.9375	0.1875	0.7	0.001	0.95
571	500	0.595	0.6706	0.6313	0.6599	49937	0.625	0.6683	0.7	0.0001	0.95
572	500	0.4889	0.6943	0.5232	0.6927	49937	0.5	0.6932	0.3	0.01	0.99
573	500	0.9743	0.0608	0.9595	0.1318	49937	0.9688	0.0568	0.3	0.001	0.99
574	500	0.9358	0.1833	0.9286	0.1767	49937	0.9688	0.1338	0.3	0.0001	0.99
575	500	0.5009	0.6934	0.4923	0.6934	49937	0.6563	0.6895	0.5	0.01	0.99
576	500	0.9572	0.1253	0.9537	0.1057	49937	0.9688	0.0797	0.5	0.001	0.99
577	500	0.9221	0.2301	0.9247	0.2143	49937	0.9375	0.2079	0.5	0.0001	0.99
578	500	0.4991	0.6938	0.5058	0.6933	49937	0.4063	0.6994	0.7	0.01	0.99
579	500	0.5009	0.6932	0.5116	0.6931	49937	0.5625	0.6929	0.7	0.001	0.99
580	500	0.9067	0.2556	0.8803	0.2755	49937	0.9375	0.2248	0.7	0.0001	0.99

Table 6: Model 4 (2017): Training From Scratch - 500 Epochs, 36 Runs

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum	Locked Layers
18	25	0.5094	7.9073	0.4963	8.1183	2437	0.5	8.059	0.3	0.01	0.5	5
19	17	0.5017	8.0314	0.5	8.059	1637	0.375	10.0738	0.3	0.01	0.9	5
20	33	0.4991	8.0728	0.5	8.059	3237	0.5	8.059	0.3	0.01	0.95	5
21	35	0.5026	8.0176	0.4982	8.0887	3437	0.4688	8.5627	0.3	0.01	0.99	5
22	39	0.9907	0.0313	0.9559	0.1185	3837	1	0.0093	0.3	0.001	0.5	5
23	23	0.5043	0.6932	0.5018	0.6931	2237	0.5938	0.6923	0.3	0.001	0.9	5
24	13	0.4872	0.6932	0.5	0.6931	1237	0.5938	0.6928	0.3	0.001	0.95	5
25	15	0.5	8.059	0.4982	8.0887	1437	0.4688	8.5627	0.3	0.001	0.99	5
26	50	0.9606	0.1267	0.971	0.0949	4937	0.9688	0.118	0.3	0.0001	0.5	5
27	42	0.9897	0.0307	0.9691	0.0749	4137	1	0.005	0.3	0.0001	0.9	5
28	34	0.9914	0.0369	0.9884	0.0394	3337	1	0.026	0.3	0.0001	0.95	5
29	43	0.9949	0.0162	0.9853	0.0587	4237	1	0.0094	0.3	0.0001	0.99	5
30	33	0.4897	0.6933	0.4982	0.6931	3237	0.4063	0.6932	0.3	0.01	0.5	10
31	27	0.4932	8.1694	0.4982	8.0887	2637	0.6563	5.5406	0.3	0.01	0.9	10
32	21	0.5086	7.9211	0.5018	8.0294	2037	0.4688	8.5627	0.3	0.01	0.95	10
33	27	0.5009	8.0452	0.5018	8.0294	2637	0.625	6.0443	0.3	0.01	0.99	10
34	50	0.9897	0.034	0.9865	0.0461	4937	0.9688	0.0551	0.3	0.001	0.5	10
35	13	0.4949	0.6933	0.5018	0.6933	1237	0.4063	0.6944	0.3	0.001	0.9	10
36	19	0.4949	0.6932	0.4963	0.6932	1837	0.4063	0.6934	0.3	0.001	0.95	10
37	12	0.506	7.9624	0.4981	8.0902	1137	0.5313	7.5554	0.3	0.001	0.99	10
38	45	0.9469	0.1629	0.9485	0.1404	4437	0.9063	0.2631	0.3	0.0001	0.5	10
39	50	0.9872	0.0437	0.9807	0.0801	4937	0.9688	0.0872	0.3	0.0001	0.9	10
40	47	0.982	0.0543	0.9449	0.2015	4637	0.9063	0.3238	0.3	0.0001	0.95	10
