UNIVERSITY OF THE WESTERN CAPE

South African Sign Language Hand Shape and Orientation Recognition on Mobile Devices Using Deep Learning

by
Kurt Jacobs

A thesis submitted in fulfillment for the
degree of Master of Science

in the
Faculty of Science
Department of Computer Science

March 2017

Declaration of Authorship

I, Kurt Jacobs, declare that this thesis "South African Sign Language Hand Shape and Orientation Recognition on Mobile Devices Using Deep Learning" is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signed:

Date: 5 December 2016

"Do the difficult things while they are easy and do the great things while they are small. A journey of a thousand miles must begin with a single step."

- Lao Tzu

UNIVERSITY OF THE WESTERN CAPE

Abstract

Faculty of Science
Department of Computer Science

Master of Science

by Kurt Jacobs

Supervisors: Mehrdad Ghaziasgar, Isabella Venter, Reginald Dodds

In order to classify South African Sign Language as a signed gesture, five fundamental parameters need to be considered. These five parameters to be considered are: hand shape, hand orientation, hand motion, hand location and facial expressions.

The research in this thesis will utilise Deep Learning techniques, specifically Convolutional Neural Networks, to recognise hand shapes in various hand orientations. The research will focus on two of the five fundamental parameters, i.e., recognising six South African Sign Language hand shapes for each of five different hand orientations. These hand shape and orientation combinations will be recognised by means of a video stream captured on a mobile device. The efficacy of Convolutional Neural Network for gesture recognition will be judged with respect to its classification accuracy and classification speed in both a desktop and embedded context.

The research methodology employed to carry out the research was Design Science Research. Design Science Research refers to a set of analytical techniques and perspectives for performing research in the field of Information Systems and Computer Science. Design Science Research necessitates the design of an artefact and the analysis thereof in order to better understand its behaviour in the context of Information Systems or Computer Science.

Acknowledgements

I would like to thank Mr. Mehrdad Ghaziasgar, Prof. Isabella Venter and Mr. Reginald Dodds for the support, guidance, and understanding for the duration of the research. Many thanks to the National Research Foundation (NRF) for the scholarship awarded for the duration of the research study - the funds were used to acquire tools and equipment required to complete this research. I would also like to extend my thanks to the Torch7 community for all their support.

Contents

List of Figures

List of Tables

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
BGR	**B**lue **G**reen **R**ed
CNN	**C**onvolutional **N**eural **N**etwork
DDR	**D**ouble **D**ata **R**ate
DSR	**D**esign **S**cience **R**esearch
ES	**E**mmbeded **S**ystems
GMM	**G**aussian **M**ixture **M**odels
HGR	**H**and **G**esture **R**ecognition
HSO	**H**and **S**hape **O**reintation
HSV	**H**ue **S**aturation **V**alue
HTS	**H**and **T**racking and **S**Segmentation
iOS	**i** **O**perating **S**ystem
kNN	**k** **N**earest **N**eighbours
MPCNN	**M**ax **P**ooling **C**onvolutional **N**eural **N**etwork
NBG	**N**o **B**ackground
BG	**B**ackground
NRS	**N**on-**R**esource **S**tarved
OCR	**O**ptical **C**haracter **R**ecogntion
OR	**O**rientation
ROM	**R**ange **O**f **M**otion
SASL	**S**outh **A**frican **S**ign **L**anguage
UWC	**U**niversity of the **W**estern **C**ape
VJF	**V**iola **J**ones **F**ramework
VGA	**V**isual **G**raphics **A**rray

Chapter 1

Introduction

1.1 Background and Motivation

Communication in all its forms is a process that we often take for granted. We make use of communication to convey information to others through an array of media. It is through these media that we are able to express ideas, emotions and share experiences and knowledge we have acquired. We also communicate to utilise services provided to us and seek help when necessary. The ability to communicate is important for the wellbeing of human beings.

According to a 2004 fact sheet [4], (a total of) approximately 2 million people suffer with hearing disabilities in South Africa. A large number, between 500 000 and 600 000, were classified as deaf and 1 500 000 were considered as hearing impaired. The population in 2004 was 47 million. This means that 4,5% of the total South African population suffered from hearing disabilities. Even though a large number of people are hard of hearing only a few speak native South African Sign Language (SASL). The estimates vary between 700 000 and 2 million.

Deaf, with a capital D, refer to people who lost the ability to hear at an early age, who belong to the Deaf community and speak SASL [5]. Deaf, with a lowercase d, refers to people who do not use SASL and communicate with the hearing world in a spoken language [6].

There are two common false notions. The first of these is that there exists only one variant of Sign Language used by the Deaf worldwide. There exists many varieties of sign language worldwide and in some cases even two or more variants for a single country. The second is that each sign language is a signed-gestural equivalent of a particular spoken language e.g. German. This is completely false, signed languages are languages

with their own grammar and syntax [7]. This leads to a false belief that the Deaf understand a spoken language and can, at the very least, read and write [8]. Because of this lack of understanding it is immediately apparent that Deaf people struggle to enjoy most services provided to those that are literate and understand spoken languages. This inability to understand spoken languages makes it even more challenging for Deaf people to obtain a job compared with those who are literate and understand spoken languages. This obstacle leads to poverty and unemployment amongst the Deaf community. Sign language interpreters are available for the Deaf community but are costly and scarce and thus it is not feasible for most of the Deaf [9]. SASL interpreters are valued but can be obtrusive when, for example, visiting a doctor. It can thus be surmised that a system that translates SASL into text or voice would be invaluable to the Deaf community.

The SASL Research Group is a subgroup within the Assistive Technologies Research Group at the University of the Western Cape. The main goal of the SASL research group is to create a machine learning translation system that is capable of translating SASL into English and vice versa. The SASL research focuses on the conversion of SASL to English which is a completely separate process to English being converted into SASL.

Research has indicated that any sign language gesture can be characterised by five fundamental parameters, these are: hand shape, hand orientation, hand location, hand motion, and facial expressions [10]. The recognition of a SASL gesture is comprised of each of these parameters being recognised within an image of a video stream.

The use of machine learning techniques to recognise SASL gestures are necessary to convert gestures within the obtained image data, using image processing into sign writing. This sign writing will be converted into English and then finally into audio. Thus, the recognition accuracy of the machine learning techniques is directly correlated with the correctness of the final audio rendering. All of the existing research within the SASL group has been done and constructed to run on desktop computers. The ultimate goal would be to construct the most performant machine learning based system and have it run on an embedded device with a camera, such as a mobile phone. These embedded devices with cameras are becoming increasingly more ubiquitous, are relatively affordable and are much better suited, in terms of convenience, to the problem domain of SASL translation than a desktop computer.

The Assistive Technologies Research Group has created systems that recognise hand shape [11], hand location [12, 13], hand motion [14, 15] and facial expressions [16, 17] of a signer in a video stream. Some notable research in the group relating to hand shape recognition has been done by Li and Foster. Li's system extracts features pertaining to hand shape from a stream of images supplied by means of a web camera [11]. These features are then classified by means of a Support Vector Machine (SVM) classifier. It

has been shown to robustly classify even with variations in skin tone, body dimensions and gender.

Using a SVM classifier Li was able to recognise a set of 10 SASL hand gestures with an accuracy of 83,3%. Building on this knowledge Foster expanded by comparing the SVM classifier to other machine learning techniques, namely, Artificial Neural Network (ANN) and Random Forests (RF). These were chosen because they have shown promise across a wide array of classification problems [18]. Fosters research has shown that SVMs are considerably quicker to train than ANNs and RFs. When compared in terms of accuracy the ANN at 85.9% accuracy outperforms both the SVM and RF, which obtained an accuracy of 81.3%, given a trained and optimised model. The RF was fastest in terms of carrying out a classification on a single image when compared to a SVM and ANN. Both the implementations for the ANN and RF were the ones provided by the OpenCV library.

The proposal for this research aimed to use Deep Learning, specifically Convolutional Neural Networks (CNNs), on mobile devices to enhance the research at the University of the Western Cape's (UWC) Assistive Technologies Research Group. Research of the literature has indicated that hand orientation recognition using monocular cameras has not been attempted before. However, the literature indicates that numerous research efforts regarding hand shape exist. Generally the focus is in a single plane with a monocular view of the front of the hand but never simultaneously with multiple hand orientations. This research investigates deep learning in relation to both the recognition of hand orientation and hand shapes in multiple orientations simultaneously and it does so in the context of mobile and embedded devices.

1.2 Aim of the Research

This research analyses the effectiveness of Deep Learning in the context of gesture classification using two gesture parameters, namely, hand shape and hand orientation. SASL researchers Foster and Li have used Artificial Neural Networks and Support Vector Machines, respectively, to recognise hand shape in a single orientation on desktop computers. This research not only focuses on the recognition of hand shape but also, on the recognition of hand orientation and hand shapes within multiple orientations using a new machine learning technique, deep learning, on mobile devices. Thus, this research focuses on the efficacy of Deep Learning, specifically CNNs, with respect to hand shape recognition in multiple orientations by analysing the computation speed and accuracy in a desktop and in an embedded context.

1.3 Research Question

The research question can be formulated as follows: "How performant is Deep Learning, specifically CNNs, on embedded devices in the context of hand shape recognition in multiple hand orientations?"

This research question can be broken down further into the following sub-questions:

1. Which parameter values for the CNNs provide an optimised network model in the context of hand shape recognition in various orientations with respect to accuracy on embedded devices?

2. How accurate are CNNs at classifying unseen images in the context of hand orientation recognition on embedded devices after an optimised network model has been established?

3. How accurate are CNNs at classifying unseen images in the context of hand shape recognition in various orientations on embedded devices after an optimised network model has been established?

4. How fast are CNNs at classifying unseen images in the context of hand shape recognition in various orientations on embedded devices after an optimised network model has been established?

1.4 Research Objectives

In order to answer these aforementioned questions, the following was completed:

1. A system was created that would segment the hand from within an image. The segmented hand was used as input into a CNN to do training and classification.

2. After training the CNN, the classification accuracy and speed was analysed based on test data. The test data was a set of unseen hand shape images.

3. The trained models were tested on a modern mobile operating system, Apple iOS.

1.5 Delimitation of Study Area

This research is subject to the following set of delimiters that describe the delimitation of the research project:

1. It is assumed that the user of the application will have a tripod on which to mount the mobile device.

2. It is assumed that the user of the system, sitting or standing, remains within the frame of the video capture device when signing SASL. The user will not need to wear any special markers or devices. The skin colour of the user in relation to their background is arbitrary.

3. It is assumed that there will only be one person within the field of vision of the video capture device when signing SASL. This assumption can be justified because a similar problem exists within the context of verbal communication. i.e., when environments are busy it is difficult to clearly hear what another person is communicating.

4. It is assumed that the user will hold up their palm until the hand tracking component of the system is initialised. This is required in order to initially locate the users hand. The signer will then be free to move their hand and sign SASL.

5. The trained models will be tested on Apple iOS.

1.6 Thesis Outline

Chapter 1 describes the research problem and the motivation for why this research is important. The rest of the thesis is outlined below:

Chapter 2 - Related Work

Chapter 2 reviews the existing literature on machine learning and image processing techniques pertaining to hand segmentation and gesture recognition. The chapter reviews the image processing techniques used in the study, they are described in depth as to how they relate to the study in Chapter 3. The rest of the chapter relates to the machine learning techniques required for gesture recognition and uncovers some of the current processes to recognise hand gestures. It shows that multiple approaches exist, namely hardware and vision based systems. These approaches have been shown to be popular and provide relatively robust classification results.

Chapter 3 - Research Methodology & Design

Chapter 3 reviews firstly, Design Science Research, the research methodology, used to do the research and secondly the process followed to answer the questions outlined in Chapter 1. The chapter also describes each of the steps in the process and how they relate to the study in order to work towards answering the research questions. The design of the artefacts is described in this chapter as well as the ethical data collection

process and how each of the testing processes will be performed in order to answer the research questions. This chapter also describes the sample and all of its characteristics.

Chapter 4 - Results & Analysis

Chapter 4 describes the experiments spanning all of the research questions and the pre-processing steps required in order to obtain the final results. After obtaining the results by way of the experiments, an analysis is carried out with the aim of answering each of the research questions outlined in Chapter 1. These research outputs, if adequate, are added to the body of knowledge for Computer Science. The results and any patterns that may have emerged are remarked upon.

Chapter 5 - Conclusion

Chapter 5 is a summary and discussion of the findings of the research and possible directions for future work are presented.

Chapter 2

Related Work

2.1 Introduction

The related work chapter considers existing approaches to the problem domain of hand gesture recognition, as well as CNNs and their application to computer vision problems. This chapter is structured as follows: section 2.2 provides discussion of image processing, section 2.3 discusses machine learning techniques, hand shape recognition, hand gesture classification, section 2.4 summarises the hand shape gesture recognition and section 2.5 reviews the use of CNNs in computer vision.

2.2 Image Processing Techniques

Image processing is the manipulation and/or analysis of still images or image sequences in a digital format in order to obtain meaningful information from the data. This research utilises a number of image processing techniques which are required to segment and track a signer's hand. In this section, the theoretical background of image processing techniques to segment a signer's hand will be discussed.

The techniques that will be covered in this section are as follows: face detection, adaptive skin detection, backprojection, thresholding, background subtraction using Gaussian mixture models, template matching and CAMShift tracking. Additional information about each of the techniques can be found in Appendix A for the interested reader.

2.2.1 Face Detection Using Viola-Jones Framework

The face detection in this research is achieved by making use of the Viola Jones object detection framework in combination with cascades that are specifically designed for detecting faces. The Viola-Jones framework (VJF) provides a set of rectangular coordinates, varying in scale, for each face found inside an image. The VJF is the current best face detection system and is able to detect faces rapidly in real time [19]. It may be suitable for use in an embedded context but requires further investigation. Further information for the interested reader is provided in Appendix A.1

2.2.2 Adaptive Skin Detection

Skin detection is an image processing technique that focuses on segmenting the pixels that represent skin from those that don't, based on a particular model [20]. It assigns some arbitrary predefined colour to the pixels that are outliers to this model (in this instance the colour black was chosen), i.e., non skin pixels, and assigns values along the gray colorspace spectrum to those pixels that resemble skin colour pixels. This intermediary step is necessary to achieve the goal of producing a binary image, i.e., an image that contains black for non skin pixels and white for skin pixels.

An adaptive skin detection method is used in this research. The adaptive nature of the skin detection used in this research pertains to the fact that the model for skin pixels is derived in real time, on a frame by frame basis, by extracting a small patch of skin from a detected face and this patch is used to construct a histogram model [21]. The histogram model construction is based on histogram backprojection as described in the next section. A sample image produced by the adaptive thresholding algorithm is shown in Figure 2.1.

FIGURE 2.1: Adaptive thresholding sample

2.2.3 Histogram Backprojection

Histogram backprojection is an image processing technique that provides a representation of how well an image fits the distribution of pixels for a given histogram model.

Thus, it is apparent that this technique can be used to construct a model of skin pixels and produce an image representing the probability of skin pixels contained in an image. Before the backprojection distribution map is computed, it is important to note that the initial colour frame should first be converted to the HSV colorspace because the sampled histogram is a Hue and Saturation histogram. A colour distribution histogram is obtained by extracting a patch section from, in the face, the image and converting it from the BGR colorspace to HSV or by using a predefined HS histogram. If a predefined HS is not used, a histogram model is constructed using the extracted patch and is computed using the Hue and Saturation channels to represent the skin colour distribution of a signer. The colour distribution histogram produced in the sampling step is used and backprojected onto the HSV version of the initial frame. A skin probability image is obtained that maps non skin pixels as black and candidate skin pixels as values spanning the grey colorspace for a given probability. This probability skin map has fixed threshold applied to it in order to obtain a binary representation of the skin probability map. The HSV colorspace is used because the separation of luminance from the colour component makes it particularly valuable because it is not as volatile to varying degrees of light [22].

A histogram, M, represents the histogram of the target object or colour distribution. The histogram of the image, I, is also computed and subsequently the histogram R is computed which is the ratio of $\frac{M}{I}$. This histogram R is backprojected onto the image to obtain the colour distribution i.e., the images values in the image are substituted for the value in the histogram, R, which they index [23]. This is given by the following formulae:

$$R_j = \frac{M_j}{I_j} \tag{2.1}$$

$$b_{x,y} = min(R_{h(c_{x,y})}, 1) \tag{2.2}$$

where j refers to each histogram bin and x, y refers to each pixel in the backprojected image. A sample of the image produced by the histogram backprojection is shown in Figure 2.2. The users face was purposefully occluded with a black rectangle so as to not disclose their identity. However, it is not occluded when executing in the proposed system.

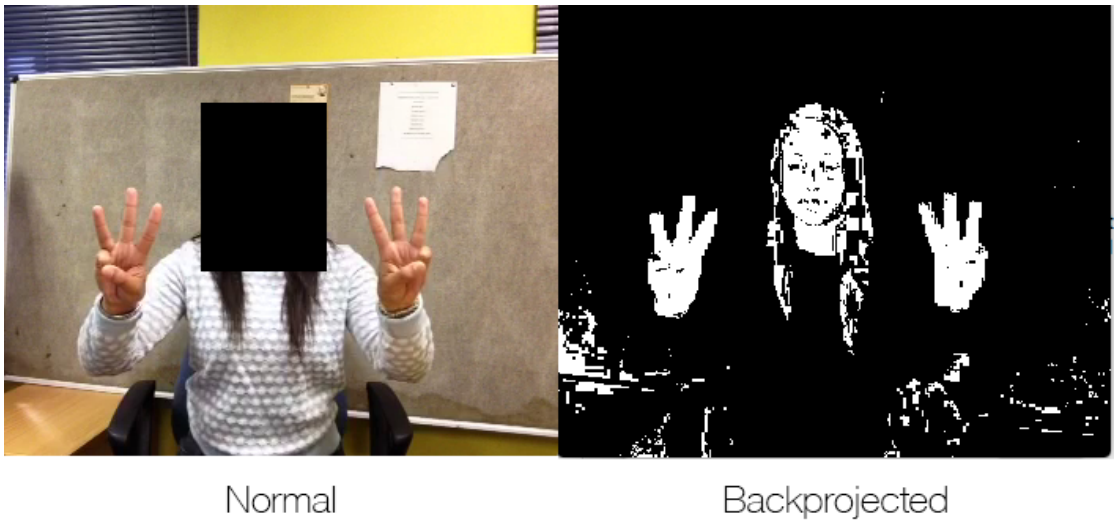


FIGURE 2.2: Histogram backprojection sample

2.2.4 Thresholding

A number of thresholding techniques exist such as simple thresholding, adaptive thresholding and Otsu's binarisation. Thresholding involves converting an image from its current representation into a binary representation for a given value or in the case of adaptive thresholding and Otsu's binarisation this value is dynamic or calculated. Adaptive thresholding and simple/fixed value thresholding are utilised in this research as described below.

Simple thresholding is the use of a static threshold value to convert a grayscale into a binary image. All values higher than this value get assigned a particular value for example, black or 0; and all values lower get assigned another value for example, white or 255. This method is used on the backprojected image because it provides for the ability to extract the skin pixels accurately.

Adaptive thresholding differs from the simple thresholding by calculating a threshold for a smaller regions of the image [24]. Different thresholds are obtained for seperate parts of the image and generally this provides a better result for images containing varying levels of illumination. This method was used on the colour frame, and the resulting thresholded image is used to detect the hand location using template matching. The simplest implementation of adaptive thresholding would be to divide the image into multiple sub-images of size $n \times m$ and computing histograms for each of these sub-images. These respective histograms are then analysed to obtain the threshold value for the related sub-image. Simple thresholding is applied to each of the sub-images using the respective threshold values obtained from the histograms [25].

2.2.5 Background Subtraction Using Gaussian Mixture Models

Background subtraction is an image processing technique that aims to identify and segment regions of interest from the background i.e., differentiating the stationary components of the image from those that are moving [26]. These regions of interest can be defined as objects in the scene that move. The most effective method to obtain the correct segmentation of these regions is to ensure that motion in the scene is exclusively translated as motion in video data. In this research the moving component that is of interest is the signer's hand. Any motion in the face region can be removed by detecting the face and removing it from the frame. More information is provided in Appendix A.2

2.2.6 Template Matching

Template matching is an image processing technique that involves searching for a reference or template image inside of another image, with a certain degree of accuracy [27]. Six widely used methods of template matching exist, namely, Squared Differencing, Cross Correlation, Correlation Coefficient and their normalised equivalents. Template matching, whilst not the most computationally efficient method, provides accurate detection of a template for an initial location window. Once located, the rectangle set of coordinates can be used to initialise the CAMShift algorithm. The CAMShift algorithm is much better suited for realtime and embedded applications in comparison to template matching and is described in the section that follows. The formulae used to perform template matching are provided in Appendix A.3.

2.2.7 CAMShift

FIGURE 2.3: CAMShift sample

Continuously Adaptive Meanshift or CAMShift is an extended version of the meanshift algorithm [28]. CAMShift is a colour based computer vision tracking algorithm and

was developed as a perceptual user interface and consequently, needs to operate in real time and attempts to use the least computational resources. Perceptual user interfaces are interfaces that allow for computers to a digitally mimic the human senses, such as giving the computer the ability to see or touch. The CAMShift algorithm is initialised using the initial location of the hand and then continually tracks the hand in subsequent frames. The need for tracking the hand is clear. It is necessary to track the hand in order to know which segment of the image to pre-process and extract for hand shape and orientation classification. A sample of the tracked hand is shown in Figure 2.3. Additional information about the CAMShift algorithm is provided in Appendix A.4

2.3 Machine Learning Techniques

ANN hand shape recognition classifiers require a process that extracts a set of features for a hand shape. These features are then used as inputs into the classifier which contains a set of predefined classes for classification [18].

The ANN refers to a group of interconnected artificial neurons that are a very basic approximation of neurons in the human brain. The ANN concept is modeled following the neuron model presented by McCullough and Pitts 1943 [29]. ANNs are information processing systems that have certain computational properties analogous to those which have been postulated for biological neural networks [29].

In contrast to ANNs, CNNs only require a hand segmentation component, its architecture contains special layers that are capable of learning filters, which are specific to the problem and which do the feature extraction. These extracted features are then used as input into a fully connected layer at the end of the network, for classification [30].

2.3.1 Hand Shape Recognition Using Machine Learning

When referring to the literature, it is evident that hand shape recognition systems can be divided into two broadly-defined categories, namely, hardware-based systems and vision-based systems. Hardware-based systems are systems that utilise special hardware in order to extract hand shape features, for example, Data Gloves [31], stereoscopic or 3D cameras [32] or colour-coded materials. The use of hardware-based systems reduces the complexity of extracting features because assumptions needn't be made. Using such a system generally results in an enhanced accuracy but at the cost of convenience. In the future, 3D cameras, such as the structure.io and Google's Project Tango device, once they become ubiquitous in mobile devices, could be used [33, 34]. Vision-based systems

are systems that only make use of a simple camera, such as a web camera or mobile phone camera module which are capable of capturing an input video stream. Because of a number of factors, such as motion, background, etc. images captured on such devices require image processing in order to reduce the noise and extract clear features [35]. A vision-based system poses many more challenges than a hardware-based systems. However, with the correct set of algorithms it is possible to construct a high accuracy classifier which will permit maximum user freedom i.e., the system is very minimalist in its hardware requirements [35].

2.3.1.1 Hardware-Based Systems

A number of research efforts using hand shape recognition for hardware based systems are described in [36–38]. A particular study by Fukui et al. proposed a hardware-based wrist watch style wearable that measures wrist contours using photo reflector arrays to recognise hand shapes [1]. The system was developed with home automation in mind but could be applied to a variety of contexts. The device consisted of two main components: the wrist watch type measurement component, a battery and a control component. The measurement component contains photo reflectors (infrared-light distance sensors) in a flexible band. These sensors can measure distances between the band and the surface of the wrist. The band has two arrays each consisting of 75 photo reflectors. The control component is comprised of a battery and a wireless module that enables communication with a PC that has a wireless module. The system works by representing a wrist cross section as a wrist contour. The wrist contour shape varies as a result of finger movements induced by activities of tendons and muscles near the wrist. An initial iteration of the device managed to classify 73.2% of eight possible hand shapes including training data from the user, but only 47.8% accuracy when excluding the users data in the training set [1]. There are small differences in the raw data for each of the hand shapes and thus it is important that the feature extraction process is done with careful consideration. Two features types are prepared, normalised contour data and contour statistics. Because of the difference in muscle and tendon thickness each sensor must have a variation in the ranges of distances they capture. The normalisation process samples the maximum and minimum distances for each sensor element and normalises the distances data to values ranging between 0 and 1, see Figure 2.4.

The contours statistics are made up of data about the wrist contour distances. Examples of these statistics are the sum of the distances, maximum distances, histograms and so forth. Each of the statistics are normalised using calibration data obtained when first wearing the device. The calibration data used the wrist contour data of an open hand and a closed fist. By normalising these statistics, slippage and personal differences

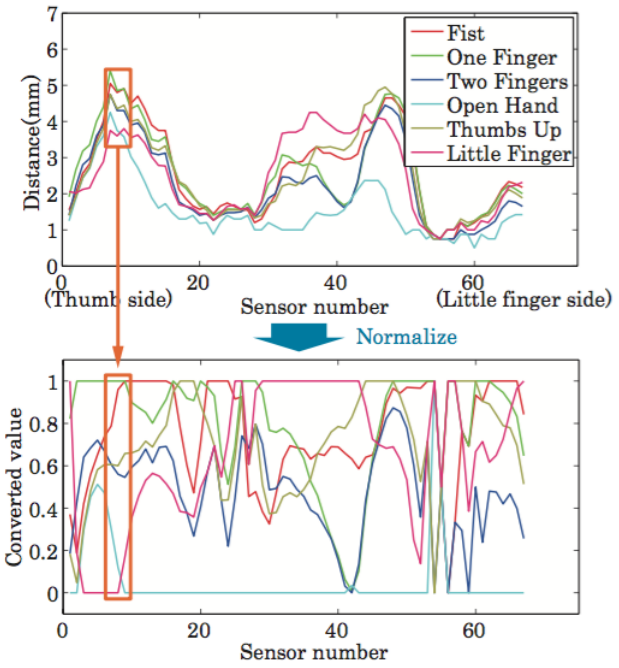


FIGURE 2.4: Contour data from the wrist wearable [1] pp. 5. The data in the orange box is emphasised through the normalisation process.

are avoided. The device uses two different techniques, k-Nearest Neighbours (kNN) or AdaBoost learning, in order to do the classification [1]. In the kNN technique, an input sample is labelled by votes of the k nearest samples. All Euclidean distances between the input sample and the training samples are calculated and used for classification of the input sample. The nearest k samples determine the class to which the input sample belongs [39]. AdaBoost is a meta machine learning technique which improves performance for a number of machine learning techniques. Decision stumps are used as the classification method and enhanced by AdaBoost as presented in [40]. Decision stumps is a machine learning technique that consists of a single layer decision tree [41]. The input samples are classified by weak learners' weighted votes. The weights on weak learners are tweaked to fit the training data during training. Decision stumps are often used as components referred to as weak learners when boosting [42]. A final successful classification of 77.9% is commendable.

2.3.1.2 Vision-based systems

A number of research efforts have been put forward in the domain of hand shape recognition using vision based systems, and these are described in [43–45]. Amongst these research efforts, Nowlan and Platt have proposed a system that tracks hands by means of a CNN architecture [3]. The system locates whether a hand is within a frame and where, the system subsequently determines whether or not the hand is open or closed.

The input to the system is a series of frames captured at a resolution of 320×240 pixels in black and white format. The system was constructed in a user independent manner to accommodate natural clutter and variable lighting conditions in indoor scenes. The system is divided into two separate components, namely, the gesture recognition and the hand tracking components. For the hand tracking component, each frame is first sub-sampled and then subtracted from a previously sub-sampled stored frame to produce a difference frame. These difference frames provide untouched velocity inputs to the system because the largest of the input points tend to occur closer to regions of movement within the frame. Two separate CNNs make predictions on hand location for both the intensity and difference frames. A voting scheme is then employed that combines the predictions of the intensity and difference frames as well as the predictions for the hand trajectory, which is computed from the last three frames. The voting scheme works as follows: First, the hand location is predicted based on the trajectory of the previous three frames. Secondly, the network responses that exceed a given threshold value from both the intensity and difference networks are computed. The location that pertains to the stronger of the two networks or the hand trajectory computation is chosen. The order of importance, greatest to least, for each of the choices is: difference network, intensity network and then hand trajectory. Threshold values are estimated using a training set and a cross validation set. The gesture recognition component accepts a 100×100 image subsection from the current frame as input. The position of this image subsection is drawn around the centre of the point determined by the hand tracking component. The performance of the gesture component is thus directly dependant on the performance of the hand tracking component. A single CNN looks at the intensity of the frame and determines whether or not the hand is in an open or closed state. The 100×100 hand image dimensions were chosen because it is allowed for a positional error rate of up to 25 pixels, whilst still maintaining most of the hand within the image. The largest positional error made by the system was 11 pixels, which falls well within the bounds of the 25 pixel error [3].

Table 2.1 summarises the performance of the tracking component on a set of 342 (18×19 test images per participant) test images. The intensity network alone reports an accuracy of 91.8% within 10 pixels. The difference frame outperforms the intensity frame by 1.8%. The gesture recognition component was able to successfully identify whether the hand was open or closed in 99.1% of the 342 images in the test set. For a set captured in a controlled lab environment, for the CNN system, a success rate of 94% was obtained using a set of 50 images.

TABLE 2.1: Summary of test performance for hand location [3] pp. 7

Information User	Test Error Rate
Intensity	8.2 %
Difference	6.4 %
Intensity + Difference	3.2 %
Intensity + Difference + 3 Frames	0.3 %

2.3.2 Hand Gesture Recognition Using Machine Learning

A similar broader split in categories is also applicable to gesture recognition and these categories are: hardware-based systems and vision-based systems.

2.3.2.1 Hardware-based systems

A number of hardware based hand gesture recognition systems have been described in [46–50]. In addition, McLeod et al. propose a hardware-based gesture recognition system, GRUBC (Gesture Recognition by Utilising Bio-Mechanical Characteristics), that utilises a glove to capture finger joint movements and classify a hand gesture based on the information derived from this data. Range of motion (ROM) is a quantity which defines joint movement by analysing the angle from its start point of motion to its end point [51]. An example of this would be a joint starting point of 20° and end point of 50° which would yield a ROM of 30°. These range of motion values are computed using the sensory values provided by the sensory device. The rationale behind capturing these computed values relative to nonparticipating sections is that they are a user-independent representation of a hand gesture.

For a sensory device with n sensors, each American Sign Language (ASL) sign can be represented with a set of n values. Let's refer to this set $S_i = \{s_1, s_2, s_3, ..., s_n\}$ where i refers to a particular ASL sign and S is the set of sensor values. One problem in gesture recognition is that for each user S_i yields different values. The task is to translate S_i to NR_i where NR_i is unique across different users. Suppose that S_0 and S_i represent the sets of initial and end values of the sensory device,respectively, for a specific ASL sign, then the range of motion tuple R_i would be calculated using $R_i = S_i - S_0$. The set is then iterated to determine both the minimum and maximum values of R_i denoted by $m(R)$ and $M(R)$ respectively. Each value in R_i is then normalised which results in values between 0 and 1 and is then represented by a set NR. Each of the values in the set NR are then discretised with a value of k, where $k > 1$. For example, if $k = 2$ each of the values would be replaced with zeroes if the initial value was below 0.5. NR represents the characteristics movement for a specific hand gesture and provides an abstraction for the gesture. This abstraction can be referred to as the signature of the gesture. This

process ensures that any user can wear the system, and also eliminates any noise that arises from the data collection [51].

In order to recognise a gesture it is necessary for the captured data to follow this same process as described above i.e., a comparison is made between the captured signature and that of a known one in the database. Once the captured data is converted to its NR signature, it goes through a process called registration in which these signatures are saved in a registration database. Each following unknown gesture is then compared to this registration database and labelled with the label of the signature that has the least distance from that of the known gesture according to a specific distance metric e.g. Euclidean distance.

Two approaches were used when testing and training the system, namely, leave-one-out and keep-one-in splits. In the leave-one-out split, 18 of the sets are used in training and the remaining one set is used for testing. This process is done until each set as been tested against 18 other sets each time switching the test set. In the keep-one-in split, one set is used for training and 18 sets are used for testing. The repetition, as in the leave-one-out, was also done which resulted in 19 iterations of training. The reported results are the average of all 19 experiments.

The use of the mechanical gesture recognition approach obtained results of 82.32% and 67.18% for GRUBC on the leave-one-out and keep-one-in approaches respectively. In the leave-one-out approach, the single layer neural network managed to obtain an accuracy of 81.82% and the multilayer neural network obtained 66.27% and 63.16% accuracy for the 9–9–1 and 18–18–1 splits respectively in comparison to GRUBC. It is evident that the results indicate that GRUBC only improves upon the single layer neural network by a small percentage—this shows both methods to be quite effective. In the case of the keep-one-in approach the single layer neural network performed poorly, with an accuracy of 32.24%. The multilayer neural network performed better than the single layer neural network with an accuracy of 54.75% [51].

2.3.2.2 Vision-based systems

A number of gesture recognition research efforts have used vision based systems as described in [52–57]. Amongst these research efforts, Gambardella et al. proposed a Max Pooling Convolutional Neural Network (MPCNN) Architecture that was able to recognise 6 different gesture classes for interaction between humans and a robot swarm. The system used a coloured glove on the signer's hands. This coloured glove allows the system to obtain the hand contour by means of colour segmentation. The contour image is then smoothened by means of image processing techniques in order to remove noisy

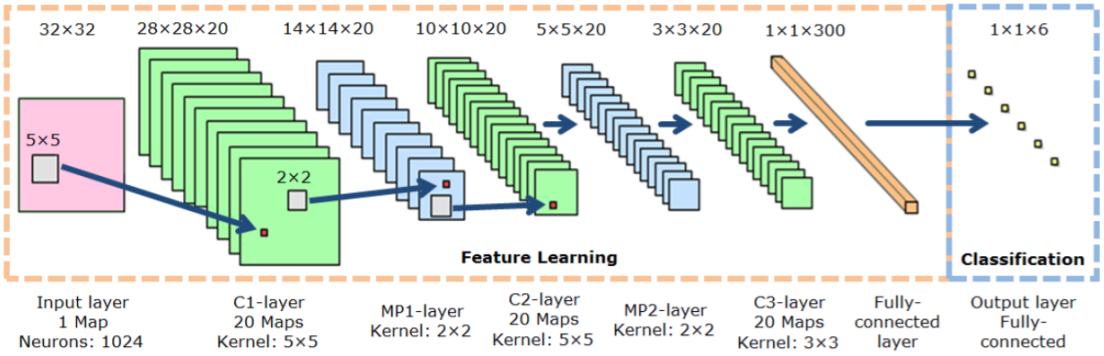


FIGURE 2.5: Max-Pooling Convolutional Neural Network (MPCNN) architecture using alternating convolutional and max-pooling layers [2] pp. 2

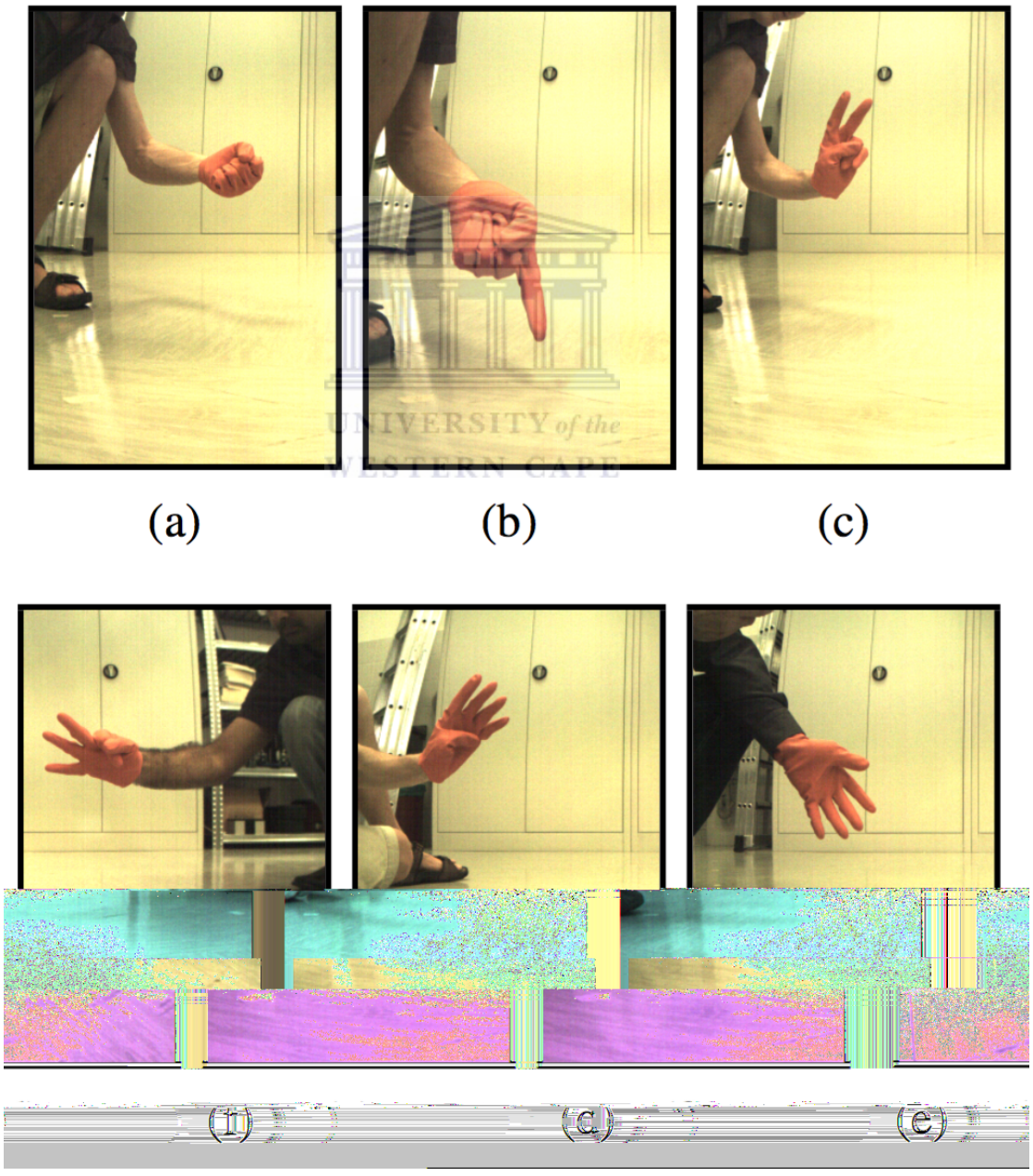


FIGURE 2.6: Gesture classes defined by the count of fingers [2] pp. 3

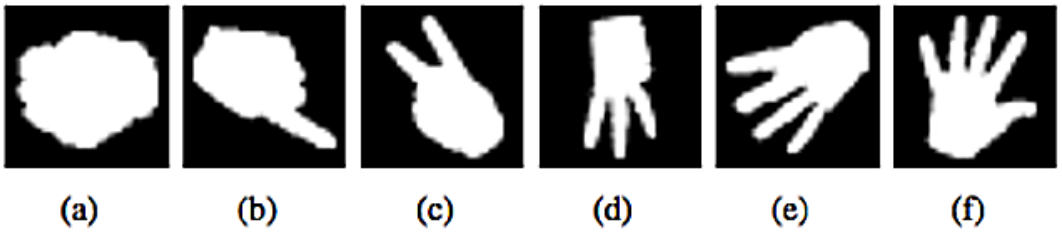


FIGURE 2.7: Images in the training and test sets. Each image represents one of the 6 gesture classes [2] pp. 4

edges. The contour image is then used as input to the MPCNN. The camera is mounted on a robot designed by the Swarmanoid project [58] and is capable of capturing images and video streams at a maximum resolution of 3 megapixels. Figure 2.5 illustrates the architecture of the network. It alternates between convolution layers and max pooling layers and finally ends with the classification layer which has 6 classes. Images are captured by the camera unit on the robot in order to retrieve the hand contour. The gesture vocabulary that the system can recognise is defined by the number of fingers i.e., 0–6 as indicated in Figure 2.6.

The MPCNN requires that all images used as input be equal in size. All input images are 28×28 pixels in size. They are padded with 4 black pixels on all sides, which results in a 32×32 pixel image as shown in Figure 2.7. The MPCNN is trained using online gradient descent and images are rotated in order to learn rotational invariant features [2]. All results are averaged by using a batch size of 100 images for training and test sets. The MPCNN with the lowest validation rate is chosen. The best error obtained was 3.23%, where the training and validation errors are 0.002% and 0.0012% respectively. It took approximately 80 epochs to reach the lowest test error. The system used to train the network for deployment on robots contained a Core i5-650 (3.20 GHz) processor with 4 GB of DDR3 RAM. The time taken per training epoch was 426.12s. The evaluation on the validation and test sets took 189.12s and 48.58s respectively. Once trained the network was deployed on a robot with an ARM 11 533 MHz processor and 128 MB RAM. It took 0.82s to capture, process and classify a single image. This shows that it is possible to implement a real-time MPCNN. The MPCNN implementation was done in C++ as in [59]. The classification rate achieved by the system was 96%.

2.4 Summary of Hand Shape and Gesture Recognition Using Machine Learning

The literature has indicated that two broad categories for hand gesture and hand shape recognition exist, namely, hardware-based and vision-based systems, each of which perform very well. A large number of research studies for both of these categories exist using a variety of machine learning techniques such as kNN, Neural Networks, Support Vector Machines, etc. The research studies highlighted in this chapter indicate that the hardware based systems perform well when one considers the number of classes in the gesture vocabulary. However, it also indicates that CNNs are comparable, obtaining high accuracies for a 6 class vocabulary. Literature has shown that there is, however, a lack of literature pertaining to the recognition of hand shapes in various orientations in a vision-based systems context.

2.5 Convolutional Neural Networks In Computer Vision

CNNs are modelled after a Multi-Stage Heubel-Wiesel Architecture. The idea behind this approach is based on the work on the cat's visual cortex done by Heubel-Wiesel in 1962 [60]. This research identified: orientation-selective simple cells with local receptive fields which plays a similar role to that of the filter bank layers in the CNN; and complex cells which plays a similar role to that of the pooling layers in a CNN.

