Convolutional and fully convolutional neural networks for the detection of landmarks in tsetse fly wing images

Mulanga Makhubele

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

Supervisor: Prof. Willie Brink

December 2021

Declaration

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

Mulanga Makhubele .....

December 2021

Abstract

Tsetse flies are a species of bloodsucking flies in the housefly family, that are only found in Africa. They cause animal and human African trypanosomiasis (AAT and HAT), commonly referred to as nagana and sleeping sickness. Effective tsetse fly eradication requires area-wide control, which means understanding the population dynamics of the tsetse flies in an area. Among the factors that entomologists believe to be critical to this understanding, fly size and fly wing shape are considered most important. Fly size can be deduced by calculating the distance between specific landmarks on a wing. The South African Centre for Epidemiological Modelling and Analysis (SACEMA) conducts research into tsetse fly population management and have a database of wings. To use landmarks on the wings for biological deductions about the tsetse flies in the area, researchers will need to manually annotate individual images of the wings by marking the important landmarks by hand, which is slow and error-prone.

The purpose of this research is to assess the feasibility of automating the process of landmark detection in tsetse fly wing images using machine learning algorithms with a limited dataset. Extensive research has been done into automatic landmark detection. Particular focus has been given to detection of human body parts but there are a number of notable cases of animal landmark detection. Convolutional neural networks (CNNs) have been used as backbone architectures for most state-of-the-art detection systems. We compare the performance of fully convolutional networks (FCNs) against conventional LeNet style CNNs for the regression task of landmark detection in a fly wing image. The FCN accepts an image input and returns a segmentation mask as output. A Gaussian function is used to convert the response coordinate pairs into heat maps, which are combined to form a segmentation mask. After model training the heat maps produced by the FCN model are converted back to coordinate pairs using a weighted average method.

Three types of models were trained: a baseline artificial neural network (ANN), LeNet style CNNs and FCNs. The ANN model had a root mean square error (RMSE) of 282.62 pixels and mean absolute error (MAE) of 181.33 pixels. The best LeNet model, LeNet3 with dropout, had an RMSE of 53.58 and MAE of 41.05. The best FCN model FCN8 with batch size 32 and Adam optimization, had an RMSE of 1.12 and MAE of 0.88. All trained models were best at predicting landmark points 5, 8 and 10 and struggled to predict landmark points 1, 4 and 6.

The results indicate that machine learning models can be used to automatically and accurately detect landmark points on tsetse fly wing images. Furthermore, for our limited dataset FCNs outperform conventional LeNet style CNNs.

Opsomming

Tsetsevlieë is 'n spesie bloedsuiende vlieë in die huisvliegfamilie, wat slegs in Afrika voorkom. Hulle veroorsaak meestal dierlike en menslike Afrika-trypanosomiasis, ook bekend as nagana en slaapsiekte. Effektiewe uitroei van tsetsevlieë benodig beheer van 'n hele gebied, wat beteken dat die bevolkingsdinamika van die tsetsevlieë in 'n gebied verstaan word. Onder die faktore wat vir entomoloë van kritieke belang is vir hierdie begrip, is die grootte van die vlieg en vliegvlerkvorm van die belangrikste. Vlieggrootte kan afgelei word deur die afstand tussen spesifieke landmerke op 'n vlerk. Die Suid-Afrikaanse Sentrum vir Epidemiologiese Modellering en Analise (SACEMA) doen navorsing oor die bestuur van tsetsevliegpopulasies en het 'n databasis van vlerke. Om landmerke op die vlerke te gebruik vir biologiese afleidings oor die tsetsevlieë in die omgewing, sal navorsers individuele beelde van vlerke moet annoteer deur die belangrike landmerke met die hand te merk, wat stadig en foutief kan wees.

Die doel van hierdie navorsing is om die uitvoerbaarheid van die outomatisering van landmerkopsporing in tsetsevliegvlerkbeelde, met behulp van masjienleeralgoritmes op 'n beperkte datastel, te ondersoek. 'n Uitgebreide ondersoek is gedoen na outomatiese opsporing van landmerke. Spesifieke fokus is gegee aan die opsporing van menslike liggaamsdele, maar daar is 'n aantal gevalle van landmerkopsporing in diere. Konvolusionele neurale netwerke (CNN's) word as ruggraat-argitektuur vir die meeste moderne opsporingstelsels gebruik. Ons vergelyk die prestasie van volledig-konvolusionele netwerke (FCN's) teen konvensionele LeNet-styl CNN's, vir die regressietaak van landmerkopsporing in 'n vliegvlerkbeeld. Die FCN aanvaar 'n beeld as toevoer en gee 'n segmenteringsmasker as uitvoer. 'n Gaussiese funksie word gebruik om die responskoördinate om te skakel na hittebeelde, wat saamgevoeg word om 'n segmenteringsmasker te vorm. Na modelafrigting word die hittekaarte wat deur die FCN-model vervaardig word, omgeskakel na koördinaatpare met behulp van 'n geweegde gemiddeldes.

Drie soorte modelle is afgerig: 'n basiese kunsmatige neurale netwerk (ANN), LeNet-styl CNN's en FCN's. Die ANN-model het 'n wortel-gemiddelde vierkantfout (RMSE) van 282.62 piksels en gemiddelde absolute fout (MAE) van 181.33 piksels. Die beste LeNet-model, LeNet3 met uitval, het 'n RMSE van 53.58 en MAE van 41.05 gehad. Die beste FCN-model, FCN8 met groepgrootte 32 en Adam-optimering, het 'n RMSE van 1.12 en MAE van 0.88 gehad. Alle afgerigte modelle het landmerke 5, 8 en 10 die beste vind, en gesukkel om landmerkpunte 1, 4 en 6 te vind.

Die resultate dui aan dat masjienleermodelle outomaties en akkuraat gebruik kan word om landmerkpunte op tsetsevliegvlerkbeelde op te spoor. Verder, vir ons beperkte datastel het FCN's beter presteer as konvensionele LeNet-styl CNN's.

Acknowledgements

This thesis has been a labour of love for myself and all the women who doubt their place in this industry.

I would like to thank God for the divine help He has given me during the writing of this thesis. I thank my parents who have encouraged me to always pursue my dreams, never seeing my gender as a limitation but always pushing me to strive for greatness in all I do.

I would like to thank my husband Marvel Makhubele for being my anchor and cheerleader, always pushing me to keep going when things got tough. Thank you to my supervisor Willie Brink for the guidance, patience and support.

Finally, I would like to thank my daughter miss Shiloh Wandeme Uriel Makhubele for her patience with mommy. For being a light and making the world so much brighter with her presence, baby girl this thesis is for you. Mommy wants you to know that nothing is beyond reach for you if you choose to pursue it with all your might. Greatness is within reach for you and you deserve to be great.

Contents

1 Introduction

Tsetse flies are a species of bloodsucking flies in the housefly (Muscidae) family, that are only found in Africa. They cause animal and human African trypanosomiasis (AAT and HAT), commonly referred to as nagana and sleeping sickness, respectively. AAT is transmitted through the bite of a tsetse fly and causes infected animals to experience high fevers, anaemia and weakness which ultimately lead to fertility and milk production issues and sometimes death. Domestic animals such as cattle, horses, sheep and goats are most affected by AAT. In Sub-Saharan Africa the agricultural sector has recorded an estimated $3 billion of animal productivity losses as a result of AAT. HAT is transmitted when tsetse flies bite humans and an estimated 1.59 million lives have been lost due to ill health, disability or early death from this disease [68, 107, 15].

The large threat that tsetse flies pose on human and animal life means that control of these species is crucial. There are a number of methods being used to control the spread of AAT, including the use of trypanocidal drug compounds which are used both prophylactically (for prevention) or curatively (for cure), the promotion of breeding livestock that are trypanotolerant (able to withstand the effects of the disease) and finally the eradication of the tsetse fly which is the transmission vector. The drug compounds and the promotion of trypanotolerant animals are not always effective due to the increased drug resistance of the AAT parasite. As such, eradication of the tsetse fly is considered the preferred way of managing the disease.

Effective tsetse fly eradication requires area-wide control, which means understanding the biology of the tsetse flies in an area and making the controls biologically relevant, as well as ensuring that isolated tsetse populations are targeted. Without these considerations, re-invasion is likely to occur.

Medical entomology is a field of medicine interested in distinguishing species and detecting those species even in places where they may not be expected. Among the factors that entomologists believe to be critical to understanding the biology of tsetse flies in an area, fly size and fly wing shape are considered most important.

According to Mbewe et al. [68] and Hargrove et al. [41], fly size can be deduced using certain wing length measurements. In Figure 1 the distance between A and B as well as the distance between C and D can be used to estimate the size of the fly. In this thesis we will refer to points A, B, C, D and other points where the fly veins intersect as landmark points.

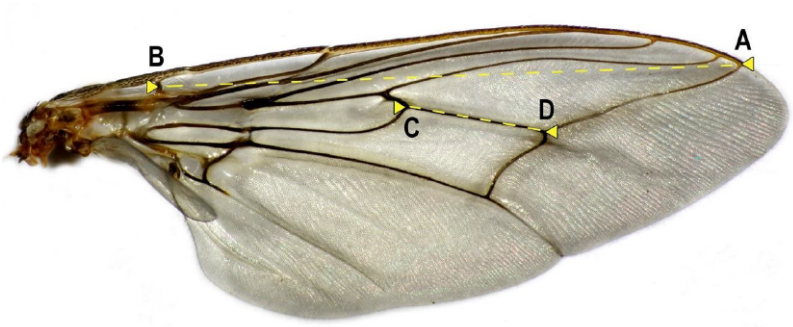


Figure 1: Right facing tsetse fly wing with landmarks A, B, C and D, and the different landmark distances that can be used to estimate the fly size [68].

1.1 Research motivation

The South African Centre for Epidemiological Modelling and Analysis (SACEMA) conducts research into tsetse fly population management. To that end, SACEMA has collected thousands of wings and has taken photographs of a subset of these wings to form a database for research purposes. Figure 2 shows four examples each of left facing and right facing wing images from the SACEMA database. To use these images and landmarks on the wings for biological deductions about the tsetse flies in the area, researchers need to manually annotate the images by marking the important landmarks by hand. Annotating landmark points manually is a slow error-prone process which might significantly delay research efforts.

Figure 2: Four left-facing and four right-facing wing images from the SACEMA database.

A landmark detection algorithm can be trained using machine learning to receive an image and automatically detect (or localize) the landmarks of interest, which would be a first step in the process of measuring fly size using wing lengths. After these landmarks are found, distances between them can be calculated which will help us to estimate the size of the fly from which the wing was obtained. This speeds up the research timeline and frees up researchers to do more important research work.

1.2 Literature review

A fair amount of research has been conducted in the computer vision community over the last few decades on landmark detection [38, 77]. For many important computer vision applications, such as human body pose estimation, 3D visualization and face recognition, the accurate detection of meaningful landmarks or key points forms an important precursor [122].

We consider some of the most popular landmark detection tasks that have been researched, focusing on human as well as animal body parts.

1.2.1 Detection of human body parts

The detection of human body parts has been widely researched, with researchers looking at topics that include hand tracking [45, 25], gesture recognition [24], facial expression recognition [51], face identity verification [106, 102], eye gaze tracking [126, 70], hand keypoint localization, facial keypoint detection [2] as well as human pose estimation [73].

The challenge of detecting facial landmarks is that facial features may vary greatly from one image to another due to differences in individuals' appearances. Facial features may also be affected by other physical factors such as face position, viewing angle, illumination and contrast.

Facial keypoint detection

The aim of facial keypoint detection algorithms is to automatically locate pre-defined key landmarks in facial images or videos. The types of algorithms and the detection problems these can solve have evolved over time.

Initial research work was aimed at working with very distinct and well-defined facial images, in an attempt to remove any kind of variation. This later evolved to deal with several variation categories that were still relatively controlled. For example, subjects in facial images that were collected for these controlled conditions could only face one direction and their expression had to be the same. Variations arose from the fact that the model was trained on different faces and the lighting on the images varied slightly. More recent research has focused on what has been termed "in-the-wild" conditions, where the types of facial images used to train models have different head poses, the images are illuminated differently, and the facial expressions also vary.

The first facial keypoint detection methods were based on active appearance models (AAMs) [29, 22, 86]. Variants of this approach focused on improving fitting algorithms [67, 6, 27, 43, 92] and feature representation by using more robust image features [44, 47]. The next wave of solutions was regression-based [23, 118, 103, 127] that directly map from image feature space to landmark locations, and have been found to perform better than PCA based AAM models and their variants [122, 115]. Of all the regression-based methods, the cascaded regression methods (CRMs) [18, 84, 55, 117] seem to be the most widely used.

Recent work in facial keypoint detection has focused on deep learning-based approaches with the use of convolutional neural networks (CNNs). Longpre et al. [66] experimented with LeNet style, VGG style and weighted ensembles of the two model types. Naimish et al. [2] used LeNet style CNNs and solved the keypoint recognition problem by building a separate CNN for every facial keypoint. Cascaded CNNs and recurrent neural networks (RNNs) have also been used [101] where three CNN levels are cascaded to make coarse to fine predictions. This focus on deep learning networks is largely due to the end-to-end nature of training deep neural networks, i.e. human supervision is not required for feature transformation design [111], which

has significantly improved performance over the hand-crafted features that previous methods utilized [122, 86].

Human body pose estimation

Another widely studied area that incorporates landmark detection on human body parts is the area of human body pose estimation. The aim of human body pose estimation algorithms is to automatically identify key locations on the human body in an image or video, such as joints and body parts of interest to a particular application. Tracking the pose and limb positions of the human body is a key building block for tasks such as action classification and body movement prediction, and is a fundamental tool in the human-computer interaction and animation fields [73, 8].

Early research into human body pose estimation made use of models called pictorial structures [49, 82]. In pictorial structures, spatial correlations of body parts are presented as tree-structured graphical models. When the limbs are fully visible these models are highly successful, but performance declines when limbs are occluded. Early research also made use of hand-crafted features such as the histogram of oriented gradients (HOG) features, edges, colour histograms and contours [119, 110]. These models performed well on the datasets and scenarios they were designed for but failed to generalize to new datasets [30].

More recent human pose estimation systems have used CNNs as architectural building blocks. For example, Fan et al. [84] used a region-based CNN (R-CNN) as the main building block, with a dual source CNN model (DS-CNN) that uses both the full body appearance and the holistic view of the local body part as model inputs. VGG [98] is a popular backbone architecture and was used by Wei et al. [113] to design a sequential convolutional network architecture that operates on belief maps from preceding stages. Su et al. [100] presented a technique called cascade feature aggregation (CFA), which cascades a number of hourglass networks for robust human pose estimation. They largely made use of ResNet-50 [42], Resnet-101 and Resnet-152 as backbone architectures. This strategy led to significant improvements on standard benchmark datasets [91, 4].

Research has moved away from single body human pose estimation in controlled environments to multi-person human pose estimation in the wild. Approaches in this area can be classified into bottom-up and top-down approaches. Bottom-up approaches [75, 80] first detect all the landmarks of every person in an image and then group these landmarks in a way that helps to identify the individuals in the image. Top-down methods [79, 104] first detect bounding boxes and then predict the human body landmarks in each box [48].

1.2.2 Detection of animal landmarks

The detection of landmarks in animal images has not been as extensively researched for various types of animals. Bird part localization [19], fine-grained classification as well as the topic of animal re-identification are some of the more popular research areas.

Fine-grained image classification

The bird part localization problem is commonly addressed to solve a particular instance of the problem of visual fine-grained classification. While general classification seeks to find distinctions between different categories of classes, e.g. dogs vs cats, fine-grained categorization goes a level deeper to find distinctions between different subclasses, e.g. British Shorthair vs Persian cats [125]. Fine-grained image classification approaches in the deep learning literature can be

classified into those that use (1) end-to-end deep neural networks for fine-grained classification, (2) deep neural networks to extract features that make part localization easier, (3) an ensemble of neural networks to differentiate fine-grained classes that are highly similar, and (4) visual attention to distinguish between image regions most useful for classification.

According to Zhao et al. [128], state-of-the-art CNNs such as AlexNet, VGG and GoogLeNet can be adopted for fine-grained image classification.

Where there may be abstruse differences in individual object body parts, semantic part localization can be utilized to isolate these differences and help facilitate fine-grained classification. Berg et al. [11], Liu et al. [62], Yang et al. [120] and Gavves et al. [34] are examples of approaches where semantic part localization is used in this manner.

The approach that uses an ensemble of neural networks to differentiate fine-grained classes that are highly similar involves the use of multiple neural networks to create one combined architecture that will improve performance. Some examples are subset feature learning networks [34], a mixture of deep CNNs also known as MixDCNN [35], CNN trees [112] and multiple granularity CNNs [109].

Instead of compressing whole images into fixed representations, the attention system makes it possible for a neural network to highlight features dynamically and can be used as part of fine-grained image classification systems. Examples of this includes two-level attention [116], attention for fine-grained categorization (AFGC) [96], fully convolutional network (FCN) attention [64], and diversified visual attentions [129]. RNNs, FCNs and CNNs are used as base architectures for attention fine-grained categorization, FCN attention and diversified visual attention, respectively.

Animal re-identification

Animal re-identification (re-ID) is an instance level recognition and retrieval problem where the aim is to distinguish individual animals and to recognize these animals upon re-occurrence [71].

Where animal re-ID is concerned, landmark points are used as extra training inputs together with the input image to create stronger and more specific embeddings, using for example architectures such as ResNet50 with weights pretrained on ImageNet [71].

Early approaches to solving the animal re-ID problem involved humans manually recording the location of unique animal characteristics and then saving those in a database and looking through the database to find animals with the highest number of similarities for re-ID.

More recently, Freytag et al. [32] investigated the effectiveness of using a combination of the BVLC AlexNet and VGG-face CNN architectures with bilinear pooling feature processing. Brust et al. [17] used the BVLC AlexNet architecture with bilinear pooling and YOLO object detection to create a ranked list of re-ID proposals. A combination of the YOLO object detection method, the ResNet50 architecture and an SVM classifier was used to localize elephant heads in [86], and a custom Siamese CNN was used in [67].

Research in animal re-ID has moved to algorithms that are able to quickly adapt to and make predictions on new images that the algorithm was not trained on, using techniques such as one-shot learning [5], generative adversarial networks [63], domain adaptation [130], and unsupervised methods [121, 16].

1.2.3 Summary

Of all the landmark detection tasks mentioned above, detecting landmarks of interest on tsetse fly wings shares a number of properties with the facial key point detection problem. Our dataset is made up of static images of wings where the main differences between wing images come from the biological differences in individual wings, positioning of the wings in the images and illumination. Similarly, for facial landmark datasets, for example those in [66, 2], the main differences in individual face image landmarks result from the differences between individuals, illumination as well as the position of the face in the image. Besides the differences, the input dataset provided for our work more closely resembles the input used in the facial keypoint detection problems addressed in [66, 2] which is a combination of images and their landmark coordinates.

Human pose estimation, like our research problem, is a regression problem. It is a different problem because of the added connections between the landmarks which form part of the output required for this task. The fine-grained categorization and animal re-ID problems differ in that they are classification tasks.

FCNs are state-of-the-art models for image segmentation problems and use spatial information when computing predictions. We will explore whether various FCN architectures can outperform CNN architectures on our particular problem. FCNs also allow input images of any size which would be more applicable to real-world scenarios. The deconvolution layers in FCN architectures increase the scale of the prediction data and the skip connections enable more robust training [129].

1.3 Data collection method

The tsetse flies that feature in our dataset were collected from Rekomitjie Research Station, Zambezi Valley, Zimbabwe. The Rekomitjie station is situated in Mana Pools National Park, which, combined with the Sapi, Hurungwe and Chewore Safari areas, covers about $10,000$ square kilometres. The image data used in this thesis comes from a subset of data that was collected from an 11-year study conducted between September 1988 and December 1999 [39].

Two main devices were used in that study:

1. stationary mechanical "epsilon" traps [72] baited with artificial host-odour, and
2. vehicle-mounted electric target, made up of an electrocuting grid, one metre tall and two metres long [108], mounted on the back of an open pickup truck [40].

Once the flies were captured, they were transferred to a laboratory and placed in individual 75×25 mm plastic tubes, under a black cloth to reduce the flies' activity. Female flies underwent a process of ovarian dissection developed specifically for tsetse flies [93, 94].

Wings of sampled flies were affixed with transparent sticky tape to a dissection record form and images of individual wings were taken, which led to our image dataset.

1.4 Dataset description

In this study we make use of a dataset that consists of a total of 2424 tsetse fly wing images. 1053 of these images contain left-facing wings, while 1371 contain right-facing wing images. Each of the images is colour and represented by a $1280 \times 1024 \times 3$ pixel matrix. Each pixel entry is an integer between 0 and 255 on the RGB scale, representing the red, green and blue intensities of each of the 1280×1024 pixels. For each of the images, annotations with eleven (x, y) coordinate

points were manually recorded for all the landmarks on the wing. These manual annotations will be used for training and testing purposes. Figure 3 provides two examples of tsetse fly wing images from the dataset, with their annotated landmark points.