41	41	0.9906	0.0254	0.9853	0.0838	4037	1	0.0058	0.3	0.0001	0.99	10
42	23	0.4983	8.0866	0.5037	7.9998	2237	0.4688	8.5627	0.3	0.01	0.5	15
43	15	0.5017	8.0314	0.4982	8.0887	1437	0.5313	7.5554	0.3	0.01	0.9	15
44	17	0.5026	8.0176	0.4982	8.0887	1637	0.3438	10.5775	0.3	0.01	0.95	15
45	23	0.5086	7.9211	0.5	8.059	2237	0.5625	7.0517	0.3	0.01	0.99	15
46	50	0.9717	0.0774	0.9653	0.082	4937	0.9375	0.149	0.3	0.001	0.5	15
47	38	0.9906	0.0259	0.9884	0.0524	3737	1	0.0047	0.3	0.001	0.9	15
48	14	0.5017	0.6933	0.4961	0.6933	1337	0.5625	0.6912	0.3	0.001	0.95	15
49	13	0.5026	0.6934	0.5	0.6932	1237	0.5938	0.6926	0.3	0.001	0.99	15
50	50	0.8998	0.2546	0.861	0.2997	4937	0.8125	0.3556	0.3	0.0001	0.5	15
51	50	0.9503	0.1413	0.9614	0.1005	4937	0.9688	0.0774	0.3	0.0001	0.9	15
52	50	0.976	0.0706	0.9768	0.0595	4937	1	0.0327	0.3	0.0001	0.95	15
53	50	0.9795	0.0595	0.9865	0.0946	4937	1	0.007	0.3	0.0001	0.99	15
54	39	0.8775	0.2867	0.8934	0.2649	3837	0.8438	0.3678	0.3	0.01	0.5	20
55	13	0.4966	8.1142	0.4982	8.0887	1237	0.625	6.0443	0.3	0.01	0.9	20
56	31	0.8878	0.2733	0.8989	0.256	3037	0.9063	0.2592	0.3	0.01	0.95	20
57	20	0.5283	0.6861	0.5792	0.6329	1937	0.5938	0.5986	0.3	0.01	0.99	20
58	31	0.7406	0.5141	0.7813	0.4446	3037	0.7188	0.4881	0.3	0.001	0.5	20
59	50	0.8904	0.3001	0.8861	0.3118	4937	0.9063	0.2078	0.3	0.001	0.9	20
60	38	0.8904	0.2599	0.8629	0.3204	3737	1	0.0723	0.3	0.001	0.95	20
61	35	0.8896	0.2732	0.8842	0.2885	3437	0.7188	0.5589	0.3	0.001	0.99	20
62	50	0.6901	0.5892	0.7625	0.5317	4937	0.75	0.5491	0.3	0.0001	0.5	20
63	49	0.7474	0.4967	0.761	0.4931	4837	0.7188	0.6377	0.3	0.0001	0.9	20
64	39	0.7669	0.4772	0.7849	0.4571	3837	0.75	0.4386	0.3	0.0001	0.95	20
65	23	0.7868	0.4463	0.7702	0.4787	2237	0.75	0.4834	0.3	0.0001	0.99	20
66	15	0.5111	0.6937	0.5	0.6932	1437	0.4375	0.6938	0.3	0.01	0.5	25
67	15	0.4906	8.2108	0.4963	8.1183	1437	0.5938	6.548	0.3	0.01	0.9	25
68	17	0.4889	0.6943	0.4982	0.694	1637	0.4375	0.6991	0.3	0.01	0.95	25
69	13	0.5094	7.9073	0.5018	8.0294	1237	0.5625	7.0517	0.3	0.01	0.99	25
70	27	0.6721	0.5984	0.7261	0.534	2637	0.4688	0.7276	0.3	0.001	0.5	25
71	29	0.5505	0.6872	0.511	0.6842	2837	0.5313	0.6834	0.3	0.001	0.9	25
72	22	0.4983	0.6932	0.4884	0.6932	2137	0.5938	0.693	0.3	0.001	0.95	25
73	18	0.4957	0.6933	0.5097	0.6931	1737	0.375	0.6946	0.3	0.001	0.99	25
74	23	0.6507	0.6332	0.7445	0.5988	2237	0.7813	0.6119	0.3	0.0001	0.5	25
75	50	0.7295	0.5361	0.6853	0.5695	4937	0.6563	0.5262	0.3	0.0001	0.9	25
