University of Cape Town

Masters Thesis

Investigating automated bird detection from webcams using machine learning

Author:
Alex Mirugwe

Supervisor:
Dr. Juwa Nyirenda
Co-Supervisor:
Dr. Emmanuel Dufourq

A half-dissertation submitted in fulfilment of the requirements

for the degree of Masters of Science

in the

Department of Statistical Sciences
Faculty of Science

January 19, 2022

Declaration of Authorship

I, Alex MIRUGWE, declare that this thesis titled, “Investigating automated bird detection from webcams using machine learning” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date:

Abstract

One of the most challenging problems faced by ecologists and other biological researchers today is to analyze the massive amounts of data being collected by advanced monitoring systems such as camera traps, wireless sensor networks, high-frequency radio trackers, global positioning systems, and satellite tracking systems being used today. It has become expensive, laborious, and time-consuming to analyze the large datasets using manual and traditional statistical techniques. Recent developments in the field of deep learning are showing promising results towards automating the analysis of these extremely large datasets.

The primary objective of this study is to test the capabilities of the state-of-the-art deep learning architectures to detect birds in the webcam captured images. A total of 10592 images were collected for this study from the Cornell Lab of Ornithology live stream feeds situated in six unique locations in United States, Ecuador, New Zealand, and Panama. To achieve the main objective of the study, two convolutional neural network object detection meta-architectures, single-shot detector (SSD) and Faster R-CNN in combination with MobileNet-V2, ResNet50, ResNet101, ResNet152, and Inception ResNet-V2 feature extractors were studied and evaluated. Through the use of transfer learning, all the models were initialized using weights pre-trained on the MS COCO (Microsoft Common Objects in Context) dataset provided by the TensorFlow 2 object detection API.

The Faster R-CNN model coupled with ResNet152 outperformed all other models with a mean average precision of 92.3%. However, the SSD model with the MobileNet-V2 feature extraction network achieved the lowest inference time (110ms) and the smallest memory capacity (30.5MB) compared to its counterparts. The outstanding results achieved in this study confirm that deep learning-based algorithms are capable of detecting birds of different sizes in different environments and the best model could potentially help ecologists in monitoring and identifying birds from other species present in the environment.

Acknowledgements

First and foremost, I thank the almighty God for the mercy, strength, and blessings throughout my research period.

I would like to express my sincere gratitude to my supervisors, Dr Juwa Nyirenda and Dr Emmanuel Dufourq for their noble support, guidance, patience, encouragement, and immense knowledge. This work would not have been achievable without your insightful feedback and advice. It is immeasurable how much I have learnt from both of you and your generous support will never be forgotten.

Very special thanks to Dr Emmanuel Dufourq for introducing me to the world of machine learning in ecology. Your guidance and support were very instrumental in the formulation of the research topic and questions and the acquisition of data used in this study.

I acknowledge the generous scholarship that I received from the Mastercard Foundation Scholar's Program at the University of Cape Town. This accomplishment would not have been possible without this financial support.

Finally, I would like to thank my family and friends for their continuous encouragement and moral support throughout my years of study. I love you all!

This dissertation is dedicated to my late father.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**Network
ILSVRC	**I**magenet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
mAP	**m**ean **A**verage **P**recision
MSCOCO	**M**icrosoft **C**ommon **O**bjects in **C**ontext
NMS	**N**on-**m**aximum **S**uppression
RCNN	**R**egion **B**ased **C**onvolutional **N**eural **N**etworks
ReLU	**Re**ctified **L**inear **U**nit
ResNet	**Res**idual **Net**works
SGD	**S**tochastic **G**radient **D**escent
SSD	**S**ingle **S**hot **D**etector
SVM	**S**upport **V**ector **M**achine
VGGNet	**V**isual **G**eometry **G**roup **Net**works

Chapter 1

Introduction

This chapter discusses the background of the research problem. Furthermore, it elaborates the motivation and the purpose of this research.

1.1 Problem definition and motivation

In the world of ecosystem preservation, domestic and wildlife animal monitoring and identification are very important areas of research as they help ecologists and conservation practitioners to: 1) monitor different species especially animals on the verge of extinction, 2) understand species abundances, 3) better understand the effect of the environment on wildlife (Anthony et al., 2015), 4) formulate conversation and management policies (Hung et al., 2017). More than ever, improvement in the animal monitoring systems/methods are needed if we are to preserve the existing wildlife from the increasing threat of climate change, human-animal poaching, and resource acquisition (Wilber et al., 2013; Root and Schneider, 2002; Villa et al., 2017).

Due to the inefficiency of the traditional wildlife monitoring techniques, several modern tools have been developed such as motion sensor camera traps (Kays et al., 2010), wireless sensor networks (Akyildiz et al., 2002; Szewczyk et al., 2004), high-frequency radio trackers (Gary and Robert, 2012), and the global positioning system (GPS) and satellite tracking systems (Godley et al., 2008). These observational technological advancements have enhanced the ability to obtain massive, long-term, cross-scale, and heterogeneous data (Guo et al., 2020). It is also helping ecologists to document all aspects of wildlife such as feeding, movement, sleeping, and interaction with one another, something hard to be done through physical human monitoring (O'Connell et al., 2010). In some cases, it is dangerous or even impossible for humans to physically monitor certain species. For example, in 2017, a wildlife ecologist known as *Krisztian Gyongyi* was attacked and killed by a rhino while he was tracking animals in *Akagera National Park* in Rwanda[1]. Therefore using automated tools can

[1]For more information about the story: https://news.mongabay.com/2017/06

be very helpful to collect data on such animals.

But these new wildlife monitoring technologies have resulted in huge data sets that have greatly outpaced the traditional manual analytical techniques as they are costly, labour intensive, and time-consuming (Schneider et al., 2018; Norouzzadeh et al., 2018; Akçay et al., 2020). Even the traditional machine learning tools such as Support Vector Machine (SVM), random forest, Linear Discriminant Analysis (LDA), K-nearest neighbor (KNN), and Principal Component Analysis (PCA) are not suitable because they quickly saturate whenever the data volume increases (Moniruzzaman et al., 2017). For example, the 225 camera traps deployed by *Snapshot Serengeti* [2] camera survey project across an area of 1125 km² in the Tanzania's *Serengeti National Park* collected 1.2 million image sets, each containing 1-to-3 images in a space of 3 years (Swanson et al., 2015). To understand how time-consuming and labour intensive this manual process could be, it took a team of 28000 and approximately 40000 registered and unregistered users respectively to annotate a 6-month batch of the Serengeti dataset (Hung et al., 2017). This observation justifies the need to automate the process of image annotation, species identification, and monitoring.

The most widely used automation technique for image classification has been deep learning since 2012 when it broke accuracy records in ImageNet classification challenge (Krizhevsky et al., 2012) and speech recognition (Hinton et al., 2012). Deep learning has continued to obtain tremendous success in several fields including ecology. For example, Barré et al. (2017) used field-photographed leaves to develop an automatic plant species identification deep learning model which obtained an average classification accuracy of 97.8% in the top-5. Several studies have also used animal sounds to build models for identifying and monitoring different species (Dufourq et al., 2021; Knight et al., 2017; Lee et al., 2006). Deep learning techniques have also been applied in identifying and counting several species in camera-trap images (Villa et al., 2017; Norouzzadeh et al., 2018; Christin et al., 2019). A study by Ditria et al. (2020) compared the identification speed and accuracy of deep learning methods against marine experts and citizen scientists in determining fish abundance in image and video underwater captured data and found that the deep learning algorithm performance was 7.1% and 13.4% better than experts and citizen scientists respectively for the image dataset, and 1.5% and 7.8% better for the video dataset. As with other wild species, monitoring birds is also a regular ecological activity. In this thesis, we propose to study and evaluate state-of-the-art deep learning architectures to detect birds in webcam captured images. The main focus is to compare how well these architectures can detect birds in images captured from different environments in addition to evaluating their speed and memory consumption. This research focused on the use of Convolution Neural Networks (LeCun and Bengio, 1995) because they have obtained better accuracy in animal identification tasks compared to other methods as discussed in chapter 2. Apart from ecological research and avian protection, bird detection is also important in multiple other applications such as wind energy farms where detection systems are needed to prevent the collision of

[2] https://www.zooniverse.org/projects/zooniverse/snapshot-serengeti

birds with wind turbines and aviation safety. In aviation, machine learning-based systems are used in differentiating radar signals of birds from abiotic objects (Major et al., 2019; Rosa et al., 2016).

1.2 Research Objective

The main objective of this research is to study, train, and evaluate state-of-the-art deep learning object detection algorithms that are capable of detecting birds in webcam-captured images. The success of this research will make it easier to develop real-time bird monitoring and detection models that use webcams. To achieve the primary objective, the study objectives below are identified:

1. Collecting and annotating webcam data for training, evaluating, and testing the algorithms. Providing a dataset of annotated images is another important contribution of this study.
2. Identifying the best deep learning algorithms suitable for detecting birds of different sizes in several environment settings.
3. Evaluating the detection performance based on speed and accuracy of the selected algorithms.
4. Provide a comparison performance between these algorithms and conclude on which is the best method for speed and accuracy.

1.3 Thesis Overview

The first chapter discusses the problem background, motivation, and purposes of the study. The rest of the chapters are organized as follows:

Chapter 2: Presents the review of related studies that are used in this study. The purpose of this discussion is to help us understand the existing studies, their successes, and research gaps.

Chapter 3: We describe the main concepts of this study and provide theory on machine learning, artificial neural networks, deep learning with a main focus on convolutional neural networks.

Chapter 4: Discusses the state-of-the-art object detection meta-architectures and feature extraction networks used in this study.

Chapter 5: Introduces the dataset, pre-processing techniques, evaluation metrics, Tensorflow object detection API used throughout this study. It also presents the implementation details of the models used.

Chapter 6: Presents a detailed analysis of the models' performance and a comparison among each other.

Chapter 7: Gives a summary of the study and also offers some suggestions for future research.

Chapter 2

Literature Review

Over the years, wildlife population, movement, and their interaction with the environment were being monitored using manual counting techniques such as total ground counts, line transects, dropping count technique, aerial count, and point count methods (Akçay et al., 2020; Merz, 1986; Hong et al., 2019).

The total ground counts method is more suitable in counting relatively large species such as elephants, antelopes, and rhinos because it uses individual animal recognition in a certain population (Hugo, 2002; Hong et al., 2019). The method is not suitable for small species like birds nor can it be used to count aquatic animals such as crocodiles unless they are found on the river/or lake banks. And the other disadvantages of the method are that it is labour intensive and time-consuming and therefore very expensive to use (Hugo, 2012).

The line transect technique estimates the abundance of the animal population through the use of a distance sampling method whereby randomly placed lines in a study region are traversed by the observer while measuring the distance of the animal from the drawn perpendicular line (Glennie et al., 2015). The method assumes that all animals in a line are certainly detected and when this is not met, errors and biases arise (Hong et al., 2019).

The dropping count technique is an approach that estimates the number of a given species by using the excrement left behind (Hong et al., 2019). In a study by Merz (1986) which aimed at counting the number of elephants in the Tai National Park used the elephant dropping to execute the task. The elephant droppings were defined as five or eight dung-ball deposits, but during the counting process, error arose from mistaking hippos excretions for elephants.

As discussed earlier in the introduction, these traditional methods are no longer suitable for analysing the large amount of data collected today which is characterised by velocity, volume, value, variety and veracity (Marr, 2015). There have been several attempts to use machine learning to automate the process of identifying and detecting animals and birds in datasets of audio and video recordings, and camera-trap

imagery data. The most popular architectures used in identifying and detecting objects are convolutional neural networks (CNNs) (Song et al., 2011). CNNs are deep learning algorithms that are mostly applied to analyze imagery data and they work similarly as artificial neural networks except they have a series of convolutional layers at the beginning (Aggarwal, 2018). They are designed to automatically and adaptively learn visual abstracts through multiple building blocks such as convolution layers, pool layers, and fully connected layers, for more details see Section 3.4. Object detection applications are mainly designed using CNN based algorithms such as Faster R-CNN, YOLO, and Single shot Detector meta-architectures which are discussed in Section 4.1. Object detection neural networks are powering advanced systems such as self-driving cars - to figure out where cars, pedestrians, and other objects are and navigate around them.

Schneider et al. (2018) trained and compared the performance of two object detection methods; that is, YOLO and Faster R-CNN for identifying and localizing multiple animal species in the Reconyx (Norouzzadeh et al., 2018) and snapshot Serengeti (Swanson et al., 2015) camera-trap image datasets with a composition of 946 and 4096 labelled images respectively. The ResNet-101 architecture was used for both techniques because of its tremendous success in other camera trap image research (Villa et al., 2017). An adaptive momentum optimizer was used to train both models, and after three successive epochs, training was stopped since there was no further loss improvement. The Faster R-CNN model registered better performance with an accuracy of 93% and 76.7% on the Reconyx and snapshot Serengeti datasets respectively, while the YOLO model achieved 73% and 40.3% accuracy. For better performance, deep learning methods require huge amounts of data (Yann et al., 2015; Jason and Luis, 2017) and therefore YOLO's poor performance might have been a result of fewer images used to train the model.

In another study, Hong et al. (2019) used unmanned aerial vehicle collected photographs to construct deep learning bird detection models that included: Single shot multibox Detector Region-based Fully Convolutional Network (R-FCN), single-shot multibox Detector (SSD), Faster R-CNN, YOLO, and Retinanet. The performance of these models was evaluated and compared using their average precision and computing speed. A total of 21914 images were used to train the models, while a total of 3874 and 3513 images were used to validate and test the models respectively. Hong et al. used a variety of architectures to design the models. For example, the Faster R-CNN meta-architecture was combined with ResNet-101 and Inception v2 base networks which returned an accuracy of 95.23% and 93.94% and computing speed of 95ms and 82ms per test image respectively. Retinanet and R-FCN models used Resnet-101 and Resnet-50 architectures respectively. Retinanet returned an accuracy of 83.70% in 75ms and R-FCN produced 84.12% in 87ms. SSD returned an accuracy of 65.20% in 23ms with the mobilenet v2 architecture while the YOLO-v3 used the darknet-53 architecture and produced an accuracy of 90.77% in 41ms. Studies carried out by Schneider et al. (2018) and Hong et al. (2019) both showed that the use of the Faster R-CNN combined with the ResNet-101 architecture pro-

vided the most accurate results. The research of Hong et al. (2019) however, found that the ability of the model to detect birds decreases as the bird's image pixel size falls to smaller than 40×40.

Akçay et al. (2020) used on-ground geo-tagged bird photographs to automate the bird detection and counting process using deep neural networks. In their study, Akçay et al. (2020) used Faster R-CNN architecture to design a generic bird detection model which could be used to match every image pixel to the corresponding output bounding boxes that contain birds. The Faster R-CNN was combined with two feature extraction methods, that is: SSD and Bags of Words (BoW). The latter is an object recognition technique that extracts local features from the input image. A histogram of these features is constructed and fed into an SVM which classifies the features into an object or background (Koniusz et al., 2016; Lowe and David, 1999). The SSD method is a feed-forward based convolution network that produces scores and bounding boxes of fixed size. And this is followed by a non-maximum suppression stage that predicts the presence of a bird in the image (Wei et al., 2016). The model trained using Faster R-CNN meta-architecture outperformed other methods in terms of bird detection accuracy, registering a precision of 94% and recall of 95% compared to BoW with 86% and 66%, and SSD with 88% and 95%. However, SSD was faster in terms of testing time since it is a single-stage method (Jang et al., 2020). The Faster R-CNN registered higher error rates for images with pixels less than 1500 and those greater than 60000. But despite the successful detection accuracy registered by Akçay et al. (2020), the model's ability to give the same accuracy when used on other datasets with birds in various environments like overlapping birds is uncertain since it was trained and evaluated on very small datasets of 491 and 156 images respectively.

A convolutional neural network built on a systematic feature learning-based classification method was used by Boudaoud et al. (2019) to design an automatic bird detection and counting model on aerial images of the ocean. They trained their model on 9881 images, validated, and tested it on 2471 and 768 images of the birds and sea species respectively. Bilinear interpolation technique was used to resize all images to 128 x 128 pixels. Bilinear interpolation is a technique used to replace missing image pixels with a weighted average of the 4-nearest pixels on the boundary (Sa, 2014). Boudaoud et al. used a CNN with seven convolutional layers, one and two flattered and fully connected layer(s) respectively. The model was trained at a learning rate of 0.0001 with batch sizes of 64. After 64 epochs, the model registered the best accuracy of 95.3%. Unlike the methods of Hong et al. (2019) and Akçay et al. (2020), Boudaoud et al. (2019) proposed method achieved good performance on low pixel images and therefore, their implementation will be followed closely since some of our images are of low resolution.

Takeki et al. (2016) worked on an investigation to design a CNN with semantic segmentation to find small birds in large image backgrounds. They proposed a three pipeline method based on the ResNet architecture as follows: 1) CNN-based detector with capabilities of background subtraction preprocess, 2) fully convolutional net-

works (FCNs) that works both as a detector and semantic segmentation, and 3) the SuperParse based semantic methods. The results of these methods were combined using the SVMs which resulted in higher bird detection performance. These models were trained on 77 images and evaluated on 44 images. Each of these pipelines produced a score or a class-wise likelihood. FCNs and SuperParsing generated pixel-wise class likelihoods while CNN generated bounding box-wise scores of class likelihoods. The models were evaluated using the precision, recall, and F-measure evaluation metrics which were calculated using true positives and false-positive counted birds. CNNs achieved the highest recall rate of 0.902 and the SuperParsing model registered the best precision rate of 1.00. An important difference of this study of Takeki et al. (2016) from that of Boudaoud et al. (2019), Akçay et al. (2020), Schneider et al. (2018), and Hong et al. (2019) is that they designed models tailored to small birds which was one of the weaknesses of the earlier studies.

Although numerous studies have been carried out on the application of deep learning techniques to automate the process of bird detection and counting (Schneider et al., 2018; Liu et al., 2018; Akçay et al., 2020; Boudaoud et al., 2019; Takeki et al., 2016), the studies have mainly focused on the use of satellite captured, camera-trap, or unmanned aerial vehicle images. This study will use webcam based photographs. To the best of our knowledge this is the first study to use webcam-based photographs. This study will use the methods as Hong et al. (2019), but the study will result in a new dataset. The newly collected images will be different to that of Hong et al. where that study used fewer images. The proposed approach also provides a broader comparison between several deep learning architectures unlike Hong et al. (2019) who studied only three architectures, ResNet-101, Inception V2, and MobileNet v2.

Chapter 3

Theoretical Background

This chapter discusses the theoretical concepts required to understand the methods used throughout this research. To lay the ground for the understanding of these methods, we provide details about machine learning, artificial neural networks, convolutional neural networks, activation functions, regularization, optimization, and object detectors.

3.1 Machine Learning

Machine learning refers to a sub-domain of artificial intelligence that enables the building of computational methods using data, which gives the computer the ability to automatically learn on their own from data and make accurate predictions (Mohri et al., 2018). The increase in data generated at workplaces necessitated more computational power, which in turn gave birth to the development of statistical models to extract insights from these large data sets. Unlike traditional statistical methods, machine learning algorithms have the capabilities of modelling highly dimensional and non-linear data with intense interaction effects (Glenn and Katharina, 2000; Goldberg and Henry, 1988; Knudby et al., 2010). The algorithms are categorized according to the nature of data used and the type of supervision received during training. These categories are supervised, unsupervised, and reinforcement learning.

3.1.1 Supervised Learning

Supervised learning refers to a machine learning sub-category used to train computational models using labelled data to predict or classify certain outcomes. The model is trained by tuning parameters until it detects the hidden patterns between the input variables and the targeted output variable. The main objective of supervised learning is to train a model $y = f(x)$, where we predict output variable, y (also known as a dependent variable) based on input variables, x (independent variables). Supervised learning algorithms are categorized into two types: classification

and regression.

Classification: Algorithms used in modeling when the output variable is categorical (Alpaydin, 2020). It approximates a mapping function from independent variables, x to a discrete dependent variable, y. These algorithms include logistic regression (Scott, 2002),decision trees and random forests (Ali et al., 2012), support vector machines (William, 2006), neural networks (Sarle, 1994), and k-nearest neighbors (Kramer, 2013).

Regression: Algorithms used in modeling when the output variable is continuous (Alpaydin, 2020). The main focus of a regression problem is to determine the relationship between independent features and dependent continuous feature. Some of the regression models include linear regression (George and Alan, 2012) and polynomial regression (Ostertagová, 2012).

3.1.2 Unsupervised Learning

Unsupervised learning is a machine learning sub-domain used in the analysis and clustering of unlabeled data sets provided with the input variable (x), but the output variable is unknown. These types of algorithms are capable of detecting hidden patterns or groupings in the data without human intervention. Unsupervised learning algorithms are mainly used for clustering, anomaly detection, dimensional reduction, and association (Barlow, 1989). The table below shows some of the common unsupervised learning algorithms.

Task	Algorithm
Clustering (Likas et al., 2003)	– K-Means – DBSCAN – Hierarchical Cluster Analysis (HCA)
Anomaly detection (Markus and Seiichi, 2016)	– One-class SVM – Isolation Forest
Dimensionality reduction (Van-Der-Maaten et al., 2009)	– Principal Component Analysis (PCA) – Kernel PCA – Locally-Linear Embedding (LLE)
Association rule learning (Chengqi and Shichao, 2003)	– Apriori – Eclat

Table 3.1: Unsupervised learning tasks and algorithms

3.1.3 Reinforcement Learning

Reinforcement learning is when a machine learning algorithm learns what to do – how to match situations to actions with a goal of maximizing a numerical reward (Sutton and Barto, 2018). Figure 3.1 shows a typical setting of a reinforcement learning operation. The controller[1] receives both the system's state and the associated reward. The controller then calculates the action and forwards it back to the system. The system transits to a new state in response, this operation is repeated until the total reward is maximized. Some of the examples of reinforcement learning include Markov Decision Process, Q-learning, and Temporal difference learning (Sutton and Barto, 2018).

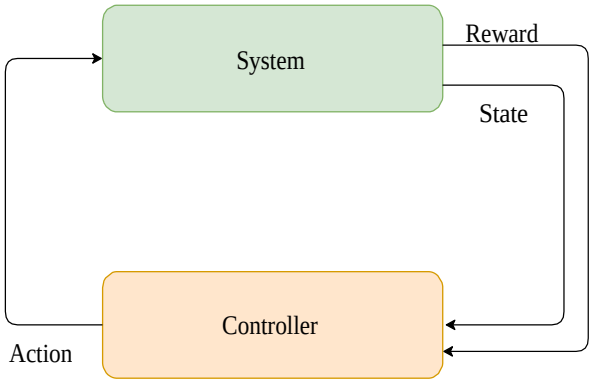


Figure 3.1: A typical reinforcement learning operation. The state of a system or a controller is the state of variables that fully describe the system and the action defines the transition between the states.

3.2 Artificial Neural Networks

Artificial neural networks (ANNs) are machine learning computational methods inspired by the operation of human brain (Jain et al., 1996; Recknagel, 2001). The human nervous system is made of cells known as *neurons*. These neurons are connected to one another by *axon* and *dendrites*, and the connection between them is known as *synapses* (see Figure 3.2(a)). This natural process is simulated using a computational network composed of neurons connected to one another through weights. These weights serve the same purpose as the biological synaptic connections. The artificial network structure is illustrated in Figure 3.2(b).