An innovative use of the CNN was to use the back propagation algorithm to train the entire system in a supervised manner. This approach was a major success in Optical Character Recognition (OCR) and hand writing recognition [30]. Based on this research, a practical solution was developed and deployed at AT&T (Bell Laboratories) in 1993 [30]. The CNN was shown to outperform all other machine learning techniques in its application to the digit recognition domain [61]. A recognition error rate of 0.95% on the initial MNIST database [61] indicates that the CNN is the better technique when compared to the results obtained for ANNs and Support Vector Machines. In addition, this shows that CNNs are robust classifiers that may be practically applied to real-world applications.

A Deep CNN was trained on the ImageNet LSVRC-2010 dataset, which is a subset of the ImageNet database that contains 1000 different classes and 1.2 million images [62]. These images were collected from the Internet and manually labelled by humans. Starting in 2010, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held whereby machine learning techniques are applied to

the database to recognise object classes. For recognition on the ImageNet database it is customary to report two error rates, top-1 and top-5. Top-5 error refers to the fraction of test images for which the correct label is not amongst the five labels considered most probable by the model. Top-1 refers to the fraction of test images that were not correctly predicted by the classifier with highest probable label [62]. In the 2010 LSVRC competition the CNN had a much lower error rate of 37.5% and 17.0% for top-1 and top-5 respectively in comparison to other techniques such as Sparse coding and SIFT+FVs [62]. The sparse coding technique reported results of 47.1% and 28.2% for top-1 and top-5. The SIFT+FV reported results of 45.7% and 25.7% for top-1 and top-5 [62]. It is obvious that CNNs show great promise against other techniques in object recognition and computer vision problems.

2.6 Conclusion

This chapter covered different computer vision and machine learning techniques used in the recognition of hand gestures as well as the application of deep learning in this context. It also briefly covered the use of deep learning in an array of computer vision problems. The literature review indicates that CNNs perform with high successful classification rates across a broad range of computer vision problems and specifically in the domain of gesture and hand shape recognition. It also illustrates that assumptions need to be made, even when making use of a hardware-based system and that these systems do not necessarily offer significant classification improvements when compared to neural network class classifiers. In both instances, CNNs perform with recognition accuracies comparable to the hardware-based system. Some vision-based systems also make assumptions in order to simplify the hand segmentation process and obtain better accuracies. In the research presented by Gambardella et al., a colour glove was used to simplify the extraction of the hand contour. This is not really ideal for use in SASL recognition because it places unnecessary constraints on the user [63].

CNNs are capable of obtaining very high accuracies and low error rates spanning a large number of computer vision classification problems. This indicates that CNNs are an ideal technique for the complex problem of hand orientation and hand shape recognition in various contexts, with the promise of great success. When looking at the success of CNNs in the ILSVRC challenge with a top-5 accuracy of 83% and an error rate of 0.95% in the context of handwritten digit recognition, it is evident that CNNs are suitable for computer vision tasks.

The next chapter describes the research methodology used to answer the questions posed in Chapter 1. It also details the design of the proposed system, the data collection process and the testing process.

Chapter 3

Research Methodology and Design

3.1 Introduction

A research methodology can be described as a protocol or guideline on how to collect, analyse and use information on a particular topic or subject matter. This research utilises a research methodology called Design Science Research (DSR) based on a model of the general process as presented by Takeda et al. [64]. DSR is a set of synthetic and analytical techniques and perspectives for performing research in the fields of Information Systems, Computer Science and Engineering. DSR entails the creation of new knowledge by means of designing novel artefacts (in this case the artefact refers to an entity or process) and the analysis thereof using performance benchmarks, along with reflection and abstraction to contribute to the field of Computer Science. Artefacts may include, but are not limited to, algorithms, human/computer interfaces, systems design methodologies and languages. The remainder of the chapter is structured as follows: Section 3.2 describes DSR, Section 3.3 describes its various phases and Section 3.5 describes the application thereof to the problem of hand orientation and hand shape recognition in various orientations and the design of the proposed system.

3.2 Design Science Research — Overview

This section provides and overview of DSR and the differences between design, research and the amalgamation of design and research in the context of DSR.

3.2.1 Research

DSR is heavily inspired by Kuhn [65] and Lakatos [66] both of whom broadly describe research as an activity that contributes to the understanding of a particular phenomenon. DSR differs slightly from their work in that all or part of the phenomenon that is under scrutiny need not have occurred naturally and instead, may have been created artificially. The phenomenon being considered is typically a process or entity that is of great interest to a single researcher or research community. The research should produce a valid contribution to knowledge, that is usually a theory which describes the behaviour of some aspect of the phenomenon. In order for the contribution to be considered valuable, it should take the form of a publication and should be interesting to research communities [67].

The activities or protocols that a research community deems appropriate towards producing a body of knowledge are called its research methods or techniques. Two different research approaches are used by research communities globally, paradigmatic and pre- or multi-paradigmatic approaches. Research communities that utilise a paradigmatic approach have a universal consensus regarding the phenomenon and the research methods pertaining to its investigation. Research communities utilising a pre- or multi-paradigmatic approach, in contrast, are different from research communities who belong to a single head community. Each of these different sub-research communities are only interested in certain parts of the phenomenon and use research methods which overlap with the head community but may differ from each other in certain regards. Computer Science is a great example of a multi-paradigmatic research community.

3.2.2 Design

Design may be defined as the purpose, planning or intention that exists or is thought to exist before an action, fact or material object comes to fruition [68]. Thus, one may easily deduce that design pertains to the development and creation of new artefacts. Design knowledge may be separated into two categories; routine knowledge and innovative knowledge. If the knowledge required to design the artefact exists, then it is referred to as routine, otherwise it is innovative. Innovative design invokes the need to conduct research that will bridge existing knowledge gaps and lead to a research publication or patent.

3.2.3 Design Science and Design Science Research

Design has been carried out for centuries and spans a wide range of study disciplines such as Architecture, Business, Education and Law, which are all fundamentally concerned with design at their core. This is a key differentiator and characteristic that separates them from the scientific discipline [69]. In the 21st century, natural sciences have dominated design across almost all disciplines except for Computer Science, Chemical Engineering and Management Science.

To intellectualise the act of design, Herbert clearly differentiates between natural science and science of the artificial [69]. Natural science pertains to a body of knowledge about a phenomenon and describes the interaction of the phenomenon with other phenomena in the world. In contrast, science of the artificial pertains to the knowledge about artificial objects or phenomena (artefacts) designed to accomplish a particular goal. Herbert extends this notion of the design of these artificial artefacts to consider the interactions between internal and external environmental factors in order to accomplish the goals set out in their design. The external environment is comprised of a set of external forces and effects acting upon the artefact. The internal environment refers to the components that the artefact is comprised of and the structure between these components. Consequently, the behaviour of the artefact is said to be constrained by both its internal and external environmental factors. The design activity can thus be defined as the development or creation of an artefact, its components and its structure, which interact with its external environment as intended.

In another view of internal and external environmental factors, one may consider the design of artefacts to be a mapping from functional space to attribute space [64]. More concisely, a functional requirement is a point within the functional space and the artefact is the corresponding mapped point within the attribute space. Using this view one can define Design Science as the knowledge that is comprised of constructs, techniques, methods, models and a well-developed theory that facilitates the mapping from functional space to attribute space. Design Science Research creates the missing knowledge by means of design, analysis, reflection and abstraction.

3.3 Design Science Research Process

The Design Science Research process is shown in Figure 3.1 and the phases of which it is comprised of are described in the subsections below, namely: awareness of the problem, suggestion, development, evaluation and conclusion phases.

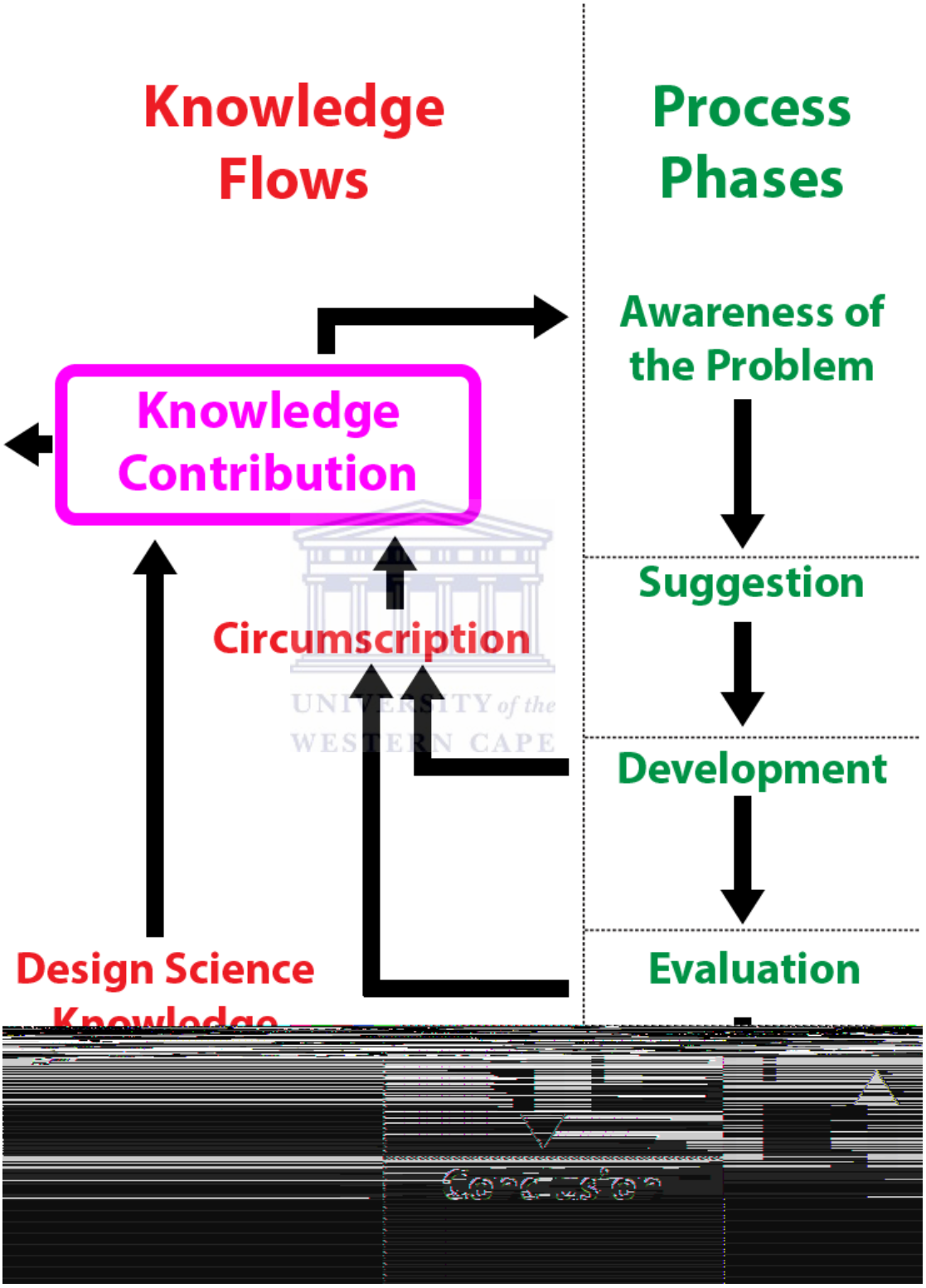


FIGURE 3.1: Design Science Research process model (The DSR Cycle)

3.3.1 Awareness of the Problem Phase

Awareness of the problem may arise from a variety of sources including new developments in industry or research groups.

3.3.2 Suggestion Phase

The suggestion phase results in a tentative design. The performance of a prototype based on the tentative design may also be included. If the tentative design does not present a solution, the research is set aside and deemed unsuitable. This phase is essentially a place of thought for envisioning new functionality based on a novel configuration of existing or new elements.

3.3.3 Development Phase

After the tentative design is accepted the development phase may commence. During the development phase the tentative design is further analysed and subsequently implemented. The techniques used, and the process followed for development, vary, depending on the artefacts. The novelty is primarily in the design and not in the construction of the artefact(s). Artefacts include, but are not limited to, human/computer interfaces, system design methodologies, languages or algorithms e.g. information retrieval. During the development phase it is possible for unforeseen problems to arise, which leads to circumscription, at which point the tentative design is re-assessed and the process is restarted from phase one. It should be noted that despite these potential problems arising, valuable information is still contributed to the knowledge base.

3.3.4 Evaluation Phase

During the evaluation phase, the artefact is tested against certain criteria that are always implicit and made explicit. Deviations from the expected results, both quantitative and qualitative, are noted, analysed and explained. The analysis of these results either confirms or contradicts the hypothesis. However, the process is restarted if the results are not satisfactory. In general, amendments are made to the explanatory hypotheses in light of any new observations, as opposed to being completely discarded.

3.3.5 Conclusion Phase

The conclusion phase pertains to the end of the research cycle and is generally the result of satisficing i.e., the results are considered presentable or of a satisfactory standard. The results are consolidated and the knowledge gained is categorised as one of the following: facts that have been learnt and repeatedly applied; behaviour that can be repeatedly invoked; or anomalous behaviour that defies explanation and may be material for the subject of future research.

3.4 Data Collection Process and Pre-Processing

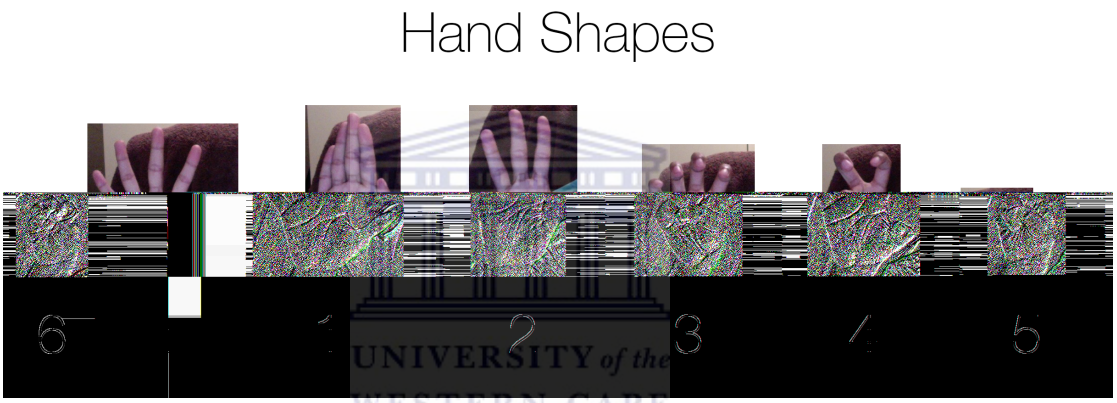


FIGURE 3.2: Hand shapes 1–6 recognised by the system

The data collection process was done ethically i.e., each of the research subjects used in the collection of the data, had the process and purpose of the data collection described to them before collection. Each of the research subjects had the opportunity to agree or decline to be a participant in the study by signing a consent form beforehand. Each participant agreed to having the video data be accessible within the research group and if any frame was used in a publication that their faces are covered.

Data collection was carried out using an iPhone 5C and the video stream was captured using a custom-designed video capture application at a resolution of 352×288 pixels. The process for data collection was as follows: the subjects used in the compilation of the dataset were each asked to perform six hand shapes in five hand orientations. The hand shapes are shown in Figure 3.2 and a matrix of these six hand shapes in each of the five orientations is shown in Figure 3.3. A program that utilises the hand tracking and segmentation component, that is described in the section that follows, was used to extract the left hands of the signees. A total of ten subjects were used in the data

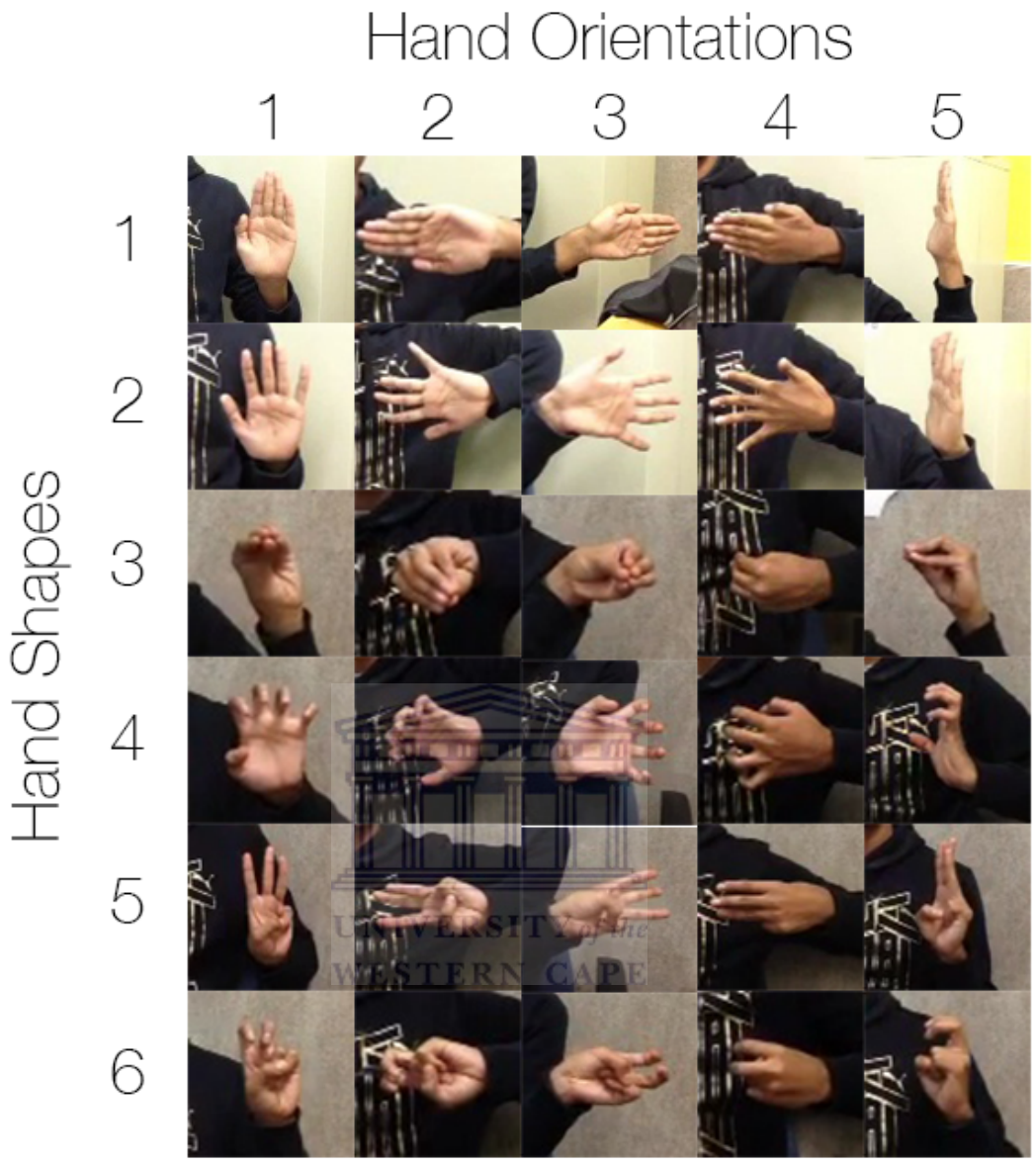


FIGURE 3.3: Hand shapes 1–6 perform in orientations 1–5. Together this results in 30 distinct classes

collection process. The images extracted from the video data were labeled for hand orientation and hand shape and a complete dataset was constructed. The dataset is split in the following manner: 5 subjects are used for the training set and the remaining 4 subjects are used for the testing set. Each of these sets—training and testing contain subjects that have variations in skin tone.

It is important to note that before any experiments were carried out, the dataset was pre-processed in the following manner: each of the images was scaled to 80×80 pixels and quantised to the greyscale colour space using the ImageMagick library. These grey-scaled images were transformed into the Torch7's native tensor format. Each of the tensors

were then normalised using per-example mean subtraction. This normalisation is applied in order to de-emphasise the illumination conditions of the image while emphasising its contents.

3.5 Application of Design Science Research to Hand Orientation Recognition and Hand Shape Recognition in Multiple Orientations

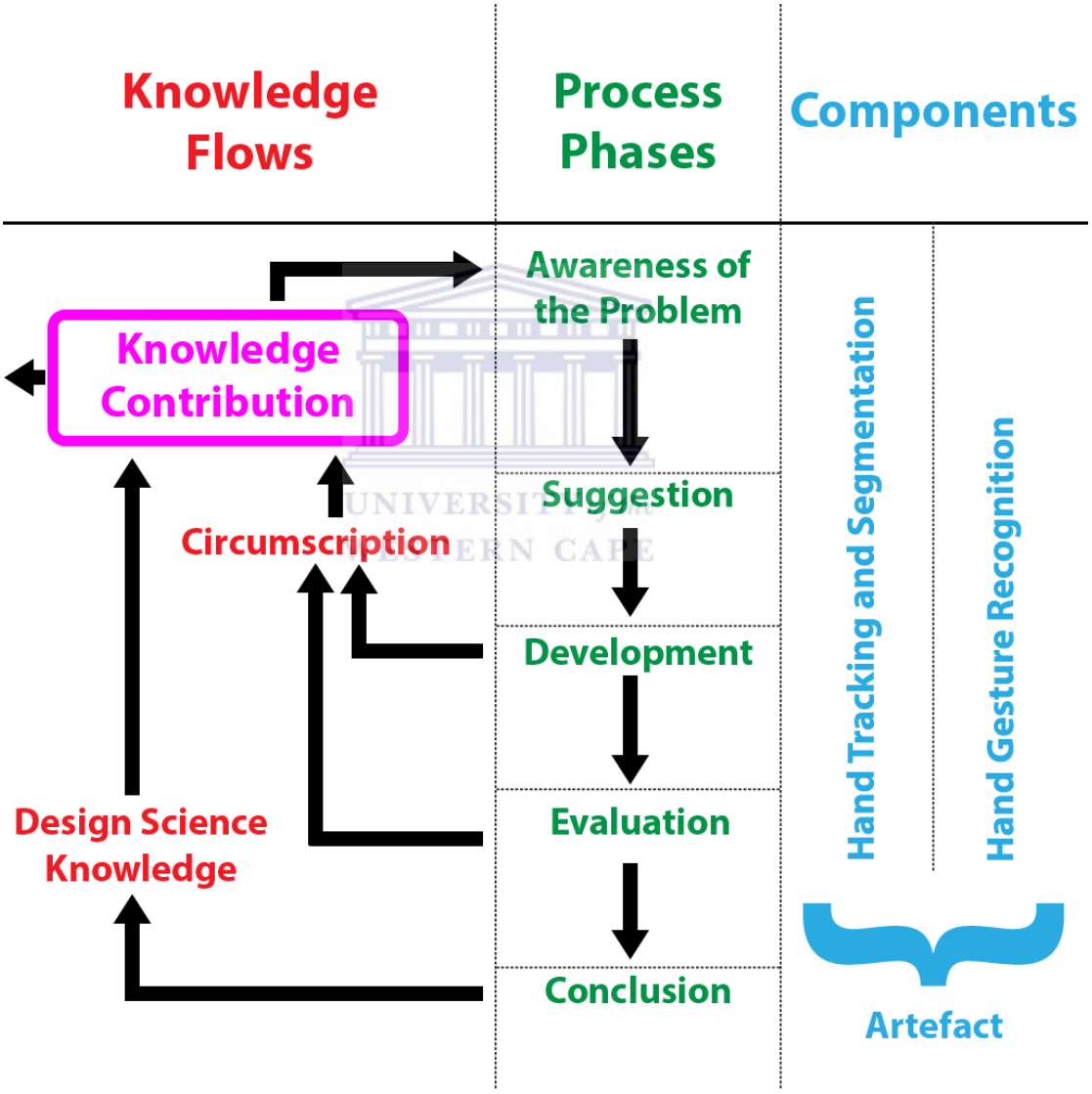


FIGURE 3.4: The Design Science Research process as applied to the research

Figure 3.4 indicates the DSR as applied to this research and the different cycles each separate component undergoes to produce the final artefact. The following section is structured as follows: Section 3.5.2 describes the hand tracking and segmentation component with respect to the various phases of DSR; Section 3.5.3 describes the hand

gesture recognition component with relation to the various DSR phases. The awareness phase, discussed in Section 3.5.1, for both of these DSR cycles are the same and only the remaining phases are discussed in the respective cycles for each component.

3.5.1 DSR–The Awareness Phase

Awareness of the South African Sign Language gesture recognition problem was brought to the attention of the researcher by the Assistive Technologies Research Group at UWC. The two sign language parameters, hand shape and hand orientation, have not yet been recognised in combination with one another within the Assistive Technologies Research Group or in the literature. However, a number of research efforts are found in the literature focusing on hand shape in a single orientation, hand location, hand motion and facial feature detection. The core research within the Assistive Technologies Research Group has placed a focus on desktop and laptop computers using webcams. It is clear that the system would be of greater value in the context of embedded devices, in particular, mobile devices. Embedded and mobile devices are becoming increasingly powerful with each passing year and are easier to carry around than a laptop or desktop computer with a webcam. Thus, an important research opportunity is the simultaneous recognition of hand orientation and hand shape on mobile and embedded devices. Consequently, a research proposal to create a system to recognise hand orientation and hand shape in multiple orientations on mobile devices was compiled.

3.5.2 DSR Cycle 1–The Hand Tracking and Segmentation Component

3.5.2.1 Suggestion Phase

A hand tracking and segmentation (HTS) component design was put forward. This HTS component is leveraged to extract hands from images or live video streams. The component is capable of tracking a single hand, the left hand of the signee.

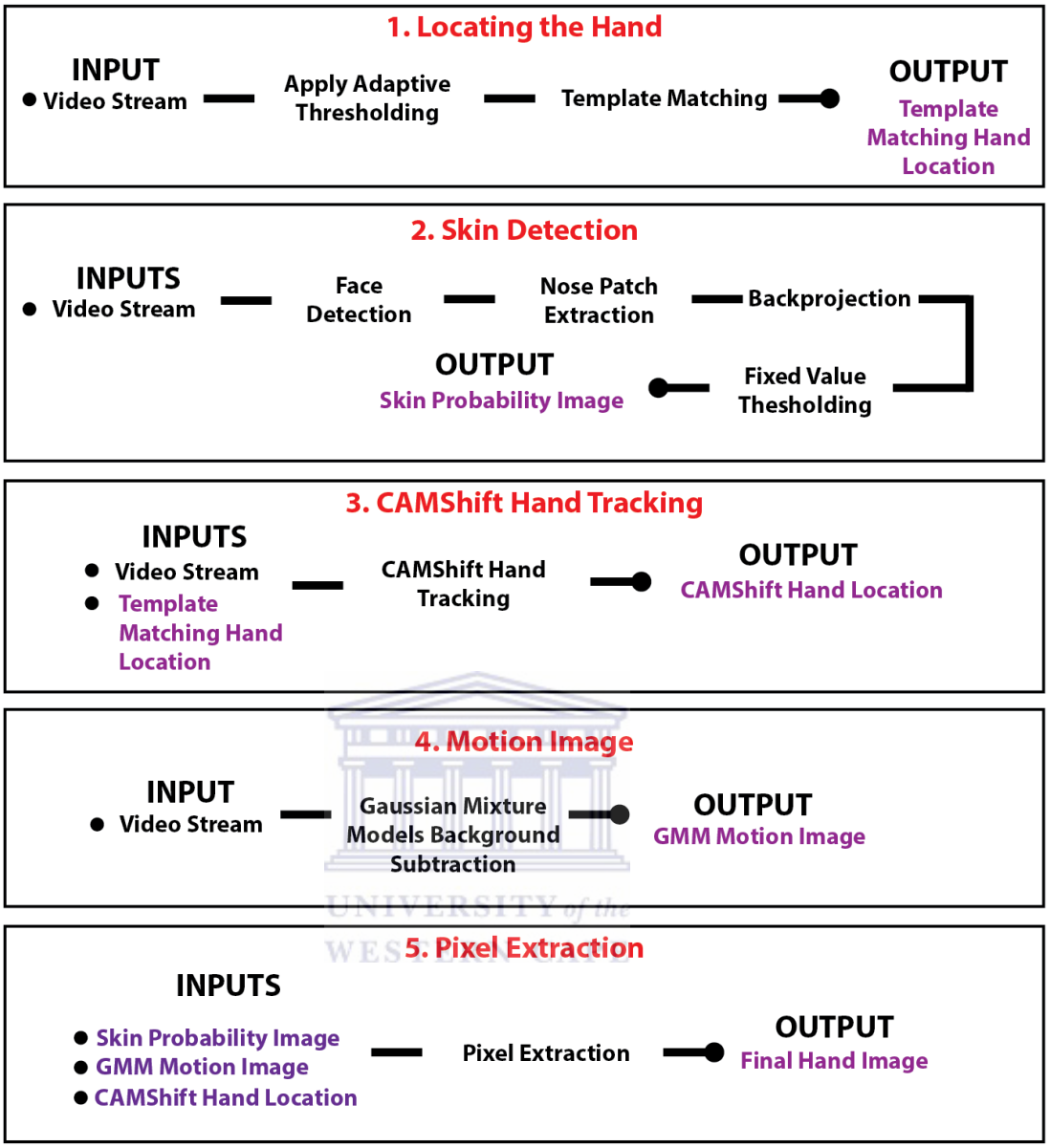


FIGURE 3.5: The image processing component's phases

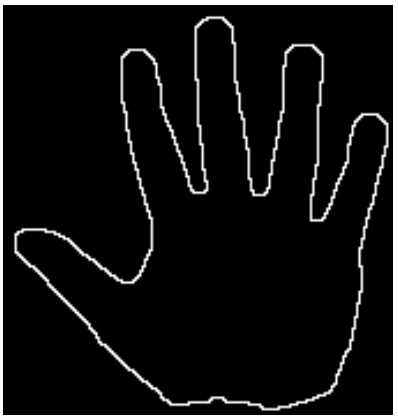


FIGURE 3.6: The hand template image used for template matching.

The HTS component design is described in Figure 3.5 and is comprised of a few key phases, namely: hand location using adaptive thresholding and template matching, face detection and back projection to obtain a skin image, CAMShift hand tracking, and background subtraction using gaussian mixture models.

Figure 3.5, using segments labelled 1–5, illustrates how these key phases are integrated to form the HTS component i.e., a each segment represents a phase of the HTS component. The outputs in the figure that are highlighted in purple are the same instances used as the purple highlighted inputs in their respective segments. The HTS component's phases are described below:

Segment 1, obtaining the hand location, is the process of locating the hand within an image and is achieved by means of template matching. Template matching is computationally expensive so it is only utilised initially, to obtain an initial set of coordinates for the location of the left hand. Only a single hand i.e., the left hand, is tracked but the system may be extended in the future to track both hands—gesture recognition should obtain similar performance for a right hand. Once the hand is located this template matching execution loop halts. The template image used for matching is shown in Figure 3.6. It is important to note that this process is a prerequisite for segments 2–5 to commence.

Segment 2, obtaining the skin image, involves the process for obtaining a skin image. This is an important phase in the recognition process because the success of extracting only the skin pixels in the image has an impact on learnable features in hand gesture recognition. The face is first detected using the Viola-Jones object detection framework and a small patch of skin is extracted in the nose region. The size of the patch is dynamically calculated in relation to the size of the rectangular coordinates of the face. The width and height dimensions are each divided by a factor of 2.8 and centered within the face box to obtain the patch region. A colour probability distribution, in the form of a histogram of this region, is constructed and back projected onto the input frame which produces a skin probability distribution image. A fixed threshold value of 60 is applied to the skin probability distribution to binarise this image [70]. All pixels exceeding this threshold value are set to white (skin) and all pixels below or equal to this value are set to black (non-skin). The output of this phase is a binary skin image.

Segment 3, CAMShift hand tracking, involves the process of hand tracking. It requires the initial coordinates of the hand obtained using template matching as input and proceeds to continuously track the hand using the CAMShift algorithm. The output of this phase is a set of hand coordinates in every subsequent frame.

Segment 4, obtaining the motion image, involves the process of obtaining a motion image. This is achieved by means of background subtraction using Gaussian Mixture Models (GMMs).

Segment 5, extracting the hand image, involves the process of obtaining the final extracted hand. In this phase, the skin and motion images are combined using a logical AND operation. The use of this combined image is important because the combined skin and motion images highlight only skin coloured objects that are moving—in this instance, a moving hand. This allows for the differentiation between any skin-coloured pixels that are incorrectly detected in the stationary background and actual skin pixels which are in motion. If a large portion of the motion image, greater than 65%, is black then only the skin image is used. The combined image and the CAMShift tracking coordinates makes it possible to extract the hand pixels from the input image.

Samples of the HTS procedure are provided in Figure 3.7 and samples of the hands that were extracted by the HTS component are shown in Figure 3.8. Section 3.4 describes the data collection process for the data used to train the HGR component.

FIGURE 3.7: Samples of the HTS phases

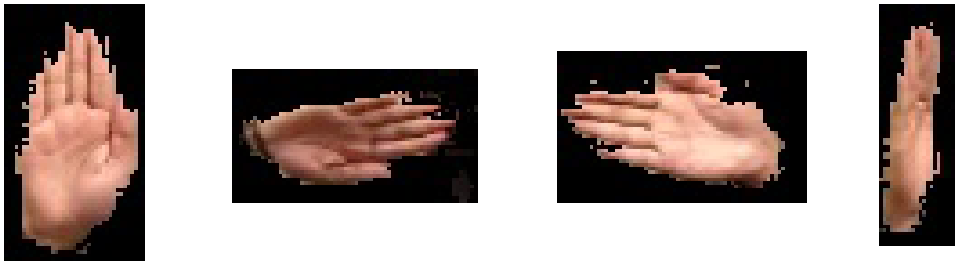


FIGURE 3.8: Sample of extracted hands

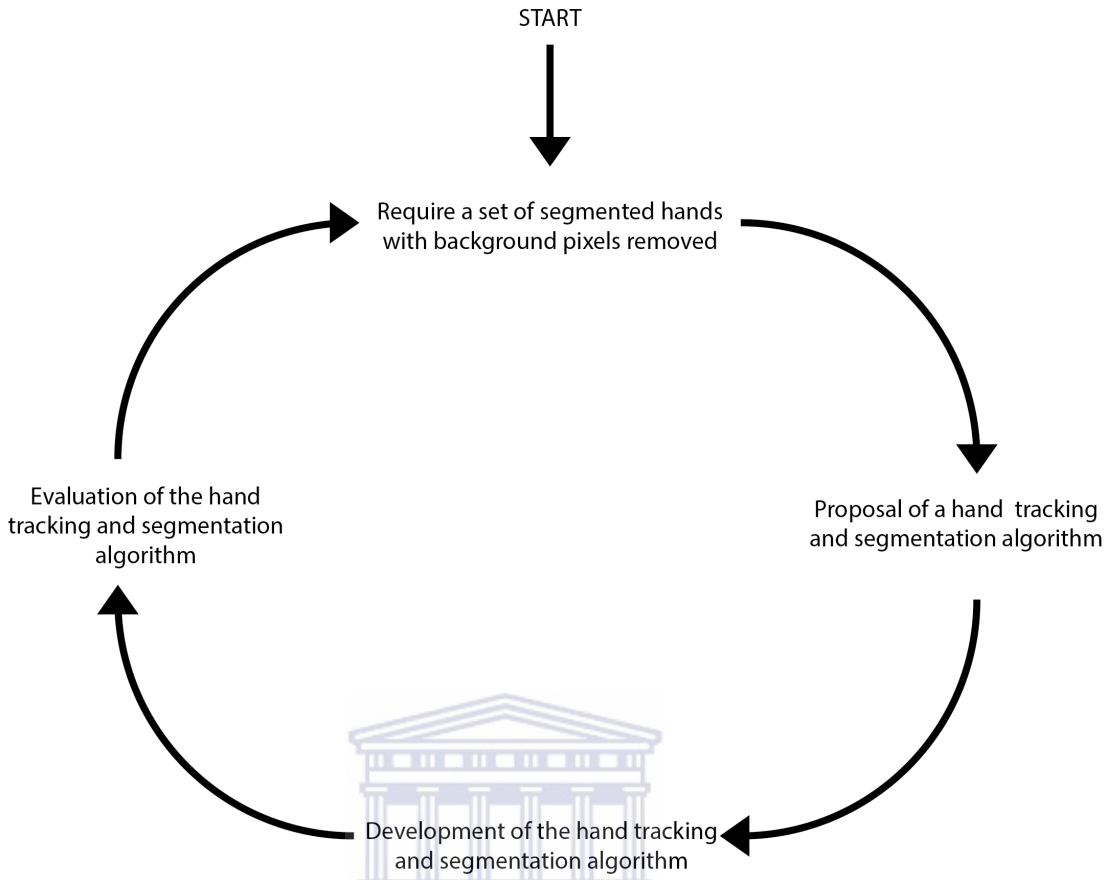


FIGURE 3.9: HTS development cycle

3.5.2.2 Development Phase

Referring to Figure 3.9, a system design is presented and developed, after which this system is evaluated based on the quality of the extracted hands. If the results are less than satisfactory then circumscription may follow.

3.5.2.3 Evaluation Phase

In this phase used the method of experiments to evaluate the performance of the HTS component. These experiments are carried out to test the robustness of the tracking and hand extraction of the HTS component. Once evaluation has been concluded, if the algorithm's performance is not satisfactory, the design process is revisited.

3.5.2.4 Conclusion Phase

The results obtained in the evaluation phase are analysed and added to the body of knowledge for computer vision. In the event that the results are less than satisfactory,

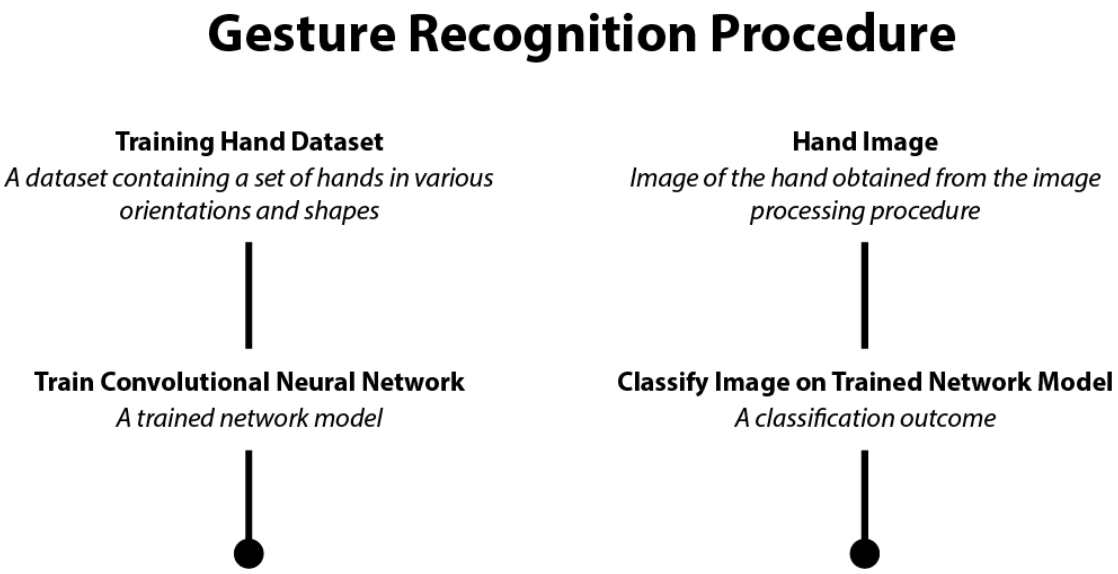


FIGURE 3.10: The hand gesture recognition component's phases

the process is revisited and enhanced to obtain a higher degree of accuracy.

3.5.3 DSR Cycle 2–The Hand Gesture Recognition Component

3.5.3.1 Suggestion Phase

Further extending on the awareness phase, a design of the hand gesture recognition (HGR) component was presented. A prototype CNN was developed and tested using the popular computer vision problem of handwritten digit recognition—the MNIST handwritten digit problem. This prototype can be altered for the problem space of HGR with a number of modifications to its architecture and micro-parameters.

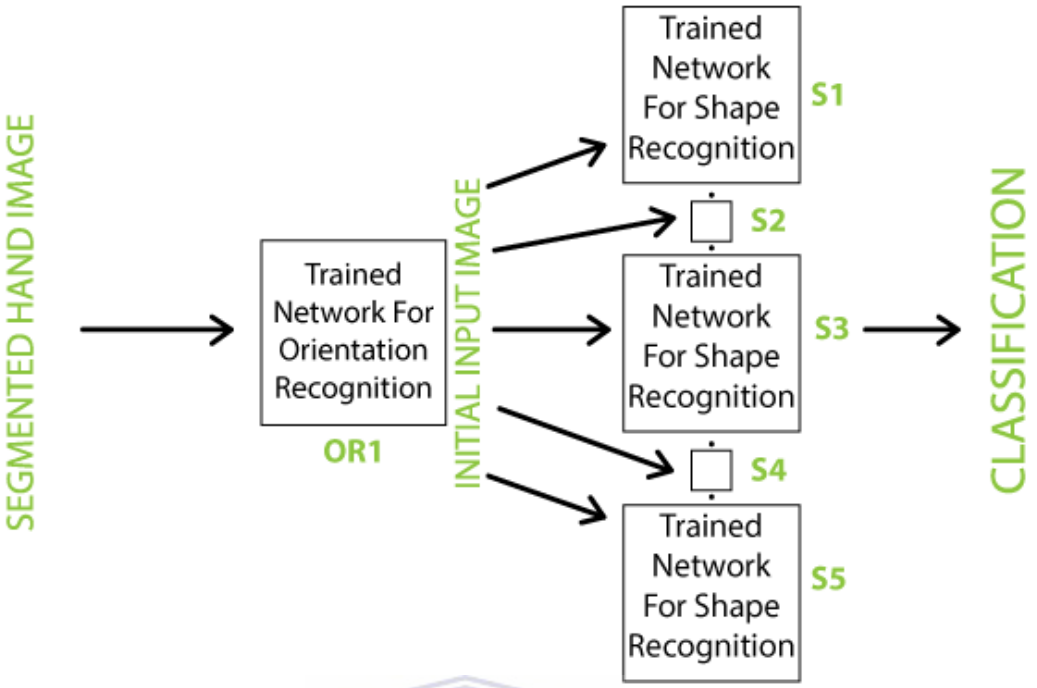


FIGURE 3.11: 2-Stage classifier construction

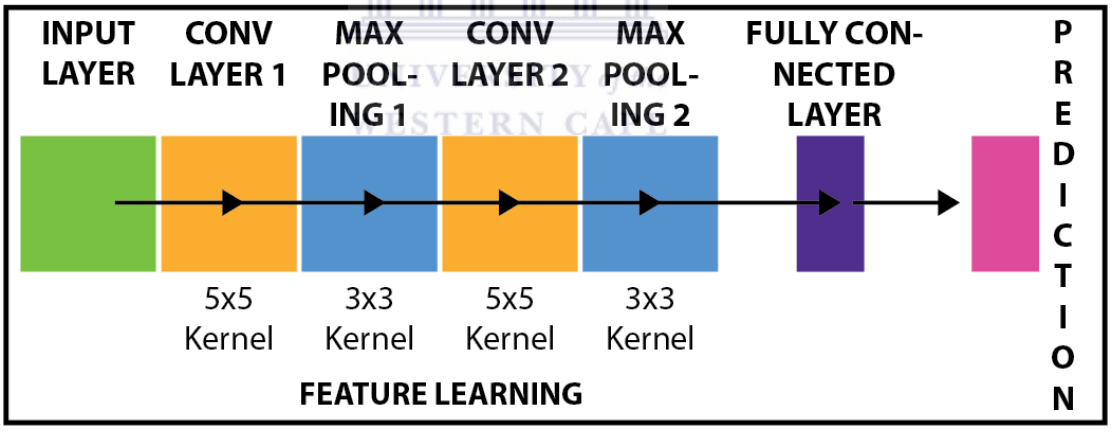


FIGURE 3.12: Architecture used for each of the MPCNNs.