76	34	0.6918	0.567	0.5579	0.698	3337	0.7188	0.559	0.3	0.0001	0.95	25
77	47	0.6404	0.6355	0.7574	0.5638	4637	0.7813	0.489	0.3	0.0001	0.99	25
78	25	0.5034	8.0038	0.4982	8.0887	2437	0.5938	6.548	0.5	0.01	0.5	5
79	19	0.4957	8.128	0.5037	7.9998	1837	0.4375	9.0664	0.5	0.01	0.9	5
80	13	0.5017	8.0314	0.5037	7.9998	1237	0.5938	6.548	0.5	0.01	0.95	5
81	15	0.506	7.9624	0.5	8.059	1437	0.625	6.0443	0.5	0.01	0.99	5
82	19	0.5197	0.693	0.4963	0.694	1837	0.5625	0.6932	0.5	0.001	0.5	5
83	13	0.5154	0.6931	0.5	0.6931	1237	0.4063	0.6943	0.5	0.001	0.9	5
84	15	0.4889	0.6935	0.5018	0.6931	1437	0.5313	0.6931	0.5	0.001	0.95	5
85	27	0.488	0.6937	0.5037	0.6931	2637	0.4688	0.6943	0.5	0.001	0.99	5
86	45	0.9478	0.1428	0.9761	0.074	4437	1	0.036	0.5	0.0001	0.5	5
87	35	0.9803	0.0564	0.9706	0.0886	3437	0.9688	0.0536	0.5	0.0001	0.9	5
88	48	0.9889	0.0353	0.973	0.1092	4737	1	0.0362	0.5	0.0001	0.95	5
89	41	0.9932	0.0227	0.989	0.0572	4037	1	0.0015	0.5	0.0001	0.99	5
90	17	0.5034	0.6929	0.5055	0.6931	1637	0.5	0.6927	0.5	0.01	0.5	10
91	21	0.4991	8.0728	0.4963	8.1183	2037	0.5625	7.0517	0.5	0.01	0.9	10
92	29	0.4991	8.0728	0.5037	7.9998	2837	0.4375	9.0664	0.5	0.01	0.95	10
93	27	0.518	7.7693	0.4982	8.0887	2637	0.4688	8.5627	0.5	0.01	0.99	10
94	50	0.9829	0.0487	0.9826	0.0659	4937	0.9688	0.0452	0.5	0.001	0.5	10
95	20	0.4829	0.6934	0.5097	0.693	1937	0.4688	0.6947	0.5	0.001	0.9	10
96	31	0.4957	0.6933	0.5018	0.6931	3037	0.5938	0.6931	0.5	0.001	0.95	10
97	13	0.5017	0.6936	0.4982	0.6937	1237	0.5	0.693	0.5	0.001	0.99	10
98	50	0.9221	0.2094	0.8764	0.3113	4937	0.9688	0.1449	0.5	0.0001	0.5	10
99	50	0.9769	0.0643	0.9826	0.1122	4937	1	0.0235	0.5	0.0001	0.9	10
100	39	0.9789	0.0613	0.9743	0.0876	3837	1	0.0253	0.5	0.0001	0.95	10
101	26	0.9914	0.0419	0.9768	0.1003	2537	1	0.0142	0.5	0.0001	0.99	10
102	13	0.476	0.6933	0.4963	0.6932	1237	0.4688	0.6935	0.5	0.01	0.5	15
103	13	0.4906	0.6938	0.5	0.6933	1237	0.5313	0.6922	0.5	0.01	0.9	15
104	17	0.4846	8.3074	0.4963	8.1183	1637	0.375	10.0738	0.5	0.01	0.95	15
105	29	0.5051	7.9762	0.5	8.059	2837	0.5	8.059	0.5	0.01	0.99	15
106	42	0.9769	0.0804	0.9575	0.1099	4137	1	0.0299	0.5	0.001	0.5	15
107	23	0.4906	0.6951	0.5	0.6933	2237	0.4063	0.7016	0.5	0.001	0.9	15
108	29	0.4914	0.6932	0.5	0.6931	2837	0.5313	0.6931	0.5	0.001	0.95	15
109	21	0.5043	0.693	0.5018	0.6932	2037	0.375	0.6978	0.5	0.001	0.99	15
110	50	0.8904	0.2776	0.917	0.2088	4937	0.9375	0.1887	0.5	0.0001	0.5	15