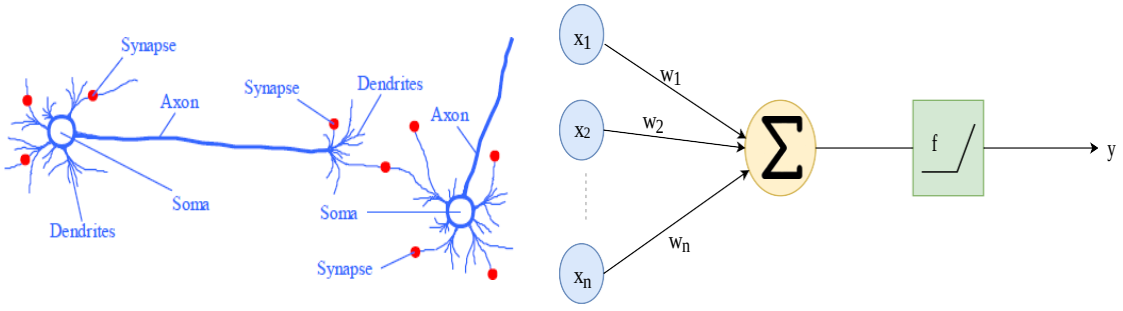
The neuron in an artificial neural network is modeled by Equation 3.1, made of a

[1]A controller is an automated instance that interacts with the system. A system is composed of states and mechanisms that effects transition between states.

weighted summation and an activation function (more details see Section 3.6).

$$y = f\left(\sum_{i=1}^{n} w_i x_i + \beta_i\right) \tag{3.1}$$

where;
$x_i \rightarrow$ Neuron data input
$w_i \rightarrow$ Weighted parameter
$\beta_i \rightarrow$ Bias
$f \rightarrow$ Activation function
$y \rightarrow$ Network output
$n \rightarrow$ Number of data inputs

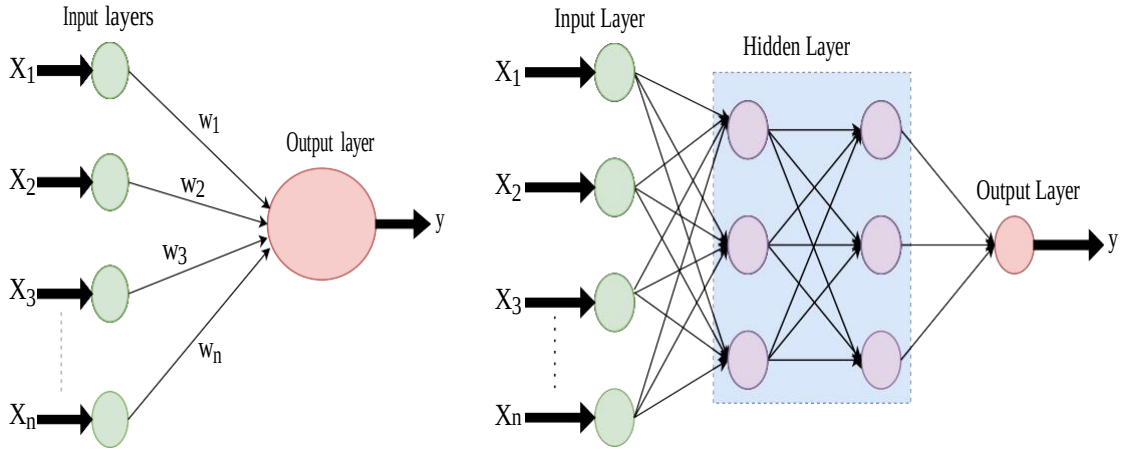


(a) Biological neural network (b) Artificial neural network

Figure 3.2: (a) Shows synaptic connections between the human nervous system while (b) shows the architecture of an artificial neural network where x_s denotes network inputs, w_s represent weights, $\sum$ is the sum of weighted inputs, f is the activation function, and y is the output of the network (adopted from: (Aggarwal, 2018)).

3.2.1 The Typical Architecture of Neural Networks

An artificial neural network can either be a single or a multi-layered neural network. A single layer neural network is one where inputs, x_1, x_2, ..., x_n (where n is the number of inputs) are directly mapped to the output (Figure 3.3(a)). This network is known as a *perceptron* (Aggarwal, 2018). The multi-layered network refers to a network with at least two layers as illustrated by Figure 3.3(b). Between the input and the output layers, there are intermediate layers known as *hidden layers*. There are called hidden layers because the computations within are not observable to the user. Hidden layers also help a neural network to model non-linear functions (Sagar and Simone, 2017).



(a) Single layer network (b) Multi-layer network

Figure 3.3: (a) The single-layer neural network. (b) The multi-layer neural network with two hidden layers. In both figures, n represents the number of input variables (adopted: (Aggarwal, 2018)).

3.3 Deep Learning

Deep learning (Yann et al., 2015) is a subset of machine learning that involves the training of multi-layered artificial neural networks. A neural network with at least two hidden layers is referred to as a deep neural network (Nielsen, 2015). In recent years, deep learning algorithms have become more attractive to researchers compared to the conventional machine learning algorithms (Yann et al., 2015), and this is due to;

1. An increase in the computation power in terms of the graphical processing unit and central processing unit.
2. Availability of huge, well-maintained, and public datasets such as Microsoft's common object and context (COCO) dataset (Lin et al., 2014), ImageNet (Jia et al., 2009), and MNIST handwritten digit database (LeCun and Corinna, 2010). This has increased the accuracy of the deep learning models over the conventional machine learning algorithms as shown in the Figure 3.5.
3. The development of novel state-of-the-art architectures such as AlexNet (Krizhevsky et al., 2012), residual networks (ResNets) (Kaiming et al., 2016a), GoogLeNet (Christian et al., 2015), and region-based CNN (Ross, 2015; Ross and Jiten-dra, 2016; Kaiming et al., 2016a) which not only outperformed the conventional machine learning algorithms but also surpassed humans in the field of classi-fication and recognition (Silver et al., 2017; Tang et al., 2017; Wang et al., 2017).

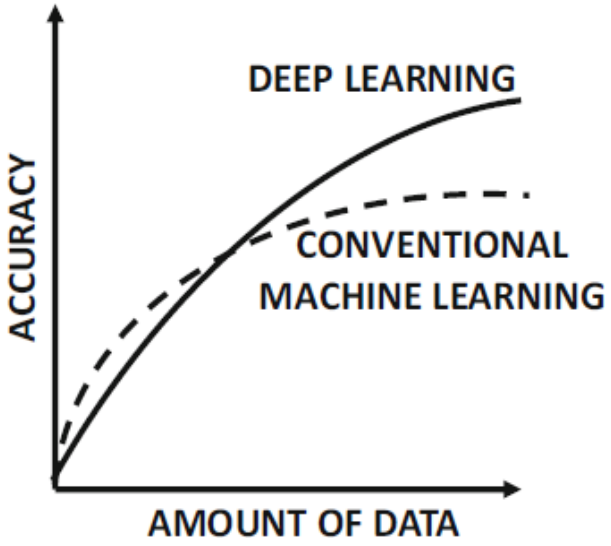


Figure 3.4: A comparison of accuracy of a deep learning algorithm and the conventional machine learning model in relation with the increase of availability of data (Aggarwal, 2018).

3.4 Convolutional Neural Networks

A convolutional neural network (CNN) is a deep learning architecture that has obtained state-of-the-art results on real-world applications such as image classification (Kaiming et al., 2016a), object detection (Ren et al., 2015), semantic segmentation (Long et al., 2015), and speech recognition (Abdel-Hamid et al., 2014). Convolutional neural networks were first introduced in the Kunihiko and Sei (1982) and later modified by Sackinger et al. (1995) to a LeNet-5 architecture which registered tremendous success in recognizing handwritings. The application of CNNs has been popularized in the computer vision community since the 2012 ImageNet challenge when *AlexNet* registered outstanding results (Krizhevsky et al., 2012). In problems such as image classification, CNNs have even outperformed humans (Kaiming et al., 2016a). A typical CNN architecture is divided into two parts, *feature extraction* layers and the *classifier*. The feature extraction part is mainly made of convolutional and pooling layers while the classifier is in most cases composed of fully connected layers as shown in Figure 3.5.

3.4.1 Convolutional Layers

The convolutional layers are the main building blocks and the first layers of the CNNs where majority of the computations take place. A convolutional layer is composed of filters (also known as kernels), that sequentially slide over the width and height of the input images while creating feature maps. This process is also known as *discrete convolution*. The filter's width and height must be smaller than

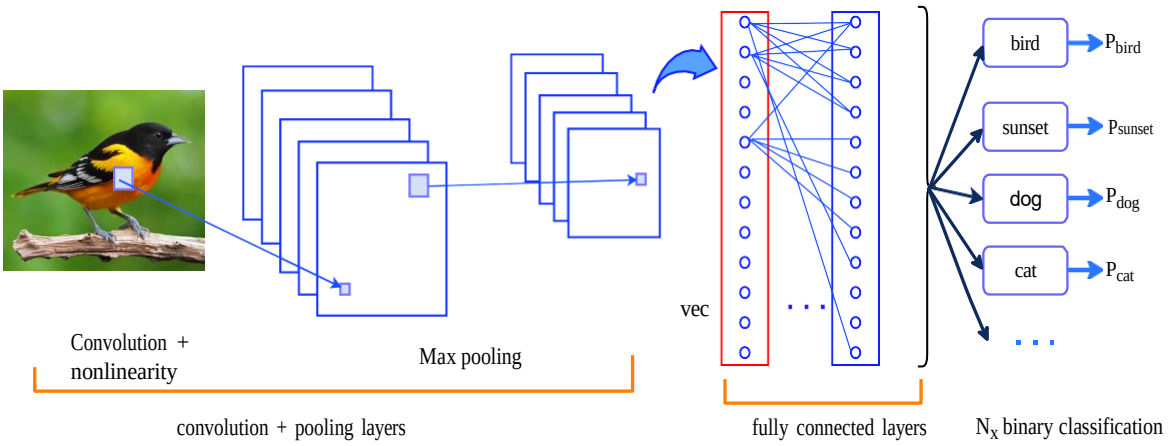


Figure 3.5: The basic CNN architecture. Every input image goes through several convolutional and pooling layers for feature extractions and followed by fully connected layers. Finally, activation function like softmax (see Section 3.6 for more details) to predict the class of object(s) in the image.

the width and height of the input data and the number of filters in a convolutional layer is a hyperparameter. We illustrate the discrete convolution operation in the Figures 3.6 and 3.7.

2	1	4	2	9	7
2	5	7	5	8	4
6	8	2	4	2	6
1	3	4	1	8	1
2	2	7	3	4	7
8	2	6	2	1	4

(a) Input data

1	0	0
1	0	1
1	1	1

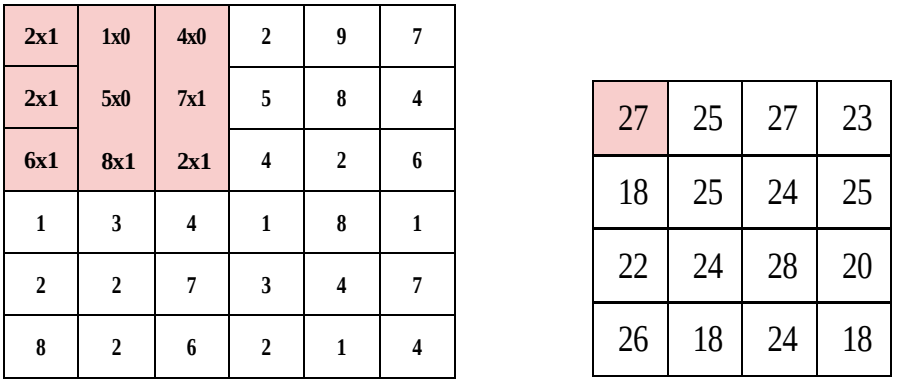
(b) Filter

Figure 3.6: Shows a convolution layer input data of 6×6 and a filter size of 3×3.

The Equation 3.2 presents the mathematical expression of the convolution process (Scherer et al., 2010).

$$G(x,y) = U * F(x,y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} U(n,m)F(x-i,y-j) \tag{3.2}$$

where $F(x, y)$ is the input image, $G(x, y)$ is the feature map, U is the filter$_*$ is the convolution operator, (x, y) is the coordinate of a pixel in the input feature map, n, m are the filter indexes.



(a) Convolution Process (b) Output feature map

Figure 3.7: The convolution process of 6 × 6 input data with a filter of 3 × 3 producing a 4 × 4 feature map. Convolution process, a filter slides over the input image while performing element-wise multiplication and then sums up the multiplied results into a single output feature map pixel. The process is repeated across the entire input image.

3.4.2 Pooling Layers

These are layers that follow convolutional layers in most cases. Pooling layers (Scherer et al., 2010) are meant to reduce the dimensional of feature maps produced by the convolutional layers. A pooling method is supposed to retain useful values in feature maps and discards the irrelevant ones. The pooling layer mainly serves two major purposes in the network firstly, to reduce the number of parameters which in turn reduces the computational time, and secondly, to control overfitting (Gholamalinezhad and Khosravi, 2020). Max pooling and average pooling are the two most popular pooling methods. For max-pooling, a window is slid over the feature map while selecting the maximum value of that window (see Figure 3.8(b)). For the average pooling layer, we compute the average value of all the values in the window as illustrated in Figure 3.8(c).

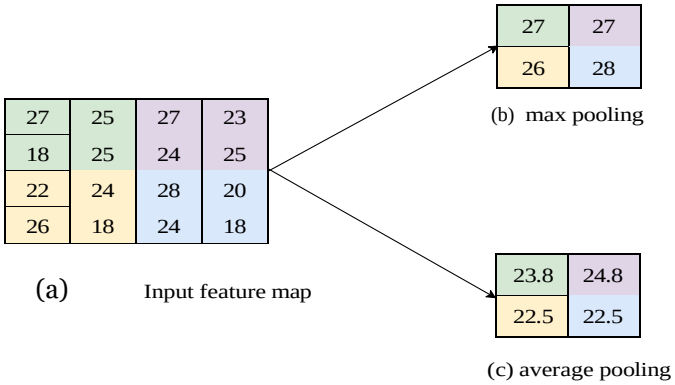


Figure 3.8: Shows a max/average pooling layer with a window size of 2 × 2 plus a stride 2. Stride length defines how many pixels moved by a filter at every step.

3.4.3 Fully Connected Layers

Fully connected layers are applied to the CNN after the convolutional and the pooling layers in the classifier part of the CNN. Depending on the network's depth, one or more fully connected layers are added to the CNN architecture (Basha et al., 2020). For example, *AlexNet* architecture by Krizhevsky et al. (2012) is made of 5 convolutional layers and 3 fully connected layers. Before feeding the output from the final feature map to the fully connected layers, it is *flattened*. Flattening is the process of converting the 3-dimensional matrix output from the pooling and convolutional layers to a vector. After passing through these layers, the final layer uses softmax activation function to classify features learnt by the network. For a typical CNN, the fully connected layers' part has the highest number of parameters because all neurons of one layer are connected to every activation unit of the neighbouring layer. For example, in the VGG-net, out of the 138 million parameters, 123 million parameters are for the fully connected layers (Simonyan and Zisserman, 2014).

3.5 ImageNet Classification with Deep CNN

The convolutional neural network that Krizhevsky et al. (2012) presented in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015) for image classification and object localization was a game changer in the deep learning community. They managed to decrease the top-5 classification error from 25.5% to 15.3% and from 50% to 34.3% for the localization error (from that of the second-best performer in the challenge). Their network was referred to as *AlexNet*. At the time, *AlexNet* was a state-of-the art network that not only improved the classification performance but also opened the door for the CNNs in other computer vision tasks such as object recognition (Donahue et al., 2013), semantic segmentation (Long et al., 2015), speech recognition (Abdel-Hamid et al., 2014), and many others. *AlexNet* introduced some novel features to CNN that were never used before and these include ReLU activation function, dropout layer, data augmentation, local response normalisation, weight regularization, and training a network on multiple graphics processing units (GPUs) (Krizhevsky et al., 2012). In the field of computer vision, *AlexNet* is considered a fundamental advancement that inspired the development of several other state-of-the-art architectures such as VGGNet and Residual Networks (ResNets) discussed in Chapter 4. The architecture of *AlexNet* is shown 3.9.

3.6 Activation Functions

Activation functions (Leshno et al., 1993) are mathematical expressions that define the output of neural networks and they are used in hidden and output layers of the CNN. These functions transform data from one layer into another representation and then used as an input to the next layer in the network. In a convolution neural network, activation functions are applied to the calculated sum of products of inputs

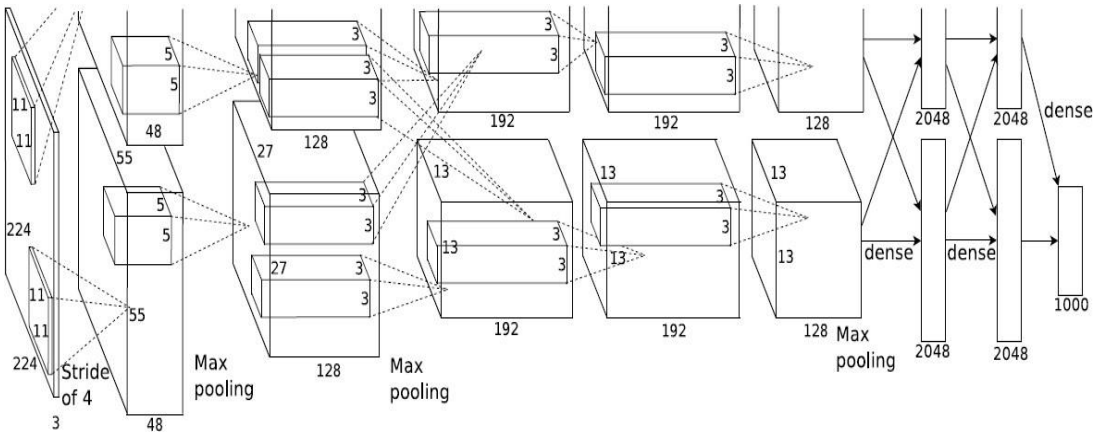


Figure 3.9: AlexNet network is made of 5 convolutional layers with some followed by max-pooling layers, 3 fully connected layers, and 1000-way softmax (Krizhevsky et al., 2012)

and their associated weights to get the output of that layer which is later used as the input to the neighbouring layer. It is necessary to use a non-linear activation function in a neural network because without it, the network's output will just be a simple linear function that does not have the ability to learn and make sense of the complex, high dimensional, and non-linear datasets such as audio, images, text, videos, etc (Sagar and Simone, 2017). Activation functions can be either linear or non-linear. A linear activation function is defined by a straight-line and its output ranges from $-\infty$ to ∞. The problem with this activation is that it cannot be used to solve most of the real-world problems since most of them are non-linear (Abhijit and Ghatak, 2019). This is the reason why non-linear activation functions are the commonly used functions in deep learning. These functions are highlighted in the following Subsections.

3.6.1 Sigmoid

Sigmoid activation function is the most commonly used function for neural network and logistic binary classifications problems (Nwankpa et al., 2018). The function takes in input values between $-\infty$ to ∞ and gives an output between 0 to 1. These values represent the predicted probabilities of a binary output (see Figure 3.10). It can be mathematically expressed as follow.

$$\delta(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

where z is a neuron input.

3.6.2 Hyperbolic Tangent

The tanh function also takes on an S-shaped curve just like the sigmoid function. The difference is that the tanh function outputs values that are between -1 and 1 as indicated in Figure 3.10. Mathematically it can be represented by:

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.4}$$

where z is a neuron input.

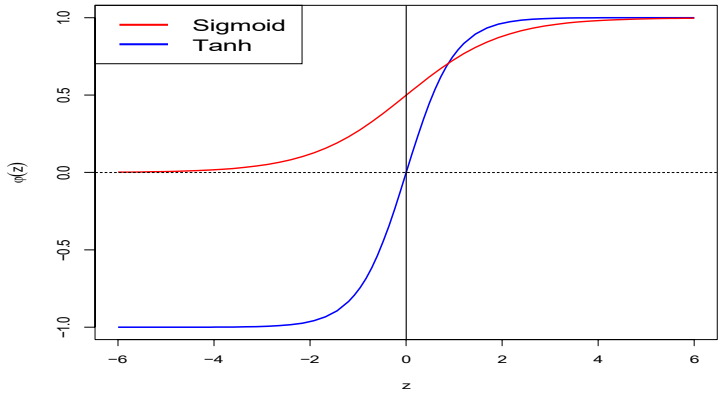


Figure 3.10: The sigmoid Vs tanh activation functions.

3.6.3 Rectified Linear Unit

Rectified linear unit (ReLU) is the most widely used activation function for the CNN hidden layers (Jarrett et al., 2009; Vinod and Geoffrey, 2010; Rifai et al., 2011). Compared to sigmoid and tanh activation functions, ReLU offers the best performance and generalization in CNN (Zeiler et al., 2013; Dahl et al., 2013) and also it does not suffer from vanishing gradient issues (Vinod and Geoffrey, 2010). The ReLU function is half rectified, as illustrated by Figure 3.11. ReLU activation function has an output of 0 for all inputs below 0, and the output is the same as the input for all inputs greater than, 0. The ReLU function can be expressed by:

$$g(z) = max(z, 0) \tag{3.5}$$

where z is a neuron input.

3.6.4 Leaky Rectified Linear Unit

The leaky rectified linear unit (leaky ReLU) (Maas et al., 2013) is an improved version of the ReLU function that takes input values between $-\infty$ to ∞. One of the

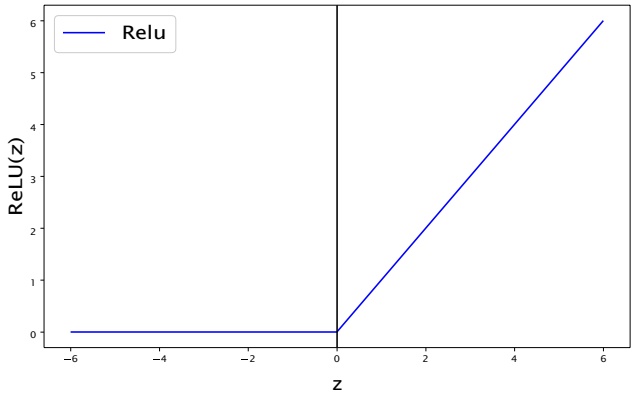


Figure 3.11: The ReLu activation function representation.

drawbacks of ReLU activation function is that it suffers from a problem known as the "dying ReLU" (Chen et al., 2017). In this case, a network becomes a constant function hence causing the ReLU neuron to be never activated again and Leaky ReLU is used to mitigate this problem by employing a small slope instead of zero when $z < 0$. The "leak" is meant to increase the range of input values of the ReLU function as shown in Figure 3.12. It's mathematically represented by:

$$g(z) = \begin{array}{ll} z, & \text{if } z > 0 \\ az, & \text{otherwise} \end{array} \tag{3.6}$$

where a is a fixed parameter in range $(1, +\infty)$ but usually assigned a value of 0.01; and it is known as *randomized relu* when $a \neq 0.01$ (Maas et al., 2013).

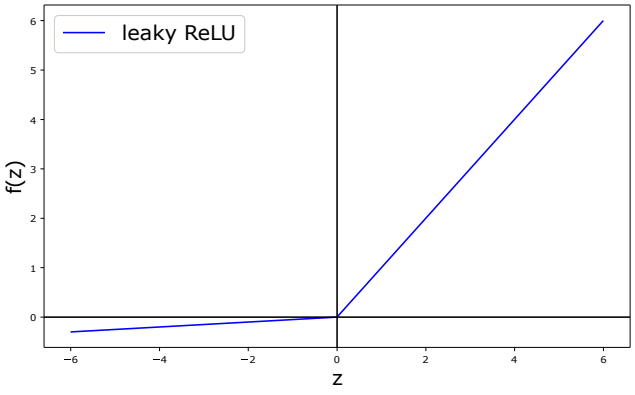


Figure 3.12: The leaky ReLU activation function representation.