The aim of this research is to detect both the hand shape and hand orientation simultaneously. The Figure 3.10 shows the flow for the recognition procedure. It indicates that a training dataset is used to obtain a model that is tested by using a test set for classification. The 2-stage classification approach is shown in Figure 3.11 and used when detecting the hand orientation and hand shape in various orientations of an unseen hand image. More information about the machine learning techniques used in this research may be found in Appendix B.

This 2-stage classification approach, involves a two-step procedure in which the detection of hand orientation and hand shape are conducted sequentially. First, the hand orientation in an image is detected using a single CNN trained to recognise only the hand orientation of any hand shape. This classifier will be called the "orientation classifier" as it only recognises the orientation of a hand image. Once a hand orientation label has been obtained, the same input image is used as input to one of five hand shape classifiers, each of which has been trained to recognise the hand shape of images in a particular orientation.

This results in one CNN for hand orientation, followed by one of the possible five CNNs for hand shape. For example, if a hand image is predicted as being aligned in orientation by the orientation classifier, the image is then passed to the hand shape classifier that is trained to differentiate and recognise the hand shape of images aligned in that orientation only. All the CNNs are implemented using the Torch7 computing framework [71] and have the same architecture as shown in Figure 3.12. The architecture consists of two sets of successive convolution and max pooling layers, followed by a final fully connected layer that leads to the output layer.

FIGURE 3.13: The various skin tones in the dataset

When training the CNNs, the dataset was split in the following manner: the images of subjects 1, 2, 3, 6 and 7, whom have a range of skin tone variations, were used to train the system. The images of the remaining subjects, 4, 5, 8, and 9, which also consisted of a range of skin tone variations, were used for testing. These skin tone variations are illustrated in Figure 3.13. In order to train the hand shape classifiers, the data was split into separate subsets, each consisting of the images of a particular hand orientation in all hand shapes, with 30 examples taken per subject.

Thus, the total number of examples per hand shape classifier can be calculated as follows: 30 examples $\times$ 5 subjects $\times$ 6 hand shapes = 900 training examples, and 30 examples $\times$ 4 subjects $\times$ 6 hand shapes = 720 testing examples per hand shape classifier. The hand orientation classifier data set was split into a training and testing set. Once again, 30 examples per subject were taken for each set. The total number of examples can be calculated as follows: 30 examples $\times$ 5 subjects $\times$ 6 hand shapes $\times$ 5 hand orientations = 4500 training examples and 30 examples $\times$ 4 subjects $\times$ 6 hand shapes $\times$ 5 hand orientations = 3600 testing examples.

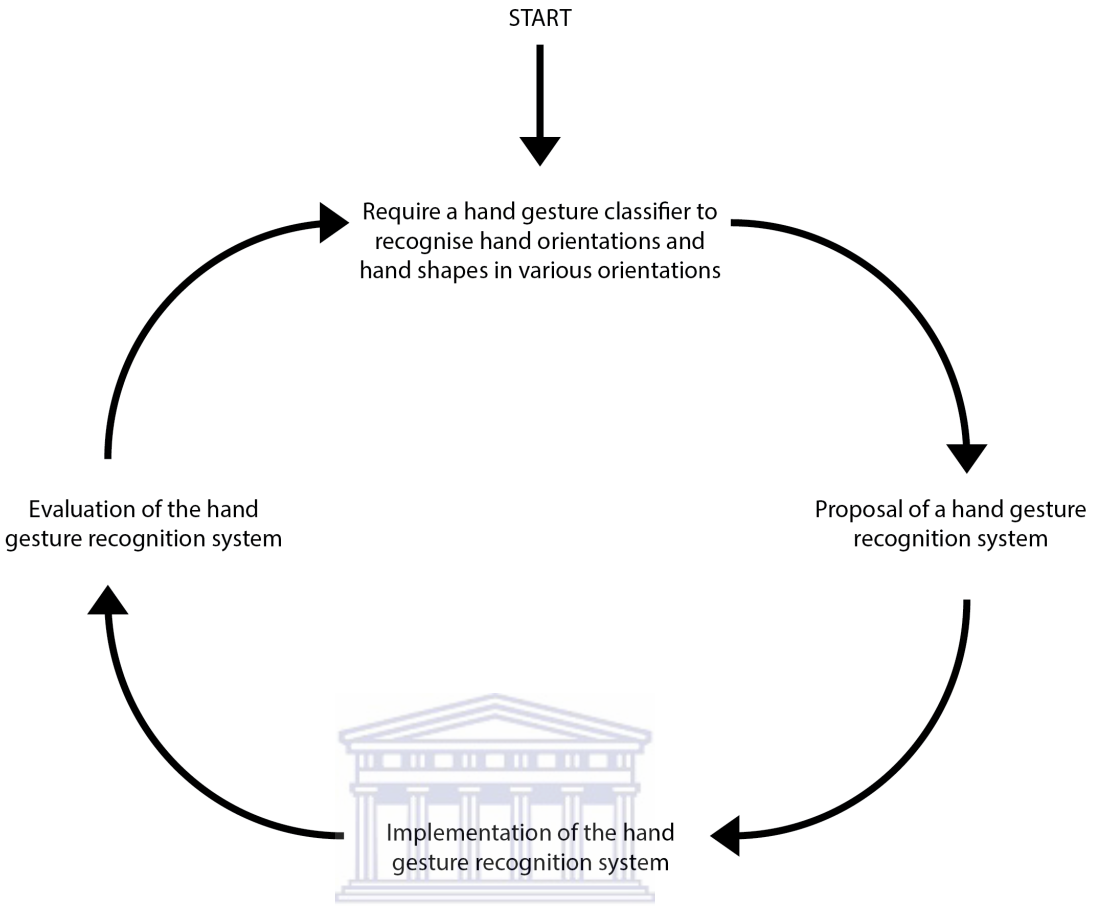


FIGURE 3.14: HGR development cycle

3.5.3.2 Development Phase

Figure 3.14 shows the requirement of a gesture recognition system that is capable of recognising hand orientations and hand shapes in different orientations. The architecture of the CNNs are proposed, implemented and trained.

3.5.3.3 Evaluation Phase

In the evaluation phase the method of experiments is used to evaluate the performance of the HGR component. The series of experiments trained CNNs and evaluated them in terms of classification accuracy and classification speed. The evaluation process for hand recognition component is outlined in Figure 3.15. This figure indicates that the dataset was split into testing and training sets and subsequently used to train and test models against their respective sets in the context of classification accuracy and classification speed. The results obtained from the HGR component were then collated and evaluated. If the results were less than satisfactory, the cycle was restarted.

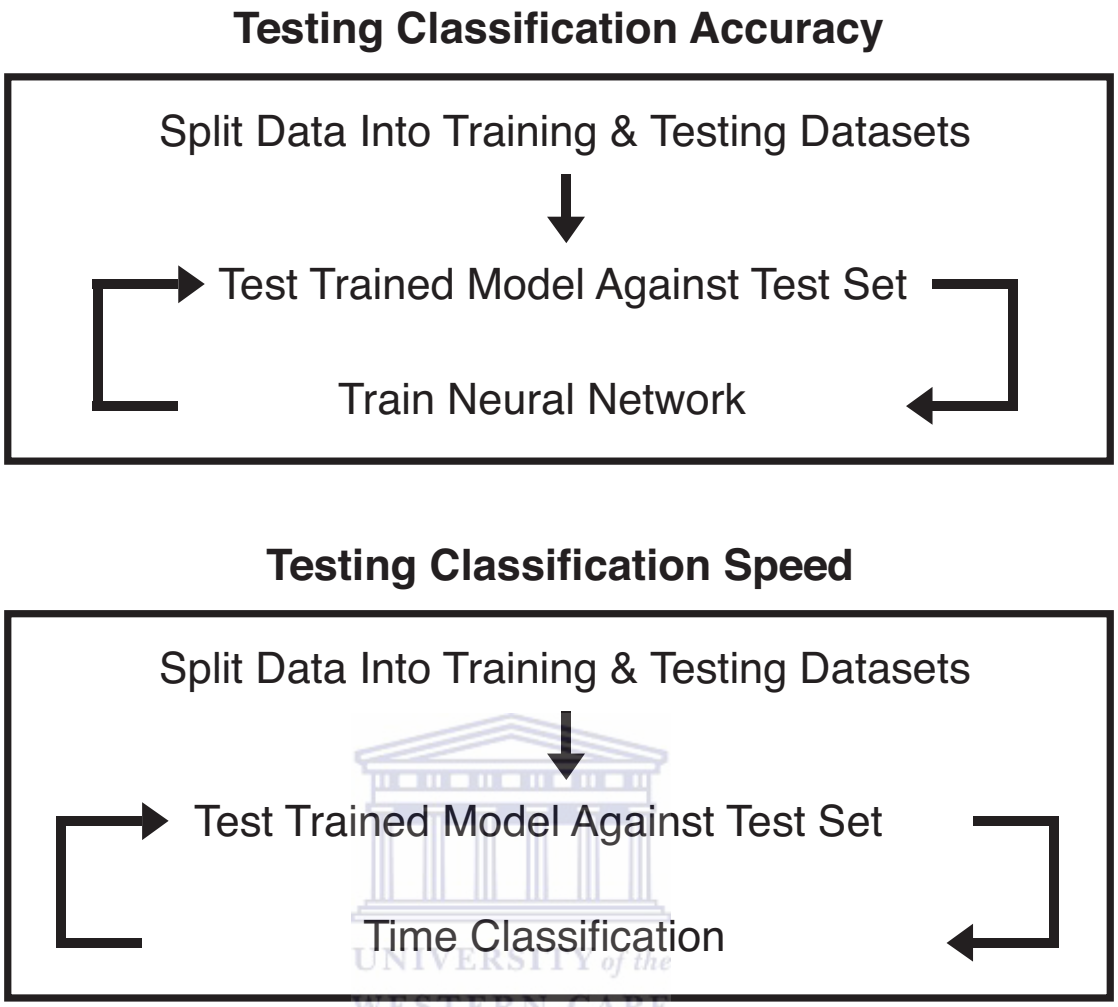


FIGURE 3.15: Process of training and testing the accuracy and speed of the gesture recognition component

3.5.3.4 Conclusion Phase

The results obtained in the evaluation phase were analysed and added to the body of knowledge for machine learning. In the event that the results were less than satisfactory, the process was revisited and enhanced to obtain a higher degree of accuracy.

3.6 The Integration of System Components

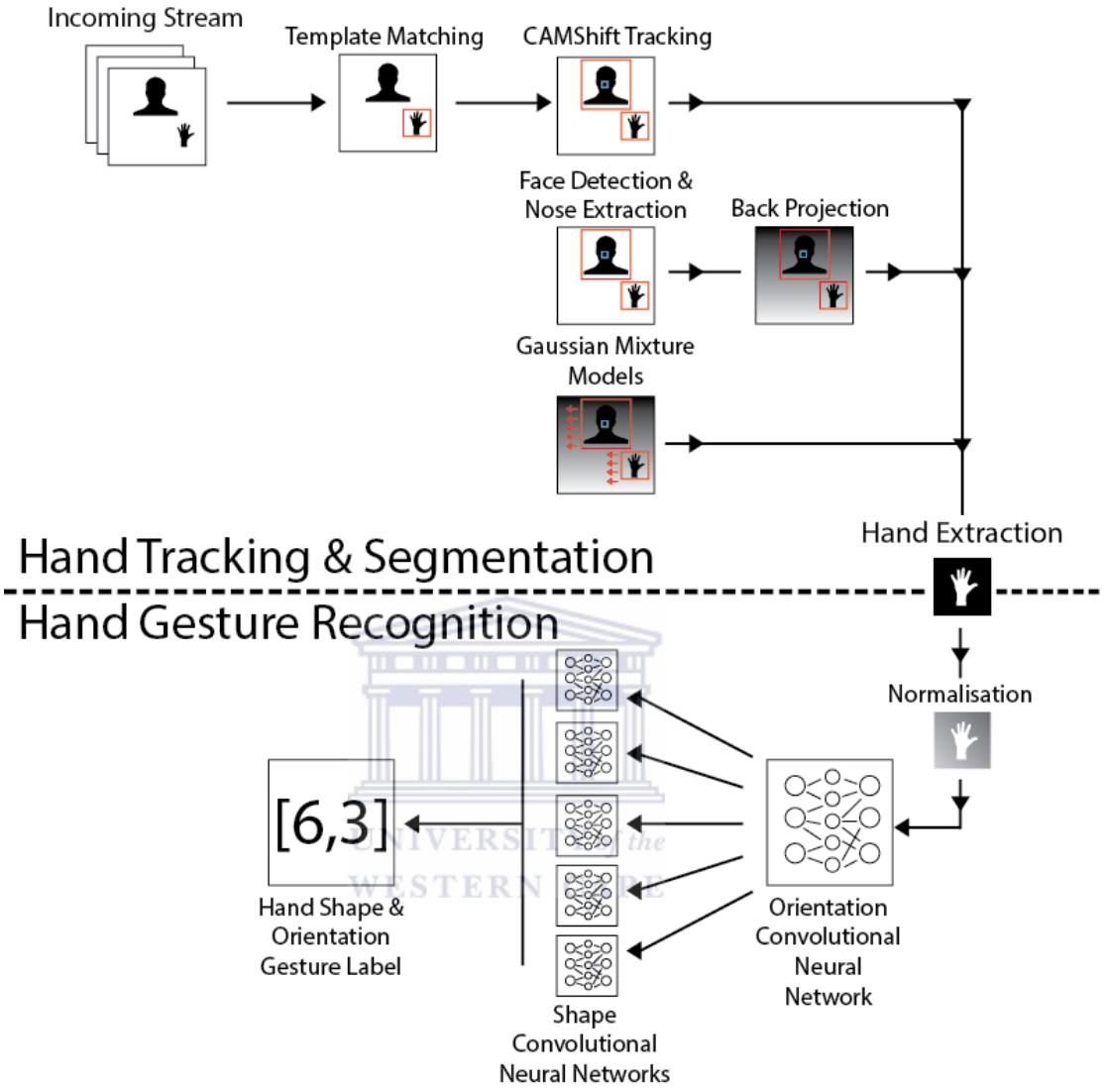


FIGURE 3.16: The integration of the HTS and HGR components

The integrated system, which is the amalgamation of the HTS and HGR components, as detailed above, is illustrated in Figure 3.16. The figure illustrates how the HTS and HGR components relate to one another and the flow of the full gesture recognition procedure.

3.7 Conclusion

This section described the DSR methodology, what it is comprised of and how it applies to the current research. In particular, it described the system and its components—the HTS component, the HGR component and the dataset used in training and testing

the HGR component. The next chapter describes the experiments carried out in the testing of the HTS and HGR components and the results of these experiments, and their analysis.

Chapter 4

Experimental Results and Discussion

This chapter provides an in-depth analysis of results obtained from experiments carried out to assess the performance of various components of the proposed system. These experiments aim to answer the research questions outlined in Chapter 1. This chapter is structured as follows: Section 4.1 describes an experiment to analyse the accuracy and speed of the face detection component on an iPhone 5C and iPhone 6+. Section 4.2 describes an experiment to analyse the accuracy of the HTS component.

Sections 4.3–4.6 describe a series of experiments carried out to analyse the accuracy of the CNNs, Hand Shape Classifiers 1–5 and the Orientation classifier, with various CNN parameters. In order, these are: comparing the use of hand images with and without a background, the use of various activation functions, the use of a number of different learnable filter and fully connected layer node configurations, and the use of different filter dimensions. Based on the result of these experiments, Section 4.7 compares each of the best Hand Shape classifiers and the Orientation classifiers to the default configuration.

Section 4.8 describes the accuracy of two-stage classification using the most performant CNN models as described in Chapter 3. Two-stage classification involves classifying a hand image to first determine its hand orientation, followed by its hand shape. In this case, a hand image is only determined to be correctly classified if both the hand orientation and hand shape are correctly predicted. This is the final CNN model accuracy. Section 4.8.1 describes the speed of classification of the classifiers executing on an iOS device.

Note that for ease of reference: hand shapes 1–6 will be referred to as H1 through H6; orientations 1–5 will be referred to as O1 through O5; each of the hand shape classifiers will be referred to as HSOi where i refers to the ith orientation, e.g. HSO1 refers to the hand shape classifier for orientation 1 etc.; the orientation classifier will simply be referred to as OR.

When assessing the accuracies of the classifiers, it is important to consider, and compare these accuracies to the accuracy of random guessing. It should be noted that any accuracy higher than $\frac{1}{x}$ for an x-class problem is considered to be effective and better than random guessing. For a 5-class problem, this equates to 20% and for a 6-class problem, approximately 17%. This accuracy will be referred to as the "random guessing" accuracy in the discussions below.

When assessing accuracy, experiments were carried out on a 3 GHz personal computer with 12 GB DDR3 RAM and a GeForce GTX 750 Ti GPU—512 MB RAM. To determine the computational speed, the iPhone 5C and iPhone 6 Plus were used for embedded context experiments. The operating system used was Ubuntu Linux 14.10 and the Torch7 distribution (GitHub commit id: 7c740d5e8ec7fc10edbc3a75f2667e481eb47180) for machine learning computation.

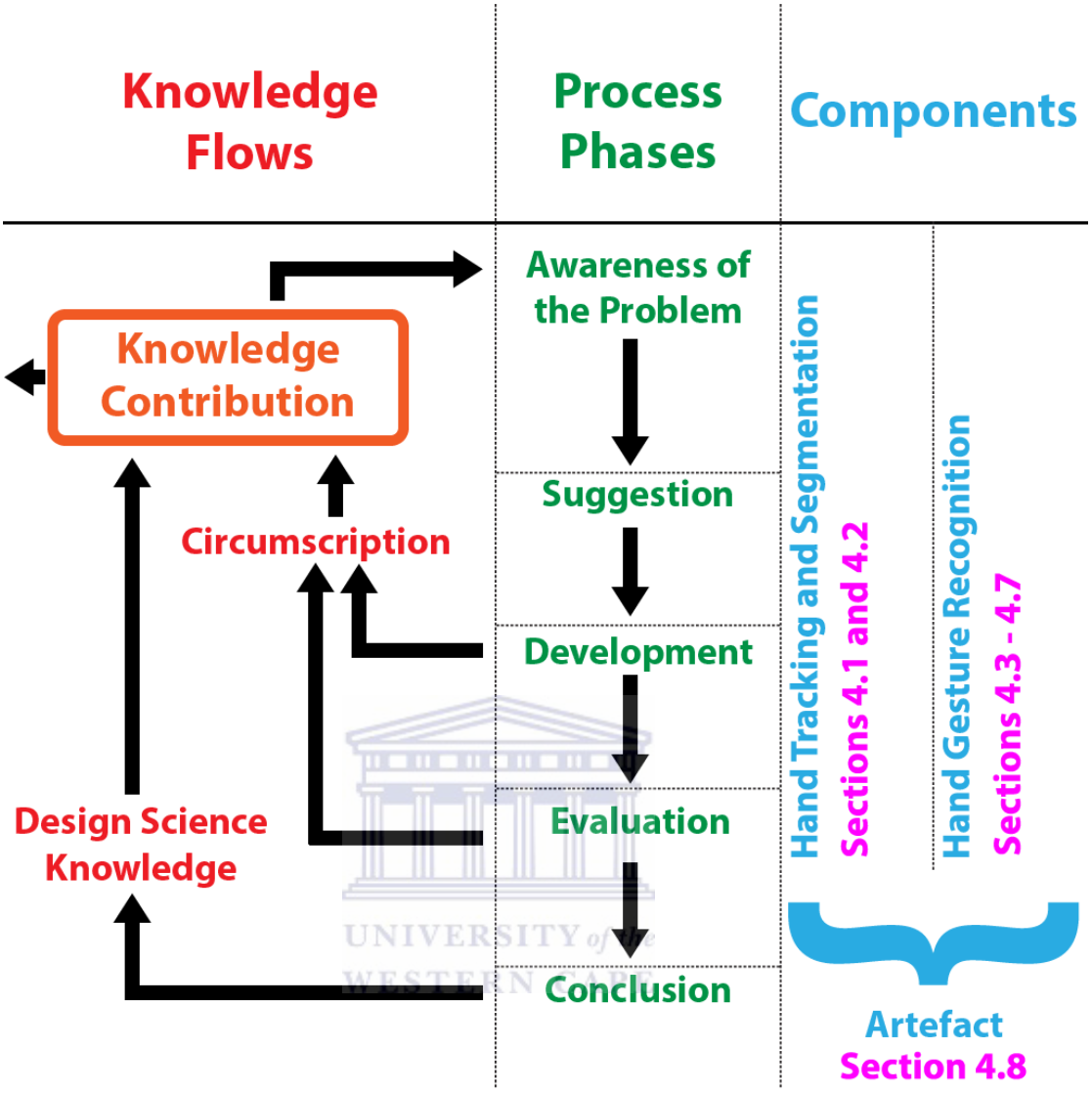


FIGURE 4.1: Results for the various artefacts in the DSR process

Figure 4.1 indicates the evaluation phase and which experiments are carried out in the testing of each cycle and component. As indicated, by the highlighted purple text in the diagram, Section 4.1 and Section 4.2 describe the experiments carried out in order to evaluate the HTS component. Sections 4.3–4.6 describe the experiments carried out in the evaluation of the HGR component. Section 4.8 describes the evaluation of the final artefact by its respective experiments.

4.1 Experiment to Assess Face Detection Performance

Face detection testing was carried out on two different devices, the iPhone 5C (A6 Chip—1.3 GHz Dual-Core, 512 MB RAM) and the iPhone 6 Plus (A8 Chip—1.4 GHz Dual-Core, 1 GB RAM). Four different face Haar cascades, created by the Intel Corporation,

were used for testing against sample images captured on the iPhone 5C at a resolution of 352×288. The dataset consisted of 1000 images of faces of the test subjects. The goal of the experiment was to test which cascade was the fastest across both devices. Each cascade was tested with all 1000 images over 50 iterations, i.e. a total of (1000 iterations $\times$ 50 images) = 50000 comparisons, and the average time for each device for a single image was recorded and is reported in Table 4.1. Although the accuracy of the cascades is expected to be the same in every iteration, the computational speed varies. Obtaining an average time over multiple iterations provides a better indication of computational speed.

TABLE 4.1: Summary of test performance for Haar cascades on two iPhones (time taken presented in seconds)

Cascade	**iPhone 5C**		**iPhone 6 Plus**		**Accuracy (%)**
	Time (s)	**Speed** (fps)	**Time** (s)	**Speed** (fps)	
frontalface_default	0.030477	32.812	0.014228	70.284	100
frontalface_alt	0.022945	43.582	0.011222	89.111	100
frontalface_alt2	0.024714	40.463	0.016270	61.463	100
frontalface_alt_tree	0.027913	35.826	0.019970	50.075	100

An analysis of the results indicates that each of the Haar cascades are comparable in terms of accuracy, with each cascade obtaining 100% accuracy. In addition, the frontalface_alt Haar cascade provides the best performance for both devices in terms of speed. On the other hand, frontalface_alt_tree is the least performant of all the Haar cascades in terms of speed. As expected the iPhone 6+ outperforms the iPhone 5C by a factor of at least 1.5 for each cascade.

The best performing Haar cascade took 0.022945 and 0.011222 seconds per frame for the iPhone 5C and iPhone 6 Plus respectively which translates to approximately 43 and 89 frames per second (FPS). This is more than sufficient for real time applications running at 24 FPS. However, all the cascades on both devices clearly perform at faster than real-time speed.

The results indicate that any of the Haar cascades may be used in terms of accuracy and will provide realtime performance in each instance. The Haar cascade, frontalface_alt, is used in the proposed system because it provides the highest frame rate.

4.2 Experiment To Assess Hand Tracking and Segmentation

This section tests the robustness of the HTS by testing whether the tracked hand is completely enclosed inside the tracking window. Videos of subjects 1–4 were used in this experiment. In each video, the subject first initialises the tracking by holding up an open palm. The tracking window is then automatically placed onto the hand and tracks it in each of the images in the video sequence. The subjects were not allowed to have their hands occlude the face region as per the assumption that was put forward in Chapter 1.

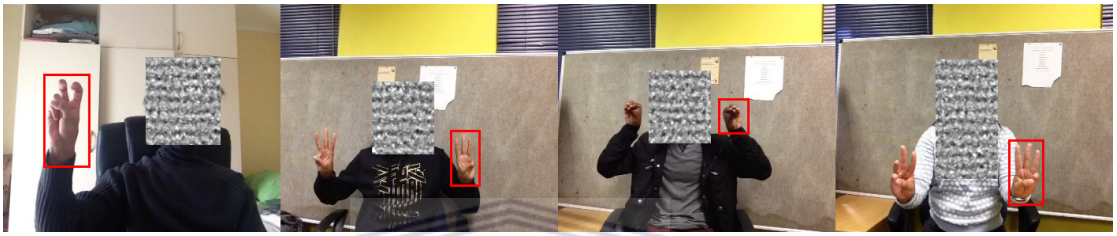


FIGURE 4.2: Samples of validly tracked frames

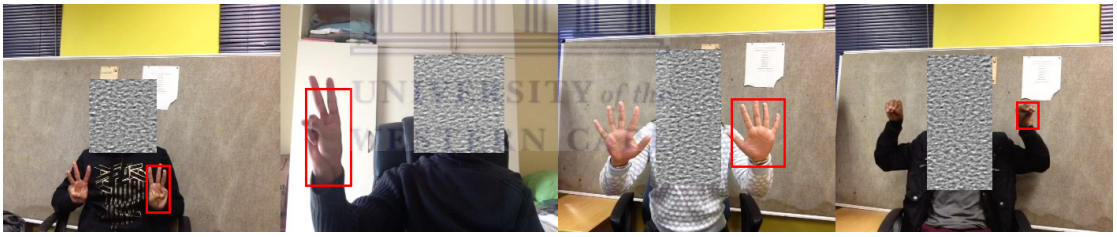


FIGURE 4.3: Samples of invalidly tracked frames

Each of the images in each video sequence was analysed manually by the researcher. The criterion for judging whether an image is tracked relates to the tracking window completely enclosing the hand, with no parts exceeding the bounding rectangle as illustrated in Figure 4.2. Figure 4.3 shows samples of invalidly tracked images in which the hand exceeds the bounding rectangle. This criterion is very stringent and categorises a frame as either being perfectly tracked or completely invalid. The faces of the subjects in these figures have been covered so as to not disclose their identities.

TABLE 4.2: Accuracies for the HTS component

Subject	Correctly Tracked Frames	Success Rate (%)
1	472/555	85.0
2	601/756	79.5
3	656/794	82.6
4	790/950	83.1
Average (%)	—	**82.58**

Table 4.2 summarises the results for each subject. It is observed that the HTS component managed to consistently track the hand, without any tracking loss, for all video sequences. Overall, a very high average accuracy of 82.58% was obtained across all four subjects. This is a very encouraging accuracy that indicates that the hands of the subjects fall completely within the tracking window, in the majority of frames.

On a per-subject basis, it is observed that the accuracies ranged from 85.0% to 79.5%, which are all very closely aligned with the average. This indicates that the proposed HTS component performs consistently across subjects. This is especially encouraging given the subjects were very diverse.

The results indicate that the HTS component is very robust in terms of tracking the user's hands and provides a solid foundation for the hand shape and hand recognition component.

4.3 Experiment To Assess CNNs On Background And Non-Background Datasets

The goal of this experiment was to determine whether removing the background from the hand images aids in CNN classification. The model architecture described in Figure 3.12 was used to train a set of CNNs. One network was trained to recognise orientation classes—the OR classifier—and five hand shape networks—HSO1–HSO5—were trained to recognise shape classes in each of the orientation classes. A default model configuration for all networks in this experiment as per [72] was used. This model is a 2-Layer Max Pooling CNN with kernel and node configurations 5C→5P→4C→3P→500L, where C = Convolution, P = Pooling, and L = Linear or Fully Connected Layers. The activation function used in this experiment was the hyperbolic tangent function (TanH). Two sets of data were used to train the CNNs, namely, hands with background (BG) and hands without background (NBG). The background for BG was removed as described in the HTS component in the previous chapter.

TABLE 4.3: Dataset Values

	Set Name	
	Train	**Test**
Number of Subjects	5	4
Shape Instances Per Subject	180	180
Orientation Instances Per Subject	900	900
Total Shape Instances	900	720
Total Orientation Instances	4500	3600

Each of the datasets, BG and NBG, are exactly the same size and the values for these datasets are tabulated in 4.3 for ease of reference.

TABLE 4.4: Average accuracies for each classifier with BG vs. NBG

Data Set	**Average Classifier Accuracy (%)**					
	HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
BG	63.6	64.4	57.3	44.3	52.7	90.8
NBG	88.6	74.3	86.8	59.8	60.6	90.7

The testing set was then used to test the resulting models. Table 4.4 summarises the average accuracies obtained for each of the classifiers for the two datasets. It is evident that NBG outperforms BG in every instance except for OR. The differences in average accuracies for HSO1–HSO5 are 25.0%, 9.9%, 29.5%, 15.5% and 7.9% respectively. These differences are very noticeable, especially in the case of HSO1, HSO3 and HSO4. This demonstrates that the removal of the background is beneficial for each HSO classifier.

It is very interesting to observe that BG performs slightly better than NBG for OR, if even by an extremely small margin of 0.1%—3 images. Contrary to expectation, this means that having unwanted noise in the images leads to a slightly better accuracy than that obtained for images with background subtracted hands. The reason for this increase in accuracy for the background dataset may be attributed to the fact that a larger number of samples allows for the OR classifier to better generalise, despite the noise in the data. OR is trained on 3600 samples, compared to the 720 samples used for each HSO. Thus, for the OR classifier, the large variations in raw hand images, coupled with an abundance of examples, potentially allows for better filters to be learnt by the CNN.

NBG provides a better accuracy when compared to BG for each of the HSO classifiers, as well as a very small difference for the OR classifier, so in general, NBG can be considered to be a better choice. Thus, it is beneficial to utilise computational resources to remove the background from the hand image using the proposed HTS component.

A further analysis of the comparison between BG and NBG for each classifier is provided below to compare the respective accuracies at the class-level. Noting that it is very difficult to determine the exact reason for a classification decision by a classifier [73], this analysis—and all subsequent analyses in this chapter—attempt to provide an indication as to the cause of incorrect classifications.

4.3.1 BG vs. NBG Comparison For HSO1

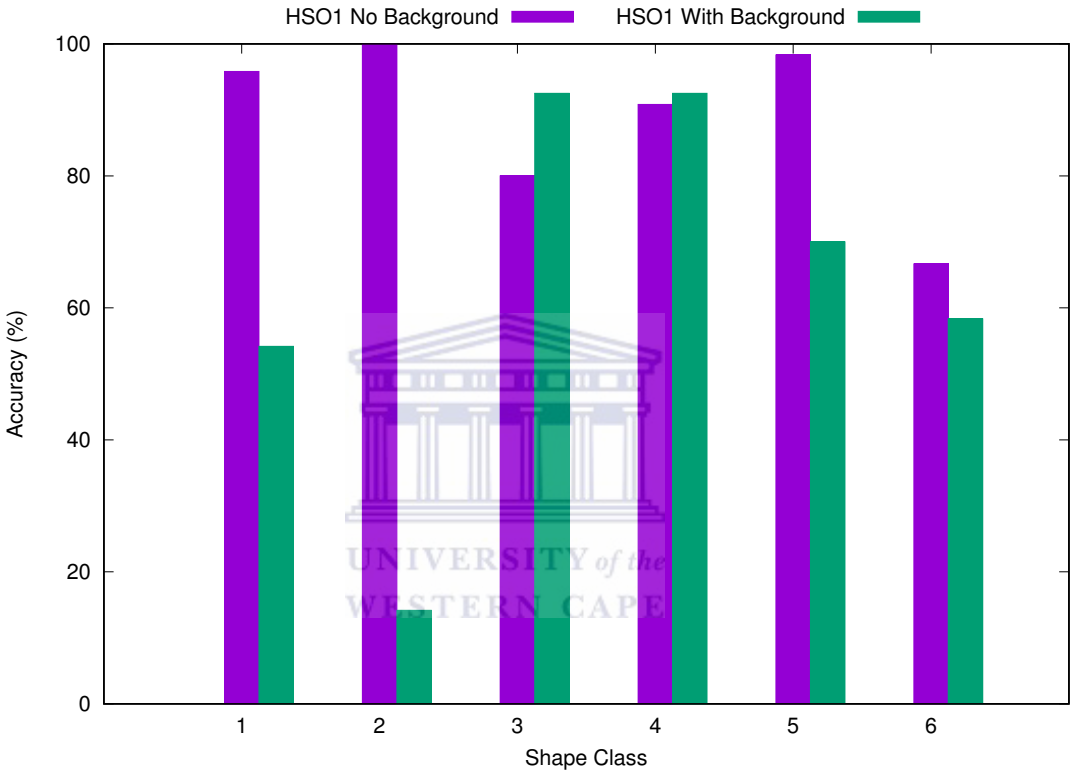


FIGURE 4.4: HSO1 classifier results for BG vs. NBG

Figure 4.4 is a graph of the average NBG and BG accuracies for each shape class in HSO1. An analysis of the graph indicates that NBG outperforms BG for four of the six possible shape classes, namely, HS1, HS2, HS5, and HS6. The difference in accuracy between BG and NBG for the fourth class is only 2%. HS3 is the only class for which BG outperforms NBG by a relatively large difference, of 12%, which equates to 14 images. This seems to indicate that when NBG outperforms BG, it does so by a large amount, and when BG outperforms NBG it does so by a smaller amount. It is, therefore evident that NBG generally outperforms BG for HSO1.

TABLE 4.5: Confusion matrix for HSO1 NBG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	115	4	0	1	0	0	95
2	0	120	0	0	0	0	100
3	1	3	96	0	10	10	80
4	4	2	2	109	0	3	90
5	0	1	0	0	118	1	98
6	1	0	8	0	31	80	66
Avg							88.6

An analysis is carried out to further analyse the accuracies of the two lowest, but by no means low, performing classes of NBG, namely HS3 and HS6. A confusion matrix of the samples for NBG is provided in Table 4.5. Observing the table, it is seen that HS3 is confused with shape classes HS5 and HS6 in most instances. Due to imperfect hand segmentation caused by variance in lighting during video capture, the fingers of HS3 are clipped off in a small number of instances. In these images, only the palm section, with the thumb seemingly crossed over it, is visible as shown Figure 4.5. HS5 and HS6 include the thumb crossed over the palm. Therefore, one possible reason that HS3 is confused with HS5 and HS6 may be that the available features in the image resemble HS5 and HS6. In these specific images, lighting changes caused the fingers of these subjects to appear differently to the skin of their faces, thereby omitting them from the back-projected and segmented image.

The class with the lowest, but by no means low, accuracy of HSO1 with NBG is HS6 which obtains an accuracy of 66%. The largest number of confused instances for HS6 is with HS5, for a total of 31 instances. The reason for these incorrect classifications could be attributed to the fact that these shapes—HS6 and HS5—appear visually similar, with only a single finger difference in shape. In the hand shapes, the bottom section of the hand shape is very similar. Therefore, HS6 may be confused with HS5. This is further confirmed by the fact that HS5 is also confused with HS6 in the confusion matrix. Samples of the incorrectly classified HS6 instances are shown in Figure 4.6.

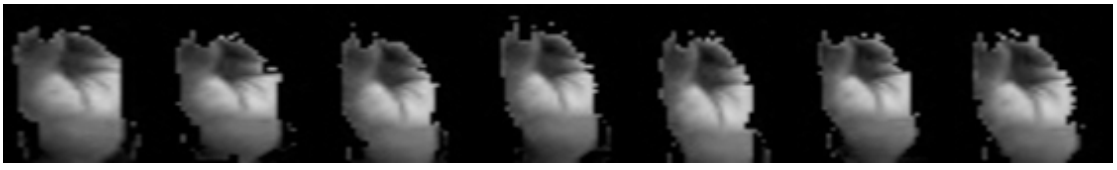


FIGURE 4.5: HSO1 with NBG—incorrectly classified instances of HS3

FIGURE 4.6: HSO1 with NBG—incorrectly classified instances of HS6

TABLE 4.6: Confusion matrix for HSO1 BG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	65	11	0	44	0	0	54
2	59	17	0	44	0	0	14
3	0	0	111	9	0	0	92
4	0	0	9	111	0	0	92
5	10	0	0	26	84	0	70
6	3	0	12	35	0	70	58
Avg							63.6

Table 4.6 is a confusion matrix of BG for HSO1. An analysis is carried out on the BG confusion matrix in order to also further analyse the incorrectly classified instances. It is interesting to note that the highest accuracies are exactly the same for shape classes HS3 and HS4 and even more so, that these classes are confused with each other in all cases, with the number of confused instances exactly the same between these two classes. Furthermore, each of the other classes has a noticeable number of confused instances with HS4. In the case of HS2, which obtains the worst accuracy with all of its confused instances being misclassified as either HS1 or HS4, it performs worse than the random classification error. The exact reason for these confusions occurring is difficult to determine. However, it may be attributed to the added noise in the images, which leads to difficulty in the classification procedure.

It is interesting to note that when comparing the results for HS3 and HS4 in HSO1 for both BG and NBG, BG obtains higher accuracies for these shapes. These drops in comparative accuracies could be attributed to the fact that the hands in NBG for these shapes, HS3 and HS4, are clipped. Thus, it appears that despite having more noise in BG, it is able to obtain a higher accuracy for these classes.

4.3.2 BG vs. NBG Comparison For HSO2

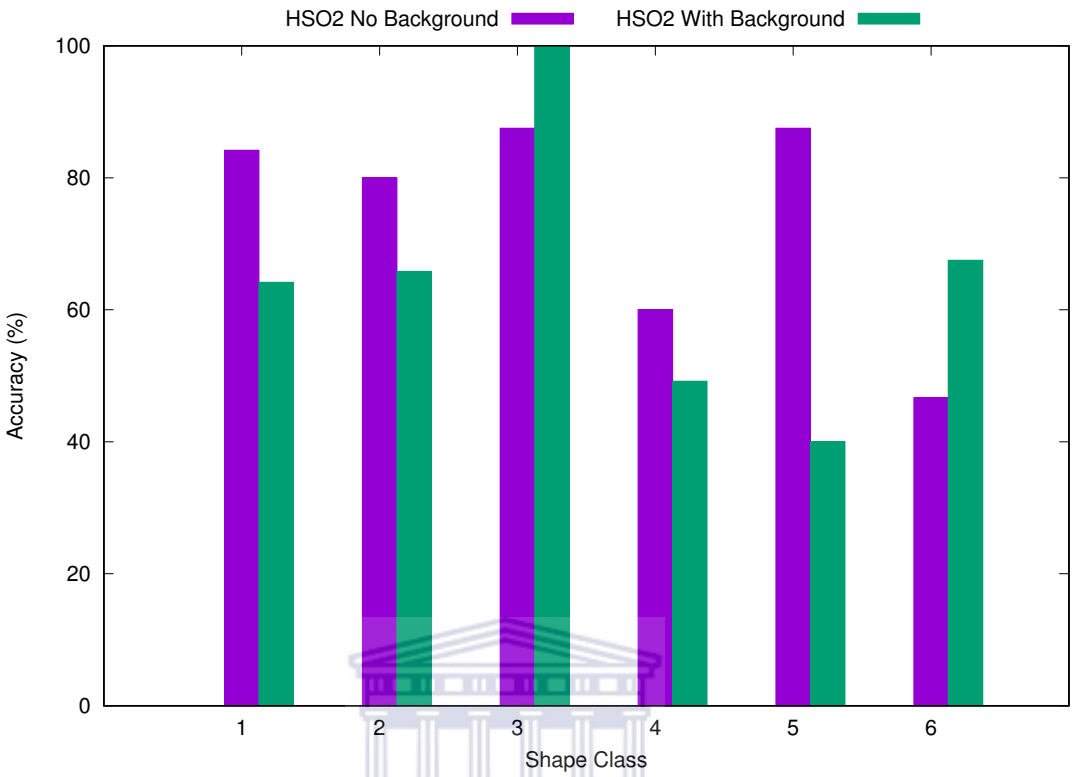


FIGURE 4.7: HSO2 classifier results for BG vs. NBG

Figure 4.7 is a graph of the average NBG and BG accuracies obtained for each shape class in HSO2. An analysis of the graph indicates that NBG outperforms BG by a substantial margin for four classes: HS1, HS2, HS4 and HS5. The graph also indicates that BG does, however, outperform NBG for classes HS3 and HS6 by a noticeable difference. The differences in accuracies, 13% and 21%, between BG and NBG are relatively large for HS3 and HS6 respectively. Overall, NBG provides a better average accuracy in more classes when compared to BG with a difference in average accuracy of 9.9%.