Table 7: Model VGG (2017): Transfer Learning VGG19 - Early Stop, 180 Runs (Part 1)

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Dropout	Learning Rate	Momentum	Locked Layers
111	50	0.9538	0.1225	0.9517	0.1323	4937	0.9375	0.1401	0.5	0.0001	0.9	15
112	50	0.9786	0.07	0.971	0.071	4937	1	0.0467	0.5	0.0001	0.95	15
113	50	0.9803	0.0583	0.9595	0.0836	4937	0.9688	0.0788	0.5	0.0001	0.99	15
114	50	0.8844	0.2835	0.888	0.2959	4937	0.8438	0.3239	0.5	0.01	0.5	20
115	30	0.8904	0.2626	0.8803	0.2634	2937	0.8438	0.3101	0.5	0.01	0.9	20
116	29	0.8998	0.2532	0.8695	0.3081	2837	0.8438	0.2854	0.5	0.01	0.95	20
117	18	0.53	0.6806	0.473	0.7035	1737	0.375	0.7296	0.5	0.01	0.99	20
118	29	0.7354	0.5328	0.7868	0.4422	2837	0.7188	0.5007	0.5	0.001	0.5	20
119	48	0.8562	0.3362	0.8629	0.3128	4737	0.7813	0.5222	0.5	0.001	0.9	20
120	44	0.8853	0.29	0.8938	0.2759	4337	0.9063	0.2364	0.5	0.001	0.95	20
121	31	0.8801	0.2961	0.8915	0.2969	3037	0.9063	0.3113	0.5	0.001	0.99	20
122	31	0.6464	0.624	0.7261	0.574	3037	0.75	0.609	0.5	0.0001	0.5	20
123	50	0.7389	0.5249	0.7876	0.4273	4937	0.6563	0.5747	0.5	0.0001	0.9	20
124	50	0.762	0.4823	0.8282	0.401	4937	0.7813	0.5029	0.5	0.0001	0.95	20
125	50	0.8759	0.3078	0.8417	0.3763	4937	0.9063	0.201	0.5	0.0001	0.99	20
126	21	0.5188	0.6933	0.5037	0.6931	2037	0.5625	0.6916	0.5	0.01	0.5	25
127	23	0.5111	7.8797	0.4982	8.0887	2237	0.5625	7.0517	0.5	0.01	0.9	25
128	19	0.5034	8.0038	0.5	8.059	1837	0.5625	7.0517	0.5	0.01	0.95	25
129	17	0.5017	8.0314	0.5018	8.0294	1637	0.5	8.059	0.5	0.01	0.99	25
130	45	0.6832	0.593	0.7004	0.559	4437	0.5625	0.6172	0.5	0.001	0.5	25
131	22	0.5351	0.691	0.4749	0.6934	2137	0.5625	0.6854	0.5	0.001	0.9	25
132	37	0.5	0.6932	0.5018	0.6931	3637	0.5313	0.693	0.5	0.001	0.95	25
133	19	0.4974	0.6936	0.5018	0.6935	1837	0.5	0.6936	0.5	0.001	0.99	25
134	27	0.6378	0.6345	0.7518	0.5937	2637	0.625	0.6165	0.5	0.0001	0.5	25
135	49	0.6961	0.5771	0.7518	0.4779	4837	0.6875	0.5126	0.5	0.0001	0.9	25
136	15	0.637	0.64	0.7426	0.6032	1437	0.75	0.5781	0.5	0.0001	0.95	25
137	43	0.4923	0.6631	0.5037	0.6516	4237	0.4688	0.6491	0.5	0.0001	0.99	25
138	12	0.5068	7.9486	0.4923	8.1835	1137	0.4375	9.0664	0.7	0.01	0.5	5
139	13	0.5086	7.9211	0.5	8.059	1237	0.625	6.0443	0.7	0.01	0.9	5
140	23	0.5043	7.99	0.5018	8.0294	2237	0.3125	11.0812	0.7	0.01	0.95	5
141	25	0.488	8.2522	0.5018	8.0294	2437	0.5	8.059	0.7	0.01	0.99	5
142	16	0.5873	0.6795	0.8166	0.5979	1537	0.7188	0.657	0.7	0.001	0.5	5
143	24	0.4966	0.6937	0.5251	0.6919	2337	0.4375	0.6953	0.7	0.001	0.9	5