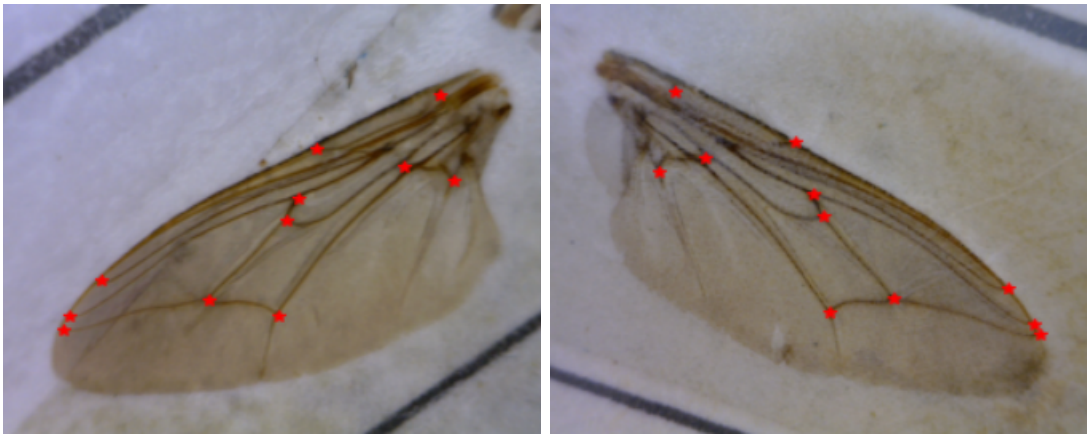


Figure 3: Examples of left-facing and right-facing tsetse fly wing images, with annotations.

Figure 4 is an example of a right facing wing with its annotations. The order of the annotation points is illustrated by the numbers on top of each point. This order is maintained for all images in our dataset. Each landmark point n in Figure 4 corresponds to coordinate (x_n, y_n) in the annotations.

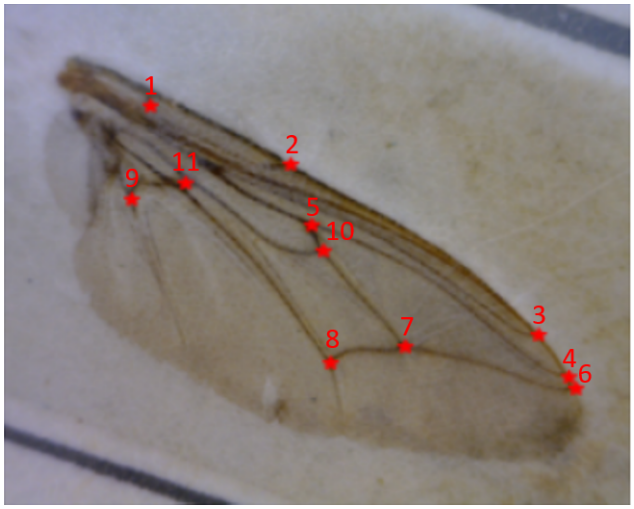


Figure 4: Right-facing tsetse fly wing with annotations and the numbering convention.

1.5 Aims and objectives

The purpose of this research is to assess the feasibility of automating the process of landmark detection using machine learning algorithms for a limited dataset. To do this we will compare the performance of FCNs against conventional CNNs for the regression task of landmark detection in the tsetse fly wing image dataset. Concretely, our aim is to adapt the facial keypoint detection techniques used in [66, 2] for our problem.

The objectives of this study are to:

- review previous methods used in landmark detection, for both human and animal body parts;
- construct and test a simple artificial neural network (ANN) model that takes our input image data and returns landmark coordinates, and investigate model performance;
- construct and test LeNet style CNN models that take our input image data and return landmark coordinates, and investigate model performance;
- construct and test FCN models that take our input image data and return landmark coordinates, and investigate model performance.

1.6 Thesis layout

The rest of this thesis is organized as follows. Chapter 2 provides background information on artificial intelligence, machine learning and computer vision required for this research. Chapter 3 describes the model architectures, the software specifications, our parameter search approach, the metrics we used to evaluate models as well as the pre-processing methods used. Chapter 4 describes the models we trained and the test results we obtained. Chapter 5 provides a conclusion of the work and gives suggestions for future work.

2 Background

This chapter is organized into four sections. Section 2.1 defines machine learning and what it means for a model to learn. It also looks at three high level categories for machine learning algorithms and two popular machine learning tasks. Section 2.2 discusses artificial neural networks (ANNs) and how these loosely mimic the human brain. This section also looks at activation functions, giving a few common examples. The multi-layer perceptron (MLP) and its three neuron types and a look into the neural network training process is also expanded upon. Section 2.3.1 focuses on convolutional neural networks (CNNs), giving an explanation of the convolution operation and expanding on different CNN layers. Finally, Section 2.4 provides a brief history of fully convolutional networks (FCNs), how CNNs can be converted to FCNs and the FCN architectures presented by Long et al. [65].

2.1 Machine learning

The field of machine learning focuses on the creation of algorithms that use data to build predictive models. This field is primarily concerned with the question of how to design these models such that they automatically improve with new data. Machine learning is multi-disciplinary and draws on principles and concepts from a number of fields, namely statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory [37]. While many theoretical problems can be formulated exactly, real world problems are often difficult to formulate exactly. Machine learning has emerged as a useful tool for modelling such real world problems. Classical computer algorithms are explicitly programmed by a human to perform a particular task, while with machine learning, a portion of the human contribution is replaced by a learning algorithm [37].

The aim of a machine learning task is to find (to learn) a function, $f : X \to Y$, that maps an input domain X of data onto an output domain Y of possible predictions. The form of the function f varies depending on the type of model and learning algorithm being used [69, 9].

2.1.1 Learning

Mitchell [9] defines learning in the following way:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

For machine learning algorithms, the dataset we provide is what we call the experience. This dataset contains a number of examples that are used to train the algorithm. If the machine learning task is classification, the accuracy of the algorithm, i.e. the proportion of times the algorithm correctly predicts the output, is used as the performance measure P. This value P tells us quantitatively how a particular machine learning algorithm is performing for a given dataset [9].

2.1.2 Classification of machine learning algorithms

The material presented in this section was adapted from Goodfellow et al. [37].

The dataset available for training is a key component of machine learning and is an important determinant of the category of learning that can be performed on the data. Machine learning algorithms can be largely classified into three categories: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning systems make use of labelled datasets containing pairs (x, y), where x represents a data point and y the corresponding true prediction (or label) for x. During training, the learning algorithm is given examples that have been labelled by humans. For example, in the object detection task we use training images where humans have marked the locations and classes of relevant objects. After learning from the examples, the algorithm should have the ability to predict the location and classes for images not yet encountered. During the training process, the algorithm uses the given input-output pairs to estimate the relationship between x and y in order to predict future input-output observations, by minimizing errors (wrong predictions) as much as possible.

Unsupervised learning uses unlabelled datasets. During training the system investigates and establishes similarities between different objects and in that way derives structure from the unlabelled data. In the case of supervised learning, the system is learning from a "teacher", while an unsupervised learning algorithm attempts to learn important features of a dataset on its own. A common example of unsupervised learning is clustering. In more recent years, unsupervised learning algorithms have been used in supervised learning tasks during the data pre-processing phase as a tool to more quickly discover key features and useful representations of the data.

Reinforcement learning systems are reward-based systems for an algorithm whose objective is to map states to actions, i.e. the algorithm is rewarded for making certain actions ("good decisions") and there are negative consequences associated with certain other actions ("bad decisions").

2.1.3 Machine learning tasks

Machine learning can be used to solve a wide variety of tasks. Regression and classification are two of the most popular machine learning tasks. For a regression problem, the model approximates the relationship between the independent input variables in order to successfully predict a continuous output. Classification tasks are those where the model is required to categorize the input data into discrete output classes [57].

2.2 Artificial neural networks

The neuron is the basic building block of the human brain. The neuron in Figure 5 has some key features, namely the dendrites, the soma, the axon and the axon terminal.

- The dendrites are cell receptors responsible for receiving communication from other cells into the cell body. The connection between the neuron that sends the signal and the neuron that receives the signal is known as the synapse. The synapse determines how much of the signal that is being sent by the sender neuron is received by the receiver neuron (the strength of the signal).

- The soma, which is also referred to as the cell body, is the spherical part of the neuron which contains the nucleus.
- The axons are cell transmitters, and are responsible for sending communication from the cell nucleus to the axon terminals.
- The axon terminals connect to the dendrites of other nuclei and are the points of contact through which signals are transmitted.

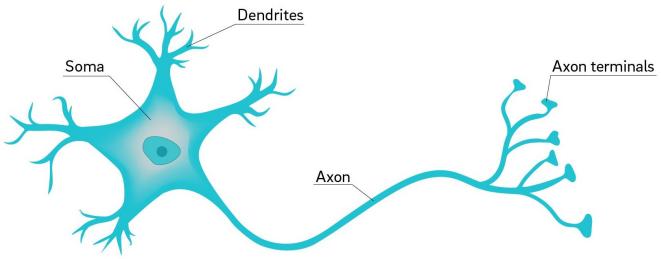


Figure 5: Biological neuron [87].

On average, the human brain has about 100 billion of these neurons, each of which may be connected to as many as 10,000 other neurons that pass signals to each other using up to approximately 1,000 trillion synaptic connections. This is equivalent, by some estimates, to a trillion bits per second processor computer [124].

According to Bataineh [7], artificial neural networks (ANNs) are biologically inspired systems meant to simulate the computational approach of the human brain. These systems have been developed and derived to have a function loosely similar to the human brain by memorizing and learning various tasks. Neural networks are trained to learn behaviour and to apply that behaviour in the future, in new scenarios. These ANNs typically consist of thousands of interconnected neurons, arranged as multiple layers stacked on top of each other. Artificial neurons are the basic computational building blocks, their structure ensures that ANNs are able to perform complex tasks and can be applied in a variety of areas [3].

The key features of the artificial neuron, shown in Figure 6, have been inspired by the neuron in the human brain shown in Figure 5.

- Lines w_0x_0, w_1x_1 and w_2x_2 are analogous to a neuron's dendrites. x_i is an input to the neuron and receives information from other neurons, while w_i is the connection weight which represents the synapse strength.
- The combination of inputs from other neurons come together in the cell body as shown in the following expression:

$$\sum w_i x_i + b. \tag{2.1}$$

  In this expression, b represents the bias of the neuron which, from a biological view point, can be seen as the threshold at which the neuron will fire. The neuron performs a predefined action on the combined inputs through a function f, which is known as the activation function, to create the resulting output:

$$f\left(\sum_i w_i x_i + b\right). \tag{2.2}$$

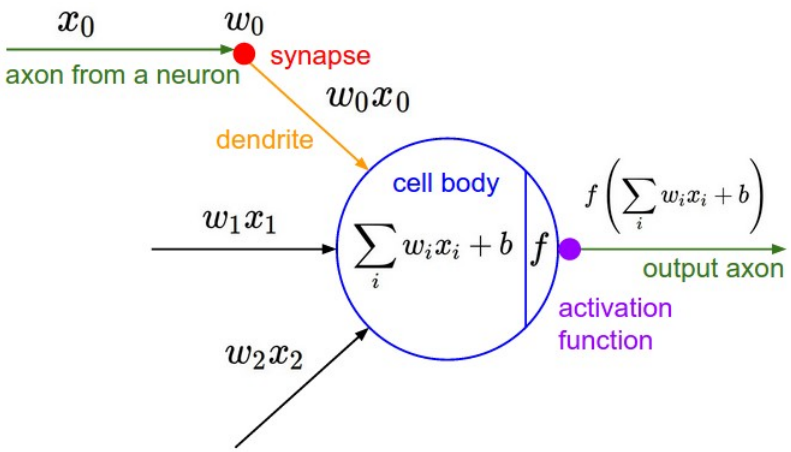


Figure 6: Artificial neuron [52].

The activation function f can be linear or nonlinear, but a nonlinear, monotonically increasing, continuously differentiable function is often chosen in practice. The reason for this is that though it might be easier to work with a linear function, most real world problems require nonlinear mappings. The monotonic and differentiable qualities of the activation function assist during the training phase and help ensure that convergence towards an optimal solution is possible.

- The transformed signal in the output axon is what the neuron will be communicating to other neurons in the network.

As a collective, a group of neurons in multiple layers can efficiently learn a wide variety of mappings in order to perform a wide variety of tasks.

Activation functions

The activation function determines the output of the neuron by performing a transformation on a linear combination of the inputs, and can be used to determine whether or not a neuron should fire [76]. There are a few commonly used activation functions that have proven to work for problems similar to the ones we will investigate.

(a) Logistic function

The S-shaped sigmoid function, an example of which can be seen in Figure 7 (a), is one of the most common activation functions in artificial neural networks. This function is monotonically increasing and exhibits both linear and nonlinear behaviours [97, 60].

The logistic function given by

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

is an example of a sigmoid function and maps the interval $(-\infty, \infty)$ to the bounded range $(0, 1)$. Using this activation function, neurons whose input value x is close to either ∞ or $-\infty$ result in a very small change in output values. This is because the gradient of the logistic function at extreme values is very low, which can be an issue during the training process as smaller gradients slow down learning. The function also contains an exponential which means that the process of obtaining gradients from such a function can be computationally expensive.

(b) Hyperbolic tangent function

The hyperbolic tangent function is given by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.4}$$

This is another example of a sigmoid function which maps the interval $(-\infty, \infty)$ to the bounded region $(-1, 1)$. It is linked to the logistic function above by the transformation: $\tanh(x) = 2\sigma(2x) - 1$.

Unlike the logistic function, the hyperbolic tangent function is rotationally symmetric around the origin and because of the larger range it provides stronger gradients when x is close to zero. For these reasons, the hyperbolic tangent activation function is often preferred over the logistic activation function. However, similar to the logistic function this function can saturate which may slow down learning [60, 26]. An example of the hyperbolic tangent function can be seen in Figure 7 (b).

(c) Rectified linear unit (ReLU) function

In recent years, ReLU has become one of the most popular activation functions. Neural networks that use this activation function train several times faster than neural networks that use the other activation functions [57]. A reason for the quick training time might be because ReLU is a simple piecewise linear function while $\tanh(x)$ and $\sigma(x)$ involve operations that are more computationally expensive.

The rectified linear unit is given by

$$r(x) = \max(0, x) = \begin{cases} x, & x > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

A notable weakness of ReLU is that for all $x < 0$ the gradient of the function is zero. Neurons that result in a zero gradient are 'dead' as far as training is concerned for that specific input. For a full neural network the situation of dead neurons can be avoided through careful weight initialization and hyperparameter tuning. An example of the ReLU function can be seen in Figure 7 (c).

(d) Leaky ReLU

The leaky ReLU activation function is defined as

$$r(x) = \begin{cases} x, & x > 0, \\ \alpha x, & x \leq 0, \end{cases} \tag{2.6}$$

where α is a small positive constant. A modification of standard ReLU, this has all the advantages of ReLU but also attempts to fix the problem of 'dead' neurons that standard ReLU experiences. Instead of the function being 0 for all values less than 0, leaky ReLU has a small positive slope.

(e) Softmax

The softmax activation function is given by

$$s_k(x) = \frac{e^{x_k}}{\sum_{i=1}^{K} e^{x_i}}, \qquad k = 1, 2, \ldots, K. \tag{2.7}$$

This is a generalization of the sigmoid activation function which is typically used for multi-class classification problems with K distinct classes, where $K > 2$.

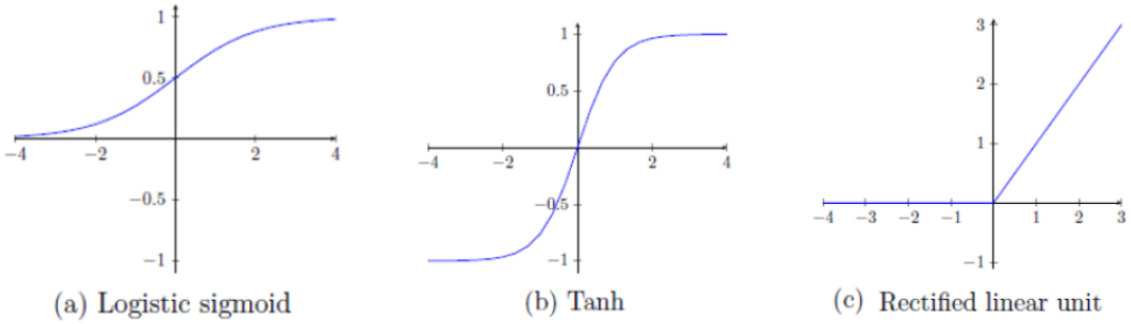


Figure 7: Activation functions [26].

Multi-layer perceptrons

Several ANN models have been proposed over the years with multi-layer perceptrons (MLP), Hopfield networks and Kohonen's self-organizing networks being among the most popular [123]. Of these three models, we will focus on the MLP model.

An MLP is made up of three neuron types.

1. Input neurons form an input vector whose primary role is to encode information about the external environment. Input neurons do not perform any transformation of the information but only pass on the received information to subsequent neurons. Because of this, we can think of them as having the identity activation function, $f(x) = x$. An MLP typically has one input layer.

2. Output neurons receive information from the preceding neurons and transform it to fit into a desired output structure. The output neurons produce the final values of the neural network. An MLP typically has one output layer.

3. Hidden neurons are the key elements in a neural network. They receive a signal from preceding neurons, which may be either input neurons or another layer of hidden neurons. The signal is then processed according to (2.2). The output is passed onto subsequent neurons, which may be either another layer of hidden neurons, or output neurons. An MLP can have one or several hidden layers. The more hidden layers a network has, the deeper the network is said to be.

Figure 8 is an example of a simple MLP with one input layer, two hidden layers and one output layer.

There are two schools of thought as far as calculating the number of layers of neural networks is concerned. When performing this calculation, one of them includes the input layer in the count, while the other does not. In this thesis we will only include the layers that modify the data, i.e. we will exclude the input layer from the count. We will consider the input layer as layer 0 and we will call the output layer L.

Neural network architecture

When constructing a neural network the first thing one must consider is the architecture, i.e. how many neurons to include in the input, output, and hidden layers as well as the number of hidden layers that should be included. Deciding on the number of input and output neurons is relatively easy as the input neurons are determined by the number of input features that will be used in the problem set and the output neurons are determined by the expected output

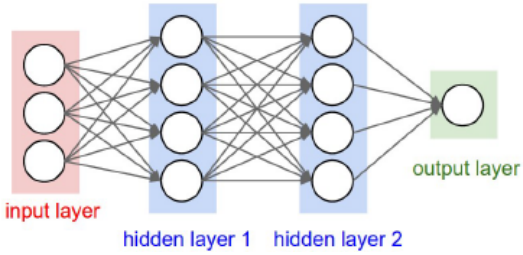


Figure 8: A neural network with three input neurons, two hidden layers made up of four neurons each and one output layer with one neuron [52].

dimensions. For example, if the network is expected to produce a binary answer of either 0 or 1, it can be deduced that this network would only need one output neuron. A tougher problem is deciding the number of neurons that need to be included in the hidden layers as well as the number of hidden layers to include. In practice, one would need to train a few networks and look at the number of hidden neurons and layers that give the smallest generalization error to answer this question [3].

The objective function

The majority of machine learning algorithms involve the minimization or maximization of a chosen function $g(v)$ by changing v. This is what is referred to as optimization. The function $g(v)$ that we wish to optimize is known as the objective function. When the objective function is being minimized it can also be referred to as a cost function, a loss function or an error function [37]. In this thesis we will refer to our objective function as a loss function.

In the context of machine learning most of the functions that we have to optimize may not be convex. A convex function is a bowl-shaped function that has only one global minimum (lowest point of the entire function) and no local minima (a point where the function $g(v)$ is lower than all the points close to v but that is not the lowest point of the entire function). We rather need to optimize functions that may have several local minima and saddle points (critical points that are neither minima nor maxima) that are surrounded by flat sections on the function space. In practice we settle for the lowest value of $g(v)$ that we can find, but this value may not necessarily be minimal by any formal definition [37].

Training a neural network consists of the following four main steps:

1. weight initialization;
2. forward propagation;
3. loss function calculation;
4. weight update through gradient descent.

Steps 2 to 4 are repeated until we converge to either a global or a local minimum for the loss function (or some other stopping condition is met).

Step 1: Model weight initialization

The weights chosen to initialize a model affect the training outcome. As such, initializing the weights correctly is an important step in creating neural networks [37].

Initializing all weights as 0 or any other constant creates symmetries. For two hidden units that are connected to the same inputs, have the same activation functions and are initialized the same, the model will constantly update these units in the same way. These two hidden units will learn the same features throughout model training [37, 54].