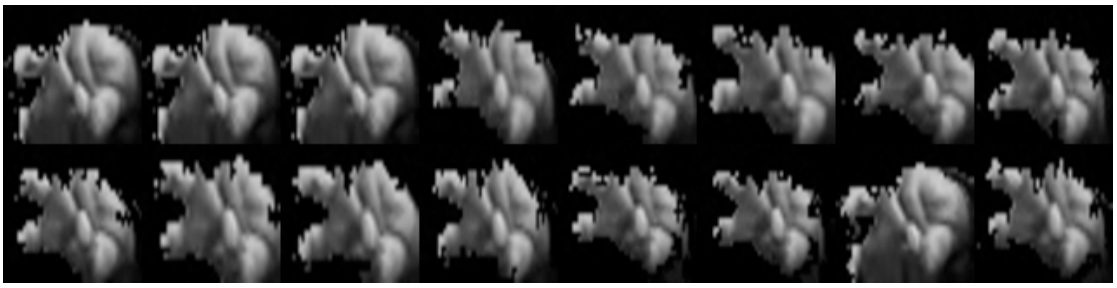
TABLE 4.7: Confusion matrix for HSO2 NBG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	101	4	0	1	11	3	84
2	0	96	0	11	13	0	80
3	1	0	105	12	0	2	87
4	0	1	22	72	18	7	60
5	0	0	0	6	105	9	87
6	2	0	4	2	56	56	46
Avg							74.3

An analysis is carried out to further analyse the accuracies of the HSO2 NBG classifier. Table 4.7 is a confusion matrix of the HSO2 classifier, and when analysed, indicates that the lowest performing shape classes are HS4 and HS6. The performance of these hand shapes is discussed in further detail below.

HS4 has a relatively large number of samples being incorrectly classified, of which a large number are being confused with HS3 and HS5. The confused instances occurring between HS4 and HS3 challenging to explain due to the distinct differences in these shapes for this orientation. The confused instances between HS4 and HS5 are also attributed to random classification error as these shapes are also very dissimilar.

Regarding the relatively lower performance of HS6, this hand shape is clearly confused with HS5 in the majority of cases. The confused instances between HS6 and HS5 could be attributed to the fact that these shapes appear visually similar for this orientation. This is further confirmed by the fact that HS5 is also confused with HS6 in the majority of its cases in the confusion matrix. These similarities are illustrated in Figure 4.8 which shows images of HS6 which are incorrectly classified, and images of HS5 in this orientation for reference.



(a) HS6.

(b) HS5.

FIGURE 4.8: Confusions between HS6 and HS5 for HSO2: a) Sample images of HS6 for HSO2 that are incorrectly predicted as being HS5; b) Sample images of HS5 for HSO2, demonstrating that they appear similar to those of HS6 in this orientation

TABLE 4.8: Confusion matrix for HSO2 BG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	77	9	30	0	4	0	64
2	22	79	0	8	11	0	65
3	0	0	120	0	0	0	100
4	0	0	34	59	0	27	49
5	0	16	0	0	48	56	40
6	0	0	36	2	1	81	67
Avg							64.4

Table 4.8 is the confusion matrix of BG for HSO2. An analysis is carried out on this confusion matrix to also further analyse the incorrectly classified instances. HSO2 also has a single high accuracy class, namely, HS3. It should be noted that classes HS1, HS4 and HS6 are being confused with class HS3 for a noticeable number of instances. Despite these random classification errors attributed to by HS1, HS4 and HS6, each of the classes in this classifier perform at least 2.5 times better than the random guessing accuracy.

The difference in accuracies for HS6 in BG and NBG could be attributed to the fact that NBG had a large number of its instances clipped at the fingers for this shape. The difference in accuracy between datasets for HS3 is difficult to determine.

4.3.3 BG vs. NBG Comparison For HSO3

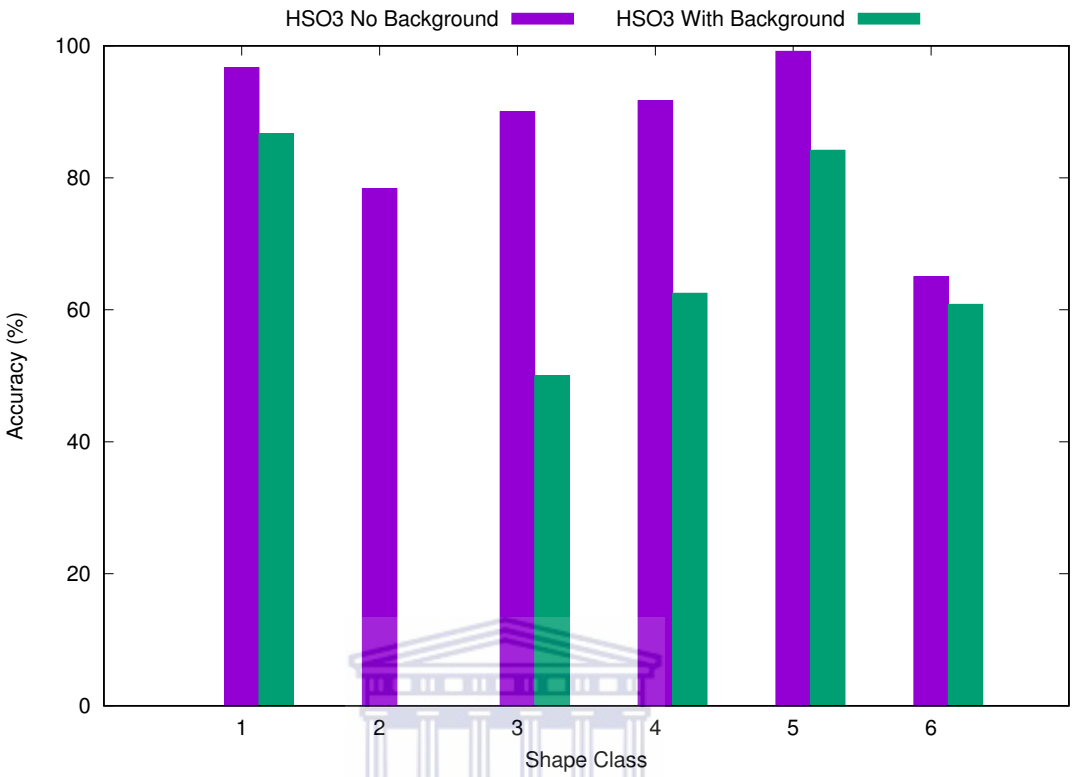


FIGURE 4.9: HSO3 classifier results for BG vs. NBG

Figure 4.9 is a graph of the average accuracies obtained for NBG and BG for each of the shape classes in HSO3. As with the previous classifiers, NBG outperforms BG and, in this orientation, it does so for every shape class, HS1 through HS6. Overall, NBG outperforms BG with a large average accuracy difference of 29.4% which amounts to a 211 image difference.

TABLE 4.9: Confusion matrix for HSO3 NBG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	116	0	0	4	0	0	96
2	26	94	0	0	0	0	78
3	0	0	108	1	8	3	90
4	7	0	0	110	1	2	91
5	0	0	0	0	119	1	99
6	1	0	5	9	27	78	65
Avg							86.8

Table 4.9 is a confusion matrix of the HSO3 NBG classifier, and when analysed, indicates that the least performant classes in HSO3 are HS6 and HS2 with accuracies of 65% and 78% respectively. The confused images that occur for these classes are further analysed below.

Similar to the previous classifiers, the largest number of confused image samples of HS6 is predicted as being HS5. As mentioned before, HS6 may be confused with HS5 because they appear visually similar. The two hand shapes have a single finger difference when clipping occurs during hand extraction. These similarities are illustrated in Figure 4.10 which shows images of HS6 which are incorrectly classified, and images of HS5 in this orientation for reference.

In the majority of cases, HS2 is incorrectly predicted as HS1. These confused images may be attributed to the fact that these shapes appear visually similar in this orientation if the gaps between the fingers in HS2 are not large enough. These similarities are illustrated in Figure 4.11 which shows images of HS2 which are incorrectly classified, and images of HS1 in this orientation for reference.

(a) HS6.

(b) HS5.

FIGURE 4.10: Confusions between HS6 and HS5 for HSO3: a) Sample images of HS6 for HSO3 that are incorrectly predicted as being HS5; b) Sample images of HS5 for HSO3, demonstrating that they appear similar to those of HS6 in this orientation

(a) HS2.

(b) HS1.

FIGURE 4.11: Confusions between HS2 and HS1 for HSO3: a) Sample images of HS2 for HSO3 that are incorrectly predicted as being HS1; b) Sample images of HS1 for HSO3, demonstrating that they appear similar to those of HS2 in this orientation

TABLE 4.10: Confusion matrix for HSO3 BG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	104	10	0	0	6	0	86
2	91	0	0	0	29	0	0
3	20	0	60	31	0	9	50
4	25	5	15	75	0	0	62
5	9	10	0	0	101	0	84
6	0	0	0	29	18	73	60
Avg							57.3

Table 4.10 is the confusion matrix of BG for HSO3. An analysis of the HSO3-BG confusion matrix indicates that there are two relatively performant classes in this classifier, namely, HS1 and HS5. A noticeable number of instances in the classes HS2–HS4, are confused with HS1. A large number of instances of HS1, HS2 and HS6 are being confused with HS5. It is very noticeable that not a single instance of HS2 for BG was correctly classified.

In the case of HSO3 it is interesting to note that NBG outperforms BG in each instance and indicates that the removal of the background is beneficial for every hand shape in this HSO.

4.3.4 BG vs. NBG Comparison For HSO4

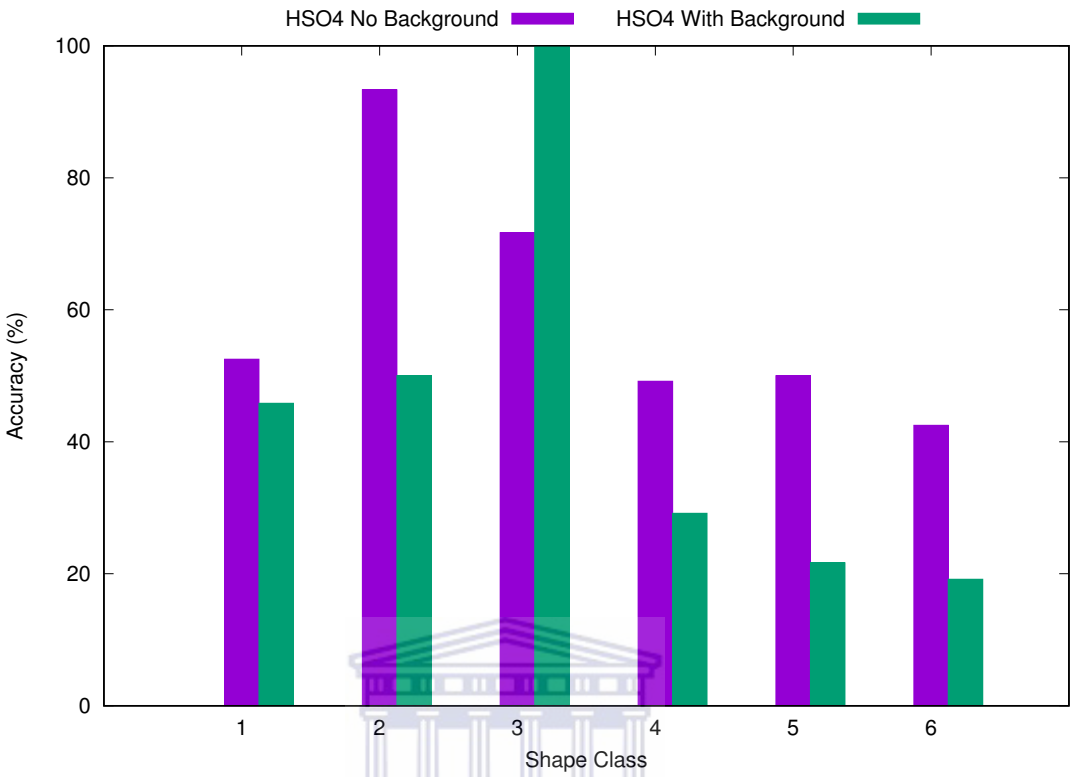


FIGURE 4.12: HSO4 classifier results for BG vs. NBG

Figure 4.12 is a graph of the average accuracies of NBG and BG for each of the shape classes in HSO4. Analysing the graph, it is observed that NBG outperforms BG in every case, save for HS3. There is a noticeable difference of 26% between BG and NBG for class HS3, in which BG outperforms NBG. It is also interesting to observe that BG obtains an accuracy of 100% for HS3. However, this difference is only for this single class and, otherwise, BG provides lacklustre performance for other classes, as compared to NBG. Once again, NBG outperforms BG, in this case with an average accuracy difference of 15%.

TABLE 4.11: Confusion matrix for HSO4 NBG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	63	30	0	15	5	7	52
2	0	112	0	4	4	0	93
3	1	0	86	1	20	12	71
4	22	1	2	59	31	5	49
5	1	1	1	14	60	43	50
6	1	0	10	14	44	51	42
Avg							59.8

Table 4.11 is a confusion matrix for HSO4 with NBG, and when analysed, indicates that several classes in HSO4—HS6, HS4, HS5 and HS1— achieve lower accuracies of 42%, 49%, 50% and 52%, respectively. The confused images that occur for each of these classes are discussed below.

The trend of HS5 and HS6 being confused with each other is observed in the confusion matrix once again. In the case of HS4, it is misclassified as HS1 and HS5 in the majority of cases. For HS1, the majority of misclassified samples are with HS2 and HS4. Apart from HS5 and HS6, generally speaking, it can only be concluded that the classes are randomly confused with each other in this orientation given the large and random distribution of misclassified samples in the confusion matrix. This may be attributed to a large number of visual inter-similarities between images of these classes. Samples of these misclassified images are illustrated in Figure 4.13. It should, however, be noted that all of the average accuracies for the shapes in HSO4 exceed the random guessing accuracy by a factor of about 2.5. Therefore, the classifier is still considered very effective.

(a) HS3.

(b) HS4.

(c) HS5.

(d) HS6.

FIGURE 4.13: Samples of the misclassified images for HSO4: a) Sample images of HS3; b) Sample images of HS4; c) Sample images of HS5; d) Sample images of HS6

TABLE 4.12: Confusion matrix for HSO4 BG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	55	0	58	0	7	0	45
2	0	60	29	31	0	0	50
3	0	0	120	0	0	0	100
4	0	3	81	35	1	0	29
5	3	24	67	0	26	0	21
6	6	0	69	22	0	23	19
Avg							44.3

Table 4.12 is the confusion matrix of BG for HSO4. An analysis of the confusion matrix indicates that the most performant class is HS3 with an accuracy of 100%. Apart from the perfect accuracy obtained for this shape, the classifier has a lacklustre distribution for every other class obtaining accuracies of 50% and below. The results for HS1, HS2, HS4, HS5 and HS6 are relatively poor, it appears that having the background included has a large impact on accuracy. In addition the accuracy reductions may also be attributed to the fact that all shapes except HS3 may appear similar in this orientation. It should

be noted that, whilst these results are relatively poor, each of the classes perform better than the random guessing classification accuracy.

4.3.5 BG vs. NBG Comparison For HSO5

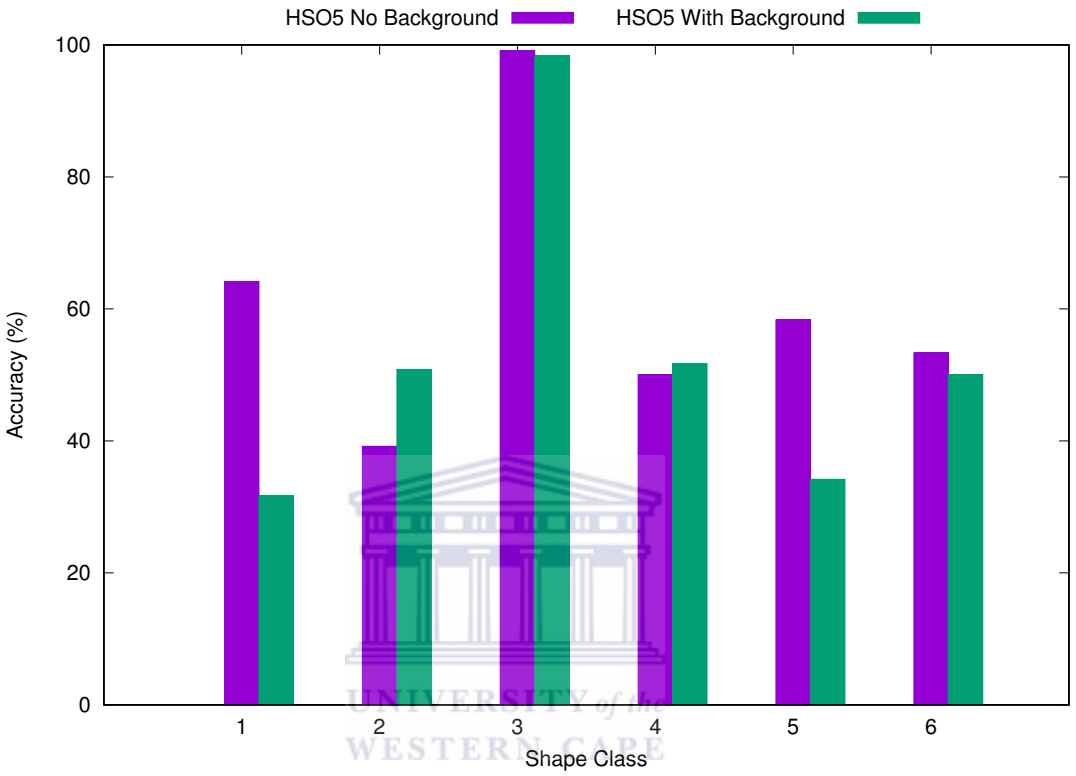


FIGURE 4.14: HSO5 classifier results for BG vs. NBG

Figure 4.14 is a graph of the average differences between NBG and BG for each shape in HSO5. Once again, the graph indicates that NBG generally outperforms BG, in this case for four of the six possible shape classes. However, BG noticeably outperforms the NBG dataset for a single class, i.e., HS2. In terms of average accuracy NBG outperforms BG with a difference of 7% which is equal to 56 images. It is of particular interest to note the near perfect accuracy for both NBG and BG—99% and 98% respectively—for HS3. All other hand shape classes are of a relatively lower accuracy for both NBG and BG. The near-perfect accuracy of HS3 is most likely attributed to the fact that this shape is distinctly dissimilar to all other shapes in its class for this particular orientation. Given the hand is orientated with the side of the hand facing the camera view, other shapes may appear very visually similar to each other, but this shape is distinct.

TABLE 4.13: Confusion matrix for HSO5 NBG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	77	18	0	1	1	23	64
2	41	47	27	4	0	1	39
3	0	0	119	0	0	1	99
4	9	19	3	60	7	22	50
5	24	9	1	1	70	15	58
6	0	6	0	37	13	64	53
Avg							60.6

Table 4.13 is a confusion matrix for HSO5 for NBG. The least performant classes are HS2, HS4, HS6 and HS5 with accuracies of 39%, 50%, 53% and 58% respectively and are discussed below.

The majority of HS2 images are confused with HS1. This could be attributed to the fact that these shapes may appear almost indistinguishable in this orientation in a 2-dimensional plane, even to the human eye. It is especially challenging for a camera at a resolution of 352×288. In some instances HS2 may have samples that appear similar to that of HS3, i.e. more of a triangular shape, as indicated in Figure 4.15.

HS4 is confused with every other class in this orientation, but those confused instances that are most notable occur with HS6 and HS2. The highest confusion count of 22 instances being incorrectly classified can be attributed to the fact that HS4 and HS6 may appear very similar for this particular orientation. The confused instances between HS4 and HS2 amount to a total of 19 instances being incorrectly classified and this may be attributed to the fact that the fingers may appear less bent for HS4 and consequently lead to visual similarities between HS4 and HS2.

HS6 has its most notable confused instances occurring with classes HS5 and HS4. The largest number of confused instances occur with HS4 with a total of 37 instances being incorrectly classified. As described above, the confused instances occurring with HS4 are subject to the same problem in classification error i.e, they may appear visually similar. The second most prominent confusion occurs between HS6 is HS5, which has been observed before.

HS5 has the majority of its incorrectly classified instances occurring with classes HS6 and HS1. The instances that are confused with HS6 are due to erroneous extraction as illustrated in Figure 4.16. The confused instances with HS1 could be attributed to the fact that visual similarities arise due to the scaling of HS1 across the x-axis i.e.,

the stretching of HS1. The confusions between HS5 and HS1 account for 20% of the accuracy loss for this particular orientation. It should be noted that all of the average accuracies for the shapes in HSO5 exceed the random guessing accuracy by a factor of about 2.5. Therefore, the classifier is still considered very effective.

FIGURE 4.15: HSO5 instances of HS2 being confused with HS1

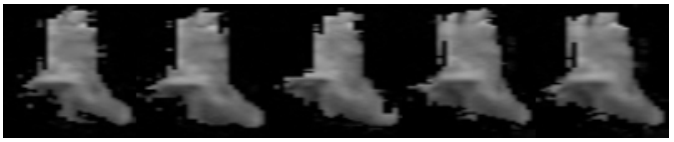


FIGURE 4.16: HSO5 HS5 erroneous extraction

TABLE 4.14: Confusion matrix for HSO5 BG results

	Predicted						
Actual	**1**	**2**	**3**	**4**	**5**	**6**	**Average %**
1	38	71	3	4	4	0	31
2	53	61	0	0	0	6	50
3	0	0	118	0	0	2	98
4	0	34	12	62	0	12	51
5	43	25	0	3	41	8	34
6	30	0	0	30	0	60	50
Avg							52.7

Table 4.14 is the confusion matrix of BG for HSO5. An analysis of the confusion matrix indicates that HS3 has the highest accuracy, which is in accordance with the reasoning described in the NBG analysis i.e. HS3 is distinctly dissimilar to any one of the other shapes in this orientation. It is also interesting to note that in this case only HS1 and HS4 have a small number of instances that are being misclassified as the highest class, HS3. HS1 and HS2 are being misclassified as one another in a relatively large number of instances, which is also similar to NBG, in that these shapes are challenging to distinguish in a 2-dimensional plane. The instances of HS4 and HS5 that are being misclassified as HS2 are attributed to random classification error because these shapes are not visually similar. Despite the relatively varied accuracies, even the lowest accuracy obtained in this classifier is still larger than the random guessing classifier accuracy by a factor of 1.9.

For this HSO classifier BG is more performant than NBG for HS2 with a difference of 11%. This is attributed to the erroneous extraction of this particular hand shape as

indicated in Figure 4.15. Thus, it appears that having more features available allows for the classifier with BG to be more performant for HS2.

4.3.6 BG vs. NBG Comparison For OR

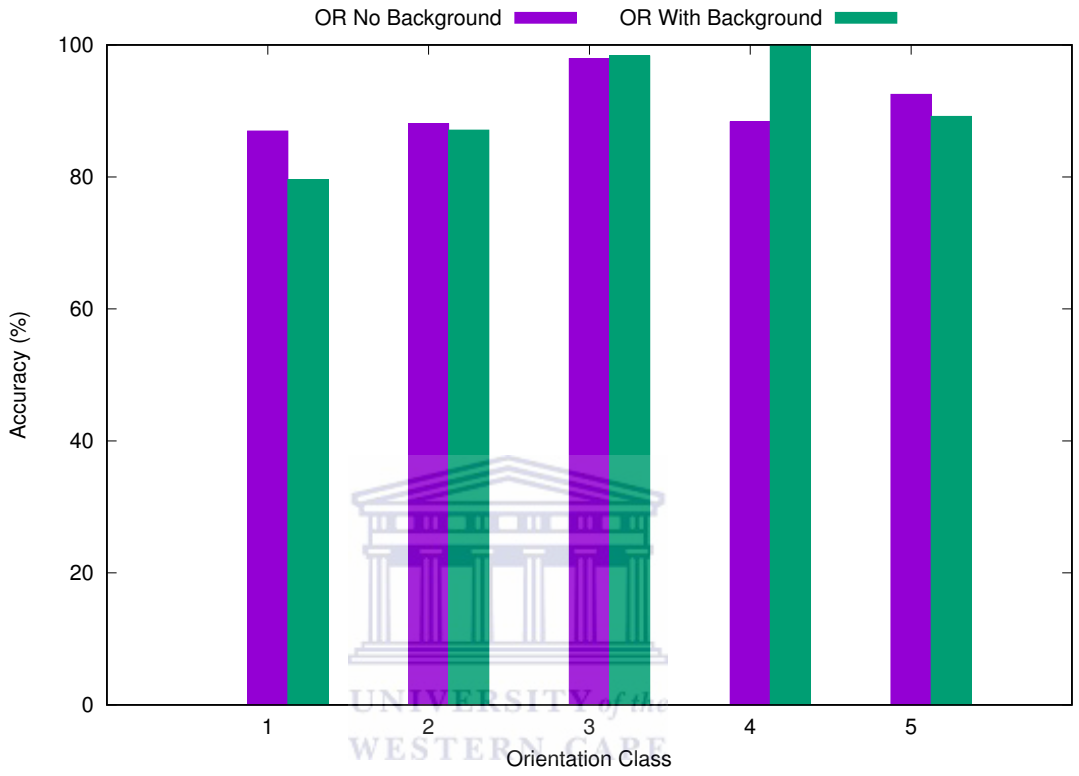


FIGURE 4.17: OR classifier results for BG vs. NBG

The Figure 4.17 is a graph of the average accuracy difference between NBG and BG for the OR classifier. For this classifier, on average, BG outperforms NBG, albeit by a very small difference of 0.06%. On closer observation, however, NBG outperforms BG in three of the five classes. It is only noticeably outperformed by BG for one class—O4—with a difference of 11.5%. For O3, they can be considered on a par. Therefore, as confirmed in the previous classifiers, NBG outperforms BG in general. This indicates that pre-processing frames to remove the background is advantageous to recognition.

TABLE 4.15: Confusion matrix for OR NBG results

	Predicted					
Actual	**1**	**2**	**3**	**4**	**5**	**Average %**
1	626	7	7	3	77	86.9
2	34	634	32	10	10	88.0
3	7	8	705	0	0	97.9
4	0	51	27	636	6	88.3
5	45	7	0	2	666	92.5
Avg						90.7

Table 4.15 is a confusion matrix for the OR classifier for NBG. Although all of the classes clearly perform at a very high accuracy, a further analysis is carried out to determine causes for the gaps in classification for O1 with an accuracy of 86.9%, which is slightly lower in accuracy compared to the other orientations.

In most misclassified cases, O1 is confused with O5. Noting that images of O1–O5 represent images of hand shapes HS1–HS6 in each respective orientation, a large number of the incorrectly predicted images of O1 could be attributed to the similarities based on the camera perspective. This is evidenced by the fact that the confused instances for both O1 and O5 occur relative to one another.

FIGURE 4.18: HS3 instances in O1 that were being misclassified as O5

TABLE 4.16: Confusion matrix for OR BG results

	Predicted					
Actual	**1**	**2**	**3**	**4**	**5**	**Average %**
1	573	9	17	32	89	79.5
2	0	627	15	78	0	87.0
3	11	0	708	1	0	98.3
4	0	1	0	719	0	99.8
5	78	0	0	0	642	89.1
Avg						90.8

Table 4.16 is the confusion matrix of BG for OR. An analysis of this confusion matrix indicates that the most performant class is O4 which only has a single misclassified instance. The largest number of confused instances occur between O1 and O5. It should be noted that O1 has 6 shape classes, each of which may be confused with any of the

shape classes in O5. Thus, even the lowest accuracy of 79% is encouraging. It should also be noted that these confused instances of O1 are similar in number to that of NBG.

For the OR classifier, BG outperforms NBG by a noticeable amount for O4. In this orientation the misclassified samples of HS1, HS2, HS3 and HS5 in O4 had holes which lead to difficulty or confusion in the classification procedure.

4.4 Experiment To Assess CNNs By Comparing Activation Functions

The goal of this experiment was to determine whether using an alternate activation function provides a better CNN accuracy. Each of the HSO classifiers and the OR classifier was trained, whilst each time altering the activation function used in the CNN—the resulting accuracies were then compared. A number of activation functions such as the ELU, ReLU, LeakyReLU, TanH, and Sigmoid exist. These activation functions may provide varied classification accuracies in the current context. For ease of reference, the terms "activation function" and "activation" may be used inter-changeably. To limit the number of comparisons and compute time, henceforth, only NBG is used to train the models.

TABLE 4.17: Average accuracies for classifiers across activations

Activation Function	**Classifier**					
	HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
ELU	90.2	70.1	85.8	58.6	62.6	91.8
HardTanh	89.3	76.3	87.0	65.0	60.9	91.3
LeakyReLU	89.1	70.4	85.4	57.3	64.4	92.2
Tanh	88.1	75.5	85.5	58.6	60.0	91.8
PReLU	89.4	68.8	86.2	59.0	62.6	90.6
ReLU	88.3	71.6	86.6	57.6	67.2	92.4
ReLU6	91.1	69.8	85.2	57.5	62.6	92.1
RReLU	88.6	70.9	85.6	58.3	65.9	91.0
Sigmoid	74.3	69.5	77.9	48.0	51.8	88.8
LogSigmoid	71.5	62.7	73.3	42.5	50.2	85.1

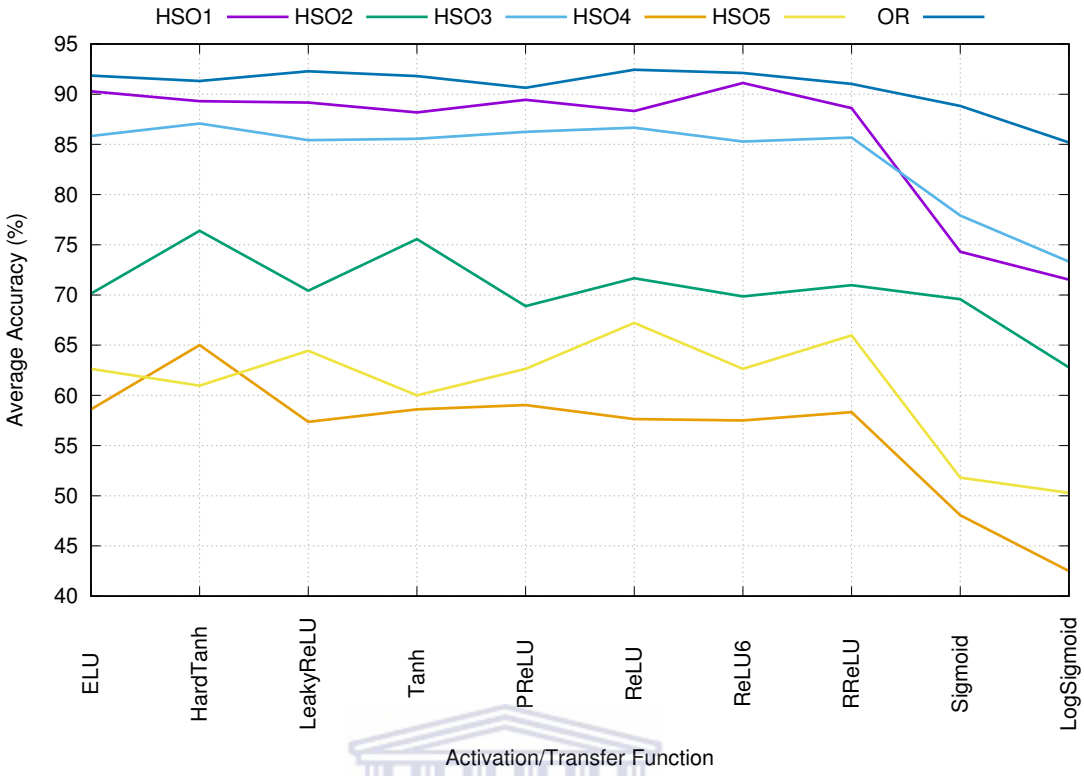


FIGURE 4.19: Average Accuracies Across Activation Functions for All Classifiers

Table 4.17 summarises the average accuracies obtained for each activation for every HSO classifier and the OR classifier. Each column in the table refers to a single classifier and each row refers to an activation function. The best activation function for each column, i.e. each classifier, is highlighted in red. An analysis of the table indicates an immediate trend across each HSO classifier and the OR classifier. The trend observed is that the Sigmoid and LogSigmoid functions, the last two rows, consistently perform at a lower accuracy for every classifier when compared to every other activation function. The Figure 4.19 is a graphical representation of the table. Each line in the graph represents a single classifier, the goal is to compare activation functions for a particular classifier. Each column in the graph is a comparison of the average accuracy obtained by a single activation function for the respective classifier. A clear dip is observed in the line graphs for the LogSigmoid and Sigmoid functions which confirms this point.

Sigmoidal functions that are symmetric about the origin, such as TanH, result in values in the range $[-1, 1]$. According to LeCun in [74], these functions are preferred because they are more likely to produce outputs that are, on average, closer to zero and may also include negative values. This is in contrast to the sigmoid function that produces values in the range $[0, 1]$, which always results in a positive output, as well as a positive mean. Due to the range of the sigmoid function, it takes longer for the sigmoid activation

function to converge, and thus, makes it less performant when compared to symmetric sigmoidal and rectified linear functions, hence the observation in the results.

This trend is confirmed by the observation that the most performant activation function for three of the six classifiers—HSO2, HSO3 and HSO4—is the symmetric sigmoidal function, HardTanh. A problem that arises when using symmetric sigmoidal activation functions is that they suffer from the vanishing gradient problem which may lead to slow optimisation convergence and in some instances, lead to a poor local minimum. Rectified linear activation functions—all the activation functions that contain ReLU in its name—attempt to address many of the issues that arise from sigmoidal functions as described in [75] but they do, however, have trouble with non-activating Rectified Linear Units (ReLU) in some instances. Therefore, it is also observed that the rectified linear activation functions, ReLU and ReLU6, also obtain higher relative accuracies for three of the six classifiers—HSO1, HSO5 and OR—and obtain higher accuracies than the symmetric sigmoidal function classifiers.

TABLE 4.18: Range in average accuracy for each classifier excluding the LogSigmoid and Sigmoid activation functions

	Average Accuracy of Classifier (%)					
	HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
Min	88	68	85	57	60	90
Max	91	76	87	65	67	92
Range	**3**	**8**	**2**	**8**	**7**	**2**

Another observation made from the graph is that, apart from the Sigmoid and LogSigmoid activations, other activations appear to perform very similarly, with relatively comparable accuracies for every classifier, whilst some of these activation functions do not perform similarly but have noticeable peaks and dips. Excluding the LogSigmoid and Sigmoid activations, the Table 4.18 shows the difference in accuracy for the least performant and most performant activation function, for each of the classifiers. It is observed that the range is relatively low, between 2% and 3% for HSO1, HSO3 and HSO5 whilst relatively higher ranges of 7% and 8% are obtained for HSO2, HSO4 and HSO5. It is evident from this table that comparing activation functions is beneficial and in some cases offers a reasonable increase in accuracy.

It should be noted that analyses of the confusion matrices of each classifier in this experiment indicate a very similar trend for causes of confused instances to the analyses discussed in Sections 4.3.1–4.3.6 for the previous experiment. Therefore, the confusion matrices and their in-depth analyses for this and other comparative experiments in Sections 4.5 and 4.6 are omitted from the main discussion, but are provided in Appendix C for the interested reader. The following subsections provide further analysis of the average accuracy of each CNN for each activation function.

4.4.1 Activation Function Analysis For HSO1

TABLE 4.19: Average accuracy for HSO1 classifiers across activations

	Shapes						
	1	**2**	**3**	**4**	**5**	**6**	**Average (%)**
ELU	99	100	70	96	94	81	90.2
HardTanh	98	100	80	89	99	69	89.3
LeakyReLU	96	98	70	95	95	78	89.1
Tanh	94	99	85	85	98	66	88.1
PReLU	95	100	70	97	94	78	89.4
ReLU	97	98	70	97	90	75	88.3
ReLU6	98	100	70	96	94	87	91.1
RReLU	95	96	71	97	94	76	88.6
Sigmoid	85	67	75	70	90	57	74.3
LogSigmoid	74	64	69	75	88	58	71.5

Table 4.19 tabulates the accuracies obtained for each shape orientated in O1 for each respective activation function used. It should be noted that each row represents a separate HSO1 classifier using the specified activation function—in the left-most column—and that the right-most column of the table is the average accuracy across all six hand shapes. Each of the other other columns represents the hand shapes across activation functions. The comparisons in the table are done across rows. An analysis of the table indicates that, even at the class-level, the Sigmoid and LogSigmoid activation function perform at a lower level for all but a single class, HS3. Excluding these functions, the other activations perform reasonably consistently and obtain very high accuracies. In this specific orientation the classifier is capable of consistently discerning between hand shapes with a very high accuracy.

4.4.2 Activation Function Analysis For HSO2

TABLE 4.20: Average accuracy for HSO2 classifiers across activations

	Shapes						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
ELU	78	62	87	54	87	50	70.1
HardTanh	85	81	89	62	88	50	76.3
LeakyReLU	77	68	88	50	85	52	70.4
Tanh	85	80	89	60	87	50	75.5
PReLU	78	60	86	50	88	48	68.8
ReLU	81	68	85	48	86	59	71.6
ReLU6	80	69	85	47	85	50	69.8
RReLU	80	70	85	49	88	51	70.9
Sigmoid	83	81	77	40	85	50	69.5
LogSigmoid	77	64	78	41	72	42	62.7

Table 4.20 shows the accuracies obtained for each shape orientated in O2 for each respective activation function. It should be noted, that once again, the comparison for this table is carried out across rows and will be the case for each subsequent table in this section. An analysis of the table indicates that each of the activation functions perform relatively well with respect to one another except for a single instance in which the Sigmoid and LogSigmoid functions perform relatively poorly, namely, HS4. Excluding the Sigmoid and LogSigmoid functions, it is interesting to note that in HS2 and HS4 the HardTanH and TanH activations perform noticeably better than any of the other activations in this class with a range of 15% and 21% respectively. However, this anomaly is difficult to explain.

4.4.3 Activation Function Analysis For HSO3

TABLE 4.21: Average accuracy for HSO3 classifiers across activations

	Shapes						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
ELU	97	75	86	88	95	71	85.8
HardTanh	97	78	91	90	99	65	87.0
LeakyReLU	100	75	86	84	95	70	85.4
Tanh	96	80	87	87	99	62	85.5
PReLU	97	75	86	90	99	68	86.2
ReLU	99	77	88	88	95	70	86.6
ReLU6	99	75	86	84	95	70	85.2
RReLU	99	75	85	85	95	73	85.6
Sigmoid	88	65	86	90	93	43	77.9
LogSigmoid	95	38	89	77	81	58	73.3

Table 4.21 shows the accuracies obtained for each shape orientated in O3 for each activation function. An analysis of the table indicates that the Sigmoid and LogSigmoid activation functions perform relatively poorly with respect to each of the other activation functions. The Sigmoid function performs reasonably well for HS4 and HS5 whilst the LogSigmoid function performs poorly for all but a single class, HS3. Excluding the Sigmoid and LogSigmoid functions, each of the other activation functions perform reasonably consistently across shapes except for HardTanH and TanH in HS6, which performs at a lower level in this instance. Thus, it is shown that the activations perform reasonably consistently.

4.4.4 Activation Function Analysis For HSO4

TABLE 4.22: Average accuracy for HSO4 classifiers across activations

	Shapes						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
ELU	61	75	66	49	67	30	58.6
HardTanh	65	95	70	57	63	39	65.0
LeakyReLU	50	64	80	38	76	34	57.3
Tanh	61	91	68	45	50	35	58.6
PReLU	60	76	77	45	68	26	59.0
ReLU	46	65	80	40	84	28	57.6
ReLU6	45	65	75	49	75	34	57.5
RReLU	50	70	78	40	81	29	58.3
Sigmoid	39	59	59	41	51	37	48.0
LogSigmoid	39	60	56	31	38	29	42.5

Table 4.22 shows the accuracies obtained for each shape orientated in O4 for each activation function. An analysis of the table indicates that HS6 is the lowest performing class across every activation function. However, relatively encouraging accuracies are obtained for each of the other shape classes. The LogSigmoid function performs consistently poorly relative to any of the other non-sigmoid activation functions. The Sigmoid function also performs poorly, but does however, obtain better relative accuracies in class HS6 and HS4. Overall, the most consistently performant activation function was the HardTanH activation.

4.4.5 Activation Function Analysis For HSO5

TABLE 4.23: Average accuracy for HSO5 classifiers across activations

	Shapes						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
ELU	57	33	97	50	82	54	62.6
HardTanh	63	30	99	50	69	53	60.9
LeakyReLU	61	32	90	63	86	51	64.4
Tanh	59	35	99	52	63	50	60.0
PReLU	58	45	94	57	68	52	62.6
ReLU	58	41	95	59	90	58	67.2
ReLU6	49	31	92	59	85	58	62.6
RReLU	59	41	91	55	89	58	65.9
Sigmoid	61	20	99	54	33	41	51.8
LogSigmoid	10	21	93	38	79	58	50.2

Table 4.23 shows the accuracies obtained for each shape orientated in O5 for each activation function. An analysis of the table shows that the LogSigmoid function performs poorly for HS1, HS2 and HS4. The Sigmoid activation function performs poorly for the shape classes HS2, HS5 and HS6 with good relative accuracies in the rest of the classes. The other activation functions perform reasonably consistently for HS1, HS3, HS4, HS5 and HS6.

4.4.6 Activation Function Analysis For OR

TABLE 4.24: Average accuracy for OR classifiers across activations

	Orientation					
	1	**2**	**3**	**4**	**5**	**Average %**
ELU	86.6	92.3	97.0	92.7	90.4	91.8
HardTanh	89.0	90.6	97.9	87.9	90.9	91.3
LeakyReLU	86.2	95.0	94.3	95.5	90.2	92.2
Tanh	87.6	91.1	97.6	90.1	92.5	91.8
PReLU	87.2	91.2	94.8	90.5	89.3	90.6
ReLU	85.9	95.4	95.1	96.1	89.5	92.4
ReLU6	86.1	95.0	94.5	94.8	90.0	92.1
RReLU	84.0	93.6	94.1	93.3	90.0	91.0
Sigmoid	88.1	85.6	96.9	83.3	90.0	88.8
LogSigmoid	75.0	84.7	91.3	84.5	90.2	85.1

Table 4.24 shows the accuracies obtained for each orientation in OR for each activation function. An analysis of the table indicates that the LogSigmoid activation function performs relatively well for O5 and O3, but does however, have a sub-par performance when compared to the other activation functions for O1, O2 and O4. Similarly, the Sigmoid activation function also performs well for O5 and O3 but obtains sub-par accuracies for O2 and O4 in comparison to the other activations. The other activation functions perform consistently. This classifier performs excellently with no orientation dropping below 80%.