144	17	0.5017	0.6932	0.5018	0.6931	1637	0.5938	0.6924	0.7	0.001	0.95	5
145	13	0.4991	8.0728	0.5	8.059	1237	0.5938	6.548	0.7	0.001	0.99	5
146	50	0.9486	0.1573	0.9672	0.109	4937	0.875	0.3157	0.7	0.0001	0.5	5
147	45	0.9837	0.0498	0.9835	0.0717	4437	1	0.0074	0.7	0.0001	0.9	5
148	50	0.994	0.0157	0.9903	0.044	4937	1	0.0029	0.7	0.0001	0.95	5
149	25	0.4777	0.6932	0.4963	0.6932	2437	0.5	0.693	0.7	0.0001	0.99	5
150	13	0.5009	8.0452	0.5	8.059	1237	0.5938	6.548	0.7	0.01	0.5	10
151	39	0.5008	8.0454	0.5	8.059	3837	0.4375	9.0664	0.7	0.01	0.9	10
152	19	0.5	8.059	0.4982	8.0887	1837	0.5	8.059	0.7	0.01	0.95	10
153	17	0.5137	7.8383	0.4963	8.1183	1637	0.4375	9.0664	0.7	0.01	0.99	10
154	18	0.5128	0.6936	0.4961	0.6939	1737	0.5938	0.682	0.7	0.001	0.5	10
155	15	0.4923	0.693	0.5037	0.6932	1437	0.5313	0.6899	0.7	0.001	0.9	10
156	27	0.4803	0.6936	0.5	0.6932	2637	0.5625	0.6926	0.7	0.001	0.95	10
157	13	0.5009	0.6933	0.5018	0.6931	1237	0.6875	0.6859	0.7	0.001	0.99	10
158	50	0.9204	0.2157	0.8919	0.2414	4937	0.9063	0.1983	0.7	0.0001	0.5	10
159	35	0.97	0.0989	0.9283	0.1927	3437	1	0.0322	0.7	0.0001	0.9	10
160	48	0.9889	0.0454	0.9788	0.0807	4737	0.9688	0.0715	0.7	0.0001	0.95	10
161	17	0.482	0.6932	0.4982	0.6932	1637	0.625	0.693	0.7	0.0001	0.99	10
162	15	0.476	0.6935	0.4982	0.6931	1437	0.3438	0.6937	0.7	0.01	0.5	15
163	35	0.4872	0.6935	0.4963	0.6934	3437	0.625	0.6894	0.7	0.01	0.9	15
164	21	0.506	7.9624	0.5037	7.9998	2037	0.4063	9.5701	0.7	0.01	0.95	15
165	23	0.4991	8.0728	0.5018	8.0294	2237	0.625	6.0443	0.7	0.01	0.99	15
166	41	0.9598	0.1155	0.9504	0.1072	4037	0.9688	0.0679	0.7	0.001	0.5	15
167	39	0.5583	0.6865	0.5864	0.6808	3837	0.625	0.6663	0.7	0.001	0.9	15
309	21	0.4949	0.6931	0.5	0.6932	2037	0.5	0.6931	0.7	0.001	0.95	15
169	15	0.4957	8.128	0.5	8.059	1437	0.4375	9.0664	0.7	0.001	0.99	15
310	50	0.8656	0.3303	0.9266	0.2017	4937	0.9688	0.2025	0.7	0.0001	0.5	15
171	50	0.9666	0.1086	0.9498	0.1333	4937	1	0.0616	0.7	0.0001	0.9	15
172	44	0.9521	0.1409	0.9479	0.1083	4337	1	0.0412	0.7	0.0001	0.95	15
173	41	0.9854	0.0468	0.9779	0.0825	4037	1	0.003	0.7	0.0001	0.99	15
174	50	0.8913	0.2629	0.8803	0.2784	4937	0.9375	0.1833	0.7	0.01	0.5	20
175	27	0.881	0.2794	0.8088	0.3835	2637	0.7813	0.3362	0.7	0.01	0.9	20
176	23	0.8853	0.3054	0.8879	0.2852	2237	0.9375	0.1906	0.7	0.01	0.95	20
177	19	0.4897	8.2246	0.5018	8.0294	1837	0.4688	8.5627	0.7	0.01	0.99	20
178	21	0.613	0.6577	0.7518	0.5947	2037	0.4375	0.706	0.7	0.001	0.5	20