3.6.5 Softmax

As explained earlier in Section 3.6.1, sigmoid activation function can only be used to handle binary classification tasks. For multi-class problems, softmax function is used. It calculates the probability of one class over other classes, and its output range is between 0 and 1. Softmax (Goodfellow et al., 2016) is mathematically defined as the ratio of given input exponential value to the total sum of all the input exponential values. It is expressed as:

$$\delta(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K}(e^{z_k})}, \qquad \text{for } j = 1, ..., K. \tag{3.7}$$

where $\delta(z)$ represents the probability of each category and n is total number of categories present in the dataset.

3.7 Regularization

Regularization (Hojjat and Mingyang, 1998) is a machine learning technique used to mitigate *overfitting* on a given training dataset. Overfitting is a tendency where a machine learning algorithm learns the training dataset so well that it performs poorly on unseen data. This problem is prevented by adding an extra term to the model and it is achieved by a number of methods, and in this Section, we discuss the most commonly used techniques.

3.7.1 Weight Decay

Weight decay is also known as L_2 *Regularization* (Loshchilov and Hutter, 2017). It is meant to reduce the model's generalization error but not the training error (Goodfellow et al., 2016). When neural network weights are large, its loss function (discussed in Section 3.8) increases and in effect, we add a weight decay parameter to the cost function that penalizes these large weights. By this, we reduce a network from heavily relying on a few features and hence reducing the possibility of model overfitting since the network is forced to train with multiple features (Vasilev et al., 2019). Practically this is achieved by adding a weight decay term by changing the network's update rule from the following:

$$w \leftarrow w - \eta \nabla(j(w)) \tag{3.8}$$

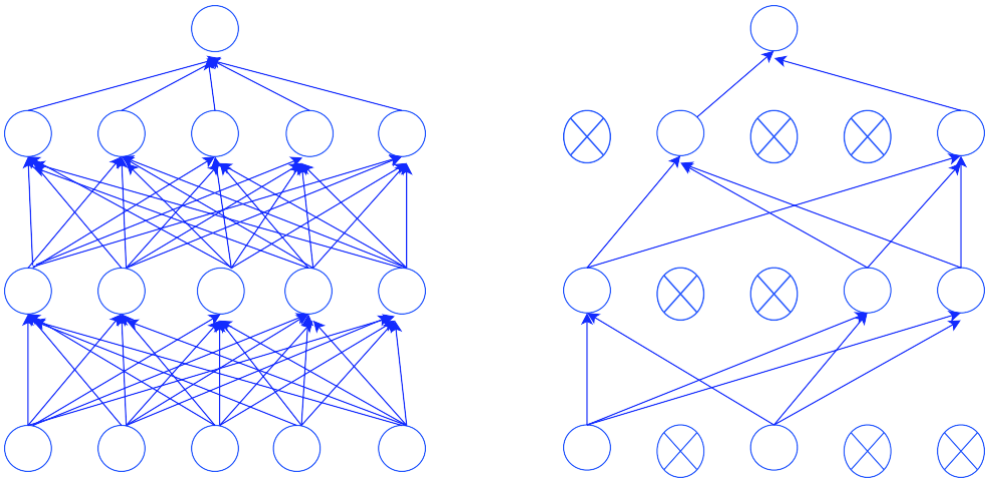
to the following:

$$w \leftarrow w - \eta(\nabla(j(w)) - \lambda w) \tag{3.9}$$

where λ is the weight decay term, η is the learning rate, and w represents the weight vector.

3.7.2 Dropout

This is a regularization method, that is applied to some of the network layers by dropping out neurons. There is temporary removal of some neurons in the network along with all their associated input-output connections (Figure 3.13). The choice of neurons to be dropped is random and periodic (Srivastava et al., 2014). During training, each neuron is retained with fixed probability p, which is independent of other neurons. This probability is determined using a validation set (but it is typically set at 0.5 (Srivastava et al., 2014)). Srivastava et al. (2014) applied the dropout method in their study on computer vision, speech recognition, and text classification problems and showed that dropout improved the performance of the neural networks on all the tasks.



(a) Standard Neural Network (b) After applying dropout

Figure 3.13: Shows a dropout technique applied a standard neural network with 2 hidden layers: Adopted from (Srivastava et al., 2014).

3.7.3 Data Augmentation

It is known that deep learning algorithms require a lot of data for them to achieve better performance (Anaby-Tavor et al., 2020). However, getting large volumes of data is sometimes costly, time-consuming or even impossible (Jason and Luis, 2017). An algorithm is likely to overfit if it is trained on a small data set. One way to solve the problem of limited data is *data augmentation*. Data augmentation is the process of artificially increasing the size of the data by transforming the existing data (Connor and Khoshgoftaar, 2019; Mikołajczyk and Grochowski, 2018). Wong et al. (2016) investigated the effects of data augmentation on classification problems and found that it improved the performance of a machine learning classifier. The most

used data augmentation methods include zoom in/out, cropping, skew, horizontal and vertical flip, and rotation. In Figure 3.14, we illustrate some of the examples:

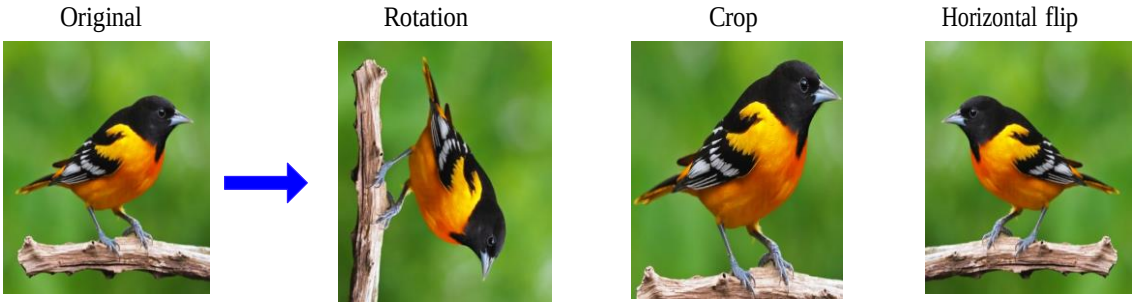


Figure 3.14: Three image augmentation techniques are applied to the original image.

3.7.4 Batch Normalization

When training a deep neural network, the distribution of every layer's input changes as the parameters of the previous layers change. Due to this change, the model's training slows down by requiring lower learning rates which at the end makes it hard to train networks with saturating non-linearities (Sergey and Christian, 2015). This problem is addressed by normalizing inputs in the intermediate layers of the network. This process is known as *Batch Normalization*. Batch Normalization (BN) is applied across mini-batches, not the entire training set for reasons of accelerating the training speed. BN also allows us to use higher learning rates. Sergey and Christian (2015) applied the batch normalization to the state-of-the-art image classification algorithm and found that BN achieved a similar accuracy to the original model but with 14 times lower training steps. They also managed to improve the top leaderboard classification ImageNet result by 4.82% for the top-5 test error when they used batch-normalized networks. BN technique helps in maintaining the mean value close to zero and standard deviation close to one.

3.7.5 Early stopping

During the training of a neural network, validation is usually used to detect when a model starts overfitting. At this point, the training process is stopped before convergence in order to prevent overfitting. This regularization technique is known as *early stopping*. The most common practice to determine the best stopping point is to split the data into training and validation set (Prechelt, 1998). This validation set is used to monitor the ongoing estimate of the model's generalization performance. If the model produces satisfactory results on the validation set, it is taken as the final model.

Algorithm 1 Batch Normalization: Applied to input x over a mini-batch

Input: Values of x over a mini-batch $\mathcal{B} = \{x_{1.....m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$ mini-batch mean

$\delta^2_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$ mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\delta^2_B +}}$ normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathbf{BN}_{\gamma,\beta}(x_i)$ scale and shift

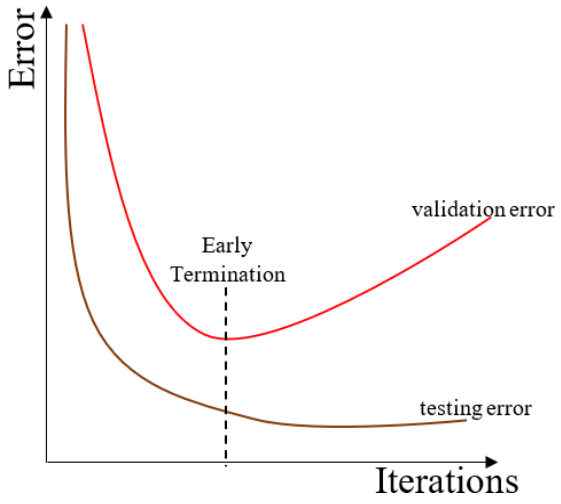


Figure 3.15: Early stopping based on Cross-validation.

3.8 Cost Function

A cost function (Karnin, 1990) measures the divergence between the model's predicted value and the expected output. The smaller the difference, the better the model is doing. Therefore the larger the difference, the more the model should be trained further to improve the prediction accuracy and in the next section, we discuss several optimization methods used in minimizing this error. Some of the most widely used loss functions include, mean squared error (MSE) most commonly used for regression based tasks and cross-entropy for classification.

3.8.1 Mean Square Error

Mean square error (MSE) computes the average squared differences between the actual and the predicted values across all the dataset observations, as seen in the equation:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.10}$$

where N is the total number of observations, $\hat{y}_i$ is the predicted value, y_i is the actual value, and $(\hat{y}_i - y_i)$ is known as the residual value.

3.8.2 Cross-entropy

Cross-entropy in classification problems, helps in quantifying the difference between two probability distributions for a given a set of random events. This is mathematically defined by the expression:

$$Loss = -\sum_{i=1}^{m} y_i \log(p_i) \tag{3.11}$$

where y_i and p_i are the predicted and actual probabilities of class i, and m is the number of classes in the dataset.

3.9 Optimization

The core goal of machine learning is to build models capable of predicting/or detecting a certain set of cases. In order to create a better performing model, optimization is very important. Optimization is a machine learning technique of minimizing the *cost function*. The ultimate goal of optimization is to get optimal parameters that would minimize the cost function as much as possible, this eventually improves the prediction accuracy of the model. This section discusses the most widely used machine learning optimization techniques.

3.9.1 Gradient Descent

Gradient descent is the most widely used optimization algorithm used in machine learning to find the local minimum of the function. Gradient descent begins with an initial estimate w^0 of the solution and updates the solution iteratively until when an optimal solution is found or the maximum number of steps have been reached. It follows two steps:

1. Calculating the derivative of the function ($f: \mathrm{R}^d \rightarrow \mathrm{R}$) at that particular point.
2. We move the coefficient values in the negative gradient direction of the steepest descent: i.e. $\frac{\delta J}{\delta w}$, where $\mathbf{J}(w)$ represents the cost function and $\mathbf{w}$ is the model parameters. The stepwise movement of the coefficients on each update is controlled by the parameter known as *learning rate*(α)). This parameter determines how slow and fast a step should be to achieve an optimal solution for the function.

The performance of the gradient descent algorithm is much determined by the choice of the learning rate values used (Curtis and Scheinberg, 2017). Choosing a good learning rate parameter can be difficult because using a small learning rate results in many small gradient steps, meaning the gradient descent will take long to converge or even never converge. Convergence means the ability of an optimization algorithm to find values for a parameter vector that gives an algorithm the smallest error possible across all the training observations. A large learning rate, on the other hand may cause the gradient descent to miss the optimal solution because it is too fast. The gradient descent algorithm for a single step of an instance j and the weight update for step time $(t+1)$ is mathematically represented by the Equation 3.12 (Curtis and Scheinberg, 2017).

$$w_j^{(t+1)} = w_j^{(t)} - \alpha \frac{\delta \mathbf{J}}{\delta w} \tag{3.12}$$

where α is the learning rate and $w^{(t+1)}$ and $w^{(t)}$ represent the parameter vector at an iteration $(t+1)$ and t respectively. This equation is applied iteratively computing weight coefficients w, which minimises the cost function $\mathbf{J}(w)$. The process is repeated until the convergence of the coefficients.

3.9.2 Batch Gradient Descent

Batch gradient descent is a variation of the gradient descent algorithm that calculates the gradient of the cost function with respect to the parameters, w for the whole training dataset. The iteration through the whole dataset is known as an *epoch* and the batch gradient descent only updates the model's weights at the end of each epoch. This makes the algorithm slow and computationally expensive mainly for large datasets since before performing one update we have to compute the gradient for the entire training dataset.

For example, finding the parameters w that can minimize the "distance" between the $h_w(x_{(i)})$ and $y_{(i)}$ for that expression,

$$\mathbf{J}_{train}(w) = \frac{1}{2m} \quad (h_w(x_{(i)}) - y_{(i)})^2 \tag{3.13}$$

To find w that minimizes $\mathbf{J}(w)$, the following update is performed,

$$w_j = w_j - \frac{\alpha}{m} \quad (h_w(x_{(i)}) - y_{(i)})x_{j\,(i)} \tag{3.14}$$

Where m represents the number of training observations, n is the number of variables, $h_o(x)$ is the linear regression hypothesis, $x_j^{(i)}$ represents the j^{th} variable of the i^{th}. For instance, if you have a dataset with one million observations ($m = 1m$), batch gradient descent is likely to take a lot of time to achieve an optimal solution. This is the reason why batch gradient descent is not recommended for large datasets.

3.9.3 Stochastic Gradient Descent

As we have seen in Section 3.9.2, the batch gradient descent algorithm is extremely slow for large datasets because one update is performed after computing the gradient of the entire training dataset. This problem is solved by using the *stochastic gradient descent* (SGD) algorithm. This algorithm uses batch size[2] 1 i.e. a single observation $(x^{(i)}, y^{(i)})$ is chosen at random and its gradient is computed. For SGD, updates are performed every after a single observation. The process is repeated until the defined number of epochs. Algorithm 2 describes the procedure (adopted from (Marina et al., 2017)).

Algorithm 2 Stochastic Gradient Descent (SGD)

Input: Training data $((x_1, y_1), ..., (x_n, y_n))$, cost function L, learning rate α, initialization $\mathbf{W}$
Output: Model final parameter W_t
repeat
  for *epoch* $\in (1, ..., N)$ **do**
    1. Random sample observation (x_i, y_i)
    2. Compute prediction $\hat{y}_i = f(x_i : \mathbf{W})$
    3. Compute loss $\mathsf{L}^{(i)} = \mathsf{L}(\hat{y}^{(i)}, y^{(i)})$
    4. Compute gradients $\frac{\delta L}{\delta W} \leftarrow$ gradients of $\mathsf{L}(\hat{y}^{(i)}, y^{(i)})$ w.r.t W
    5. update parameters $W_t \leftarrow W_{t-1} + \alpha \frac{\delta L}{\delta W}$
  end
until stopping criterion is met;

3.9.4 Mini-Batch Gradient Descent

This is a variant of the SGD algorithm that uses a subset of the training dataset, this allows the algorithm to perform more updates per iteration compared to the batch gradient descent. Mini-batch gradient descent seeks to take both the efficiency of the

[2]The batch size determines the total number of observations presented to the network at each training iteration.

batch gradient descent and the SGD. And based on this, mini-batch gradient descent is the most widely used algorithm for deep learning problems (Xin and Diego, 2020).

When training a deep learning model, using a large mini-batch size reduces the training time. For example, Kaiming et al. (2016a) managed to reduce the ImageNet-ResNet-50 training time from 29 hours to about 1hour when they used a large mini-batch size of 256. However, increasingly larger and larger batch sizes result in poor model's generalization (Yao et al., 2018; Klaus-Robert et al., 2012). To generalize better, it is advisable to use smaller batch sizes though this comes at the expense of training time as a model trains for a longer time (Kandel and Mauro, 2020).

Mini-batch gradient descent weight update rule can be expressed as

$$w_j := w_j - \alpha \frac{1}{n} \sum_{i=n.k}^{(k+1).n} \nabla_{w_j} \mathsf{L}(\hat{y}^{(i)}, y^{(i)}) \tag{3.15}$$

where n represents the batch size and $k = \{1, \frac{N}{n}\}$ (N is the number of batches).

3.9.5 Adaptive Gradient Algorithm

Adaptive Gradient Algorithm (Adagrad) (Ruder, 2016) is a gradient-based optimization algorithm that divides the learning rate by the sum of the gradient squares which helps the algorithm to adaptively scale the learning rate for every dimension. This makes the algorithm more suitable when dealing with sparse datasets. The Adagrad parameter update equation can be expressed as;

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{+ G_{t,ii}}} . g_{t,i} \tag{3.16}$$

where w is the weight to be updated, α is the initial learning rate, $g_{t,i}$ is the gradient estimate of the objective function at a time step, t (defined by t he E quation 3.17). To avoid a scenario where a function is divided by zero, we use a smoothing term . $G \in \mathsf{R}^{d \times d}$ here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t (Duchi et al., 2011), g_t is the sum of the square of gradients, which is defined by Equation[3] 3.18.

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \mathsf{L}(\hat{y}^{(i)}, y^{(i)}, w_t) \tag{3.17}$$

[3]Refer to https://openreview.net/pdf?id=SJxImoVtwH for details about the Adagrad algorithm.

$$G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^\top \tag{3.18}$$

3.9.6 Root Mean Square Propagation

Root Mean Square Propagation (RMSProp) is another adaptive gradient algorithm like Adagrad but for this, it divides the learning rate with an exponentially decaying average of the squared gradients. This algorithm is mostly used in online and non-stationary frameworks (Nitish and Swersky, 2012). RMSProp parameter update rule is defined by the expressions below.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{3.19}$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{E[g^2]_t + }} g_t \tag{3.20}$$

where γ is a decay parameter, α is the learning rate, and $g_t^2 = \frac{\delta L^2}{\delta W}$. Nitish and Swersky (2012) suggests that and γ are set to 10^{-8} and 0.9 respectively.

3.9.7 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) (Diederik and Jimmy, 2014), is an optimization algorithm where individual adaptive learning rates are computed for every parameter from estimates of both first and second gradient moments. According to Diederik and Jimmy (2014), the Adam algorithm combines both the advantages of RMSProp and Adagrad as discussed in Subsections 3.9.5 and 3.9.6. The method's operational procedure is defined by the algorithm 3 below (adopted from (Khaire and Dhanalakshmi, 2020)).

Algorithm 3 Adam optimization method

Input: Stochastic objective function $f(\theta)$, initial parameter vector $f\theta_0$, learning rate α, exponential decay rates $\beta_1, \beta_2 \in (0, 1)$

$m_0, v_0, t \leftarrow [0,0,0]$ //Initial moment vector and time-step

while θ_t not converged **do**

1. $t \leftarrow t + 1$ //update the time-step
2. $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$//compute gradient w.r.t stochastic objective at time-step t.
3. $m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1).g_t$ //updating the 1^{st} moment estimate
4. $v_t \leftarrow \beta_2.v_{t-1} + (1-\beta_2).g^2_t$ //updating the 2^{nd} moment estimate(g^2_t represent the element-wise square $g_t \odot g_t$)
5. $\hat{m} \leftarrow \frac{mt}{1-\beta_1^t}$ //create bias-corrected 1^{st} moment estimate $\hat{m}_t$
6. $\hat{v} \leftarrow \frac{vt}{1-\beta_2^t}$ //create bias-corrected 2^{nd} moment estimate $\hat{v}_t$
7. $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t}+\lambda}.\hat{m}_t$ //parameter update rule

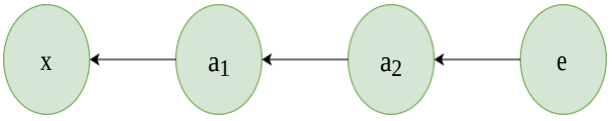
end

return θ_t //final parameter

3.10 Backpropagation

Backpropagation provides an algorithm that computes the gradient of the loss function for every single weight in the neural network using the *chain rule* (Werbos, 1990). It helps us to understand how the changing weights and biases change the error function in the network. In simple terms, backpropagation adjusts the weight by adding Δw to the initial weight: $w_{new} = w - \alpha \frac{\delta E}{\delta w_i}$.

The chain rule in backpropagation is implemented by multiplying partial derivatives that help us to obtain the effect of errors from the output layer to the input. In the example below, we compute the $\frac{de}{dx}$, which defines the effect of the total error on input(x).



$$\frac{\delta e}{\delta x} = \frac{\delta e}{\delta a_2} \cdot \frac{\delta a_2}{\delta a_1} \cdot \frac{\delta a_1}{\delta x} \tag{3.21}$$

The backpropagation algorithm for a unit j, a single output $y\hat{}$, dataset $D = \{(x_n, y_n), n = 1, ..., N\}$ used to determine the components of $\nabla_w E_n(w)$ comprising of partial derivatives of $E_n(w)$ with respect to the weights in $j, w_{i,j}$ can be summarized as:

1. For every dataset pair $(x_n, y_n) \in D$ forward pass the input x_n through a neural network to calculate $\hat{y}(x_n)$. Through this process, we compute the activations a_j for all hidden and output neurons.

2. Using $\delta_j = \sigma(a_j)(\sigma(a_j) - y_n)$, we evaluate δ_j for every output unit.
3. Utilizing $a_j = \sigma(a_j) \sum_{k=1}^{k} w_{jk}\delta_k$ to determine the δ_j for every hidden unit in the network.
4. Using $\frac{\delta E_n}{\delta w_{ij}} = \delta_j z_i$ to compute the derivatives for every node in the network.
5. Finally, weights are updated using $w_{ij} \leftarrow w_{ij} - \alpha \frac{\delta E_n}{\delta w_{ij}}$.

3.11 Introduction to Object Detection

Object detection is a computer vision technique that helps in classifying and localizing one or more objects in images or videos which enables computers to interact or understand the surrounding environment (Zou et al., 2019). This is achieved by drawing a bounding box around the detected object along with its class. Besides predicting the class of the object like image classification, object detection also predicts the location of objects in the image, as shown in Figure 3.16. Researchers in industry and academia have studied and applied object detection to tasks such as autonomous vehicles (Lu et al., 2017), automated surveillance systems (Omar and Mubarak, 2002), image captioning (Karpathy and Fei-Fei, 2015; Xu et al., 2015), medical imaging and many others (Vahab et al., 2019).

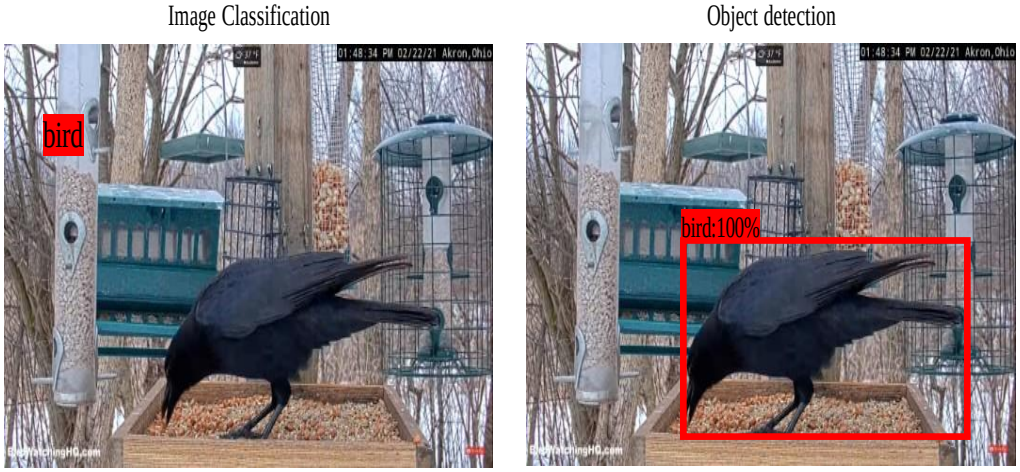
Object detection algorithms are grouped into two classes:

1. Single-stage object detectors.
2. Two-stage object detectors.

Single-stage detectors include YOLO (you only look once), and SSD (single-shot detector) while the two-stage include region-based convolutional neural network (R-CNN), Fast R-CNN, Faster R-CNN, and Mask R-CNN architecture. For more details on object detection CNN architecture see Section 4.1.