The lower accuracies of Sigmoid and LogSigmoid were discussed in the analysis of the summarised table at the start of this chapter. It appears that even at the class level, Sigmoid and LogSigmoid perform at a lower level in comparison to each of the other activation functions, though in a small number of instances it performs comparably to the rest of the activations at a class level.

4.5 Experiment To Assess CNNs By Adjusting Learnable Filter And Fully Connected Layer Node Counts

The goal of this experiment is to inspect the impact that the number of learnt filters at each convolutional layer has on the CNNs accuracy. In addition, a variable number of nodes in the fully connected layer were also added for each learnable filter configuration.

A number of filters are learnt at each convolution layer. This number is variable and may increase or decrease accuracy. The number of nodes in the fully connected layer may also have an impact on accuracy.

One of the key aims for this research is the application of CNNs for hand orientation recognition and hand shape recognition in multiple orientations and in an embedded or mobile phone context. Mobile devices have limited resources. The number of convolutions, storage and memory increase proportionally to the number of filters learnt and the number of fixed layer nodes. Thus, to remain feasible, the number of learnt filters need to be restricted to a maximum of 40 feature maps per layer in an ES context. Therefore, henceforth, this experiment and subsequent experiments will consider two sets of CNNs: one set of CNNs for mobile devices that limit the number of filter maps learnt per convolution layer—the ES context; and another set of CNNs that have no limit—the non-resource starved (NRS) context.

Each of the HSO classifiers and the OR classifier was trained, whilst each time altering the number of nodes in the linear layer as well as the number of features learnt for each convolution layer; C1 and C2. The learnable filter count refers to C1 and C2 respectively e.g. 160 320 refers to C1 = 160 filters and C2 = 320 filters, 160 80 refers to C1 = 160 filters and C2 = 80 filters, and so forth. To limit the number of comparisons carried out, henceforth, only the most performant activation function for each classifier from the previous experiment is used to train these models.

The format of the tables in this experiment is different due to the 2-dimensional nature of the parameters that are being compared, namely, the learnable filter count and the fully connected layer node count. In this overall discussion, the configurations for the best performing networks and their respective parameters are tabulated in Table 4.25 and compared in an NRS context and ES context.

TABLE 4.25: Accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer—Best configurations

		Classifiers					
		HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
NRS	**Learnable Filters**	160 320	320 160	160 80	320 160	320 320	320 320
	Linear Layer Nodes	500	200	600	300	200	500
	Accuracy (%)	90.9	80.8	91.5	65.1	71.1	92.7
ES	**Learnable Filters**	20 40	20 20	20 20	40 20	20 20	20 40
	Linear Layer Nodes	600	300	200	600	400	400
	Accuracy (%)	90.0	79.7	90.0	61.8	65.5	92.3

It should be noted that each of the accuracies in the NRS context had configurations that learnt over at least 80 filters per convolution layer. Referring to the NRS section of the Table 4.25, these configurations obtained relatively higher accuracies for HSO1-HSO5 and OR than those of the corresponding ES context configurations.

Observing the number of filters, it is interesting to note that the largest number of filters, i.e. 320 320 only provides the best accuracy for two of the six classifiers—HSO5 and OR in the NRS context. For five of the six classifiers in the table, learning 160 filters or more provides a better accuracy. However, generally speaking, a larger number of filters does appear to provide for a better accuracy as evidenced by the fact that not a single configuration below 80 filters outperforms these larger configurations.

On the other hand, the number of fully connected nodes that provides the best accuracy varies across classifiers and there seems to be no correlation between the number of nodes in the fully connected layer and a good choice for the number of learnable filters.

Referring to the ES section of the table, these results take the device limitations into account by limiting the number of learnable filters in each layer to a maximum of 40 filters. Regarding the best number of filters, it is interesting to observe that none of the classifiers in the ES context use the largest learnable filter count available, i.e. 40 40, to obtain the best accuracy and can benefit when that at least one of the layers learns 20 filters per feature map. In this case, a larger number of filters does not necessarily provide the best accuracy. Similar to the NRS context, the ES context also seems to have no correlation between the number of fully connected layer nodes and the learnable filters.

This is very interesting to note that despite applying the limitation of learning only a maximum of 40 filters in the ES context, a comparison of each of the average accuracies obtained for each classifier and configuration indicates that the ES context classifiers perform at accuracies very comparable to the NRS context classifiers. It is evident that for this specific problem and number of classes, i.e. five orientations and six shapes, learning 20 or 40 filters is sufficient for the model to generalise with very reasonable accuracies. The number of fully connected layer nodes varies and may need to be adjusted on a case by case basis, although a deeper analysis on a classifier-specific basis will provide further insight into this.

The following subsections provide further analysis of the learnable filter and fully connected layer node counts for each HSO classifier and the OR classifier. Configurations in the following section will be referred to using the following format: [LC1]-[LC2]-[LIN], where LC is the learnable count learnt at a specific layer depth and 1 is the first convolution layer, 2 is the second convolution layer and LIN refers to the number of nodes in the fully connected layer, e.g. 20-40-500 refers to 20 filters learnt at the first convolution layer, 40 filters in the second convolution layer and 500 nodes in the fully connected layer. In each case the best performing network in both the NRS context and ES context are highlighted in red. The tabulation of results at the individual class level for each configuration is not feasible given the large dimensionality of the results. The confusion matrices for the most performant configurations are provided in Appendix C.2.

4.5.1 Learnable Filter And Fully Connected Layer Node Count Analysis For HSO1

TABLE 4.26: HSO1 accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer

		Fully Connected Nodes Count				
		200	**300**	**400**	**500**	**600**
Learnable Filter Count	**20 20**	88.8	86.2	88.4	88.6	86.2
	20 40	85.9	89.7	87.3	89.4	90.0
	40 20	89.7	86.9	89.3	87.5	86.2
	40 40	87.5	89.7	88.4	86.6	87.6
	40 80	88.1	87.7	88.0	89.0	89.8
	80 40	87.0	89.0	88.0	89.1	88.1
	80 80	88.8	87.7	90.0	88.0	88.8
	80 160	89.1	90.4	89.5	88.4	90.8
	160 80	89.7	89.0	89.3	89.8	89.7
	160 160	87.7	90.1	88.0	89.4	88.7
	160 320	88.7	86.6	90.2	90.9	89.1
	320 160	89.0	89.7	89.1	90.0	89.0
	320 320	90.2	90.2	89.3	90.2	90.2

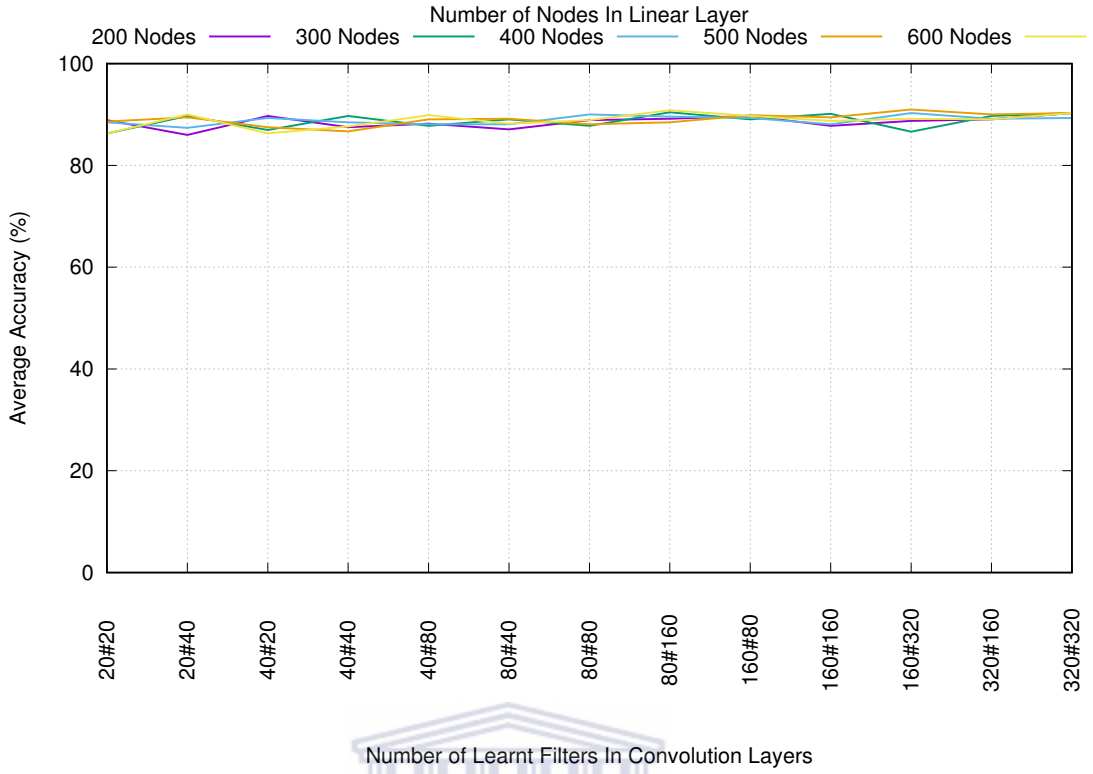


FIGURE 4.20: HSO1 number of learnable filters learnt and number of nodes in the fully connected layer

Table 4.26 tabulates the average accuracies for each of the filter and node configurations for HSO1. An analysis of the table does not indicate any strong outliers with respect to the configurations and also shows that, despite the highest accuracy being obtained by the configuration 160-320-500, a similar accuracy could be obtained with less computation using a configuration of 20-40-500 or 20-40-600, as observed previously. The Figure 4.20 is a graphical representation of the table and does not depict any strong positive trend for learning a number of filters exceeding 40. There appears to be no large difference in accuracies for the different node configurations. Thus, in this instance we can conclude that a larger number of nodes in the fully connected layer does not necessarily provide any benefit to accuracy.

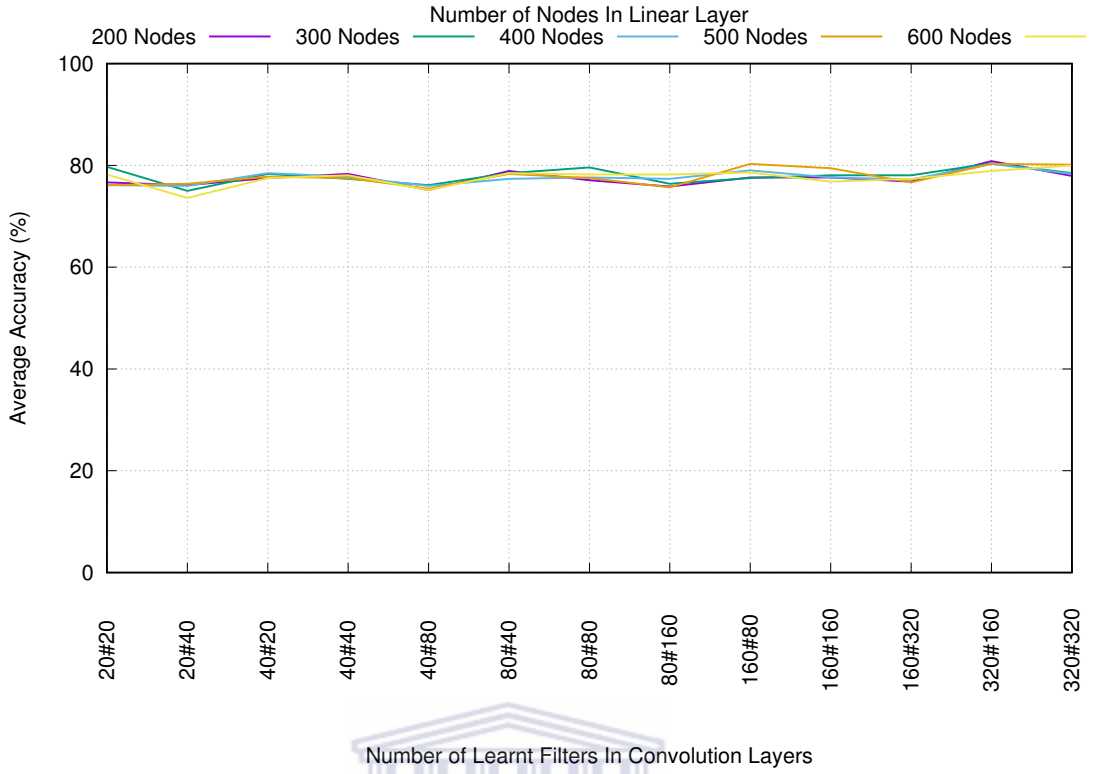


FIGURE 4.21: HSO2 number of learnable filters learnt and number of nodes in the fully connected layer

4.5.2 Learnable Filter And Fully Connected Layer Node Count Analysis For HSO2

TABLE 4.27: HSO2 accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer

		Fully Connected Nodes Count				
		200	**300**	**400**	**500**	**600**
Learnable Filter Count	**20 20**	76.6	79.7	76.1	76.1	78.1
	20 40	76.1	75.0	75.9	76.3	73.6
	40 20	77.5	78.3	78.4	77.7	77.5
	40 40	78.3	77.3	77.7	77.5	78.0
	40 80	75.1	76.1	75.9	75.4	75.1
	80 40	78.8	78.4	77.3	78.3	78.4
	80 80	77.0	79.5	77.6	77.5	78.1
	80 160	75.8	76.3	77.3	75.6	78.1
	160 80	77.6	77.5	79.0	80.2	78.6
	160 160	77.6	78.0	77.6	79.4	76.8
	160 320	76.8	78.0	77.2	76.6	77.3
	320 160	80.8	80.4	80.2	80.2	78.8
	320 320	77.9	78.4	78.3	80.1	80.0

Table 4.27 tabulates the average accuracies for each of the filter and node configurations in HSO2. Similar to HSO1, the analysis for the HSO2 table does not indicate any strong outliers. The largest difference between accuracies in all of the configurations is at most 7%. The Figure 4.21 is a graphical representation of the table and further illustrates the point, as in HSO1, that there is no proportional increase in accuracy as a larger number of filters are learnt. Rather, the accuracies appear to be randomly distributed.

4.5.3 Learnable Filter And Fully Connected Layer Node Count Analysis For HSO3

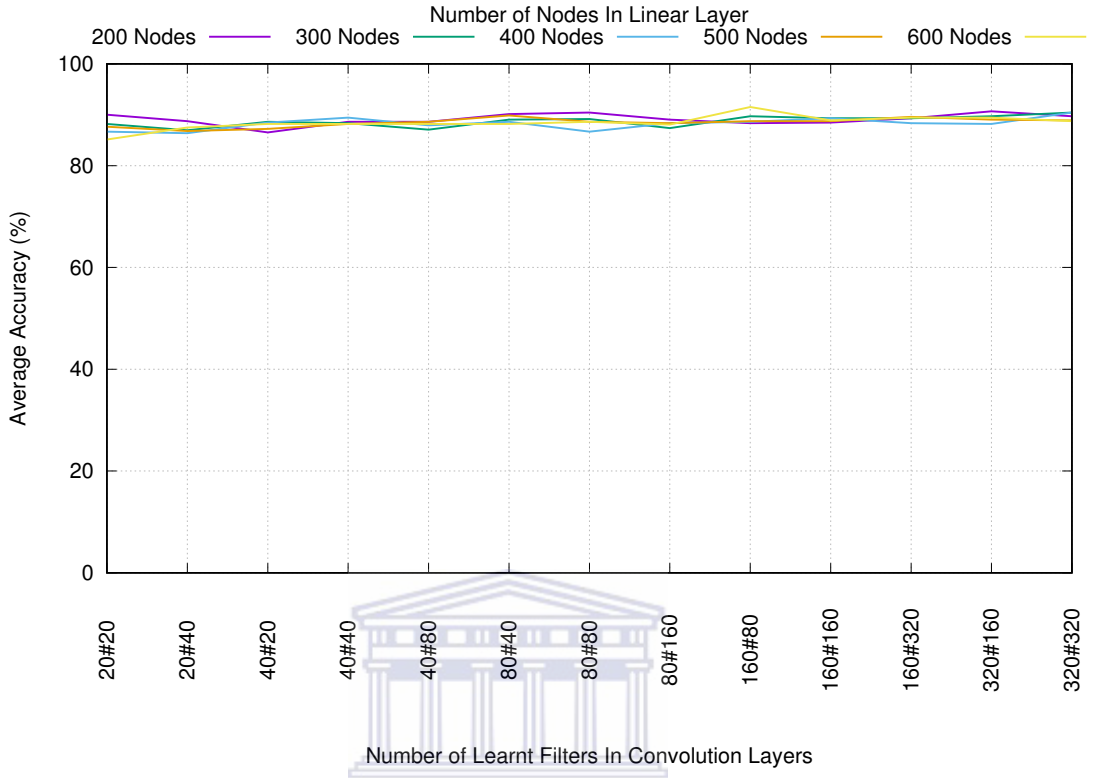


FIGURE 4.22: HSO3 number of learnable filters learnt and number of nodes in the fully connected layer

TABLE 4.28: HSO3 accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer

		Fully Connected Nodes Count				
		200	**300**	**400**	**500**	**600**
Learnable Filter Count	**20 20**	90.0	88.1	86.6	87.6	85.1
	20 40	88.7	86.9	86.3	86.8	87.5
	40 20	86.5	88.6	88.4	87.2	88.1
	40 40	88.6	88.3	89.4	88.1	88.6
	40 80	88.6	87.0	87.9	88.6	88.1
	80 40	90.1	89.0	88.6	89.8	88.1
	80 80	90.4	89.1	86.6	88.6	88.6
	80 160	89.0	87.3	88.3	88.3	88.0
	160 80	88.3	89.7	88.6	88.7	91.5
	160 160	88.4	89.3	89.3	88.7	88.8
	160 320	89.3	89.3	88.3	89.5	89.4
	320 160	90.6	89.7	88.1	89.0	89.4
	320 320	89.7	90.4	90.4	88.8	88.7

Table 4.28 tabulates the average accuracies for each of the filter and node configurations in HSO3. An analysis of the table indicates that there are also no strong outliers with

the difference between the largest value and smallest value being at most 6%. The Figure 4.22 is a graphical representation of the table and indicates no trend towards a proportional increase in accuracy relating to the learnt filter configuration.

4.5.4 Learnable Filter And Fully Connected Layer Node Count Analysis For HSO4

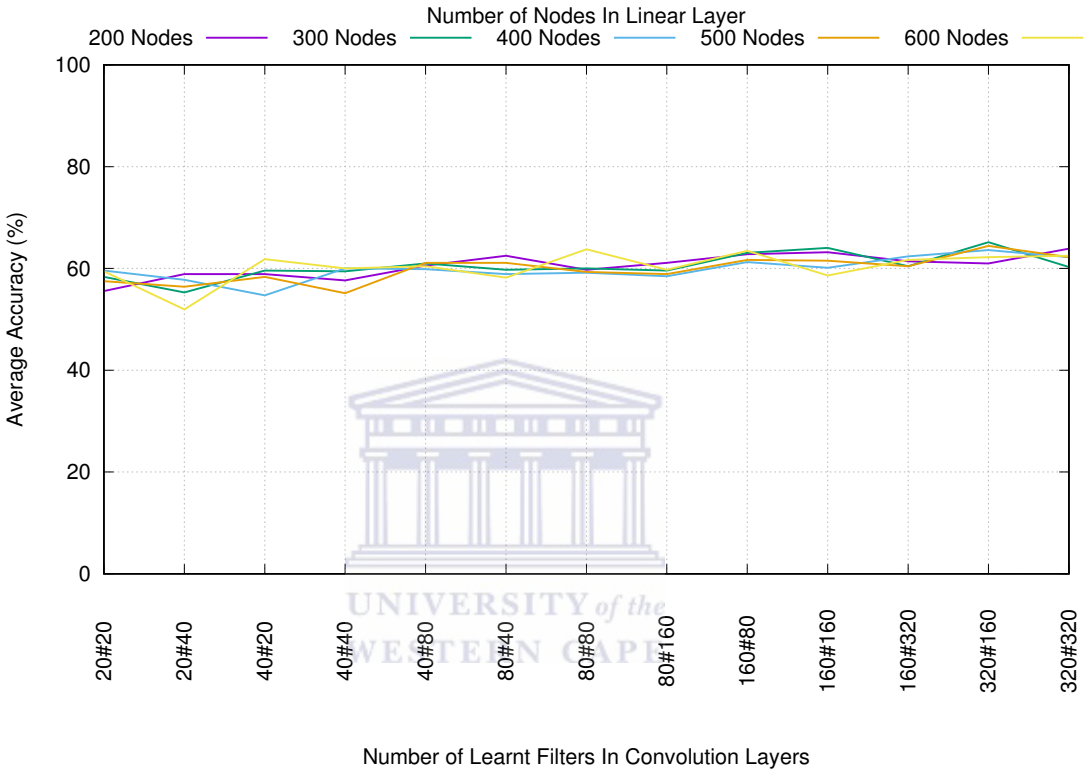


FIGURE 4.23: HSO4 number of learnable filters learnt and number of nodes in the fully connected layer

Table 4.29 tabulates the average accuracies for each of the filter and node configurations in HSO4. The Figure 4.23 is a graphical representation of the table. The graph is slightly more erratic than any of the previous classifiers and indicates that there is a very slight increasing trend in accuracy for each linear node configuration when more learnable filters are added, although it does not do so uniformly.

TABLE 4.29: HSO4 accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer

		Fully Connected Nodes Count				
		200	**300**	**400**	**500**	**600**
Learnable Filter Count	**20 20**	55.5	58.3	59.5	57.5	59.4
	20 40	58.8	55.2	57.7	56.3	51.9
	40 20	58.8	59.5	54.7	58.3	61.8
	40 40	57.6	59.4	60.0	55.1	60.0
	40 80	60.5	60.9	59.8	61.1	60.4
	80 40	62.5	59.7	58.8	61.1	58.1
	80 80	59.7	60.0	59.1	59.3	63.7
	80 160	61.1	59.5	58.4	58.8	59.7
	160 80	62.7	63.0	61.2	61.6	63.4
	160 160	63.1	64.0	60.1	61.5	58.6
	160 320	61.3	60.4	62.3	60.4	61.6
	320 160	60.9	65.1	63.6	64.4	62.2
	320 320	63.8	60.2	62.2	62.2	62.5

4.5.5 Learnable Filter And Fully Connected Layer Node Count Analysis For HSO5

TABLE 4.30: HSO5 accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer

		Fully Connected Nodes Count				
		200	**300**	**400**	**500**	**600**
Learnable Filter Count	**20 20**	59.1	59.1	65.5	55.9	61.3
	20 40	63.4	60.9	62.6	58.6	59.5
	40 20	61.5	54.8	65.2	63.4	64.7
	40 40	59.8	58.4	63.6	56.8	62.7
	40 80	65.1	61.2	58.4	64.4	58.3
	80 40	63.1	65.0	62.3	59.8	56.3
	80 80	60.4	66.9	65.9	63.8	61.5
	80 160	62.3	67.5	61.8	65.0	63.6
	160 80	62.2	60.6	63.1	65.4	62.3
	160 160	65.9	63.0	65.0	64.8	63.4
	160 320	65.0	66.1	63.4	68.1	66.1
	320 160	59.3	65.6	64.3	63.7	65.2
	320 320	71.1	65.4	65.8	67.2	64.3

Table 4.30 tabulates the average accuracies for each of the filter and node configurations in HSO5. An analysis of the table does not indicate any strong outliers nor an increase

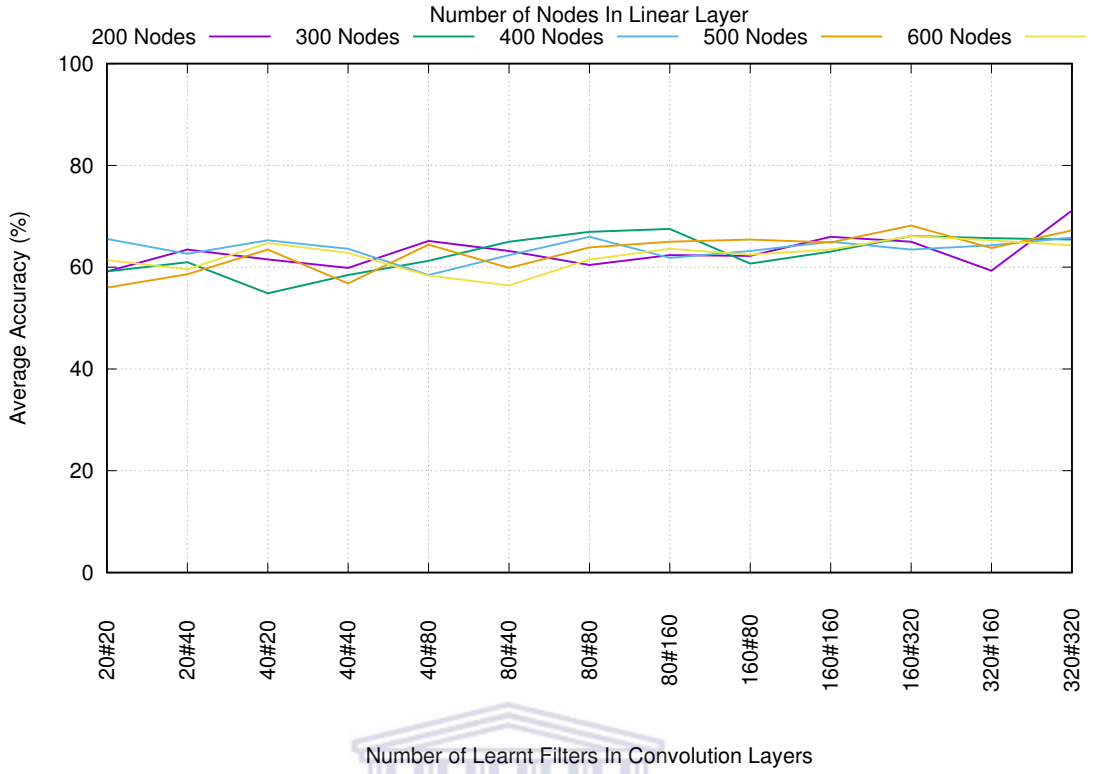


FIGURE 4.24: HSO5 number of learnable filters learnt and number of nodes in the fully connected layer

in accuracy as more learnable filters are added. The Figure 4.24 is a graphical representation of the table and illustrates the random distribution of the accuracies. The graph indicates that for HSO5, the accuracies are more erratic than any of the previous classifiers.

4.5.6 Learnable Filter And Fully Connected Layer Node Count Analysis For OR

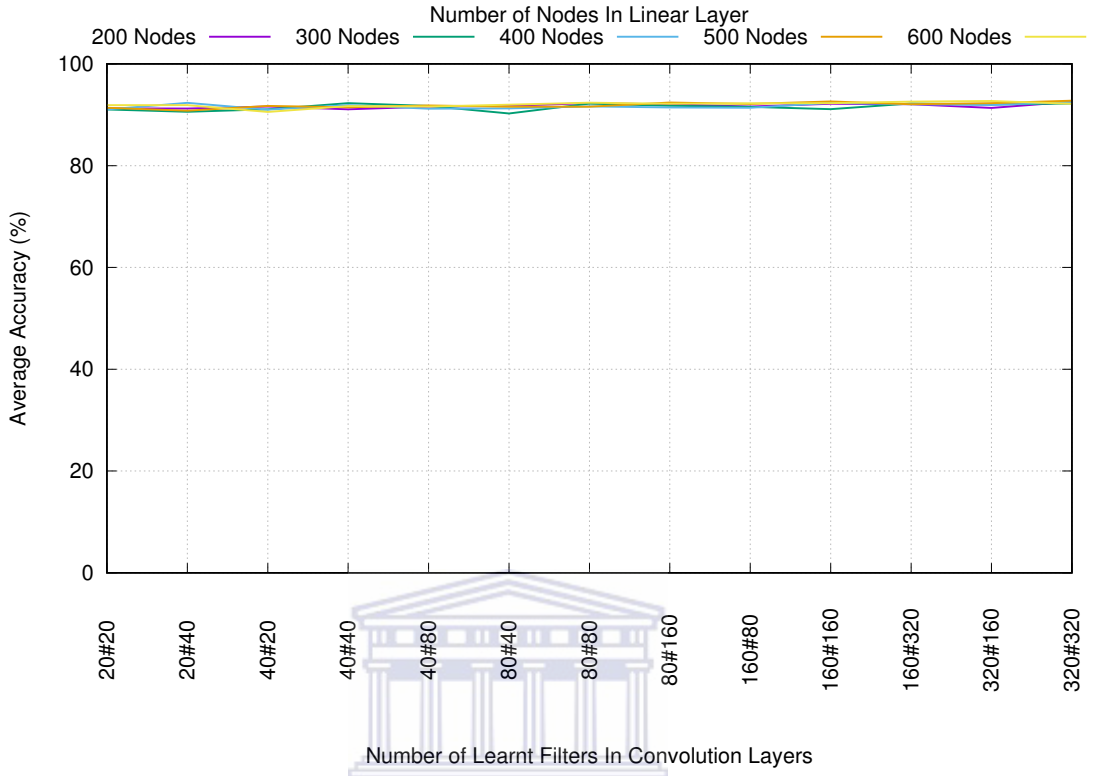


FIGURE 4.25: OR number of learnable filters learnt and number of nodes in the fully connected layer

TABLE 4.31: OR accuracies for a variable number of learnable filters and variable number of nodes in the fully connected layer

		Fully Connected Nodes Count				
		200	**300**	**400**	**500**	**600**
Learnable Filter Count	**20 20**	91.2	91.0	91.0	91.3	91.8
	20 40	91.2	90.5	92.3	90.8	91.8
	40 20	91.6	91.0	91.0	91.7	90.5
	40 40	92.0	92.2	91.8	91.4	91.6
	40 80	91.5	91.7	91.1	91.8	91.6
	80 40	91.7	90.2	91.2	91.5	91.9
	80 80	92.2	92.1	91.6	91.5	92.3
	80 160	92.0	91.7	91.4	92.3	92.1
	160 80	91.6	91.5	91.3	92.1	92.2
	160 160	92.1	92.1	92.3	92.6	92.2
	160 320	92.0	92.2	92.0	92.0	92.5
	320 160	91.3	92.0	91.9	92.2	92.6
	320 320	92.5	92.2	92.5	92.7	92.1

Table 4.31 tabulates the average accuracies for each of the filter and node configurations in OR. An analysis of the table does not indicate any strong outliers nor an increase in

accuracy as more learnable filters are added. In fact the 20-40-400 configuration has an accuracy very similar to the 320-320-500 configuration. The Figure 4.25 is a graphical representation of the table and illustrates the similarity of the accuracies.

After the full analysis of each of the classifiers, this confirms the initial assertion that learning 20 or 40 filters is sufficient for this specific problem i.e., the six and five class problem. Learning more filters adds an immense amount of computation for little added value in terms of accuracy. Similarly, the increase in the number of nodes in the fully connected layer provides no substantial increase in accuracy. Thus, it can be concluded that 200 nodes in the fully connected layer provides a reasonable accuracy for the least amount of computation power.

4.6 Experiment To Assess CNNs By Adjusting Convolution And Pool Filter Dimensions

CNNs focus primarily on two assumptions when learning features. These assumptions are that low level features are local and that the usefulness of a feature in a certain segment might be useful elsewhere. Convolution and pool filter dimensions should be chosen according to how strongly this holds true for the task at hand. Thus, the goal of this experiment was to observe the effect that kernel dimensions of the convolution and pool layers have on the accuracy of each of the classifiers. To limit the number of comparisons carried out, henceforth, only the most performant number of learnable filters and fully connected layer nodes for each classifier from the previous experiment are used to train these models. Both, the NRS and ES context models from the previous experiment, along with their respective best configurations, are carried over to this experiment, where the ES context classifiers are limited to a maximum of 40 learnable filters, as explained in the previous experiment. The effect of the dimensions of the convolution and pool layers on the classifiers of each context are then separately investigated and compared.

Each of the HS classifiers and the OR classifier is trained whilst each time altering the dimensions of the convolution filters CD1 and CD2, and the dimensions of the pooling filters, PD1 and PD2. The convolution filter dimensions will be referred to as CD1 and CD2 respectively e.g. 5 3 refers to CD1 = 5 and CD2 = 3, and so forth. The format used in the text refer to each of the tuples is as follows: CD(X_1,X_2) or PD(X_1,X_2) where CD is the convolution layer, PD is the pooling layer, and X_1 and X_2 refer to the convolution or pool layer dimensions.

The format of the tables and the discussion are similar to that of the previous experiment due to the 2-dimensional nature of the parameters that are being compared. In this overall discussion, Table 4.32 tabulates the average accuracies for the most performant convolution and pool kernel dimensions for each classifier.

TABLE 4.32: Average accuracies for the best convolution and pooling filter dimensions

		Classifiers					
		HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
NRS	**Pooling Filter**	5 3	5 3	3 4	5 3	5 3	5 3
	Convolution Filter	5 3	4 6	3 6	4 5	3 4	6 4
	Accuracy (%)	90.2	81.6	90.5	66.2	68.7	92.8
ES	**Pooling Filter**	6 2	5 3	5 3	5 3	5 3	3 4
	Convolution Filter	3 5	6 5	5 4	3 5	4 3	6 5
	Accuracy (%)	90.9	80.2	89.5	66.9	65.9	92.4

An analysis of the NRS section in Table 4.32 indicates, first, that the best pooling filter tuple is PD(5,3) as evidenced by the fact that it obtains the highest accuracy for all but a single classifier. Regarding the number of convolution filters, there does not seem to be a prominent convolution filter tuple that performs best for every classifier. It appears to be very classifier-specific.

An analysis of the ES section in the table indicates that the best pooling filter tuple is also PD(5,3) as evidenced by the fact that it is the most common filter size for four of the six classifiers. There is also no common convolution filter that performs consistently well in the ES context. It should be noted that in both contexts the model depth and image size, the pooling tuple PD(5,3) performs best. These sizes indicate that a relatively large part of the feature maps is being down sampled. This leads to the belief that either the initial input image sizes should be scaled beforehand or that a deeper network may be capable of providing further learning of deeper hierarchical features if any exist.

Comparing the NRS and ES contexts, the differences in accuracies are observed to be minimal for each classifier using the most performant convolution and pool filter tuples. In fact, the ES context classifiers are more performant, if even very slightly, in two instances—HSO1 and HSO4. It is very encouraging to note that this indicates that applying the ES context restriction leads to a more computationally cost-effective configuration for an accuracy very similar to that of the NRS context.

The following subsections provide further analysis of the results obtained for the filter dimension configurations for each classifier of each context. Graphs of the average accuracies obtained for each of the configurations in the NRS and ES contexts are provided below. The corresponding tabulated data is provided in Appendix C in Section C.3.

4.6.1 Filter Dimension Analysis For HSO1

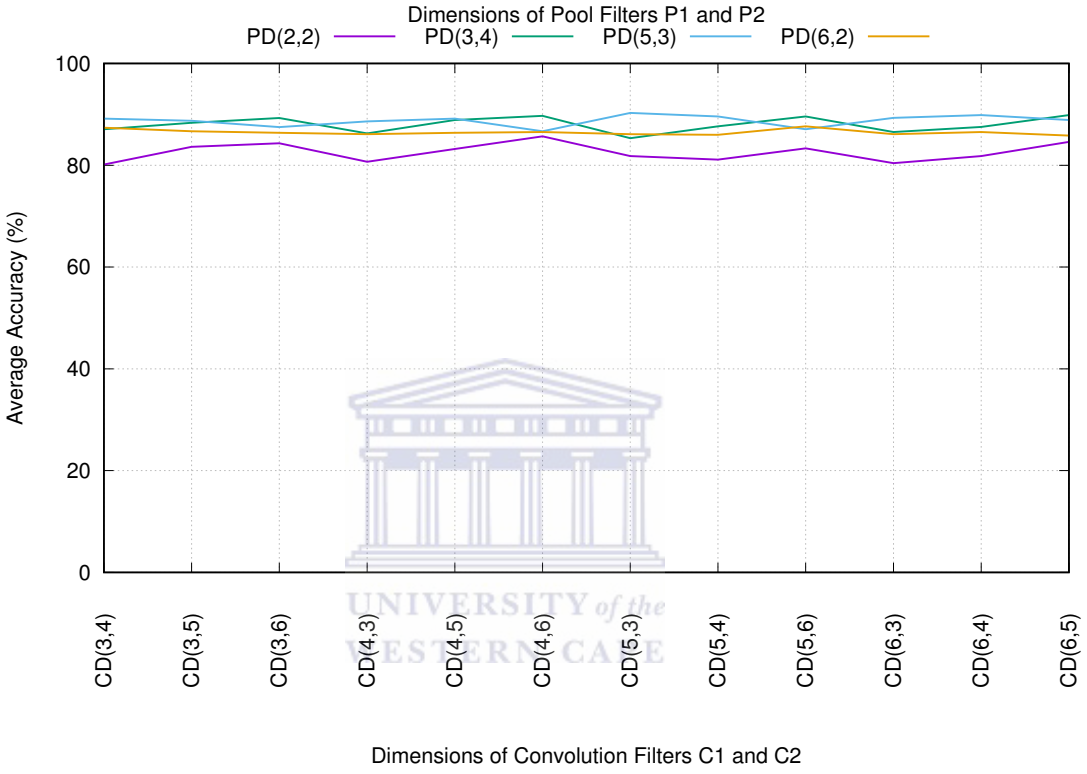


FIGURE 4.26: HSO1 NRS accuracies for varying convolution and pooling filter dimensions

Table C.13 in Appendix C tabulates the average accuracies obtained for the various convolution and pool filter dimensions for HSO1 in the NRS context. The graph of this data is shown in Figure 4.26. An analysis of the graph indicates that each of the pool tuples performs relatively comparably to one another. However, the PD(2,2) tuple—the purple line in the graph—does perform at a slightly lower level in comparison to the other tuples, whilst other pool dimension configurations all perform very similarly across convolution dimensions.

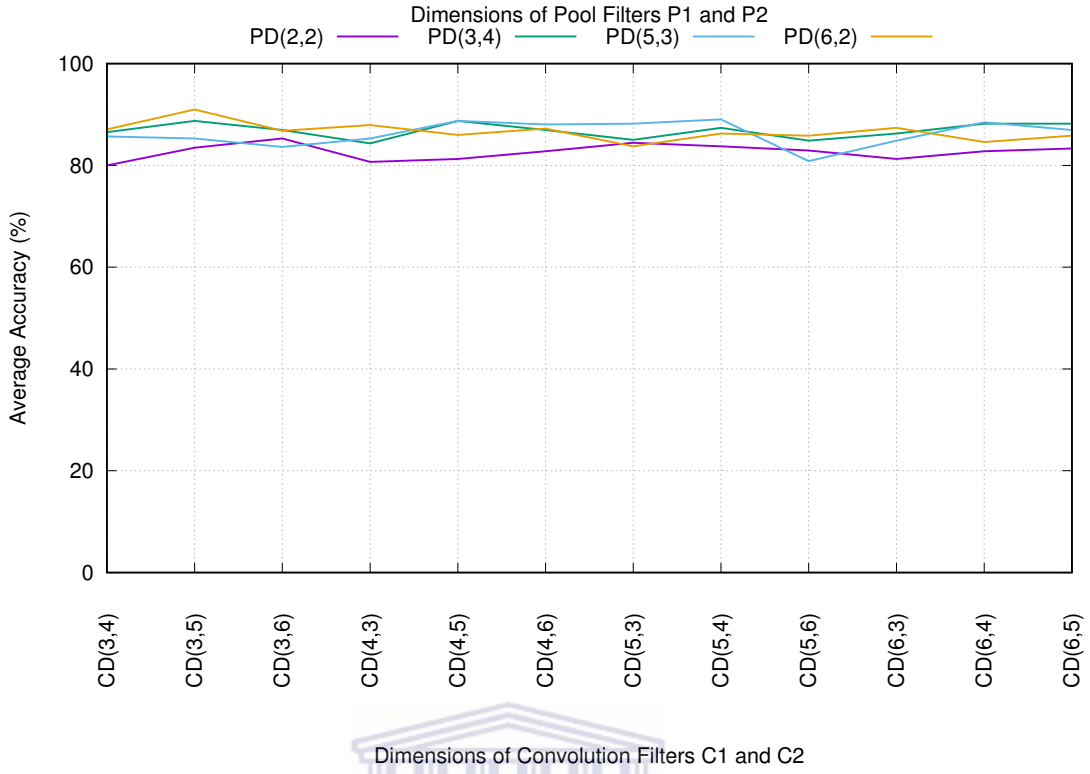


FIGURE 4.27: HSO1 ES accuracies for varying convolution and pooling filter dimensions

Regarding the result of the ES context classifier, Table C.14 in Appendix C tabulates the average accuracies obtained for the various convolution and pool filter dimensions for HSO1 in the ES context. The graph of this data is shown in Figure 4.27. An analysis of the graph indicates that, in this case as well, the tuple PD(2,2) performs at a slightly lower level in comparison to the other tuples, but that all other tuples perform similarly.

4.6.2 Filter Dimension Analysis For HSO2–HSO5 and OR

The results obtained for HSO2–HSO5 are similar to that of HSO1. The pool tuples perform similarly in each instance except for PD(2,2) which performs at a slightly lower accuracy by comparison. However, there are two instances in which HSO5 and OR perform similarly, for PD(2,2), in relation to the other pool tuples for the ES context only. These graphs are shown in Figures C.8 and C.10. In these instances, the graphs clearly demonstrate that PD(2,2) is on a par with each of the other pool tuples. The results also generally indicate that the convolution filter dimension has no observable impact on the accuracy.