179	39	0.8024	0.4249	0.8401	0.3451	3837	0.5938	0.6532	0.7	0.001	0.9	20
180	41	0.899	0.2892	0.8952	0.2752	4037	0.9375	0.2291	0.7	0.001	0.95	20
181	33	0.9024	0.279	0.8676	0.3145	3237	0.8438	0.4069	0.7	0.001	0.99	20
182	33	0.637	0.6424	0.7151	0.6133	3237	0.6563	0.6184	0.7	0.0001	0.5	20
183	18	0.595	0.6679	0.6486	0.6329	1737	0.4688	0.7407	0.7	0.0001	0.9	20
184	39	0.6943	0.5857	0.7555	0.5232	3837	0.5938	0.641	0.7	0.0001	0.95	20
185	50	0.8442	0.3553	0.8031	0.39	4937	0.8438	0.3014	0.7	0.0001	0.99	20
186	17	0.5017	8.0314	0.5	8.059	1637	0.5	8.059	0.7	0.01	0.5	25
187	33	0.5103	7.8935	0.4963	8.1183	3237	0.4063	9.5701	0.7	0.01	0.9	25
188	19	0.5017	8.0314	0.4963	8.1183	1837	0.4375	9.0664	0.7	0.01	0.95	25
189	15	0.4914	8.197	0.5018	8.0294	1437	0.5313	7.5554	0.7	0.01	0.99	25
190	19	0.5205	0.693	0.5588	0.6674	1837	0.2813	0.7487	0.7	0.001	0.5	25
191	23	0.5034	0.6932	0.4982	0.6931	2237	0.5313	0.6929	0.7	0.001	0.9	25
192	23	0.4957	0.6933	0.5129	0.6932	2237	0.4063	0.6943	0.7	0.001	0.95	25
193	15	0.494	0.6942	0.5	0.6933	1437	0.5625	0.691	0.7	0.001	0.99	25
194	50	0.6147	0.6544	0.7452	0.5902	4937	0.6563	0.6319	0.7	0.0001	0.5	25
195	27	0.5976	0.6597	0.7224	0.6177	2637	0.5938	0.6721	0.7	0.0001	0.9	25
196	14	0.5205	0.702	0.5676	0.6836	1337	0.5625	0.6814	0.7	0.0001	0.95	25
197	12	0.4837	0.7	0.4807	0.6935	1137	0.5	0.6974	0.7	0.0001	0.99	25

Table 8: Model VGG (2017): Transfer Learning VGG19 - Early Stop, 180 Runs (Part 2)

Stellenbosch University https://scholar.sun.ac.za

Execution	Epoch	Epoch Acc	Epoch Loss	Epoch Val Acc	Epoch Val Loss	Batch	Batch Acc	Batch Loss	Learning Rate	Momentum
353	45	0.9983	0.0046	0.9835	0.1127	4437	1	0.0054	0.01	0.5
354	22	0.5043	7.99	0.5	8.059	2137	0.4375	9.0664	0.001	0.5
355	20	0.4932	8.1694	0.4982	8.0887	1937	0.5	8.059	0.0001	0.5
356	14	0.512	7.8659	0.5018	8.0294	1337	0.625	6.0443	0.01	0.9
357	45	0.9966	0.0115	0.9761	0.1304	4437	0.9688	0.0388	0.001	0.9
358	12	0.5103	7.8935	0.5037	7.9998	1137	0.5938	6.548	0.0001	0.9
359	31	0.9991	0.0025	0.9903	0.0954	3037	1	0	0.01	0.95
360	50	0.9923	0.0205	0.9761	0.0904	4937	1	0.0026	0.001	0.95
361	22	0.506	7.9624	0.5037	7.9998	2137	0.4375	9.0664	0.0001	0.95
362	18	0.5009	8.0452	0.5018	8.0294	1737	0.5625	7.0517	0.01	0.99
363	22	0.5068	7.9486	0.5018	8.0294	2137	0.6563	5.5406	0.001	0.99
364	31	0.9974	0.0063	0.9761	0.0899	3037	1	0.0012	0.0001	0.99

Table 9: Model Inception (2017): Transfer Learning InceptionV3 - Early Stop, 12 Runs