3.12 Conclusion

This chapter provided a detailed discussion of machine learning and deep learning methods. The reader has been introduced to several deep learning terminologies including convolutional neural networks, cost function, regularization, and optimization techniques. Furthermore, we presented the backpropagation algorithm. Lastly, an introduction to object detection was discussed. The discussion was mainly limited to techniques used in this thesis.



(a) (b)

Figure 3.16: Image classification Vs Object detection. (a) shows an example of a classification task whereby the model has to predict whether or not the image contains a bird or not. (b) shows an example of an object detection task whereby the model has to detect the object by predicting coordinates of a bounding object around the bird along with the associated probability that the box contains a bird (in this case 100%).

Chapter 4

Object Detection Methods

The major aim of this research is to train object detection models capable of accurately detecting birds of different sizes in photographs. To achieve this objective, it is paramount to select appropriate methods for doing so. In this chapter, some of the most popular state-of-the-art object detection meta-architectures and feature extraction networks are discussed. Meta-architectures are deep learning object detection networks with several building blocks that work in combination with feature extraction networks to detect objects in images, videos, or audio. In the discussion we endevour to address the following three questions.

1. What are the best architectures for detecting birds in photographs?
2. What are the strengths and limitations of these architectures?
3. What are the best and the most suitable hyperparameters for the chosen architectures?

Figure 4.1 shows a workflow diagram of the proposed methodology that comprises of different phases involved in the implementation of the bird detection models. At stage A, we capture images from Cornell Lab Bird Cam [1] live stream feeds (see Section 5.1 for more details). Stage B is for data labeling, see Section 5.1.1. Stage C is for dataset preparation and pre-processing which is discussed in Section 5.2. At stage D, multiple choices are made and these include deciding on convolutional neural network meta-architectures, feature extractors, choices of hyperparameters, and evaluation methods to be used, which are discussed in Chapters 5 and 6. Stage E, explains the application of the models after applying the intersection over union method that determines the final detections, for more details see Section 4.1.5.

[1] https://www.allaboutbirds.org/cams/

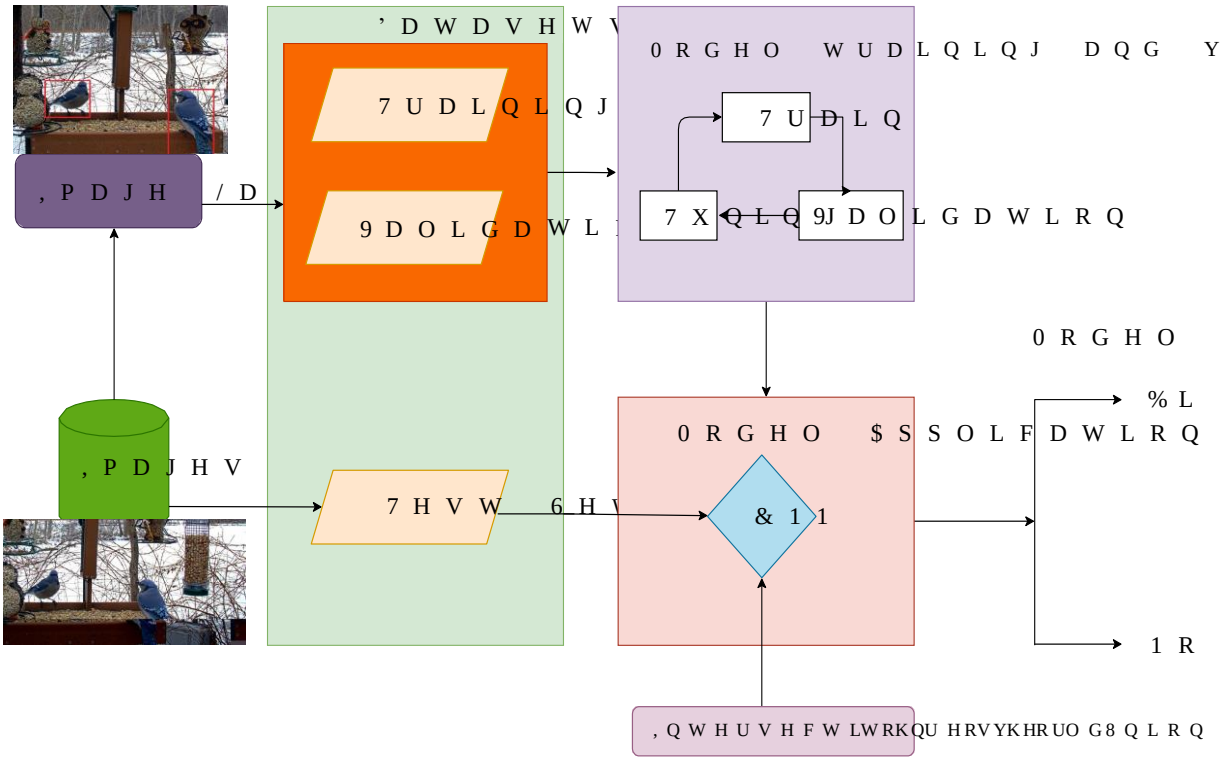


Figure 4.1: Workflow diagram of the proposed methodology

4.1 Meta-architectures

In this study we primarily focus on two state-of-the-art CNN meta-architectures, namely: Faster R-CNN and the SSD. From the studies surveyed in the previous chapter, these object detectors outperformed other techniques in terms of speed and accuracy. More than 50% of entries in the MS COCO object detection competition[2] since 2015 used the Faster R-CNN meta-architecture. It is also known to be a superior object detector in identifying small objects compared to the single stage architectures (Hong et al., 2019). Other common meta-architectures for object detection include, YOLO, Region-based Fully Convolutional Network (R-FCN), Region-based Convolutional Neural Network (R-CNN) and Fast Region-based Convolutional Neural Network (Fast R-CNN). However this thesis is not concentrated on R-CNN and Fast R-CNN, we feel that a brief description of these meta-architectures is appropriate since the Faster R-CNN was built on top of these techniques.

[2] https://cocodataset.org/#detection-leaderboard

4.1.1 Single Shot Detector

The Single Shot Detector (Wei et al., 2016) is a single-stage[3] object detection model based on a feed-forward convolutional network that predicts the presence of an object(s) independently in images using multi-scale convolutional bounding box outputs (multi-scale feature maps). An input image and its ground truth boxes are passed through multiple convolutional layers of the backbone network (or feature extraction network) (see Figure 4.2) extracting feature maps at different points. Each location of these feature maps is evaluated using different scale filters although the 4×4 and 8×8 filters are used most often (Wei et al., 2016; Kumar et al., 2020) to judge a small set of the default boxes (equivalent to anchor boxes of the Faster R-CNN). The default boxes are attentively selected bounding boxes based on their positions, sizes, and aspect sizes across the targeted image (Wei et al., 2016). For every default box, both bounding box offsets and the confidences (or the class probabilities) are predicted. The final detection is decided by the non-maximum suppression algorithm. The SSD network has been used in several object detection studies and it has produced highly competitive results (Hong et al., 2019; Junhwan and Sungho, 2019; Kim et al., 2016). Wei et al. (2016) compared the performance of SSD against its object detector counterparts in terms of accuracy and speed and found that it was favorably competitive.

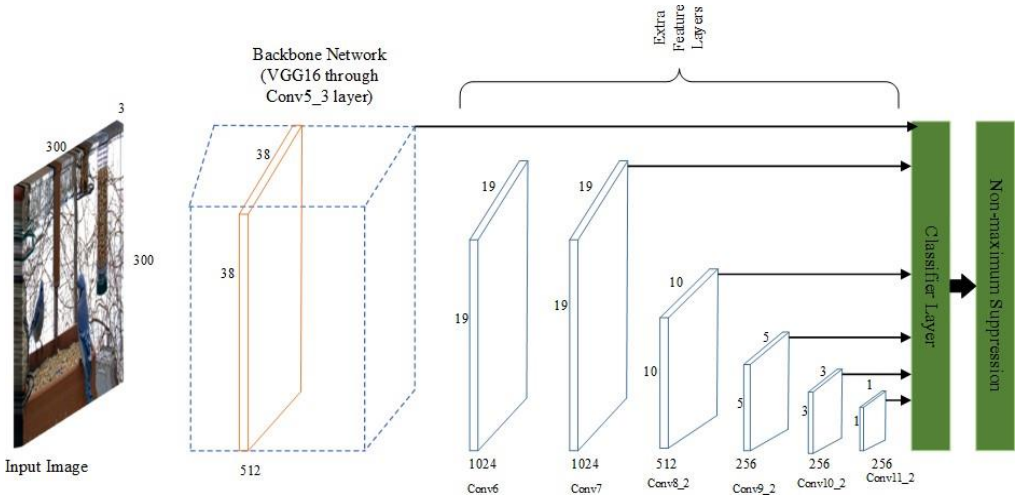


Figure 4.2: The basic architecture of the SSD network model.

Figure 4.2 shows the SSD network. The network is made of three parts: feature extractor, basic network, and the detector. In the basic network, the two last fully-connected layers of the VGG16, namely; FC6 and FC7 are replaced by convolutional layers Conv6 and Conv7. These are followed by four more other convolutional layers Conv8, Conv9, Conv10, and Conv11. The network uses a multitask loss function,

[3]Single stage means that localization and classification are executed simultaneously in the network.

which involves two parts: location loss (loc) and confidence loss (conf) (Cao et al., 2020). The total loss function is the summation of the weighted sum of location and confidence losses, which is mathematically expressed as follows.

$$L(x, c, l, g) = \frac{1}{N}(L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g)) \tag{4.1}$$

where l is the predicted box; g represents the ground truth box; c is the objectness score of the predicted box; N is the total number of the predicted boxes that can effectively match the ground-truth box; α is the weight coefficient of location confidence losses; x is the matching result of the regional candidate box and the ground-truth box of different categories.

4.1.2 You Only Look Once

You only look once (YOLO) is an object detection algorithm that was developed by Redmon et al. (2016). The algorithm frames object detection as a regression problem that spatially separates bounding boxes and associated class probabilities. It performs the classification and localization of objects from images in one evaluation using a single neural network. YOLO divides every input image into a grid of $s \times s$ and every grid predicts N bounding boxes and confidence scores, as shown in figure 4.3. The confidence score reflects how accurate the bounding box is, and whether it actually contains the intended object. This results in a large number of candidate bounding boxes that are consolidated into a final prediction using the non-maximum suppression method (NMS) (see Section 4.1.5) (Elgendy, 2020).

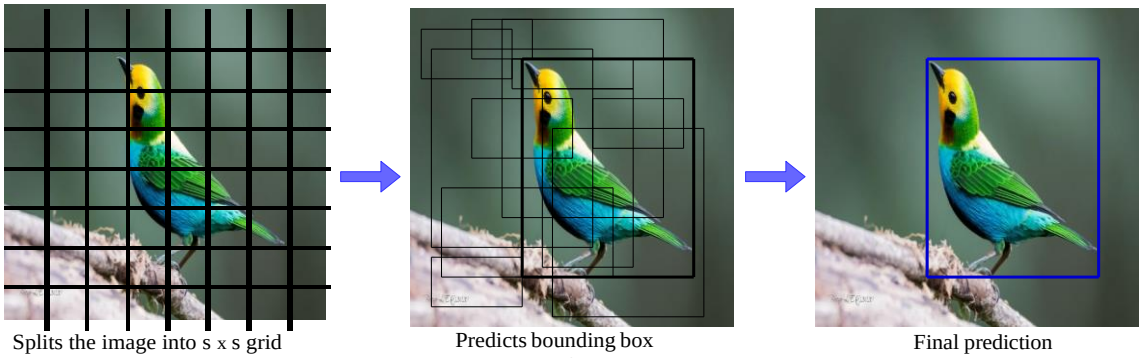


Figure 4.3: YOLO algorithm splits the input image into $s \times s$ grid, predicts objects for each grid, and then uses NMS method to make final predictions.

Over the years, YOLO has gone through several improvements and below are some of the versions:

- YOLOv1 - It was proposed by Redmon et al. (2016) and they described the general architecture of the YOLO algorithm. This version was called "unified, real-time object detection" because it is a single neural network that unifies the two components of a detector: object detector and class predictor (Elgendy, 2020). This design enabled end-to-end training and achieved real-time detection speeds while maintaining a high mAP score.
- YOLOv2 - This variant was introduced by Redmon et al. (2017) and they used predefined anchor boxes to improve bounding box proposals. This version is also known as YOLO9000 because it can detect over 9000 object categories. YOLOv2 used a darknet-19 deep architecture, which is a 19-layer network supplemented with other 11 layers meant for object detection. With this 30-layer network, YOLOv2 performed poorly on small objects. This poor performance was attributed to the loss of fine-grained features as the layers downsampled the input image (Elgendy, 2020).
- YOLOv3 - The design was proposed by Redmon et al. (2018) and it achieved a state-of-the-art performance compared to other YOLO family object detectors. YOLO v1 and v2 used softmax function for the class scores but in v3, Redmon et al. decided to use the sigmoid function because softmax imposes an assumption that each bounding box has exactly one class, which is not always the case. Another key difference, YOLOv3 uses a darknet-53 architecture. In total, it has 106-layers and it is the reason behind the slowness of YOLOv3 but this came with a great boost in mAP score.

4.1.3 Region-based Convolutional Neural Network

Region-based Convolutional Neural Network (R-CNN) was proposed by Girshick et al. in 2014 (Girshick et al., 2014). This network leverages the advantages of the CNN to generate regional proposals to localize and classify objects in images (Girshick et al., 2014). The network was trained on PASCAL VOC 2012 (Everingham et al., 2010) and ImageNet 2012 (Russakovsky et al., 2015) benchmark datasets. Training an R-CNN model involves the following three components:

- Firstly, the network uses selective search algorithm [4] to scan input images and it generates around 2000 region proposals per image. These regions are perceived to have a high probability of containing the object of interest. All these proposed regions are then warped to have a fixed size since selective search generates regions of varying sizes yet CNNs accept fixed-size input images, as we discussed in Section 3.4.

[4]Selective search is a greedy search method that is applied to a CNN to generate region proposals that are perceived to contain intended objects (Uijlings et al., 2013). This method leverages the potential of both the exhaustive search algorithm (that examines all the possible locations in the image) and the bottom-up segmentation algorithm (that hierarchically groups similar regions) to obtain all the possible object locations (Elgendy, 2020).

- Secondly, using a feature extraction network (see Section 4.2), features of each proposed regions are computed.
- Lastly, a classifier such as linear SVM is trained to classify detections based on the derived features from the previous step. Finally, the model outputs real-valued numbers for each of the K object classes to tighten bounding boxes (Elgendy, 2020).

The R-CNN achieved state-of-the-art results on both PASCAL VOC-2012 and ImageNet 2013 object detection challenges. On the PASCAL VOC dataset, it achieved a 62.4% mAP score which was 22.0% better than the second-best algorithm. On the ImageNet, RCNN achieved 31.4% that was 7.0% more than the second-best model.

4.1.4 Fast Region-based Convolutional Neural Network

As discussed above, R-CNN is not a single end-to-end algorithm that learns to localize objects through a deep neural network. Training an R-CNN is a multi-stage process that involves three components: CNN feature extraction network, the SVM classifier, and the bounding box regressors. Training this multi-stage pipeline is very complex, time-consuming, and expensive. Training an R-CNN involves fine-tuning each of the 2000 region proposals and also training multiple class-specific SVMs and bounding box regressors to adjust bounding boxes which makes the process expensive in times of time (Elgendy, 2020). To address these drawbacks, Ross came up with an improved version of R-CNN known as Fast Region-based Convolutional Neural Network (Fast R-CNN) (Ross, 2015). The architecture of Fast R-CNN is like that of R-CNN in many ways but different in the following ways.

- The Fast R-CNN generates region proposals based on the last feature map not from the original image like the R-CNN. Because of this, we can just run one ConvNet over the entire image instead of the 2000 ConvNets over the 2000 intersecting regions.
- Instead of training the multiple SVM algorithms to classify each object, Fast R-CNN applies a single softmax layer that outputs the object class probabilities directly. Therefore, we have only one neural network to train instead of the one network and multiple SVMs.

The entire architecture of the Fast R-CNN consists of the following components:

1. It starts with a feature extraction network that extracts features from the full image.
2. Like the R-CNN algorithms, regions of interest (RoI) are still proposed by the selective search method (Section 4.1.5 gives more details about RoI). It creates about 2000 region candidates for every image.
3. Before feeding the RoIs into a series of fully connected layers, the RoI pooling layer was introduced to extract a fixed-size window. The fully connected layers

are finally divided into two heads: Firstly, the softmax classifier layer that outputs the class of every RoI and secondly the bounding-box regressor layer that predicts offset values relative to the RoI.

The Fast R-CNN increased detection accuracy to 68.4% for the PASCAL VOC 2012 dataset. However, the Fast R-CNN advanced the training and testing time, the selective search algorithm remains the major bottleneck. Selective search algorithm very slow and computationally expensive in generating region proposals (Elgendy, 2020). To solve this problem, Ren et al. proposed a new object detection algorithm known as Faster R-CNN that replaced the selective search algorithm with region proposal network (Ren et al., 2015).

4.1.5 Faster R-CNN

Faster R-CNN (Ren et al., 2015) is a two-stage CNN meta-architecture composed of a Region Proposal Network (RPN) and the Fast R-CNN detector network, see Figure 4.4 below.

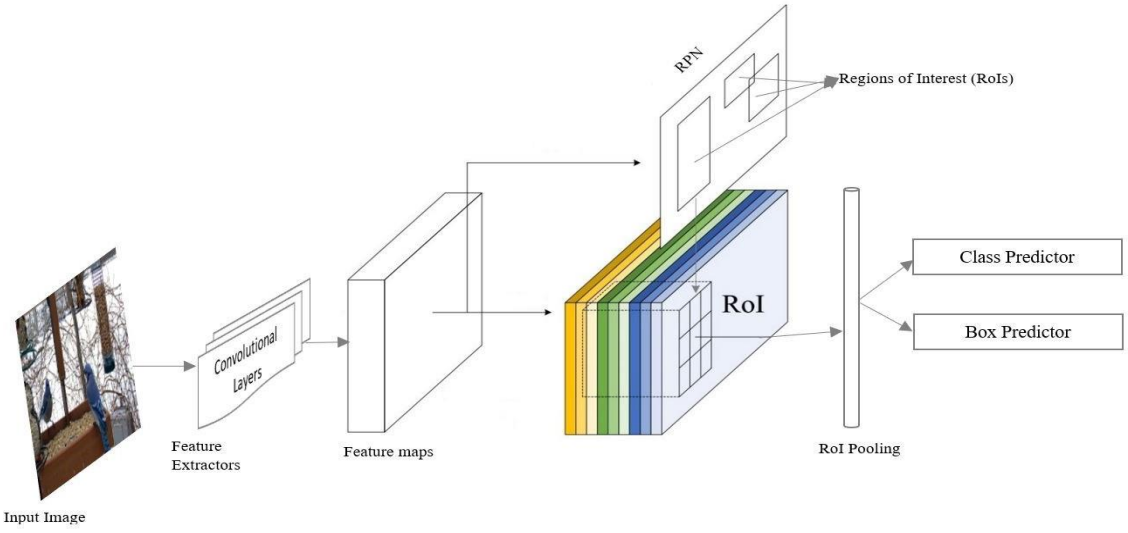


Figure 4.4: The Faster R-CNN object detection architecture. It has three parts: the feature extractors (produces feature maps), a Region Proposal Network (RPN) that generates Regions of Interest (RoIs), and a Fast R-CNN detector head that predicts the classes and boxes.

The Faster R-CNN uses feature extraction networks (discussed in Section 4.2) to obtain feature maps from the input images through several convolutional and max-pooling layers. The generated feature maps are used by the RPN to produce regions of interest[5] (RoI) through a series of convolutional and max-pooling layers (Ren

[5]RoIs are proposed candidate object regions that are thought to contained the object being

et al., 2015). RoIs are proposed candidate object regions, each with an associated objectness score[6] that determines whether the proposal contains an object or not (Figure 4.4). But RPN produces many proposals with potentially a large number of overlapping areas and these multiple detections per image are removed using a non-maximum suppression (NMS) technique (Hosang et al., 2017). A description of the NMS technique is given below. Finally, the proposed regions are fed into a Fast R-CNN detector which predicts whether a bird is contained in the RoI or not. The RPN and Fast R-CNN detector are merged into a single network through sharing their convolutional features (Nguyen et al., 2020). The combination of the two helps Faster R-CNN to achieve better accuracy than the single-stage networks but the accuracy comes at the expense of speed.

Non-maximum suppression

NMS is an object detection post-processing method used to select a single bounding box out of the several overlapping boxes. It makes sure that objects are detected only once. The following steps explains how this method works.

1. The objectness scores of the multiple detections are sorted in decreasing order. Let the list of these proposed detections be B.
2. The proposal with the highest score is removed from B and added to the final detections list D (which is initially empty).
3. Then, compare the overlap (intersection over union, IoU) of the bounding box of the proposal with the highest score in D with the bounding boxes of the other proposals in B.
4. Remove all proposals from B whose IoU scores are greater than the threshold value N.
5. We again select a proposal with the highest score in B and move it to D. Once again we calculate the IoU of the selected proposal with all the remaining proposals in B and boxes with the IoU score greater than N are eliminated.
6. Steps 2 to 5 are repeated until all proposals in B are removed.

The algorithms's pseudo code is given in Algorithm 4 and Figure 4.5 shows how the output looks like.

IoU is the ratio of the area of overlap to the area of union. There is a lot of overlap when the IoU value is close to 1 and little or no overlap when its value is closer to 0. A value of IoU greater than 0.5 is considered to represent the foreground whereas a value of IoU less than 0.5 is considered to represent the background. The value of 0.5 is inspired by the Faster R-CNN study (Villa et al., 2017) but other score values will be investigated in this study.

investigated.

[6]The score is defined to measure how well the detector identifies.

Algorithm 4 The Non-Maximum Suppression

$B \leftarrow$ Generate Proposals($image$)
$B \leftarrow$ SORT(B)
$D \leftarrow \emptyset$
while $B \neq \emptyset$ **do**
    $p \leftarrow$ B p is the highest objectness score in B
    $discard \leftarrow$ False
    for $p \in D$ **do**
        $iou \leftarrow$ GenerateIoU(b, p) b is objectness scores in list B and p is the highest objectness score.
        if $iou > N$ **then**
            $discard \leftarrow$ True N represents the the threshold IoU score
        end
    end
    if ***not*** $discard$ **then**
        PUSH(p, D)
    end
    return p
end

Figure 4.5: Showing ground truth (red box) and predicted bounding boxes (black box).

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

4.2 Feature extractors

This section presents the architectures used by deep learning meta-architectures to extract feature maps. Both meta-architectures described in Section 4.1, subject input images to multiple convolutional layers (or feature extractors) to obtain feature maps. The performance of the object detectors in terms of accuracy, speed, and memory is highly influenced by the number of layers and parameters used in these feature extractors (Huang et al., 2017). Feature extractors used in this study have performed extremely well in terms of accuracy (measured in mean average precision

(mAP)) in the MS COCO object detection competitions (Lin et al., 2014) (seen Figure 4.6) and all their Tensorflow implementations are open source.