Weight initialization is in general dependent on the chosen activation function. Random initializations break the symmetries that would arise from constant initializations. Activation functions such as sigmoid and tanh tend to saturate for very high or very low values, therefore initializing the weights too low or too high could slow down learning. Another issue that may arise from making the weights too large is the problem of exploding gradients. This is where gradients get larger and larger with each training iteration [10], which may result in divergence. In the case of a sigmoid or tanh activation function, making the weights too small may lead to a problem known as vanishing gradients, where gradients get smaller with each iteration, which will also slow down learning significantly and may, in extreme cases, stop learning. Activations such as ReLU saturate in only one direction and are therefore more robust against the vanishing gradients problem.

Step 2: Forward propagation

During the forward pass, information is passed through the network from the input layer to the output layer, and the network calculates the output of each neuron in each layer and that output is then used as input for subsequent layers.

Concretely, for every neuron j in layer l (with the exception of input neurons), the output $y_j^{(l)}$ is computed from the previous layer's output signals $y_j^{(l-1)}$ as follows:

$$s_j = \sum_{i=1}^{m_l} w_{ij}^{(l)} y_i^{(l-1)}, \tag{2.8}$$

$$y_j^{(l)} = f(s_j). \tag{2.9}$$

If we let $y^{(l-1)} = \{y_j^{(l-1)}\}$ denote the vector of inputs in layer l, $s^{(l)} = \{s_j^{(l)}\}$ the vector of activations in layer l, and $W^{(l)}$ the weights of all (i, j) connection pairs received in layer l, then the feed-forward algorithm can be summarized as follows:

i. let $y^{(0)}$ be a vector of the model inputs

ii. let $\hat{y}$ be the vector of corresponding outputs

iii. for (l in $1, 2, ..., L$) do:

$$s^{(l)} = W^{(l)} y^{(l-1)}$$

$$y^{(l)} = f(s^{(l)})$$

iv. end for

v. $\hat{y} = y^{(L)}$

Step 3: Loss function calculation

Loss functions help us understand how well the network is mapping the given inputs to the target outputs.

The mean squared error, also known as the L_2 loss function, is the most common regression loss function and is represented by the formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.10}$$

where y_i is the true response value for the i^{th} observation, and $\hat{y}_i$ is the regression model's prediction for the i^{th} observation. The MSE is the squared difference between the regression model's prediction and the true response values. The closer the predictions from the regression model are to the true response values, the lower the MSE value.

Because differences are squared, the MSE penalizes observations with large differences much more than the advantage it gives to those observations that are close to the true response value.

Alternatively, the mean absolute error, also known as the L_1 loss, is given by the formula:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{2.11}$$

This loss calculates the mean absolute difference between the response values predicted by the regression model and the true response values. The closer these values are to each other, the lower the MAE. The MAE is more robust to outliers than the MSE and is preferred as a loss function for regression tasks with datasets that contain outliers.

Step 4: Weight update through gradient descent

The gradient descent method can be explained in the context of a landscape with peaks and valleys. Depending on how the weights are initialized, we start at a certain point on the landscape. We use the loss function to determine how far we are from our target and use gradient descent to iteratively move us to the lowest point we can get to on the landscape. It is at the lowest point where our model will be closest to the target and the loss will be minimized.

Concretely, assume we have a function $y = f(x)$, where x and y are real numbers. The derivative of this function, denoted by $\frac{dy}{dx}$, gives the slope at the point x. This derivative tells us how a small change in the input, x, would change the output, y. For example, assuming that ϵ is a small positive number, $f(x - \epsilon \text{ sign}(f'(x)))$ is less than $f(x)$ for a small ϵ. We know then that moving the value of x with a small step whose sign is opposite that of the gradient would result in a reduction in $f(x)$, which is our objective [37]. For the gradient descent method to find a true minimum, $f(x)$ must be a sufficiently smooth convex function.

Gradient descent uses the following update rule:

$$W(t+1) = W(t) - \eta\frac{\partial E(W)}{\partial W}, \tag{2.12}$$

where η is a small positive number that is referred to as the learning rate. The learning rate is the step size and it determines how much of the gradient will be used to update the model weights at each learning step. It is important that we choose the learning rate correctly. We need to ensure that we do not make the number too small that training takes so long that convergence becomes infeasible. On the other hand, making the learning rate large, which

would make the rate of convergence faster, may lead to divergence, as this could mean steps so large that the minimum point may be overshot entirely [60, 89].

Below are a number of popular gradient descent algorithms.

1. Adaptive gradient descent (AdaGrad)

The AdaGrad algorithm uses the information it has learned about the geometry of the data that it observed in past iterations to ensure that the gradient based updates are more informed. Parameters that are associated with features that occur frequently are given lower learning rates while parameters that are associated with features that are infrequent are given higher learning rates [28, 89]. This adaptation of the gradient descent method ensures that the model also takes notice of comparatively rare features that are predictive. To achieve this, AdaGrad makes use of a different learning rate for every weight in W at every time step t, and the following equations for its updates:

$$G(t) = G(t-1) + \left(\frac{\partial E}{\partial W}\right)^2, \tag{2.13}$$

$$W(t+1) = W(t) - \frac{\eta}{\sqrt{G(t)+\epsilon}}\frac{\partial E}{\partial W}, \tag{2.14}$$

where $G(t) \in \mathbb{R}^{d\times d}$ is a diagonal matrix where each diagonal element (i,i) is the sum of the squares of the gradients with respect to W_i [28].

2. The root mean square propagation (RMSProp) optimizer

RMSProp is an unpublished algorithm that is widely used. It is a generalization of RProp (resilient propagation), a technique for learning rate acceleration that was created exclusively for batch gradient descent [85], to widen the scope and ensure that it can be used with mini-batch gradient descent.

The RProp algorithm was proposed for the purpose of remedying AdaGrad's flaw of having learning rates that diminish too quickly. As previously mentioned, small learning rates might make training to convergence infeasible. The learning rule for RMSProp is given by the following two equations, where $\left(\frac{\partial E}{\partial W}\right) = g_t$:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2, \tag{2.15}$$

$$W(t+1) = W(t) - \frac{\eta}{\sqrt{E[g^2]_t+\epsilon}}g_t. \tag{2.16}$$

Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001 [89].

3. Adaptive moment estimation (Adam)

The Adam optimization technique was proposed by Kingma and Ba [56] and is one of the most popular optimization techniques in the deep learning field. It was designed to combine the advantages of AdaGrad and RMSProp, to create a technique that works well with sparse gradients as well as for online and non-stationary settings. Adam dynamically stores the moving average of past gradients in a variable $N(t)$, as well as the moving average of past squared

gradients in a variable $U(t)$. Adam uses the following sets of equations as its update rule:

$$N(t) = \phi_1 N(t-1) + (1-\phi_1)\frac{\partial E}{\partial W}, \tag{2.17}$$

$$U(t) = \phi_2 U(t-1) + (1-\phi_2)\left(\frac{\partial E}{\partial W}\right)^2, \tag{2.18}$$

$$\hat{N}(t) = \frac{N(t)}{1-\phi_1(t)}, \tag{2.19}$$

$$\hat{U}(t) = \frac{U(t)}{1-\phi_2(t)}, \tag{2.20}$$

$$W(t+1) = W(t) - \frac{\eta}{\sqrt{\hat{U}(t)+\epsilon}}\hat{N}(t). \tag{2.21}$$

Kingma and Ba propose default values of 0.001 for η, 0.9 for ϕ_1, 0.999 for ϕ_2, and 10^{-8} for ϵ [56].

Gradient descent variants

There are three main variants of gradient descent, which differ due to the amount of data that each of these uses to calculate the gradient [60].

1. Batch gradient descent, also known as vanilla gradient descent, calculates the gradient of the loss function for the weight parameters, W, and updates the weights of the model after going through the entire training dataset of N samples. The update rule for this variant is:

$$W(t+1) = W(t) - \eta\frac{\partial E(W)}{\partial W}. \tag{2.22}$$

2. Stochastic gradient descent (SGD) calculates and updates the gradient for a single sample $(x(i), y(i))$ uniform randomly chosen from the dataset. The update rule for this variant is:

$$W(t+1) = W(t) - \eta\frac{\partial E(W, x^{(i)}, y^{(i)})}{\partial W}. \tag{2.23}$$

3. Mini-batch gradient descent calculates and updates the gradients for a mini-batch of size n, randomly chosen from the dataset, where $1 < n < N$. The update rule for this variant is:

$$W(t+1) = W(t) - \eta\frac{\partial E(W, x^{(i:i+n)}, y^{(i:i+n)})}{\partial W}. \tag{2.24}$$

Backpropagation

The idea of the backpropagation algorithm is to repeatedly apply the chain rule to calculate the effect of each weight in the network on a loss function. Once the values and signs of the partial derivative are obtained, the gradient is used in the update rule to be applied to each weight [85].

To explain the backpropagation algorithm we make use of a small example network.

We will be using matrix notation in this explanation. Let $W^{(L)}$ be the matrix of weights corresponding to layer L in our network, $\mathbf{x}$ a vector of inputs, $\mathbf{b}^{(L)}$ the vector of biases, $\mathbf{z}^{(L)}$ a vector that is a linear combination of $\mathbf{x}$, W and $\mathbf{b}$, and f an activation function.

The input layer itself makes up the first set of activations i.e. $x_i = a_i^{(0)}$ for all i.

The activation $\mathbf{a}^{(1)}$ at the first layer ($L = 1$) is calculated by taking the weight matrix $W^{(1)}$ in this layer, multiplying that with the input vector $\mathbf{x}$ for this layer, adding the bias $\mathbf{b}^{(1)}$ for this layer and putting the result through the activation function f:

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)},$$
$$\mathbf{a}^{(1)} = f(\mathbf{z}^{(1)}).$$

The same procedure can be applied for all activation functions in subsequent layers, up to the final output layer:

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)},$$
$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}).$$

The steps above constitute the forward pass. After the forward pass we use the loss function E to calculate how far off the predicted value $\mathbf{s}$ is from the desired output, $\mathbf{y}$. The backpropagation algorithm adjusts the weights repeatedly, with the aim of minimizing the difference between the model prediction and the desired output. To achieve this, the chain rule is used to calculate gradients. For a single weight, the gradient is

$$\frac{\partial E}{\partial w_{jk}^{(l)}} = \frac{\partial E}{\partial z_j^{(l)}} . \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}. \tag{2.25}$$

Recall that

$$z_j^{(l)} = \sum_{k=1}^{m} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}, \tag{2.26}$$

where m is the number of neurons in layer l.

Then

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = a_k^{(l-1)}, \tag{2.27}$$

which implies that

$$\frac{\partial E}{\partial w_{jk}^{(l)}} = \frac{\partial E}{\partial z_j^{(l)}} a_k^{(l-1)}. \tag{2.28}$$

A similar set of equations can be applied for $b_j^{(l)}$. The gradient then allows us to optimize the model's parameters using one of the gradient descent variants that have been outlined above.

Training terms

An epoch is one complete presentation cycle of a chosen dataset that is to be learned by a model. This consists of the forward and the backward pass.

The batch size is the number of examples that are presented to the model in one step of mini-batch gradient descent.

Model performance

Generalization is defined as the ability to perform well on previously unobserved inputs [37] and is what we hope the machine learning models will do well. According to Goodfellow et al. [37], underfitting and overfitting are the two main problems for machine learning and a model that either underfits or overfits is said to not generalize well and therefore performs poorly. When a model is unable to obtain a sufficiently small error on the training set we say the model underfits the data. This problem can arise if the model has too few hidden layers. The opposite problem is when the model gets to know the training data so well, and learns the detail and the noise of the training data such that the model struggles to model any new data. In this case, we say that the model has overfit the training data, and this can be caused by too many layers in the model's architecture [3, 37].

Regularization

In an effort to avoid overfitting, the concept of regularization was introduced. The aim of regularization is to discourage a model from learning the noise from a dataset by putting in measures that encourage model generalization. A few regularization techniques are discussed below.

1. L_1 regularization

The L_1 regularization technique works by using a penalty term which encourages the model to minimize the absolute sum of the parameter values. L_1 regularization is best used in cases where the user believes that several of the features used in the model should be ignored, because it has been observed to drive parameters down to zero [74].

The loss function can be adjusted to incorporate regularization using the equation below:

$$E_{reg} = E + \lambda \sum_{k=1}^{M} |w_k|, \tag{2.29}$$

where λ is a parameter that controls how important the regularization term is to the overall loss that will be minimized, and M is the number of weights [74].

2. L_2 regularization

L_2 regularization also works by using a penalty term. This penalty term encourages the model to minimize the sum of the squares of the parameter values [74].

The loss function can be adjusted to incorporate L_2 regularization using the following equation:

$$E_{reg} = E + \lambda \sum_{k=1}^{M} (w_k)^2. \tag{2.30}$$

3. Early stopping

The early stopping regularization technique can be explained using the following training steps.

Step 1: Split training data into training and validation set.

Step 2: Use only the training set when training, and the validation set to evaluate the error once in a while (e.g. after every sixth epoch).

Step 3: Terminate training as soon as the error in the validation set is higher than what it was in previous checks. In Figure 9, the early stopping point is the optimal number of epochs that would provide the lowest training loss before an increase in validation loss is observed.

Step 4: The weights of the network obtained from the previous run should be used as the training results [83].

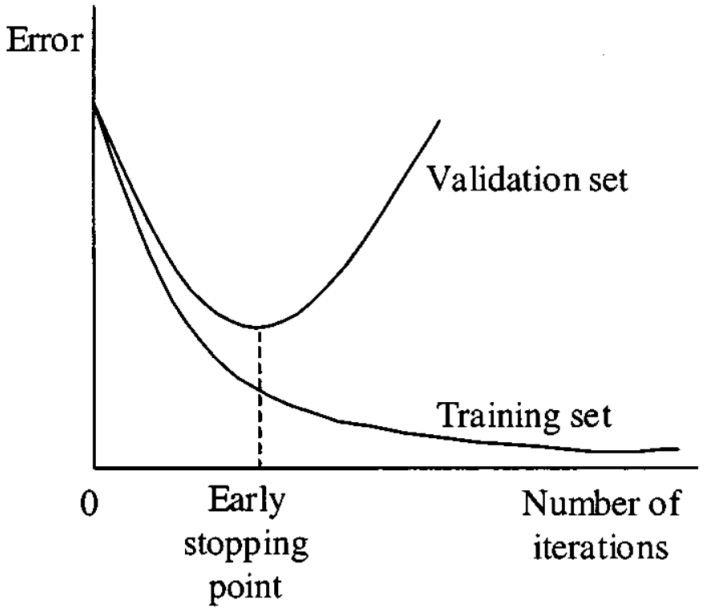


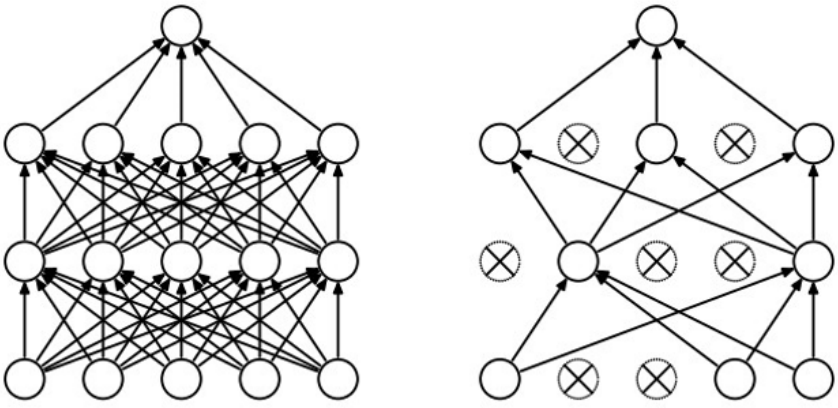
Figure 9: Early stopping [36].

4. Dropout

Dropout is a regularization technique proposed by Srivastava et al. [99]. During training dropout is achieved by keeping a neuron active with probability p, or setting the unit (as well as its incoming and outgoing connections) to zero which temporarily removes the neuron from the network. In Figure 10, (a) is an example of neuron connections for a standard neural network iteration, and (b) is an example of neuron connections with the dropout regularization technique applied.

This means that there is a high likelihood that the combination of neurons used for network training changes with each iteration, which helps to reduce overfitting and to improve the model's ability to generalize.

This technique is only applied when training a neural network, and not when testing the performance of the network.



(a) Standard Neural Net (b) After applying dropout.

Figure 10: Dropout [99].

5. Batch normalization

Batch normalization is a regularization technique developed by Ioffe and Szegedy [46]. This regularization technique normalizes the output of previous layers by subtracting the batch mean from each output value and dividing the result by the batch standard deviation, in an effort to make the neural network more stable. The normalized output serves as input for the following layer. Figure 11 illustrates how this normalization is performed.

The shift in activation output that results from this calculation might mean that the weights may no longer be optimal. To minimize the loss function, stochastic gradient descent may undo the normalization. To prevent this from happening, batch normalization includes two trainable parameters in each layer. As such, the normalized output is divided by γ (a standard deviation parameter) and has β (a mean parameter) added to it. This means that to denormalize the inputs in an effort to make the weights optimal, stochastic gradient descent can only change β and γ.

Networks that use batch normalization can be considerably more robust to bad initialization [53].

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Figure 11: Batch normalization [46].

2.3 Convolutional neural networks

Convolutional neural networks (CNNs) are a type of artificial neural network, loosely inspired by the visual cortex of mammals. The visual cortex is the part of the brain that plays a key role in the reception and processing of visual information.

A CNN uses a mathematical operation called convolution, which is a specialized linear function. Put simply, CNNs are artificial neural networks that use the convolution operation instead of full matrix multiplication in one or more of their layers [37].

2.3.1 Convolution

To better understand convolution, consider two finite discrete 1-dimensional functions f and g. The convolution of these two functions is defined as:

$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m). \tag{2.31}$$

Within the above, f (the first argument) is commonly referred to as the input and g (the second argument) as the kernel or filter. The resulting output is then referred to as a feature map [52].

For the purpose of dealing with image data, we use convolutions over more than one dimension. When performing convolutions on a 2-dimensional image f, we would also want to use a 2-dimensional kernel g, and the above equation would be modified as follows:

$$(f * g)(i,j) = \sum_{m}\sum_{n} f(i-m, j-n)g(m,n). \tag{2.32}$$

The convolution operation can be used in image processing to extract different features from an image. Figure 12 is an example of a hand-crafted convolutional filter g being used to detect horizontal edges in an image f.

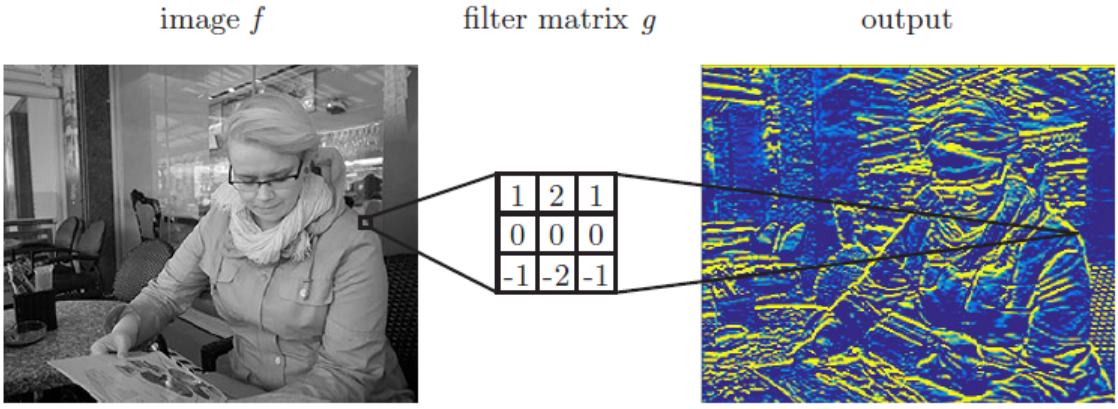


Figure 12: A convolution filter to detect horizontal edges in an input image [105].

An image is a matrix of numbers, and it would seem reasonable to take such a matrix, flatten it into an array and feed it into an ANN. There are two reasons why this might not work well.

Firstly, consider the images from the MNIST dataset [61] where the input size of images are $28 \times 28 = 784$. Suppose we use two hidden layers made up of 100 neurons each and an output with one neuron. For this shallow ANN that accepts very small input images, the number

of training parameters is already over 80,000, which becomes computationally expensive to train over. Using the same neural network architecture with a basic VGA colour image of size $640 \times 480 \times 3$, we would have over 900,000 input parameters and the network would have over 90 million parameters to train. If we were to make this network deeper the parameters would increase even further, quickly making training computationally infeasible.

Secondly, ANNs ignore the structure of the input. The input of an ANN can be received in any order and, after training, the model outcome would not be affected. Images however have strong local structures, as pixels that are close to each other tend to be highly correlated. Flattening the image pixels for complex images that have location-based pixel dependencies would mean that the resulting ANN would be unable to directly exploit those dependencies.

By applying 2-dimensional filters, CNNs can capture spatial dependencies in an image and provide some amount of invariance to shift, scale and distortion through three fundamental principles: local receptive fields, shared weights, and spatial subsampling [59].