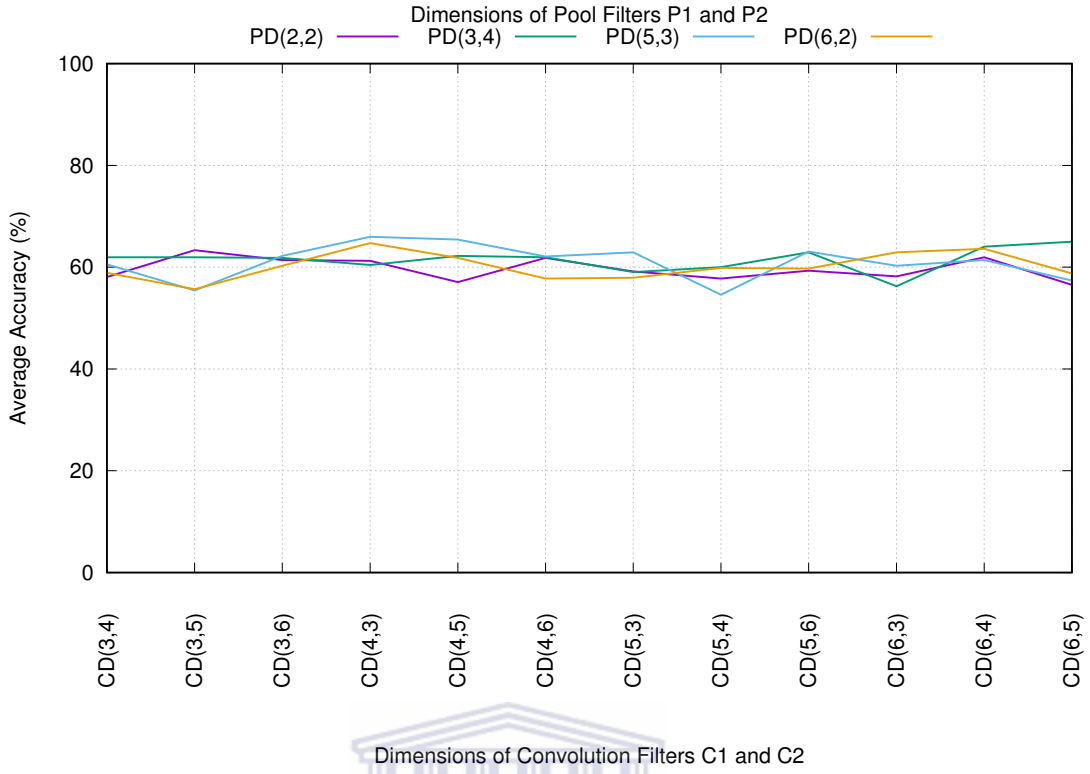


FIGURE 4.28: HSO5 ES accuracies for varying convolution and pooling filter dimensions

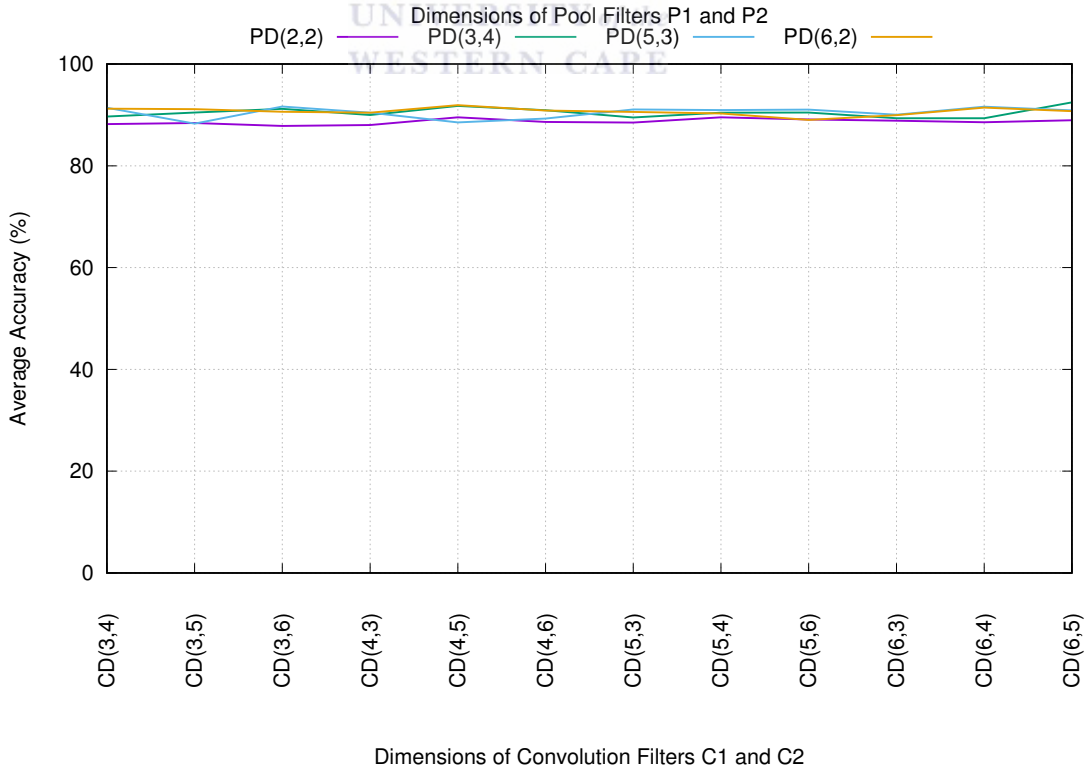


FIGURE 4.29: OR ES accuracies for varying convolution and pooling filter dimensions

The tables and graphs for these results are provided in the Appendix, Tables C.13–C.24 and Figures C.1–C.10, for the interested reader.

TABLE 4.33: Final convolution layer output filter sizes for pooling tuples on an 80×80 image

	Pooling			
	2 2	**3 4**	**5 3**	**6 2**
Min	16×16	5×5	3×3	4×4
Max	18×18	5×5	4×4	5×5

A very clear trend that is observed across classifiers is that the pool tuple PD(2,2) performs at a consistently lower level in comparison with the other pool tuples. However, although in some instances it not as noticeable, this is generally the case.

One reason for the difference between PD(2,2) and all the other investigated tuples could be attributed to the size of the final output feature maps corresponding to each tuple. Table 4.33 shows the minimum and maximum final output feature map sizes obtained when using the different convolution tuples for the provided model architecture. The final output feature maps being used as inputs to the fully connected layer are relatively large for P(2,2). Consequently, it could then be surmised that, at this output feature size, the size of filters could have a bigger negative impact on accuracy. It could also be surmised that scaling the initial input image from 80×80 pixels to a lower resolution may allow PD(2,2) to be more performant than the other tuples given that the feature map sizes will then be more in line with those of PD(3,4), PD(5,3) and PD(6,2) in the 80×80 context. Future testing and analysis is required to confirm this conjecture. However, if this holds true it could also reduce a myriad of factors such as training time, recognition time, and model size in memory. This is left to future work.

It was originally thought that PD(5,3) was the most performant tuple based on Table 4.32 which contains the best performing tuple configurations. Whilst this is true, it performs better only by a marginal amount in most instances across each of the classifiers.

To conclude, for an 80×80 image the models seems to function best with the pool tuples that allow for smaller features maps at the final convolution and pooling layer. It is also observed that varying the convolution filter dimensions does not yield a noticeable difference in accuracy.

4.7 Comparison Of The Default And Best NRS and ES Classifiers

This section summarises the average results obtained for each of the classifiers in the default, NRS and ES contexts. The Table 4.34 tabulates all the accuracies obtained. The default configuration refers to the same model used in the NBG vs BG experiment i.e., Section 4.3.

TABLE 4.34: Accuracies for each classifier for default, NRS context and ES context

	Classifiers					
	HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
Default Accuracy (%)	88	74	86	59	60	90
NRS Accuracy (%)	90	80	91	65	71	92
ES Accuracy (%)	90	79	90	61	65	92

An analysis of the table indicates, very clearly, that both the NRS and ES classifiers outperform each of the default classifiers in every instance and in some cases by a noticeable amount. However, the accuracy of the default classifier cannot be considered as being considerably lower than that of the ES or NRS classifiers and this is attributed to the fact that the default model was a reasonably efficient model to begin with—the same as used in the deep MNIST classifier [72, 76]. Had the default model used the Sigmoid or LogSigmoid activation functions, it is likely that the default classifier's accuracy would have been noticeably lower. The small difference by which the NRS classifier outperforms the ES classifier is encouraging and indicates that a smaller model can obtain very comparable results to that of a larger model. This demonstrates the feasibility of using deep learning models to perform classification on embedded and mobile devices.

4.8 Experiment To Assess The 2-Stage CNNs

The previous experiments provided two sets of configurations that allowed for the best accuracy to be noted in both an ES and an NRS context. Each of these HSO and OR models was used to construct two final 2-stages classifiers. Their configurations are tabulated in Table 4.35. The goal of this experiment was to determine the accuracies of, and compare, these combined 2-stage classifiers.

TABLE 4.35: Final configurations for each classifier for NRS context and ES context where: LF = Learnable Filters, FCN = Fully Connected Layer Node Count, AF = Activation Function, PFD = Pool Filter Dimension and CFD = Convolution Filter Dimension

		Classifiers					
		HSO1	**HSO2**	**HSO3**	**HSO4**	**HSO5**	**OR**
NRS	**LF**	160 320	320 160	160 80	320 160	320 320	320 320
	FCN	500	200	600	300	200	500
	AF	ReLU6	HardTanH	HardTanH	HardTanH	ReLU	ReLU
	PFD	5 3	5 3	3 4	5 3	5 3	5 3
	CFD	5 3	4 6	3 6	4 5	3 4	6 4
ES	**LF**	20 40	20 20	20 20	40 20	20 20	20 40
	FCN	600	300	200	600	400	400
	AF	ReLU6	HardTanH	HardTanH	HardTanH	ReLU	ReLU
	PFD	6 3	5 3	5 3	5 3	5 3	3 4
	CFD	3 5	6 5	5 4	3 5	4 3	6 5

It is important to note that the six hand shapes and five orientations in the 2-stage classifiers effectively result in 30 unique classes, i.e., one class for each unique hand shape and orientation pair. The random guessing accuracy for a 30-class problem is approximately 3%.

The results obtained for each 2-stage classifier are provided in tables: C.25 and C.26 in Appendix C. Overall, the averages obtained for the ES and NRS classifiers were very high accuracies of 73.4% and 73.2% respectively. This is an outstanding result given the complexity of the classification problem at hand. It is observed that the ES classifier obtains a marginally higher overall accuracy than the NRS classifier. The marginal difference by which the ES classifier outperforms the NRS classifier is encouraging and indicates that a smaller model can obtain good, if not better, results than a larger model. This demonstrates the feasibility of using deep learning models to perform classification on embedded and mobile devices.

TABLE 4.36: Average orientation accuracies across shapes for the 2-stage classifier

	Classifier	
Orientation	**ES**	**NRS**
1	79	79
2	78	76
3	82	87
4	67	64
5	60	57

An analysis of the results on a per orientation basis, across all hand shapes, indicates that in four of the five instances, the ES classifier outperforms the NRS classifier by a difference of at most 3%. The NRS classifier does, however, outperform the ES classifier for O3. Generally speaking, no outliers are observed. The highest performing orientation is O3 across all shapes, followed by O1 and O2. O4 and O5 obtain relatively lower accuracies, the reasons for which have been described at length in the experimental analysis previously. These accuracies are by no means low and are still, in the worst case, 57%, which is 19 times better than the random guessing classification accuracy for a 30-class problem.

TABLE 4.37: Average shapes across orientations accuracies for the 2-stage classifier

	Classifier	
Shape	**ES**	**NRS**
1	66	69
2	77	80
3	76	72
4	75	73
5	82	82
6	63	60

TABLE 4.38: Accuracies of HS1 and HS6 across all orientations

Orientation	**ES**		**NRS**	
	Shape 1	**Shape 6**	**Shape 1**	**Shape 6**
1	63	72	80	70
2	88	58	91	57
3	80	70	90	71
4	55	45	42	45
5	45	68	41	59

An analysis of the results on a per shape basis, across all hand orientations, indicates that best classes for both the NRS and ES contexts are HS2, HS3, HS4 and HS5 which each obtain accuracies that exceed 72%. There are no observed outliers between the NRS and ES classifiers with a difference of at most 4%. The two lowest accuracies, but by no means low, obtained by either the NRS or ES classifier are for HS1 and HS6 with accuracies ranging between 60% and 70%. It should be considered that these accuracies are at least 20 times better than the random guessing classification accuracy for a 30-class problem.

Table 4.38 tabulates the results for the lowest performing classes, i.e., HS1 and HS6. An analysis of this table indicates that the relatively lower accuracies obtained for HS1 are attributed to specific orientations O2 and O4. The relatively lower accuracies for HS6 are attributed to O4 and O5. The lower accuracies for HS6 can be attributed to the fact that for these orientations it was shown that this hand shape performed at a relatively lower level in their respective HSO classifiers. The interested reader may find the results of these HSO classifiers in Appendix C.

TABLE 4.39: Correctly classified instances per subject

Test Subject	**Accuracy (%)**	
	ES	**NRS**
4	73	75
5	66	66
8	73	72
9	80	79

The Table 4.39 shows the percentage of correctly classified instances per subject. Comparing the NRS and ES classifier accuracies across test subjects in the table indicates that both the NRS and ES classifiers perform relatively consistently even across subjects with very little difference between the number of instances being correctly classified per subject. It should be noted that these classifiers use different configurations. Consequently, it is very good to see this consistency amongst subjects for both of the classifiers.

Comparing the accuracies of each test subject, it is seen that there is very little difference in accuracy across subjects, with the difference between the lowest and highest subject being only 7% for the ES classifier, and 8% for the NRS classifier. Variations in accuracy of this magnitude are very encouraging given the wide variations in skin tone, body dimension and gender between test subjects. This small difference suggests that the classifiers are robust to variations in test subjects.

A further analysis was carried out to determine in which stage each 2-stage classifier failed to recognise an image correctly, i.e. in the first stage at the orientation classifier or in the second stage at any of the hand shape classifiers. Detailed tables of these results are provided in Tables C.27 and C.28 in Appendix C for the interested reader. The Table 4.40 is a summary of these tables and shows the overall percentage classification errors as a result of rejection in each stage.

TABLE 4.40: Overall percentage classification errors as a result of rejection at the first stage (orientation classifier) or the second stage (any of the hand shape classifiers)

	ES Error (%)	**NRS Error (%)**
Stage 1	7.4	8.1
Stage 2	19.1	18.6
Overall	26.5	26.7

An analysis of the tables indicates that there were fewer instances of rejection in stage one, the OR classifiers, than in the second stage, the HSO classifiers. The lower rejection rate in stage 1 is in accordance with the high average accuracies, of over 90%, obtained for the OR classifiers. Similarly the reason for the relatively higher classification error of stage 2, in comparison to stage 1, is attributed to the relatively lower accuracy of each of the five hand shape classifiers as compared to OR.

The results for the ES and NRS classifier for both stages are very similar with minute differences in error amounting to 0.7% and 0.5% respectively. This is propagated into the overall errors which are different by only 0.2%. This is very encouraging considering the ES classifiers were limited to a maximum of 40 filters. This clearly demonstrates the efficacy of deep learning towards hand shape and hand orientation recognition in an embedded system and mobile device context.

4.8.1 Convolutional Neural Network Classification Speed

The full 2-stage classifier was constructed by combining the best ES classifiers. These models were converted and transferred for use on an iPhone device using the ES neural network library developed by the researcher. The goal of this test was to illustrate that an iPhone is capable of performing real time recognition of images using a CNN. 100 images were stored on the mobile device memory and copied into a memory buffer before timing began i.e., the time to load the images was excluded from the recognition time. The average time for a single instance to be classified and the number of frames processed per second is reported in Table 4.41. The individual iteration times are provided in Appendix C.29.

TABLE 4.41: Recognition speeds for best ES configuration

Classification Time (seconds)		
	iPhone 5C	**iPhone 6+**
Average Time (s)	0.0267	0.0074
Average FPS	37	135

The table tabulates the results obtained when performing recognition on the iPhone 5C and iPhone 6+ using the ES 2-stage classifier. Analysis of the table indicates that real time classification is possible even using an older device such as the iPhone 5C. The iPhone 5C and iPhone 6+ obtained an recognition speeds of 37 FPS and 135 FPS respectively. It should be noted that these models were executed on an iPhone 5C and iPhone 6+ with an A5 processor and A8 processor respectively. Newer devices will be able to do these convolutions even faster on the CPU. In addition to this, many of these operations can be parallelised on the mobile GPU for an even faster recognition time. This mGPU implementation is left for future work.

4.9 Summary

This chapter covered all the experiments to test both the HTS and HGR components. The face detection algorithm employed in this research was shown to execute at 43 FPS and 89 FPS on the iPhone 5C and iPhone 6+ respectively. The HTS component, which utilises the face detection algorithm, was shown to be robust with an average tracking accuracy of 82.58%.

Experiments were carried out to test the HGR component. The first experiment—testing the NBG and BG datasets—indicated that it is beneficial to remove the background from the segmented hand images. This generally provides a better classification accuracy.

The second experiment–The comparison of activation functions—indicated that the choice of activation function provides relatively consistent performance for each of the activation functions. However, the Sigmoid and LogSigmoid functions generally perform at a lower level in comparison to the other activation functions.

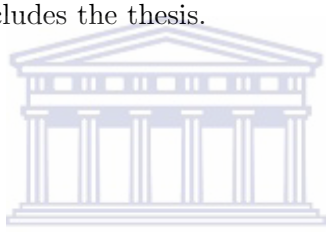
The third experiment—Assessing the impact of learnable filters in convolution layers and the number of fully connected layer nodes on accuracy—indicates that learning more filters does not necessarily equate to a noticeably better model. The highest accuracies were obtained learning, using more than 40 filters, however, even learning with fewer than or equal to 40 filters resulted in very comparable accuracies.

The fourth experiment—Assessing the impact of the convolution and pool filter dimensions on accuracy— revealed that the changes in the convolution filter dimensions provided no uniform increase or decrease in accuracy for each of the pool filter dimensions configurations. However, it did indicate that the pool filter dimensions had a larger impact on the accuracy of the classifier.

After these experiments were completed the models were used to construct the 2-stage classifier. The most performant ES model was shown to obtain a final accuracy of 73.4% using the said 2-stage approach. This is a very encouraging accuracy for this 2-dimensional machine learning problem of recognising hand gestures, which involves recognising two gesture parameters simultaneously, namely, shape and orientation.

This same model was shown to be realtime even with a limit being imposed on the number of learnable filters. The CNN is capable of recognising new images at a speed of 37 FPS and 135 FPS using the iPhone 5C and iPhone 6+ respectively.

The following chapter concludes the thesis.



Chapter 5

Conclusion

This research focused on deep learning in an embedded context, specifically CNNs, and its applications to South African Sign Language gesture recognition using two of the five fundamental gesture parameters - hand shape and hand orientation. CNNs were shown to be very good candidates for image recognition and classification tasks in the literature and performed very well in a gesture recognition context for two gesture parameters. Two major factors were considered in this research: classification accuracy and classification speed in an embedded context.

With respect to the first question: "Which parameter values for the CNNs provide an optimised network model in the context of hand shape recognition in various orientations with respect to accuracy on embedded devices?", It was shown that the most performant activation functions were generally either the rectified linear functions or the symmetric sigmoidal function - hard hyperbolic tangent.

In response to the second question: "How accurate are CNNs at classifying unseen images in the context of hand orientation recognition on embedded devices after an optimised network model has been established?", the results obtained for the orientation classifier show that it obtained an overall accuracy of 92% and is capable of running on an iPhone 5C or better.

In response to the third question: "How accurate are CNNs at classifying unseen images in the context of hand shape recognition in various orientations on embedded devices after an optimised network model has been established?", it was shown that the final 2-stage classifier managed to obtain a very high accuracy of 73.4% which is capable of running on an iPhone 5C or better.

In response to the fourth question: "How fast are CNNs at classifying unseen images in the context of hand shape recognition in various orientations on embedded devices

after an optimised network model has been established?", The classifier does so at a very reasonable speed by obtaining speeds of 37 FPS and 137 FPS on an iPhone 5C and iPhone 6+ respectively.

Thus, in response to the final research question "How performant is Deep Learning, specifically CNNs, in the context of hand shape recognition in multiple hand orientations on embedded devices?" It was concluded that deep learning and specifically, CNNs, are very performant in terms of both accuracy and speed in the context of hand shape recognition in multiple orientations on embedded devices such as the iPhone.

This research has added considerable value and novel results to the field of gesture recognition in a South African Sign Language context. This research has also provided considerable insight for the Assistive Technologies Research Group. It achieved this insight by providing the research into the recognition of hand orientation and hand shape in multiple orientations using CNNs on mobile devices.

5.1 Future Work

Although a satisfactory accuracy was obtained by the CNN classifier there are a number of other steps that may improve the accuracy of the classifier. These are:

- Increasing the number of training and testing samples, i.e., obtaining a larger dataset. A larger volume of data may help the classifier improve generalisation, especially in certain instances where poorer accuracies were obtained.
- Recent research has indicated that using Generative Adversarial Networks (GANs) to train on a subset of the training samples, combined with subsequently training a CNN, has proved to provide excellent results, when trained against the Google SVHN (Street View House Number) dataset.
- Adding a dropout layer in the case where classifiers are overfitting. Dropout layers have shown to be effective in cases where classifiers are overfitting.
- Altering the learning rate or optimisation method and selecting different mini-batch sizes.
- Introducing regularisation where models are overfitting or under-fitting, i.e., when the models are either too simple or complex.

In order to speed up each of the components, future work should also focus on parallelising convolution and pooling operations on the mobile GPU. In addition to the machine

learning process optimisation, computer vision tasks should also be done on the mobile GPU where possible.

5.2 Concluding Remarks

The researcher has concluded that this research and experiments conducted has added considerable value to his life and and it was proved to be a fulfilling learning experience. It is hoped that this research can serve as a basis for the selection of machine learning techniques in an embedded context for other sign language parameters by the Assistive Technologies Research Group, and for classification problems in general.

Appendix A

Appendix - Image Processing

A.1 Face Detection Using Viola-Jones Framework

The face detection in this research is done by applying the Viola Jones object detection framework in combination with cascades that are specifically designed for detecting faces [19]. The Viola-Jones framework (VJF) is able to detect faces rapidly in real time.

A.1.1 Notable VJF Contributions to Object Detection

The VJF introduced a number of new contributions to object detection but three of these contributions are particularly notable. These contributions are discussed below:

A.1.1.1 Integral Image

The VJF introduced a novel new image representation called the *Integral Image*. This representation allows for very fast feature evaluation. Inspired by the work of Papageorgiou et al. the VJF does not work directly with image intensities but rather a set of features that are similar to Haar basis functions. The integral images allows for these features to be hastily computed at various scales. The integral image can be computed with only a few operations per pixel and, once computed, allow for any of these Haar-like functions to be computed in constant time.

A.1.1.2 Adaboost on Haar-Like Features

The VJF also introduced a classifier that selects a small number of important features by using the Adaboost machine learning algorithm. An image sub-window can contain

a large number of Haar-like features far exceeding the number of pixels contained in the full image. To provide real-time performance it is crucial that only the most important features be selected and the majority of available less crucial features be discarded. The Adaboost algorithm is modified such that weak learners are constrained in such a way that each returned weak classifier is only dependent on a single feature. Thus, each stage of the boosting process, which returns a weak classifier, can be viewed as a feature selection process [77–79].

A.1.1.3 Cascade Classifiers

The VJF introduces a method for successively combining complex classifiers in a cascade structure which dramatically increases the speed of the the face detector by limiting computation power and focusing only the most important regions of an image. This notion of focusing on the important regions of the image is that it is possible to determine where an object might occur [80–82]. A crucial metric in the success of this process is that majority or all of the object instances must be selected in the focus filter.

A.1.2 Features

The simplest features used are similar to Haar basis functions used by Papageorgiou et al. [79]. More precisely, three features are used, namely: *two rectangle features*, *three rectangle features*, and *four rectangle features*.

Two rectangle features are computed by calculating the difference between two vertically or horizontally adjacent rectangular regions. It should be noted that these regions are required to be the same size. A three rectangle feature is computed by calculating the difference between the sum of the outer two rectangles and the centre rectangle. Lastly the four rectangle feature is computed by calculating the difference between diagonal pairs of rectangles. Each of these features is convolved across the image at a number of scales and the appropriate calculation occurs for a particular feature. If this calculation exceeds a acceptable value the object is considered to be detected by the covered feature region.

A.1.3 Integral Image and the Computation of Features

To calculate the features in an image at a variety of scales and positions is costly in terms of computation. Consequently the intermediary Integral Image was introduced because it allows for the rapid calculation of features.

An Integral Image is comprised of the sum of all pixels to the left and above, given an arbitrary point (x,y), inclusive given by the formula:

$$G(x, y) = \sum_{i} \leqslant x, j \leqslant y \tag{A.1}$$

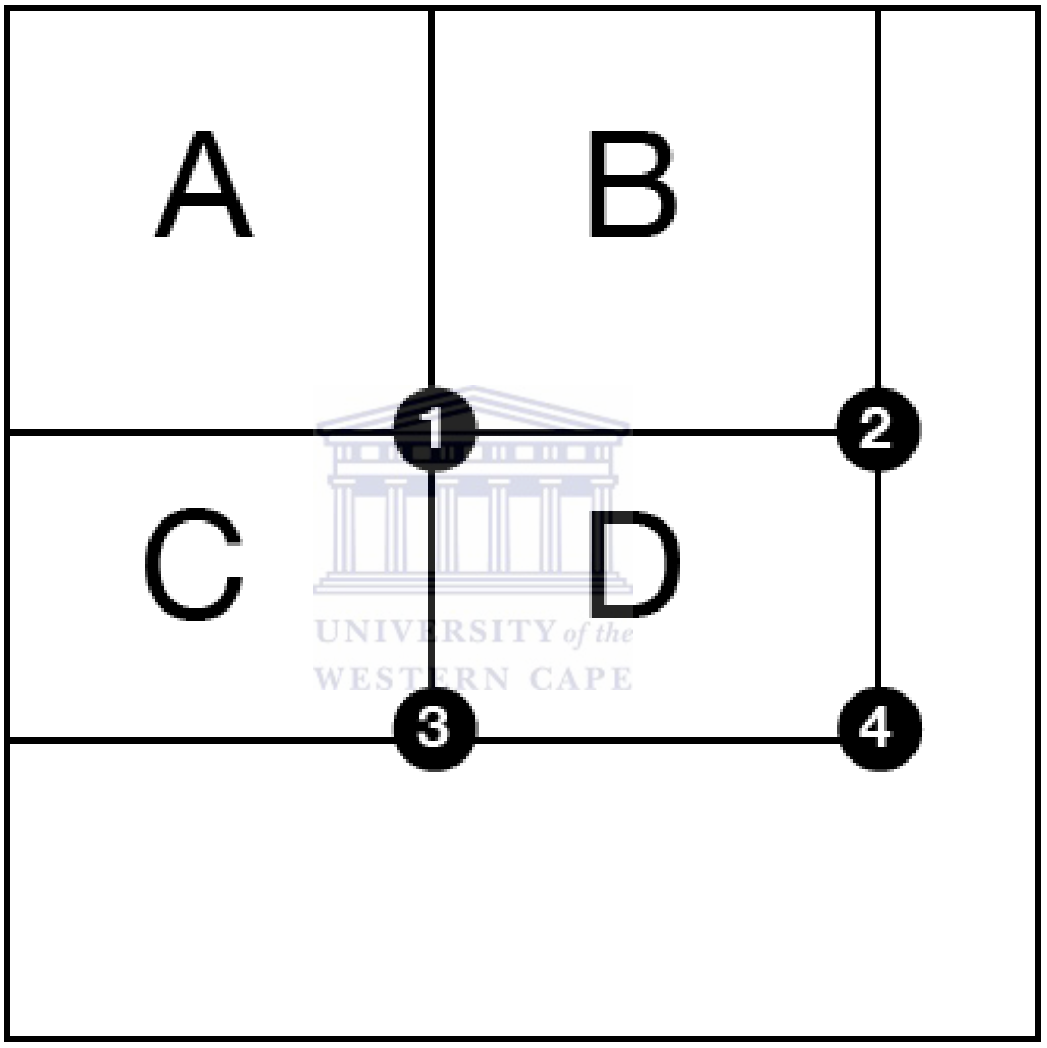


FIGURE A.1: Integral Image calculations.

It is possible to compute a Haar-like feature by computing the sum of any rectangle in the original image. If one observes Fig A.1 its clear that it is possible to compute the sum of the pixels in D by subtracting the Integral Image value at *4* from the sum of the Integral Image values at *2* and *3*, *1* is added to this difference to account for the excess caused by the intersection of rectangles (A and B) represented by *2* and rectangles (A and C) represented by *3*.

In doing so it is apparent that one can calculate the sum of pixels in any rectangle and consequently compute any Haar-like features at any scale or location.

A.1.4 Adaboost in Feature Selection

Adaboost is a learning algorithm that selects only the best features from a large list of features. The VJF uses a modified version of the algorithm to select only a few of the most critical Haar-like features. There is a large number of Haar-like features, often exceeding the number of pixels in the image. In order to speed up computation it is critical that only the best features be used to train a classifier.

Despite the fact that each of the features can be computed without much computational cost the sheer number of features can lead to a slow classifier. Thus it is crucial that only those features, that best distinguish between positive and negative examples, be used, in order to achieve the strongest classifier.

A.1.5 Rejection Cascade for Weak Feature Classifiers

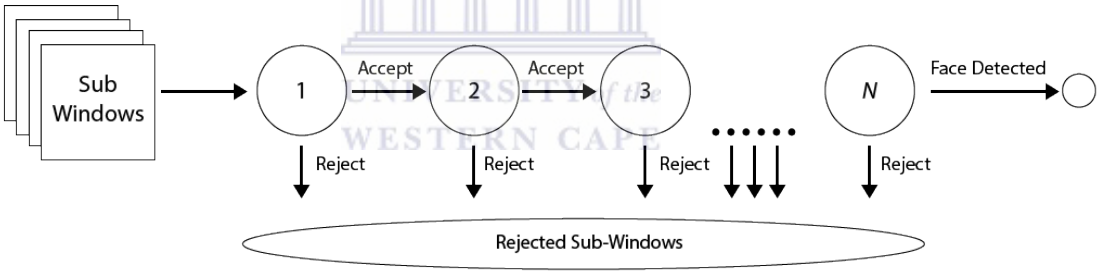


FIGURE A.2: Structure of the rejection cascade.

The rejection cascade is constructed in such a manner as to achieve a high accuracy while significantly lowering the computational cost for negative examples. The notion behind this rationale is that a simpler, and faster boosted classifier can reject most negative sub-windows whilst still being capable of detecting the positive instances.

The rejection cascade takes the form of a degenerate decision tree as shown in Figure A.2. Taking a look at Figure A.2 it is clear that each sub-window needs to pass through a set of classifiers and if at any stage during this pass any of the classifiers infer a negative result the sub-window is rejected. In doing so the computational power is used optimally for sub-windows that do not contain a face.

A.2 Gaussian Mixture Models

The use of stationary cameras for surveillance is a common use case. Detecting foreign or moving objects is a common theme when analysing a capture video or scene. An assumption that can be made is that the images of a scene without the foreign objects exhibit some behaviour that can be described by a statistical model. If a statistical model of the scene is constructed then foreign objects can easily be detected as outliers to this model. This process is known as "background subtraction" [26].

This background subtraction can be performed by a probabilistic method called Gaussian Mixture Models. For a single Gaussian model a common bottom-up approach is used to model the scene with a probability density function for each pixel. Any subsequent pixels from a new image are considered to be part of the background if the pixel can be described by the density function. For a static scene a simple model could be an image of the scene without foreign objects. This would be followed by the estimation of appropriate values for the variances in pixel intensities in the image. This single Gaussian model was used in the construction of the Pfinder system as described in [83]. Pixel values have complex distributions and models that can better describe these are required. A Gaussian mixture model (GMM) was proposed for background subtraction in as in [84].

GMMs are a probabilistic method used for background subtraction, it achieves this background subtraction by highlighting motion pixels across a number of frames such that pixel brightness indicates the recency in motion for a pixel between 0 and 255. The pixels (i, j) in a sequence of images I at a time t can be described as follows:

$$\{P_{1,i,j}...P_{x,i,j}\} = \{I(i, j, x) : x \in [1, t]\} \tag{A.2}$$

Each pixel in the image can be modelled as a mixture of n Gaussian distributions where $W_{y,t}$ represents a weight estimate for the n^{th} Gaussian. The probability that a pixel has the value $P_{x,i,j}$ at the time x can be expressed by the following equation:

$$P(P_{x,i,j}) = \sum_{y=1}^{n} W_{y,t} \times \eta(P_{x,i,j}, \mu_{y,t}, \Sigma_{y,t}) \tag{A.3}$$

$\eta(P_{x,i,j}, \mu_{y,t}, \Sigma_{y,t})$ describes the normal distribution of the y^{th} Gaussian component with a mean of $\mu_{y,t}$ expressed by the following equation:

$$\eta(P_{x,i,j}, \mu_{y,t}, \Sigma_{y,t}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_{y,t}|^{\frac{1}{2}}} e^{\frac{-1}{2}(P_{x,i,j}-\mu_{y,t})^T \sum_{y,t}^{-1}(P_{x,i,j}-\mu_{y,t})} \tag{A.4}$$

where $\Sigma_{n,t} = \sigma_{n,t}^2 \times I$ is the co-variance of the n^{th} Gaussian given that I is the identity matrix.

A fitness value $\frac{W_{y,t}}{\sigma y,t}$ is used when ordering the n distributions, the first M distributions are used to model the background scene and the estimate given by the following formula:

$$M = argmin_m(\sum_{y}^{m} W_{y,t} > T_h) \tag{A.5}$$

where T_h refers to the threshold value that represents the minimum portion of the background model.

Given an updated background, foreground detection is then achieved by labelling all pixels which are determined to be more than a standard deviation of 2.5 away from any of the M distributions as foreground pixels. If there is a match between the test value and the x-th Gaussian component Wx,t, it is updated as shown below:

Given a new background, pixels belonging to the foreground are labelled as such if a difference in standard deviation exceeds 2.5 from any of the M distributions. If a pixel satisfies the constraint against the y^{th} Gaussian component then $W_{y,t}$ is updated as shown below:

$$W_{y,t} = W_{y,t-1} \tag{A.6}$$

$$\mu_{y,t} = (1-\rho)\mu_{y,t} + \rho P_{x,i,j} \tag{A.7}$$

$$\sigma_{y,t}^2 = (1-\rho)\sigma_{y,t-1}^2 + \rho(P_{x,i,j} - \mu_{y,t})^T(P_{x,i,j} - \mu_{y,t}) \tag{A.8}$$

$$\rho = \alpha\eta(P_{x,i,j}|\mu_n, \Sigma_n) \tag{A.9}$$

where $\frac{1}{\alpha}$ refers to the time constant that determines the rate of change. If the pixel does not satisfy the constraint it is updated as shown below:

$$W_{y,t} = (1 - \alpha)W_{y,t-1} \tag{A.10}$$

$$\mu_{y,t} = \mu_{y,t-1} \tag{A.11}$$

$$\sigma^2_{y,t} = \sigma^2_{y,t-1} \tag{A.12}$$

If the pixel does not satisfy any of the Gaussian components then a new Gaussian component is constructed with high variance. A low weight parameter and the pixel, as its mean, replaces the Gaussian component with the lowest probability.

A.3 Template Matching Formula

Template matching is a computer vision technique for finding areas of an image that are similar to those of a specified template or patch image [27]. A number of template matching methodologies are available, however, this research makes use of the Normalised Cross Correlation Template Matching. The image correlation is calculated by the sum of pairwise multiplications of corresponding pixel values in the images [85].

A.3.1 Cross Correlation Template Matching Formulae

$$R_{crossCorr}(x, y) = \sum_{x'y'}(T(x', y') * I(x + x', y + y')) \tag{A.13}$$

$$R_{NorCrossCorr}(x, y) = \frac{\sum_{x'y'}[(T(x', y') * I(x + x', y + y'))]}{\sqrt{\sum_{x'y'} T(x', y')^2 * \sum_{x'y'} I(x + x', y + y')^2}} \tag{A.14}$$

A.4 CAMSHIFT

Continuously Adaptive Meanshift (CAMShift) is an extended version of the meanshift algorithm [28]. CAMShift is a colour based computer vision tracking algorithm and was developed as a perceptual user interface, and consequently needs to operate in real time and should use the least amount of computational resources. Perceptual user interfaces are interfaces that allow for computers to digitally mimic the human senses such as giving the computer the ability to see or touch.

The CAMShift algorithm uses a non-parametric technique to escalate a gradient and find the mode (or highest peak) of probability distributions which is called the mean-shift algorithm. To be applicable to the tracking of a hand within a video sequence the algorithm is modified to find the mode of a colour distribution in a video sequence. To obtain this colour distribution a colour histogram is used. Because the colour distributions of the video data change over time the meanshift algorithm needs to be modified to adapt to the dynamic nature of the probability distribution it is tracking. This is one of the distinct differences between CAMShift and meanshift.

In order for CAMShift to track coloured objects in a sequence of frames, a probabilistic distribution image of the desired colour is necessary. This is achieved by using a 1-dimensional hue colour histogram, this algorithm colour model in particular places a singular focus on the hue channel. In order to obtain this colour histogram the hand first needs to be located at least once using template matching as described in section A.3. Once the hand is located the colour histogram can be created by binning the skin pixels in the image section containing the hand.

During operation, the histogram is used as a model, or lookup table, to convert subsequent video frame pixels into a probability of skin image. Tracking, using CAMShift, can take place on this skin probability image. CAMShift cleverly only does the skin probability test on a smaller search window as opposed to a larger search space, the full image, in order to save computational resources. The search window size is a tradeoff between computation and accuracy, and should be sized in such a manner that allows for the detection of the object, and its motion in any direction. Thus, the search window is given by the initial location of the hand plus some padding for motion in any direction.

Another difference between meanshift and CAMShift mainly is that CAMShift is an extension of the meanshift algorithm and adapts to the rotation and size changes of the object within the frame.

More concisely, the meanshift algorithm is calculated as follows:

1. Choose a search window size.

2. Choose the initial location of the search window.

3. Compute the mean location in the search window.

4. Center the search window at the mean location computed in Step 3.

5. Repeat Steps 3 and 4 until convergence (or until the mean location moves less than a preset threshold).

For discrete 2D probability distribution images the mean or centroid (steps 3 and 4 in the meanshift algorithm) is calculated as follows:

Find the zeroth moment given by the formula

$$M_{00} = \sum_{x=1}^{n}\sum_{y=1}^{n} ProbabilityDist(x, y) \tag{A.15}$$

Find the first moment for x and y

$$M_{10} = \sum_{x=1}^{n}\sum_{y=1}^{n} x \times ProbabilityDist(x, y); M_{01} = \sum_{x=1}^{n}\sum_{y=1}^{n} y \times ProbabilityDist(x, y) \tag{A.16}$$

Consequently the mean search window location or centroid is given by

$$Centroid_x = \frac{M_{10}}{_M00}; Centroid_y = \frac{M_{01}}{_M00}; \tag{A.17}$$

After locating the centroid using the meanshift algorithm the search window size is updated according to the CAMShift specification. The width and height for the search window is given by the following formulae:

$$SearchWindow_W = 2 \times \sqrt{\frac{M_{00}}{Prob_{MAX}}} \tag{A.18}$$

$$SearchWindow_H = 1.2 \times SearchWindow_W \tag{A.19}$$

A.5 Thresholding

A number of thresholding techniques exist such as simple thresholding, adaptive thresholding and Otsu's binarisation [86]. Thresholding involves converting an image from its current representation into a binary representation given a particular value or in the case of adaptive thresholding and Otsu binarisation this value is calculated. These types of thresholding are explained below.

A.5.1 Simple Thresholding

Simple thresholding is the use of a static threshold value to convert a grayscale into a binary image. All values higher than this value get assigned a particular value (for example black or 0) and all values lower get assigned another value (for example white or 255).

A.5.2 Adaptive Thresholding

Adaptive thresholding differs from the simple thresholding by calculating a threshold for a smaller region of the image. Then there are different thresholds for different regions of the image and generally this gives a better result for images with varying levels of illumination.

A.5.3 Otsu's Binarisation

Otsu's method of thresholding provides a mechanism for choosing a good candidate threshold value for a bimodal image. Bimodal images are images that contain a two large subsets of a particular pixel value or more clearly a histogram calculated from an image that contains two peaks. Otsu's method says that the value between these two peaks is a good candidate threshold value.

Appendix B

Appendix - Machine Learning

B.1 Convolution Operator

A convolution forms the basis of the convolutional layers in a convolutional neural network. It may be described as a sliding window function that is applied element wise to a matrix, in this case the matrix refers to an image [87]. This sliding window is more aptly named a kernel, filter or feature detector [88]. k feature maps are learnt by convolving k filters across the image. The size of each feature map is calculated as follows: $fm_w = (i_w - k_w) + 1$, $fm_h = (i_h - k_h) + 1$ where fm is the feature map, i is the image and k is the kernel. Thus a single convolution layer would produce an array of k feature maps. Each map is computed by convolving kernel n (where n is an element of k) across the image and summing together each of the resulting outcomes of the convolution as shown in Figure B.1. Furthermore, k feature maps are given by means of the following function: $fm = \sigma(weight \times kernel + bias)$ where the weights and bias are those from the visible layer to the hidden layer. The sigmoid function could be replaced by any other non-linear transfer function such as ReLU, TanH, etc.