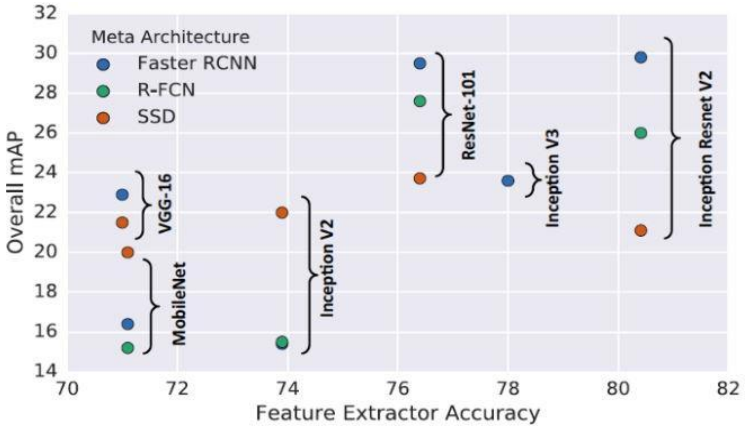


Figure 4.6: Accuracy of detector (as measured in mean average precision (mAP) on MS COCO dataset) vs accuracy of different feature extractors (Huang et al., 2017).

4.2.1 VGGNet

The VGGNet was proposed by Simonyan and Zisserman (2014) and is available in two versions: the VGG-16 and VGG-19 consisting of between 16 and 19 weight layers respectively. They both have very small convolutional filters. In the ILSVRC-2014 competition (Russakovsky et al., 2015), the VGGNet emerged first, in the localization track, and second, in classification. In this study, the implementation of VGGNet closely follows the methodology presented by Simonyan and Zisserman (2014) where we configure a 13 layer convolutional network, all with the same filter size of 3×3. ReLU is used for these 13 layers as the non-linearity activation function. To reduce the spatial dimensions of the feature maps, 5 max-pooling layers of window size 2×2 and 2 strides are used between the convolutional layer blocks. The convolutional layers are followed by three fully-connected layers of different depth: the first two layers have each 4096 channels while the third is a softmax layer with 1000 units. Figure 4.7 is a schematic diagram of the VGG-16 architectural design and the same design is used in all the meta-architectures considered in this study.

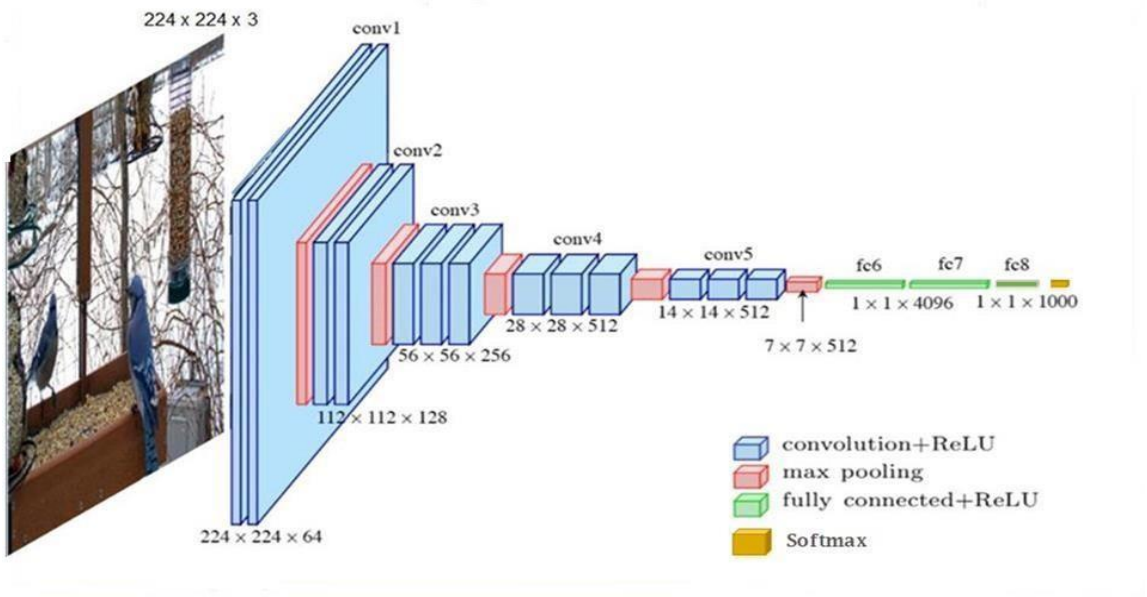


Figure 4.7: The VGG-16 architecture as proposed by Simonyan and Zisserman (2014). During model training, the input to the CNN are fixed sized 224 × 224 × 3 RGB images.

4.2.2 Residual networks

Residual networks (ResNets) were first presented by Kaiming et al. (2016a) in 2015 and at the time the authors had reported improved results on the ImageNet dataset. They presented a 152 layer network that was 8 times deeper than the VGGNets. This network achieved a Top-5 error of 3.5% and this result won the 2015 ILSRVC classification challenge. The Top-5 error is the percentage of the time that the classifier did not include the correct class among its top 5 guesses (Krizhevsky et al., 2012). Submissions based on this deep ResNets architecture went on to win several other challenges including: the COCO detection and segmentation, and ImageNet object detection and localization challenges.

Similar to VGGNets, ResNet presents a similar structure where most convolutions are 3×3 with a 1 zero-padding and the same number of filters of 3×3 in each stage. Stages are the network's building blocks where each stage is composed of several convolutional layers as shown in Figure 4.8. The first convolutional layer uses 64 filters of size 7×7. Between stages, spatial rather than max-pooling is performed by convolutions of stride 2 except for stage 2 where a 3×3 size max-pooling layer and stride 2 are used (Kaiming et al., 2016a). The convolutional layers are all followed by the ReLU activation function. Finally, the last layer consists of a 7×7 global average pooling layer and a one fully connected (FC) classification layer with a softmax. There are three types of residual networks, namely: ResNet-50, ResNet-101 and ResNet-152 (defined in Figure 4.8). In this study, we are going to

investigate the performance of all three residual networks as presented by Kaiming et al. (2016b).

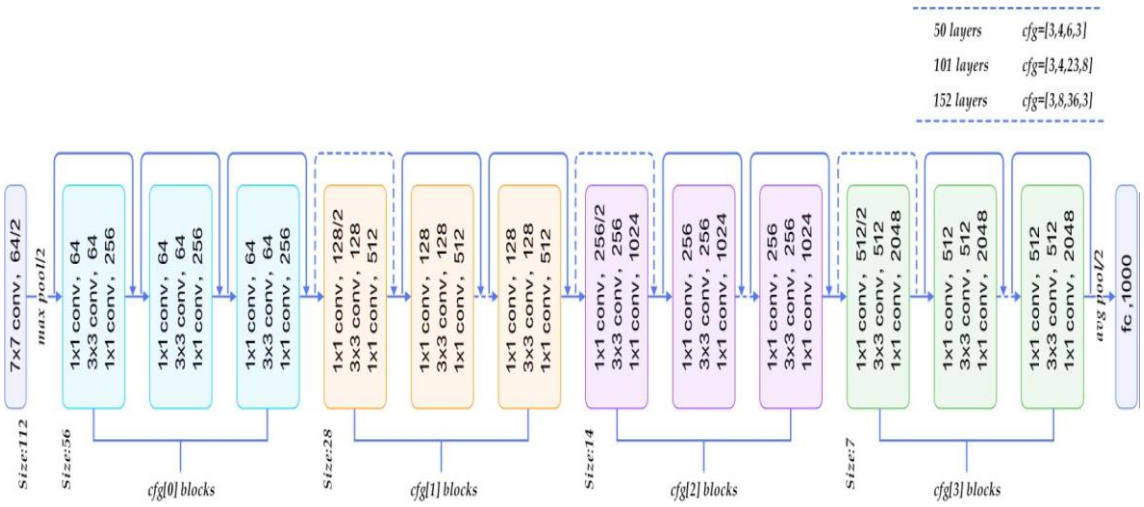


Figure 4.8: The architectural stages and building blocks of the ResNet-50, ResNet-101, ResNet-152 (Deng et al., 2020). The building blocks of the network use bottleneck architecture made of $1\times$ 1, $3\times$ 3, and 1×1 convolutional layers. The $1\times$ 1 layers are responsible for reducing and increasing the channel numbers and the $3\times$ 3 convolutional layer is let to operate on lower dimensions.

4.2.3 MobileNet

MobileNet (Howard et al., 2017) is a lightweight feature extraction network designed for use in limited memory systems. The model is based on the depth-wise separable convolutions (Howard et al., 2017) and it factorizes a standard convolution into a depth-wise convolution and a 1×1 point-wise convolution (Conv). The depth-wise separable convolution gets its name from the fact that, it splits a kernel into 2 separate kernels namely: the depth-wise convolution and the point-wise convolution (He et al., 2019). All layers of the MobileNet are followed by batch normalization (BN) and ReLU activation function apart from the fully connected layer. Figure 4.9 illustrates the architecture of the MobileNet. It is mainly used to design machine learning mobile applications and it was the first TensorFlow computer vision model in that area. It also reduces the computational cost and number of parameters drastically compared to ResNets and VGGNets but with same number of input and output channels (He et al., 2019).

4.2.4 Inception-V2, Inception-V3, and Inception ResNet-V2

At the ILSVRC competitions of 2014, Christian et al. (2015) presented a high performance deep CNN architecture named “Inception” that demonstrated improved

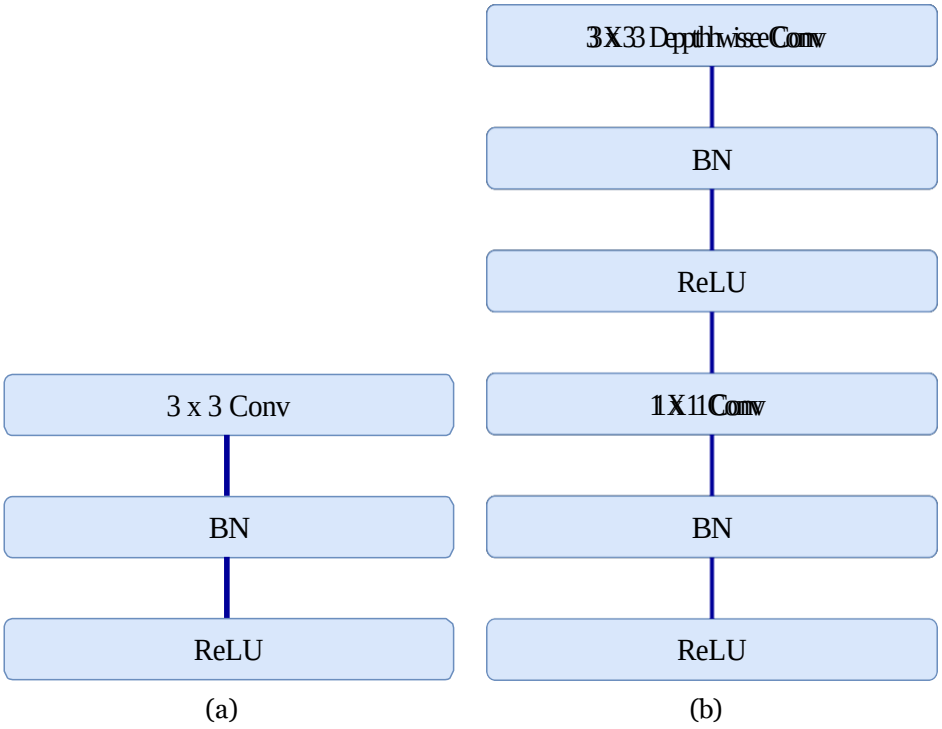


Figure 4.9: (a) Typical MobileNet convolutional network layer with a batch normalization and ReLU activation function. (b) Depth-wise separable convolution with depth-wise and point-wise each layer followed by batch normalization and ReLU (Adopted: Howard et al. (2017))

computational cost compared to ResNets. The original network used three different convolutions 1×1, 3×3, and 5×5. In 2015, Szegedy et al. (2016) proposed several changes to the inception architecture to reduce computational complexity and improve the computational speed, and accuracy. The changes included: replacing the 5×5 convolution with two 3×3 convolutions as shown in Figure 4.10 and factorizing $n \times n$ convolutions to $1 \times n$ and $n \times 1$ combination. Szegedy et al. (2016) found out that their method was six times cheaper computationally and used at least five times less parameters than the best ResNet of Kaiming et al. (2016a).

4.3 Transfer Learning

When training deep learning networks to solve specific problems one of the two problems may arise. The first is not having enough labeled data and the second is having to train a deep network from scratch. Not having enough data would force us to collect and annotate large amounts of data but in most cases the data may not be available. Training a deep network from scratch is a challenging problem because the process may take hours or even days since these weights begin with random values. The optimizer takes a lot of time to converge if at all these initialized values are

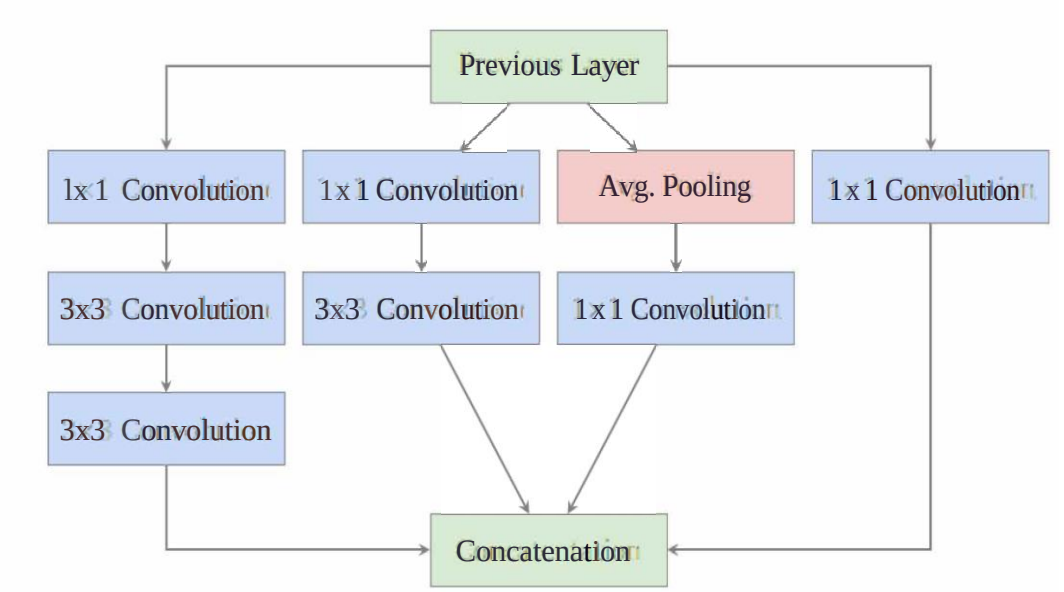


Figure 4.10: Inception-v3 architecture. The average pooling is executed by a stride of 1×1 and padding is applied to all layers to produce the same spatial dimension of the feature maps, which are concatenated at the end of the inception module. (Szegedy et al., 2016)

far from the optimal solution (Elgendy, 2020). One of the ways used to overcome both problems is by utilizing the network weights from pre-trained models. This process is also known as *transfer learning*. Transfer learning is a deep learning technique that allows leveraging the knowledge generated from previous training to a new but related problem (Athanasiadis et al., 2018). We make an assumption that many of the factors that explain the variations in the earlier problem are relatively similar to variations that need to be captured for learning the new problem. For example, in speech recognition modelling, an acoustic model trained to recognize one language can be applied to another with very little re-training data (Dong and Fang, 2015). This technique is not only helpful in situations of little data but also improves prediction accuracy. To confirm, Yabuki et al. (2018) compared object detection accuracy between two deep learning models applied to a dataset of images where the first model used MS COCO pre-trained weights (with transfer learning) and the second model was trained from random weights (without transfer learning). The results of the study showed that the model with transfer learning achieved a detection accuracy of 80% while the model without transfer learning registered a detection accuracy of 48.9%. Furthermore, utilizing the pre-trained models gives us a much faster start of utilizing the previously learned feature maps. In our study, we used this technique by repurposing weights pre-trained on the MS COCO dataset (Lin et al., 2014) for our novel dataset (more details about the MS COCO dataset and the pre-trained models used are discussed in Section 5.6).

Chapter 5

Data and Implementation

To automate the process of detecting birds in photographs, we selected state-of-the-art object detection models, namely; SSD and Faster R-CNN and are both implemented in Python v2.8, Keras v2.2.5 (Chollet et al., 2015), TensorFlow v2.5.0 (Abadi et al., 2015), and OpenCV v4.5.2 (Bradski, 2000) all of which are open source packages. We start this chapter by discussing how the dataset used in this study was generated and prepared for modeling. The process of collecting and preparing data is a time-consuming task but a very important part of any study. Next, we discuss the performance evaluation metrics, and finally, a detailed explanation on the implementation of the training and evaluation of the models.

5.1 Dataset Details

A new dataset was collected for this study. This was collected from the live feed watcher cams[1] of Cornell Lab of Ornithology situated in 6 unique locations around the world as follows:

1. Treman bird feeding garden at the Cornell Ornithology Laboratory in Ithaca, New York, USA. At this station, Axis P11448-LE cameras are used to capture the recordings from feeders perched on the edge of both sapsucker woods and its 10-acre ponds. This site mainly attracts forest species such as chickadees (*Poecile atricapillus*), red-winged blackbirds (*Agelaius phoeniceus*), and woodpeckers (*Picidae*). A total of 2065 images were captured from this location.
2. Fort Davis in Western Texas, USA. At this site, a total of 30 hummingbird feeder cams are hosted at an elevation of over 5500 feet. From this site, 1440 images were captured.
3. Sachatamia Lodge in Mindo, Ecuador. This site has a live hummingbird feed watcher that attracts over 132 species of hummingbirds including Fawn-breasted Brilliant (*Heliodoxa rubinoides*), White-necked Jacobin (*Florisuga*

[1] https://www.allaboutbirds.org/cams/

mellivora), Purple-bibbed Whitetip (*Urosticte benjamini*), Violet-tailed Sylph (*Aglaiocercus coelestis*), Velvet-purple Coronet (*Boissonneaua jardini*), and many others. A total of 2063 images were captured from this location.

4. Morris County, New Jersey, USA. Feeders at this location attract over 39 species including Red-bellied Woodpecker (*Melanerpes carolinus*), Red-winged Blackbird (*Agelaius phoeniceus*), Purple Finch (*Haemorhous purpureus*), Blue Jay (*Cyanocitta cristata*), Pine Siskin (*Spinus pinus*), Hairy Woodpecker (*Leuconotopicus villosus*), and others. Footage at this site is captured by an Axis P1448-LE Camera and Axis T8351 Microphone. A total of 1876 images were recorded from this site.

5. Canopy Lodge in El Valle de Anton, Panama. Over 158 bird species visit this location annually and these include Gray-headed Chachalaca (*Ortalis cinereiceps*), Ruddy Ground-Dove (*Columbina talpacoti*), White-tipped Dove (*Leptotila verreauxi*), Green Hermit (*Phaethornis guy*), and others. A total of 1600 images were captured.

6. Southeast tip of South Island, New Zealand. At this site, nearly 10000 seabirds visit this location annually and a total of 1548 images were captured.

The Cornell Lab of Ornithology is an institute dedicated to biodiversity conversation with the main focus on birds through research, citizen science, and education. The autoscreen[2] software was used to capture the images from the live feeds and images of approximately 1 Megapixel (Joint Photographic Experts Group) JPEG coloured images of resolution $1366 \times 768 \times 3$ pixels were collected. The software took a new image every 30 seconds and were captured during different times of the day in order to avoid a sample biased dataset. In total, 10592 images were collected for this study and Figure 5.1 shows some of the collected images. Our dataset was made publicly available on *Zenodo* (https://zenodo.org/record/5172214#.YRMYdYgzZhF).

5.1.1 Data Labeling

The object detection models need labelled data for training (Girshick et al., 2014), and labelled data in this sense means images with corresponding labels and bounding box coordinates. The bounding box is made of coordinates (x, y, w, h) where x and y are coordinates that describe the center of the bounding box, w and h are the width and height of the object in the image as shown in Figure 5.2 below.

All images used in this study were labelled manually using *LabelImg* (Tzuta, 2018), a graphical image annotation software tool known. The outputs of the LabelImg tool are saved as XML files in PASCAL VOC format with *xmin*, *ymin*, *xmax*, *ymax* positions for every bird as shown in Figure 5.3. This is because it is easier to convert XML files into TF-records which are accepted by TensorFlow (Abadi et al., 2015) (see Section 5.2).

[2] https://sourceforge.net/projects/autoscreen/

Figure 5.1: Sample of images from the dataset collected. To have a dataset with different biases, we collected images in several light conditions, captured birds of multiple sizes from a variety of angles in different environments, partially visible birds in the images, and clouded birds.

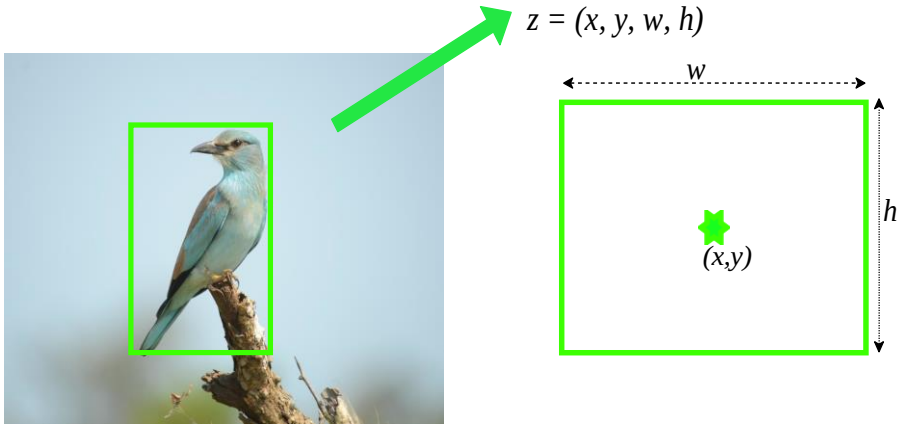


Figure 5.2: Definition of an object detection bounding box.

(a) Image in the LabelImg annotation tool.

```
<annotation>
    <folder>photos</folder>
    <filename>2021_001.jpeg</filename>
    <path>C:/Users/amiru/2021_001.jpeg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>1366</width>
        <height>768</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>bird</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>436</xmin>
            <ymin>376</ymin>
            <xmax>1222</xmax>
            <ymax>713</ymax>
        </bndbox>
    </object>
</annotation>
```

(b) The XML annotation tool output.

Figure 5.3: (a) Shows an image imported into the LabelImg too, where the green box is the ground-truth annotation, and (b) The XML LabelImg annotation tool output containing the four bounding box coordinates shown as *xmin*, *ymin*, *xmax*, and *ymax*. For the image with multiple birds, the XML file will have multiple bounding boxes coordinates representing every bird in the image.

5.2 Pre-Processing

This section examines the pre-processing techniques used to prepare the images before feeding them into the feature extractors. The dataset is split into training and testing sets in the ratio of 4:1 respectively, implying that there is approximately 8474 images in the training set and 2118 images in the testing set. The training set is used to train the models and the testing set is for model performance evaluation.

For hyper-parameter tuning and over-fitting monitoring, 1695 images are set aside from the training set. This set is also referred to as the validation set in the study. The same splitting ratio has been used in a number of machine learning projects and it has produced outstanding results (Norouzzadeh et al., 2018; F. Redmon, 2017). After splitting the dataset, images and labels of the training and the validation sets were converted to TF-Record format files that are compatible with Tensorflow object detection framework (Abadi et al., 2015). After generating the TF-record files for both training and validation sets, a label map that contains a bird unique ID was created. The label map is used by the object detector during training and detection processes to correctly identify object class names.

5.3 Performance Evaluation Metrics

Object detectors aim to predict the location of objects of a given class in an image or video with a high confidence. They do so by placing bounding boxes to identify the positions of the objects. To decide whether predicted bounding box is correct with respect to an object or not, the IoU or Jaccard Index is used (see Section 4.1.5 for details of the IoU). The IoU therefore forms a basis for most of the metrics that are used to evaluate object detection algorithms. For object detection, the most commonly used evaluation measure is the average precision (AP) and mean average precision (mAP). In this thesis we also intend using these metrics in addition to inference time. The objective of this section is to review these metrics and understand how they are used.