1. Local receptive fields

Images have a strong 2-dimensional local structure. Because CNNs restrict the receptive fields of hidden neurons to be local, through the architectural design, they force the extraction of local features [59].

2. Shared weights

Rumelhart et al. [90] describe weight sharing as a situation where a single weight or parameter controls several connections, i.e. requiring that the connection strengths of different edges be equal [58]. By applying equality constraints to filter weights, we ensure the following:

- filters that are used on input images extract the same features in different areas of the input [59];
- important information relating to the geometry and topology of the task is expressed [59];
- the network is able to automatically achieve shift and scale invariance;
- the number of parameters that the model needs to learn is significantly reduced.

3. Temporal subsampling

Subsampling in CNNs is mainly used to make the model more robust against noise and small distortions [95]. Subsampling achieves this objective by reducing the resolution of the feature map, which in turn decreases the precision with which the model encodes the position of distinguishing features and how sensitive the output is to small distortions [59].

2.3.2 CNN layers

CNNs consist of three main kinds of layers:

1. convolutional layers, which are responsible for performing discrete 2-dimensional convolution operations on their inputs using a set of filters and further applying nonlinear activations to the output;
2. pooling layers, which perform the function of summarizing neurons for the purpose of reducing the size of their input [95];
3. fully connected layers, which take the input that has been convoluted and pooled and map it to the expected output dimension.

Convolutional layers

Convolutional layers are key to the architecture of a CNN and give this type of network its name. During training the convolution operation is computed on the layer input and a trainable filter is used to produce a modified version of the input. A feature map is a 2-dimensional arrangement of neurons, where all neurons in one feature map have input weight sharing constraints, i.e. they must all share one set of weights and a bias. The weight matrix used to produce a given feature map is what we referred to as the trainable filter.

ANNs have inputs that are presented as columns while the inputs of CNNs are represented as neurons with a rectangular structure. We construct the trainable filter that is used in the convolution operation to have a smaller dimension than the input. The filter is shifted systematically across the image region by region and a convolution is computed at each region. The region that the filter covers at each convolution instance is referred to as the receptive field.

The number of filters used determines the number of feature maps that will be created from a given input. The size of each filter as well as the number of filters that will be utilized in the layer are hyperparameters that must be chosen prior to training.

Pooling layers

The objective of the pooling layers is to make the model robust to, for example, the position of an object in the picture (i.e. spatial invariance) by decreasing the resolution of the feature maps.

The pooling operation is performed on each feature map from the previous layer, creating a pooled feature map which is often much smaller than the original. The size of the pooling layer can also be considered a hyperparameter of the model.

We evaluate two different pooling operations: max pooling and average pooling, using a set of examples.

Given a 4×4 input and a 2×2 pooling filter with a stride of 2, the max pooling and average pooling operations create new feature maps, as illustrated in Figure 13 and Figure 14.

Max pooling

For each 2×2 filter region represented by the different colours in our 4×4 single depth slice, we take the maximum value of that region and create a new output matrix where each element is the maximum of a region in the original input.

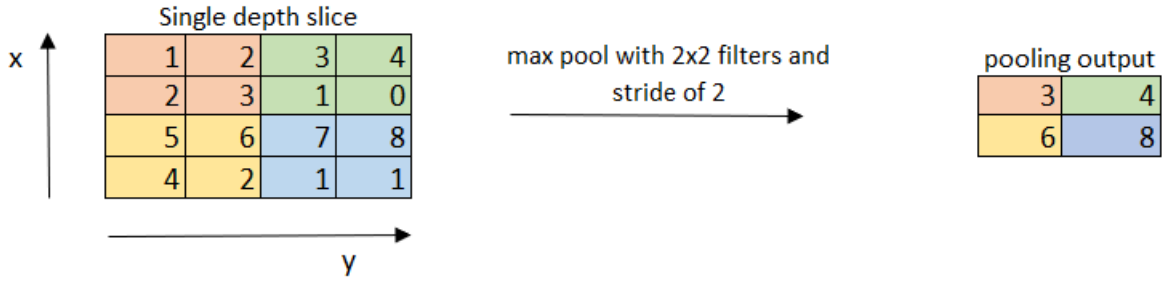


Figure 13: An example of max pooling applied to a 4×4 input.

Average pooling

In the case of average pooling the resulting output feature map is created by taking the average value of a region and creating a new output matrix where each element is the average of a region in the original input.

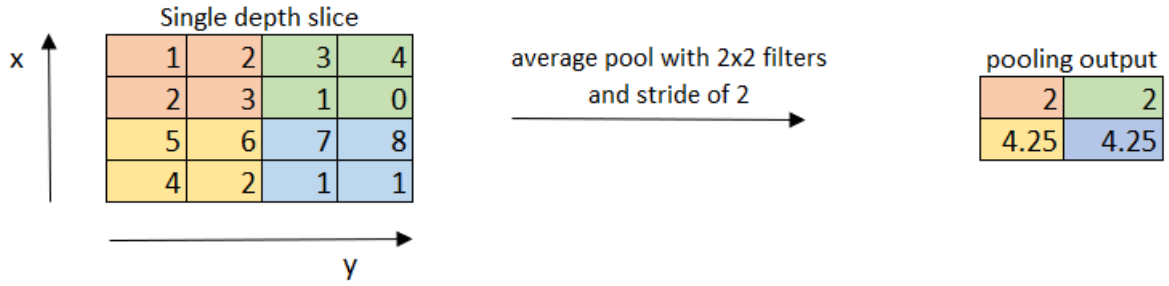


Figure 14: An example of average pooling applied to a 4×4 input.

Both sub-sampling operations result in a feature map that has lower resolution than its input [95].

Fully connected layers

In fully connected layers all the neurons in the current layers are connected to all the activations in previous layers. Figure 15 is an illustration of how fully connected layers connect two 2×2 feature maps to output neurons.

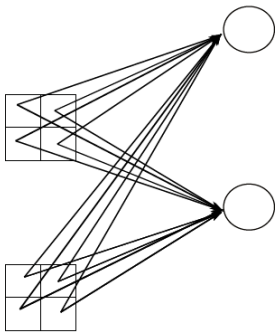


Figure 15: Illustration of a fully connected layer. Two subsampled feature maps of dimension 2×2 have each of their neurons connected to every neuron in the output [26].

A typical CNN architecture is shown in Figure 16. Fully connected layers are often used as the last few layers of a CNN to map all the features extracted from previous convolution and pooling layers to the target output [58].

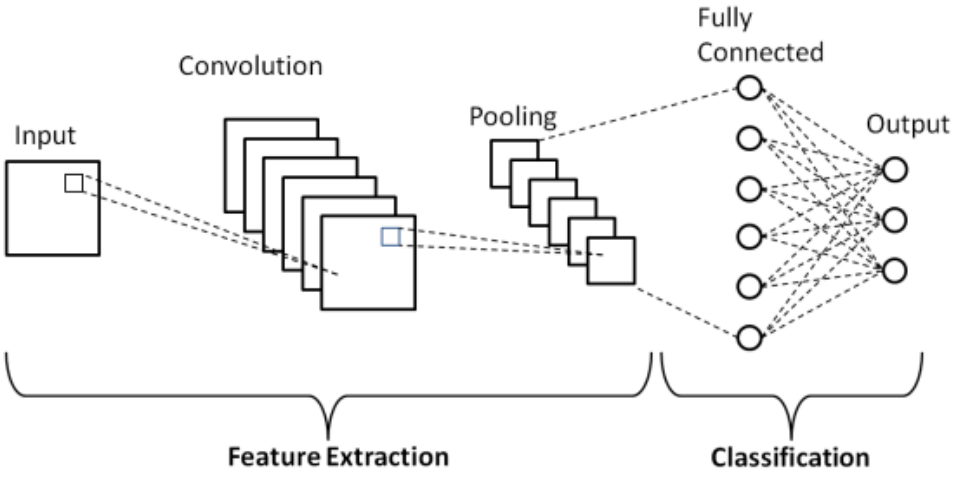


Figure 16: Typical CNN architecture [81].

Other CNN hyperparameters

Stride

The filter stride is the size of the steps taken between two convolutions that are next to each other. As illustrated by the example in Figure 17, this hyperparameter influences the number of convolutions performed [52].

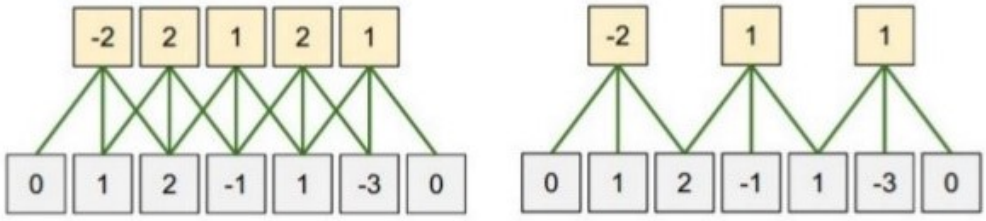


Figure 17: The image on the left shows the convolution operation performed on an input using a stride of 1 while the image on the right shows the convolution operation performed on an input using a stride of 2 [52].

Padding

Padding, which is also referred to as input padding, involves changing the resolution or size of the input by adding artificial borders. Figure 18 shows examples of input padding techniques.

Neural network frameworks detail three types of input padding: ‘same’, ‘valid’ and ‘full’.

- Same padding adds padding to the input in a way that ensures that the resulting feature map is the same size as the input.
- Valid padding adds no artificial padding to the input image and the convolution operation is conducted on the input as-is. The resulting feature map will be smaller than the original input.
- Full padding involves padding the input image such that the resulting feature map is

larger in size than the original input. This framework is not used much in practice and is included here for the sake of completeness [114, 52].

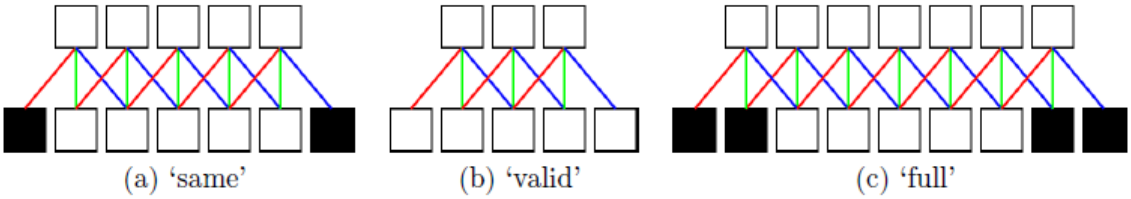


Figure 18: Examples of input padding techniques. The white squares on the bottom represent the original input while the black squares represent input padding. The squares in the top layer are convolution outputs using a filter of size three represented by the red, green and blue lines [114].

2.4 Fully convolutional networks

More commonly known by their abbreviation, FCNs (fully convolutional networks) were first presented by Long et al. [65] where they took common CNN architectures such as AlexNet, which are made up of convolutional and fully connected layers, and converted them into architectures that contain only convolutional layers. To do this conversion, they used the principle that fully connected layers can be re-interpreted as convolutional layers.

To explain this principle, consider the AlexNet architecture in Figure 19. $C1$ to $C5$ are convolutional layers, while $FC6$ to $FC8$ are fully connected layers. This architecture will be considered an FCN when all its layers are convolutional layers. We will therefore look at converting the fully connected layers, $FC6$ to $FC8$, to convolutional layers.

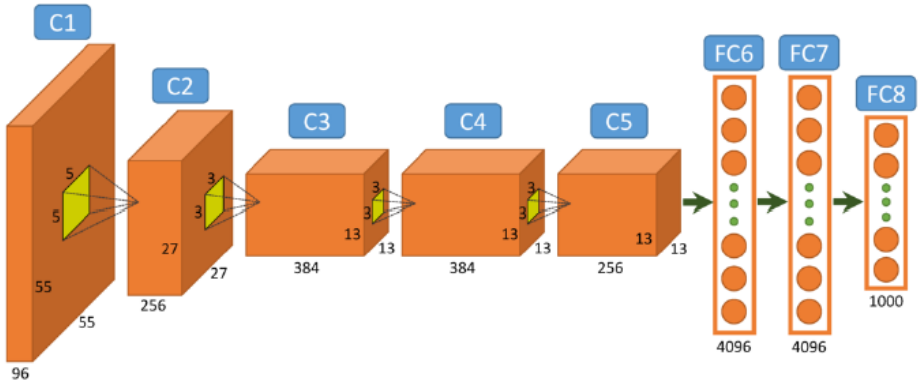


Figure 19: A simplified version of the AlexNet CNN architecture [21].

To convert $FC6$ to a convolutional layer, $C6$, we first need to consider its input which is of size $256 \times 6 \times 6$, due to a pooling layer which down-samples $C5$ from size $256 \times 13 \times 13$ to $256 \times 6 \times 6$. We use the number of neurons in $FC6$, 4096, as the number of feature maps in our new convolutional layer, $C6$. Each kernel of $C6$ needs to cover the entire input of $FC6$, and as such we will have kernels of size $256 \times 6 \times 6$. The full weight matrix for $C6$ will be $4096 \times 256 \times 6 \times 6$, and using the same activation function used for $FC6$ we run the convolution between the input and the filters to get a result of size $4096 \times 1 \times 1$. This convolutional layer can be used to replace $FC6$.

We follow the same process to create the convolutional layer $C7$ which will replace $FC7$. The input for $FC7$ has size $4096 \times 1 \times 1$, and as such the convolutional layer $C7$ will have kernels of size 1×1 with 4096 channels, and feature maps of size $4096 \times 1 \times 1$.

Next we move on to the softmax output layer, $FC8$, which for the AlexNet architecture corresponds to a classification task with 1000 classes. This will be replaced with a convolutional softmax layer $C8$, which will receive and input of size $4096 \times 1 \times 1$, use 1×1 kernels with 4096 channels and will output $1000 \times 1 \times 1$ sized feature maps.

Replacing $FC6$, $FC7$ and $FC8$ with $C6$, $C7$ and $C8$ results in an architecture that is equivalent to the original CNN without the size restrictions. Because fully convolutional layers do not contain fully connected layers, they are able to accept input images of any size large enough for valid convolutions to be performed across the architecture. While conventional CNNs produce single labels as outputs, the output of an FCN is a full map that can be the same size as the input image.

Skip connections

The first fully convolutional networks were created by simply implementing the process above, i.e. converting all fully connected layers to convolutional layers. The resulting output can be good at detecting the coarse elements of an image but not very good at detecting finer details.

Skip connections take the output from one layer, skip at least one layer in the network, and feed that output to another layer in the neural network [78]. Long et al. [65] explored the use of skip connections which combine the coarse detailed information of higher layers with the finer details of lower layers. These skip connections allow layers to capture both the global and the local image information, and can produce segmentation maps that are more refined.

Deconvolution

For a typical CNN architecture, the resulting classification is obtained from a down-sampled version of the original image, whereas for image segmentation, the output needs to be the same size as the input image. Therefore, after the initial down-sampling, we also need up-sampling layers and this process of up-sampling feature maps is often referred to as deconvolution or fractionally strided convolution. If we assume that a convolution has stride s, then the deconvolution operation will have stride $1/s$ which essentially reverses the convolution operation [65].

FCN architectures

FCN-32, FCN-16 and FCN-8, as shown in Figure 20, are variants of the FCN architecture created by Long et al. [65] whose differences lie mainly in the spatial precision of their outputs. These differences are a result of the variations in both the final convolution stride and the skip connections that create the output segmentation maps.

1. FCN-32: For this architecture a segmentation map is directly produced from conv7, using a deconvolution layer with 64×64 and stride 32. The softmax activation function is then applied over the feature maps produced by the deconvolution layer, resulting in the output labelled FCN-32s.

2. FCN-16: For this architecture, conv7 is up-sampled by a factor of 2 using a deconvolution layer with 4×4 filters and a stride of 2, and summed with pool4. This summation is further up-sampled using a deconvolution layer with 32×32 filters and a stride of 16. Similar to FCN-32, the softmax activation function is applied over the feature maps, creating the output

labelled FCN-16.

3. FCN-8: For this architecture, conv7 is up-sampled by a factor of 4, pool4 is up-sampled by a factor of 2 and summed with pool3. This summation is further up-sampled using a deconvolution layer with a stride of 8. Similar to the FCN-32 and the FCN-16 architectures, the softmax activation function is applied over the feature maps and the final output is labelled FCN-16.

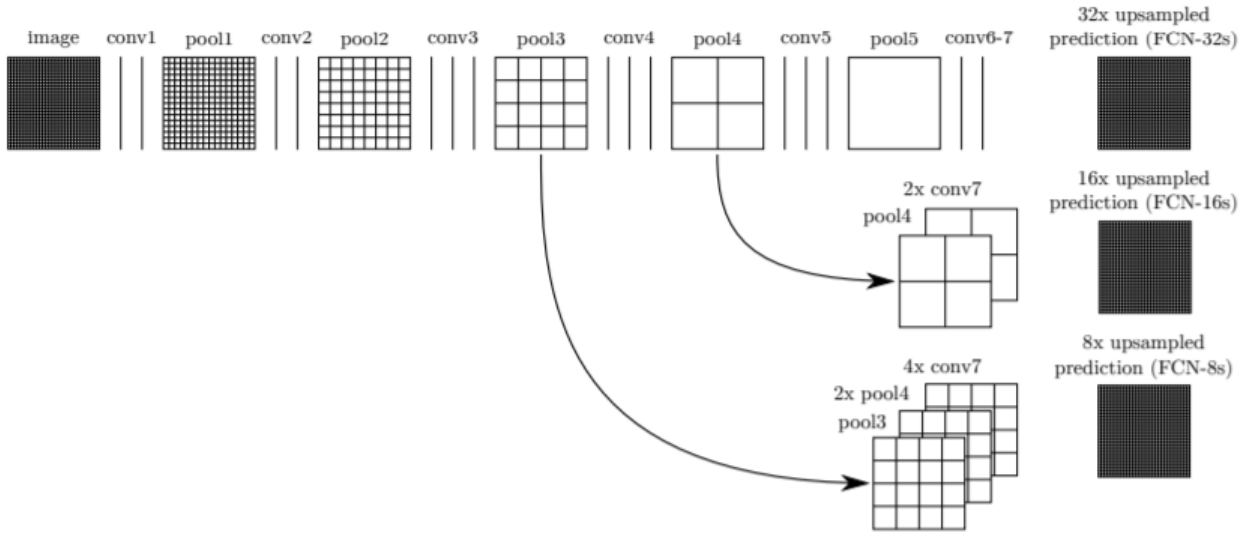


Figure 20: FCN-32, FCN-16 and FCN-8 architectures [65].

The U-Net architecture

The U-Net model is made up of what is termed a 'contracting' and an 'expansion' path [88]. The contracting path is on the left (input) side of the architecture and is made up of five encoder blocks. Four of the five encoder blocks follow the conventional CNN architecture and consist of two 3×3 unpadded convolutions, a max pooling operation with a stride of 2 which is used for down-sampling and the ReLU activation function. The number of filters used are doubled at each down-sampling step. The fifth encoder block is made up of two 3×3 convolutions and the ReLU activation function.

The expansion path is on the right (output) side and is made up of four decoder blocks. The decoder blocks are made up of 2×2 deconvolutional layers with a stride of 2. These deconvolution operations upsample the feature map at every step. The deconvolutions have a stride of 2 and are followed by a concatenation function that concatenates the deconvolution step with the corresponding convolutional function from the contracting path. The 'copy and crop' grey arrows in Figure 21 show this concatenation. This is followed by two 3×3 convolutions with the ReLU activation function.

A 1×1 convolution is used in the final layer to map the feature vector to the number of output classes.

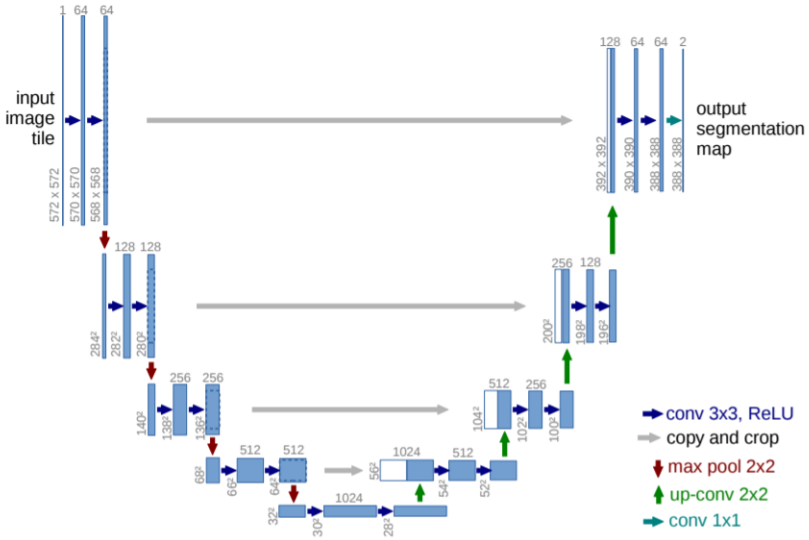


Figure 21: Example of the U-Net architecture for a 32×32 image input. The blue boxes are multiple filter feature maps and the numbers of filters are on top of the boxes. The white boxes represent features that were copied. The arrows are operations as indicated in the legend [88].

3 Landmark detection

This chapter is divided into five sections. Section 3.1 describes the architectures of the models that were trained for this research. Section 3.2 details the software used to train models. Section 3.3 discusses the parameter search that was conducted and gives an overview of the parameters that were tuned during model training. Section 3.4 defines the evaluation metrics we used, and Section 3.5 describes the pre-processing steps followed for each model that we trained.

3.1 Model architectures

The following architectures were experimented on in this research.

1. ANN

A three-layer ANN model made up of an input, a hidden layer (500 neurons) and an output layer was used as a baseline. The model input is a 320×320 greyscale image, flattened to a single vector, and the output is a 22-dimensional vector containing (x, y) coordinates for each of the 11 landmarks. The model incorporated the ReLU activation function, and was trained with the stochastic gradient descent (SGD) optimizer with a learning rate of 0.0001 and momentum of 0.9. The mean square error (MSE) was used as a loss function.

2. CNN

Various LeNet style architectures with dropout and batch normalization were trained, the best of which was a model with three 3×3 convolutional layers, three 2×2 average pooling layers, two 500-neuron and one 84-neuron fully connected layers, and six dropout layers ($p = 0.1$). Each of the three convolutional layers was followed by an average pooling layer, ReLU activation and a dropout layer. The fully connected layers were followed by ReLU activation and dropout layers.

3. FCN

A U-Net style fully convolutional model and a number of FCN8 models were trained. The U-Net model has the same architecture as the one described in Section 2.4. The FCN model architectures are the same as the architecture explained in Section 2.4. During training the batch size and the optimizer were tuned on the validation dataset to improve performance, and the FCN8 model with the Adam optimizer and a batch size of 32 was the best of the FCN models.

3.2 Software implementation

All data pre-processing was done in a Python Jupyter Notebook running on a CPU.

All neural network models used in this research were built in Python using the Keras application program interface (API) [20] and the TensorFlow library [1].

Model training for the ANN and CNN models was conducted using the free GPU resources provided by Google Colab [14].

The FCN models required more computational power and were trained on Google Cloud Platform's AI Notebook instance [13] on two Nvidia Tesla K80 GPUs.

3.3 Parameter search

Neural networks can be difficult to design due to the number of hyperparameters that need to be set. For a standard neural network, parameters such as the number of hidden layers, the number of nodes in each hidden layer, the learning rate and the optimizer need to be set. For more complex architectures like the CNN and FCN architectures that are used in this research the number of hyperparameters increases as filter sizes, stride lengths, padding and number of convolutions are added to the list.

Training and evaluating all possible hyperparameter combinations is infeasible. As such, a few values were considered for each hyperparameter. Hyperparameter optimization for neural networks is still an active area of research and there are more effective ways of exploring and using hyperparameters for model training [12].

For this research standard LeNet, U-Net and FCN architectures were adopted and the parameters that were tuned were the optimizer, batch size, and the application of regularization. Where dropout was applied, the dropout rate was also tuned.

3.4 Evaluation metrics

The work in this section references material from [33]. To assess how well our trained models fit our test data, we consider two regression metrics namely the root mean square error (RMSE) and the mean absolute error (MAE).

Root mean square error

To explain the root mean squared error we start with the mean squared error which is given by the formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.1}$$

where $\hat{y}_i$ is the prediction that the model gives for the i^{th} observation, y_i is the true output value of that observation, and n is the number of observations.

The MSE calculates the squared difference between the response values predicted by the regression model and the true response values. The closer the predictions from the regression model are to the true response values, the lower the MSE value. Because differences are squared, the MSE penalizes observations with large differences much more than the advantage it gives to those observations that are close to the true response value.

The root mean squared error (RMSE) is the square root of the MSE and has the same unit of measurement as the predicted variable.

Mean absolute error

The mean absolute error is given by the formula:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{3.2}$$

This statistic calculates the absolute difference between the response values predicted by the regression model and the true response values. The closer these values are to each other, the lower the MAE value. The MAE is more robust to outliers than the RMSE.

3.5 Data pre-processing

The RGB images were resized from their original 1280×1024 resolution to 320×320, and converted to greyscale, decreasing the number of input parameters for each image from $1280 \times 1024 \times 3 = 3932160$ to $320 \times 320 = 102400$. Figure 22 is an example of a resized greyscale image.

We then created a classification algorithm to detect whether a wing was right or left facing, and to ultimately flip all left facing images so that all images in the dataset are right facing. This was done to increase the training images for our model while making the landmark detection problem slightly less complex.

Figure 22: Resized greyscale tsetse fly wing image.

After these pre-processing steps the dataset was split randomly into training, validation and test data, using a 60:20:20 split (approximately). Table 1 shows the size of each set.

	Images
Train set	1452
Validation set	484
Test set	484

Table 1: Sizes of training, validation and test datasets.

3.5.1 ANN model

Input variable

For the ANN model the image pixels were normalized so that all values are in the range $(0, 1)$

and flattened in order to transform the 320×320 pixel matrix into a 102400-dimensional vector. The final training matrix for the simple ANN model has the shape $(1452, 102400)$.

Response variable

The target outputs were standardized to ensure that all response variable values range between $(-1, 1)$. The final training response variable has the shape $(1452, 22)$.

3.5.2 CNN model

Input variable

Image pixels were normalized and the images were reshaped to (320,320,1). The 1 in this case is the channel element, indicating that the images are greyscale. The final training matrix for the CNN models has the shape $(1452, 320, 320, 1)$.

Response variable

The target outputs were standardized to ensure that all response variable values range between $(-1, 1)$. The final training response variable has the shape $(1452, 22)$.

3.5.3 FCN model

Input variable

The image data was pre-processed the same as it was for CNNs. The pixels for each image were normalized, ensuring pixel values lie in the range (0,1), and the final training matrix shape for the FCN models was $(1452, 320, 320, 1)$.

Response variable

Pre-processing for the target outputs was more involved. FCNs are traditionally used for segmentation and the response variables for segmentation problems are usually images that are made up of single channel per-pixel classification values, commonly referred to as image segmentation masks [50]. Image segmentation masks group pixels using the objects in an image. All pixels in one object are assigned the same value and are considered one class. An example of an image and its segmentation mask is shown in Figure 23. Here the classes identified for the image are person, purse, plants/grass, sidewalk and building/structures. Each of these classes have associated pixel values and the semantic label is a representation of the segmentation mask that will be used during training. In practice the segmentation mask can have the same number of pixels and dimensions as the input image.

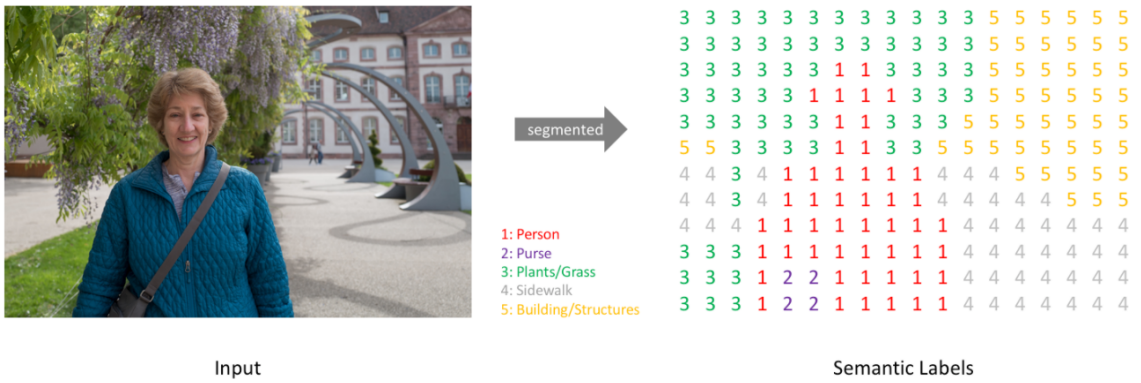


Figure 23: Image segmentation input image and an example low resolution segmentation mask [50].

Similar to how we handle categorical variables, the semantic labels need to be one-hot encoded as a pre-processing step. This means that the segmentation mask in Figure 23 with five classes needs to be converted to five segmentation masks, where each mask only contains non-zero values for one class and all other pixel values are 0. An example of this is shown in Figure 24.

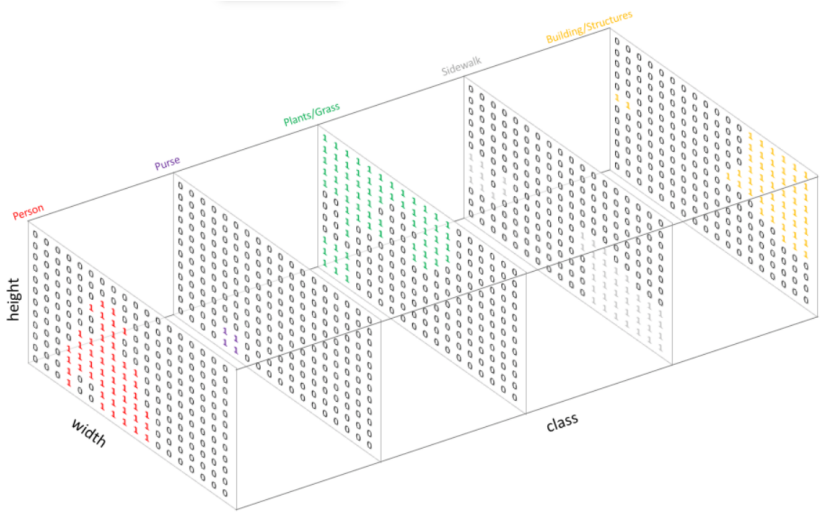


Figure 24: An example of how to one-hot encode the semantic labels displayed in Figure 23 [50].

Figure 25 is an example of a low resolution representation of an image from our dataset with its semantic labels. In our case each of the 11 landmarks represents a class. We have used a white space to represent unclassified pixels to help make the image clearer.

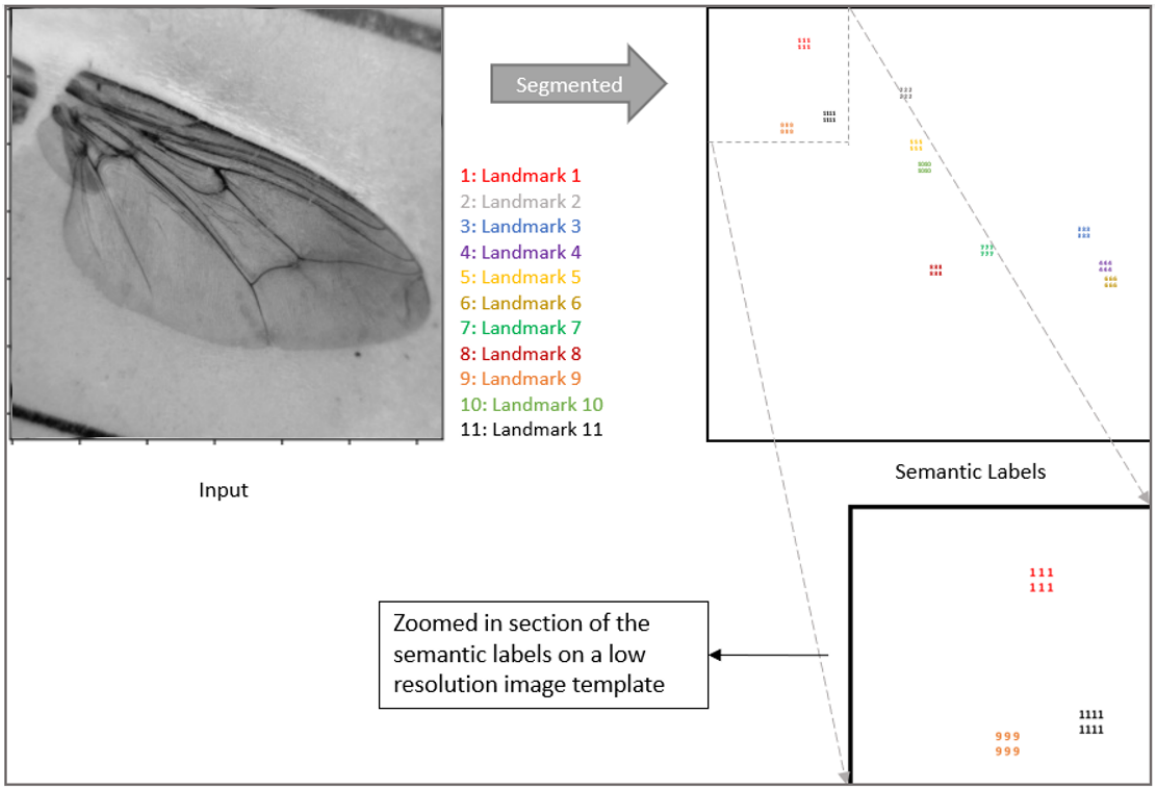


Figure 25: Input image and an example low resolution segmentation mask. The white areas are unclassified and have a pixel value of 0.

Converting coordinate points to segmentation masks

We need to convert the target response coordinates into a semantic mask format, as illustrated in Figure 23. We will treat each of the 11 coordinate pairs (x, y) as a class and convert each point into a heat map using the Gaussian function. We transform each coordinate point into a heat map of size 320×320, which corresponds to the size of the input image. Figure 26 gives an example of the conversion of coordinate pair $(4, 4)$ in an 8×8 image into a heat map of size 8×8. The Gaussian function was used for this conversion and calculations were performed at each coordinate point in a window around the point $(4, 4)$.

We demonstrate below how we calculated the pixel value at point $(3, 3)$ when generating the heat map centred at $(4, 4)$ for an 8×8 image. In this case we have $(x_0, y_0) = (4, 4)$ and we have chosen a standard deviation σ of 0.6. The value for pixel (3,3) is calculated as follows:

$$pixel_{(x,y)} = \exp\left[-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right] \times 255 \tag{3.3}$$

$$\therefore pixel_{(3,3)} = 15.3 \tag{3.4}$$

The Gaussian exponential returns a result in the range $(0, 1)$ which we multiply by the constant value 255 to ensure the answer is on the pixel scale. As shown in Figure 26, through this calculation the point coordinate (4,4) is transformed into a heat map.

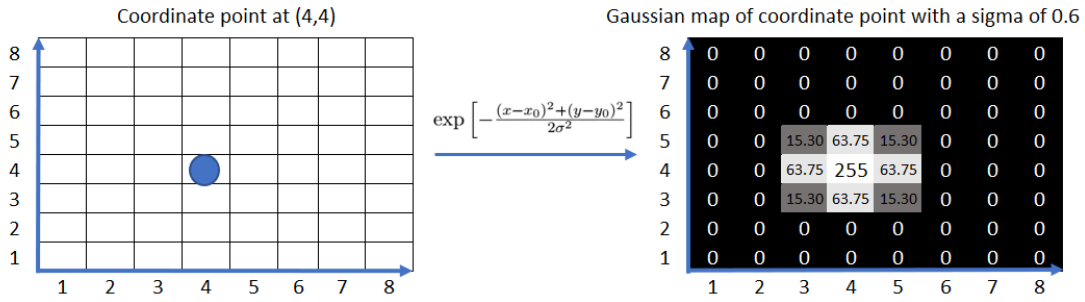


Figure 26: A simple example demonstrating how the coordinate point (4,4) can be converted to a heat map using the Gaussian function.

These calculations are carried out for each of the 11 landmark annotations on each one of the images in the dataset, and 11 heat maps are generated for each image. Figure 27 shows three example images and the 11 heat maps that were generated for each annotated landmark point on each image. A σ value of 15 was chosen to generate the heat maps, based on how small we wanted the illuminated part of the heat maps to be. A better way might be to make σ a hyperparameter that can be optimized during model training and validation.

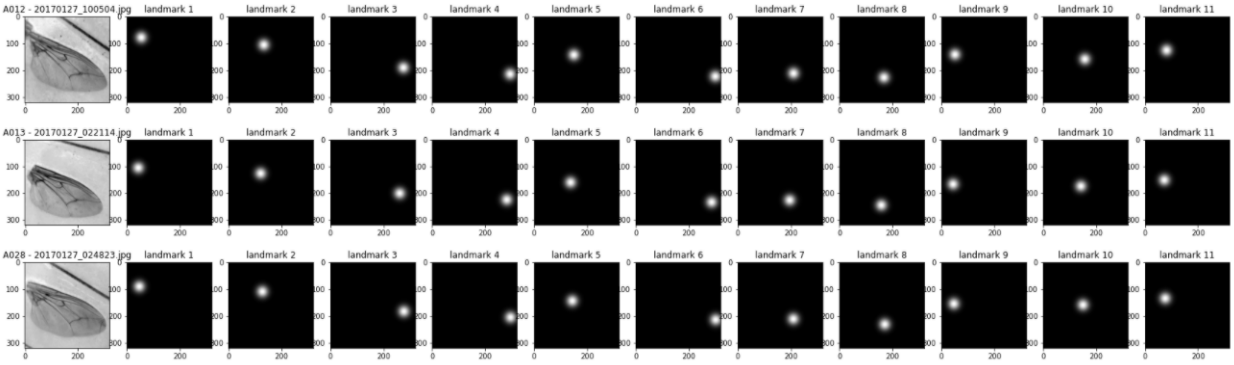


Figure 27: Three example images from our training set with 11 heat maps generated from the 11 landmark annotations for each image.

The above procedure was used to convert the training response variable from $(1452, 22)$ into a $(1452, 320, 320, 11)$ matrix.

4 Results and discussion

This chapter is divided into three sections for the three model architectures that were investigated. Section 4.1 gives a high-level overview of the model architecture and training results for the ANN model which was used as our baseline. Section 4.2 gives an overview of the various LeNet style CNN architectures that were trained, the training results for the best models, the methods explored to prevent overfitting as well as the results associated with these methods. Section 4.3 details our FCN post-processing approach, as well as the various FCN models that were trained and their results.

4.1 Model A: ANN

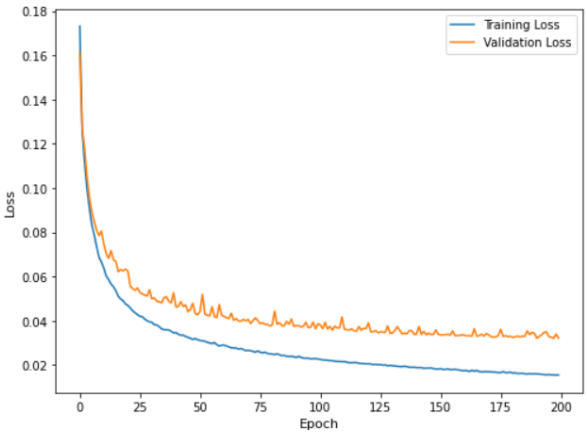
4.1.1 Architecture

To create a baseline we started with a simple ANN model with one hidden layer. The input layer of this model accepts a 320×320 greyscale image, the hidden layer has 500 nodes with the ReLU activation function, and the output layer has 22 nodes that are aligned with the 11 landmarks. The SGD optimizer with a learning rate of 0.0001 and momentum of 0.9 was used, and the mean square error (MSE) was used as the training loss function. We further split the training data using the 80:20 split, where 1162 of the dataset images were used for training and the remaining 290 were used for validation during training. This is our baseline model.

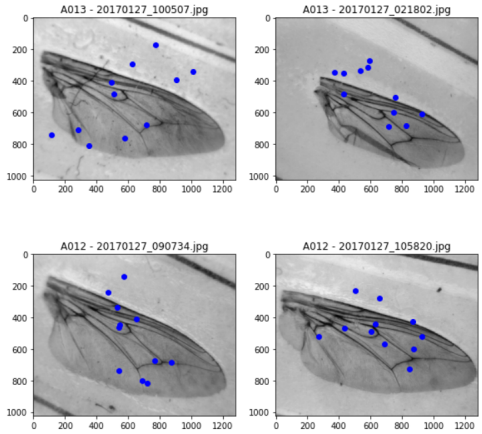
4.1.2 Model training and results

Test set results for this simple ANN network are as follows: RMSE = 282.62, MAE = 181.33.

Figure 28 (a) shows the training and validation loss functions for this model. The model was trained on a GPU enabled Colab instance, for a training time of approximately 3 minutes. It reached a validation loss of 0.0332 and a training loss of 0.0171. The gap between the training loss and the validation loss suggests that the model is overfitting slightly, which means it is not generalizing well. Figure 28 (b) superimposes model predictions onto example images from the test set, demonstrating model fit results.



(a) ANN training and validation loss by training epoch.



(b) ANN model performance results on example test images.

Figure 28: ANN model training and validation loss, and performance on example test images.

Figure 29 displays the mean absolute error between the true (x, y) coordinates and the landmark locations predicted by the ANN model for the 484 test images. Overall, the ANN model struggled to fit x variables more than to fit the y variables, i.e. the average absolute error between the predicted values and the true values was much higher for x variables than it was for the y variables. This might be because of the extra compression that we had to do on the x axis to convert the image from a rectangular (1280×1024) to a square (320×320) image. The model was the worst at predicting the (x_6, y_6) pair with a mean absolute error of 574.71 for x_6 and 89.92 for y_6. It was best at predicting (x_5, y_5) with a mean absolute error of 92.41 and 49.38 for x_5 and y_5 respectively. The landmark point corresponding to y_5 is typically more pronounced and located in a higher contrast part of the image than the other landmarks, which may be the reason why the model was much better at predicting this point overall, while the lines marking the (x_6, y_6) pair are difficult even for the human eye to localize.

x1	y1	x2	y2	x3	y3	x4	y4
435.3975	57.694014	118.420098	61.880529	466.121688	73.535595	556.605229	76.659119

x5	y5	x6	y6	x7	y7	x8	y8
92.411377	49.377339	574.712433	89.916328	186.170571	56.156716	104.424371	55.972777

x9	y9	x10	y10	x11	y11
455.88378	68.399889	78.686861	68.617729	324.1357	51.388692

Figure 29: Mean absolute difference between the true annotations and the landmark coordinates predicted by the ANN model, over the 484 test images.

The landmarks of interest from an entomology perspective, from those illustrated in Figure 1, are the coordinate pairs (x_1, y_1), (x_6, y_6), (x_7, y_7) and (x_{10}, y_{10}). The coordinate pair (x_1, y_1) was amongst the pairs the model struggled the most to predict with mean absolute errors of 435.40 and 57.70 for x_1 and y_1 respectively. The (x_7, y_7) pair had a mean absolute error of 186.17 and 56.16 for x_7 and y_7 respectively. For the (x_{10}, y_{10}) coordinate pair the mean absolute error between the true coordinates and the ANN model predictions were 78.69 and 68.62 for x_{10} and y_{10} respectively, which was the second best coordinate pair predictions for this model.

4.2 Model B: CNN

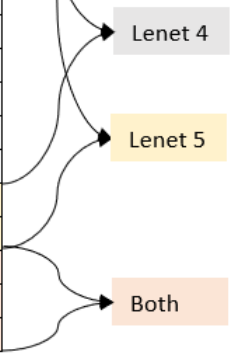
4.2.1 LeNet models

LeNet style architectures typically have a number of convolutional layers, followed by maximum or average pooling layers, and then fully connected layers as the last few layers of the network. Our model architectures were modelled after the standard LeNet architecture [59]. We experimented with a number of convolutional layers, training LeNet3, LeNet4 and LeNet5 models whose architectures are shown in Figure 30. In an effort to avoid overfitting we added dropout as a regularization technique, and further added batch normalization. We found that batch normalization did not improve RMSE or MAE for the validation set, in any of the CNN models, and as a result this regularization technique was excluded from our final models. Figure 30 provides a detailed description of the top three LeNet architectures we experimented with. (a) describes the 3-layer LeNet with dropout and (b) describes LeNet4 and LeNet5. In Figure 30 (b) all layers 0 to 13 form part of the LeNet5 architecture and to obtain the LeNet4 architecture, layers 9 to 10 are to be excluded.

Layer	Name	Size
0	Input	320x320x1
1	conv2d1	316x316x6
2	avg_pool2d2	158x158x6
3	dropout3	158x158x6
4	conv2d4	157x157x16
5	avg_pool2d5	78x78x16
6	dropout6	78x78x16
7	conv2d7	77x77x32
8	avg_pool2d8	38x38x32
9	dropout9	38x38x32
10	dense7	500
11	dense8	84
12	output	22

(a)

Layer	Name	Size
0	Input	320x320x1
1	conv2d1	316x316x6
2	avg_pool2d2	158x158x6
3	conv2d3	157x157x16
4	avg_pool2d4	78x78x16
5	conv2d5	77x77x32
6	avg_pool2d6	38x38x32
7	conv2d7	37x37x64
8	avg_pool2d8	18x18x64
9	conv2d9	17x17x128
10	avg_pool2d10	8x8x1288
11	dense7	500
12	dense8	84
13	output	22



(b)

Figure 30: LeNet style architectures that were experimented with.

The results for the top three performing LeNet architectures based on the RMSE and MAE from our validation set are shown in Table 2.

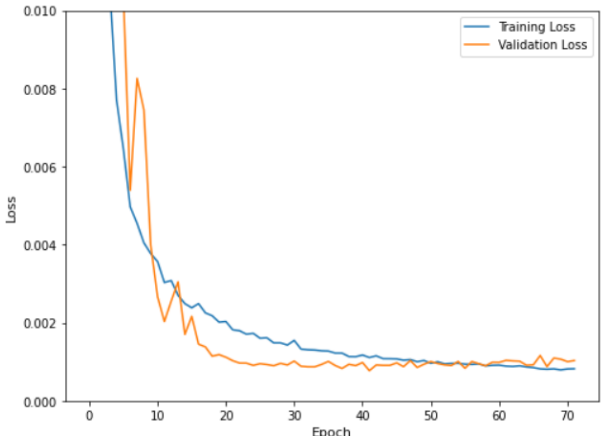
Model	Optimizer	Training loss	Validation loss	RMSE	MAE
LeNet3 + dropout	Adam	0.00079	0.0009	53.5810	41.0500
LeNet4	Adam	0.00139	0.0098	54.3842	41.9449
LeNet5	Adam	0.00197	0.0098	54.7572	42.4231

Table 2: Training results from the three best performing LeNet models. RMSE and MAE are measured on the validation set.

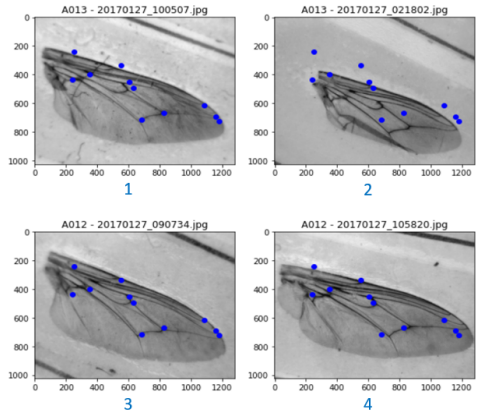
The LeNet3 model with dropout performed best. This model was trained using the Adam optimizer, no padding, the ReLU activation function for nonlinearity, and mean squared error (MSE) was used as the loss function. To ensure that we minimized training time and avoid overfitting, the early stopping technique was used to determine the number of training epochs.

4.2.2 Model training and results

Training and validation loss over number of epochs, as well as model performance on example test images for the LeNet3 model with dropout are shown in Figure 31 (a) and (b). This model was trained on a GPU enabled Colab Notebook for approximately 8 minutes. The validation loss started out as unstable in the beginning of training, then decreased sharply until it was lower than the training loss. Around the 60-epoch mark, the training loss became lower than the validation loss and the latter stabilized at around 0.008145. Model predictions were superimposed onto test images, and results for some of the best and worst performing images are shown in Figure 31 (b).



(a) LeNet3 + dropout training and validation loss by training epoch.



(b) LeNet3 + dropout performance results on example test images.

Figure 31: LeNet3 + dropout training and validation loss, and example results when predictions are superimposed on test images.

For the four example images shown in Figure 31 (b) the model best fits image number 3. This might be because the wing is not folded and is well centred in the image frame. The image with the worst fit is image 2. The wing in image 2 is folded and its shape slightly distorted, and the model was unable to detect the landmarks on that wing.

Figure 32 displays the mean absolute error between the true (x, y) coordinates and the landmark locations predicted by the LeNet3 model with dropout for the 484 test images. The model was the worst at predicting the (x_6, y_6) pair with a mean absolute difference of 44.93 for x_6 and 50.19 for y_6. It was best at predicting (x_{10}, y_{10}) pair with a mean absolute difference of 37.98 for x_{10} and 35.26 for y_{10}. The landmark point corresponding to (x_{10}, y_{10}) is more pronounced and is located in a higher contrast part of the image than the other landmark points, which may be the reason why the model was much better at predicting this point overall, while the lines marking the (x_6, y_6) pair would be difficult even for a human labeller to locate accurately.

x1	y1	x2	y2	x3	y3	x4	y4
44.436172	44.60685	40.394287	36.246876	43.752652	45.074976	44.488118	49.092354

x5	y5	x6	y6	x7	y7	x8	y8
38.138513	35.280575	44.934306	50.187156	38.779538	37.436802	38.316593	36.435097

x9	y9	x10	y10	x11	y11
39.12819	44.285222	37.975948	35.262146	38.950951	40.047301

Figure 32: Mean absolute difference between the true annotations and the landmark coordinates predicted by the LeNet model with dropout, over the 484 test images.

From an entomology perspective, we look at the coordinate pairs of interest (x_1, y_1), (x_6, y_6), (x_7, y_7) and (x_{10}, y_{10}). The coordinate pair (x_1, y_1) was amongst the pairs the model struggled most to predict with mean absolute errors of 44.44 and 44.61 for x_1 and y_1 respectively. The (x_6, y_6) pair had the worst mean absolute error for this model. (x_7, y_7) had a mean absolute difference of 38.78 and 37.44 for x_7 and y_7 respectively, which was among the best mean absolute errors for this model. For the (x_{10}, y_{10}) coordinate pair the mean absolute error between the true (x, y) coordinates and the model predictions were the best for this LeNet model with dropout.

The model overfits the training data, and the resulting predictions for different images are very similar, with individual predictions between images only differing by a few decimals. This means that while the model may do well on this particular dataset, it will not generalize well to other datasets where the wings are positioned differently in the images.

We proceed with additional experiments that might help the model to better generalize. We consider common methods that have been shown to reduce overfitting and help improve model performance. We look at the following methods:

- getting more training data;
- increasing the dropout rate.

4.2.3 Getting more training data

We created more training examples by translating all the images in the training dataset by a predefined number of pixels. We created copies of all the training images that were translated 10 pixels up, 10 pixels down, 10 pixels left and 10 pixels right; we renamed the images and translated the training annotations accordingly. Figure 33 shows an example of the translated images with their annotations.

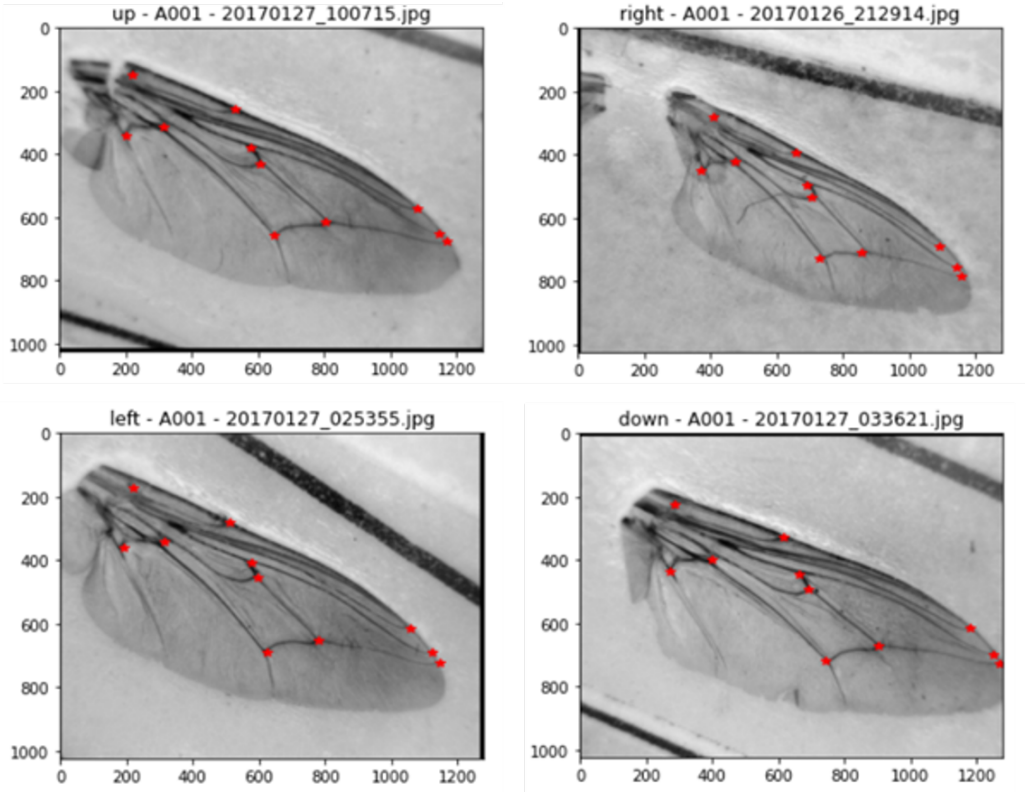


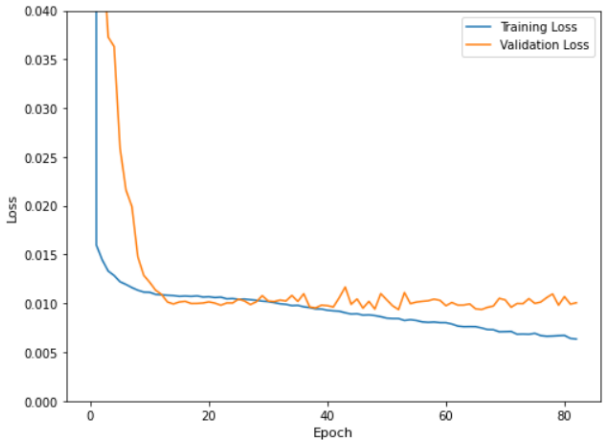
Figure 33: Four example images showing the additional images that we created by translating an image from the dataset 10 pixels up, 10 pixels to the right, 10 pixels to the left and 10 pixels down.

We used a combination of the original images and the translated images to create a new, augmented training dataset. Table 3 lists the number of images in our augmented training dataset.

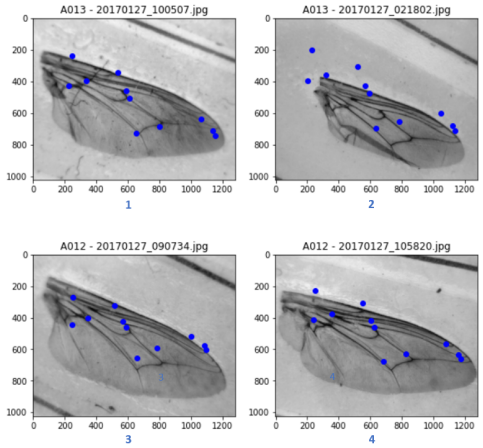
Translation	Data examples count
No translation (original images)	1452
Left translated	363
Right translated	363
Translated upwards	363
Translated downwards	363
Total	2904

Table 3: The data composition for the augmented training set that includes translated images.

Figure 34 (a) shows the training and validation loss over epochs when training the LeNet3 + dropout model with the larger dataset. It shows that the model was once again attempting to overfit the training data when early stopping kicked in. This model was trained on a GPU enabled Colab Notebook for approximately 6 minutes. The model did not fit the data as well as the original LeNet3 + dropout model, as can be seen in the four examples in Figure 34 (b). Like the LeNet3 + dropout, this model overfits the data and has lower RMSE and MAE values of 54.23 and 41.69 respectively.



(a) Training and validation loss for the LeNet3 + dropout model trained on the larger dataset that includes translated images.



(b) Performance of the LeNet3 + dropout model trained on the larger dataset, on example test images.

Figure 34: LeNet3 + dropout training and validation loss, and example results when predictions are superimposed on test images, for the larger dataset.

Figure 35 displays the mean absolute error between the true (x, y) coordinates and the landmark locations predicted by the LeNet3 model with dropout trained on the larger dataset for the 484 test images. While this model performed worst overall, it was good at predicting the (x_8, y_8) coordinate pair and the y_5 variable with mean absolute errors of 38.53 for x_8, 36.38 for y_8 and 35.52 for y_5.

x1	y1	x2	y2	x3	y3	x4	y4
46.527176	45.241062	41.56581	36.502353	43.87762	45.138025	44.404597	49.258391

x5	y5	x6	y6	x7	y7	x8	y8
40.210664	35.521517	44.809253	50.266793	39.045817	37.446295	38.531426	36.383558

x9	y9	x10	y10	x11	y11
41.001917	45.005868	39.598566	35.543393	40.735899	40.567274

Figure 35: Mean absolute difference between the true annotations and the landmark coordinates predicted by the LeNet model with dropout trained on a larger dataset, over the 484 test images.

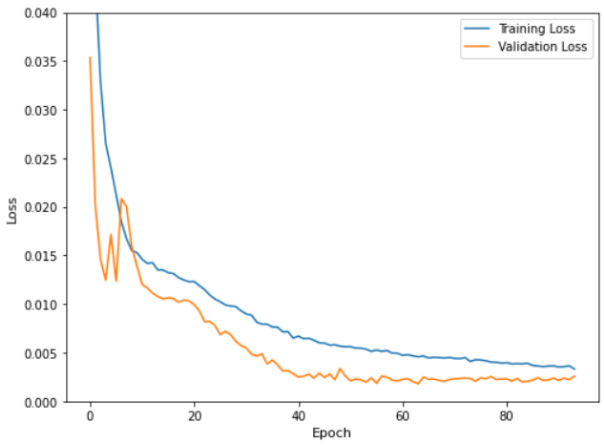
From an entomology perspective, we looked at the coordinate pairs of interest (x_1, y_1), (x_6, y_6), (x_7, y_7) and (x_{10}, y_{10}). x_1 and y_6 showed the worst mean absolute errors of 46.53 and 50.27 respectively. The pairs (x_7, y_7) and (x_{10}, y_{10}) had some of the best mean absolute errors of 39.05 for x_7, 37.45 for y_7, 39.60 for x_{10} and 35.54 for y_{10}. As stated previously the positioning and thickness of the veins that intersect to create landmark point 10 makes it easier to identify while the faint appearance of the veins associated with landmark point 6 makes it difficult even for the human eye to localize.

Because increasing the training examples did not improve model performance, we maintain after this experiment that the LeNet3 model with dropout, trained on the original 1452 training images, is still our best model and further experiments will be conducted on that model.

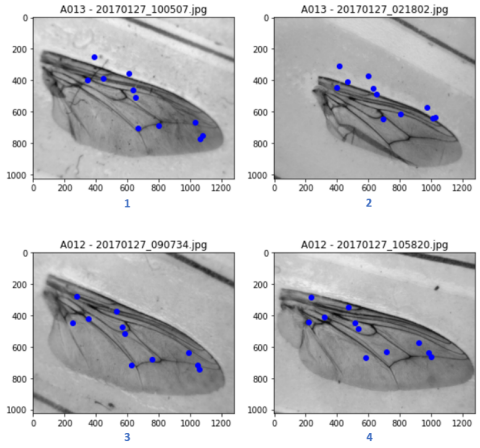
Our next experiment involves an increase of the dropout rate in a further attempt to reduce the model from overfitting.

4.2.4 Increasing dropout rate

In an attempt to stop the model overfitting, we increased the dropout rate from 0.1 to 0.5 at each layer of the LeNet3 architecture. This model was trained on a GPU enabled Colab Notebook for approximately 8 minutes. Training and validation loss for this model are displayed in Figure 36 (a). The resulting model did not overfit the data, however the model performed slightly poorer on the test data. The RMSE for this model was 65.46 and the MAE was 52.71. This indicates that increasing the dropout rate decreased model performance. The performance of the model can also be seen by how it fits the example test images in Figure 36 (b).



(a) Training and validation loss of LeNet3 with increased dropout, by training epoch.



(b) LeNet3 with increased dropout performance results on example test images.

Figure 36: LeNet3 with increased dropout training and validation loss, and example results when predictions are superimposed on test images.

Figure 37 displays the mean absolute error between the true (x, y) coordinates and the landmark locations predicted by the LeNet3 model with increased dropout for the 484 test images. Overall the model was worst at predicting the coordinate pair (x_1, y_1) with mean absolute error of 61.60 and 54.12 for x_1 and y_1 respectively. The (x_8, y_8) coordinate pair had some of the best mean absolute errors of 41.00 for x_8 and 45.64 for y_8. The y_8 variable was surpassed only by y_{10} which had the best mean absolute error of 44.16 over the 484 test images.

x1	y1	x2	y2	x3	y3	x4	y4
61.601006	54.116476	52.589602	47.064934	46.534131	48.416416	45.432156	52.515931

x5	y5	x6	y6	x7	y7	x8	y8
48.919597	45.046217	45.584788	53.497094	42.145835	44.019811	41.002263	45.644354

x9	y9	x10	y10	x11	y11
53.832116	58.566994	46.056726	44.159688	53.473952	55.648603

Figure 37: Mean absolute difference between the true annotations and the landmark coordinates predicted by the LeNet model with increased dropout, over the 484 test images.

From an entomology perspective, we look at the coordinate pairs of interest (x_1, y_1), (x_6, y_6),

(x_7, y_7) and (x_{10}, y_{10}). The model was worst at predicting the coordinate pair (x_1, y_1). This might be because there are a lot of veins in the region where (x_1, y_1) is positioned, making this point difficult to locate. The variables x_6, y_6, x_7, y_7 and x_{10} had mean absolute error values of 45.58, 53.50, 42.15, 44.02 and 46.06 respectively. y_{10} had the best mean absolute error value for the y variables of 44.16.

In the real world the model's ability to generalize to new contexts would weigh heavier than how well the model is fitting the validation set while training. However, given that RMSE and MAE are what we are using to assess the models, we still maintain that the LeNet3 model with 0.1 dropout rate is the best due to its RMSE and MAE values.

4.3 Model C: FCN

The standard FCN8 architecture described in Section 2.4 is used as the model architecture. All the models in this section used same padding, max pooling, the ReLU activation function and the MSE loss function. We experimented with the batch size and the optimizer.

4.3.1 Post-processing

The FCN model outputs 11 heat maps for an input image. To calculate the RMSE for this model, we need to convert these heat maps to (x, y) coordinates. Figure 38 is an example output of a test image with the predicted and the true heat maps for each of the 11 landmark points.

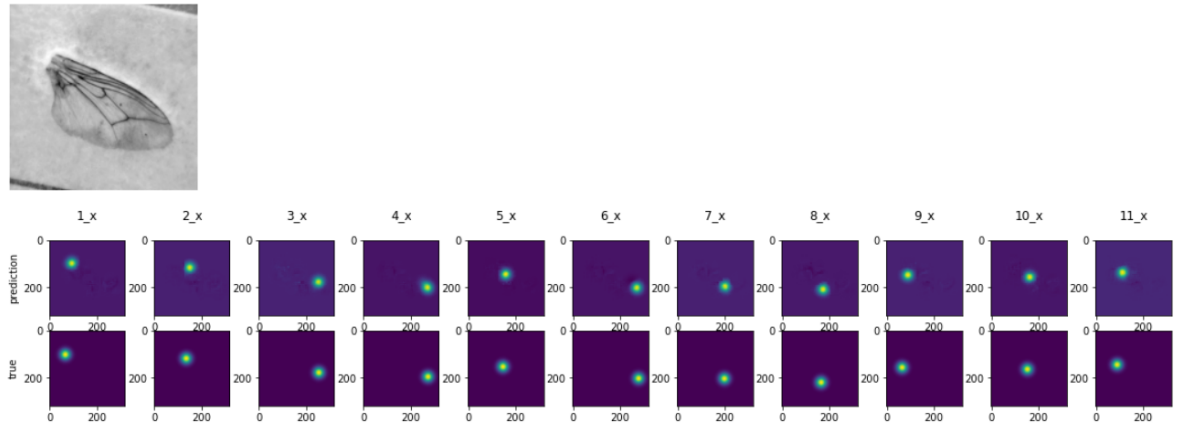


Figure 38: An example of the heat maps predicted by one of the FCN8 models (Adam optimizer and a batch size of 32), and the true heat maps.

The simplest way to convert a heat map to a coordinate pair would be to find the (x, y) coordinates that correspond to the largest pixel value, and use that as the final answer. The problem is that the coordinates we would get with this method will be integer values while the true coordinate values are not necessarily integer values.

We may instead use a weighted average of a neighbourhood around the highest pixel value as our final (x, y) coordinates. Here we would need to decide on the number of neighbouring pixels to consider when calculating the weighted average. Our goal is to find the (x, y) coordinates that minimize RMSE on the validation set. We therefore chose a set of numbers to test with, which we will refer to as N. We decided to limit the size of N, and we chose squares of numbers between 1 and 9, that is $N = \{1, 4, 9, 16, 25, 36, 49, 81\}$.

To find the best value for N, we perform the following:

1. retrieve the true (x, y) coordinates of the landmarks in images from the validation set;
2. for each n in N:

   calculated the RMSE between the true (x, y) coordinates and the (x, y) coordinates for the true heat maps that we generated (see Section 3.5.3), using n as the number of neighbouring pixels to use in the weighted average calculation;
3. finally, compare RMSE values and choose the n that returns the lowest RMSE value.

This final value of n was used in our weighted average calculation to convert our heat maps into (x, y) coordinate point predictions for the test dataset.

We demonstrate below how we calculate the weighted average and show that the final coordinate points that are derived from a weighted average calculation for non-integer coordinate points is likely to yield a lower RMSE score than defaulting to the highest pixel value point.

To do this we will consider the example in Figure 39. We assume the true coordinate point is (3.8,4.4), and we will use two approaches, the simple approach and the weighted average approach, to convert the heat map to (x, y) coordinates. We will calculate the RMSE for the resulting coordinate points to illustrate the advantage of using weighted averages when working with coordinate points that are non-integer.

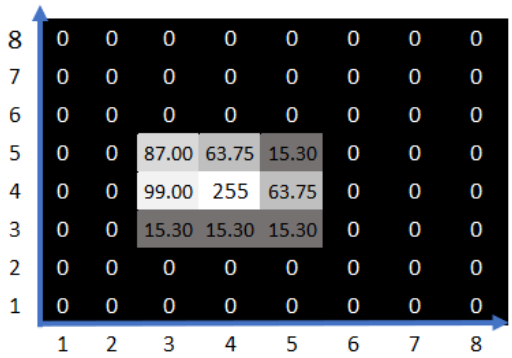


Figure 39: Example FCN8 prediction heat map for an 8×8 image.

Simple approach

When using the simple method on the example heat map in Figure 39, we look for the highest pixel value in the image. As illustrated in Figure 40, the highest pixel value is 255 and corresponds to the coordinate pair (4,4).

The RMSE for this method is approximately 0.32.

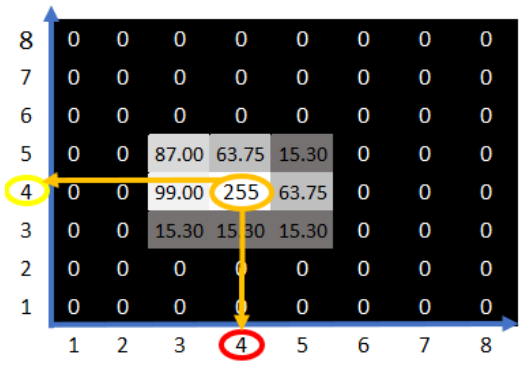


Figure 40: Example heat map and an illustration of the simple method resulting in a final coordinate pair of (4,4).

Weighted average approach

For the weighted average approach, we start by choosing n, the number of neighbouring pixels that will be included in the weighted average calculation. For this example, we set $n = 9$. This implies that all non-zero pixel values will be used in the weighted average calculation.

To convert the pixel values to coordinate points using the weighted average approach we do the following:

- Step 1: A sum of the n pixel values is calculated. The sum of the n pixel values in our example is 629.7.
- Step 2: Each of the n pixel values is divided by the sum (629.7) to create the weight for each pixel value.

  Figure 41 illustrates the pixel value weights, highlighting $x = 5$ and $y = 5$ to show the values that will be added to form the final weight for each coordinate point and the associated coordinate points. The coordinate points and their final weights are shown in Figure 42.
- Step 3: The (x, y) coordinate pair is calculated as the sum product of the point coordinates and their weights.

The resulting coordinate pair for this method is (3.73265, 4.20671). The RMSE for this coordinate pair is approximately 0.14.

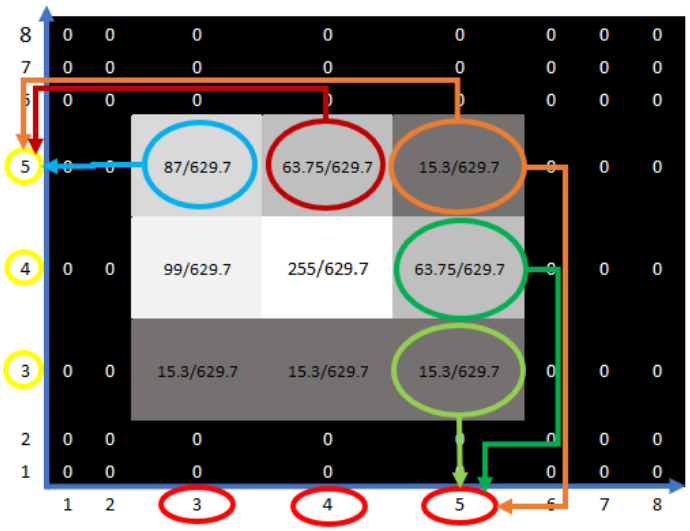


Figure 41: Example heat map illustrating pixel value weights and the corresponding coordinate points whose weight they contribute to. This example only highlights how the pixel value weights contribute towards the weights of $x = 5$ and $y = 5$.

x	weight
3	0.34632
4	0.57471
5	0.07897

y	weight
3	0.07897
4	0.63535
5	0.28568

Figure 42: Coordinate points for x and y and their weights.

4.3.2 Model training and results

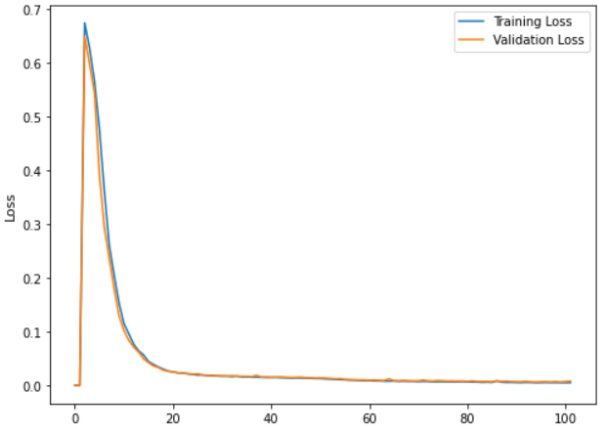
The training loss, validation loss, RMSE and MAE for the models are shown in Table 4. The FCN8 model with a batch size of 32 that used the Adam optimizer returned the best RMSE of 1.12 and MAE of 0.88.

Training and validation losses for this model are shown in Figure 43 (a). This model was trained on two Nvidia Tesla K80 GPUs for approximately 72 minutes. Both the training loss and validation loss started off very low, shot up at around epoch 10 and started decreasing as training progressed until epoch 100 where the final training and validation losses were recorded as 0.0051 and 0.0082 respectively. The validation and training losses were close to each other throughout training and the model does not seem to overfit.

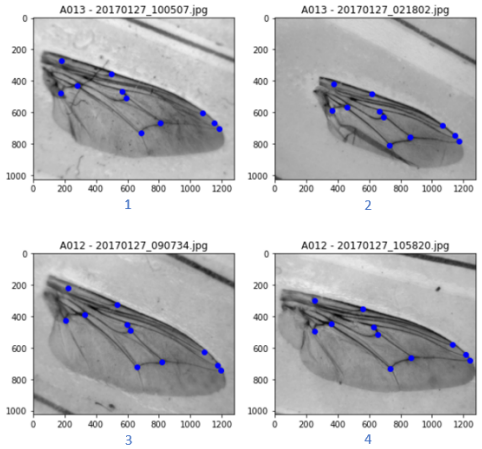
Model	Optimizer	Batch size	Training loss	Validation loss	RMSE	MAE
FCN8	RMSprop	5	0.0093	0.0013	1.78	1.25
FCN8	RMSprop	32	0.0356	0.0373	2.15	1.66
FCN8	Adam	32	0.0051	0.0082	1.12	0.88
U-Net	Adam	32	0.4655	0.4657	8.87	7.22

Table 4: Top four FCN model training results.

Example prediction results from the FCN8 model trained with Adam and a batch size of 32 are shown in Figure 43 (b). Unlike the LeNet models, this model was able to accurately predict the landmarks on wings that are centred and not folded, as well as for those images whose wings are slightly distorted. This is evidenced by how well the model located the landmarks for image number 2 in Figure 43 (b).



(a) Training and validation loss over epochs of the FCN8 model with batch size 32 and the Adam optimizer.



(b) FCN8 model with batch size 32 and Adam performance results on example test images.

Figure 43: FCN8 model with batch size 32 and Adam training and validation loss, and example results when predictions are superimposed on test images.

Figure 44 displays the mean absolute error between the true (x, y) coordinates and the landmark locations predicted by the FCN8 model with batch size 32 and Adam for the 484 test images. Overall the FCN8 model with a batch size of 32 and Adam optimizer predicted the annotations for the 484 test examples near perfectly. The model struggled the most with the prediction of the coordinate pair (x_4, y_4) with mean absolute error values of 2.41 and 4.70 for x_4 and y_4 respectively. The coordinate pair (x_8, y_8) was the best prediction pair with mean absolute error values of 0.82 and 0.65 for x_8 and y_8 respectively, while x_{10} had the best mean absolute error value of 0.78 for all predicted x variables.

x1	y1	x2	y2	x3	y3	x4	y4
0.902608	0.984657	1.101732	0.779811	1.00073	0.714961	2.413326	4.701336

x5	y5	x6	y6	x7	y7	x8	y8
0.929856	0.805761	2.216347	4.040697	0.948144	0.698058	0.820511	0.654451

x9	y9	x10	y10	x11	y11
0.808344	0.989962	0.779883	0.852795	1.401209	1.042191

Figure 44: Mean absolute difference between the true annotations and the landmark coordinates predicted by the FCN8 model with batch size 32 and Adam, over the 484 test images.

From an entomology perspective, we look at the coordinate pairs of interest (x_1, y_1), (x_6, y_6), (x_7, y_7) and (x_{10}, y_{10}). The coordinate pair (x_6, y_6) displayed the second worst mean absolute errors with values of 2.22 and 4.04 for x_6 and y_6 respectively. The variables x_1, y_1, x_7, y_7 and

y_{10} had mean absolute error values of 0.90, 0.98, 0.95, 0.70 and 0.85 respectively. The model's best mean absolute error was for x_{10}.

5 Conclusion

The aim of this research was to compare the performance of FCN architectures against conventional CNN architectures for the regression task of detecting landmarks on tsetse fly wing images with a limited dataset. Landmarks are points where the wing veins intersect, and detecting these points is a first step towards using the wing images to estimate fly size. Once the landmarks are located, distances between them can be used to answer questions in the field of entomology where fly size and fly shape play a major role.

5.1 Summary

To tackle this research we started by exploring the approach that Longpre et al. [66] used for facial key point detection. Detecting landmarks of interest on tsetse fly wing images shares a number of properties with the facial key point detection problem. Our dataset is made up of static images of wings where the main differences between wing images come from biological differences in individual wings, positioning of the wings in the images and illumination. Similarly, for facial landmark datasets in [66], the main differences in individual landmarks result from the differences between individuals, illumination as well as the position of the face in the image. Besides the differences, the training data provided for our work more closely resembles the input used in the facial keypoint detection problems addressed in [66] which is a combination of images and annotated landmark coordinates. Finally, the results that Longre et al. [66] obtained with such a small dataset are promising and a good place to start.

We constructed a simple ANN model with three layers: an input layer, a 500-node hidden layer and an output layer. This baseline model gave us an RMSE of 282.62 and MAE of 181.33. A MAE analysis at a variable level revealed that the model was the worst at predicting the (x_6, y_6) coordinate pair with MAE values of 574.71 for x_6 and 89.92 for y_6. The best mean absolute error values for this model were for the coordinate pair (x_5, y_5) with values of 92.41 for x_5 and 49.38 for y_5.

Next we trained various LeNet architectures, which we called LeNet3, LeNet4 and LeNet5. These models where trained with and without regularization such as dropout and batch normalization. The LeNet style model that performed the best for the data was LeNet3 with dropout rate of 0.1 which had an RMSE of 53.58 and MAE of 41.05. However, the model overfit the data. The MAE analysis for this model revealed that it was the worst at predicting the coordinate pair (x_6, y_6) with MAE values of 44.93 for x_6 and 50.19 for y_6. The coordinate pair (x_{10}, y_{10}) showed the best MAE values of 37.98 for x_{10} and 35.26 for y_{10}.

To reduce overfitting, we used data augmentation techniques that artificially increase the training data. To achieve this, we translated the training images 10 pixels up, 10 pixels down, 10 pixels to the left and 10 pixels to the right creating a training dataset that was twice the size of the original dataset made up of both the original images and the translated images. We trained the LeNet3 architecture with 0.1 dropout on this new dataset and obtained an RMSE of 54.23 and MAE of 41.69 respectively. This was a worse model than the LeNet3 model that was trained on the original dataset. An MAE analysis revealed that the model was the worst at

predicting variables x_1 and y_6 with MAE values of 46.53 and 50.27 respectively. The coordinate pair (x_8, y_8) and the variable y_5 showed the best MAE values of 38.53, 36.38 and 35.52 for x_8, y_8 and y_5 respectively.

For our second attempt at reducing overfitting, we used the LeNet3 model architecture increasing the dropout rate from 0.1 to 0.5. This model did not overfit and had a RMSE of 65.46 and MAE of 52.71, making this a worse model still. The MAE analysis revealed that the coordinate pair (x_1, y_1) displayed the worst MAE values of 61.60 and 54.12 for x_1 and y_1 respectively. The best MAE values were for the coordinate pair (x_8, y_8) and the variable y_{10} with values of 41.00, 45.64 and 44.16 for x_8, y_8 and y_{10} respectively.

While a model's ability to generalize is important in the real world, we conclude that based on the RMSE and MAE values the LeNet3 model that used the original training examples with a dropout of 0.1 is the best LeNet model.

We went on to train three FCN8 models and a U-Net model. To train these models we transformed our true response variables from coordinate points into heat maps using the Gaussian function. A post-processing step to convert the predicted heat maps back to coordinate points was also added. The best FCN model we trained was the FCN8 model that had a batch size of 32 and used the Adam optimizer. The RMSE and MAE for this model were 1.12 and 0.88 respectively. A MAE analysis revealed that the coordinate pair (x_4, y_4) displayed the worst MAE values of 2.41 and 4.70 for x_4 and y_4 respectively. The best MAE values were for the coordinate pair (x_8, y_8) and the variable y_{10} with values of 0.82, 0.65 and 0.78 for x_8, y_8 and y_{10} respectively.

The FCN models required significantly more computational resources and training time than was used to train the LeNet models. The LeNet models took on average just over 7 minutes to train on a GPU enabled Colab Notebook while the FCN model took 72 minutes to train on two Nvidia Tesla K80 GPUs.

CNNs were chosen for this research because of how widely they have been used to obtain state-of-the-art results in landmark detection tasks. The flexibility of FCNs on input image size is attractive for real world application and given their state-of-the-art performance on segmentation tasks we wanted to understand how well these models would perform if they were adapted for landmark detection.

Basing our model performance on RMSE and MAE leads us to conclude that the FCN model performance for this dataset is significantly better than the performance of any of the CNN models. All FCN models had RMSE and MAE that were better than any CNN model that we trained. We can therefore conclude that for a dataset like ours, where training resources are available, the dataset is limited and time is not a major constraint, training a landmark detection model using an FCN is the recommended approach.

Overall these results indicate that machine learning models can be used to automatically and accurately detect landmark points on tsetse fly wing images. The ability to automatically annotate landmarks on tsetse fly images will help us avail a large database of images and their landmarks, that can be studied for research into tsetse fly control measures at a fraction of the time it would take researchers to annotate the images manually.

5.2 Limitations and future work

The number of training examples that were used in this research was a limitation for our CNN model training. CNNs require a lot more training data to train for accurate results and as

such one might want to investigate whether adding more training examples might give us a better LeNet style CNN model. Given that resource and time requirements to train LeNet style CNN models are generally lower than that required to train FCNs, this would be a worthwhile approach to investigate for users that have hardware and time constraints.

From an entomology perspective, there might be particular interest in those landmarks that are currently used to estimate fly size, as shown in Figure 1. The models struggled to predict landmarks A and B which are associated with the coordinate pairs (x_6, y_6) and (x_1, y_1) respectively, but were much better at predicting the landmarks C and D which are associated with coordinate pairs (x_{10}, y_{10}) and (x_7, y_7) respectively. We could explore the idea of building models to only detect the two coordinate pairs (x_7,y_7) and (x_{10}, y_{10}), given our limited dataset. Making the task simpler might help improve model results and focusing on these coordinate pairs that we have gotten favourable results from might be a good approach.

Image pre-processing techniques for the CNN models can be explored, such as changing the contrast of the images to further highlight the intersections and landmarks on the images. Furthermore, other CNN model architectures such as the common VGG architecture which was explored in [66] and ensemble models can be looked at. We can also try a transfer learning approach by using weights pre-trained on much larger datasets, and retraining the last layers of the network to improve CNN model results.

The models we have built still require researchers to trap flies, remove wings and take individual photographs of each captured fly. These images are located at roughly the same place in the image and are for the most part images that are flat on the image surface. A next phase in this research could be building models that can locate fly wing landmarks in the wild. These could be models that can annotate images even when their positioning might be distorted or occluded in some way. This would mean that researchers would have greater freedom when taking and preparing image datasets.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M. (2016). TensorFlow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (pp. 265-283).

[2] Agarwal, N., Krohn-Grimberghe, A. and Vyas, R. (2017). Facial key points detection using deep convolutional neural network-NaimishNet. arXiv preprint arXiv:1710.00977.

[3] Aharkava, L. (2010). Artificial neural networks and self-organization for knowledge extraction. Master's thesis, Charles University in Prague.

[4] Andriluka, M., Pishchulin, L., Gehler, P. and Schiele, B. (2014). 2D human pose estimation: New benchmark and state of the art analysis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3686-3693).

[5] Bak, S. and Carr, P. (2017). One-shot metric learning for person re-identification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2990-2999).

[6] Baker, S., Gross, R., Matthews, I. and Ishikawa, T. (2003). Lucas-Kanade 20 years on: A unifying framework: Part 3. Technical Report CMU-RI-TR-03-35, Carnegie Mellon University Robotics Institute.

[7] Bataineh, M.H. (2012). Artificial neural network for studying human performance. Master's thesis, The University of Iowa.

[8] Bearman, A. and Dong, C. (2015). Human pose estimation and activity classification using convolutional neural networks. CS231n Course Project Reports, Stanford University.

[9] Bekkerman, R., Bilenko, M. and Langford, J. eds. (2011). Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press.

[10] Bengio, Y., Simard, P. and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2) (pp. 157-166).

[11] Berg, T. and Belhumeur, P.N. (2013). Poof: Part-based one-vs.-one features for fine-grained categorization, face verification, and attribute estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 955-962).

[12] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(2) (pp. 281-305).

[13] Bisong, E. (2019). An overview of Google Cloud Platform services. In Building Machine Learning and Deep Learning Models on Google Cloud Platform (pp. 7-10). Apress, Berkeley, CA.

[14] Bisong, E. (2019). Google Colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform (pp. 59-64). Apress, Berkeley, CA.

[15] The Editors of Encyclopaedia Britannica. (2017). Tsetse fly. Encyclopedia Britannica. [online] Available at: https://www.britannica.com/animal/tsetse-fly. Accessed: 13/12/2020.

[16] Bruslund Haurum, J., Karpova, A., Pedersen, M., Hein Bengtson, S. and Moeslund, T.B. (2020). Re-identification of zebrafish using metric learning. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision Workshops (pp. 1-11).

[17] Brust, C.A., Burghardt, T., Groenenberg, M., Kading, C., Kuhl, H.S., Manguette, M.L. and Denzler, J. (2017). Towards automated visual monitoring of individual gorillas in the wild. In Proceedings of the IEEE International Conference on Computer Vision Workshops (pp. 2820-2830).

[18] Cao, X., Wei, Y., Wen, F. and Sun, J. (2014). Face alignment by explicit shape regression. International Journal of Computer Vision, 107(2) (pp. 177-190).

[19] Chai, Y., Lempitsky, V. and Zisserman, A. (2013). Symbiotic segmentation and part localization for fine-grained categorization. In Proceedings of the IEEE International Conference on Computer Vision (pp. 321-328).

[20] Chollet, F. (2015). Keras. [online] Available at: https://github.com/fchollet/keras. Accessed: 12/01/2021.

[21] Collet, S. (2017). Object detection part 1. [online] Available at: https://www.saagie.com/blog/objectdetection-part1. Accessed: 12/09/2020.

[22] Cootes, T.F., Edwards, G.J. and Taylor, C.J. (2001). Active appearance models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(6) (pp. 681-685).

[23] Dantone, M., Gall, J., Fanelli, G. and Van Gool, L. (2012). Real-time facial feature detection using conditional regression forests. In IEEE Conference on Computer Vision and Pattern Recognition (pp. 2578-2585).

[24] Dardas, N., Chen, Q., Georganas, N.D. and Petriu, E.M. (2010). Hand gesture recognition using bag-of-features and multi-class support vector machine. In IEEE International Symposium on Haptic Audio Visual Environments and Games (pp. 1-5).

[25] Datcu, D. and Lukosch, S. (2013). Free-hands interaction in augmented reality. In Proceedings of the 1st Symposium on Spatial User Interaction (pp. 33-40).

[26] Dlamini, G. (2018). Machine learning methods for individual acoustic recognition in a species of field cricket (Master's thesis, University of Cape Town).

[27] Donner, R., Reiter, M., Langs, G., Peloschek, P. and Bischof, H. (2006). Fast active appearance model search using canonical correlation analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(10) (pp. 1690-1694).

[28] Duchi, J., Hazan, E. and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(7) (pp. 2121-2159).

[29] Edwards, G.J., Taylor, C.J. and Cootes, T.F. (1998). Interpreting face images using active appearance models. In Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition (pp. 300-305).

[30] Eichner, M., Marin-Jimenez, M., Zisserman, A. and Ferrari, V. (2012). 2D articulated human pose estimation and retrieval in (almost) unconstrained still images. International Journal of Computer Vision, 99(2) (pp. 190-214).

[31] Fan, X., Zheng, K., Lin, Y. and Wang, S. (2015). Combining local appearance and holistic view: Dual-source deep neural networks for human pose estimation. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 1347-1355).

[32] Freytag, A., Rodner, E., Simon, M., Loos, A., Kuhl, H.S. and Denzler, J. (2016). Chimpanzee faces in the wild: Log-Euclidean CNNs for predicting identities and attributes of primates. In German Conference on Pattern Recognition (pp. 51-63).

[33] Gareth, J., Daniela, W., Trevor, H. and Robert, T. (2013). An introduction to statistical learning: with applications in R. Springer.

[34] Ge, Z., McCool, C., Sanderson, C. and Corke, P. (2015). Subset feature learning for fine-grained category classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 46-52).

[35] Ge, Z., Bewley, A., McCool, C., Corke, P., Upcroft, B. and Sanderson, C. (2016). Fine-grained classification via mixture of deep convolutional neural networks. In Winter Conference on Applications of Computer Vision (pp. 1-6).

[36] Gencay, R. and Qi, M. (2001). Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. IEEE Transactions on Neural Networks, 12(4) (pp. 726-734).

[37] Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT Press.

[38] Gouidis, F., Panteleris, P., Oikonomidis, I. and Argyros, A. (2019). Accurate hand keypoint localization on mobile devices. In International Conference on Machine Vision Applications (pp. 1-6).

[39] Hansen, M.C., Potapov, P.V., Moore, R., Hancher, M., Turubanova, S.A., Tyukavina, A., Thau, D., Stehman, S.V., Goetz, S.J., Loveland, T.R. and Kommareddy, A. (2013). High-resolution global maps of 21st-century forest cover change. Science, 342(6160) (pp. 850-853).

[40] Hargrove, J.W. (1999). Reproductive abnormalities in tsetse flies in Zimbabwe. Entomologia Experimentalis et Applicata, 92(1) (pp. 89-99).

[41] Hargrove, J., English, S., Torr, S.J., Lord, J., Haines, L.R., Van Schalkwyk, C., Patterson, J. and Vale, G. (2019). Wing length and host location in tsetse (Glossina spp.): implications for control using stationary baits. Parasites & Vectors, 12(1) (pp. 1-13).

[42] He, K., Zhang, X., Ren, S. and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778).

[43] Hou, X., Li, S.Z., Zhang, H. and Cheng, Q. (2001). Direct appearance models. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 828-833).

[44] Hu, C., Feris, R. and Turk, M. (2003). Real-time view-based face alignment using active wavelet networks. In Proceedings of the IEEE International SOI Conference (pp. 215-221).

[45] Hu, K., Canavan, S. and Yin, L. (2010). Hand pointing estimation for human computer interaction based on two orthogonal-views. In Proceedings of the International Conference on Pattern Recognition (pp. 3760-3763).

[46] Ioffe, S. and Szegedy, C. (2015), June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (pp. 448-456).

[47] Jiao, F., Li, S., Shum, H.Y. and Schuurmans, D. (2003). Face alignment using statistical models and wavelet features. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. I-I).

[48] Jin, S., Xu, L., Xu, J., Wang, C., Liu, W., Qian, C., Ouyang, W. and Luo, P. (2020). Whole-body human pose estimation in the wild. In European Conference on Computer Vision (pp. 196-214).

[49] Johnson, S. and Everingham, M. (2010). Clustered pose and nonlinear appearance models for human pose estimation. In Proceedings of the British Machine Vision Conference (pp. 12.1-12.11).

[50] Jordan, J. (2018). An overview of semantic image segmentation. [online] Available at: https://www.jeremyjordan.me/semantic-segmentation/. Accessed: 12/11/2020.

[51] Kahou, S.E., Pal, C., Bouthillier, X., Froumenty, P., Gulcehre, C., Memisevic, R., Vincent, P., Courville, A., Bengio, Y., Ferrari, R.C. and Mirza, M. (2013). Combining modality specific deep neural networks for emotion recognition in video. In Proceedings of the ACM on International Conference on Multimodal Interaction (pp. 543-550).

[52] Karpathy, A. (2016a). Cs231n convolutional neural networks for visual recognition course notes. [online] Available at: https://cs231n.github.io/convolutional-networks/. Accessed: 05/07/2019.

[53] Karpathy, A. (2016b). Cs231n convolutional neural networks for visual recognition course notes. [online] Available at: http://cs231n.github.io/neural-networks-2/. Accessed: 02/08/2018.

[54] Katanforoosh, K., and Kunin, D. (2019). Initializing neural networks. [online] Available at: https://www.deeplearning.ai/ai-notes/initialization/. Accessed: 19/11/2019.

[55] Kazemi, V. and Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 1867-1874).

[56] Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[57] Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25 (pp. 1097-1105).

[58] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. and Jackel, L.D. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4) (pp. 541-551).

[59] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11) (pp. 2278-2324).

[60] LeCun, Y.A., Bottou, L., Orr, G.B. and Muller, K.R. (2012). Efficient backprop. In Neural Networks: Tricks of the Trade (pp. 9-48).

[61] LeCun, Y., Cortes, C., Burges, C.J.C., The MNIST database of handwritten digits. [online] Available at: http://yann.lecun.com/exdb/mnist/. Accessed: 15/10/2020.

[62] Liu, J., Kanazawa, A., Jacobs, D. and Belhumeur, P. (2012). Dog breed classification using part localization. In European Conference on Computer Vision (pp. 172-185).

[63] Liu, J., Zha, Z.J., Chen, D., Hong, R. and Wang, M. (2019). Adaptive transfer network for cross-domain person re-identification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 7202-7211).

[64] Liu, X., Xia, T., Wang, J. and Lin, Y. (2016). Fully convolutional attention localization networks. arXiv preprint arXiv: 1603.06765.

[65] Long, J., Shelhamer, E. and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3431-3440).

[66] Longpre, S. and Sohmshetty, A. (2016). Facial keypoint detection. CS231n Course Project Reports, Stanford University.

[67] Matthews, I. and Baker, S. (2004). Active appearance models revisited. International Journal of Computer Vision, 60(2) (pp. 135-164).

[68] Mbewe, N.J., Saini, R.K., Torto, B., Irungu, J., Yusuf, A.A. and Pirk, C. (2018). Effects of vector control on the population structure of tsetse (Glossina fuscipes fuscipes) in western Kenya. Acta Tropica, 179 (pp. 1-9).

[69] Mitchell, T.M. (1997). Artificial neural networks. Machine Learning, 45 (pp. 81-127).

[70] Mora, K.A.F. and Odobez, J.M. (2012). Gaze estimation from multimodal Kinect data. In IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 25-30).

[71] Moskvyak, O., Maire, F., Dayoub, F. and Baktashmotlagh, M. (2020). Learning landmark guided embeddings for animal re-identification. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision Workshops (pp. 12-19).

[72] Muzari, M.O. and Hargrove, J.W. (1996). The design of target barriers for tsetse flies, Glossina spp.(Diptera: Glossinidae). Bulletin of Entomological Research, 86(5) (pp. 579-583).

[73] Newell, A., Yang, K. and Deng, J. (2016). Stacked hourglass networks for human pose estimation. In European Conference on Computer Vision (pp. 483-499).

[74] Ng, A.Y. (2004). Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In Proceedings of the International Conference on Machine Learning (p. 78).

[75] Nie, X., Feng, J., Xing, J. and Yan, S. (2017). Generative partition networks for multi-person pose estimation. arXiv preprint arXiv:1705.07422.

[76] Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378.

[77] Oktay, O., Bai, W., Guerrero, R., Rajchl, M., de Marvao, A., O'Regan, D.P., Cook, S.A., Heinrich, M.P., Glocker, B. and Rueckert, D. (2016). Stratified decision forests for accurate anatomical landmark localization in cardiac images. IEEE Transactions on Medical Imaging, 36(1) (pp. 332-342).

[78] Orhan, A.E. and Pitkow, X. (2017). Skip connections eliminate singularities. arXiv preprint arXiv:1701.09175.

[79] Papandreou, G., Zhu, T., Kanazawa, N., Toshev, A., Tompson, J., Bregler, C. and Murphy, K. (2017). Towards accurate multi-person pose estimation in the wild. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4903-4911).

[80] Papandreou, G., Zhu, T., Chen, L.C., Gidaris, S., Tompson, J. and Murphy, K. (2018). Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. In Proceedings of the European Conference on Computer Vision (pp. 269-286).

[81] Phung, V.H. and Rhee, E.J. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. Applied Sciences, 9(21) (p. 4500).

[82] Pishchulin, L., Andriluka, M., Gehler, P. and Schiele, B. (2013). Poselet conditioned pictorial structures. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 588-595).

[83] Prechelt, L. (1998). Early stopping-but when? In Neural Networks: Tricks of the Trade (pp. 55-69).

[84] Ren, S., Cao, X., Wei, Y. and Sun, J. (2014). Face alignment at 3000 fps via regressing local binary features. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1685-1692).

[85] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In IEEE International Conference on Neural Networks (pp. 586-591).

[86] Robinson, J.P., Li, Y., Zhang, N., Fu, Y. and Tulyakov, S. (2019). Laplace landmark localization. In Proceedings of the IEEE International Conference on Computer Vision (pp. 10103-10112).

[87] Roos, M. (2019). Deep learning neurons versus biological neurons. [online] Available at: https://towardsdatascience.com/deep-learning-versus-biological-neurons-floating-point-numbers-spikes-and-neurotransmitters-6eebfa3390e9. Accessed: 15/04/2019.

[88] Ronneberger, O., Fischer, P. and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In International Conference on Medical Image Computing and Computer-assisted Intervention (pp. 234-241).

[89] Ruder, S. (2019). An overview of gradient descent optimisation algorithms. [online] Available at: https://arxiv.org/abs/1609.04747. Accessed: 05/08/2020.

[90] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning representations by back-propagating errors. Nature, 323(6088) (pp. 533-536).

[91] Sapp, B. and Taskar, B. (2013). MODEC: Multimodal decomposable models for human pose estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3674-3681).

[92] Saragih, J. and Goecke, R. (2007). A nonlinear discriminative approach to AAM fitting. In IEEE International Conference on Computer Vision (pp. 1-8).

[93] Saunders, D.S. (1960). The ovulation cycle in Glossina morsitans Westwood (Diptera: Muscidae) and a possible method of age determination for female tsetse flies by the examination of their ovaries. Transactions of the Royal Entomological Society of London, 112(9) (pp. 221-238).

[94] Saunders, D.S. (1962). Age determination for female tsetse flies and the age compositions of samples of Glossina pallidipes Aust., G. palpalis fuscipes Newst. and G. brevipalpis Newst. Bulletin of Entomological Research, 53(3) (pp. 579-595).

[95] Scherer, D., Muller, A. and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In International Conference on Artificial Neural Networks (pp. 92-101).

[96] Sermanet, P., Frome, A. and Real, E. (2014). Attention for fine-grained categorization. arXiv preprint arXiv:1412.7054.

[97] Simon, H. (1999). Neural networks: a comprehensive foundation. Prentice Hall.

[98] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[99] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1) (pp. 1929-1958).

[100] Su, Z., Ye, M., Zhang, G., Dai, L. and Sheng, J. (2019). Cascade feature aggregation for human pose estimation. arXiv preprint arXiv:1902.07837.

[101] Sun, Y., Wang, X. and Tang, X. (2013). Deep convolutional network cascade for facial point detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3476-3483).

[102] Sun, Y., Wang, X. and Tang, X. (2014). Deep learning face representation by joint identification-verification. arXiv preprint arXiv:1406.4773.

[103] Sun, Y., Liang, D., Wang, X. and Tang, X. (2015). DeepID3: Face recognition with very deep neural networks. arXiv preprint arXiv:1502.00873.

[104] Sun, K., Xiao, B., Liu, D. and Wang, J. (2019). Deep high-resolution representation learning for human pose estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5693-5703).

[105] Stenroos, O. (2017). Object detection from images using convolutional neural networks (Master's thesis, Aalto University).

[106] Taigman, Y., Yang, M., Ranzato, M. and Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (p. 6).

[107] University of Glasgow. Animal African Trypanosomiasis. [online] Available at: https://www.gla.ac.uk/research/beacons/onehealth/animalafricantrypanosomiasis/#:~:text=Animal%20African%20Trypanosomiasis%20(AAT)%2C,countries%20in%20sub%2DSaharan%20Africa. Accessed: 18/12/2020.

[108] Vale, G.A. (1974). New field methods for studying the responses of tsetse flies (Diptera, Glossinidae) to hosts. Bulletin of Entomological Research, 64(2) (pp.199-208).

[109] Wang, D., Shen, Z., Shao, J., Zhang, W., Xue, X. and Zhang, Z. (2015). Multiple granularity descriptors for fine-grained categorization. In Proceedings of the IEEE international Conference on Computer Vision (pp. 2399-2406).

[110] Wang, F. and Li, Y. (2013). Beyond physical connections: Tree models in human pose estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 596-603).

[111] Wang, W., Tulyakov, S. and Sebe, N. (2016). Recurrent convolutional face alignment. In Asian Conference on Computer Vision (pp. 104-120).

[112] Wang, Z., Wang, X. and Wang, G. (2018). Learning fine-grained features via a CNN tree for large-scale classification. Neurocomputing, 275 (pp. 1231-1240).

[113] Wei, S.E., Ramakrishna, V., Kanade, T. and Sheikh, Y. (2016). Convolutional pose machines. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4724-4732).

[114] Wiehman, S. (2018). Investigating fully convolutional networks for bio-image segmentation (Master's thesis, Stellenbosch University).

[115] Wu, Y. and Ji, Q. (2019). Facial landmark detection: A literature survey. International Journal of Computer Vision, 127(2) (pp. 115-142).

[116] Xiao, T., Xu, Y., Yang, K., Zhang, J., Peng, Y. and Zhang, Z. (2015). The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 842-850).

[117] Xiong, X. and De la Torre, F. (2013). Supervised descent method and its applications to face alignment. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 532-539).

[118] Yang, H. and Patras, I. (2013). Privileged information-based conditional regression forest for facial feature detection. In IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (pp. 1-6).

[119] Yang, Y. and Ramanan, D. (2012). Articulated human detection with flexible mixtures of parts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(12) (pp. 2878-2890).

[120] Yang, S., Bo, L., Wang, J. and Shapiro, L. (2012). Unsupervised template learning for fine-grained object recognition. Advances in Neural Information Processing Systems, 25 (pp. 3122-3130).

[121] Yang, Q., Yu, H.X., Wu, A. and Zheng, W.S. (2019). Patch-based discriminative feature learning for unsupervised person re-identification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3633-3642).

[122] Yu, X., Zhou, F. and Chandraker, M. (2016). Deep deformation network for object landmark localization. In European Conference on Computer Vision (pp. 52-70).

[123] Zhang, G., Patuwo, B.E. and Hu, M.Y. (1998). Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting, 14(1) (pp. 35-62).

[124] Zhang, J. (2019). Basic neural units of the brain: neurons, synapses and action potential. arXiv preprint arXiv:1906.01703.

[125] Zhang, N., Donahue, J., Girshick, R. and Darrell, T. (2014). Part-based R-CNNs for fine-grained category detection. In European Conference on Computer Vision (pp. 834-849).

[126] Zhang, X., Sugano, Y., Fritz, M. and Bulling, A. (2015). Appearance-based gaze estimation in the wild. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4511-4520).

[127] Zhang, Z., Luo, P., Loy, C.C. and Tang, X. (2014). Facial landmark detection by deep multi-task learning. In European Conference on Computer Vision (pp. 94-108).

[128] Zhao, B., Feng, J., Wu, X. and Yan, S. (2017). A survey on deep learning-based fine-grained object classification and semantic segmentation. International Journal of Automation and Computing, 14(2) (pp. 119-135).

[129] Zhao, B., Wu, X., Feng, J., Peng, Q. and Yan, S. (2017). Diversified visual attention networks for fine-grained object classification. IEEE Transactions on Multimedia, 19(6) (pp. 1245-1256).

[130] Zhong, Z., Zheng, L., Luo, Z., Li, S. and Yang, Y. (2019). Invariance matters: Exemplar memory for domain adaptive person re-identification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 598-607).