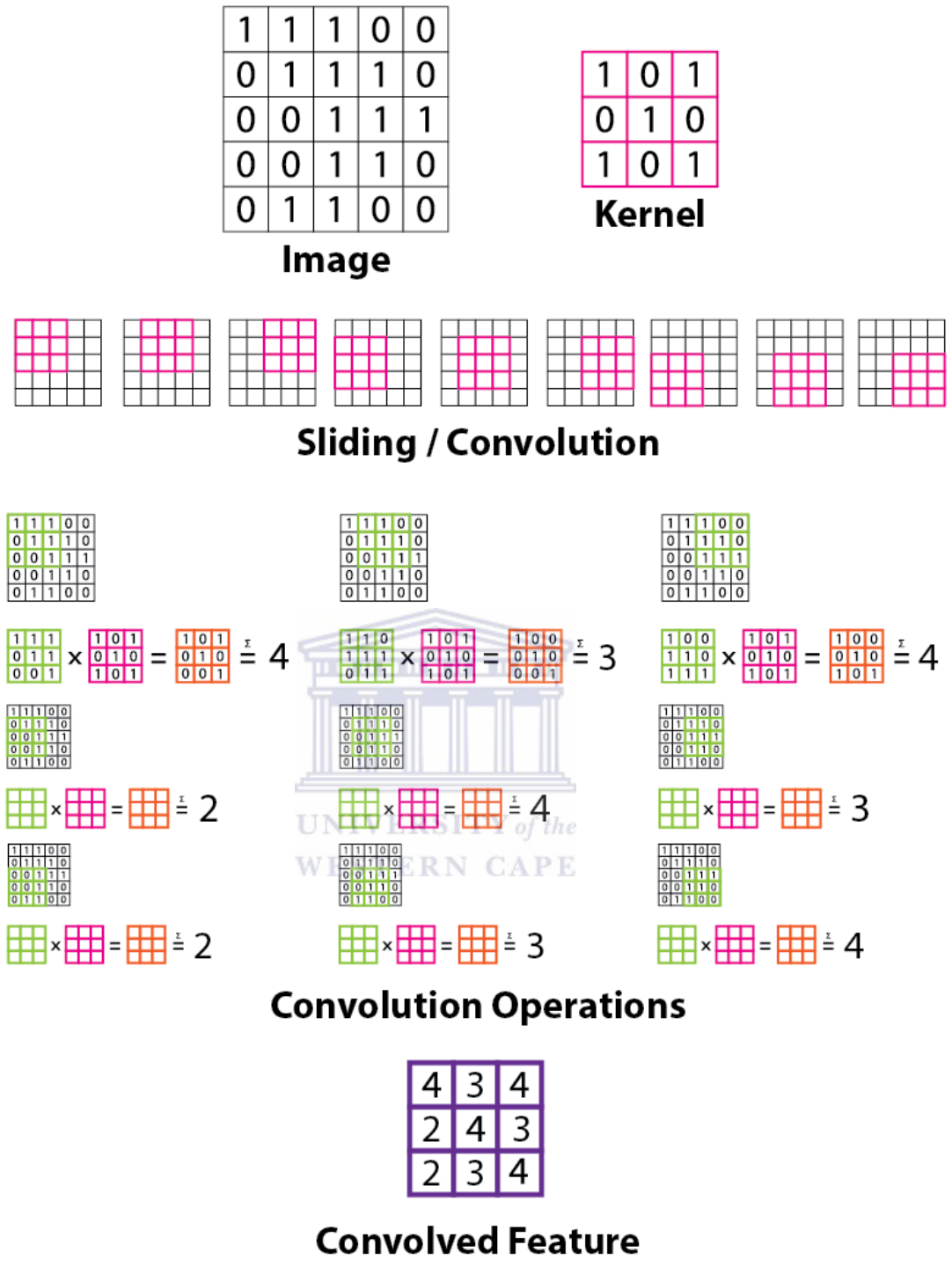
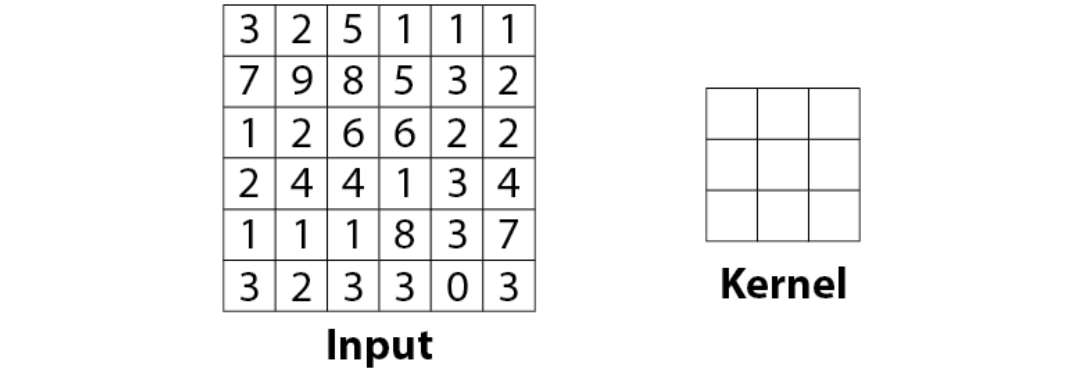


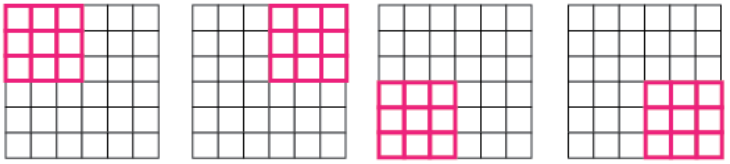
FIGURE B.1: Graphical representation of how convolution operations work.

B.2 Max- and Average- Pooling Operators

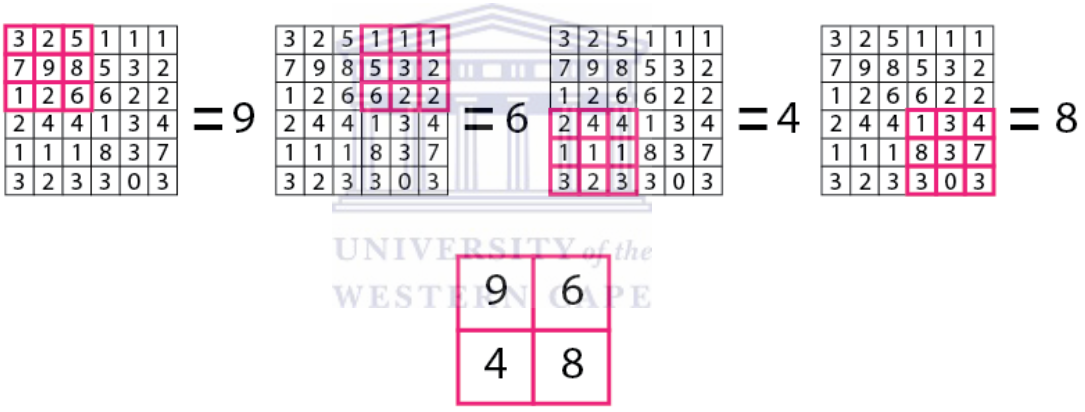
Convolutions may result in a large number of features and can be computationally expensive. Thus, a mechanism to reduce the feature set size has been devised, called

Max-Pooling and Mean- or Average-Pooling i.e., these mechanisms reduce the dimension of the input data. Each pooling layer consists of a kernel that has the size $n \times m$. The output of the pooling layer is dependant on the type of the pooling operation. In Max-Pooling and Average-Pooling the output is computed as shown in the Figure B.2. Max-Pooling involves sliding a kernel or grid across an image ensuring that these regions do not overlap and finding the maximum value within the kernel. Average- or Mean-Pooling operates under the same kernel procedure, however instead of using the maximum value, the sum of all the values is taken and divided by the kernel size. Step sizes may be added when sliding the kernel, the default rounding operation is floor but the ceiling operator may also be used.

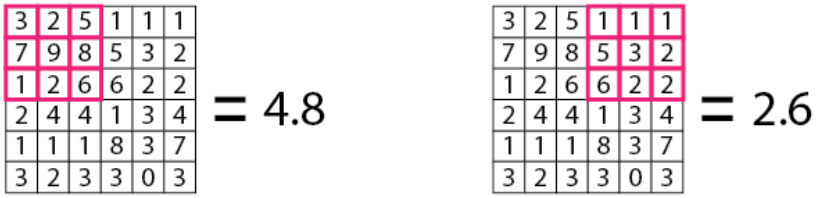


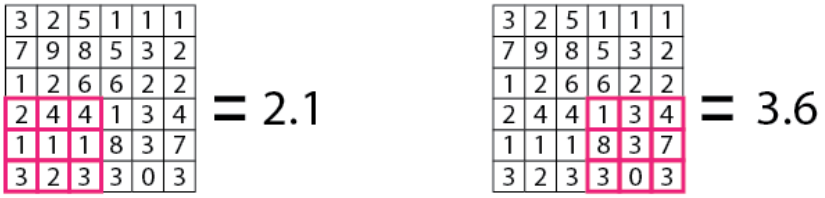


Sliding / Pooling



Max Pooling





Average / Mean Pooling

B.3 Activation & Transfer Functions

Before one can understand what an activation function is it is crucial that one understands the fundamental process of forward propagation in neural networks. In essence weights are multiplied by an input and compared with some threshold value which in turn activates a particular artificial neuron in a neural network. Before checking whether a neuron should be activated i.e., exceeds a certain threshold, it has a non-linear transfer function applied to it in order to transform it from a linear to non-linear space. An example of this transformation is the ReLU transfer function, $output = max(0, data)$ [89]. The importance of non-linear activation functions can be illustrated with a simple example.

Imagine 4 points A1, A2, B1 and B2, A1 is close to A2 and B1 is close to B2, however, A1 is far from B1 and B2 and consequently so is A2. Operating on this example, using a linear transform allows for limited outcomes as opposed to non-linear transforms. For instance, increasing the distance between A1 and A2 also increases the distance between B1 and B2. Similarly if the distance between A1 and B1 is decreased the distance between A2 and B2 also decreases. Many linear transforms exist but the results are mostly similar when compared to non-linear transforms. In contrast if a non-linear transform is used it is possible to increase the distance between A1 and A2 whilst simultaneously decreasing the distance between B1 and B2. Each time a non-linear transform is applied an increasingly complex relationship is formed between these points. In the context of deep learning each non-linear transfer function creates increasingly complex features with each layer.

As an alternative to the example above, The features of a many layers of pure linear transforms can be reduced to a single layer because of the fundamental way matrix multiplication works for linear transforms i.e., any number of linear transform matrices can be represented by a single transform matrix. This is why non-linear transfer functions are fundamentally important in deep learning. Figure B.3 visually illustrates the difference between linear and non-linear combinations.

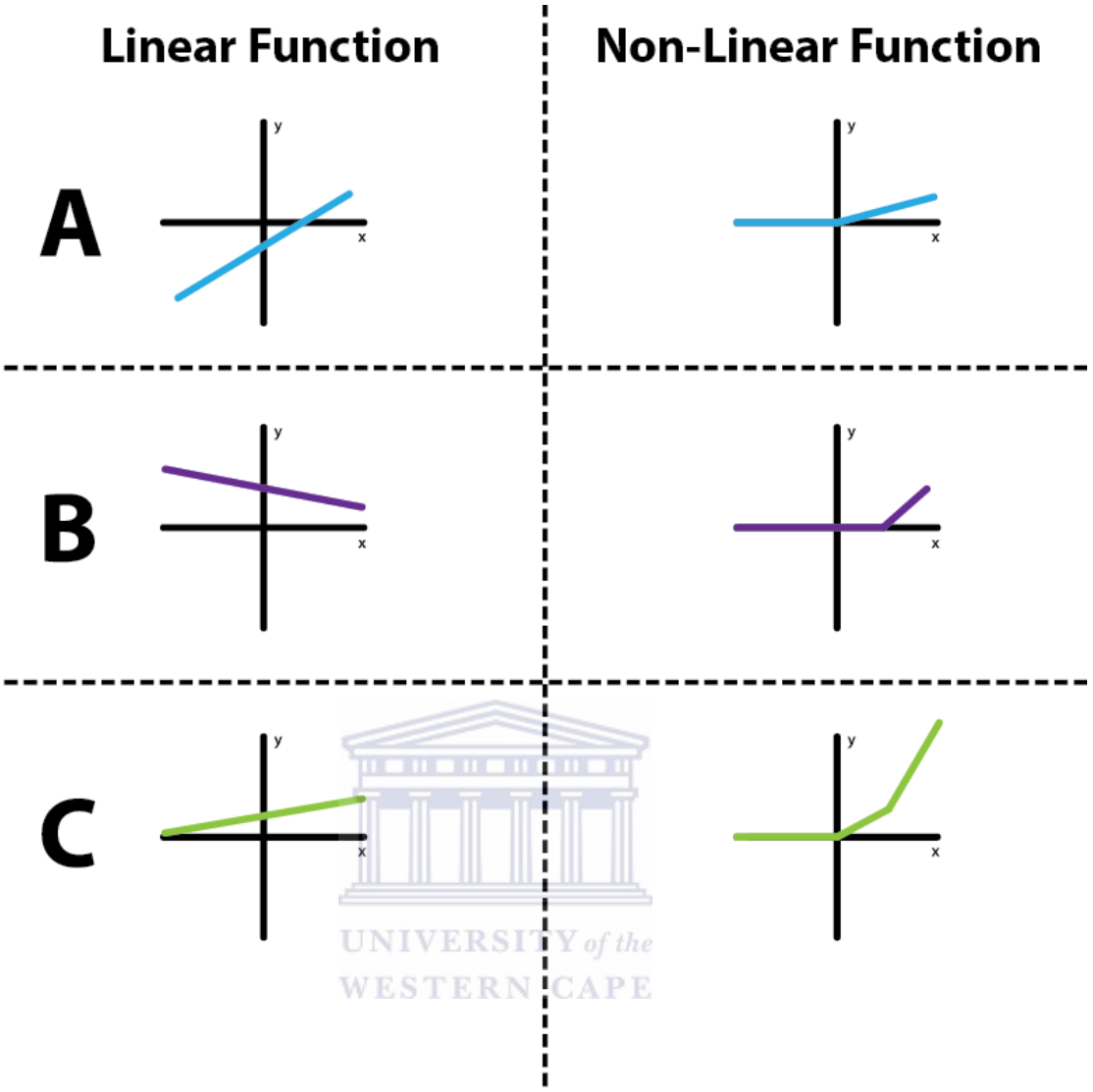


FIGURE B.3: Visual representation of combining linear vs non-linear functions.

B.4 Gradient Descent

Gradient descent is the arguably the most popular method to optimise neural networks, Three variants exist; batch gradient descent, stochastic gradient descent, and mini batch gradient descent and these are discussed in the following section [87]. Each of these variants refers to the amount of data used to compute the gradients for a particular objective function. The accuracy of parameter update is directly proportional to the amount of data used to compute the gradients. Thus, depending on the size of the dataset a suitable variant is chosen that provides reasonable accuracy as well as a suitable to time to update the parameter.

B.4.1 Batch Gradient Descent

Batch gradient descent (BGD) is known to as the default variant of gradient descent. BGD computes the gradient of the cost function across the entire dataset. BGD is slower than other variants because it has to calculate the gradients for each training example within the dataset with respect to every other example within the dataset. BGDs slowness comes with an advantage; it is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

B.4.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD), similarly, computes the gradient for the whole dataset however it differs from BGD in that parameter updates are done per training example i.e., each x and y tuple in the dataset. SGD is faster than BGD because it does not have to do redundant computations for each example. This approach does however have a caveat; high variance is known to occur with the frequent updates resulting in heavy fluctuation of the objective function. Because of these fluctuations SGD has the potential to find better local minima or even worse local minima. To mitigate the impact of this fluctuation studies have shown that SGD will perform similarly to BGD in terms of convergence when slowly decreasing the learning rate.

$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$

B.4.3 Mini-Batch Gradient Descent

Mini-batch gradient descent (MBGD) is a combination of both the ideas behind SGD and BGD. MBGD takes a smaller subset or mini-batch of size n of the dataset and computes the gradients for each example with respect to the mini-batch. The advantages of this variant is that it reduces the variance in the parameter updates which may lead to a more stable convergence when compared to that obtained using SGD.

$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

B.5 Back-propagation In Neural Networks

Back-propagation is a learning algorithm that allows for the observation of the change in error or cost with respect to weights and biases [90]. Algorithmically, back-propagation can be described as follows:

1. Setting the input layers values to the feature vector, x, where x is an input sample.

2. Perform the feedforward operation for each layer following the input layer. This is achieved by calculating the weighted input give by $z^l = w^l a^{l-1} + b^l$. Subsequently an activation function is applied to the weighted input, z, given by the formula $a^l = \sigma(z^l)$ in order to obtain the next layers activations.

3. Calculate the error for the output layer for the chosen cost function. The error vector is calculated as given by the formula $\delta^L = \nabla_a C \odot \sigma'(z^L)$. $\nabla_a C$ refers to the vector of partial derivatives $\frac{\partial C}{\partial a_j^L}$ where L is the output layer of the network, j refers to the index of the neuron in layer L and C refers to the cost function e.g. $MeanSquaredError(MSE)$ and $\sigma'(z^L)$ refers to the rate of change for the activation function σ with respect to z_j^L.

4. Back propagate the errors for each layer preceding the output layer computed by the formula $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$. $(w^{l+1})^T$ refers to the transposed weight matrix for layer $(l+1)$ and δ^{l+1} refers to the errors of the activations in layer $(l+1)$.

5. Compute the gradient of the cost function with respect to the weights give by the formulae $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

B.5.1 Back-propagation In Convolutional Neural Networks

Back-propagation in Convolutional Neural Networks are similar to that of standard Artificial Neural Networks, however, the differ distinctly because of the convolution and pooling operators [90].

When dealing with high-dimensional inputs such as images it is not feasible to connect all neurons (or pixels in the case of images) to neurons in the subsequent layer. Instead each neuron is only connected to the local region of the image. This is called the receptive field of the neuron and is the same size as the convolution layers filter. The depth of the output volume equals that of the input volume. For example if a grayscale image of 32×32 is given a filter of size 3×3, the result will be $3 \times 3 \times 1 = 9$ weights.

When using convolution layers connections are cut and shared across filters. This is achieved by using parameter sharing which is a scheme to control the total number of parameters in the network. If there exists a convolution layer that learns 96 filters (or

kernels), with a filter size of 11×11 and an output volume size of $55 \times 55 \times 1$ pixels after the first convolution, then the result would be $55 \times 55 \times 96 = 290400$ neurons and each filter would be $11 \times 11 \times 1 = 121$ weights and 1 bias. Together these calculation result in a total of $290400 \times 122 = 35428800$ parameters. Because of this high number of parameters it would be beneficial if this number could be reduced.

It turns out that this number can be dramatically reduced by making one reasonable assumption. This assumption is that if a feature is useful to compute at one position it should also be useful to compute at another location. Another way of looking at this would be to imagine that there is a 2-dimensional slice with 96 depth slices each of size 55x55. The depth slices are constrained to using only a single set of weights and biases, with this parameter sharing in place, the number of parameters is dramatically reduced to $96 \times 11 \times 11 \times 1 = 11616$ parameters. During back-propagation every neuron will compute the gradient for its weights but these gradients will update a single set of weights per slice.

Figure B.4 illustrates how parameters are shared. Figure B.4 shows that the feedforward operation is identical to that of a convolution in the case of a convolutional neural network, and the weights in this case are the filters.

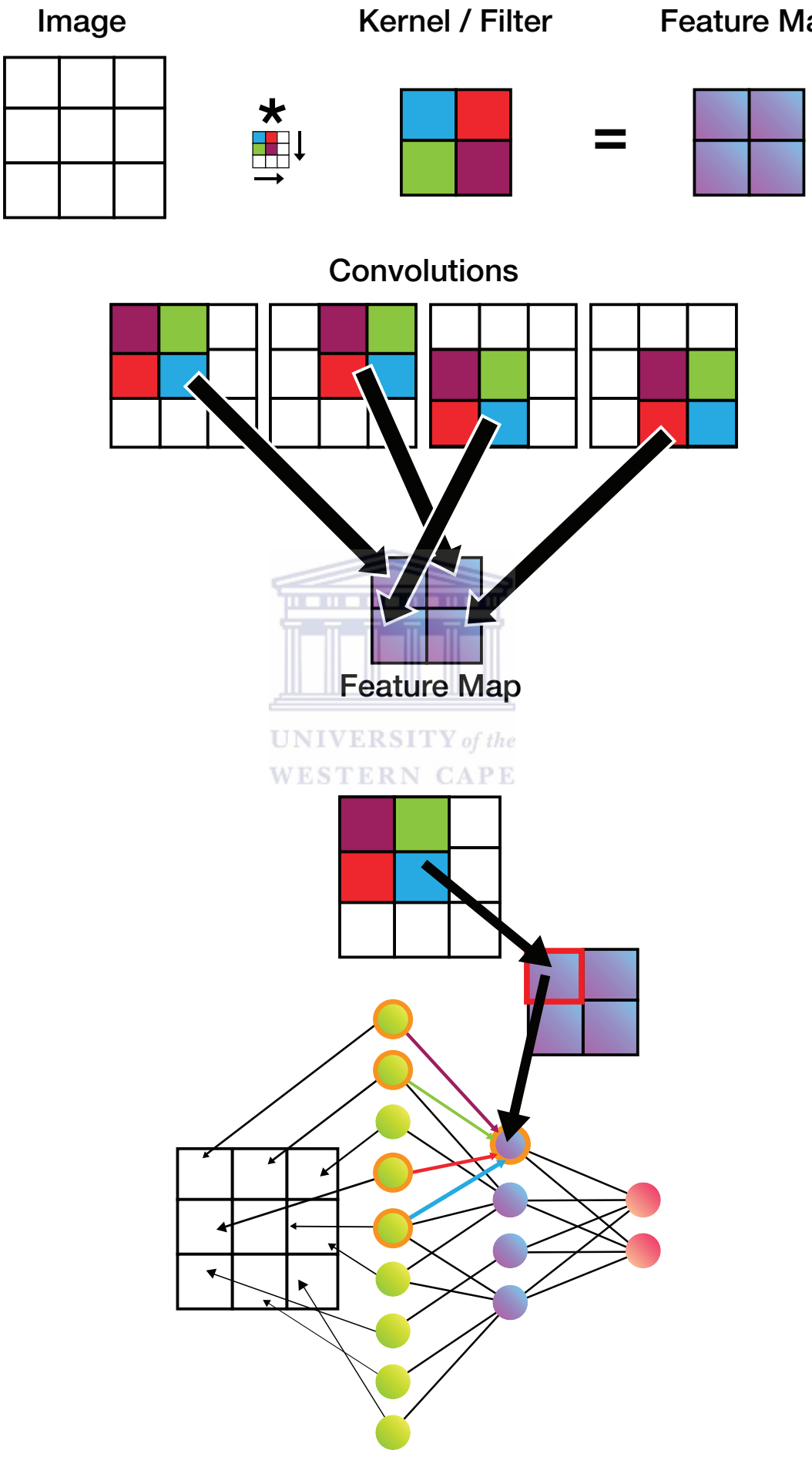


FIGURE B.4: Forward propagation in convolutional neural networks

Figure B.5 illustrates how to calculate the gradients for convolution layers by back-propagating the errors similar to a standard neural network. The only difference here is that instead of dot products the back-propagation in the convolutional neural network uses convolutions, as illustrated in Figure B.5 and Figure B.6. Figure B.5 in particular focuses on visually representing how the gradients are calculated between layers, with connections being highlighted with the appropriate colours in the kernel. The calculations are depicted in the big grey block as well as visually in the Full Convolution section in Figure B.6 and a full convolution is performed between the rotated kernel and the errors from the subsequent layer.

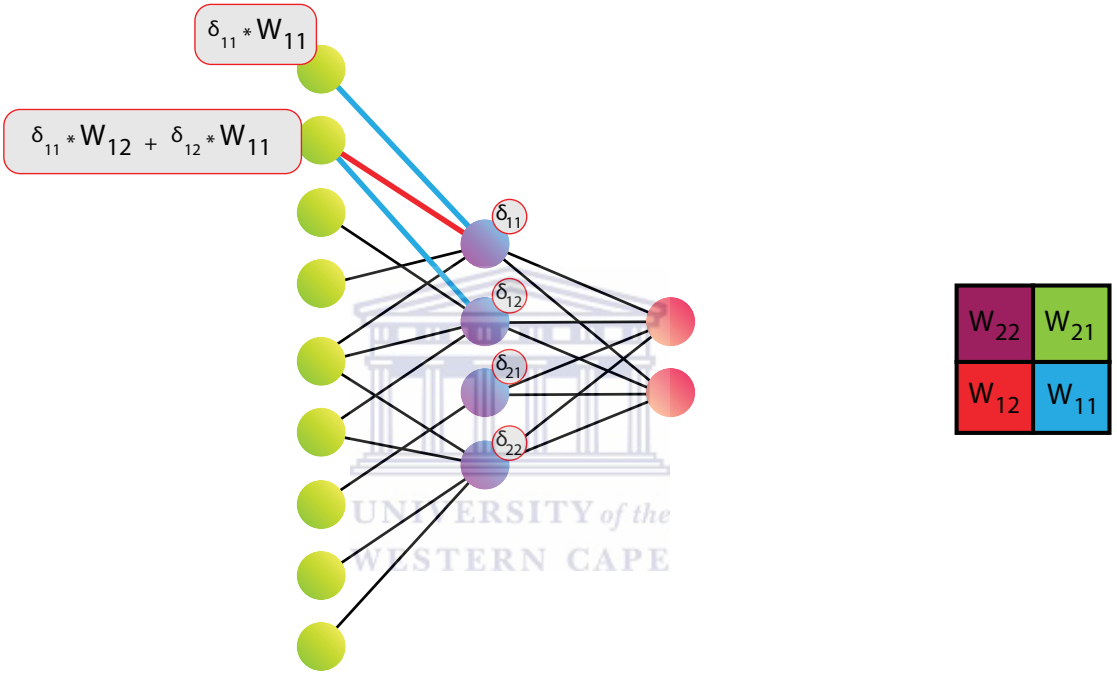


FIGURE B.5: Gradient calculation in convolutional neural networks - a visual depiction.

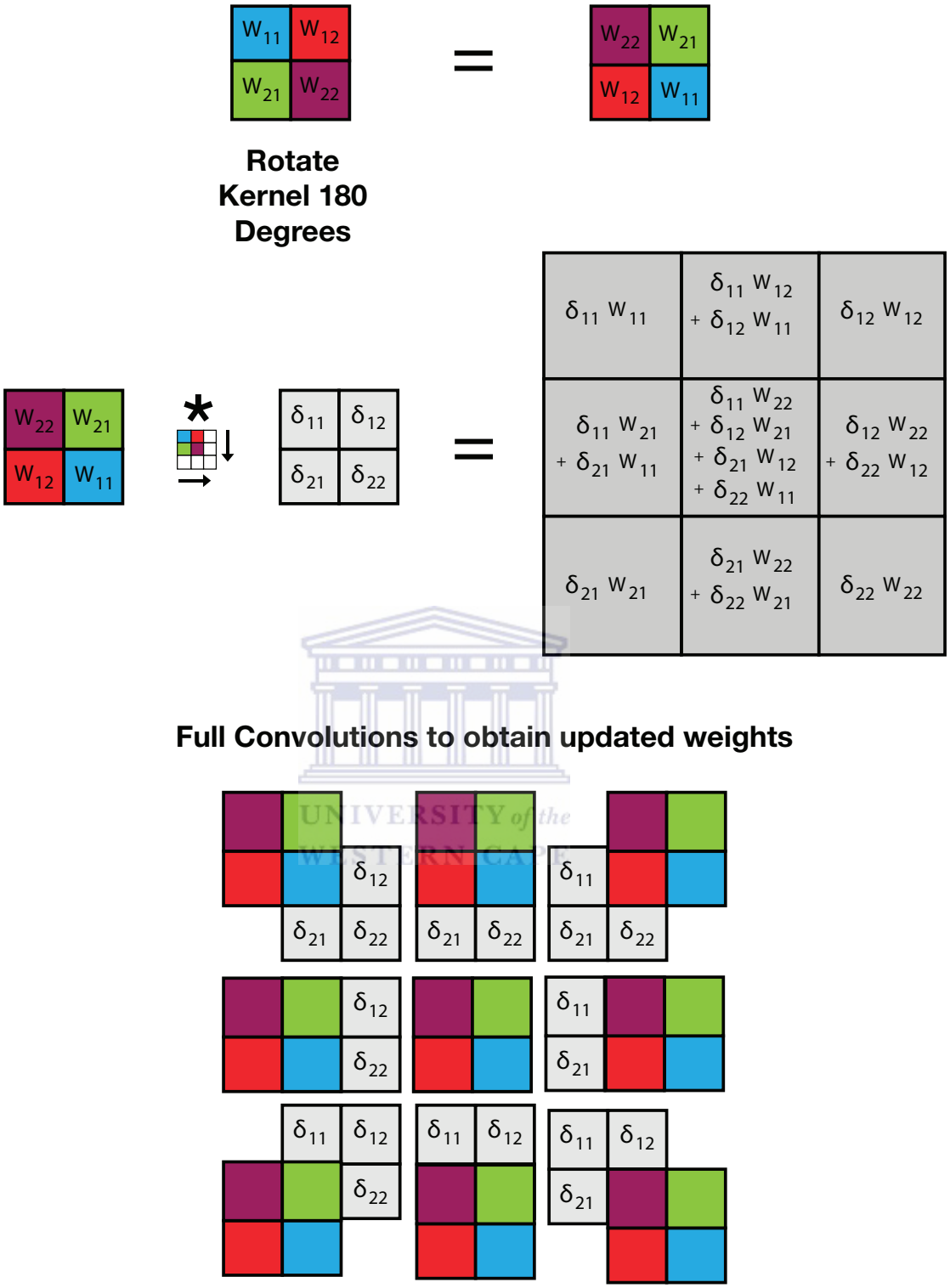


FIGURE B.6: How to calculate the gradients in a convolutional neural network.

Algorithmically the convolutional neural network back-propagation can be described as follows:

1) Setting the activation values, a^1 for the the input.

2) Perform the feed forward operation for each layer after the initial layer. This is achieved by computing $z^l_{x,y} = w^l_{x,y} \times \sigma(z^l_{x,y}) + b^l_{x,y}$ and $a^l_{x,y} = \sigma(z^l_{x,y})$.

3) Calculate the error for the output layer for a given cost function. The error vector, δ^L, is computed using the following formula: $\delta^L = \nabla_a C \odot \sigma'(z^L)$

4) Back propagate the errors for each layer preceding the output layer computed by the formula $\delta^l_{x,y} = \delta^{l+1}_{x,y} \times ROT180(w^{l+1}_{x,y})\sigma'(z^l_{x,y})$.

5) The gradient of the cost function is given by the following formula: $\frac{\partial C}{\partial w^l_{a,b}} = \delta^l_{a,b} \times \sigma(ROT180(z^{l-1}_{a,b}))$.

Appendix C

Appendix - Results & Analysis

C.1 Experiment 2 Results

TABLE C.1: HSO1 with the best activation (ReLU6)

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	118	1	0	1	0	0	98.33
2	0	120	0	0	0	0	100.00
3	0	3	84	0	19	14	70.00
4	0	3	1	116	0	0	96.67
5	0	0	0	0	113	7	94.17
6	0	1	1	1	12	105	87.50
Avg							91.11

TABLE C.2: HSO2 with the best activation (HardTanh)

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	103	0	0	3	14	0	85.83
2	0	98	0	9	13	0	81.67
3	0	0	107	13	0	0	89.17
4	0	1	22	75	19	3	62.50
5	0	0	0	8	106	6	88.33
6	1	0	0	2	56	61	50.83
Avg							76.39

TABLE C.3: HSO3 with the best activation (HardTanh)

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	117	0	0	1	0	2	97.50
2	26	94	0	0	0	0	78.33
3	0	0	110	3	4	3	91.67
4	10	0	0	109	0	1	90.83
5	0	0	0	0	119	1	99.17
6	0	0	4	8	30	78	65.00
Avg							87.08

TABLE C.4: HSO4 with the best activation (HardTanh)

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	78	29	0	7	4	2	65.00
2	0	114	0	3	3	0	95.00
3	1	0	84	0	19	16	70.00
4	16	0	5	69	26	4	57.50
5	1	3	5	12	76	23	63.33
6	7	3	8	14	41	47	39.17
Avg							65.00

TABLE C.5: HSO5 with the best activation (ReLU)

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	70	23	0	0	5	22	58.33
2	33	50	8	16	2	11	41.67
3	0	0	115	0	0	5	95.83
4	5	18	1	71	6	19	59.17
5	5	1	0	0	108	6	90.00
6	0	9	4	23	14	70	58.33
Avg							67.22

TABLE C.6: OR with the best activation (ReLU)

	Orientation					
	1	**2**	**3**	**4**	**5**	**Average %**
1	619	0	21	0	80	92.44
2	13	687	14	2	4	95.42
3	17	14	685	4	0	95.14
4	1	16	5	692	6	96.11
5	51	22	0	2	645	89.58
Avg						92.44

C.2 Experiment 3 Results

TABLE C.7: HSO1 with the best learnable filter count and fully connected layer node count

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	117	0	0	3	0	0	97
2	0	114	0	0	6	0	95
3	0	3	85	0	19	13	70
4	0	0	1	117	2	0	97
5	0	0	0	1	113	6	94
6	0	2	1	1	20	96	80
Avg							89

TABLE C.8: HSO2 with the best learnable filter count and fully connected layer node count

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	108	0	0	0	12	0	90
2	0	101	0	6	13	0	84
3	0	0	113	6	0	1	94
4	3	0	19	80	16	2	66
5	0	0	0	3	106	11	88
6	1	0	0	1	52	66	55
Avg							79

TABLE C.9: HSO3 with the best learnable filter count and fully connected layer node count

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	117	0	0	2	0	1	97
2	23	97	0	0	0	0	80
3	0	0	108	0	0	5	90
4	3	0	0	115	1	1	95
5	0	0	0	0	119	1	99
6	0	0	5	11	27	77	64
Avg							87

TABLE C.10: HSO4 with the best learnable filter count and fully connected layer node count

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	63	30	0	11	10	6	52
2	0	116	0	1	3	0	96
3	1	0	88	0	17	14	73
4	18	1	11	65	20	5	54
5	1	1	6	11	68	33	56
6	9	0	10	10	45	46	38
Avg							61

TABLE C.11: HSO5 with the best learnable filter count and fully connected layer node count

	Shape						
	1	**2**	**3**	**4**	**5**	**6**	**Average %**
1	69	20	0	1	9	21	57
2	39	40	9	22	1	9	33
3	0	0	115	0	0	5	95
4	6	16	1	79	5	13	65
5	1	0	0	0	108	11	90
6	0	7	1	28	13	71	59
Avg							66

TABLE C.12: OR with the best learnable filter count and fully connected layer node count

	Orientation					
	1	**2**	**3**	**4**	**5**	**Average %**
1	611	0	18	1	90	84
2	11	673	29	5	2	93
3	11	23	682	4	0	94
4	1	13	13	685	8	95
5	54	18	0	0	648	90
Avg						91

C.3 Experiment 4 Results

TABLE C.13: HSO1 accuracies for variable convolution and pooling filter sizes - desktop context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	80.14	87.08	89.17	87.36
	3 5	83.61	88.33	88.75	86.67
	3 6	84.31	89.28	87.50	86.39
	4 3	80.69	86.25	88.61	86.11
	4 5	83.19	88.89	89.17	86.39
	4 6	85.69	89.72	86.67	86.53
	5 3	81.81	85.33	90.28	86.11
	5 4	81.11	87.64	89.58	85.97
	5 6	83.33	89.58	87.08	87.64
	6 3	80.42	86.53	89.31	86.11
	6 4	81.81	87.36	89.86	86.53
	6 5	84.58	89.86	88.89	85.83

TABLE C.14: HSO1 accuracies for variable convolution and pooling filter sizes - embedded context

Convolution	Pooling 2 2	Pooling 3 4	Pooling 5 3	Pooling 6 2
3 4	80.00	86.53	85.69	87.08
3 5	83.47	88.75	85.28	90.97
3 6	85.28	86.94	83.61	86.81
4 3	80.69	84.31	85.28	87.92
4 5	81.25	88.75	88.75	85.97
4 6	82.77	86.94	88.05	87.22
5 3	84.44	85.00	88.19	83.75
5 4	83.75	87.36	89.03	86.25
5 6	82.91	84.86	80.83	85.83
6 3	81.25	86.25	84.86	87.36
6 4	82.77	88.19	88.47	84.58
6 5	83.33	88.19	86.94	85.83

TABLE C.15: HSO2 accuracies for variable convolution and pooling filter sizes - desktop context

Convolution	Pooling 2 2	Pooling 3 4	Pooling 5 3	Pooling 6 2
3 4	66.94	77.36	77.92	76.53
3 5	62.64	75.00	74.17	74.44
3 6	63.19	74.86	80.28	75.83
4 3	64.86	76.58	78.89	77.36
4 5	64.17	74.17	78.06	77.50
4 6	61.39	76.81	81.67	76.11
5 3	62.78	75.97	77.92	76.53
5 4	60.83	77.78	79.03	74.17
5 6	63.06	74.58	80.69	74.44
6 3	63.33	76.25	77.92	76.11
6 4	62.78	75.28	79.72	74.17
6 5	64.03	75.97	80.14	78.61

TABLE C.16: HSO2 accuracies for variable convolution and pooling filter sizes - embedded context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	67.36	75.56	74.03	75.56
	3 5	66.53	71.94	73.33	74.44
	3 6	65.42	77.08	78.06	73.89
	4 3	64.86	75.28	74.44	73.75
	4 5	65.97	75.56	76.25	73.33
	4 6	62.78	70.42	75.97	70.14
	5 3	66.81	75.56	76.39	75.14
	5 4	65.83	74.44	76.39	74.17
	5 6	68.19	73.61	76.11	78.75
	6 3	64.44	76.25	76.67	75.97
	6 4	67.08	77.50	76.67	75.28
	6 5	65.97	73.33	80.28	76.11

TABLE C.17: HSO3 accuracies for variable convolution and pooling filter sizes - desktop context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	80.83	87.82	89.44	88.33
	3 5	77.92	88.75	86.81	87.50
	3 6	73.89	90.56	86.81	88.33
	4 3	75.00	86.94	89.58	84.72
	4 5	73.06	90.42	88.06	87.22
	4 6	77.08	88.89	87.36	83.47
	5 3	79.86	86.39	87.92	85.97
	5 4	76.94	88.33	87.92	84.86
	5 6	75.42	88.75	86.53	84.86
	6 3	80.00	86.53	87.92	86.81
	6 4	69.17	88.75	88.75	82.08
	6 5	77.64	89.86	87.64	86.53

TABLE C.18: HSO3 accuracies for variable convolution and pooling filter sizes - embedded context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	77.78	85.00	88.61	89.44
	3 5	78.47	88.06	82.08	88.06
	3 6	76.94	88.19	85.42	88.19
	4 3	80.00	86.25	86.94	83.47
	4 5	80.69	87.78	80.14	83.61
	4 6	74.44	88.33	86.25	84.86
	5 3	79.58	84.17	85.56	84.17
	5 4	80.00	87.78	89.58	85.42
	5 6	75.14	89.31	87.78	84.86
	6 3	78.47	82.78	87.08	85.14
	6 4	76.11	85.97	85.83	84.44
	6 5	77.22	88.33	84.31	80.69

TABLE C.19: HSO4 accuracies for variable convolution and pooling filter sizes - desktop context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	48.89	61.53	60.42	53.33
	3 5	45.83	61.11	65.00	57.36
	3 6	50.14	57.64	60.42	54.58
	4 3	43.89	60.97	65.56	53.75
	4 5	49.31	61.39	66.25	53.75
	4 6	43.75	57.36	62.36	55.28
	5 3	50.69	63.47	62.50	55.00
	5 4	50.28	62.78	62.50	52.78
	5 6	46.11	59.58	60.42	55.14
	6 3	45.97	63.06	64.72	54.58
	6 4	47.08	60.42	60.83	54.58
	6 5	49.31	56.53	63.47	55.69

TABLE C.20: HSO4 accuracies for variable convolution and pooling filter sizes - embedded context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	51.81	59.17	56.94	54.58
	3 5	50.56	61.94	66.94	58.75
	3 6	49.44	59.86	58.61	55.97
	4 3	50.56	57.36	59.72	55.00
	4 5	50.97	59.03	62.36	54.44
	4 6	45.83	60.42	58.89	56.94
	5 3	52.08	57.36	63.61	52.50
	5 4	48.33	61.81	59.31	56.39
	5 6	56.53	61.11	60.97	51.39
	6 3	49.86	54.17	60.69	50.42
	6 4	51.94	54.72	59.86	53.33
	6 5	48.19	59.03	62.22	56.53

TABLE C.21: HSO5 accuracies for variable convolution and pooling filter sizes - desktop context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	59.44	67.22	68.75	64.31
	3 5	57.92	68.33	66.81	63.75
	3 6	58.61	67.92	63.75	63.06
	4 3	60.14	67.08	66.67	65.97
	4 5	59.86	68.06	66.11	68.19
	4 6	50.00	64.72	68.06	62.64
	5 3	60.42	68.06	65.97	64.72
	5 4	59.86	66.25	66.81	63.89
	5 6	58.47	63.61	64.44	63.33
	6 3	58.47	65.28	65.14	64.58
	6 4	58.47	66.11	65.00	64.03
	6 5	61.67	66.94	68.61	67.22

TABLE C.22: HSO5 accuracies for variable convolution and pooling filter sizes - embedded context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	58.06	61.94	60.56	58.89
	3 5	63.33	61.94	55.42	55.69
	3 6	61.39	61.81	62.22	60.28
	4 3	61.25	60.42	65.97	64.72
	4 5	57.08	62.22	65.42	61.81
	4 6	61.81	61.94	62.08	57.78
	5 3	59.17	59.03	62.92	57.92
	5 4	57.78	60.00	54.58	59.86
	5 6	59.31	62.92	63.06	59.72
	6 3	58.19	56.25	60.28	62.92
	6 4	61.94	64.03	61.39	63.61
	6 5	56.53	65.00	57.36	58.75

TABLE C.23: OR accuracies for variable convolution and pooling filter sizes - desktop context

		Pooling			
		2 2	**3 4**	**5 3**	**6 2**
Convolution	**3 4**	85.64	90.89	90.67	91.06
	3 5	76.92	90.56	89.72	89.28
	3 6	82.78	90.06	90.97	90.00
	4 3	84.94	89.97	90.61	90.33
	4 5	83.58	90.72	90.08	90.53
	4 6	84.28	90.47	89.53	85.53
	5 3	85.64	90.83	91.19	90.89
	5 4	86.06	91.06	92.64	91.08
	5 6	80.33	91.50	91.39	89.64
	6 3	87.39	90.11	91.75	91.00
	6 4	84.81	90.89	92.81	91.19
	6 5	84.06	90.64	90.86	89.53

TABLE C.24: OR accuracies for variable convolution and pooling filter sizes - embedded context

Convolution	Pooling 2 2	Pooling 3 4	Pooling 5 3	Pooling 6 2
3 4	88.19	89.67	91.39	91.22
3 5	88.39	90.47	88.28	91.14
3 6	87.83	91.19	91.64	90.61
4 3	88.00	89.97	90.44	90.42
4 5	89.53	91.75	88.53	91.92
4 6	88.61	90.92	89.28	90.86
5 3	88.50	89.50	91.06	90.58
5 4	89.53	90.42	90.94	90.28
5 6	89.11	90.47	91.03	89.00
6 3	88.86	89.33	90.03	89.94
6 4	88.56	89.33	91.61	91.44
6 5	88.94	92.47	90.89	90.75