5.3.1 Precision and Recall

The AP and mAP are both derived from two well known evaluation measures used in classification models, namely precision and recall. Precision is the ability of a model to identify only relevant objects. That is, of all the positive predictions, how many are true positives. Recall is the ability of a model to find all ground-truth bounding boxes. That is, of all the actual positives, how many are predicted as true positives (Hui, 2018). To calculate the precision and recall values, each detected bounding box must first be classified as:

- True positive (TP): A correct detection of a ground-truth bounding box;
- False positive (FP): An incorrect detection of a non-existing object or a misplaced detection of an existing object;
- False negative (FN): An undetected ground-truth bounding box.

Based on the above definitions, if given an image which can be thought of as made up of objects with G ground-truths and a model that outputs N detections, of which S are correct ($S \leq G$), then precision, P, is defined as:

$$P = \frac{\sum_{j=1}^{S} \mathrm{TP}_j}{\sum_{j=1}^{S} \mathrm{TP}_j + \sum_{j=1}^{N-S} \mathrm{FN}_j} \tag{5.1}$$

whereas recall, R, is defined as:

$$R = \frac{\sum_{j=1}^{S} \mathrm{TP}_j}{\sum_{j=1}^{S} \mathrm{TP}_j + \sum_{j=1}^{G-S} \mathrm{FP}_j} \tag{5.2}$$

It is clear that, the higher the precision, the more confident the model is when it predicts a bounding box as positive. The higher the recall, the more positive bounding boxes the model correctly predicts as positive. When a model has high recall but low precision, then the model predicts most of the positive bounding boxes correctly but it has many false positives (i.e. predicts many negative bounding boxes as positive). When a model has high precision but low recall, then the model is accurate when it predicts a bounding box as positive but it may predict only some of the positive bounding boxes. An important intermediary step in the computation of the mAP is the construction of the precision-recall curve discussed next.

5.3.2 Precision-Recall Curve

A prediction is considered to be true positive if the value of the IoU value is greater than a threshold, and false positive if the IoU value is less than the threshold. Suppose that the threshold value is λ. Following Padilla et al. (2020), equations 5.1 and 5.2 can be rewritten to reflect the dependence of the detections on the threshold λ as:

$$P(\lambda) = \frac{\sum_{j=1}^{S} \mathrm{TP}_j(\lambda)}{\sum_{j=1}^{S} \mathrm{TP}_j(\lambda) + \sum_{j=1}^{N-S} \mathrm{FP}_j(\lambda)} \tag{5.3}$$

$$R(\lambda) = \frac{\sum_{j=1}^{S} \mathrm{TP}_j(\lambda)}{\sum_{j=1}^{S} \mathrm{TP}_j(\lambda) + \sum_{j=1}^{G-S} \mathrm{FP}_j(\lambda)} \tag{5.4}$$

The relationship between $P(\lambda)$ and $R(\lambda)$, can be plotted to generate what is known as a precision-recall curve. The curve shows the trade-off between the precision and recall values for different λ values. This curve helps to select the best threshold value to maximize both metrics (Padilla et al., 2020).

5.3.3 F_1 metric

In practice, graphically, deciding the best threshold value that maximises both the precision and recall from the precision recall curve is not easy as the curve tends

to be complex. An alternative is to use a metric called the F_1 score, given by the following equation.

$$F_1 = 2\frac{P(\lambda) \times R(\lambda)}{P(\lambda) + R(\lambda)} \tag{5.5}$$

This metric measures the balance between precision and recall. A high value of F_1, means that both the precision and recall are high. A low F_1 score means greater imbalance between precision and recall (Hui, 2018).

5.3.4 Average Precision

A related measure that is derived from the precision-recall curve is the AP. The AP for a particular class is equal to the area under the precision-recall curve. The model is considered a good predictive model if the precision stays high as the recall increases. Hence, a large area under the curve (AUC) tends to indicate both high precision and high recall (Padilla et al., 2020). Computing the area under the curve in order to obtain AP for a particular class can be achieved using the following formula:

$$\text{AP} = \sum_{\lambda=0}^{n-1} R(\lambda) - R(\lambda+1) \times P(\lambda) \tag{5.6}$$

where $R(n) = 0$, $P(n) = 1$ and n is the total number of thresholds values to be assessed. We are now ready to compute the mean average precision (mAP).

5.3.5 mean Average Precision (mAP)

mAP is the average precision averaged over all the object categories in the data set (Hui, 2018). Suppose that the image has objects belonging to C classes, then based on the APs for the C classes, the mAP for the model can be calculated using the following formula:

$$\text{mAP} = \frac{1}{C}\sum_{j=1}^{C} \text{AP}_j \tag{5.7}$$

where AP_j = AP of the j^{th} class and C = number of classes.

The mAP is the most popular object detection evaluation metric. It measures the overall accuracy of an object detection model (Padilla et al., 2020; Szegedy et al., 2016; Wei et al., 2016). The higher the mAP, the better the detection.

5.4 Inference Time

Inference time refers to the time taken by the model to produce the detection results and it is mainly measured in milliseconds (ms) (Marco et al., 2020). Inference time helps us understand how fast the models are in detecting birds in images. This time is obtained through a process known as deep learning inference. Deep learning inference refers to the process of subjecting the trained deep learning model to unseen data to make predictions.

5.5 Hardware

We ran the experiments on an MSI GL75 Leopard 10SFR laptop with CUDA 11.0, cuDNN SDK 8.0.4, and Windows 10 x64. The hardware configuration of the laptop is as follows: 10th Gen Intel Core i7-10750H, GeForce RTX 2070 8GB GDDR6 graphics processing unit (GPU), and 32GB DDR4 RAM. The NVIDIA CUDA Deep Neural Network library (cuDNN) refers to the GPU-accelerated package that helps in accelerating the deep learning frameworks such as TensorFlow and Keras. It also provides a platform with access to tuned implementations of deep learning standard routines such as convolution, normalization, pooling, forward and backward propagation, and the activation layers (Chetlur et al., 2014).

5.6 Tensorflow object detection API

TensorFlow (Abadi et al., 2015) is an end to end machine learning open-source platform developed by Google, that transforms data and images into *tensors*[3] to be processed by neural networks. Tensorflow makes it easier to build and deploy deep learning based applications because of the flexible tools, community resources, supported APIs, and numerous libraries provided. One of the several APIs provided by the TensorFlow library is the object detection API[4]. TensorFlow's Object Detection API is an open source framework built on the top of Tensorflow that facilities the training and deployment of object detection models.

In this study, the pre-trained object detection model weights that come with TensorFlow's Object Detection API were utilized. The pre-trained models were built on the MS COCO dataset (Lin et al., 2014) made of 2500000 labeled objects in 328000 images with 90 different classes of objects. MS COCO has been used as a benchmark dataset for many object detection researchers because it has proportionately more instances per category than any other available public datasets such as PASCAL VOC (Everingham et al., 2010) and also contain more objects (7.7 per image) than the popular ImageNet and PASCAL VOC with 3 and 2.3 objects per image respectively (Lin et al., 2014).

[3]Tensors are multi-dimensional data arrays used by TensorFlow.
[4]https://github.com/tensorflow/models

Using the pre-trained weights helps to reduce the training time as opposed to initialising the model with random weights. Based on the literature review and methods discussed in Chapter 3 and Chapter 4 respectively we selected two meta-architectures, namely, Faster R-CNN and SSD implemented using the following feature extractors, MobileNet, ResNet-50, ResNet-101, ResNet-152, and Inception ResNet v2. The choice of the feature extractors was based on the reported outstanding results in a number of studies (Ren et al., 2015; Wei et al., 2016; Huang et al., 2017; Kaiming et al., 2016a; Christian et al., 2015).

The structural organization of the Tensorflow object detection API configuration pipeline consists of the following blocks:

1. The pre-processor that performs image downsampling and data normalization (input data converted to [0,1] interval). The number of classes in the training dataset are also indicated at this point.
   Normalization is performed using:

$$z_{ij} = \frac{x_{ij} - min_j}{max_j - min_j} \tag{5.8}$$

   where min_j and max_j are minimum and maximum values respectively of every attribute value x_{ij} for the j_{th} dimension of the i_{th} point.

2. The first stage feature extraction base network.
3. The first stage anchor generator that produces anchor coordinates.
4. The first stage box predictor that helps in calculating prediction classes and boxes.
5. The decoder that facilities the decoding of boxes using data and anchors generated by the first stage box predictor.
6. First stage multi-class non maximum suppression where non maximum suppression is performed.
7. The second stage feature extraction that extracts features in the predicted regions of interest.
8. The second stage box predictor that process box coordinates according to the second stage feature extraction base network.
9. Post-processor stage which is a detection output layer that predicts the final boxes.

5.7 Model Training

This section discusses the procedural steps taken in training and hyper-parameter tuning of eight models developed using the two considered object detection meta-

architectures: Faster R-CNN and SSD with the following five different feature extraction networks, namely: MobileNet, ResNet-50, ResNet-101, ResNet-152, and Inception ResNet V2. The developed code for pre-processing and model implementation is available at https://github.com/mirugwe1/bird_detection.git.

5.7.1 SSD MobileNet-V2

In this method, SSD is used as the meta-architecture and MobileNet as the feature extraction network. The network was initialized using weights that were pre-trained (SSD MobileNet v2 coco)[5] on the MS COCO dataset. The original input image frames consisted of $1366 \times 768 \times 3$ (width $\times$ height $\times$ depth) pixels and were resized to the MobileNet standard size of $320 \times 320 \times 3$ pixels (Howard et al., 2017) before feeding them to the network. To achieve the best detection results on our dataset, during training and fine-tuning of the network, we adopted similar hyper-parameters as Huang et al. (2017) because of the good performance they yielded. The network hyper-parameters were accordingly set as follows: number of epochs 10; kernel size 3; optimiser - stochastic gradient descent with momentum of 0.9; batch size 16; learning rate 0.08. For the post-processing part, we used 1×10^{-8} and 0.6 as the score and IoU thresholds respectively while the weights of both localization and classification were set to 1. The execution of the best model took four hours to train and run for 50000 steps during hyper-parameter tuning.

The training and parameter tuning processes were visualized using Tensorboard. Tensorboard[6] is a tool developed by Google to interface with TensorFlow, and it is used to monitor and visualize both training and validation progress of the neural networks in real-time.

5.7.2 SSD ResNet50-V1

We trained SSD detection model with a ResNet-50 feature extractor network whose weights were pre-trained on the MS COCO dataset (SSD ResNet-50 coco)[7], and we followed the experimental procedural setup as used by Kaiming et al. (2016b) and Huang et al. (2017). The parameters of the network were configured as follows: all images fed to the network were resized to $320 \times 320 \times 3$ pixels (width $\times$ height $\times$ depth), the loss function was minimized using the SGD optimization algorithm with momentum value of 0.9 trained for 30000 steps and 10 epochs, the bounding boxes and classification probabilities of the feature maps were predicted using a convolutional kernel of size 3×3, and a dropout probability of 0.8. The learning rate of 0.04, a weight decay of 0.0004, and batch size of 16 were used. For the post-processing part, a score threshold of 1×10^{-8} with an IoU value of 0.6 was used. Finally, both

[5] http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz

[6] https://www.tensorflow.org/tensorboard

[7] http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz

weights of localization and classification were set to 1.

5.7.3 SSD ResNet101-V1

For this model, we utilize SSD (Wei et al., 2016) as the meta-architecture with a ResNet-101 feature extraction network Kaiming et al. (2016b). The model is trained using the MS COCO SSD Resnet-101[8] pre-trained weights. We used the same hyper-parameters as the SSD with ResNet-50 except that the network was trained for 27000 steps.

5.7.4 SSD ResNet152-V1

This model also utilizes the SSD (Wei et al., 2016) meta-architecture coupled with a 152 layer residual base network (ResNet-152) Kaiming et al. (2016b) for feature extraction. We use the pre-trained weights of SSD Resnet-152 v1[9] to train the model with similar hyper-parameters as the SSD with ResNet-50 except for the number of training steps, the learning rate, and the batch size. The model was tuned for 120000 steps with a learning rate of 3×10^{-3} using an image batch size 4.

5.7.5 Faster R-CNN ResNet50-V1

The Faster R-CNN object detection model (Ren et al., 2015) usually begins with a region proposal network that generates proposals. This phase was implemented using a 50 layer residual feature extraction network (ResNet-50) Kaiming et al. (2016b) with the pre-trained weights of Faster RCNN Resnet-50 v1[10]. The input images were resized to $1024 \times 1024 \times 3$ dimensional size. We followed closely the implementation setup used by (Huang et al., 2017) to fine-tune the model on our dataset. We set the network's grid anchor size to 16×16 pixels scaled to [0.25, 0.5, 1.0, 2.0]. The non-maximum suppression-IoU threshold is set to 0.7, localization loss weight to 2.0, initial crop size to 14×14 pixels and objectiveness weight to 1.0. Finally, the kernel size of 2×2 with stride of 2.0 was set. The prediction IoU threshold score value was set to 0.6, the loss function is optimized using SGD optimizer with the momentum of 0.9 training for 20000 steps at a learning rate of 4×10^{-3}. The model's batch size was set to 2 and the number of epochs to 100.

5.7.6 Faster R-CNN ResNet101-V1

This model also uses the Faster R-CNN (Villa et al., 2017) as a detection architecture while ResNet-101 (Kaiming et al., 2016b) as the feature extraction network. The

[8] http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet101_v1_fpn_640x640_coco17_tpu-8.tar.gz

[9] http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet152_v1_fpn_1024x1024_coco17_tpu-8.tar.gz

[10] http://download.tensorflow.org/models/object_detection/tf2/20200711/faster_rcnn_resnet50_v1_1024x1024_coco17_tpu-8.tar.gz

model was initialized with MS COCO Faster RCNN Resnet-101 v1[11] pre-trained weights. Except for the learning rate and the number of training steps, the training process of the model uses the same hyper-parameters as the Faster R-CNN with ResNet-50 described in Section 5.7.5 above. Initially, the model was trained for 10000 steps at a learning rate of 0.002, and then decreased to 0.0002 for the next 7000 iterations.

5.7.7 Faster R-CNN ResNet152-V1

The model is using a 152 layer residual network (ResNet-152) (Kaiming et al., 2016b) to extract features from the input images. All the images were downsampled to $640 \times 640 \times 3$ pixels. We utilized the Faster RCNN Resnet-152 v1[12] pre-trained weights to train model on our customized dataset. We used the hyperparameters as applied to the Faster R-CNN with ResNet-50 except that this model trained for 30000 iterations. The model was initially trained at a learning rate of 0.004 for 20000 steps and later reduced to 0.0004 for the next 10000 steps.

5.7.8 Faster R-CNN Inception ResNet-V2

We finally trained our last Faster R-CNN model that employed Inception ResNet-V2 as the feature extraction network. The model was initialized using the Faster RCNN Inception-ResNet V2[13] pre-trained weights. We fine-tuned this model using hyper-parameters similar to those described in the subsection 5.7.2. The difference was only in the number of training steps and the learning rate. The network was initially trained for 40000 iterations with a learning rate of 0.03 and then the network was trained for 15000 iterations with a learning rate reduced to 0.003. In total the network was trained for 55000 iterations These hyper-parameters were adopted from the network of Ren et al. (2015) which registered outstanding performance in both the ILSVRC and COCO 2015 competitions.

[11] http://download.tensorflow.org/models/object_detection/tf2/20200711/faster_rcnn_resnet101_v1_1024x1024_coco17_tpu-8.tar.gz

[12] http://download.tensorflow.org/models/object_detection/tf2/20200711/faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8.tar.gz

[13] http://download.tensorflow.org/models/object_detection/tf2/20200711/faster_rcnn_inception_resnet_v2_640x640_coco17_tpu-8.tar.gz

Chapter 6

Results and Discussion

In this chapter, a comprehensive discussion and comparison of results achieved by the SSD and Faster R-CNN object detection meta-architectures using the pre-trained MobileNet-v2, ResNet-50, ResNet-101, ResNet-152, and ResNet Inception V2 feature extraction networks with and without fine-tuning is provided.

6.1 Single Shot Detector

After training and evaluating the SSD model with MobileNet, ResNet-50, ResNet-101, ResNet-152 feature extractors, we summarise the model's detection performance results in Tables 6.2 and 6.3 below. We also evaluated the model's average precision (AP) and average recall (AR) on the different range of bird sizes (small, medium, and large) as defined by the standard MS COCO evaluation tool[1]. Table 6.1 shows how this tools defines the different bounding box sizes.

	Min bird size	Max bird size
Small	0 x 0	32 x 32
Medium	32 x 32	96 x 96
Large	96 x 96	∞ x ∞

Table 6.1: MS COCO's definition of small, medium, and large objects.

Regardless of the backbone network used, the model's performance in detecting small birds is poor but it achieves slightly higher accuracy in detecting large sized birds with AP of 73.5%,78.5%, 88.3% and 92.1% for SSD with MobileNet, ResNet-50, ResNet-101, and ResNet-152 respectively. Figure 6.1 compares the performance between MS COCO pre-trained models with fine-tuning and those without fine tuning of the models. The pre-trained models without fine-tuning were not able to detect the birds very well and for this reason, we fine-tuned the derived MS COCO weights further on our data set via transfer learning. There was a significant improvement

[1] https://cocodataset.org/#detection-eval

in the average precision in all the fine-tuned models. The SSD ResNet-152 model achieved the overall best performance with an average precision score of 89.4%. It also outperformed other SSD models in all other evaluation metrics. Table 6.2, shows that all models performed better when the IoU was 0.5 (AP_{50}) than when it was 0.75 (AP_{75}).

Meta-architecture	**Backbone**	**AP**	**AP_{50}**	**AP_{75}**	**AP_S**	**AP_M**	**AP_L**
SSD	MobileNet-V2	67.5%	96.1%	78.2%	19.0%	51.1%	73.5%
SSD	ResNet-50	73.3%	97.6%	84.8%	39.1%	58.5%	78.5%
SSD	ResNet-101	80.1%	98.4%	88.7%	40.7%	63.1%	88.3%
SSD	ResNet-152	**89.4%**	**98.5%**	**89.0%**	**41.3%**	**64.7%**	**92.1%**

Table 6.2: The evaluation metric scores of the SSD models built using MobileNet, ResNet-50, ResNet-101, and ResNet-152 feature extractors.The AP_{50} represent AP when IoU = 0.5 and AP_{75} means AP when IoU = 0.75. AP_S represents AP for small birds, AP_M and AP_L stands for AP of medium and large sized birds respectively.

Like AP, the highest AR was achieved in images with large and mid-sized birds compared to the images with small-sized birds. This happened across all the models meaning that the resolution of feature maps influences the performance of the networks. As shown in Table 6.3, the SSD model with ResNet-152 outperformed its counterparts in all the AR based evaluation metrics. Furthermore, the MS COCO evaluation tool also measures how well object detectors perform based on the number of objects per image. It categorizes this performance into three classes namely: AR_1, AR_{10}, AR_{100} representing AR given 1, 10, and 100 bird detections per image respectively. On these AR variants, all models performed well with 10 and 100 bird detections per image and worse for one bird detection per image.

Meta-architecture	**Backbone**	**AR_1**	**AR_{10}**	**AR_{100}**	**AR_S**	**AR_M**	**AR_L**
SSD	MobileNet-V2	31.6%	72.3%	73.7%	30.0%	61.1%	79.0%
SSD	ResNet-50	33.1%	77.2%	78.0%	40.9%	66.8%	82.8%
SSD	ResNet-101	40.0%	86.9%	87.8%	46.4%	76.4%	92.7%
SSD	ResNet-152	**42.1%**	**88.3%**	**87.9%**	**48.5%**	**78.0%**	**92.6%**

Table 6.3: Shows the comparison of Average Recall of the SSD models evaluated on different bird sizes and given number of bird detection per image.

The Figure 6.1 confirms what has been reported in recent works that very deep models produce better results (Zifeng et al., 2019; Yang et al., 2017). It is also noticed that after fine-tuning the models via transfer learning, the mean average precision increased across all the models. Thus, the mAP of the MobileNet, ResNet-50, ResNet-101, and ResNet-152 models increased by 68.3%, 68.1%, 57.7% and 68.4% respectively. This proves that transfer learning can be used to improve the accuracy of deep learning models as earlier stated by Yabuki et al. (2018) and Annegreet et al. (2014).

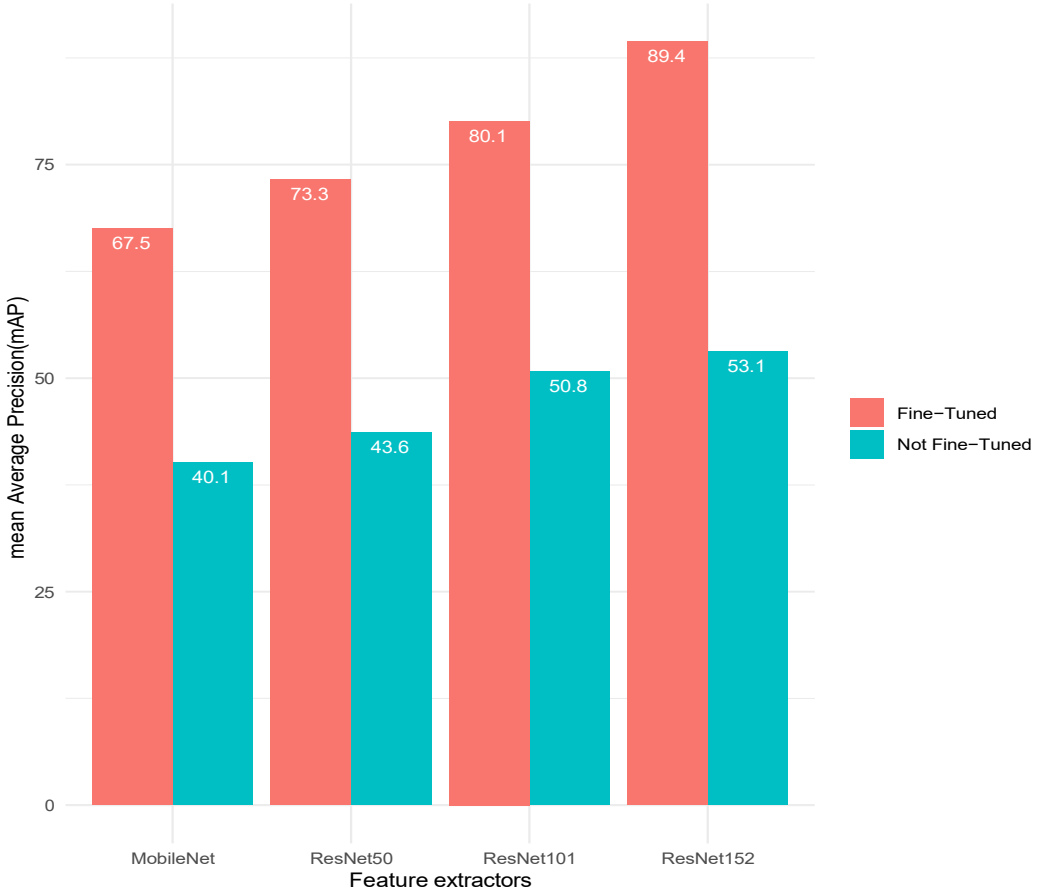


Figure 6.1: Summary of the mean average precision scores of the single short detector meta-architecture built using the MobileNet, ResNet-50, ResNet-101, and ResNet-152 feature extractors for both not fine-tuned and tuned models on the test set.

After training and evaluating the models, the associated inference graphs were extracted and used to perform bird detection on our test set. Figure 6.2 shows some of the examples of birds detected using the SSD best performing model of ResNet152. The performance is so outstanding that the model managed to detect birds of different bounding scales in most of the images in the test dataset.

Figure 6.2: Examples of correctly detected birds by the SSD ResNet152 model.

6.2 Faster R-CNN

Table 6.4 shows the detection performance of the Faster R-CNN meta-architecture combined with ResNet-50, ResNet-101, ResNet-152, and the Inception ResNet-v1 feature extractors. The performance of all the models in detecting both large and small birds was better than that of the SSD models. The model trained with the ResNet152 feature extractor yielded better results across all the evaluation metrics. It achieved an overall mean average precision of 92.3% but still, the other models also registered good performances as shown in Figure 6.3. It is clear that across all models the highest average precision was obtained when the IoU was set to 0.5 (AP_{50}) as opposed to 0.75 IoU.

Faster R-CNN models evaluation metrics									
Backbone Networks	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AR_S	AR_M	AR_L
ResNet-50	75.2	97.8	86.3	57.9	64.3	80.3	52.0	71.2	84.1
ResNet-101	90.4	98.5	86.6	58.4	69.1	91.0	63.8	74.0	90.0
ResNet-152	**92.3**	**98.6**	88.1	**60.0**	**70.8**	93.4	**64.9**	75.1	92.9
ResNet Inception-v2	80.3	98.5	**88.2**	56.0	70.2	**94.8**	64.1	**76.4**	**93.0**

Table 6.4: The summary evaluation metric scores of the individual Faster R-CNN models with different layer ResNet backbone networks after fine-tuning. Notably, all the models registered relatively high average precision and average recall across all the scales with the highest being in the large sized birds (AP_L and AR_L). This continues to confirm that feature map resolution influences the performance of object detectors.

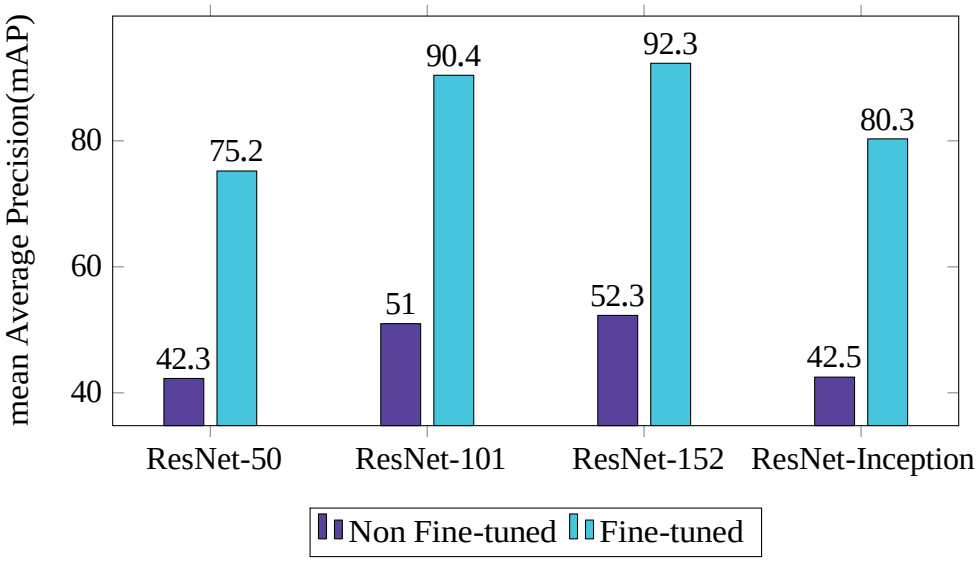


Figure 6.3: The mean Average Precision (mAP) of the Faster R-CNN meta-architecture built using the ResNet50, and ResNet-101, ResNet-152, and ResNet Inception-v2 feature extractors for both non fine-tuned and tuned models on the test set.

From Table 6.4, it can be observed that a Faster R-CNN model with Inception ResNet as the feature extractor outperformed its counterparts in detecting large birds but on the other hand it had the lowest AP for small birds. Looking at Figure 6.3, there was a significant improvement in the performance of all the models after fine-tuning and retraining them on our dataset. The mAP for Faster R-CNN with ResNet-50 increased by 77.8%. It increased by 72.2% and 76.5% when combined with ResNet-101 and ResNet-152. But it is also evident that even without fine-tuning, the MS COCO pre-trained models still give a fair performance.

6.3 Overall Evaluation

The overall results show that Faster R-CNN models achieved better results (in terms of detection accuracy) ranging from 75.2% to 92.3% than the SSD models whose range of detection accuracy was between 67.5% and 89.4%. In both meta-architectures, models trained with ResNet-152 feature extraction network achieved the highest mean average precision (see Figures 6.1 & 6.3). For the training time, Figure 6.4 shows that Faster R-CNN models trained faster than the SSD models except for the SSD with mobilenet which took the shortest training time of four hours. However, SSD models were remarkably faster than the Faster R-CNN models in terms of inference time (see Table 6.5). It is also observed in Table 6.5 that the deeper the backbone network, the slower it gets to train. For example, the ResNet-50 model is faster to train than ResNet-152 in both SSD and Faster R-CNN meta-architectures. This is because deeper networks need more parameters to learn (Ba and Caruana, 2013).

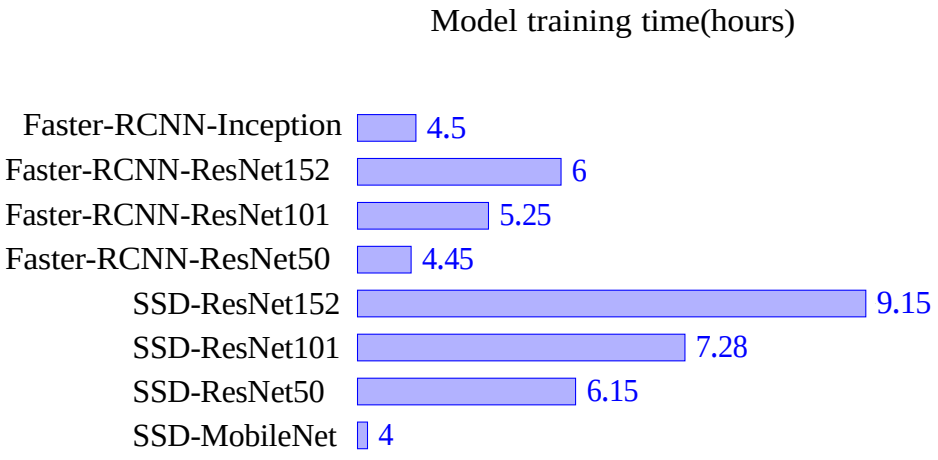


Figure 6.4: The time (in hours) taken to train the models with the different meta-architectures and feature extractors.

Table 6.5 shows that the SSD model with MobileNet feature extractor is the smallest in terms of inference graph file size compared to all other models. Therefore, this

Meta-architecture	Extractor	Model size (MB)	Inference time (ms/image)
SSD	MobileNet	**30.5**	**110**
	ResNet50	258	165
	ResNet101	405	183
	ResNet152	532	197
Faster R-CNN	ResNet50	221	251
	ResNet101	371	270
	ResNet152	495	283
	Inception ResNet-V2	469	256

Table 6.5: Shows the inference time per image on each meta-architecture and the models' parameter file sizes.

model can be deployed in memory-constrained systems because its light, fast (110ms) and registered quite a fair mAP of 67.5%. These results also show that SSD models consume more memory than the Faster R-CNN with the same feature extraction networks. For example, SSD combined with ResNet-152 consumes 37MB more than Faster R-CNN with the same base network.

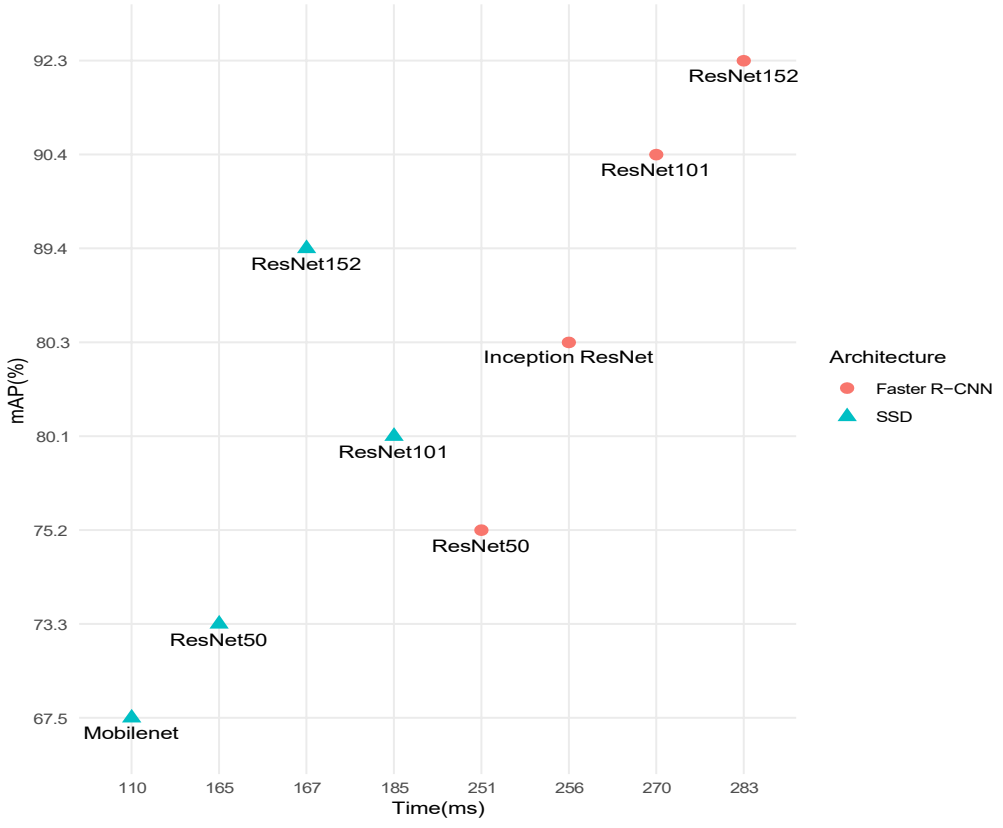
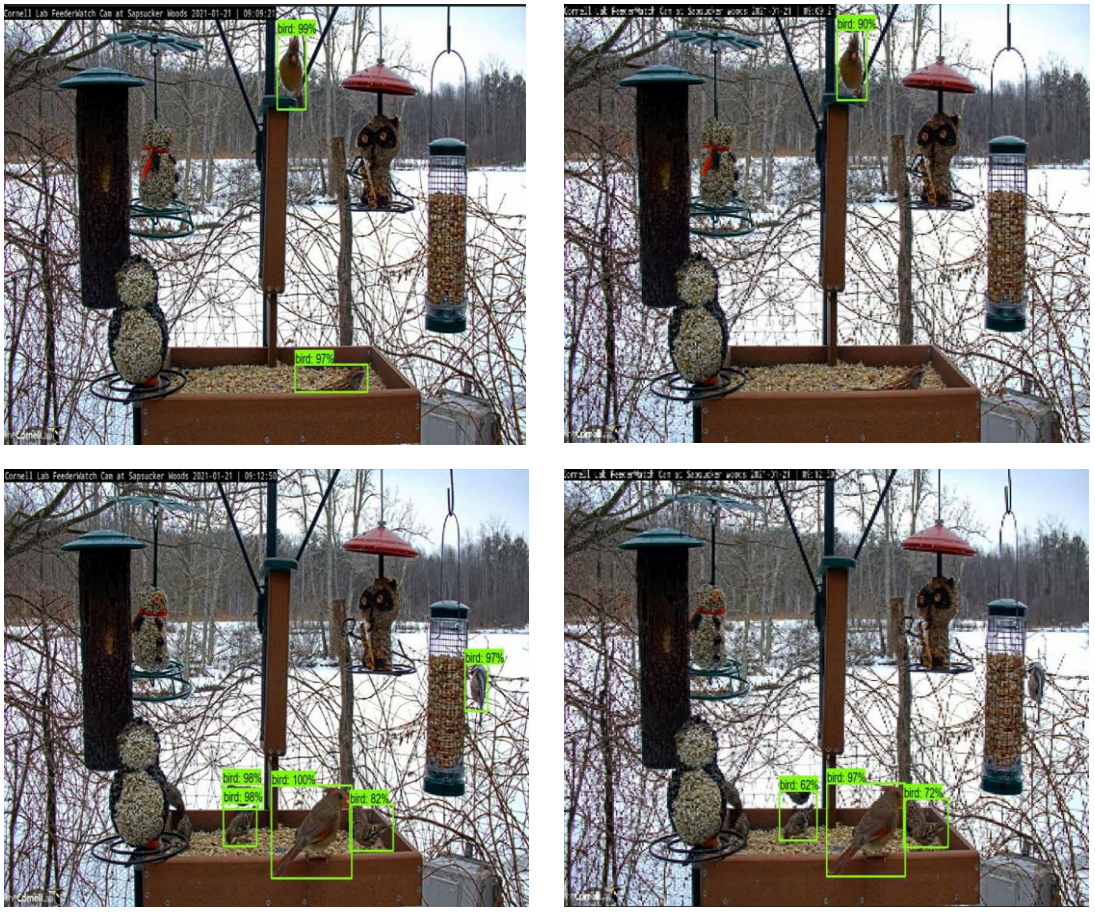


Figure 6.5: Shows a comparative performance between the different object detection models in terms of speed and accuracy.

Figure 6.5 demonstrates the trade-off in speed and accuracy. It is seen that Faster R-CNN models, in particular Faster R-CNN with ResNet101 and ResNet152, have the highest accuracy but their accuracy came at the expense of speed as they exhibited the worst testing time. In terms of speed, the SSD models proved to be faster than the Faster R-CNN models.

The Faster R-CNN models also outperformed the SSD models in detecting small and overlapped birds in images. In the Figure 6.6, we show some of the examples where the Faster R-CNN model did well in detecting the small and overlapped birds compared to the SSD .



(a) Faster R-CNN Prediction (b) SSD Prediction

Figure 6.6: Shows the sample of predictions on small and overlapping birds. It is seen that Faster R-CNN outperformed the SSD. When you look at the bottom images, the original image had 5 birds: the Faster R-CNN detected all the 5 birds but the SSD only managed to detect 3.

In Figure 6.7, we show random samples of images where the Faster R-CNN detected the birds with higher objectness scores than the SSD models. In this case, two models both built using ResNet101 feature extractor were compared. This continues

to confirm what was reported earlier that Faster R-CNN is better than SSD in terms of detection accuracy.

(a) Faster R-CNN ResNet101 (b) SSD ResNet101

Figure 6.7: Highlights the differences in the detection objectness scores of Faster R-CNN model with ResNet101 and SSD with the same extractor. The image has four birds and the objectness scores for the Faster R-CNN are 100%, 98%, 99%, 92% as opposed to 93%, 78%, 78%, 86% for the SSD model.

It was also observed that background affects the detection accuracy as all models failed to detect birds in images whose appearance is relatively similar to the background (Figure 6.8). This poor performance might have been caused by the limited number of images - in the training dataset - where the appearance of birds is similar to the background (uniform background images).

Figure 6.8: Shows birds that were not detected by all models (False Negatives). The birds' appearances are relatively similar to the background. The birds are not well distinguishable from the background.

The faster R-CNN model with the ResNet152 feature extractor- our best performing model, in terms of detection accuracy- was subjected to different images randomly obtained from Google with environmental conditions not captured in the test set images. This was done to determine how well the model would respond to these new domains. The model's performance was extremely good in detecting birds especially those taken at night/with dim light, images of birds with people, and images of birds captured at different angles. Therefore, this indicates that our model can be used to detect birds in different environment settings. In Figure 6.9, we show some of the detected images.

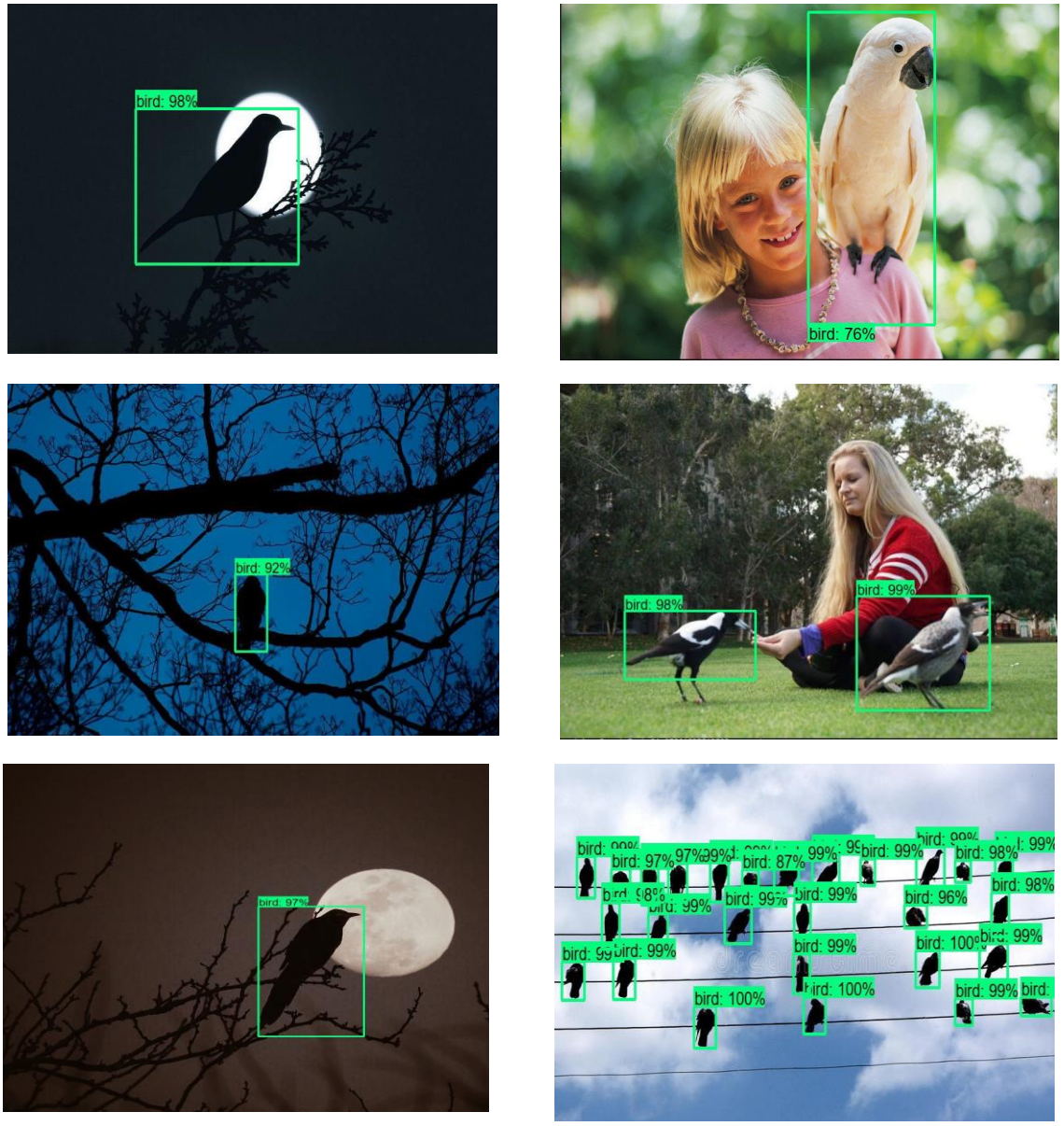


Figure 6.9: Birds detected in images captured at night/with dim light, images of birds with people, and different angle captured images.

The faster R-CNN model with the ResNet152 model was tested on images captured far from the cameras for example birds captured in the sky, and the detection performance was relatively good as seen in Figure 6.10. All this was done to determine how well the model can be deployed in different environments and camera angles.

(a) (b)

Figure 6.10: (a) An image of eight birds with six detected birds and two false positive results (b) Six well detected birds with higher confidence scores.

Chapter 7

Conclusion and Future Work

7.1 Summary and Conclusion

In recent years, wildlife conservation has received much attention because of the need to protect endangered species, estimate and monitor wildlife abundance, and generate the information necessary to make well-informed conservation and monitoring wildlife policies (Weinstein, 2018; Klug et al., 2017; Norouzzadeh et al., 2018). Due to technological advancement of equipment used in monitoring wildlife, ecologists are collecting large amounts of data that can no longer be analyzed using manual techniques and the traditional machine learning tools such as Support Vector Machine, Linear Discriminant Analysis, Principal Component Analysis because they quickly saturate whenever data volume increases (Moniruzzaman et al., 2017; Farley et al., 2018). Thanks to the development of deep learning which has successfully and predominantly been used to identify and count plant and animal species with outstanding results compared to the traditional machine learning techniques (Norouzzadeh et al., 2018; Rzanny et al., 2017; Tabak et al., 2019). Despite the achievements registered in recent years, there is still a need to improve the deep learning performance of detection in many ecological areas. Therefore, this thesis focused on studying, designing, and evaluating state-of-the-art deep learning object detection algorithms that are capable of detecting birds in webcam-captured images and to the best of our knowledge, this is the first study to use such images for deep learning modelling. In the following paragraphs we restate the objectives and in turn indicate the extent to which each of them was was addressed.

Objective 1: Collecting and annotating data for training, evaluating, and testing the algorithms.

Images used in the study were captured from the live feed watcher cams of Cornell Lab of Ornithology situated in 6 unique locations: three in United States, one in Panama, Ecuador, and the other in the Southeast tip of the New Zealand's South Island. The images were manually labeled using *Labelimg* annotation software tool. Since deep learning models require large amounts of data for better performance

(Dong and Fang, 2015; Anaby-Tavor et al., 2020), a total of 10592 images were collected. The task was laborious and took most of the research time. This objective was met in the sense that the required number of images to train the models were achieved. After using the dataset, it was contributed to an open source platform called Zenodo where those who wish to use it for research purposes are free to do so and it can be accessed at this link: https://zenodo.org/record/5172214#.YVTaQZpBxhH. Details on how this objective was attained is discussed in Section 5.1.

Objective 2: Identifying the best deep learning algorithms suitable for detecting birds of different sizes in several environment settings.

After obtaining knowledge about various object detection algorithms and application of machine learning in ecology through literature review (discussed in Chapter 2), Faster R-CNN (Ren et al., 2015) and SSD (Wei et al., 2016) meta-architectures combined with MobileNet-V2 (Howard et al., 2017), ResNet-50 (Kaiming et al., 2016b), ResNet-101 (Kaiming et al., 2016b) and ResNet-152 (Kaiming et al., 2016b), and Inception ResNet-v2 Christian et al. (2015) feature extraction networks were identified as the most suitable state-of-the-art techniques to perform bird detection. The literature also revealed that the use of transfer learning technique in deep learning algorithms yields outstanding performance in terms of accuracy and computational time. Transfer learning is a machine learning technique where knowledge obtained from one task is applied to other but related tasks (Athanasiadis et al., 2018). Encouraged by these observations, in this study, we explored the application of Faster R-CNN and SSD meta-architectures in combination with various feature extractors in the detection of birds from webcam images. The models were pre-trained on the MS COCO dataset provided by the TensorFlow object detection API. A comprehensive review of the theoretical background and architectures of these algorithms have been discussed and presented in chapters 3 and 4 respectively.

Objective 3: Evaluating the detection performance based on speed and accuracy of the selected algorithms.

Before training and evaluating the algorithms, the dataset was pre-processed. This involved splitting the dataset into training, validation, and testing sets in a ratio of 60:20:20 respectively. After dataset partitioning, TF-Record format files of the training and validation sets that are compatible with the Tensorflow object detection framework were created (Abadi et al., 2015). The training TF-record file was used to train the algorithms and the validation file was used for algorithm evaluation. All algorithms were initialized using weights pre-trained on the MS COCO dataset and fine-tuned on our dataset. Through several training and evaluating sessions, hyperparameters were fine-tuned until satisfactory results were attained. The dataset, pre-processing and implementation details of the algorithms are discussed fully in chapter 5. The code used to implement all the models is available at https://github.com/mirugwe1/bird_detection.git.

All models were evaluated using inference time metric to determine the detection speed of the and mean average precision metric for the accuracy. Based on the experimental test results, it has been concluded that between the two meta-architectures implemented in this study, Faster R-CNN showed the best bird detection performance. The Faster R-CNN model implemented with ResNet-152 feature extractor showed the best overall mAP of 92.3%. But for speed, the SSD models outperformed the Faster R-CNN models. The SSD model combined with a MobileNet feature extraction network is the fastest model with an inference time of 110ms per image. Therefore, if the main intention is about the accuracy, one should choose Faster R-CNN with ResNet-152 and if it is speed, SSD combined with MobileNet is the best choice. SSD with MobileNet is also the most suitable model to run on devices with small memory capacity as it was found to be the lightest in contrast with others.

Objective 4: Provide a comparison performance between these algorithms and conclude on which is the best method for speed and accuracy.

The algorithms were assessed and compared using mean average precision (mAP), recall, inference time, model size and training time evaluation metrics to enable the selection of the best performing algorithm. The results have been discussed with detailed analysis in chapter 6. For the training time, results showed that Faster R-CNN models were much faster compared to their SSD models counterparts except for the SSD model trained with MobileNet base network. The performance of SSD models in detecting small, overlapped, and obstructed birds was slightly lower than that of Faster R-CNN models. It was also observed that very deep based networks performed better than the shallow ones for both SSD and Faster R-CNN meta-architectures. Overall, we conclude that the Faster R-CNN combined with the ResNet-152 feature extractor as the best for achieving the highest mean average precision compared to other architectures, and the SSD with MobileNet as the best model in terms of speed and smaller memory consumption. The state-of-the-art results obtained in this study confirm that deep learning-based algorithms are capable of detecting birds of different sizes in different environments and we recommend that our best overall model can be used by ecologists in monitoring and identifying birds from other species.

7.2 Future Work

In future work, more data especially images with small birds and uniform backgrounds will be collected to address the poor performance achieved by the best model in detecting small birds and birds whose colour is uniform to the image background. The main objective of this study was to identify birds in webcam images, in future the scope of the models will be broadened beyond mere bird identification to individual bird recognition as Verstraeten et al. (2010) but with CNNs. This means the model will be capable of identifying and categorizing birds into their classes.

Furthermore, the focus of future study will the real-time implementation of bird detection in the webcam live streams. This will be coupled with a significant amount to improve the inference time without compromising the detection accuracy. Our fastest model (SSD with MobileNet) takes 100ms for bird detection per image which is clearly slow for a real-time monitoring application.

Bibliography

Abadi, Martın et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. url: http://tensorflow.org/.

Abdel-Hamid et al. (2014). "Convolutional Neural Networks for Speech Recognition". In: *IEEE/ACM Transactions on audio, speech, and language processing* 22.10, pp. 1533–1545.

Abhijit and Ghatak (2019). *Deep Learning with R*. Springer Singapore. doi: 10.1007/978-981-13-5850-0. url: https://doi.org/10.1007%2F978-981-13-5850-0.

Aggarwal, Charu C. (2018). "An introduction to neural networks". In: *Neural Networks and Deep Learning*. Springer, pp. 1–52.

Akçay, Hüseyin Gökhan et al. (2020). "Automated bird counting with deep learning for regional bird distribution mapping". In: *Animals* 10.7, p. 1207.

Akyildiz et al. (2002). "Wireless sensor networks: a survey". In: *Computer networks* 38.4, pp. 393–422.

Ali et al. (2012). "Random forests and decision trees". In: *International Journal of Computer Science Issues (IJCSI)* 9.5, p. 272.

Alpaydin, Ethem (2020). *Introduction to machine learning*. MIT press.

Anaby-Tavor, Ateret et al. (2020). "Do not have enough data? Deep learning to the rescue!" In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05, pp. 7383–7390.

Annegreet et al. (2014). "Transfer learning improves supervised image segmentation across imaging protocols". In: *IEEE transactions on medical imaging* 34.5, pp. 1018–1030.

Anthony et al. (2015). *Conservation and Management*. second. Vol. 16. Blackwell Publishing.

Ashrf, Nasef, Marjanović-Jakovljević Marina, and Angelina Njeguš (2017). "Stochastic gradient descent analysis for the evaluation of a speaker recognition". In: *Analog Integrated Circuits and Signal Processing* 90.2, pp. 389–397.

Athanasiadis et al. (2018). "A framework of transfer learning in object detection for embedded systems". In: *arXiv preprint arXiv:1811.04863*.

Ba, Lei Jimmy and Rich Caruana (2013). "Do deep nets really need to be deep?" In: *arXiv preprint arXiv:1312.6184*.

Barlow, Horace B. (1989). "Unsupervised learning". In: *Neural computation* 1.3, pp. 295–311.

Barré et al. (2017). "LeafNet: A computer vision system for automatic plant species identification". In: *Ecological Informatics* 40, pp. 50–56.

Basha et al. (2020). "Impact of fully connected layers on performance of convolutional neural networks for image classification". In: *Neurocomputing* 378, pp. 112–119.

Boudaoud et al. (2019). "Marine bird detection based on deep learning using high-resolution aerial images". In: *OCEANS 2019-Marseille*. IEEE, pp. 1–7.

Bradski, G. (2000). "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools*.

Cao et al. (2020). "Front vehicle detection algorithm for smart car based on improved SSD model". In: *Sensors* 20.16, p. 4646.

Chen et al. (2017). "Outlier detection with autoencoder ensembles". In: *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, pp. 90–98.

Chengqi, Zhang and Zhang Shichao (2003). *Association rule mining: models and algorithms*. Vol. 2307. Springer.

Chetlur et al. (2014). "cudnn: Efficient primitives for deep learning". In: *arXiv preprint arXiv:1410.0759*.

Chollet, François et al. (2015). *Keras*. https://keras.io.

Christian, Szegedy et al. (2015). “Going deeper with convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

Christin et al. (2019). “Applications for deep learning in ecology”. In: *Methods in Ecology and Evolution* 10.10, pp. 1632–1644.

Connor, Shorten and Taghi M Khoshgoftaar (2019). “A survey on image data augmentation for deep learning”. In: *Journal of Big Data* 6.1, pp. 1–48.

Curtis, Frank E and Katya Scheinberg (2017). “Optimization methods for supervised machine learning: From linear models to deep learning”. In: *Leading Developments from INFORMS Communities*. INFORMS, pp. 89–114.

Dahl et al. (2013). “Improving deep neural networks for LVCSR using rectified linear units and dropout”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, pp. 8609–8613.

Deng et al. (2020). “Region-based CNN method with deformable modules for visually classifying concrete cracks”. In: *Applied Sciences* 10.7, p. 2528.

Diederik, Kingma P and Ba Jimmy (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Ditria et al. (2020). “Automating the analysis of fish abundance using object detection: optimizing animal ecology with deep learning”. In: *Frontiers in Marine Science* 7, p. 429.

Donahue et al. (2013). “A deep convolutional activation feature for generic visual recognition”. In: *UC Berkeley & ICSI, Berkeley, CA, USA* 1.

Dong, Wang and Zheng Thomas Fang (2015). “Transfer learning for speech and language processing”. In: *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE, pp. 1225–1237.

Duchi, John, Elad Hazan, and Yoram Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7.

Dufourq, Emmanuel et al. (2021). “Automated detection of Hainan gibbon calls for passive acoustic monitoring”. In: *Remote Sensing in Ecology and Conservation*.

Elgendy, Mohamed (2020). *Deep Learning for Vision Systems*. Simon and Schuster.

Everingham et al. (2010). “The pascal visual object classes (voc) challenge”. In: *International journal of computer vision* 88.2, pp. 303–338.

Farley et al. (2018). “Situating ecology as a big-data science: current advances, challenges, and solutions”. In: *BioScience* 68.8, pp. 563–576.

Gary, White and Garrott Robert (2012). *Analysis of wildlife radio-tracking data*. Elsevier.

George, Seber AF and Lee J Alan (2012). *Linear regression analysis*. Vol. 329. John Wiley & Sons.

Gholamalinezhad, Hossein and Khosravi (2020). “Pooling Methods in Deep Neural Networks, a Review”. In: *arXiv preprint arXiv:2009.07485*.

Girshick, Ross et al. (2014). “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Girshick et al. (June 2014). “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Glenn, De’ath and Fabricius Katharina (2000). “Classification and regression trees: a powerful yet simple technique for ecological data analysis”. In: *Ecology* 81.11, pp. 3178–3192.

Glennie et al. (2015). “The effect of animal movement on line transect estimates of abundance”. In: *PloS one* 10.3, e0121333.

Godley et al. (2008). “Satellite tracking of sea turtles: where have we been and where do we go next?” In: *Endangered species research* 4.1-2, pp. 3–22.

Goldberg, David and Holland John Henry (1988). “Genetic algorithms and machine learning”. In: *Kluwer Academic Publishers-Plenum Publishers*, pp. 95–99.

Goodfellow et al. (2016). *Deep learning*. MIT press.

Guo et al. (2020). “Application of deep learning in ecological resource research: Theories, methods, and challenges”. In: *Science China Earth Sciences*, pp. 1–18.

He et al. (2019). “Depth-wise Decomposition for Accelerating Separable Convolutions in Efficient Convolutional Neural Networks”. In: *arXiv preprint arXiv:1910.09455*.

Hinton et al. (2012). “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal processing magazine* 29.6, pp. 82–97.

Hojjat, Adeli and Wu Mingyang (1998). “Regularization neural network for construction cost estimation”. In: *Journal of construction engineering and management* 124.1, pp. 18–24.

Hong et al. (2019). “Application of deep-learning methods to bird detection using unmanned aerial vehicle imagery”. In: *Sensors* 19.7, p. 1651.

Hosang et al. (July 2017). “Learning Non-Maximum Suppression”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Howard et al. (2017). “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861*.

Huang et al. (July 2017). “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hugo, Jachmann (2002). “Comparison of aerial counts with ground counts for large African herbivores”. In: *Journal of Applied Ecology* 39.5, pp. 841–852.

— (2012). *Estimating abundance of African wildlife: an aid to adaptive management*. Springer Science & Business Media.

Hui, Jonathan (2018). *mAP (mean Average Precision) for Object Detection*. url: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

Hung, Nguyen et al. (2017). “Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring”. In: *2017 IEEE international conference on data science and advanced Analytics (DSAA)*. IEEE, pp. 40–49.

Jain et al. (1996). “Artificial neural networks: A tutorial”. In: *Computer* 29.3, pp. 31–44.

Jang et al. (2020). “Propose-and-attend single shot detector”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 815–824.

Jarrett et al. (2009). “What is the best multi-stage architecture for object recognition?” In: *2009 IEEE 12th international conference on computer vision*. IEEE, pp. 2146–2153.

Jason, Perez and Wang Luis (2017). “The effectiveness of data augmentation in image classification using deep learning”. In: *arXiv preprint arXiv:1712.04621*.

Jia, Deng et al. (2009). “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Junhwan, Ryu and Kim Sungho (2019). “Chinese Character Boxes: Single Shot Detector Network for Chinese Character Detection”. In: *Applied Sciences* 9.2, p. 315.

Kaiming, He et al. (2016a). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

— (2016b). “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer, pp. 630–645.

Kandel, Ibrahem and Castelli Mauro (2020). “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset”. In: *ICT express* 6.4, pp. 312–315.

Karnin, Ehud D. (1990). “A simple procedure for pruning back-propagation trained neural networks”. In: *IEEE transactions on neural networks* 1.2, pp. 239–242.

Karpathy, Andrej and Li Fei-Fei (2015). “Deep visual-semantic alignments for generating image descriptions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137.

Kays et al. (2010). “Monitoring wild animal communities with arrays of motion sensitive camera traps”. In: *arXiv preprint arXiv:1009.5718*.

Khaire, Utkarsh Mahadeo and R Dhanalakshmi (2020). “High-dimensional microarray dataset classification using an improved adam optimizer (iAdam)”. In: *Journal of Ambient Intelligence and Humanized Computing* 11.11, pp. 5187–5204.

Kim et al. (2016). “On-road object detection using deep neural network”. In: *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, pp. 1–4.

Klaus-Robert, Müller et al. (2012). “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.

Klug et al. (2017). “Analysis of high-frequency and long-term data in undergraduate ecology classes improves quantitative literacy”. In: *Ecosphere* 8.3, e01733.

Knight et al. (2017). “Recommendations for acoustic recognizer performance assessment with application to five common automated signal recognition programs”. In: *Avian Conservation and Ecology* 12.2.

Knudby et al. (2010). “New approaches to modelling fish–habitat relationships”. In: *Ecological Modelling* 221.3, pp. 503–511.

Koniusz et al. (2016). “Higher-order occurrence pooling for bags-of-words: Visual concept detection”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.2, pp. 313–326.

Kramer, Oliver (2013). “K-nearest neighbors”. In: *Dimensionality reduction with unsupervised nearest neighbors*. Springer, pp. 13–23.

Krizhevsky et al. (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25, pp. 1097–1105.

Kumar et al. (2020). “Object detection in real time based on improved single shot multi-box detector algorithm”. In: *EURASIP Journal on Wireless Communications and Networking* 2020.1, pp. 1–18.

Kunihiko, Fukushima and Miyake Sei (1982). “Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position”. In: *Pattern recognition* 15.6, pp. 455–469.

LeCun and Yoshua Bengio (1995). “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

LeCun and Cortes Corinna (2010). “MNIST handwritten digit database”. In: url: http://yann.lecun.com/exdb/mnist/.

Lee et al. (2006). “Automatic recognition of animal vocalizations using averaged MFCC and linear discriminant analysis”. In: *pattern recognition letters* 27.2, pp. 93–101.

Leshno et al. (1993). “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6, pp. 861–867.

Likas et al. (2003). “The global k-means clustering algorithm”. In: *Pattern recognition* 36.2, pp. 451–461.

Lin et al. (2014). “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer, pp. 740–755.

Liu et al. (2018). “Performance comparison of deep learning techniques for recognizing birds in aerial images”. In: *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, pp. 317–324.

Long et al. (2015). “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Loshchilov, Ilya and Frank Hutter (2017). “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101*.

Lowe and David (1999). “Object recognition from local scale-invariant features”. In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee, pp. 1150–1157.

Lu et al. (2017). “No need to worry about adversarial examples in object detection in autonomous vehicles”. In: *arXiv preprint arXiv:1707.03501*.

Maas et al. (2013). “Rectifier nonlinearities improve neural network acoustic models”. In: vol. 30. 1. Stanford University Press, p. 3.

Major et al. (2019). “Vehicle detection with automotive radar using deep learning on range-azimuth-doppler tensors”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 924–932.

Marco, Vicent Sanz et al. (2020). “Optimizing deep learning inference on embedded systems through adaptive model selection”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 19.1, pp. 1–28.

Markus, Goldstein and Uchida Seiichi (2016). “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data”. In: *PloS one* 11.4, e0152173.

Marr, Bernard (2015). “Big Data: 20 mind-boggling facts everyone must read”. In: *Forbes magazine*.

Merz (1986). “Counting elephants (Loxodonta africana cyclotis) in tropical rain forests with particular reference to the Tai National Park, Ivory Coast”. In: *African Journal of Ecology* 24.2, pp. 61–68.

Mikołajczyk, Agnieszka and Michał Grochowski (2018). “Data augmentation for improving deep learning in image classification problem”. In: *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, pp. 117–122.

Mohri et al. (2018). *Foundations of machine learning*. MIT press.

Moniruzzaman et al. (2017). “Deep learning on underwater marine object detection: A survey”. In: *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, pp. 150–160.

Nguyen et al. (2020). “An evaluation of deep learning methods for small object detection”. In: *Journal of Electrical and Computer Engineering* 2020.

Nielsen, Michael A. (2015). *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA.

Nitish, Srivastava and Kevin Swersky (2012). *Lecture 6d- a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning*.

Norouzzadeh et al. (2018). “Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning”. In: *Proceedings of the National Academy of Sciences* 115.25, E5716–E5725. issn: 0027-8424. doi: 10.1073/pnas.1719367115. eprint: https://www.pnas.org/content/115/25/E5716.full.pdf. url: https://www.pnas.org/content/115/25/E5716.

Nwankpa et al. (2018). “Activation functions: Comparison of trends in practice and research for deep learning”. In: *arXiv preprint arXiv:1811.03378*.

O’Connell et al. (2010). *Camera traps in animal ecology: methods and analyses*. Springer Science & Business Media.

Omar, Javed and Shah Mubarak (2002). “Tracking and object classification for automated surveillance”. In: *European Conference on Computer Vision*. Springer, pp. 343–357.

Ostertagová, Eva (2012). “Modelling using polynomial regression”. In: *Procedia Engineering* 48, pp. 500–506.

Padilla et al. (2020). “A survey on performance metrics for object-detection algorithms”. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, pp. 237–242.

Prechelt, Lutz (1998). “Early stopping-but when?” In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Recknagel, Friedrich (2001). “Applications of machine learning to ecological modelling”. In: *Ecological modelling* 146.1-3, pp. 303–310.

Redmon, Joseph Ali, and Farhadi (2018). “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767*.

Redmon, Farhadi (July 2017). “YOLO9000: Better, Faster, Stronger”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Redmon, Joseph, Santosh Divvala, et al. (2016). “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Redmon, Joseph and Ali Farhadi (2017). “YOLO9000: better, faster, stronger”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271.

Ren et al. (2015). “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28, pp. 91–99.

Rifai et al. (2011). “Contractive auto-encoders: Explicit invariance during feature extraction”. In: *Universite de Montreal Press*.

Root, Terry and Schneider (2002). “Climate change: overview and implications for wildlife”. In: *Wildlife responses to climate change: North American case studies* 10.2002, pp. 765–766.

Rosa et al. (2016). “Classification success of six machine learning algorithms in radar ornithology”. In: *International Journal of Avian Science* 158.1, pp. 28–42.

Ross, Girshick (2015). “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Ross, Girshick and Donahue Jitendra (2016). “Region-based convolutional networks for accurate object detection and segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 38.1, pp. 142–158.

Ruder, Sebastian (2016). “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747*.

Russakovsky et al. (2015). “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3, pp. 211–252.

Rzanny et al. (2017). “Acquiring and preprocessing leaf images for automated plant identification: understanding the tradeoff between effort and information gain”. In: *Plant methods* 13.1, pp. 1–11.

Sa, Yang (2014). “Improved bilinear interpolation method for image fast processing”. In: *2014 7th International Conference on Intelligent Computation Technology and Automation*. IEEE, pp. 308–311.

Sackinger, Eduard et al. (1995). “Comparison of learning algorithms for handwritten digit recognition”. In: *International conference on artificial neural networks*. Vol. 60. Perth, Australia, pp. 53–60.

Sagar, Sharma and Sharma Simone (2017). “Activation functions in neural networks”. In: *Towards Data Science* 6.12, pp. 310–316.

Sarle, Warren S. (1994). “Neural networks and statistical models”. In:

Scherer et al. (2010). “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *International conference on artificial neural networks*. Springer, pp. 92–101.

Schneider et al. (2018). “Deep learning object detection methods for ecological camera trap data”. In: *2018 15th Conference on computer and robot vision (CRV)*. IEEE, pp. 321–328.

Scott, Menard (2002). *Applied logistic regression analysis*. Vol. 106. Sage.

Sergey, Ioffe and Szegedy Christian (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR, pp. 448–456.

Silver et al. (2017). “Mastering the game of go without human knowledge”. In: *nature* 550.7676, pp. 354–359.

Simonyan, Karen and Andrew Zisserman (2014). “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556*.

Song, Zheng et al. (2011). “Contextualizing object detection and classification”. In: *CVPR 2011*. IEEE, pp. 1585–1592.

Srivastava et al. (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Sutton, Richard S and Andrew Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Swanson et al. (2015). “Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna”. In: *Scientific data* 2.1, pp. 1–14.

Szegedy et al. (2016). “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Szewczyk et al. (2004). “An analysis of a large scale habitat monitoring application”. In: *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 214–226.

Tabak et al. (2019). “Machine learning to classify animal species in camera trap images: Applications in ecology”. In: *Methods in Ecology and Evolution* 10.4, pp. 585–590.

Takeki et al. (2016). “Detection of small birds in large images by combining a deep detector with semantic segmentation”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE, pp. 3977–3981.

Tang et al. (2017). “Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero”. In: *Control Theory & Applications* 34.12, pp. 1529–1546.

Tzuta, Lin (Dec. 2018). *tzutalin/labelImg*. url: https://github.com/tzutalin/labelImg.

Uijlings, Jasper RR et al. (2013). “Selective search for object recognition”. In: *International journal of computer vision* 104.2, pp. 154–171.

Vahab et al. (2019). “Applications of object detection system”. In: *International Research Journal of Engineering and Technology (IRJET)* 6.4, pp. 4186–4192.

Van-Der-Maaten et al. (2009). “Dimensionality reduction: a comparative”. In: *J Mach Learn Res* 10.66-71, p. 13.

Vasilev, Ivan et al. (Jan. 2019). *Python Deep Learning*. 2nd ed. Vol. 2. Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow. Birmingham: Packt Publishing. isbn: 978-1-78934-846-0.

Verstraeten, Willem W et al. (2010). “Webcams for bird detection and monitoring: A demonstration study”. In: *Sensors* 10.4, pp. 3480–3503.

Villa, Alexander et al. (2017). “Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks”. In: *Ecological informatics* 41, pp. 24–32.

Vinod, Nair and Hinton E Geoffrey (2010). “Rectified linear units improve restricted boltzmann machines”. In: *Icml*.

Wang et al. (2017). “Automatic image-based plant disease severity estimation using deep learning”. In: *Computational intelligence and neuroscience* 2017.

Wei, Liu et al. (2016). “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer, pp. 21–37.

Weinstein, Ben G. (2018). “A computer vision for animal ecology”. In: *Journal of Animal Ecology* 87.3, pp. 533–545.

Werbos, Paul J (1990). “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Wilber et al. (2013). “Animal recognition in the mojave desert: Vision tools for field biologists”. In: *2013 IEEE Workshop on Applications of Computer Vision (WACV)*. IEEE, pp. 206–213.

William, Noble S (2006). “What is a support vector machine?” In: *Nature biotechnology* 24.12, pp. 1565–1567.

Wong et al. (2016). “Understanding data augmentation for classification: when to warp?” In: *2016 international conference on digital image computing: techniques and applications (DICTA)*. IEEE, pp. 1–6.

Xin, Qian and Klabjan Diego (2020). “The impact of the mini-batch size on the variance of gradients in stochastic gradient descent”. In: *arXiv preprint arXiv:2004.13146*.

Xu et al. (2015). “Show, attend and tell: Neural image caption generation with visual attention”. In: *International conference on machine learning*. PMLR, pp. 2048–2057.

Yabuki et al. (2018). “Automatic object detection from digital images by deep learning with transfer learning”. In: *Workshop of the European Group for Intelligent Computing in Engineering*. Springer, pp. 3–15.

Yang et al. (2017). “Does resnet learn good general purpose features?” In: *Proceedings of the 2017 International Conference on Artificial Intelligence, Automation and Control Technologies*, pp. 1–5.

Yann et al. (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.

Yao et al. (2018). “Large batch size training of neural networks with adversarial training and second-order information”. In: *arXiv preprint arXiv:1810.01021*.

Zeiler et al. (2013). “On rectified linear units for speech processing”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 3517–3521.

Zifeng et al. (2019). “Wider or deeper: Revisiting the resnet model for visual recognition”. In: *Pattern Recognition* 90, pp. 119–133.

Zou et al. (2019). “Object detection in 20 years: A survey”. In: *arXiv preprint arXiv:1905.05055*.