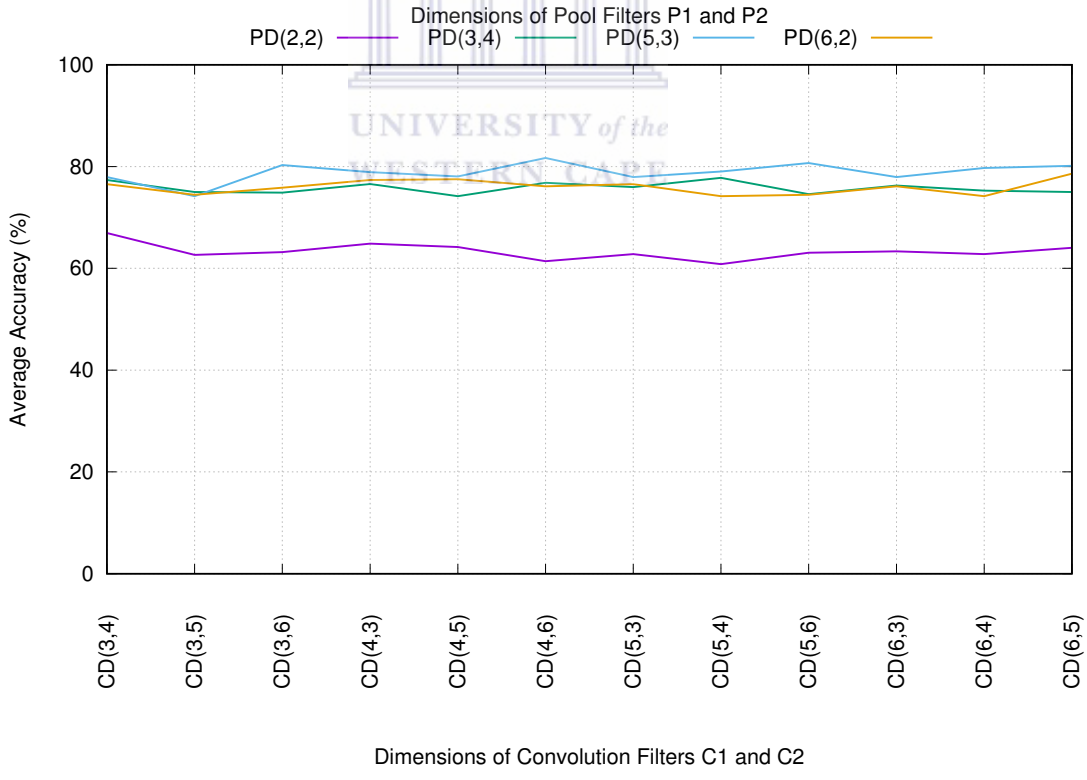


FIGURE C.1: HSO2 NRS accuracies for varying convolution and pooling filter dimensions

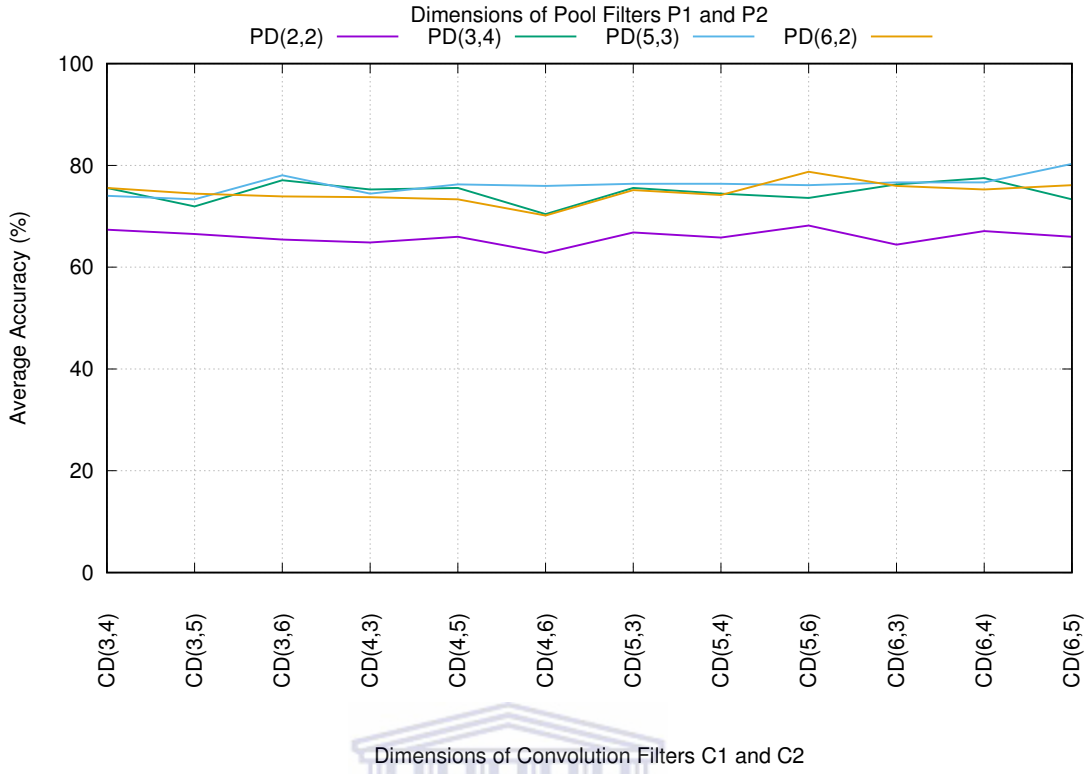


FIGURE C.2: HSO2 ES accuracies for varying convolution and pooling filter dimensions

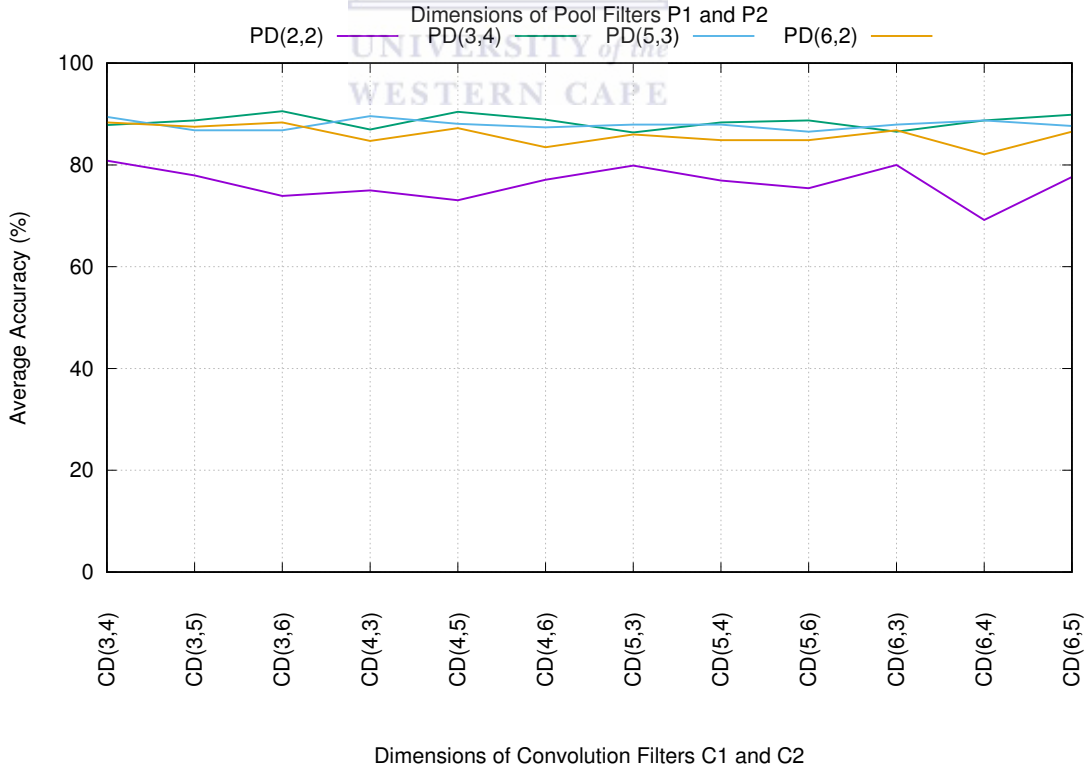


FIGURE C.3: HSO3 NRS accuracies for varying convolution and pooling filter dimensions

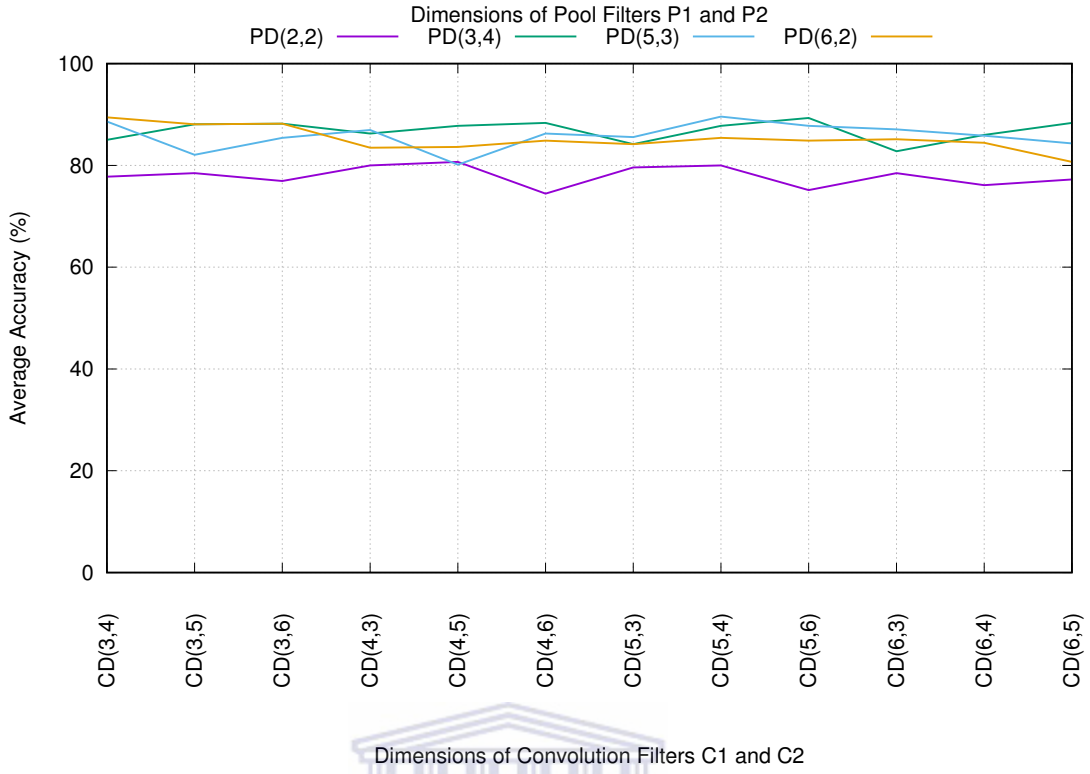


FIGURE C.4: HSO3 ES accuracies for varying convolution and pooling filter dimensions

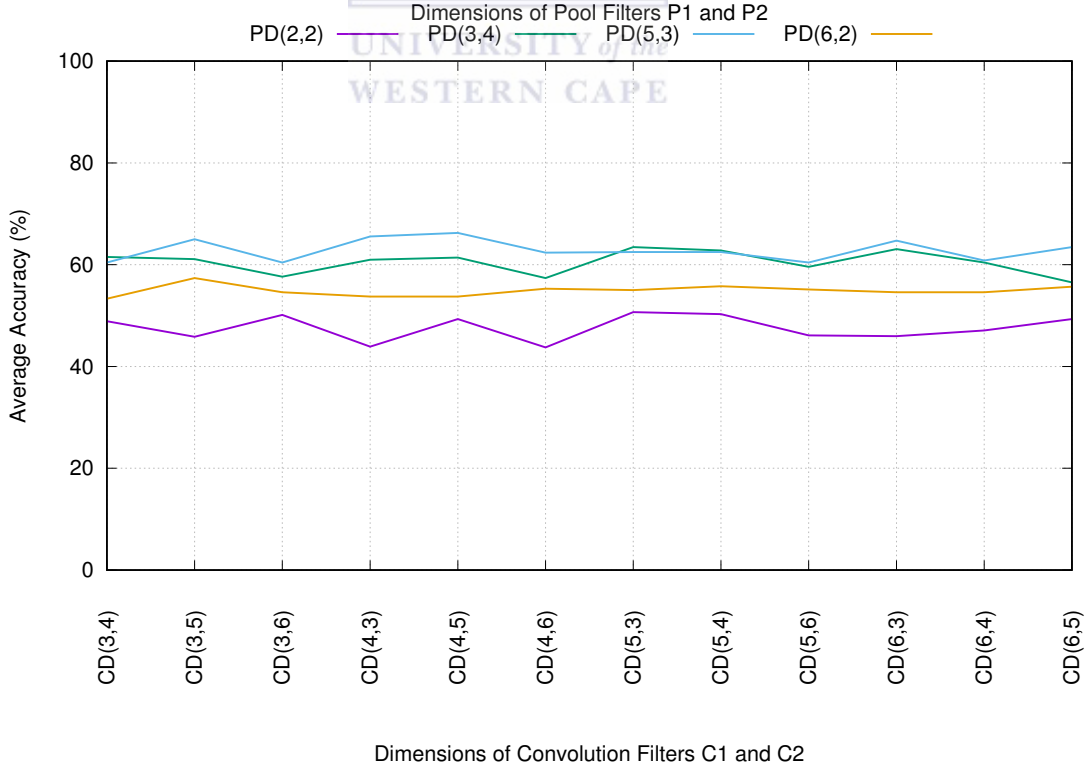


FIGURE C.5: HSO4 NRS accuracies for varying convolution and pooling filter dimensions

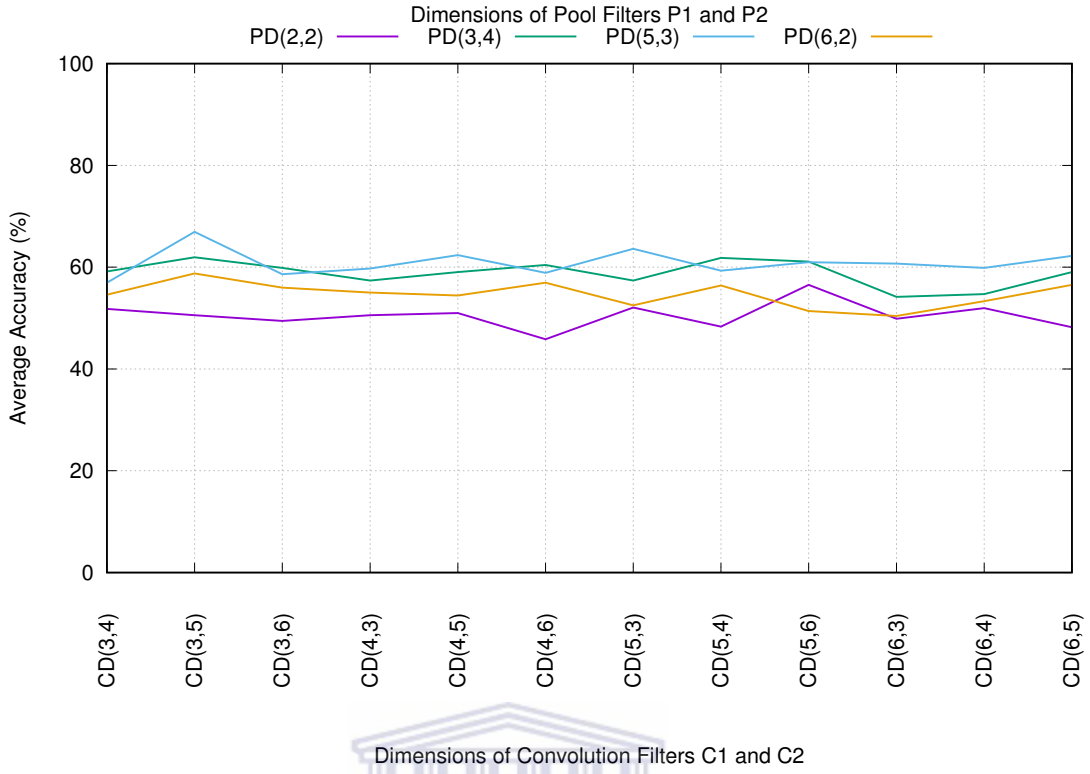


FIGURE C.6: HSO4 ES accuracies for varying convolution and pooling filter dimensions

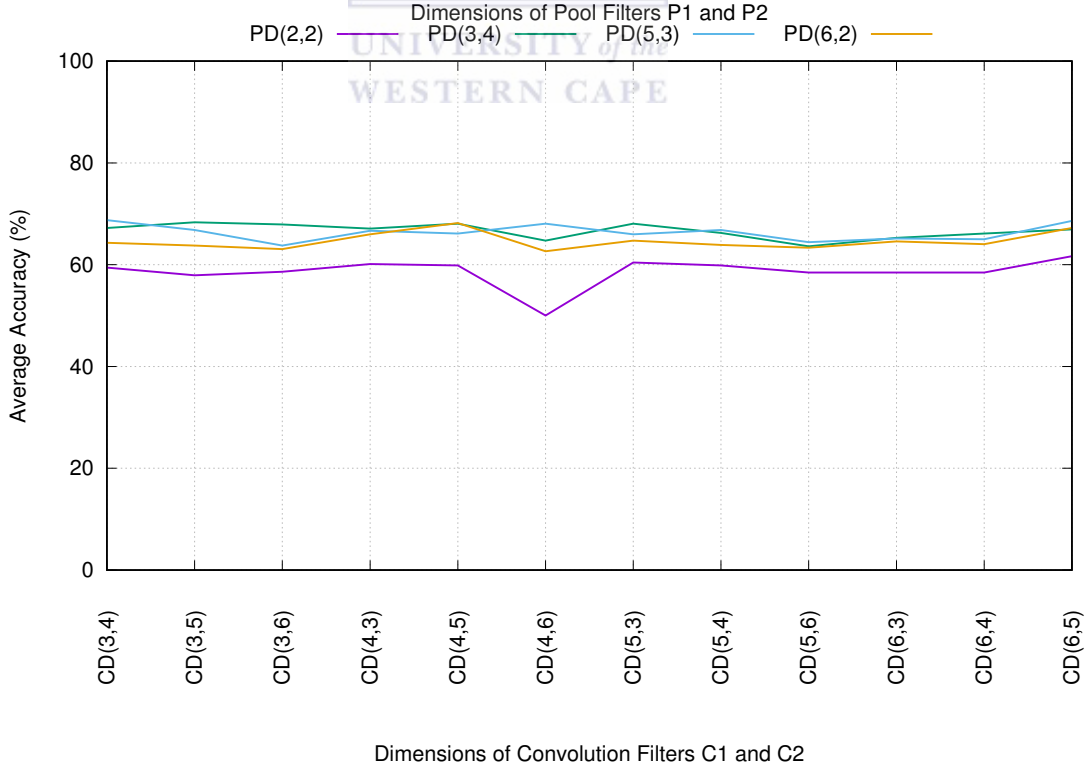


FIGURE C.7: HSO5 NRS accuracies for varying convolution and pooling filter dimensions

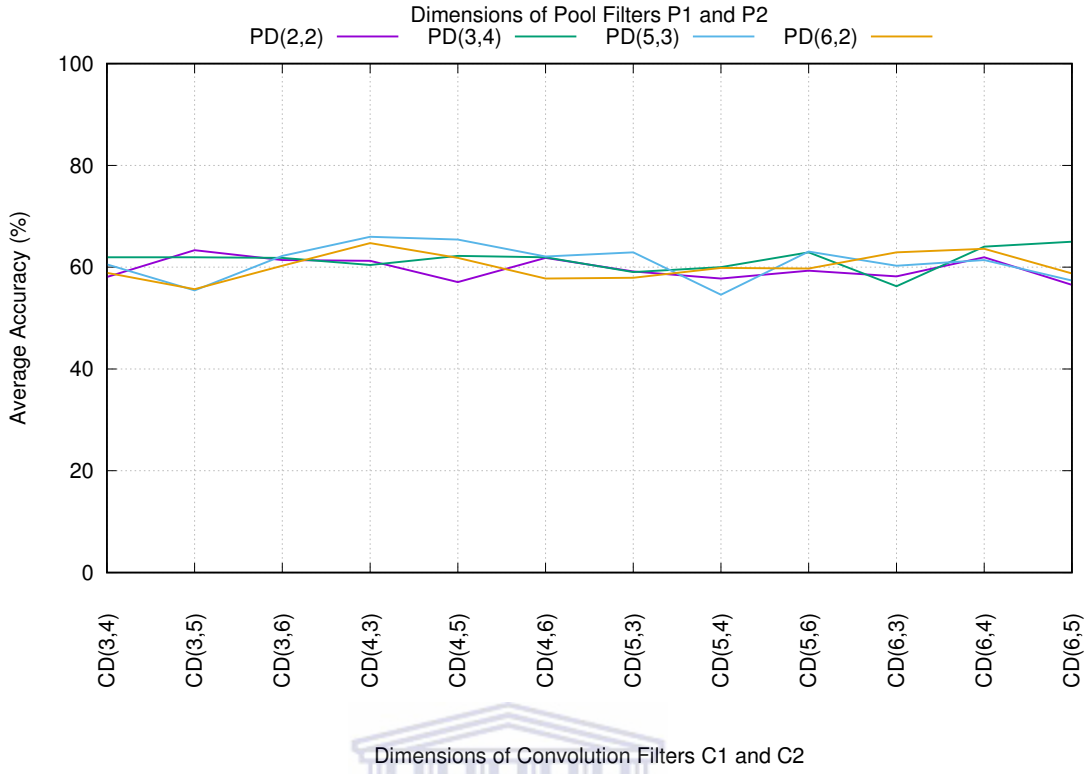


FIGURE C.8: HSO5 ES accuracies for varying convolution and pooling filter dimensions

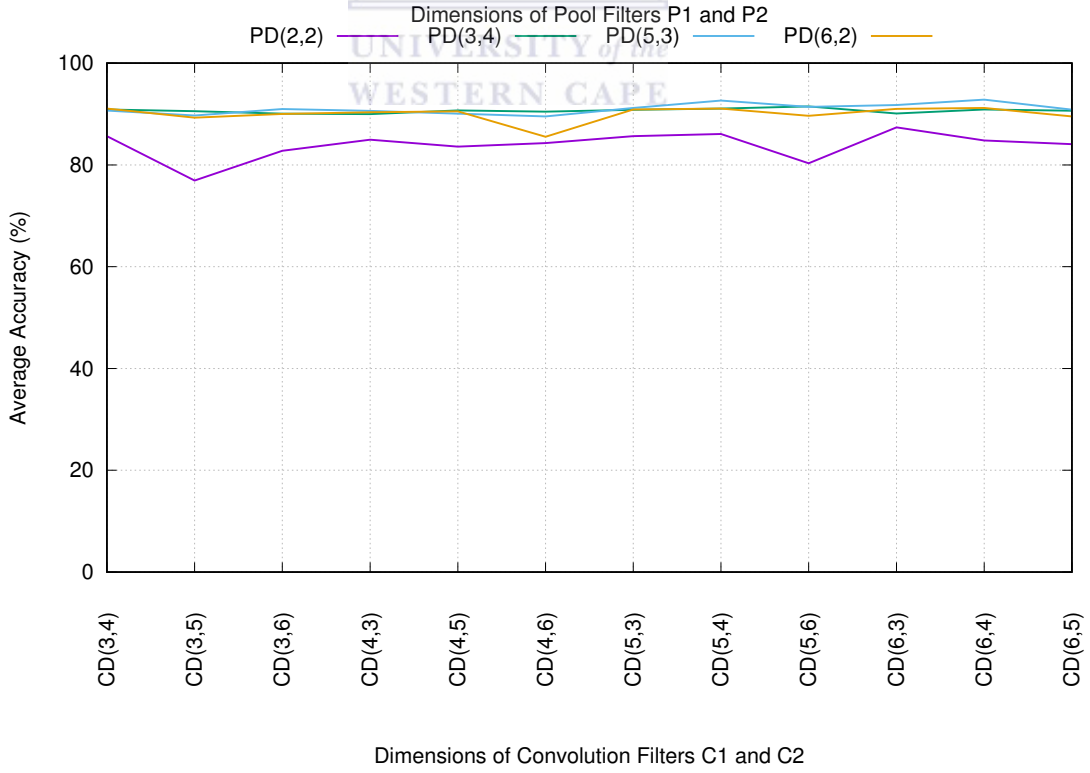


FIGURE C.9: OR NRS accuracies for varying convolution and pooling filter dimensions

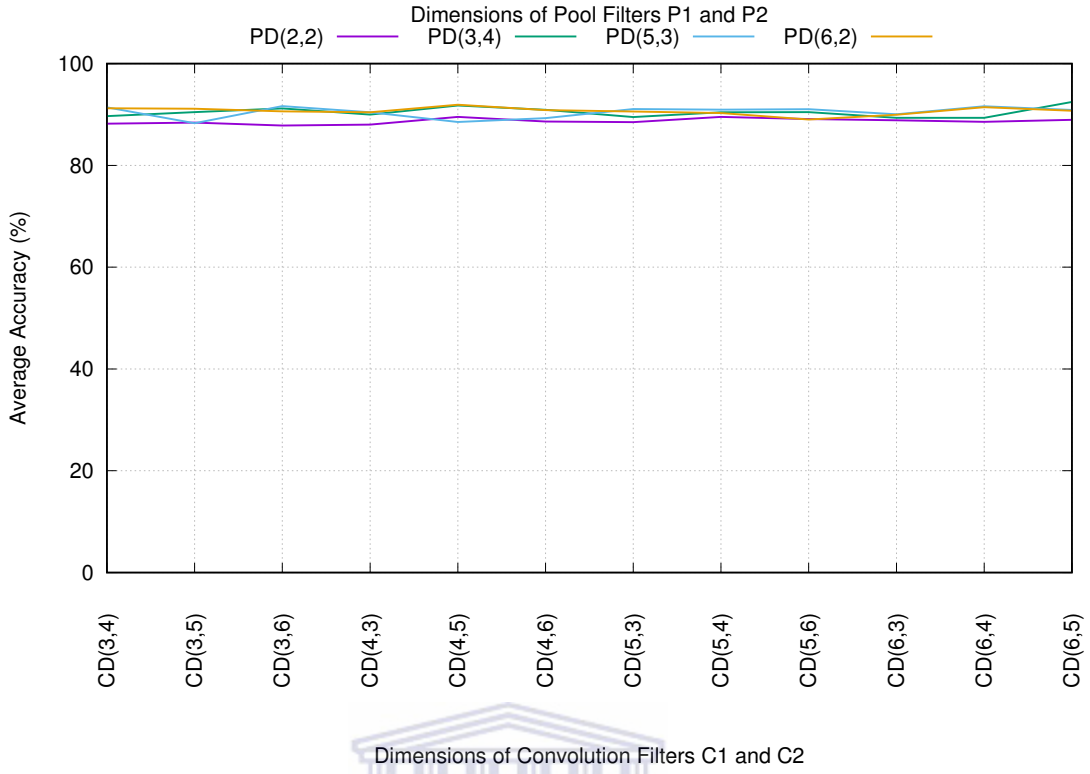


FIGURE C.10: OR ES accuracies for varying convolution and pooling filter dimensions

C.4 Experiment 5 Results

TABLE C.25: Accuracy of final 2-stage network; each column represents the accuracy results of a single hand shape classifier. These results pertain to the best embedded systems models

Shape	**Orientation**					**Avg**
	1	**2**	**3**	**4**	**5**	
1	63.87	88.33	80.00	55.83	45.83	66
2	100.00	84.17	71.67	90.00	39.17	77
3	58.33	80.83	85.83	75.00	80.83	76
4	90.83	75.83	91.67	60.00	60.83	75
5	91.67	84.17	93.33	75.00	66.67	82
6	72.50	58.33	70.00	45.83	68.33	63
Average	79	78	82	67	60	**73.4**

TABLE C.26: Accuracy of final 2-stage network; each column represents the accuracy results of a single hand shape classifier. These results pertain to the best desktop model

Shape	**Orientation**					**Avg**
	1	**2**	**3**	**4**	**5**	
1	80.67	91.67	90.00	42.50	41.67	69
2	90.83	85.83	84.17	93.33	49.17	80
3	51.67	78.33	87.50	74.17	70.83	72
4	91.67	64.16	93.33	60.83	58.33	73
5	93.33	81.67	99.17	72.50	65.00	82
6	70.83	57.50	71.67	45.83	59.17	60
Average	79	76	87	64	57	**73.2**

TABLE C.27: Classification instances errors per subject as a result of rejection at the first stage (orientation classifier) or the second stage (any of the hand shape classifiers)

Test Subject	**Embedded**			**Desktop**		
	Stage 1	**Stage 2**	**Total**	**Stage 1**	**Stage 2**	**Total**
4	183	59	242	161	61	222
5	191	110	301	187	119	306
8	214	23	237	222	30	252
9	99	75	174	98	83	181

TABLE C.28: Percentage classification errors per subject as a result of rejection at the first stage (orientation classifier) or the second stage (any of the hand shape classifiers)

Test Subject	**Embedded Error (%)**			**Desktop Error (%)**		
	Stage 1	**Stage 2**	**Overall**	**Stage 1**	**Stage 2**	**Overall**
4	8.2	25.4	33.6	8.5	22.4	30.8
5	15.3	26.5	41.8	16.5	26.0	42.5
8	3.2	29.7	32.9	4.2	30.8	35.0
9	10.4	13.8	24.2	11.5	13.6	25.1
Overall	7.4	19.1	26.5	8.1	18.6	26.7

C.5 Convolutional Neural Network Classification Speed Results

TABLE C.29: Recognition speeds for the best ES configuration

Classification Time (seconds)		
	iPhone 5C	**iPhone 6+**
Iteration	Time (s)	Time (s)
1	0.0269	0.0075
2	0.0279	0.0075
3	0.0275	0.0075
4	0.0267	0.0075
5	0.0267	0.0075
6	0.0268	0.0076
7	0.0264	0.0075
8	0.0265	0.0075
9	0.0266	0.0074
10	0.0266	0.0075
11	0.0265	0.0076
12	0.0265	0.0074
13	0.0265	0.0073
14	0.0267	0.0073
15	0.0268	0.0076
16	0.0265	0.0072
17	0.0265	0.0073
18	0.0266	0.0075
19	0.0265	0.0074
20	0.0265	0.0073
Average (s)	0.0267	0.0074
FPS	37	135

Bibliography

[1] R. Fukui, M. Watanabe, M. Shimosaka, and T. Sato. *Hand Shape Classification with a Wrist Contour Sensor (Comparison of Feature Types and Observation of Resemblance among Subjects)*, volume 88, pages 939–949. Springer, 2013.

[2] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Ciresan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *ICISPA '11*, 2011.

[3] S. J. Nowlan and J. C. Platt. A convolutional neural network hand tracker. In *Advances in Neural Information Processing Systems*, pages 901–908, 1995.

[4] University of the Witwatersrand. Deaf facts, 2015. URL http://www.hihopes.co.za.

[5] M. Marschark. *Psychological Development of Deaf Children*. Oxford University Press On Demand, 1997.

[6] R. C. Johnson. Sign language and the concept of deafness in a traditional Yucatec Mayan village. In C. J. Erting, R. C. Johnson, D. L. Smith, and B. D. Snider, editors, *The Deaf Way: Perspectives from the International Conference on Deaf Culture*, pages 102–109. Gallaudet University Press, Washington, D.C, 1994.

[7] Council of Europe. Facts on sign language, 2015. URL edl.ecml.at/FAQ/FAQsonsignlanguage/tabid/2741/Default.aspx.

[8] K. Asmal and W. James. Education and democracy in South Africa today. *Daedalus*, pages 185–204, 2001.

[9] M. Glaser and W. D. Tucker. Telecommunications bridging between Deaf and hearing users in South Africa. In *Proceedings of the Conference and Workshop on Assistive Technologies for People with Vision and Hearing Impairments*, Granada, Spain, 2004.

[10] W. C. Stokoe. Sign language structure: An outline of the visual communication systems of the American deaf. studies in linguistics: Occasional papers. Technical

report, Buffalo: Dept. of Anthropology and Linguistics, University of Buffalo, Tech. Rep. 8, 1960.

[11] P. Li. Hand shape estimation for South African sign language. Master's thesis, Department of Computer Science, University of the Western Cape, 2010.

[12] I. Achmed. Upper body pose recognition and estimation towards the translation of South African sign language. Master's thesis, Department of Computer Science, University of the Western Cape, 2010.

[13] D. Brown. Upper body pose recognition and estimation towards the translation of South African sign language. Master's thesis, Department of Computer Science, University of the Western Cape, 2013.

[14] I. Frieslaar. Robust South African sign language gesture recognition using hand motion and shape. Master's thesis, Department of Computer Science, University of the Western Cape, 2014.

[15] W. Nel. An integrated sign language recognition system. Master's thesis, Department of Computer Science, University of the Western Cape, 2014.

[16] D. Mushfieldt. Robust facial expression recognition in the presence of rotation and partial occlusion. Master's thesis, Department of Computer Science, University of the Western Cape, 2014.

[17] J. Whitehill. Automatic real-time facial expression recognition for signed language translation. Master's thesis, Department of Computer Science, University of the Western Cape, 2006.

[18] R. Foster. A comparison of machine learning techniques for hand shape recognition. Master's thesis, Department of Computer Science, University of the Western Cape, 2014.

[19] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[20] R. Kjeldsen and J. Kender. Finding skin in color images. In *Proceedings of the 2Nd International Conference on Automatic Face and Gesture Recognition (FG '96)*, FG '96, pages 312–, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7713-9. URL http://dl.acm.org/citation.cfm?id=524467.795998.

[21] Michael J Taylor and Tim Morris. Adaptive skin segmentation via feature-based face detection. In *SPIE Photonics Europe*, pages 91390P–91390P. International Society for Optics and Photonics, 2014.

[22] T. A. El-Hafeez. A new system for extracting and detecting skin color regions from PDF documents. *International Journal on Computer Science and Engineering*, 2 (9):2838–2846, 2010.

[23] M.J. Swain and D.A. Ballard. Indexing via color histograms. *IEEE*, pages 390–393, 1990.

[24] J. Bersen. Dynamic thresholding of gray-level images. *International Conference For Pattern Recognition*, pages 1251–1255, 1986.

[25] C.K. Chow and T. Kaneko. Automatic boundary detection of the left ventricle from cineangiograms. *Comp. Biomed. Res*, pages 388–410, 1972.

[26] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *ICPR*, 2004.

[27] J. P. Lewis. Fast normalized cross-correlation. *Vision Interface*, pages 120–123, 1995.

[28] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface. In *Fourth IEEE Workshop on Applications of Computer Vision*, pages 214–219, 1998.

[29] D. Kriesel. A brief introduction to neural networks, 2007. URL http://www.dkriesel.com.

[30] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *International Symposium on Circuits and Systems (ISCAS'10)*, Paris, 2010.

[31] Y. Tabata, T. Kuroda, and K. Okamoto. Development of a glove-type input device with the minimum number of sensors for Japanese finger spelling. In *Proceedings of International Conference on Disability, Virtual Reality and Associated Technologies*, pages 305–310, 2012.

[32] L. K. Phadtare, R. S. Kushalnagar, and N. D. Cahill. Detecting hand-palm orientation and hand shapes for sign language gesture recognition using 3D images. In *Image Processing Workshop (WNYIPW)*, pages 29–32, Western New York, 2012.

[33] Structure IO. Structure camera module, 2015. URL http://structure.io/.

[34] Google. Project Tango, 2015. URL https://www.google.com/atap/project-tango/.

[35] A. Aichert. Feature extraction techniques. In *CAMP Medical Seminar WS0708*, Jan 2008.

[36] P. Vamplew. Recognition of sign language gestures using neural networks. *Proc. 1st Euro. Conf. Disability, Virtual Reality and Assoc. Tech*, pages 27–33, 1996.

[37] D. J. Sturman and D. Zeltzer. A survey of glove based input. *Computer Graphics and Applications*, pages 30–39, 2002.

[38] J. J. LaViola. A survey of hand posture and gesture recognition techniques and technology. 1999.

[39] O. Sutton. Introduction to k nearest neighbour classification and condensed nearest neighbour data reduction. Technical report, University of Leicester, Feb 2012.

[40] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, 1995.

[41] W. Iba and P. Langley. Induction of one-level decision trees. In *ML '92: Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240, Aberdeen, Scotland, July 1992.

[42] L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *ICML '06: Proceedings of the 23rd International Conference on Machine Learning*, pages 753–760, 2006.

[43] A. Sengar P. Nagar and M. Sharma. Hand shape based gesture recognition in hardware. *Arch. Applied Sciences Research*, pages 261–269, 2013.

[44] A. Kumar and D. Zhang. Personal recognition using hand shape and texture. *Transactions on Image Processing*, pages 2454–2461, 2006.

[45] Y. Shirai Y. Hamada, N. Shimada. Hand shape estimation under complex backgrounds for sign language recognition. *International Conference on Automatic Face and Gesture Recognition*, 2004.

[46] T. G. Zimmerman, J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill. A hand gesture interface device. *ACM*, pages 189–192, 1987.

[47] T. Takahashi and F. Kishino. Hand gesture coding based on experiments using a hand gesture interface device. *SIGCHI Bulletin*, pages 67–74, 1991.

[48] P. Mistry and P. Maes. Sixthsense: A wearable gesture interface. *SIBGRAPH*, 2009.

[49] R. Liang and M. Ouhyoung. A real-time continuous gesture recognition system for sign language. 2009.

[50] T. Starner and A. Pentland. Real-time american sign language recognition using desk and wearable computer based video. *Transactions on Pattern Analysis and Machine Intelligence*, pages 1371–1375, 1998.

[51] F. Parvini, D. McLeod, C. Shahabi, B. Navai, B. Zali, and S. Ghandeharizadeh. An approach to glove-based gesture recognition. In *Human-Computer Interaction. Novel Interaction Methods and Techniques Lecture Notes in Computer Science*, volume 5611, pages 236–245. Springer, 2009.

[52] A. S. Ghotkar and Dr. G. K. Kharate. Study of vision based hand gesture recognition using indian sign language. *International Journal on Smart Sensing and Intelligent Systems*, pages 96–115, 2014.

[53] Shalini and Dr. R. Patil. Real time control system based on hand gesture detection and recognition. *International Journal of Science and Research*, pages 743–749, 2015.

[54] Y. Yan D. Guo and M. Xie. Vision-based hand gesture recognition for human-vehicle interaction.

[55] X. Liu and K. Fujimura. Hand gesture recognition using depth data. *International Conference on Automatic Face and Gesture Recognition*, 2004.

[56] Z. Ren, J. Meng, J Yuan, and Z. Zhang. Robust hand gesture recognition with kinect sensor. *ACM Multi Media*, pages 759–760, 2011.

[57] A. Ramamoorthya, N. Vaswania, S. Chaudhurya, and S. Banerjee. Recognition of dynamic hand gestures. *Journal of Pattern Recognition*, pages 2069–2080, 2012.

[58] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. A. D. Caro, F. Ducatelle, E. Ferrante, A. Förster, J. M. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. M. de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. Technical Report TR 14-11, IRIDIA, Brussels, Belgium, 2011.

[59] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proc. of 22nd Intl. Joint Conf. on Artificial Intelligence*, pages 1237–1242, 2011.

[60] D. H. Wiesel and T. N. Hubel. receptive fields of single neurones in the cats striate cortex. *Journal of Physio*, 148:573–591, 1959.

[61] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1989.

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[63] Feng Jiang, Jie Ren, Changhoon Lee, Wuzhen Shi, Shaohui Liu, and Debin Zhao. Spatial and temporal pyramid-based real-time gesture recognition. *Journal of Real-Time Image Processing*, pages 1–13, 2016.

[64] H. Takeda, P. Veerkamp, T. Tomiyama, and H. Yoshikawam. Modeling design processes. *AI Magazine*, 1990(Winter):37–48, 1990.

[65] T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, 1996.

[66] I. Lakatos. *The Methodology of Scientific Research Programmes*. Eds. J. Worral and G. Currie, Cambridge University Press, 1978.

[67] S. Gregor and A. Hevner. Positioning and presenting design science research for maximum impact. *MIS Quarterly*, 30(2):337–355, 2013.

[68] Oxford Dictionaries. Design: definition, 2016. URL http://www.oxforddictionaries.com/us/definition/american_english/design.

[69] Herbert Simon. *The Sciences of the Artificial, Third Edition*. MIT Press, 1996.

[70] D. Brown. Faster upper body pose recognition and estimation using Compute Unified Device Architecture. Master's thesis, University of the Western Cape, Computer Science, 2013.

[71] R. Collobert, C. Farabet, K. Kavukcuoglu, and S. Chintala. Torch7 a computing framework, 2010. URL http://torch.ch.

[72] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten Zip code recognition. *Neural Computation*, 1:541–551, 1989.

[73] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[74] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient Backprop*. Springer, 1998.

[75] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, 2013.

[76] X. Han and Y. Li. The application of convolution neural networks in handwritten numeral recognition. *International Journal of Database Theory and Application*, 8 (3):367–376, 2015.

[77] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26 (5):1651–1686, 1998.

[78] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[79] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *International Conference on Computer Vision*, New York, 1998.

[80] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, November 1998.

[81] Yali Amit, Donald Geman, and Kenneth Wilder. Joint induction of shape features and tree classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1300–1305, 1997.

[82] John K. Tsotsos, S. M. Culhane, W. Y. K. Wai, Y. H. Lai, N. Davis, and F. Nuflo. Modeling visual-attention via selective tuning. *Artificial Intelligence Journal*, 78 (1):507–545, 1995.

[83] C. R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[84] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. *In Proceedings Thirteenth Conf. on Uncertainty in Artificial Intelligence*, 1997.

[85] N. Perveen, D. Kumar, and I. Bhardwaj. An overview on template matching methodologies and its applications. *International Journal of Research in Computer And Communication Technology*, 2(10):988–995, 2013.

[86] G. R. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[87] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

[88] J. Wu. Introduction to convolutional neural networks. Technical report, National Key Lab for Novel Software Technology, Nanjing University, China, 2016.

[89] X. Pan and V. Srikumar. Expressiveness of rectifier networks. In *International Conference on Machine Learning*, New York, 2016.

[90] Stanford University. Stanford deep learning, 2015. URL http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial.