

https://dspace.org/

Deformable part model with CNN features for facial landmark detection under occlusion

Brink, Hanno

2018

https://hdl.handle.net/10539/26699

Brink, Hanno (2018) Deformable part model with CNN features for facial landmark detection under occlusion, University of the Witwatersrand, Johannesburg, <http://hdl.handle.net/10539/226699>

Downloaded from a DSpace repository. https://dspace.org/



University of the Witwatersrand

Masters Thesis

Deformable part model with CNN features for facial landmark detection under occlusion

Author:
Hanno Brink - 1371627

Supervisor:
Dr. Hima Vadapalli

October 18, 2018

Abstract

Detecting and localizing facial regions in images is a fundamental building block of many applications in the field of affective computing and human-computer interaction. This allows systems to do a variety of higher level analysis such as facial expression recognition. Facial expression recognition is based on the effective extraction of relevant facial features. Many techniques have been proposed to deal with the robust extraction of these features under a wide variety of poses and occlusion conditions. These techniques include Deformable Part Models (DPMs), and more recently deep Convolutional Neural Networks (CNNs). Recently, hybrid models based on DPMs and CNNs have been proposed considering the generalization properties of CNNs and DPMs. In this work we propose a combined system, using CNNs as features for a DPM with a focus on dealing with occlusion. We also propose a method of face localization allowing occluded regions to be detected and explicitly ignored during the detection step.

Declaration of Authorship

I, Hanno Brink, declare that this dissertation titled, "Deformable part model with CNN features for facial landmark detection under occlusion" and the work presented in it are my own. I confirm that:

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

Signed:

_____

Date:

_____

Contents

1 Introduction **2**
- 1.1 Research aims . . . 4
- 1.2 Research questions . . . 5
- 1.3 Research objectives . . . 5
- 1.4 Summary . . . 5

2 Background **6**
- 2.1 Deformable part models . . . 6
- 2.2 Feature generation techniques . . . 10
  - 2.2.1 Histogram of oriented gradients . . . 10
- 2.3 Classification techniques . . . 11
  - 2.3.1 Support vector machines . . . 11
  - 2.3.2 Neural networks . . . 12
    - 2.3.2.1 The artificial neuron . . . 13
    - 2.3.2.2 Structure of neural networks . . . 15
    - 2.3.2.3 Training neural networks . . . 15
- 2.4 Convolutional neural networks . . . 25
  - 2.4.1 Convolutional layers . . . 26
  - 2.4.2 Rectified linear units . . . 29
  - 2.4.3 Spatial pooling layers . . . 29
  - 2.4.4 Classification layers . . . 31
  - 2.4.5 Training of a CNN . . . 32
    - 2.4.5.1 Subject independence . . . 32
- 2.5 Summary . . . 32

3 Literature Review **33**
- 3.1 Region localization without significant occlusion . . . 33
  - 3.1.1 Region localization using hand-crafted features . . . 33
    - 3.1.1.1 Deformable part models . . . 34
    - 3.1.1.2 Mixtures of trees . . . 35

3.1.2 Region localization using learned features . . . 35
3.1.2.1 Active appearance models . . . 35
3.1.2.2 Region localization using CNNs . . . 36
3.2 Region localization under occlusion . . . 39
3.2.1 Region localization using hand-crafted features . . . 39
3.2.1.1 Robust cascaded pose regression (RCPR) . . . 39
3.2.1.2 Consensus of exemplars . . . 40
3.2.1.3 Occlusion coherence . . . 40
3.2.2 Region localization using learned features . . . 41
3.2.2.1 Active appearance models . . . 41
3.2.2.2 Regressing local binary features . . . 42
3.2.2.3 Convolutional neural networks . . . 42
3.3 Conclusion . . . 44

4 Methodology **46**
4.1 Face localization . . . 47
4.1.1 Face localization using a face CNN . . . 48
4.1.1.1 Non-maximum suppression . . . 48
4.1.1.2 Stride . . . 50
4.1.1.3 Dealing with different input face sizes . . . 50
4.1.1.4 Dealing with rotation . . . 51
4.1.2 Face localization using a face-part CNN . . . 51
4.1.3 Iterative alignment . . . 53
4.2 Face-part localization . . . 53
4.2.1 Geometric spring cost . . . 54
4.2.1.1 Spring cost of non-parts . . . 57
4.2.2 Dealing with occlusion when detecting parts . . . 57
4.2.3 Non-max suppression and final localization . . . 57
4.2.4 Dealing with rotation . . . 58
4.3 Summary . . . 58

5 Experimental set up **60**
5.1 Training dataset . . . 60
5.2 Data Preprocessing . . . 61
5.2.1 Data for face CNN . . . 62
5.2.1.1 Face data augmentation . . . 63
5.2.1.2 Generation of non-face samples . . . 64
5.2.2 Data for face part CNN . . . 64
5.2.2.1 Face-part data augmentation . . . 65
5.2.2.2 Generation of negative face-part samples . . . 65
5.2.3 Training data for geometric features . . . 65

5.3 CNN architecture and training . . . 66
5.3.1 Training process . . . 66
5.3.2 Training, validation, and testing datasets . . . 67
5.3.3 ANN performance metrics . . . 68
5.3.4 Face CNN architecture and training . . . 69
5.3.5 Face-part CNN architecture and training . . . 75
5.4 Geometric feature training . . . 82
5.5 Summary . . . 85

6 Results **86**
6.1 Testing dataset . . . 86
6.2 Preparation of test data . . . 88
6.3 Face and face-part CNN performance . . . 89
6.4 System performance . . . 89
6.4.1 Measurement metrics . . . 90
6.4.2 Face and face-part localization results . . . 91
6.4.3 Face-part localization . . . 91
6.5 Discussion . . . 93
6.5.1 Face and face-part localization . . . 93
6.5.1.1 Face part localization using a face CNN . . . 93
6.5.1.2 Face part localization using a face CNN with iterative alignment . . . 94
6.5.1.3 Face part localization using a part CNN . . . 96
6.5.1.4 Face part localization using a part CNN with iterative alignment . . . 97
6.5.2 Face-part localization . . . 98
6.5.3 Impact of neural network performance on localization . . . 99
6.5.4 Impact of occlusion on face and face-part localization . . . 100
6.5.5 Impact of rotation on face and face-part localization . . . 101
6.6 Comparison to other work . . . 103
6.7 Summary . . . 104

7 Conclusion **105**
7.1 Future work . . . 105

List of Figures

2.1 A representation of the line segments and "springs" from a pictorial structure model of a face [Fischler and Elschlager, 1973]. 7

2.2 On the left side an example detection of a person is depicted. The second image shows the coarse root template that is used for the initial detection, the third image depicts the higher resolution templates for each part, and the final image shows the spatial model that describes the possible placements of each of the part models [Felzenszwalb et al., 2010]. 8

2.3 The input image at different scales are depicted on the left and the right side of the image depicts the calculated HOG features. The placement of the coarse root filter is depicted near the top of the pyramid and the placements of the finer-grained part models are depicted near the bottom of the pyramid [Felzenszwalb et al., 2010]. 9

2.4 Hyper-planes separating 2 classes denoted by dark and light dots on a plane. a) A non-optimal separating hyper-plane with a small margin. b) A separating hyper-plane with a large margin [Moore, 2001]. 11

2.5 A simple demonstration of dealing with non-separable data by introducing higher dimensions a) Two non-separable classes on a one dimensional plane. b) By casting the data-points into a higher dimension the classes now becomes separable using a simple linear plane [Moore, 2001]. 12

2.6 A simplified drawing of the structure of biological neurons [Gershenson, 2003, Demuth et al., 2014]. 13

2.7 A conceptual drawing of the structure of an artificial neuron [Demuth et al., 2014]. 14

2.8 Some of the most common neuron activation functions: a) A hard-limit function. b) A purely linear function. c) A Log-Sigmoid function [Demuth et al., 2014]. 15

2.9 **A single layer of neurons with multiple inputs** [Demuth et al., 2014]. 16

2.10 This figure depicts a simple curve that depicts the error produced by the ANN on the y axis, by the weight value w shown on the x axis. The red dot labelled 1 shows the error produced by the corresponding weight value during the first iteration. The red dot labelled 2 and 3 depict the error produced by the subsequent adjustments to the weight w. The black cross shows the true point where the error is minimized [Thomas, 2017]. 17
2.11 A simple 3 layer ANN with 3 inputs, three layers and one output. 19
2.12 An example of how δ is back-propagated to a node in the previous layer with only a single node in the output layer, using the weight of the connection to the current node. 21
2.13 An example of how δ is back-propagated to a node in the previous layer with many nodes in the output layer, using a weighted sum of the the errors propagated from the next layer. 22
2.14 A convolution of a single filter over an image and the output result. The input layer is shown on the left and the activation of the neurons in the filter is showed on the right hand side. The weights that connect each neuron in the filter layer to the local input values are shown in the weights grid at the top of the image. The produced output shows high values where the input matches the weights well and low values where the input does not match the weights well. This output layer is essentially a map of the positions where the feature described by the weights are present. 27
2.15 One input layer producing a number of filter maps in the following layer . . 28
2.16 a) A convolution with a stride of 1 pixel. b) A convolution with a stride of 2 pixels. 28
2.17 a) ReLU function. b) Leaky ReLU, where a_i is fixed and Parametric ReLU where a_i is learned. c) Randomized ReLU where the a_{ij} is a random number sampled withing a range during training and is kept consistent during testing [Xu et al., 2015]. 30
2.18 An example of max-pooling with a width of 2 and a stride of 2 30
2.19 Example images of learnt features reconstructed from different layers in a CNN [Lee et al., 2009]. On the top-left the simple features detected by the first layer can be seen. The bottom left shows the features that the second layer looks for. These features are made up of combinations of the features detected in the first layer. On the top-right the features from the third layer can be seen. Finally on the bottom-right the features from the fourth layer can be seen. The aim is to demonstrate how a CNN is able to combine low-level features in the first layers into more complex shapes in later layers. 31

3.1 The first row depicts different configurations of the model proposed by Ghiasi and Fowlkes [2014] that consists of a tree of parts. The root node of each part is denoted by a black circle and has no HOG template associated with it, but is connected to a set of leaf nodes denoted by red or green circles. The green leaf nodes denote the visible feature points and the red leaf nodes show feature points where the feature point is occluded. The second row depicts the HOG appearance model of each of the visible key points. 41

4.1 Architecture of the proposed face, and face part localization system. 47
4.2 a) Face localization using a face CNN model representing the face in its entirety that is convolved over the entire image. b) A face localization algorithm that calculates the response at a point as the result of the responses of the parts detected at certain locations. 48
4.3 A map of face centre proposals from the face CNN where white indicates the highest certainty of a face and black the lowest on an image from the HELEN dataset [Le et al., 2012]. The map shows the face centre as a potential region for a face. 49
4.4 a) Face region proposals with white as the highest response and black as the lowest. b) The non-max suppression algorithm applied on the face region proposals. Outlier proposals are shown as red boxes, blue boxes correspond to proposals taken into account, and yellow is the final detection based on the average of all the proposals that aren't outliers. c) The final cropped region. 50
4.5 A few face region proposals generated by varying stride lengths with the corresponding non-max suppression results on an image from the HELEN dataset [Le et al., 2012]. It is important to note that the proposals are not generated by a fully trained face CNN. a) 16 pixel stride. b) 8 pixel stride. c) 4 pixel stride. 51
4.6 How the part-based face detector identifies and ignores occlusion 52
4.7 A visualization of the iterative alignment process. In this image the responses of the part CNN's eye neuron is shown over the face region where blue denotes the lowest response and red the highest. The responses are also plotted from the top left corner of the image, as these points correspond to the top left corner of the detection window for the face-part CNN. 54
4.8 a) A map showing the expected nose center-point relative to the cropped face based on the geometric information where blue denotes the lowest response and red the highest. b) A map of responses of the nose part points based on the responses from the Face-Part CNN . c) The sum of (a) and (b), this also shows the most probable placements of the part. The original image is from the HELEN dataset [Le et al., 2012] 55
4.9 Face geometry spring-costs function 56

4.10 Face geometry model where white denotes the most likely placements for each part and black the least likely position for each part. The spring-cost map used in this image is a combined representation of the spring-costs of each part. 56
4.11 a) A map of responses based on the placement of the nose relative to the cropped face region where the nose part is occluded. Red parts are the highest responding, and blue are the lowest responding parts. This image demonstrates that the highest likelihood of placement is within the occluded part of the face based on the spring cost. b) A map of responses based completely on the nose-part responses of the face-part CNN. These responses are largely ignored because of the occlusion present. c) The sum of (a) and (b), this also shows how the response of the nose-part is adjusted by most probable placements of the part, which generates good responses in the occluded part of the image which is correct. 58

5.1 Example images from the HELEN Dataset [Le et al., 2012]. 61
5.2 Example image from the HELEN Dataset depicting the positions of the 149 feature points [Le et al., 2012]. 61
5.3 A flowchart depicting how the data is pre-processed and split into training, testing and validation data. 62
5.4 a) Original image from the COFW dataset [Burgos-Artizzu et al., 2013]. b) Normalized image with the face rotated correctly and an 8-pixel region around the edges of the face feature points. 63
5.5 Some examples of non-face images generated from the HELEN dataset [Le et al., 2012]. 64
5.6 Some examples of negative face part images generated from the HELEN dataset [Le et al., 2012]. 65
5.7 A flowchart depicting the process of training a CNN. 67
5.8 A flowchart depicting the process of testing the resulting CNN on validation data after training. 68
5.9 Proposed architecture for a face CNN based on a variation on the LeNet5 architecture as described by the Theano Development Team [2018]. 70
5.10 Accuracy of classification of samples from training and validation datasets as training progressed for the architecture proposed by the Theano Development Team [2018]. 71
5.11 Proposed architecture for a face CNN based on the architecture described by Garcia and Delakis [2004]. 72
5.12 Accuracy of classification of samples from training and validation datasets as training progressed for the architecture proposed by Garcia and Delakis [2004]. 72
5.13 Proposed custom architecture for a face CNN. 73

5.14 Accuracy of classification of samples from training and validation datasets as training progressed for the custom architecture. 74
5.15 A modified version of the architecture by Garcia and Delakis [2004] for a face CNN 74
5.16 Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on Garcia and Delakis [2004]. 75
5.17 Face CNN architecture training loss comparison 76
5.18 Proposed architecture for a face-part CNN based on the modified LeNet architecture described by the Theano Development Team [2018] 77
5.19 Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on the modified LeNet architecture by the Theano Development Team [2018]. 78
5.20 Proposed architecture for face-part CNN based on the architecture described by Garcia and Delakis [2004] 79
5.21 Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on Garcia and Delakis [2004]. 79
5.22 Proposed architecture for face-part CNN based on a modified version of the LeNet architecture proposed by the Theano Development Team [2018]. . . . 80
5.23 Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on the Theano Development Team [2018]. 81
5.24 Proposed architecture for Part detector neural network 81
5.25 Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on Garcia and Delakis [2004]. 82
5.26 Face-part CNN architecture training loss comparison 83
5.27 Confidence Ellipses calculated from observed points. 84

6.1 Example images from the COFW Dataset [Burgos-Artizzu et al., 2013]. . . 87
6.2 Example feature point locations on images from the COFW Dataset [Burgos-Artizzu et al., 2013]. Red points are occluded and green points are visible . 87
6.3 Samples from the normalized COFW dataset that were used to test the performance of the proposed system. 88
6.4 Samples from the normalized COFW dataset that were used to test the performance of the proposed system. 89
6.5 Cumulative error graph for face part localization performance on a sample of 100 images from the COFW dataset. 91
6.6 Cumulative error graph for part localization performance on a sample of 100 normalized face images from the COFW dataset. 92

6.7 a) An example of the most accurate localization using a face CNN with an average error of 34,79 %. b) The face CNN response over the image that was used to calculate the face region. 94
6.8 a) An example of one of the least accurate localizations using a face CNN with an average error of 284,91%. b) The face CNN response over the image that was used to calculate the face region. 94
6.9 a) An example an accurate localization using a face CNN combined with an iterative alignment step, resulting in an average error of 32,21% of the inter-ocular distance. The red square shows the initial localization for the face region, and the blue square shows the localization after the alignment step has been performed. b) The face CNN response for faces over the image that was used to calculate the face region. 95
6.10 a) An example of one of the least accurate localizations using a face CNN combined with an iterative alignment step, resulting in an average error of 384,19% of the inter-ocular distance. b) The CNN response over the image that was used to calculate the face region. 95
6.11 a) An example an accurate localization using a part CNN with an average error of 34,34%. b) A map of responses calculated by the face-part CNN algorithm. 96
6.12 a) An example of one of the least accurate localizations using a face-part CNN with an average error of 580,85%. b) A map of responses calculated by the part-based detector. 97
6.13 a) An example an accurate localization using a face-part CNN with iterative alignment with an average error of 49,86% of the inter-ocular distance. b) A map of responses calculated by the face-part CNN. 98
6.14 a) An example an inaccurate localization using a face-part CNN with iterative alignment with an average error of 599,83% of the inter-ocular distance. b) A map of responses calculated by the part-based detector. 98
6.15 a) An example an accurate localization using a face-part detection CNN with an average error of 5,75%. b) The responses of the face-part CNN combined with the spring-cost that was used to calculate the part locations. 99
6.16 a) An example of one of the least accurate localizations using a face-part detection CNN with an average error of 48,11 %. b) The responses of the face-part classifier neural network combined with the spring-cost that was used to calculate the part locations. 99
6.17 Face region proposals for face CNNs with a loss of: a) 0,29992 b) 0,25329 c) 0,19691 d) 0,15731 e) 0,09852 f) 0,00591 100

List of Tables

5.1 Confusion matrix for the face CNN based on the work from the Theano Development Team [2018] on validation data. 70
5.2 Confusion matrix for the face CNN based on the work from Garcia and Delakis [2004] as tested on validation data. 71
5.3 Confusion matrix for a proposed face CNN tested on validation data 73
5.4 Confusion matrix for the modified face CNN by Garcia and Delakis [2004] as tested on validation data 75
5.5 A table where accuracy of the neural networks are compared 75
5.6 Confusion matrix for a part detector based on the work from the Theano Development Team [2018] tested on 1000 validation images 77
5.7 Confusion matrix for the face-part CNN based on the work from Garcia and Delakis [2004] on 1000 validation images 80
5.8 Confusion matrix for a proposed face-part CNN 80
5.9 Confusion matrix for final face-part CNN as tested on 1000 validation samples. 82
5.10 A table where accuracy of the face-part detector neural networks are compared 82

6.1 Confusion matrix for the final trained face CNN architecture on unseen testing data. 89
6.2 Confusion matrix for the final part CNN as tested on 1000 unseen testing images. 90
6.3 Statistics around the results. Error is measured as a percentage of the distance between the eyes. 92
6.4 Face part detection results 93
6.5 A comparison between the error for images with light occlusion and images with heavy occlusion. Error is measured as a percentage of the distance between the eyes. 101
6.6 A comparison between the error for images with low out-of-plane rotation and images with heavy out-of-plane rotation. Error is measured as a percentage of the distance between the eyes. 102
6.7 Statistics around the results 104

List of abbreviations

Abbreviation	Definition
AAM	Active Appearance Model
ACDCNN	Adaptive Cascade Deep Convolutional Neural Networks
ANN	Artificial Neural Network
AU	Action Unit
CLM	Constrained Local Model
CNN	Convolution Neural Network
COFW	Caltech Occluded Faces in the Wild
CPR	Cascaded Pose Regression
DPM	Deformable Part Model
FACS	Facial Action Coding System
FER	Facial Expression Recognition
HDPM	Hierarchical Deformable Part Model
HDT	Hierarchical Deformable Templates
HOG	Histogram of Gradients
IoU	Intersection over Union
LBP	Local Binary Patterns
LM	Local Minima
LSVM	Linear Support Vector Machine
PCA	Principal Component Analysis
RCPR	Robust Cascaded Pose Regression
SIFT	Scale-Invariant Feature Transform
SVM	Support Vector Machine

Chapter 1

Introduction

"The face is a picture of the mind with the eyes as its interpreter."
– Marcus Tullius Cicero

Humans are creatures that can experience a wide range of complicated and powerful emotions. These emotions play a big part in the way we see and interact with the world. Emotion can be defined as a force, guiding perception and attention. The emotional state of a subject does not only influence the behaviour of the individual, but recognizing emotional state in others can also instil strong emotions in the observer and influence how he or she reacts to the subject [Izard, 1993]. This can be seen in our need to sympathize with individuals expressing sadness, assist those displaying fear and rejoice with those who are happy [Izard, 1993]. This type of interaction can be extended to, not only other humans, but also the tools we use, such as a computer.

Studies conducted on the interaction between humans and computers have concluded that human-computer interaction is fundamentally social [Nass et al., 1994]. From this came the relatively young field of computer science called Affective Computing. Affective computing is defined as computing that relates to, arises from, or influences emotions [Picard, 1995]. The idea was first proposed by Picard [1995] in an essay entitled *Affective Computing*. In that article, the author makes a case for "Feeling" computer systems. Knowing the emotional state of an individual can greatly influence and improve interaction and communication between individuals. The emotional state can be used to interpret speech in order to better extract the intended meaning of the message. In order to improve interaction between humans and computers, computers should be able to detect the emotional state of the user and attempt to adjust its behaviour in order to better satisfy the needs of the user. An affective computing system can have many applications in the real world.

One area that can benefit from the use of affective computing systems is the field of electronic learning. These systems could better perform teaching tasks if they are able to

detect and manage the emotional state of the student [Whitehill et al., 2008]. These systems should be able to keep the student engaged if they seem to get bored or explain concepts in greater detail to students who are frustrated or confused. However, estimating the emotional state of a person is a difficult task.

Emotional state cannot be directly observed by another person, but the expression of that emotion through the face, voice or other media can be observed and interpreted [Picard, 1995]. The voluntary or involuntary expression of the emotional state is often referred to as "sentic modulation" [Picard, 1995]. Humans are not only capable of expressing this inner state through methods like facial expressions and verbal communication, but also capable of observing the emotional state of other humans based on these cues. This is usually done by humans with minimal effort and with fair accuracy. However, it is quite difficult for computers to recognise and categorise these complex subjective cognitive states of mind.

Some of the early work done on facial expression was by Charles Darwin in 1872, and published in his book, *The Expression of Emotions in Man and Animals* [Darwin and Prodger, 1872]. This work opened up the study of facial expressions, but it was not until almost a hundred years later that a series of in-depth studies were done by Ekman and Izard [Ekman and Dachter, 2014]. Before the studies conducted by Ekman and Friesen [1978], the face was considered an inaccurate source of mostly culture specific, stereotypical information. Ekman and Friesen [1978] in their paper "Facial action coding system: A technique for the measurement of facial movement", proved the universality of facial expressions and also defined a quantitative method for describing and interpreting these expressions. Ekman and Friesen [1978] developed what is called the Facial Action Coding System (FACS). They have described a series of facial action units (AU's) that correspond to certain movements in the face. Combinations of these facial action units can be translated to the expression of an emotional state. The development of computer systems, capable of automatically detecting the emotional state of a subject, is still a challenging task.

Detecting emotion from images or video sequences of the face is an attractive option because it is usually a non-intrusive method to extract the necessary information. This principle of quantifying facial expression is the foundation of the computer systems now being developed to interpret these emotional states. In order to deduce the emotional state experienced by a subject from facial data, the displayed expression needs to be described using parameters that can be used for classification. Automatic extraction of this information from facial images or video is a computer vision problem. Systems that are capable of detecting and classifying facial expression are referred to as a Facial Expression Recognition (FER) system [Tian et al., 2011]. FER systems are subject to much of the same problems faced by any computer-vision system. They should be able to perform accurately and robustly under a large variety of conditions and without human intervention.

The main factors responsible for the decrease in accuracy of these systems, are illumination conditions, face pose variations, and the presence of occlusion. There are many ways by which facial occlusion can occur in the wild. Common occlusion such as the wearing of scarves, glasses, facial hair, veils, and hats can cause parts of the face to be obscured. Out-of-plane rotation of the face to certain degrees can also cause parts of the face to be occluded [Min et al., 2014]. Occlusions make it very difficult to extract reliable facial feature information from images, as some of the important reference points in the image may not be present. This may lead to inaccurate interpretations of the user's current emotional state. However, users cannot be expected to always interact with the system in a manner that is consistent with laboratory conditions and will often use these systems with little or no regard for the FER component. Creating a FER system that is capable of handling these occlusions will greatly improve the system accuracy in "real-world" situations and ultimately improve the practicality of such a system.

Face-part localization is one of the basic building blocks of higher level detection or classification tasks. This is the process of looking at a raw image and localizing the face or faces in that image, as well as localizing all the parts within the localized face-region, such as eyes, nose mouth, forehead etc. Once the face-parts has been aligned, the resulting information about the face geometry can be used as a feature for FER systems, or the geometric information can aid in selecting regions of interest for FER systems. Many methods have been proposed to handle the problem of occlusion during face-part localization, such as the use of learned shape information as well as inferring the positions of occluded parts from the positions of visible parts. This works well when detection is done on a "part" or "sub-region" level and relationships between these parts are inferred, as seen in the work done by Burgos-Artizzu et al. [2013] and Ghiasi and Fowlkes [2014]. The approach of modelling an object as the sum of its parts and the relationships between those parts, is referred to as a Deformable Part Model (DPM) approach. The constraints between the parts of a DPM are enforced using a geometric face model that is used to calculate a "spring cost" that pulls or pushes parts towards plausible positions. The other way that these problems are addressed, is with the use of richer descriptors that allow for greater shape variation and are robust to some occlusion [Nguyen et al., 2015]. One of the methods that is often used as a robust descriptor are Convolutional Neural Networks (CNNs). With the introduction and popularization of modern machine learning techniques, such as CNNs, there seems to be new potential in this field. In this work, a DPM using CNN's is proposed with the aim of detecting face-parts under a wide variety of occlusion and rotation conditions.

1.1 Research aims

The aim of this research is to implement a face detection and part-localization system using a DPM architecture with CNN appearance descriptors that show robustness to typical

occlusion conditions that are encountered in the wild. The proposed method also uses a simple elliptical part model to calculate spring-cost.

1.2 Research questions

1. Can a single CNN be used to effectively distinguish between the four face-parts such as eyes, nose and mouth parts?
2. Can a simple CNN architecture successfully be used to localize occluded faces?
3. Can a simple DPM-system with rich descriptors effectively localize unoccluded face-parts and can these descriptors be used to successfully detect and ignore occlusion?
4. Can a simple statistical part-placement model effectively model part positions in a DPM model and be used to infer the locations of occluded parts?

1.3 Research objectives

1. Build and test the efficacy of a single CNN that is able to detect the four basic face-parts.
2. Build a CNN that is able to robustly locate occluded face regions on images.
3. Build a DPM consisting of CNNs as descriptors that is able to robustly locate and ignore occlusion explicitly when detecting faces.
4. Build and test the efficacy of a simple model that uses learned knowledge of the positions of face-parts to search for plausible regions for face-parts and infer positions of occluded parts.

1.4 Summary

In this section, a brief history about FER systems was given as well as the importance and applications of robust face-part localization. The challenge of face-part localization under occlusion is discussed and methods of handling it was briefly discussed. Finally, the research aims, as well as the main research questions and objectives were outlined.

Chapter 2

Background

In this chapter a brief background of the methods and tools commonly used in facial landmark detection tasks is described. Firstly, the history and workings of deformable part models is described in Section 2.1. Then feature generation methods and classifiers commonly used for DPMs is briefly discussed in Section 2.2. This is followed by a discussion of how these features are classified in Section 2.3. Finally, CNNs and how they function as both a feature generation and extraction method and a classifier is described in Section 2.4.

2.1 Deformable part models

Many methods have been proposed to deal with the computer vision problem of face-part localization. Some of these methods include Active Appearance Models (AAMs), 3D model based alignment, and Deformable Templates [Ivan, 2007, Roth et al., 2016, Yuille et al., 1989]. These methods work by iteratively searching over a huge search space to align the templates, and are very computationally expensive. These methods also have the problem of having to deal with local minima in the search space where a searching algorithm does not continue searching after it achieves a point where the evaluation metric appears to be optimized, however it is not aligned with the global optimal placements. To deal with some of these problems an, old idea, called pictorial structures, initially introduced by Fischler and Elschlager [1973], was revisited and updated with modern features and machine learning. The problem that Fischler and Elschlager [1973] were trying to solve is simply the detection of some object in a scene, given a description of that object. One of the big contributions from this work was the idea of introducing "springs" to a set of local models. These springs are rigidly attached to their part section of the object being detected as can be seen in Figure 2.1. The springs enforce constraints between the parts, by increasing the "placement cost" as the placement distance from the tip of the vector increases. These springs make the model "stretchy" and more general meaning that the same set of local

parts can be reconfigured. The "stretchiness" of the model is measured by how much the placement cost changes based on the distance from the tip of the vector, and serves to constrain the possible placements of the parts relative to each other. The idea is to place the model on the image, matching local parts to the image, while also minimizing the total "cost" of placing the model parts at certain points.

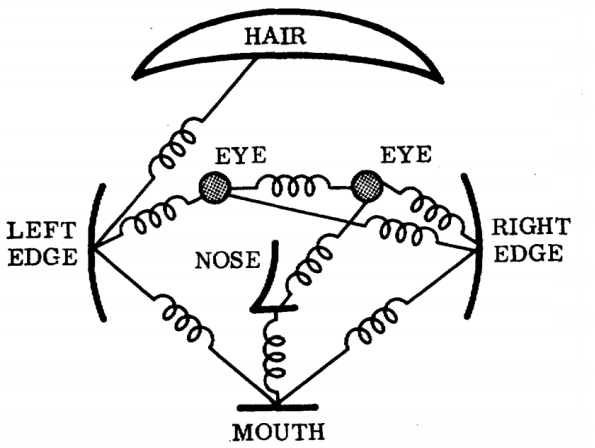


Figure 2.1: A representation of the line segments and "springs" from a pictorial structure model of a face [Fischler and Elschlager, 1973].

Some years later Felzenszwalb and Huttenlocher [2000] noted that this technique was not widely used for matching and recognition tasks because of the computational complexity of matching these models to images. Felzenszwalb and Huttenlocher [2000] improved on the models proposed by Fischler and Elschlager [1973] by enforcing a restriction that the parts should be structured in a tree model to allow for greater computational simplicity. They have also found that using a star topology is quite a natural fit for the modelling of most objects. They also introduce a new set of "springs" allowing them to more accurately model different types of joints, for example, joints that allow for free movement such as that connecting human arms to the torso, rigid springs that allow for a car's wheels to move along the bottom of the chassis, and spring joints like those seen in the work by Fischler and Elschlager [1973] that "pull" one part with relation to the other. The model used a very simple colour based method to match parts, and was more focused on the role of the part-model to enforce realistic constraints, and the results were quite remarkable.

Later, the Dalal-Triggs detector was based on a very effective set of features they introduced, based on a grid of histograms of oriented gradients, also known as "HOG" features [Dalal and Triggs, 2005]. HOG features are described in more detail in Section 2.2.1. These features were calculated for both positive and negative samples of the object they were try-

ing to detect and the resulting features were used as input features to an Support Vector Machine (SVM) classifier that was trained to distinguish between the presence or absence of the object. This detector consisted of a single template for the entire object and achieved results that were an improvement over similar methods at the time. This improved feature generation technique led to an improvement in other part localization techniques. One of these methods were DPMs. DPMs were popularized by Felzenszwalb et al. [2010] in their paper titled "Object Detection with Discriminatively Trained Part Based Models". With their proposed system they were able to achieve state of the art detection of generic objects with highly variable pose and inter-class shape variations. Felzenszwalb et al. [2010] improved on the Dalal-Triggs detector with the introduction of a multi-scale deformable part model. The idea was that an object can be described not only as the appearance of the object as a whole, but also as the combination of its parts. The model for each object consists of a coarse template depicting the object in its entirety, a set of higher resolution part templates, and finally a spatial model for the location of each part. The child parts are connected to the root part in a star-topology. This model is depicted in Figure 2.2.

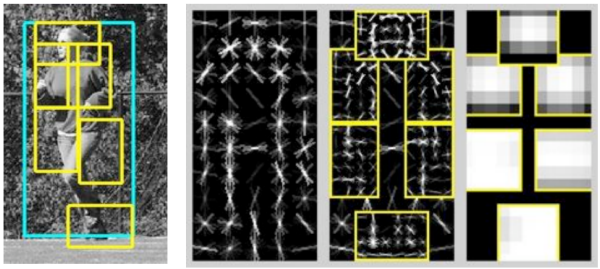


Figure 2.2: On the left side an example detection of a person is depicted. The second image shows the coarse root template that is used for the initial detection, the third image depicts the higher resolution templates for each part, and the final image shows the spatial model that describes the possible placements of each of the part models [Felzenszwalb et al., 2010].

The root model works using the same as the method described by Dalal and Triggs [2005], meaning that the HOG features for the input image are calculated and the root template is iterated over the HOG representation of the image and matched using an SVM classifier. Once the root model is matched to a set of HOG features in the image, a higher resolution HOG feature map is generated in order to match the parts using the location of the root filter as a reference for the positions of the part models. The part models are allowed to be moved within a region describing plausible positions of the part relative to the root node. The spatial model also describes the cost of placing the part models at certain positions. Responses of the child parts are calculated as the similarity in appearance, determined by an SVM classifier using principal component analysis with HOG features, and the distance from the ideal placement of each part as described by the spatial model that uses the root

node as a reference position. This system selects positive detections by scoring the root position based on the response of the root filter plus the best responses of the child parts minus the deformation costs. The highest scoring positions are then selected as positive detections. This method handles scale and shape variation by iterating through different image scales, and pose variations for each part. This way of dealing with scale variation is called a pyramid of features and is performed by calculating HOG features of the input image at different scales and attempting to match both the root node and the part models at each scale by convolving the root model over the entire HOG representation of the image. This method is depicted in Figure 2.3

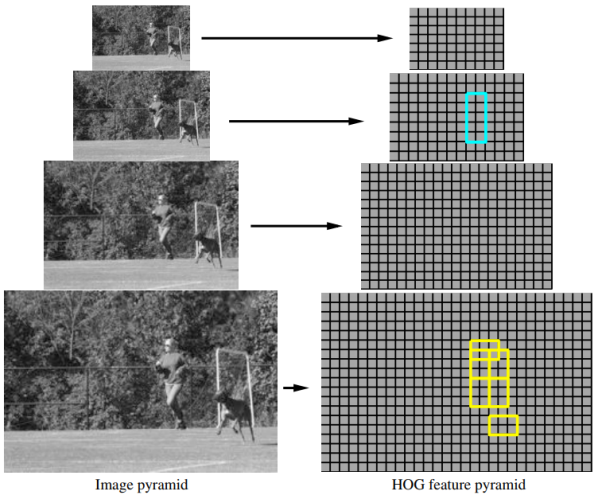


Figure 2.3: The input image at different scales are depicted on the left and the right side of the image depicts the calculated HOG features. The placement of the coarse root filter is depicted near the top of the pyramid and the placements of the finer-grained part models are depicted near the bottom of the pyramid [Felzenszwalb et al., 2010].

Zhu et al. [2010] expanded on the work done DPMs by recognizing that one of the limitations of DPMs at the time was that they where quite limited in the ways they could represent deformable objects. The answer to this problem was what they called Hierarchical Deformable Templates (HDT) where an object model is represented as a tree, where each layer of the tree encodes more granular representations of the object. The leaf nodes in this structure represents the object boundary. This structure allows the model to represent the object with varying degrees of richness. This means that this algorithm is not merely concerned with object detection, but also allows for the localization of key-points that describe the shape of the object. This idea would lay the groundwork for later extensions to the DPM algorithm.

Hierarchical Deformable Part Models (HDPMs) were introduced by Ghiasi and Fowlkes [2014], building on the successes of DPMs and the ideas from HDTs. These models expanded on the star topology by implementing a hierarchical structure consisting of multiple layers, each with a higher level of granularity. This course-to-fine detection significantly reduces the search space and speeds up detection quite drastically. This structure introduces intermediary nodes that do not have a root template associated with it, that allows for the encoding of shape and occlusion information about that part. The shape and occlusion information is typically learned from training data. This deeper representation allows this model to determine occluded key-points and estimate probable localizations based on the information encoded in the higher level representations. This differs from DPMs in that DPMs only allow for high level detection of parts and have no built in mechanisms that can be used to localize visible key-points or infer positions of occluded key-points.

The latest research on DPMs has explored the use of CNNs to replace the feature descriptors used by previous methods [Savalle et al., 2014]. The use of CNNs allows for a rich description of the part appearance model. This approach shows very promising results but is still relatively unexplored.

2.2 Feature generation techniques

One of the important aspects of DPMs is the fact that their performance is very reliant on the feature generation method used and much research has been done in this field. Feature generation is the process of taking raw unstructured data and generating a set of features that are more concise yet still effectively describes the original data. In a computer vision context, these generated features are often sets of metrics that are used in classification algorithms. Many techniques have been proposed as an effective feature generation technique, including HOG features [Dalal and Triggs, 2005], Gabor filters [Huang et al., 2005], Scale-Invariant Feature Transform (SIFT) descriptors [Chen et al., 2014], Local Minima (LM) [Islam and Auwatanamongkol, 2013], Local Binary Batterns (LBP) [Ahonen et al., 2006], and Haar wavelets [Heisele et al., 2000]. Another technique that is often employed is feature extraction, where a subset of the generated features are selected that reduces the number of variables that a machine learning algorithm has to consider. More recently CNNs have been proposed as an effective feature generation and extraction technique for DPMs [Nguyen et al., 2015, Savalle et al., 2014, Yang et al., 2015, Triantafyllidou et al., 2017].

2.2.1 Histogram of oriented gradients

The HOG features employed by the Dalal-Triggs detector is a very simple yet very effective method of describing local appearance patches in an image. This was proposed as an alternative to existing edge detectors at the time. The HOG method uses a densely tiled

grid of cells in the part descriptor. Each HOG cell uses the gradients calculated from every pixel and their neighbours and adds all the calculated gradients to the corresponding "bin". A "bin" is used to track the number of pixels in a cell that has a gradient that falls within a given range of angles. This is done for every pixel in the cell. This process produces a histogram for each cell describing the gradients for that patch of pixels. These cells are then grouped into blocks and a strong contrast-normalization process is applied to each block to make the descriptors more robust to illumination variation. All the normalized cells are then grouped into a local appearance part [Dalal et al., 2006].

2.3 Classification techniques

After these feature generation methods have been applied to the image to reduce the input space, the resulting values are fed through a simple learning algorithm. This is traditionally done by using SVMs or simple ANNs algorithms. In this section these techniques are briefly described and the motivation for the use of CNNs is discussed. Finally the workings of CNNs are discussed in brief.

2.3.1 Support vector machines

The SVM classifier is a relatively modern classification technique frequently used in literature for computer-vision problems, and has been shown to be quite effective [LeCun et al., 1995]. The basis for SVMs as they are understood today was defined by [Cortes and Vapnik, 1995] for binary classification problems. The basic idea of this algorithm can be summed up as looking for the optimal hyper-plane, separating two classes by maximising the distance from the closest data points to the separating plane Meyer and Wien [2001]. This can be seen in Figure 2.4. The distance between the closest data-points and the hyperplane is often referred to as the margin, or the support vectors.

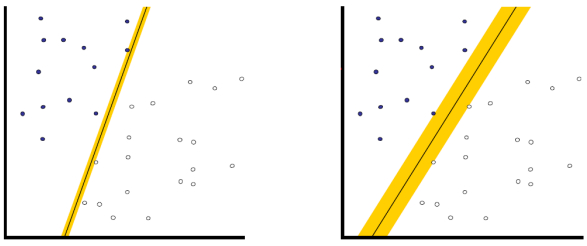


Figure 2.4: Hyper-planes separating 2 classes denoted by dark and light dots on a plane. a) A non-optimal separating hyper-plane with a small margin. b) A separating hyper-plane with a large margin [Moore, 2001].

These types of problems can be solved quite effectively using a principal called quadratic

programming. This type of simple classifier is called a Linear SVM (LSVM) and has a few challenges to overcome in real-world problems, such as noisy data, or classes that cannot be separated by a simple line, as well as allowing for classification of multiple classes. To allow data that cannot be separated by a simple line, it is possible to "warp" the data or project it to a higher dimensional plane where the points then are separable by a simple plane. This is demonstrated in Figure 2.5. The data points are projected to a higher dimension using some function, or combination of functions called "basis functions" [Moore, 2001]. There are a few types of basis functions such as polynomial, radial and sigmoid basis functions that can be employed to successfully cast data-points to higher planes [Moore, 2001]. In order to overcome the challenge of dealing with multi-class classification, a combination of SVMs are often trained and a voting system is used to determine the final class [Moore, 2001].

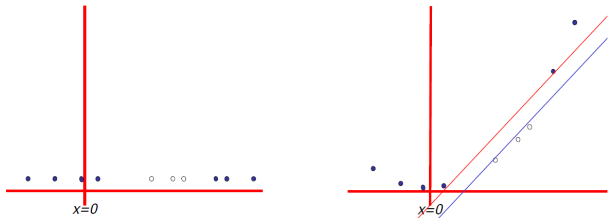


Figure 2.5: A simple demonstration of dealing with non-separable data by introducing higher dimensions a) Two non-separable classes on a one dimensional plane. b) By casting the data-points into a higher dimension the classes now becomes separable using a simple linear plane [Moore, 2001].

2.3.2 Neural networks

An ANN is a machine learning technique inspired by the biological processes used by animals to learn and reason in nature [Demuth et al., 2014]. It is a surprisingly old idea compared to other mathematical machine learning techniques such as SVMs, being first defined in the early 40's by McCulloch and Pitts [1943]. In their work, they showed that these simple artificial neurons could, in principle, learn to compute any logical function. This is considered the origin of neural networks [Demuth et al., 2014, Jain et al., 1996]. Over the next 40 years the problem of training these artificial neurons was addressed by the introduction of many new training algorithms, but it wasn't until the 80's that the back-propagation algorithm was introduced by Rumelhart et al. [1985], however there are reports of earlier work that made use of similar methods [Hecht-Nielsen, 1992]. This started a resurgence in the field of artificial neural networks. In recent years many of the advancements have been attributed to the greater availability of high-powered computers and bigger datasets. There have also been a wide variety of practical applications in recent years [Demuth et al., 2014].

2.3.2.1 The artificial neuron

The basic building block of ANNs are "artificial neurons" [Sharma et al., 2012, Jain et al., 1996, Haykin et al., 2009]. An artificial neuron is inspired by the structure of the biological neuron as depicted in Figure 2.6.

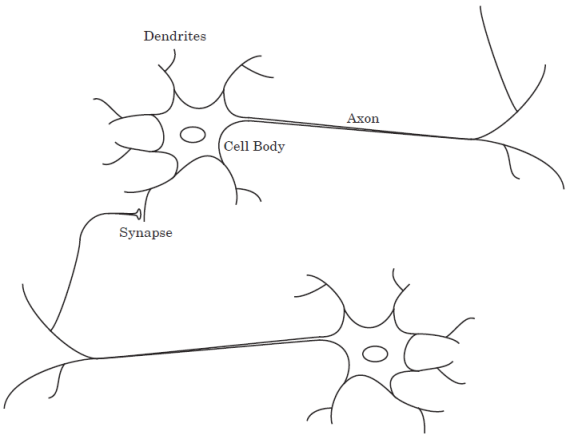


Figure 2.6: A simplified drawing of the structure of biological neurons [Gershenson, 2003, Demuth et al., 2014].

The biological neuron consists of three basic parts that are useful for the understanding of artificial neurons [Gershenson, 2003, Jain et al., 1996]. The first of these parts are the dendrites, a tree-like structure of nerves that are responsible for propagating electrical signals into the cell body [Demuth et al., 2014]. The cell body takes these signals and if the combined stimulus is great enough, the cell body "fires" sending a signal along the axon. The axon is a long nerve that moves the signal out of the cell body and to other neurons. The points where an axon of one cell connects to the dendrites of a neighbouring cell is called the synapse. The signal is moved from the axon to the following dendrite based on the strength of these synaptic connections. The strength of these connections are determined by a complex chemical reaction. Although there are some neural structures defined at birth, the structure is capable of changing throughout life. The structural changes happen mainly in the form of the strengthening or weakening of the synaptic contacts between neurons [Demuth et al., 2014, Jain et al., 1996]. This forms the basis on which artificial neurons are built.

An artificial neuron mirrors a simplified structure and functions of their biological counterparts [Gershenson, 2003]. The common notation for these artificial neurons can be seen in Figure 2.7. The neuron receives inputs from external sources or other neurons. Each of

the input values p_1 to p_R are then multiplied by the corresponding weight w_i to determine the value that is moved into the "cell". This mirrors the behaviour of the synapses and the dendrites. Once these values are inside the cell-body, these values, as well as a bias variable b are summed to determine the net stimulus the cell receives. It is important to note that both the weights and the bias variables of the neurons are completely adjustable and this adjustment is an important part of the training process. The resulting value n is then fed into a function f that determines the output of the neuron a. This function is referred to as an "activation" function and can be a linear or non-linear function [Demuth et al., 2014, Jain et al., 1996, Haykin et al., 2009].

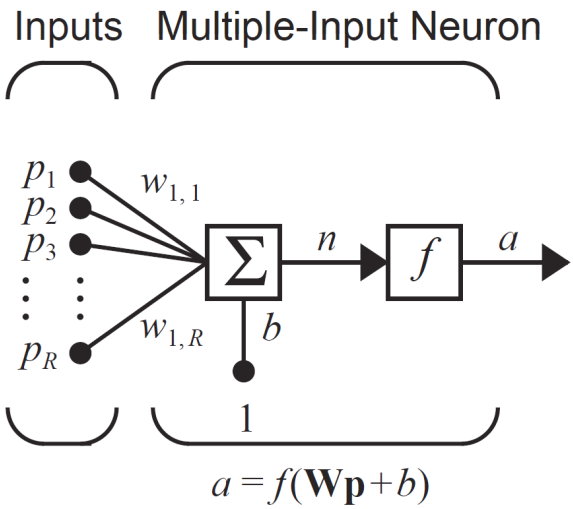


Figure 2.7: A conceptual drawing of the structure of an artificial neuron [Demuth et al., 2014].

The activation function is chosen based on the specific problem being addressed. Some of the popular activation functions are hard limit, linear transfer, and log sigmoid functions as can be seen in Figure 2.8.

The linear transfer function can be described as:

$$f(n) = n \tag{2.1}$$

The hard limit function can be described mathematically as:

$$f(n) = \begin{cases} 1, & \text{if } n \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

The log-sigmoid function can be described as:

$$f(n) = \frac{1}{1 + e^n} \tag{2.3}$$

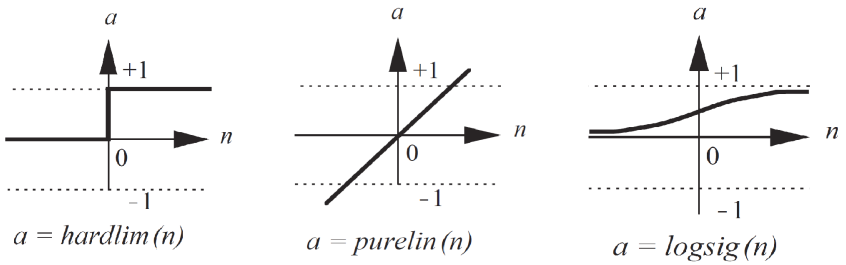


Figure 2.8: Some of the most common neuron activation functions: a) A hard-limit function. b) A purely linear function. c) A Log-Sigmoid function [Demuth et al., 2014].

2.3.2.2 Structure of neural networks

Neurons are typically structured into a series of layers in a feed-forward network Jain et al. [1996]. When stacking neurons into a fully connected layer, it is important to note that all outputs from the first layer are connected to every neuron on the following layer Jain et al. [1996]. This type of layer is called a highly connected layer. This holds true for every layer except for the output layer where the outputs of that layer are used as the output of the network [Demuth et al., 2014].

A simple layered network architecture usually takes the following structure. Firstly, there are a series of input values that are determined by the properties of the problem being solved. This is called the input layer. The input layer has as many neurons as there are input parameters, each parameter having a dedicated input neuron. By convention, many authors do not count this first layer when they refer to how many layers an ANN has Jain et al. [1996]. These input values are fed into the first "hidden" layer of the network. This layer contains the first stack of fully connected neurons, each connected to each of the inputs. There can be any number of hidden layers between the input and the output layers of the network. Each hidden layer is also allowed to contain any number of neurons. The final layer of the network is the output layer, where it is simply a layer of neurons corresponding to the number of output classes [Demuth et al., 2014]. A representation of a single layer of neurons can be seen in Figure 2.9.

2.3.2.3 Training neural networks

To train a neural network, there are generally five steps that are followed. The first step is to initialize the network with random values as the weights between neurons. These values are typically sampled from a zero-mean Gaussian distribution with a standard deviation of around 0,01 [Hinton, 2012] The second step is presenting the inputs from a training sample to the network and feeding the input values through the network to calculate an output.

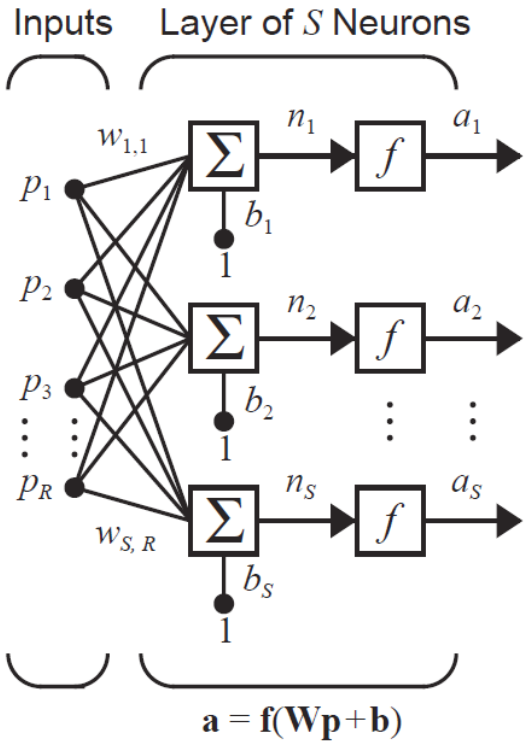


Figure 2.9: A single layer of neurons with multiple inputs [Demuth et al., 2014].

Then the "back-propagation" algorithm is used to calculate the error gradients of all the weights in the network and use gradient descent to adjust these weights [Demuth et al., 2014]. In other words, the weights are adjusted in proportion to how much they contribute to the total error in an attempt to minimize the error [Rumelhart et al., 1985]. Calculating how much the weights should be adjusted in a multi-layered neural network with non-linear transfer functions is an especially challenging task as the relationship between the network weights and the error is more complex.

Given a set of m input-output vector pairs (x, y), the network is given x as an input which produces some output. The produced output is compared with the with the expected output x in order to calculate the error. This is done by calculating how much the produced values differ from the expected values using the mean squared error formula [Demuth et al., 2014]:

$$TotalError = \sum \frac{1}{2}(values_{expected} - values_{output})^2 \tag{2.4}$$

The weights of the ANN are then adjusted in order to minimize the total error. The idea is to adjust the weights in the direction and magnitude that is expected to minimize the error produced by the network. This problem is addressed by the gradient descent algorithm. The gradient descent algorithm is quite complex, however it builds on the very simple principal of looking for the global minima [Thomas, 2017]. A simple one-dimensional error curve, seen in Figure 2.10, is used to demonstrate the basic gradient descent algorithm.

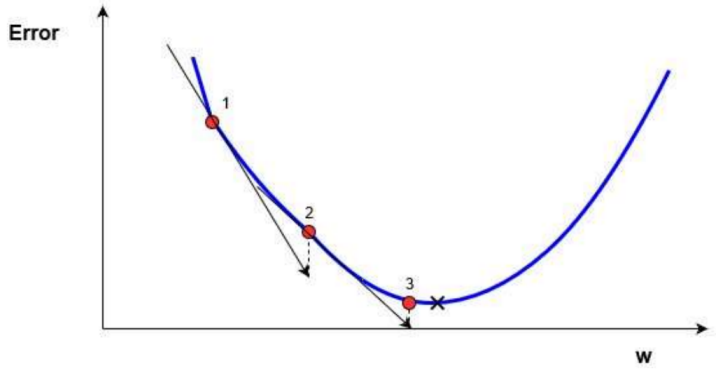


Figure 2.10: This figure depicts a simple curve that depicts the error produced by the ANN on the y axis, by the weight value w shown on the x axis. The red dot labelled 1 shows the error produced by the corresponding weight value during the first iteration. The red dot labelled 2 and 3 depict the error produced by the subsequent adjustments to the weight w. The black cross shows the true point where the error is minimized [Thomas, 2017].

The image shows the weight at which the error is minimized by a black cross, however, it is not known where this point is. The algorithm will attempt to search for this point by starting with a random value for w and iteratively calculating the error and making adjustments to the weight w. The weights are updated according to the following policy:

$$w_{new} = w_{old} - \alpha \times \delta Error$$

Where w_{new} is the updated value of the weight, w_{old} is the current value of the weight, α is the learning rate, and $\delta Error$ is the gradient of the error at w_{old}. The learning rate will describe by how much the weight should be adjusted in the direction of the gradient, and has a large impact on how fast the network will converge, and should therefore be tuned. As the weights approach a point close to the optimal values, the error gradient will approach 0, and the weights will no longer be adjusted drastically, resulting in only small improvements to the error. When this point is reached, the training is stopped [Thomas,

2017].

Often the gradient of the error of a function is simply calculated as the derivative of the function at w_{old}, however in neural networks the gradient descent algorithm should often search a multi dimensional function for the global minima. This is where the back-propagation algorithm comes in. In real-world examples, the error function is not as simple as the derivative of a simple function. There is a more general cost function that serves the same goal as the simple error function described previously. This cost function J is calculated using the sum-of-squares error for any given input (x^z, y^z) as:

$$J(w, b, x^z, y^z) = \frac{1}{2} \parallel y^z - y_{pred}(x^z) \parallel^2 \tag{2.5}$$

Where y^z is the expected output values for sample z, $y_{pred}(x^z)$ is the output produced by the ANN for the given input x^z. This cost function is for a single (x, y) pair. In order to minimize the cost function for m samples the error can be calculated using the mean squared error algorithm:

$$J(w, b) = \frac{1}{m} \sum_{z=0}^{m} \frac{1}{2} \parallel y^z - y_{pred}(x^z) \parallel^2$$

$$= \frac{1}{m} \sum_{z=0}^{m} J(w, b, x^z, y^z)$$

Where m is the number of samples. Given this cost function the weights in the ANN can be trained.

There is a notation that is used to describe each weight and bias value in the neural network. A weight can be described as w_{ij}^l where l is the layer, i is the neuron in the current layer, and j is the neuron in the previous layer. A bias can be described as b_i^l where l describes the layer and i is the neuron in the layer that this bias belongs to. The gradient for every weight w_{ij}^l and bias b_i^l can be calculated using the following:

$$w_{ij}^l = w_{ij}^l - \alpha \frac{\partial}{\partial w_{ij}^l} J(w, b)$$

$$b_i^l = b_i^l - \alpha \frac{\partial}{\partial b_i^l} J(w, b)$$

These equations are similar to the equation $w_{new} = w_{old} - \alpha \times \delta Error$ described earlier where the value to the left is the updated value, and the value to the right of the equation is the current value, α is the learning rate, and the gradient is calculated as the partial derivative of the cost function with regards to the weight or the bias. The problem with

ANNs are that there is no simple cost function that can be used to calculate the gradient. The back propagation algorithm allows the error to be shared to all the weights in the network and allows for the calculation of how much a weight is contributing to the total error.

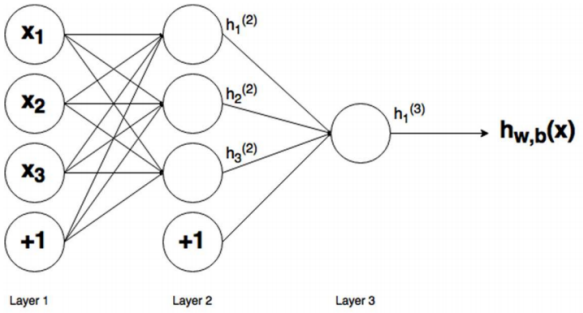


Figure 2.11: A simple 3 layer ANN with 3 inputs, three layers and one output.

In order to examine the back-propagation algorithm, the simple ANN in Figure 2.11 can be considered. In Figure 2.11 the notation h_i^l is used to show the activation of a node i in layer l. The activation of the output node for this ANN can be calculated as

$$h_{W,b}(x) = h_1^{(3)} = f(w_{11}^{(2)} h_1^{(2)} + w_{12}^{(2)} h_2^{(2)} + w_{13}^{(2)} h_3^{(2)} + b_1^{(2)})$$

The above can be simplified as $h_1^{(3)} = f(z_1^{(2)})$ where $z_1^{(2)}$ is

$$z_1^{(2)} = w_{11}^{(2)} h_1^{(2)} + w_{12}^{(2)} h_2^{(2)} + w_{13}^{(2)} h_3^{(2)} + b_1^{(2)}$$

In order to calculate how much a change of weight $w_{11}^{(2)}$ will have on the cost function J, the function $\frac{\partial J}{\partial w_{11}^{(2)}}$ has to be calculated. To do this, the "chain rule" in calculaus has to be applied. This states that given functions g and f, if g is differentiable at point c, and f is differentiable at point $g(c)$ then the function $f \circ g(x)$ is differentiable at c and that $(f \circ g)'(c) = f'(g(c)) \cdot g'(c)$ [Cottrill, 1999]. Applied to this problem we can write the following:

$$\frac{\partial J}{\partial w_{11}^{(2)}} = \frac{\partial J}{\partial h_1^{(3)}} \frac{\partial h_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

This means that it is possible to create $\frac{\partial J}{\partial w_{11}^{(2)}}$ from a few other partial derivatives. The first

partial derivative $\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$ can be calculated as

$$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = \frac{\partial}{\partial w_{11}^{(2)}}(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)})$$

Because the partial derivative of $z_1^{(2)}$ will only operate on the term $w_{11}^{(2)}$, all other terms will remain constant and, when the derivative is calculated, will give a value of one, these terms can be removed.

$$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = \frac{\partial}{\partial w_{11}^{(2)}}(w_{11}^{(2)}h_1^{(2)})$$

$$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = h_1^{(2)}$$

This results in the derivative being the activation of the node in question. The next partial derivative in the chain is the term $\frac{\partial h_1^{(3)}}{\partial z_1^{(3)}}$, which is the partial derivative of the activation function of the output node. This means that any activation function should be differentiable. If the output node had a sigmoid activation function, the derivative of the function would be

$$\frac{\partial h}{\partial z} = f'(z) = f(z)(1 - f(z))$$

where $f(z)$ is the activation function.

Finally to calculate the derivative of the last term $\frac{\partial J}{\partial h_1^{(3)}}$. This is the derivative of the loss function as described in Equation 2.3.2.3. Using the chain rule, this equation can be written as:

$$Let\ u = \parallel y_1 - h_1^{(3)}(z_1^{(2)}) \parallel \ and\ J = \frac{1}{2}u^2$$

$$Using\ \frac{\partial J}{\partial h} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial h} : \frac{\partial J}{\partial h} = -(y - h_1^{(3)}) \tag{2.6}$$

Where y_1 is the target value for the output node. These derivatives can be combined into the general term

$$\delta_i^{(n_l)} = -(y_i - h_i^{(n_l)}) \cdot f'(z_i^{(n_l)}) \tag{2.7}$$

Where i is the node number of the layer n_l. This means that the complete cost function derivative for a node can be written as

$$\frac{\partial}{\partial W_{(ij)}^{(l)}} J(W, b, x, y) = h_j^{(l)}\delta_i^{(l+1)} \tag{2.8}$$

This equation makes sense for the output layer as the error can be directly calculated. However, for weights that are not connected to the output and are deeper within the neural network, the backpropagation algorithm is used. The only connection that each node has to the output of the network is through the $\delta_i^{(n_l)}$ term that describes how any given node contributes to the output, and thus, this term is propagated through the network.

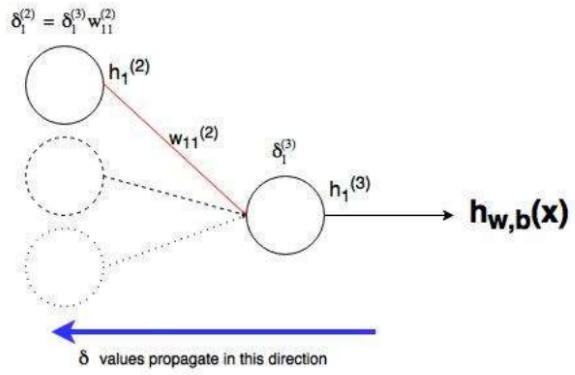


Figure 2.12: An example of how δ is back-propagated to a node in the previous layer with only a single node in the output layer, using the weight of the connection to the current node.

Figure 2.12 depicts how δ gets back propagated from a single output node to a node in the previous layer, using the weight between these two nodes to update δ as it propagates back. In a single neuron example like in Figure 2.12 the updated δ function can be written as

$$\delta_j^{(l)} = \delta_1^{(l+1)} w_{1j}^{(l)} f'(z_j)^{(l)} \tag{2.9}$$

where j is the number of the node in layer l.

In a case where there are more than one output node in layer $l+1$, as depicted in Figure 2.13, $\delta_j^{(l)}$ is calculated using a weighted sum of all the error functions from the following layer. This means that each δ is used, however it is weighted according to their respective weights. This can be written as:

$$\delta_j^{(l)} = (\sum_{i=1}^{S_{(l+1)}} w_{ij}^{(l)} \delta_i^{(l+1)}) f'(z_j^{(l)}) \tag{2.10}$$

where j is the node number in layer l and i is the node number in layer $l+1$. $S_{(l+1)}$ denotes the number of nodes in the layer $l+1$.

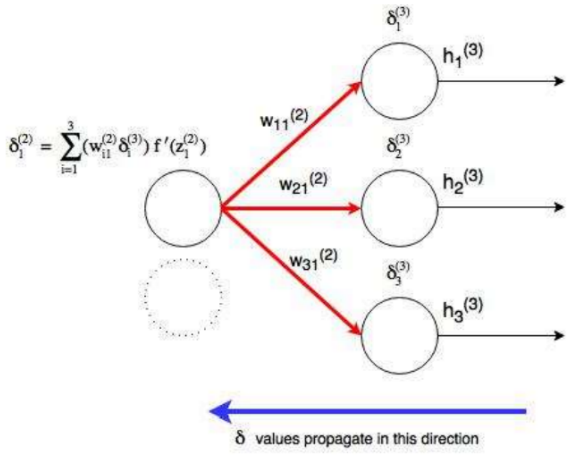


Figure 2.13: An example of how δ is back-propagated to a node in the previous layer with many nodes in the output layer, using a weighted sum of the the errors propagated from the next layer.

The gradient of a weight can be calculated using the following expression

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b, x, y) = h_j^{(l)} \delta_i^{(l+1)} \tag{2.11}$$

The same application of the chain rule can be used to arrive at the derivative of of biases.

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b, x, y) = \delta_i^{(l+1)} \tag{2.12}$$

With these two derivatives calculated, the original gradient descent problem can be calculated as:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial}{\partial w_{ij}^{(l)}} J(w, b) \tag{2.13}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(w, b) \tag{2.14}$$

This is not an efficient way to calculate the updated weights and biases computationally, however, how the algorithm can be sped up is outside of the scope of this section. With the standard gradient decent algorithm, the final gradients by which the weights are adjusted are calculated by looking at the gradients generated by looking at the entire dataset. However, this is often very slow when the dataset is very large because adjustments to the ANN are only made after the algorithm has looked at every sample in the dataset. One solution

to this is using Stochastic Gradient Descent (SGD), where adjustments are made after the algorithm has calculated the gradients of single, or small batches of samples. This is often called iterative, or stochastic gradient decent because by not looking at all the samples, the calculated gradient is not the exact gradient but rather a stochastic approximation of the true gradient. This has been shown to converge quicker, especially when used with other learning improvements such as adaptive learning rates and momentum [Thomas, 2017, Haykin et al., 2009].

After the weights have been adjusted one would expect the network to produce results closer to the desired output if we were to propagate the same input image through the network. This would mean that the network has successfully learnt to classify a certain set of input data. The process of feeding a set of input values into a network and adjusting the weights according to the error is repeated for every example in the training set [Demuth et al., 2014]. There are many more aspects to the successful design and training of neural nets that fall out of the scope of this discussion. Besides the structure of the neural network, there are a few hyper parameters that are responsible for various aspects of the learning process and will have an effect on the overall performance. Some of these parameters are learning rate, decay, and momentum.

Learning rate When calculating the error and the optimal gradients to minimize the error, it is often the case that if these weights were to be applied to the network weights, the network would often over-correct and "forget" what was previously learned and have a difficult time learning general features. To combat this, a learning rate is introduced. Every calculated gradient is first multiplied by the learning rate before it is subtracted from the current weights [Haykin et al., 2009]. Values can range from $0,0001$ up to 1, but a typical learning rate is around $0,01$ [Demuth et al., 2014].

Weight decay It is often useful to reduce the learning rate as the network approaches the optimal set of weights. It is also useful for preventing over fitting of training data by constraining the values that weights are allowed to take. It also allows very large weights to be shrunk, resulting in fewer "useless" weights where the connected neurons are always on or always off. Reducing the learning rate like this is often called "decay" and has been shown to improve training [Hinton, 2012]. This works by simply adding an extra term to the normal gradient that penalizes large values. This simplest form this term could take is called "L2 Decay" and this penalty is calculated as half the sum of the squared weights times a coefficient. Typical values for L2 are usually between 0,01 and 0,0001 [Hinton, 2012].

An alternative to L2 decay is a method called "L1 Decay" is calculated as the derivative of the sum of the absolute values of the weights. This type of decay often leads to more interpratable and localized receptive fields for CNNs [Hinton, 2012]. According to Hinton

[2012] small changes in the decay values are unlikely to have a dramatic effect on the performance of the ANN.

Convergence and momentum Often when training a neural net, there comes a point where the learning algorithm calculates the gradient by which the weight should be adjusted as 0 or very close to 0 and does not change the structure of the neural net. This is called convergence, and this occurs when the error function is minimised [Demuth et al., 2014, Haykin et al., 2009]. The hope is that this occurs on the global minima, at which point the network is fully trained, but could often occur on a local minima, meaning that the network is not truly optimized, but is unable to continue training as the current values seem optimal. There are methods of dealing with this problem such as the introduction of "momentum", a way to allow gradient descent to continue through local minima [Demuth et al., 2014]. This works by simulating momentum when adjusting the gradients. The weight vector could in effect build up "velocity" in the direction with consistent gradients allowing it to descend much quicker than simple steepest descent and also travel over local minima. This value is usually between $0,5$ and $0,9$ as the new weights will now be calculated as the gradient times the learning rate, multiplied by $1/(1-\alpha)$ where alpha is the current total momentum [Hinton, 2012].

Over-fitting and generalization It is possible for neural networks to be trained on the training data, and the error on these examples to be very small, but as soon as the network is asked to classify unseen data, then the network fails [Haykin et al., 2009]. This is called "over-fitting" and is caused by a network having too many "free parameters" or weights and biases [Demuth et al., 2014, Haykin et al., 2009]. This could be translated into a network having too many neurons for the complexity of the training data.

One of the methods used to deal with over-fitting is the introduction of a dropout layer [Demuth et al., 2014]. In this layer, some of the weights are randomly dropped during each training iteration. This forces the network to look for more general properties and prevents the network from simply mapping inputs to outputs.

The opposite of over-fitting is when a network performs comparably well on unseen data as on training data. This happens when a network manages to generalize [Demuth et al., 2014]. Generalization means that the network managed to learn general properties of the data that can be applied to any example it is presented with, and not just a simple mapping between training inputs and training outputs.

One of the suggested methods for ensuring improved generalization is called "early stopping" where an unseen validation set of data is used and tested against [Demuth et al., 2014]. This would allow the user to monitor how well the network has generalized and stop

the network when it is optimal.

2.4 Convolutional neural networks

Convolutional neural networks are a special type of feed-forward neural network that has its origins in the study of the visual cortex of spider monkeys. It has the interesting property when used as features in DPMs, that it does both feature generation and extraction, and classification, making it an attractive option.

Hubel and Wiesel [1968] conducted a study, focusing on identifying different types of cells and architecture contained in the visual cortex of spider monkeys. They attempted to identify different types of cells in a certain part of the brain by stimulating the retinas of a lightly sedated spider monkey. Those experiments showed that the visual cortex consisted of two basic cell types. Simple cells that are stimulated by straight lines with specific orientations within their receptive fields, and complex cells that are stimulated by larger shapes that are insensitive to the exact positions of the edges in the field.

The findings by Hubel and Wiesel [1968] was used in the 80s to develop a special type of neural network called the "neocognitron" consisting of a variety of different cells [Fukushima, 1988]. This neural network has the unique property of having different types of cells. Some cells are dedicated to extracting local features while other cells are used to combine these features and handle deformities such as shift. This paved the way for modern convolutional neural networks as we know them today.

The first use of "modern" CNNs was pioneered by the work done by LeCun through the 1990's building on the successes of the work done in the late 80's [LeCun et al., 1990, Matan et al., 1990, Bengio et al., 1994, LeCun et al., 1995]. The structure of the modern CNN was outlined in the 1990 paper entitled "Handwritten digit recognition with a back-propagation network" by LeCun et al. [1990]. In this paper the author also outlines the now famous LeNet5 network architecture that achieved remarkable results on various handwritten digit classification tasks. What makes this work so revolutionary is the fact that, unlike its predecessors, there was no preprocessing done on the images and the images were fed directly to the network, and the network was able to classify these images with remarkable accuracy. This demonstrated the ability of CNNs to deal with large amounts of low-level data. Although some of the applications of CNNs has been seen as early as the 90s, the use of CNNs haven't been widespread until the ImageNet image classification challenge was won by a CNN [Krizhevsky et al., 2012].

Modern convolutional neural networks have 3 distinguishing features [Dumoulin and Visin, 2016]. The first is that layers of neurons are connected in a 3-dimensional configuration

[Dumoulin and Visin, 2016, Le et al., 2015]. Each 3-dimensional layer has height, width and depth, with height and width corresponding to the spatial dimensions of the input layer and the depth to the number of local features or "filters" the neural net is looking for [Dumoulin and Visin, 2016, Le et al., 2015]. The second defining aspect of CNNs is the fact that neurons are locally connected, with each neuron only connected to a small number of spatially local neurons from the previous layer [Dumoulin and Visin, 2016, Le et al., 2015]. This spatially local connection is called the "receptive field". This structure allows each locally connected neuron to respond to a feature described by the weights of that receptive field. The following layers are then able to combine the previously observed features into more complex and increasingly global features. The final distinguishing feature of neural networks is the fact that weights are shared between groups of neurons [Dumoulin and Visin, 2016, Le et al., 2015]. The weights between every neuron to the local inputs are the same for every neuron that looks for a specific feature. Neurons that look for the same features in different locations of the input image are grouped into what is called a "filter". The output of one of these filters is a spatially correlated set of activations that describe the presence of a certain feature on the input image. These layers extract low-level features in the earlier layers and higher level features in later layers. The final layer is usually a classic fully connected layer [Dumoulin and Visin, 2016].

These layers are the basic building-blocks of CNN architectures. In the following section the workings of each layer and how it aids in accurately classifying images is discussed.

2.4.1 Convolutional layers

Convolutional layers derive their name from the convolution operator commonly used in image processing tasks. This is to iteratively apply some function to a subsection of pixels by "sliding" the kernel over the image. Convolutional layers take the raw image values as input and produce an output describing the prevalence of certain features [LeCun et al., 2010]. In CNNs this means that a certain set of weights look at a small subsection of the input image. These weights can be represented as a matrix [Dumoulin and Visin, 2016]. As with classical neural networks, the input values are multiplied by the weights connecting them to the neuron in the next layer. The value of the output neuron is calculated as the sum of the input values multiplied by their respective weights [Dumoulin and Visin, 2016]. It is important to note that unlike classical neural networks a CNN has the unique property of having shared weights from each patch of input neurons to each output neuron. A set of weights that are learned and shared across a layer is referred to as a "filter", "feature detector" or a "kernel" [Dumoulin and Visin, 2016, LeCun et al., 2010]. We will refer to a set of shared weights as a filter. The value of each output neuron is equivalent to the similarity of the input patch to the learned feature encoded in the weights of that layer. This can be seen demonstrated in Figure 2.14.

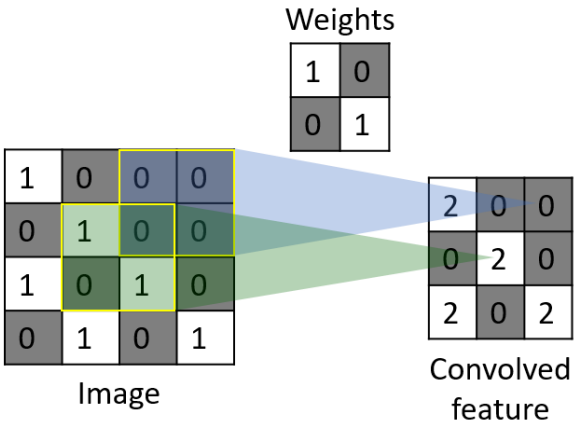


Figure 2.14: A convolution of a single filter over an image and the output result. The input layer is shown on the left and the activation of the neurons in the filter is showed on the right hand side. The weights that connect each neuron in the filter layer to the local input values are shown in the weights grid at the top of the image. The produced output shows high values where the input matches the weights well and low values where the input does not match the weights well. This output layer is essentially a map of the positions where the feature described by the weights are present.

The output produced by a convolution of a filter is referred to as a "feature map", "convolved feature" or "activation map". The output or the feature map is a map representing the presence and position of a learned feature in the input image. This has the interesting property of being able to extract certain features from an image, but preserving the spatial relationship between these features. Each filter will produce a different output matrix because the features that each filter is responding to is different. One of the powerful features of CNNs is that these filters are learned during training, meaning that the network could learn only the features most useful in recognizing the images it is trained to recognize. It is also important to note that the number of filters still needs to be specified during network design. The more filters we allow the network to learn, the more features and feature combinations we allow the network to look for during classification. The size of the resulting feature-map is based on 3 parameters. These are discussed briefly, as this is one of the unique features of CNNs.

Depth One of the interesting features about CNNs is the fact that it can contain 3D layers of neurons. The first parameter influencing the output layer size is the number of filters the network is to learn. Each feature produces its own 2D matrix. These feature maps are usually represented as stacked layers of neurons as can be seen in Figure 2.15. In Figure 2.15 the "depth" of the feature map can be said to be 4, because there are 4 filters each producing its own 2D matrix of responses. It is also not uncommon for input layers to

be 3 dimensional as there are usually an input for each of the basic colour channels namely red, green and blue.

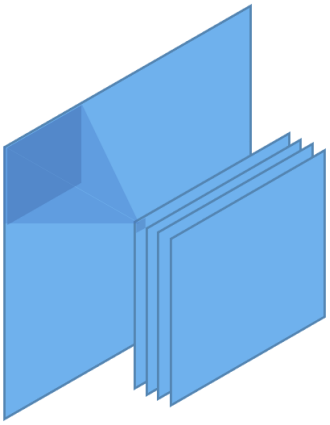


Figure 2.15: One input layer producing a number of filter maps in the following layer

Stride Note that in Figure 2.14, the input patches overlap by one pixel width. This is because of a property of the convolutional layer called the "stride" and describes the distances between input patches [Dumoulin and Visin, 2016]. The stride can be any number of pixels greater than zero as demonstrated in Figure 2.16. Larger stride distances produce less dense feature maps as output resulting in a feature map with a smaller height and width. Another way of thinking about stride is not as how much the kernel is translated but rather as a type of sub-sampling where only some of the produced output is retained [Dumoulin and Visin, 2016].

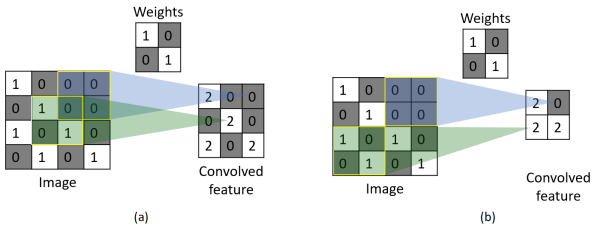


Figure 2.16: a) A convolution with a stride of 1 pixel. b) A convolution with a stride of 2 pixels.

Zero-Padding Zero-padding is the practice of adding a border zero's of some width around the edge of a matrix [Dumoulin and Visin, 2016]. This can be useful in controlling

the size of the output matrix [Dumoulin and Visin, 2016]. Doing convolution on a matrix without zero-padding is sometimes referred to as "narrow" convolution, and when doing convolution on a zero-padded matrix it is often referred to as "wide" convolution.

2.4.2 Rectified linear units

A rectified linear unit, also referred to as "ReLU", is a non-linear activation function applied to each neuron in a layer in order to prune the negative values [Xu et al., 2015]. This is just one of the many possible non-linear activation functions that could be used in CNNs, however, this seems to be one of the methods that are commonly used. This very simple activation function simply replaces all negative values in the input matrix with zero. Using this type of activation function has proven to improve training time, and also improve discriminative performance over networks using sigmoid activation functions [Dahl et al., 2013]. Neural networks are often described as universal function approximators [Nielsen, 2015]. This means that sufficiently complex neural networks could be used to approximate any function [Nielsen, 2015]. If neural networks are thought of as universal function approximators, it makes sense that the introduction of non-linearities is conducive to creating complex functions using simple functions. The activation function is often expressed as $Max(0, x)$ but can be written in a more mathematical expression as:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

This simple non-linear activation function has been shown to perform better than other activation functions such as sigmoid or tanh functions in a variety of computer vision tasks [Dahl et al., 2013, LeCun et al., 2010]. More recent research has shown the effectiveness of so called "leaky ReLU's" in various machine learning tasks [Xu et al., 2015, LeCun et al., 2010]. Leaky ReLUs are similar to normal ReLU functions in that it does not affect positive values, however, these functions produce diminished negative values for inputs smaller than 0. Some modifications on this allows for the rate by which negative values are diminished to be learned, called parametric ReLU, or to be randomized during training, called randomized ReLU. From the literature, it seems that there are three predominant ReLU functions as can be seen in Figure 2.17. Experiments have shown that Leaky ReLU and Randomized ReLU, outperforms Parametric ReLU and normal ReLU layers.

2.4.3 Spatial pooling layers

One of the big challenges CNNs face is trying to effectively and meaningfully reduce the high number of input values to meaningful features. The way this problem is solved is by the use of a technique called spatial pooling. These pooling layers are designed to reduce the dimensionality of a feature map by retaining only the most important features [Scherer

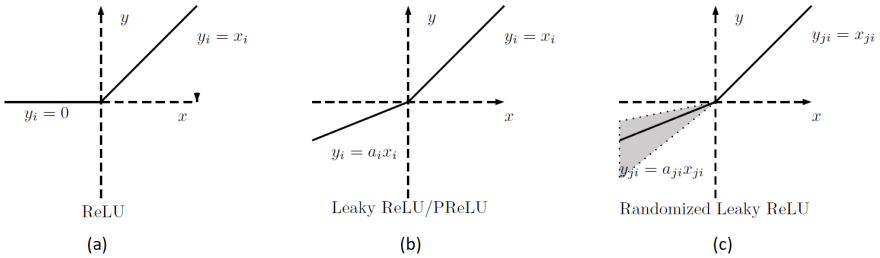


Figure 2.17: a) ReLU function. b) Leaky ReLU, where a_i is fixed and Parametric ReLU where a_i is learned. c) Randomized ReLU where the a_{ij} is a random number sampled withing a range during training and is kept consistent during testing [Xu et al., 2015].

et al., 2010, Dumoulin and Visin, 2016, LeCun et al., 2010]. Reducing the number of tunable parameters in the model also has the effect of managing over-fitting of the model. Another benefit that the introduction of these layers have, is the fact that it makes the model somewhat robust to small variations of location [LeCun et al., 2010]. There are a couple of pooling architectures such as max, average and sum-pooling [Dumoulin and Visin, 2016]. Pooling operations work by looking at a patch of values in the input matrix such as a 2×2 region and applying the pooling function to produce the output value. The most commonly used algorithm is max pooling, where the highest input value is used as the output value for that region [Scherer et al., 2010, Boureau et al., 2010]. Pooling layers also have a stride as can be seen in Figure 2.18.

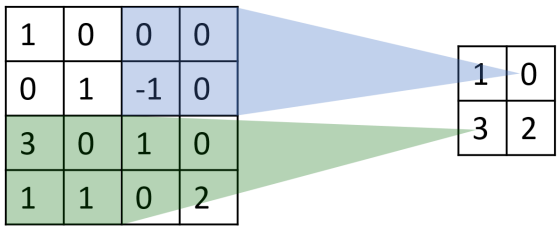


Figure 2.18: An example of max-pooling with a width of 2 and a stride of 2

Max pooling layers usually have a stride that is similar to the window width to prevent overlapping as there are no significant performance increases observed over non-overlapping pooling windows [Scherer et al., 2010]. It has also been found that max-pooling outperforms other techniques on some classification tasks [Boureau et al., 2010].

2.4.4 Classification layers

All the layers up to this point have been focused on extracting features from the raw images. Successive convolution, ReLU and pooling layers combine lower level learned features into higher-level, more descriptive and somewhat scale and translation invariant features. A visual example of this can be seen in Figure 2.19.

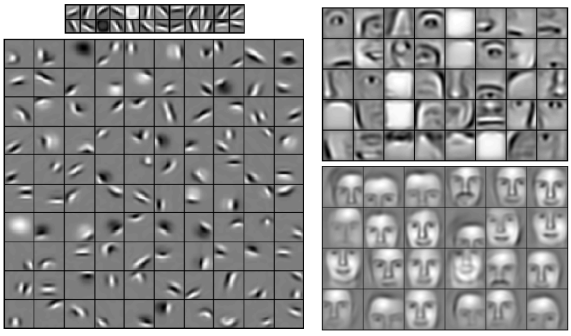


Figure 2.19: Example images of learnt features reconstructed from different layers in a CNN [Lee et al., 2009]. On the top-left the simple features detected by the first layer can be seen. The bottom left shows the features that the second layer looks for. These features are made up of combinations of the features detected in the first layer. On the top-right the features from the third layer can be seen. Finally on the bottom-right the features from the fourth layer can be seen. The aim is to demonstrate how a CNN is able to combine low-level features in the first layers into more complex shapes in later layers.

After all these features have been extracted the classification task falls to a few classical artificial neuron layers [Guo et al., 2016]. These layers are fully connected, meaning that every node in the input layer is connected to every node in the output layer. Unlike convolution layers, every connected neuron has a learned weight and no weights are shared and most of the learned parameters are contained in this layer [Guo et al., 2016]. The large number of parameters make this layers very expensive to train [Guo et al., 2016]. The feature outputs of the previous convolution layers act as inputs for the layer of artificial neurons. This layer is also responsible for combining all of these high-level features in order for the network to make good classifications. These nodes also sum all of the input values and the result is fed through an activation function. Common activation functions are sigmoid, tanh and ReLU functions.
The final layer of the fully connected layers and also of the entire network is the output layer. The output layer corresponds to the number of classes that are being classified and the output neurons often have an activation function that ensures that all the output probabilities have values between 0 and 1 and that the sum of all the classes equal 1. This is due to a special activation function called soft-max that has the ability to take a collection

of arbitrary real values and transform each value to be between 0 and 1 and all values to sum to 1.

2.4.5 Training of a CNN

We can see that the role of the pooling and convolutional layers are simply to extract meaningful features from the raw image and the fully connected layers act as the classifier. Convolutional neural networks are trained using a standard back-propagation algorithm as described in classical neural networks. Each layer type has a separate derivative that is used during back propagation and these are well described in literature. This is because the layers have quite a different structure and effect on the resulting error. It is important to note that the hyper-parameters such as the number and size of filters are fixed into the architecture and do not change during training.

2.4.5.1 Subject independence

There is a problem with training of ANNs, that is specifically important when it comes to the classification of faces or related data, although it applies to all forms of ANN training. This is the fact that faces are all unique and differ based on the individual [Matsugu et al., 2003]. In order to ensure that the CNN is able to generalize to all faces and not only to those of the subjects used during training great care should be taken to ensure "Subject independence" [Matsugu et al., 2003]. This means that the faces used for training of the CNN should not be the same as those used during testing and validation of the CNN. Good performance on a subject independent dataset should translate to good performance with other unseen samples.

2.5 Summary

In this chapter a short history and description of DPMs was given. A set of common feature generation techniques for DPMs are listed. An overview on basic neural networks was given and was followed up by an explanation of the basic workings of the layers of CNNs. A general method for the training of ANNs was discussed as well as many techniques that could be applied to improve performance such as ReLU activation functions to improve training accuracy and time. From the literature it was shown that hierarchical part models offer major improvement over traditional part models when it came to identifying low level feature points as well as handle occlusion. It was also evident that more advanced feature extraction and classification techniques such as CNNs offer a major advantage over simpler methods such as HOG features with an SVM as a classification technique.

Chapter 3

Literature Review

In this section, the aim is to compile a collection of works related to the field of face-region localization. This is by no means an exhaustive list, however, it should highlight some of the main approaches taken in the alignment of face-regions. In this section face-region localization refers to the alignment of facial regions as points or patches of pixels to a face image. This could mean that patches such as eye, nose and mouth regions are localized on an image of a face, as well as points corresponding to the edges of these features are calculated as a set of coordinates. This chapter is divided into two main sections. Section 3.1 contains work that achieves face alignment for samples without significant occlusion present. Section 3.2 contains work that is specifically focused on face-alignment on samples containing occlusion. Each section is then divided according to how the features are generated.

3.1 Region localization without significant occlusion

Detecting and localizing facial feature points reliably on an image is often the first step towards applying localization techniques in many practical applications. When localization is done on samples where no, or very little occlusion is present it significantly reduces the complexity of the problem as there aren't any sources of significant noise present in the images, and all the expected parts are visible. This also removes the need to interpolate the locations of hidden parts from the visible parts. There are various ways of approaching these problems and the field is relatively well explored. In this section a few of the most popular techniques are explained.

3.1.1 Region localization using hand-crafted features

Hand-crafted features are features that are designed to extract meaningful information from the raw pixels that can either be used as an evaluation metric in searching for a region or be used as input to a classifier. These features usually serve to reduce the complexity of

considering raw pixels as input and describes patches of raw pixels by its most significant feature. It often also allows variability such as lighting differences to be ignored. For example, one popular feature, the HOG descriptors, extracts the dominant gradients from patches of pixels that are normalized to remove lighting variations and are used as an input to a classification algorithm. It is also important to note that these features are not general-purpose and not specifically designed to represent domain specific image patches. This type of preprocessing is what is referred to as hand-crafted features.

3.1.1.1 Deformable part models

DPMs were based on an old idea called "pictorial structures" originally proposed by Fischler and Elschlager [1973]. In work done by Felzenszwalb et al. [2008], they proposed a system for general object detection that was highly accurate, fast and far outperformed the results achieved by similar systems at the time. Their work achieved a two-fold increase in detection accuracy over the best-performing systems in the 2006 PASCAL person detection challenge. Before their work, DPMs were often outperformed by simpler systems such as rigid templates. This system used a sliding window approach that convolved a "root" filter over an image. Several part models were attached to the "root" filter. Each part model specified a spatial model and a part filter. The spatial model described a set of plausible placements of the part relative to the "root" model as well as a deformation cost for each placement. The part filter defined the visual representation of the part using a robust feature descriptor. In this work, the feature descriptor that was used were HOG features. There were two scales of HOG features used in this work. The first was a coarse set of features that covered the entire object being detected. The second was a set of finer grained features for each of the part models that can move relative to the root filter. This structure meant that the entire model could be arranged in a star topology where the coarse template served as the root node. The part model was often convolved over the image and a score was given to each placement of parts based on the likelihood of an object being present in that position as calculated by some scoring function. The score for the detection at each point in the image was calculated as the score of the root filter, plus the sum of the parts, where the score for each part is the best detection for that part, minus the deformation or "spring" cost. Detection score for each part filter, as well as the root filter was calculated as the dot product between the HOG features of the part and the image at that position. The part models were defined using twice the spatial resolution of the root model as these require greater detail to be placed accurately. All the models in this method were defined with fixed sizes. To deal with scale differences a "pyramid of features" was used. This means that the input image was scaled to different sizes and the model of fixed size was iterated over the image. It is interesting to note that the score in this work was calculated as the dot product of the part filter and the HOG features of the image, and a simple threshold was used to classify positive samples rather than a classifier.

3.1.1.2 Mixtures of trees

Zhu and Ramanan [2012] proposed a system that simultaneously addresses face detection, pose estimation and landmark estimation. This model was based on a mixture of trees with a shared group of parts. This worked by modelling every facial landmark as a feature and capture topological changes by using mixtures of trees. The system was based on a group of parts each corresponding to a single feature point on the face. Each part had multiple templates that correspond to the appearance of that feature point in different poses. The templates associated with the feature points were HOG descriptors calculated during training. The entire model was arranged in a tree structure, meaning that all parts were connected to its neighbours without forming any closed loops. This method used a series of views, each view, corresponding to a change in pose. Certain parts will only be visible in certain views. Parts were also shared between views, meaning that a high number of views could be modelled with low complexity. Views could be mixed to create a large variety of possible face structures and poses. The tree structure of the proposed model allowed for dynamic programming to be used to traverse the possible search space efficiently. This system did not use a classifier, but simply looked for a solution that optimized the value for a scoring function. The system was measured against Active Appearance Models (AAM), as described in section 3.1.2.1, and Contrained Local Model (CLM) implementations on the MultiPIE dataset that contains 750000 images of 337 people from different viewpoints. The average error was normalized, not over the inter-ocular distance, but rather over the width of the face. This was because the inter-ocular distance assumes that both eyes are visible. This system managed to outperform comparable systems with a relative error of only 2,3% in comparison to the CLM implementation with an error of 2,8% relative error.

3.1.2 Region localization using learned features

In this section learned features are refer to features that are specifically learned in conjunction with a classification or searching algorithm such as CNNs. This means that no pre-processing was performed prior to part localization that describes pixel patches in terms of its most defining characteristics, and removes variability. Learned features are also intended to be task specific, meaning that it will only be able to represent the features necessary to effectively represent images from the domain in question.

3.1.2.1 Active appearance models

One of the popular methods to deal with face alignment or facial feature point localization, and does not rely on the use of hand crafted features, is Active Appearance Models (AAM) developed by Cootes et al. [1998]. This system was based on the idea of "interpretation by synthesis". It worked by creating a "shape-free" version of the face region and using Principal Component Analysis (PCA) on the face appearance data. The same was performed on the face label data. While training on a dataset of labelled faces, a statistical

model of shape variation was built. This produced a set of meaningful parameters that could be changed in ways that correlate to concepts that are intuitive, such as face width or degrees of smiling. The "shape free" face model with tuned parameters was mapped to the geometric labels to synthesize an image that could be compared to the image being tested. A search algorithm was used in order to modify the learned parameters and placement of the image. At each iteration, the difference between the synthesized image and the test image was compared in order to calculate the accuracy of the parameters of the synthesized model. This was done using a derivative estimate combined with the current error which determines the next direction of search. This process continued until convergence. Training of the method proposed by Cootes et al. [1998] was performed using a small set of 88 hand-labelled images and testing was done on a different dataset of about 100 faces with an average width of 200 pixels. Experiments show that models usually converged within 20 iterations and produced results with a median error of 3,5 pixels given good initial estimations. It is important to note that these results were not normalized over inter-ocular distance as is the norm for many papers in this field. This method was extremely robust to pose and shape variation but was however quite sensitive to initial estimation, with performance deteriorating as the initial estimate deviated from the true location of the face. This method also did not explicitly handle occlusion and was also most applicable to almost fully frontal face images. There have been many variations on this method, including variations that cater for the use of AAMs on occluded images [Gross et al., 2006].

3.1.2.2 Region localization using CNNs

Work done by Matsugu et al. [2003] achieved remarkable face detection results using series of CNNs trained on different face-parts at different scales. CNNs allowed the classification algorithm to dynamically train feature extractors during training of the classifier, only extracting the features that contribute most to the final classification. In their network each layer was trained individually. One very interesting aspect of their work is the fact that they only used their CNN for feature extraction. The output of the network was not a class, but rather the output of the final convolutional layer. This resulted in an image with a response of a filter detecting whole faces. One of the interesting features of their proposed CNN is the fact that it also took the results of skin-colour segmentation of the image as an input for face detection. They found that the use of more positive samples compared to negative samples resulted in more robust localizations. They used about 14700 images to train the face detection layer. They also varied the sizes of the input parts from 0,7 to 1,5 times the size of the original parts to make detection more robust. Their proposed system was able to consistently achieve results of 97,6% accuracy on smile detection and displayed remarkable robustness to variations in pose and occlusion. This same architecture was then used to classify facial expressions from video sequences. One of the insights of this work was that the training on robust face detectors using only a fraction of negative samples

during training can improve detection rates. This was one of the first instances of a subject independent facial expression system with robustness to variation in both the appearance and location of the facial region [Matsugu et al., 2003].

Work done by Savalle et al. [2014] focused on the extent to which the use of CNNs in the DPM paradigm can improve performance given that the use of CNNs in other computer vision tasks have yielded such drastic improvements. This work also addressed the shortcomings of the widely used Region-based (R-CNN) method, specifically in the application of landmark localization as well as pose estimation. This was because R-CNN only partially captures the complexity of object parts as well as the relationships between them. This was tested by taking the work done by Felzenszwalb et al. [2010] and simply replacing the image representation from HOG features to CNN features. The performance was then measured against the HOG features as a baseline. The CNN architecture that was used was based on the first five layers of the CNN proposed by Iandola et al. [2014] and later layers using the fine-tuning proposed by Girshick et al. [2014]. This work focused on object recognition and classification. These experiments concluded that in this case, the use of CNN features over HOG features resulted in a substantial performance increase however it was still lagging behind the results of R-CNN for detection and classification. The average precision as a percentage was 48,2% versus the 33,7% achieved by the DPM with HOG features.

Nguyen et al. [2015] argued that both CNNs and DPMs have distinct advantages that could be combined to improve detection accuracy. They argued that deep neural networks are a good fit for detection tasks as they can effectively model high-level appearance of images. However, CNNs fall short when it comes to the ability of DPMs to model a face in a more flexible structure. This is because deep neural nets fail to effectively model the explicit relationship between the lower-level features. Their work was also based on the tree-structure of parts proposed by Zhang et al. [2007]. They used two separate models, the first was a 5 part model to represent 2 eyes, a forehead, nose and mouth part in frontal view. And the second was a 4 part model, modelling the nose, one eye, mouth, and side of the head parts. With these two models, they were able to model out-of-plane rotation from 0 to 45 degrees and from 45 to 90 degrees. The choice of whether to use the 4 or 5 part model was decided by the CNN. The face detector CNN used image patches of $112 \times 112 \times 3$ centred around the faces as input. They have found that reducing the stride after each layer in the CNN allowed them to extract the most meaningful features in later layers. One of the other aspects of training a CNN that became very apparent in their work was that the use of a drop-out layer significantly improved training speed and the quality of the features that were being extracted. Their work has also shown promising results using their new intuitive non-maximum suppression to eliminate false positive detections of the neural net and determine good bounding box locations. In most other work a simple "intersection over union" non-max suppression algorithm was used. A hard limit was used to suppress all bounding boxes that do not meet a certain threshold using this

criterion. There were variations on this theme, but Zhang et al. [2007] highlighted the fact that this method does not discriminate between high-scoring and low-scoring boxes. They proposed using a MeanShift clustering algorithm to cluster bounding boxes and looking for a local minima in the region to determine the new centre point. To deal with scale variance during detection, the input image was scaled to D different sizes where the original size is $2/D$. Their system was trained on the Face Detection Data Set and Benchmark (FDDB) by Jain and Learned-Miller [2010] which consists of 2845 images. The system achieved state-of-the-art detection, and significant improvements where noticed when introducing Intuitive Non-Max suppression. The best results achieved by this system was 87,06%. It should also be noted that this method was remarkably robust to occlusion, however this was not the primary goal of this work.

A method developed by Dong and Wu [2015] employed the use of a CNN to look at the face region and output a list of coordinates that reflect the positions for the left eye, right eye, nose, and mouth corner parts. Dong and Wu [2015] proposed the use of a simplified hierarchical model for face alignment that showed remarkable robustness to occlusion. The proposed method was named Adaptive Cascade Deep Convolutional Neural Networks (ACDCNN) and achieved robust facial landmark localisation in two distinct steps. The first CNN consisted of 6 layers to detect the whole face and to initialize the facial region. The second step used the facial region as a starting point for 5 Local Adaptive Cascade Networks (LACN) to iteratively refine the localization of 5 feature points centred around the left eye, right eye, nose tip, left mouth-corner, and right mouth-corner. Every part was detected by a unique detector because each part has a different appearance. What made this method quite interesting is the fact that part-localization happens iteratively, using the output of the previous network as inputs for the subsequent networks. Each network produced an error denoting the distance and direction from the true localization and following networks used this output to adjust the detection area. This process was repeated until the error is no longer reduced. The system was trained on 10000 images from the LFW dataset, as well as data that were downloaded from the internet. The dataset did contain some occlusion however this was not the main focus of the dataset. In order to train the local part patches, many samples were taken around a centre point in a Gaussian distribution. The results achieved by their proposed system was comparable to the state of the art, with the exception of experiments where it was tested on purely gray-scale images as this system was trained on colour images.

Face detection and alignment are often seen as separate goals, and CNNs have proven an effective tool to solve both of these problems, however Zhang et al. [2016] proposed a framework that exploits the correlation between these two tasks in order to boost performance on both of these tasks. This work also indicated that there are significant computational expenses associated with various DPM techniques. One of the challenges this work addressed was the high computation cost associated with some of the previously proposed

CNN architectures that do not exploit the correlation between the face localization and alignment tasks. The proposed method utilized a shallow CNN for region proposal, a deeper CNN to reject a large number of false positives, and finally used a deep CNN to refine the detection regions and output facial landmark positions. The idea was that the reduction of computational cost will allow real-time face alignment. The first CNN was called the Proposal-Network (P-Net) and used a non-max suppression algorithm to merge highly overlapping regions. Then the regions were fed through a CNN called a Refine-Net (R-Net) to remove a large number of false positives as well as refine the bounding boxes through regression. The final network determined the facial features of the detected faces using a similar approach to that of the R-Net. It is important to note that the same convolutional layers were used for each CNN with only the final layers differing depending on whether the network was trying to classify, do regression, or perform landmark localization. The system was trained and tested using the FDDB, WIDER FACE and the AFLW datasets. This system consistently outperformed other methods such as Robust Cascaded Pose Regression (RCPR) and Faceness.

3.2 Region localization under occlusion

Dealing with occlusion in the facial region is a challenging area, because of the fact that some of the vital information for localization is not present. There are various ways of dealing with this problem, from ignoring the region and inferring the localization from the positions of other parts, to simply attempting to identify and reconstruct the missing data. Although some of the methods explored thus far have shown some robustness to occlusion, none of these methods have had the specific goal of detecting and dealing with occlusion. This differs from the methods in this section that propose strategies to specifically handle occlusion.

3.2.1 Region localization using hand-crafted features

3.2.1.1 Robust cascaded pose regression (RCPR)

Burgos-Artizzu et al. [2013] used HDPMs with a variation on Cascaded Pose Regression (CPR) to iteratively refine facial-feature points of faces with significant occlusions present. This was achieved by modelling explicit occlusion patterns into the face parts and using a statistical model to determine the presence of certain occlusions. In CPR, a series of regressors were trained, each with the goal of minimizing the difference between the estimation of the previous regressor and the ground-truth values. Each regressor attempted to minimize the localization error starting from some initial estimate, however, occlusion often caused errors in the estimation, and because a cascade of regressors was used, the error was often propagated, causing large errors in localization. The idea was that a cascade of regressors could be used to refine feature point localizations as in traditional CPR, however, instead

of training regressors on vectors of only x and y coordinates, a third value could be added, describing the visibility of the point. The pose regression algorithm was then allowed to use all three variables during searching, thus allowing occlusions to be identified and ignored by the regressors while localization was happening. They achieved remarkable results with an average error rate of 8,5% of the inter-ocular distance on the very challenging Caltech Occluded Faces in the Wild (COFW) dataset.

3.2.1.2 Consensus of exemplars

Belhumeur et al. [2013a] proposed a method that uses consensus of exemplars to localize face-parts. The approach used the output of a set of local feature descriptors combined with a set of non-parametric global models to calculate face parts. The use of global models for matching added some robustness to occlusion. The local descriptors that were used were scale-invariant feature transform (SIFT) descriptors with a simple SVM to produce a local response. A set of global face geometry models was learned from training data and was used in combination with the outputs from the local detectors in order to determine the face geometry model that maximizes the score of the given localization. The experiments were run on the BioID and the LFPW datasets. For the LFPD dataset, the images were split into 1100 training images and 300 testing images with a further 1600 held out for subsequent evaluations. The results were calculated as the localization error normalized over the inter-ocular distance and achieved a mean error rate of 3% of the inter-ocular distance.

3.2.1.3 Occlusion coherence

Ghiasi and Fowlkes [2014] proposed a variation on the DPM algorithm that was able to deal with occlusion when localizing facial images. Instead of viewing occlusion as unstructured noise, Ghiasi and Fowlkes [2014] suggested that there are aspects to occlusion that can be statistically defined and by explicitly modelling these occlusion patterns, it is possible to build a system that is robust to occlusion. The authors argued that occlusion is caused by an object and this means that there is a definite structure to occlusions that could be identified. The authors used the example of a scenario where every second feature point along the face is occluded. This is a highly unlikely case and shows very little coherence. They term these predictable occlusion patterns “occlusion coherence”. The other key insight in this paper was the fact that the occlusion should not only be identified by the lack of evidence for feature points, but rather by positive evidence of an occluding object that explains the lack of feature points. The method made use of a combination of key point regression and model alignment by grouping feature points into distinct shapes. They introduced a “root” template that did not have a feature representation but rather encoded information about the shape and occlusion patterns of the grouping of feature points associated with that particular root template. All these root templates of parts were

connected via a tree structure a central root node. These root templates also encoded their position with respect to their parent root node. Each root node had a set of leaf nodes that was connected to the root node in a star topology, with keypoint appearance represented by a small HOG template as can be seen in Figure 3.1. During training, the shape and occlusion patterns were observed and clustered using the k-means algorithm in order to come up with a few descriptive shape and occlusion patterns based on viewpoint. During searching these shape and occlusion patterns could be mixed in order to best describe the observed image. This system was tested on the COFW dataset and outperformed the RCPR algorithm with an average localization error of 0,0746 with a 13,24% error rate at a threshold of 0,1 where the RCPR algorithm achieved an average localization error of 0,085 and a 36,36% error rate at a threshold of 0,1.

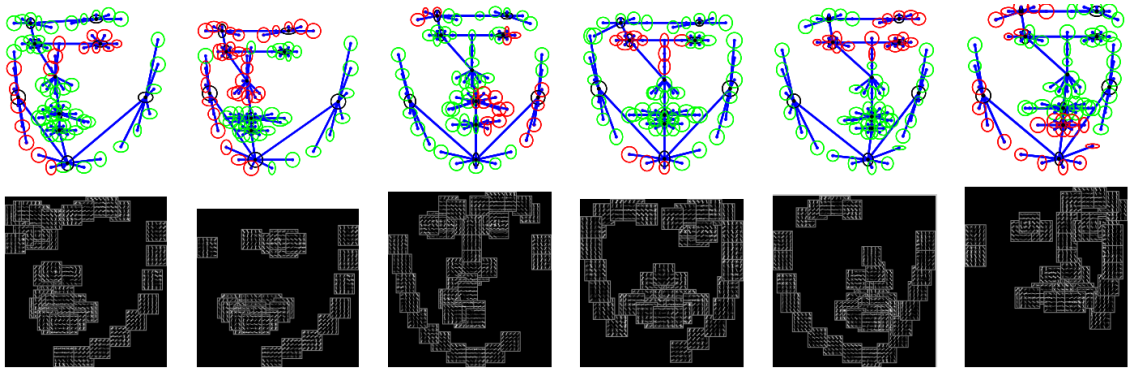


Figure 3.1: The first row depicts different configurations of the model proposed by Ghiasi and Fowlkes [2014] that consists of a tree of parts. The root node of each part is denoted by a black circle and has no HOG template associated with it, but is connected to a set of leaf nodes denoted by red or green circles. The green leaf nodes denote the visible feature points and the red leaf nodes show feature points where the feature point is occluded. The second row depicts the HOG appearance model of each of the visible key points.

3.2.2 Region localization using learned features

3.2.2.1 Active appearance models

In order to deal with feature point localization in occluded faces, Gross et al. [2006] proposed a variation on the AAM algorithm that could effectively deal with face occlusion both during training as well as when novel images are localized. This was done by modifying the traditional project-out fitting algorithm to be more robust to occlusions during fitting. This new algorithm was called the normalization algorithm and had performance equivalent to that of the project-out algorithm when tested on non-occluded faces, but far outperformed the traditional method as the level of occlusion increased.

Storer et al. [2009] proposed a variation on the AAM method, using PCA to localize facial feature points under occlusion. AAMs are widely used in the field of computer vision but are not especially well equipped to deal with occlusion. To overcome this problem, it is proposed that a PCA-based algorithm could be used to reconstruct the occluded facial regions. Since PCA reconstruction is computationally expensive, they proposed a variation, called Fast Robust PCA (FR-PCA) that achieved results much quicker than the standard PCA algorithm but also achieved superior results. The final algorithm achieved more reliable results when occlusion is present than that of normal PCA with mean errors of less than half of that of the PCA algorithm. All of this was achieved by the ability of the algorithm to reconstruct or "interpolate" parts of the image.

3.2.2.2 Regressing local binary features

Ren et al. [2014] proposed a very fast regression algorithm for face detection using a set of learned local binary features that perform joint regression in order to add shape context. They argued that the use of general purpose hand-crafted features are not ideal, as these features are not task-specific. The method relied on the training of very descriptive local binary features per part and used this to jointly learn a linear regression algorithm for the final output. During training both the regression matrix and the feature mapping function was learned. The feature mapping function was a set of local mapping functions each learned using a standard random forest algorithm. This work was tested on the LFPW dataset where 717 images were used for training and 249 images for testing. The error was normalized over inter-ocular distance and achieved an average error of 3,35%, outperforming RCPR on this dataset.

3.2.2.3 Convolutional neural networks

Garcia and Delakis [2004] proposed the use of a simple CNN architecture for fast and robust face detection in images. This work was also some of the first in this field proposing using only learned features and not making use of any hand-crafted features or image pre-processing during classification. This work made specific mention of robustness to variability in image rotation of up to 20 degrees in in-plane image rotation and up to 60 degrees in pose rotation. The proposed CNN architecture in this work was based on the work done by LeCun et al. [1998]. The network was trained on 3702 highly variable face images of 32×36 pixels gathered from various sources. The faces were normalized to ensure that the retinas were in approximately the same position for every cropped image. This system produced quite stable and robust detections with a very small false positive rate in comparison to other techniques at that time. The system also showed robustness to various forms of occlusion.

Yang et al. [2015] proposed a new deep CNN architecture that used face part responses to

make accurate detections of faces in an image even with severe occlusion and pose variation. This method employed a carefully formulated scoring method that performed well in challenging cases where the face is only partly visible. This method worked by using a series of part detectors, corresponding to left-eye, right-eye, nose, mouth, beard and hair parts, on the entire image as input. Each part detector was also trained to recognize attributes such as colour, style, race and other attributes allowing for more detailed descriptions of detections than simply the positions. They used an effective method to reason about the likeliness of a face being present based on the constraints between the part-response regions. After this step, a face-classification and bounding box regression neural network was trained and used to refine the final face proposals. This resulted in simultaneous detection of the face and face parts. The resulting face proposals were evaluated with the Intersection over Union (IoU) metric to determine the final face detections. The system was trained and tested on the FDDB, PASCAL and AFW datasets, and performance was measured using the precision-recall protocol. The results achieved by Yang et al. [2015] outperformed comparable systems with a recall rate of $90,99\%$ even under conditions of severe occlusion and pose variation.

More recently Triantafyllidou et al. [2017] proposed a very simple CNN architecture using a technique called "hard sample mining" for face detection. They introduced a novel training methodology called hard sample mining where the network is trained with hard positive and negative samples in the early phases of training and more difficult samples are introduced later in the training process. This allowed the system to handle partially occluded faces and other difficult positive samples much better. The fact that this simplified model only had a fraction of the tunable parameters, in comparison to previously proposed methods, meant that the amount of data used in training the system was significantly reduced and the performance during testing was significantly increased. A third interesting idea presented in this work was the use of pooling layers later in the network in order to smoothen the heat-map produced when the network is convolved over the image. They also introduced a regression neuron in the output layer of the detector estimating the width of the detected face. One interesting aspect of the final CNN architecture was that face detection and part detection were done simultaneously in a single pass, and these separate outputs were combined in a combination layer where further convolutions were done and finally output the probability of a face being present in that subsection of the image. The fact that the visibility of each individual part also plays a role in the final output of the network, meant that the system was less affected by occlusion of parts of the face. The system was tested on the challenging Face Detection Data Set and Benchmark (FFDB) and was compared to the performance of other systems on this dataset. The resulting system performance was comparable to the state of the art and it showed remarkable robustness to occlusion because of the use of a combination of separate parts during detection.

Jiang and Learned-Miller [2017] proposed a variation on the Region-based CNN originally

proposed by Girshick et al. [2014]. R-CNNs are widely used for object detection tasks and works in two phases. In the first a selective search is used to generate category independent region proposals. The second step consists of transforming the region proposals into regions of fixed sizes and using AlexNet to generate a feature vector of 4096 dimensions to be used in both a regressor, to refine the detection position, as well as a classifier to detect the class of the object. This method is still computationally heavy because of the number of times each region proposal has to pass through the network. The work by Jiang and Learned-Miller [2017] however, improved the performance of R-CNN by using a single region proposal network to search over various scale and aspect ratios simultaneously, as well as sharing convolutional layers between the region proposal network and the detector network. The system was trained on the WIDER face dataset, that contains a total of 12880 images containing 159424 images. One of the very challenging aspects of this dataset is the fact that there are large scale and pose variations in this dataset. Although this system was designed for general object detection, it did achieve remarkable results for face detection in images with large scale and pose variations.

Recently Ranjan et al. [2017] proposed a very interesting approach to simultaneous face detection, landmark localization, pose estimation and gender recognition under occlusion using only a deep CNN architecture. The architecture managed to extract all this information in one pass by using the intermediate layers of the network to assist in determining the more low-level information such as landmark-localization and pose estimation, whereas the later layers encoded much more high-level and class specific information useful for tasks such as face detection and gender classification. To get all of the low-level and high-level features extracted in order to simultaneously determine face-presence, feature-point positions and visibility, pose estimation and gender classification, the hyper features extracted from all levels of the network were combined in a fusion layer and passed through a separate part of the network. These two levels of the network could be trained in an end-to-end fashion. Evidence from this research shows that training of all of these tasks simultaneously produced better results than training any of these tasks individually. It is also interesting to note that different loss functions were used to determine the results of each of the tasks. The system was trained on 20997 images from the Annotated Facial Landmarks in the Wild (AFLW) dataset. This system managed to outperform current systems on all tasks. However, due to the complexity of this network, a large amount of training data was needed.

3.3 Conclusion

The use of CNNs as a way to learn features for localization does achieve state of the art results and has very promising potential in the use of face-part localization. From the literature, it is also evident that the use global shape models simplifies the handling of

occlusion, as seen in the work by Belhumeur et al. [2013a]. The model proposed in this work takes a similar approach to Dong and Wu [2015] but the localization of face parts is done using a single CNN discriminating between 4 parts similar to the approaches taken by Zhang et al. [2016], Triantafyllidou et al. [2017] and Ranjan et al. [2017], and uses geometric information similar to that used in DPMs to improve part detections without using as many training samples. DPMs also allow for more flexible and more general models. The combination of neural networks and geometric information allows the system to detect and estimate the location of occluded parts fairly easily without the need for costly interpolation and CNNs allow for easy classification of face, and non-face parts. The idea of cascading a series of CNNs such as the work done by Yang et al. [2015], Zhang et al. [2016], Triantafyllidou et al. [2017] has also proven very effective and is utilized in this work.

Chapter 4

Methodology

In this chapter, the workings of the proposed face-part localization system is given. The goal of the system is to look at images containing a face, and calculate the location of the left eye, right eye, nose and mouth parts. This process can be seen as broadly consisting of 2 steps. The first step is face localization, and two separate methods are proposed to achieve this goal. The goal of this first step is to locate and normalize the face-region in terms of size and rotation in preparation for the next step, which is face-part localization. The second step takes the localized face region and detects the 4 face-parts within that region. The result is the image with the 4 face-parts localized. A flowchart, outlining the localization steps can be seen in figure 4.1.

This chapter is presented according to these two steps. Section 4.1 describes two alternative face localization algorithms. The first method of localizing faces uses a single face CNN to propose regions of potential faces as described in Section 4.1.1. The second face detection method uses the responses of a face-part CNN at positions corresponding to the typical positions of face parts to propose possible regions for face localizations as described in Section 4.1.2. Both of these methods generate the region proposals following the same format, this allows the following steps to be agnostic of which method is used. A simple non-max suppression algorithm is used to make a make the final face localization based on the proposed regions. This simple non-max suppression algorithm is described in Section 4.1.1.1. The final part of the face localization section describes an optional iterative alignment step that aims to improve the initial face localization. The iterative alignment setp is described in Section 4.1.3.

This section is followed by a description of how face-part localization is performed on the pre-cropped and normalized face-region calculated in Section 4.1. After localization of the facial region, the face parts are localized using a new face-part CNN that is convolved over the region and uses the learned face geometry to enforce constraints between face parts

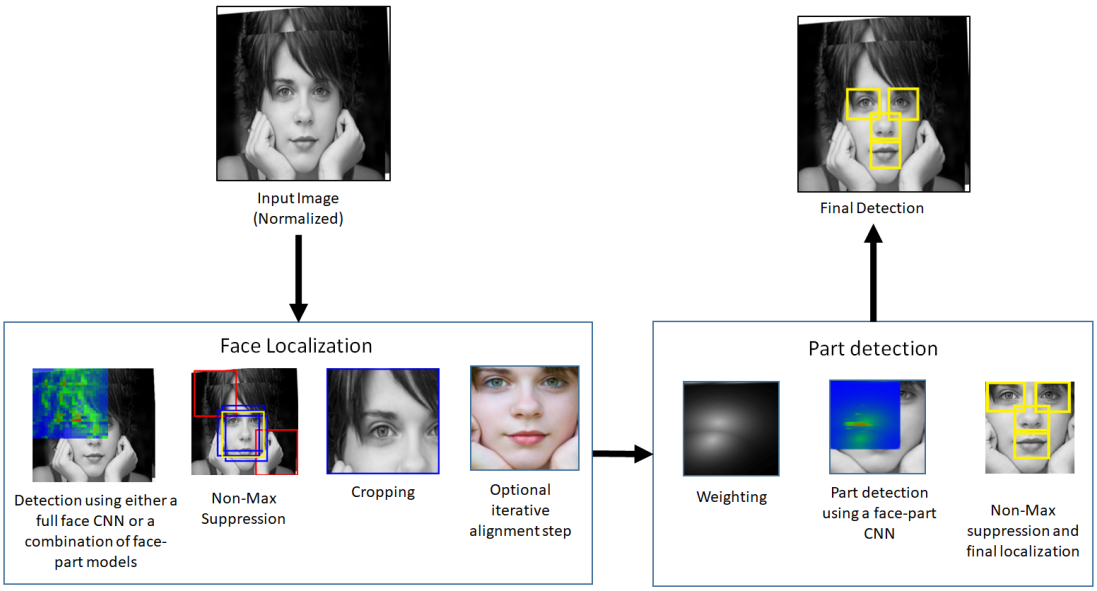


Figure 4.1: Architecture of the proposed face, and face part localization system.

to ensure plausible facial structures after detection. The face-part geometry is enforced as the cost of placing the face-parts at certain points in the face region. These points are calculated as the expected points relative to the center of the face. This means that the it is paramount that the face localization be very accurate as an incorrect face localization would enforce incorrect constraints on the parts. The iterative alignment step mentioned in the face localization process aims to increase the accuracy of the face-part placements by getting a more accurate face localization and aligning visible face-parts to expected placements of face-parts. The face-part detection process is described in Section 4.2.

4.1 Face localization

In this section, two options for dealing with face localization are proposed with the fundamental difference between the two being, the use of a face CNN that generates face localization proposals, versus a technique that uses a face-part CNN that generates face localization proposals based on the response of a combination of individual face parts. The first method works by convolving a face CNN over the image. The second method generates proposals by convolving a set of face-part CNNs over the image using images cropped from predefined locations within the region tested for being a face. The second method also has the advantage of being able to generate proposals without including the responses of occluded parts. The map of proposals generated by the two techniques take

the same form and the same process was followed after the proposals were generated. The two different methods are represented in Figure 4.2.



Figure 4.2: a) Face localization using a face CNN model representing the face in its entirety that is convolved over the entire image. b) A face localization algorithm that calculates the response at a point as the result of the responses of the parts detected at certain locations.

4.1.1 Face localization using a face CNN

This technique generates face localization proposals by using a face CNN, capable of distinguishing faces and non-faces, convolved over the image as demonstrated by Figure 4.2 (a). This method assumes that a CNN has the ability to learn some robustness to occlusion and out-of-plane rotation when classifying groups of pixels that potentially belong to a face. The face CNN was convolved over the input image in small increments of pixels called the "stride", meaning that the region of the image being considered by the face CNN was moved incrementally and fed through the face CNN. The output of the face CNN was recorded for that region and is called the "response". Localization proposals were generated by considering regions where the "face" output neuron of the face CNN responded better than the "non-face" output neuron for the given input. An example of region proposals for a face image can be seen in Figure 4.3. Each pixel in Figure 4.3 represents a potential face center. All these region proposals were used in calculating the final face localization as described in Section 4.1.1.1. In order to create a face CNN capable of effectively discriminating between faces and non-faces a series of experiments with different CNN architectures were conducted.

4.1.1.1 Non-maximum suppression

As the face CNN is convolved over the image it generates a series of face region proposals as demonstrated in Figure 4.3. In order to effectively ignore false positives and to consolidate multiple true positives into an accurate bounding box a simple non-maximum suppression algorithm was applied. This section outlines a novel and simple non-max suppression



Figure 4.3: A map of face centre proposals from the face CNN where white indicates the highest certainty of a face and black the lowest on an image from the HELEN dataset [Le et al., 2012]. The map shows the face centre as a potential region for a face.

algorithm that is useful in the specific case where only a single detection per image is expected. Because there was exactly one face region per image, the assumption was that the proposals should cluster around the true face region and the outliers in terms of the x and y positions of the proposals could be calculated and ignored as false positives. The outliers were calculated by ordering all the detections by x position and the mean value was calculated. The set of detections was then split into two groups based on whether the x values are higher or lower than the mean value. The mean values of both of these groups were then calculated. These values are known as the first quartile, or Q_1 for the lowest mean value, and the third quartile, or Q_3 for the highest mean value. The inter-quartile range (IQR) was then calculated by calculating the distance between Q_3 and Q_1. The minor fence was calculated by multiplying the IQR by $1,5$. The upper boundary was calculated by adding the minor fence value to Q_3 and the lower boundary was calculated by subtracting the minor fence value from Q_1. All the detections with x values greater than the upper boundary, or smaller than the lower boundary were ignored. This process was repeated for the y values of the remaining detections. The remaining positive detections were expected to cluster around the true facial region as all the outliers were removed. The remaining detections were used to calculate the true detection position. This was calculated as the average of the x and y coordinates of all detections. This region can then be cropped and face-part localization can be done on the face region. This process is demonstrated in Figure 4.4.

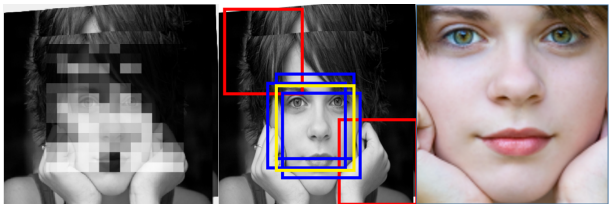


Figure 4.4: a) Face region proposals with white as the highest response and black as the lowest. b) The non-max suppression algorithm applied on the face region proposals. Outlier proposals are shown as red boxes, blue boxes correspond to proposals taken into account, and yellow is the final detection based on the average of all the proposals that aren't outliers. c) The final cropped region.

4.1.1.2 Stride

The responses of a complete convolution of the face CNN can be seen in Figure 4.3. To calculate these responses of a convolution can often take very long especially when the images that are being searched are quite large. Searching over the entire image on a pixel by pixel bases is also not always necessary. The sliding window does not have to stop at every point in the image and steps of a couple of pixels can be taken. This is called the "stride" and has a significant impact on the speed and accuracy of the detection. Smaller strides will allow more fine-grained detection with more positive responses close to the true face location, but will decrease detection speed as it is more resource intensive to feed more sections of the image through the classifier. Larger strides will speed up the searching process but have a higher chance of missing regions of interest where true positive detections would have been made. It also reduces the effectiveness of the simple non-max suppression algorithm employed during detection. An example of how variations in stride can cause regions of interest to be missed during localization can be seen in Figure 4.5. In some preliminary experiments, the use of a stride of 16 pixels yielded a good balance between positive detections and speed.

4.1.1.3 Dealing with different input face sizes

In the wild, the size of the face on the image will not necessary be the same size as the input required for the detector. In order to achieve multi-scale detection, a pyramid of input features can be used.The input image is scaled to different sizes and the face CNN of a constant size is convolved over the image. This has the effect of detecting faces of different sizes without having to train multiple sizes of the model. The region proposals of all the scales are concatenated and the best regions are determined. However, in our experiments, the face images used for testing were normalized in terms of scale, before detection started in order to simplify and speed up the experiments. This made it unnecessary to apply this search method to our technique however it could be added in future work without

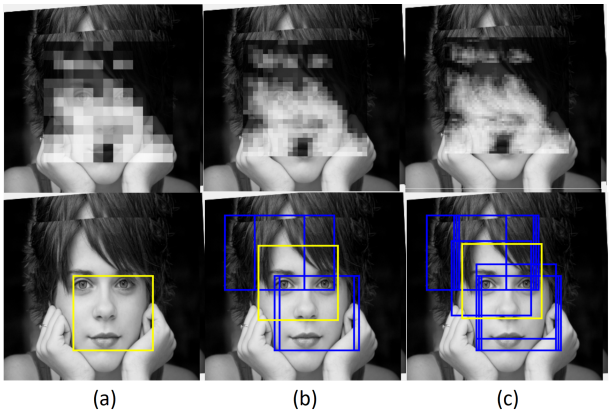


Figure 4.5: A few face region proposals generated by varying stride lengths with the corresponding non-max suppression results on an image from the HELEN dataset [Le et al., 2012]. It is important to note that the proposals are not generated by a fully trained face CNN. a) 16 pixel stride. b) 8 pixel stride. c) 4 pixel stride.

modifying the overall technique. All images were normalized to ensure that the entire face is centred and contained within a square 128×128 pixel region.

4.1.1.4 Dealing with rotation

Just like dealing with different input face sizes, dealing with rotation of faces in the input image is also a requirement for applying face detection techniques in the wild. The proposed model has no built-in mechanism of dealing with in-plane rotation, however, robustness to slight out-of-plane rotation was built into the model as it was trained on images of faces with minor pose variation. The easiest solution to the problem of in-plane rotation is to do an explicit search over the possible in-plane rotations [Ghiasi and Fowlkes, 2014]. This is similar to the way that variations in scale are handled by the pyramid of features approach. Another approach to dealing with the problem of in-plane and out-of-plane rotation is to train the face detector to deal with the specific rotation and perhaps output the degree and direction in which the face region is rotated. In the conducted experiments rotated faces were normalized by using the provided part localization labels to calculate and correct in-plane rotation before detection started as described in Section 5.2.1. This was done in order to simplify and speed up the experiment.

4.1.2 Face localization using a face-part CNN

A second method for face localization attempted to deal with occlusion in faces explicitly by only looking for face parts at median face part locations. The responses from the face-part

CNN at each of the expected face-part locations could then be combined to the likelihood of a face being present. In other words, instead of looking for a full face, this method looked for eyes, nose and mouth parts, in their expected positions and if these were detected in a pattern that suggests the presence of a face, a high response was produced. The median placements of each face part is described by a face geometry model that describes likely placements of each part. This model lends itself well to the problem of face localization under occlusion as this allows for certain parts of the face to be ignored as it might be occluded. For face localization using a face-part CNN, a model with the median distances between the feature points was convolved over the entire input image. Another way of stating the above is that for each region of the input image that the model was evaluating for the presence of a face, the face-part CNN was applied to sub-sections of that region at locations that correspond to the median placements for eyes, nose and mouth parts. To determine whether the region is a valid proposal the responses of the 4 parts were considered. If the face-part CNN classifies the parts at the locations of the eyes nose and mouth correctly, the region is considered a valid proposal. In order to build in robustness to occlusion, an allowance is made for 2 of the 4 parts to be classified incorrectly as the assumption is that for a face to be valid, at least half of the face should be visible. This also prevents a single part being classified correctly from generating region proposals that are not valid. This still allows detection of partial faces but eliminates many false positives. This model is shown in Figure 4.2 (b). This process is outlined in Figure 4.6. After a set of face region proposals were produced, non-max suppression was done on the resulting proposals for each part and the final face region was cropped.

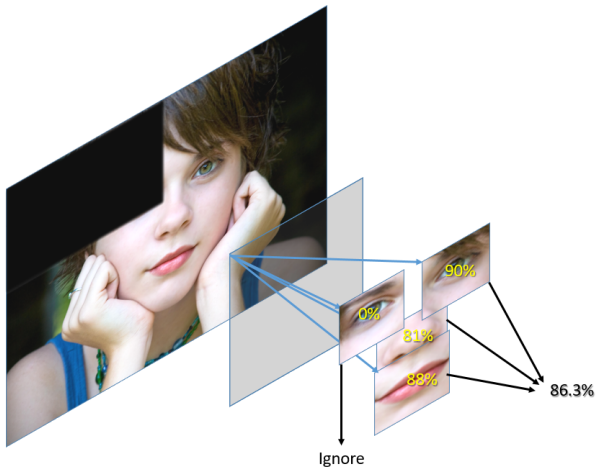


Figure 4.6: How the part-based face detector identifies and ignores occlusion

4.1.3 Iterative alignment

In this section, the use of the part detector to iteratively improve the detection of the face region is discussed. In this system, part localization is highly dependent on the initial region proposed by the face detection algorithm because it is used as an initial estimation of part localizations. This means that if face localization is not done with a high degree of accuracy, the part localizations produced is highly inaccurate as the final face localization is used as a point of reference. In order to counter this compounding error caused by misalignment of the face localization, a simple technique can be used to eliminate the need for a perfect face localization on the first detection. It is however still necessary that the initially estimated localization is quite close to the true localization. This step works by aligning visible face-parts to expected placements of face-parts as described by the learned face geometry. The part detector can be convolved over the estimated face region without any constraints enforced when calculating part responses. The part detection with the highest response is taken as the true localization of the part and a new face detection frame is calculated using this part as a reference. The position of the face region can be inferred from the location of a single part, because the median placements of parts are used to estimate the location of the new cropped region. The cropping is adjusted to align the face to match the placement of the part with the best response. This can be seen in Figure 4.7. In other words, the system knows where the median placements of the eyes, nose and mouth should be, and uses this information to match the cropping such that the best responding part, ie. a nose, matches the average placements of a nose as learned from training data. This step is optional and the effect of the addition of this step is measured in the Results section.

4.2 Face-part localization

In this section, we describe the localization of face parts given a localized and cropped facial region. This only happens after face detection has been performed. In this step, the face-part CNN, described in Section 5.3.5, was combined with the knowledge of the constraints between these parts to detect the face-parts denoting eye, nose, and mouth regions. These constraints are referred to as the "spring-cost". The constraints between these parts are simply represented as offsets from the centre of the image with a rotated ellipse denoting 95% confidence ellipse of all the points provided in training data for that specific part. This allows the system to accurately model the most likely part positions as well as describe the plausible range of part positions relative to the center of the cropped image. The part localization step works in much the same way as the face localization step, in that a CNN is moved over the cropped face region. However, the part detector has access to the spring-cost that it can exploit to more accurately localize face parts and eliminate false positives. This combination gives the appearance and location information of the part the is being localized. This is done by combining the output of the face-part

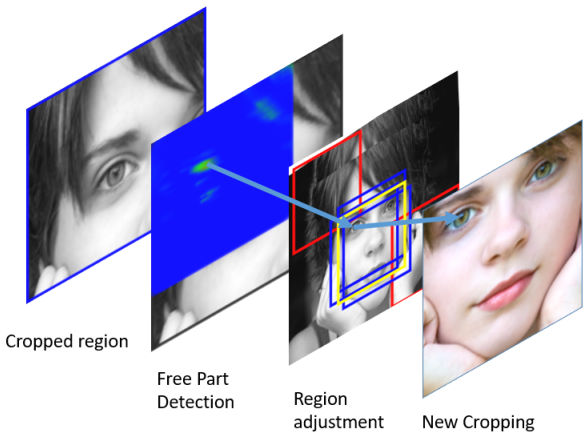


Figure 4.7: A visualization of the iterative alignment process. In this image the responses of the part CNN's eye neuron is shown over the face region where blue denotes the lowest response and red the highest. The responses are also plotted from the top left corner of the image, as these points correspond to the top left corner of the detection window for the face-part CNN.

CNN and the spring-cost defined by the geometric face model calculated during training. To achieve accurate face part localization, a face-part CNN with an input corresponding to the part size, and 4-outputs, corresponding labels associated with eyes, nose, mouth and no face-part was used. Differentiating between left and right eyes points is very difficult because of the similarity in appearance. To handle this problem, one output was used for both eyes, and the spring-cost was used to select plausible left and right eye candidates. The detector was iterated over the cropped face region in small increments and the response of the detector for each class was recorded. This results in an $M \times N \times 4$ matrix where M corresponds to the width of the face region minus the width of the input to the face-part CNN divided by the stride as the detection patch only produces responses for sections that are completely within the bounds of the whole image. The term N corresponds to the height of the face region minus the height of the face-part CNN divided by the stride, and the final dimension is 4 as the system keeps track of the score of each class, of which there are 4. This can be seen in Figure 4.8.

4.2.1 Geometric spring cost

In this section, the technique for calculating the placement, or spring cost is described. The spring cost is the cost associated with placing a face-part at a certain position in the facial region. This spring cost is based on the geometric features calculated from the dataset. These geometric features describe the face geometry in terms of 4 rotated ellipses that encircles the most plausible placements of each part respectively. These ellipses are calculated from the feature points in the training dataset as described in Section 5.4. The

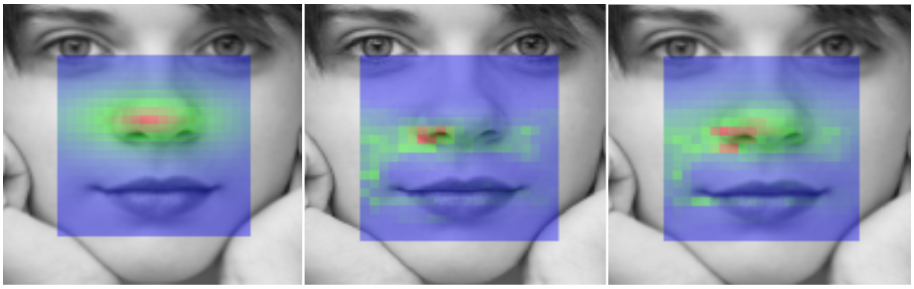


Figure 4.8: a) A map showing the expected nose center-point relative to the cropped face based on the geometric information where blue denotes the lowest response and red the highest. b) A map of responses of the nose part points based on the responses from the Face-Part CNN . c) The sum of (a) and (b), this also shows the most probable placements of the part. The original image is from the HELEN dataset [Le et al., 2012]

goal of the spring-cost is to make it more feasible for parts to be placed in more plausible regions of the face during localization. The spring cost at each point of the convolution is calculated by a rotated 2-dimensional elliptical sigmoid function, offset by the total variance in the direction of the current point from the centre point of the ellipse. This results in a value between 0 and 1 where 1 shows that the point is a very plausible position for the part to be placed and 0 shows that the point is at a highly unlikely position for the part to be placed. The first step in calculating the spring-cost is to determine the total variance of the ellipse in the direction of the detection. This value corresponds to the radius of the ellipse in the direction of detection and can be expressed by the following equation.

$$angle = \arctan \frac{G_y - P_y}{G_x - P_x} - G_a - 90 \tag{4.1}$$

$$V_t = \sqrt{\frac{1}{(\sin(angle)/V_x)^2} + \frac{1}{(\cos(angle)/V_y)^2}} \tag{4.2}$$

where G_x and G_y corresponds to the learned geometry point's x and y coordinates and G_a is the geometry point's angle. V_x and V_y refer to the major and minor axis of the geometry ellipse and P_x and P_y are the x and y coordinates of the position of the current detection. The spring cost can then be calculated as:

$$SpringCost = \frac{1}{1 + e^{x - V_o - V_t}} \tag{4.3}$$

where x is the Euclidean distance between the centre of the ellipse and the point being tested. V_o is an offset that is used in order to adjust how much distance outside of the radius can still be tolerated before responses start to degrade. This gives us control over how "stiff" or "stretchy" the spring cost is allowed to be. In our experiments this value is

set to 6 as this has proved to be a good balance between flexibility and accuracy.

The result can be seen in Figure 4.9.

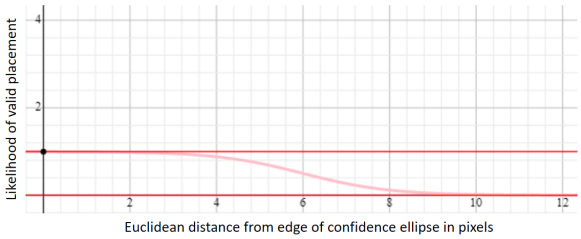


Figure 4.9: Face geometry spring-costs function

This value can be calculated for any part at any point in the image given the distance from the expected point of placement, and the variance and rotation of the training data. The resulting values can be seen in Figure 4.10.

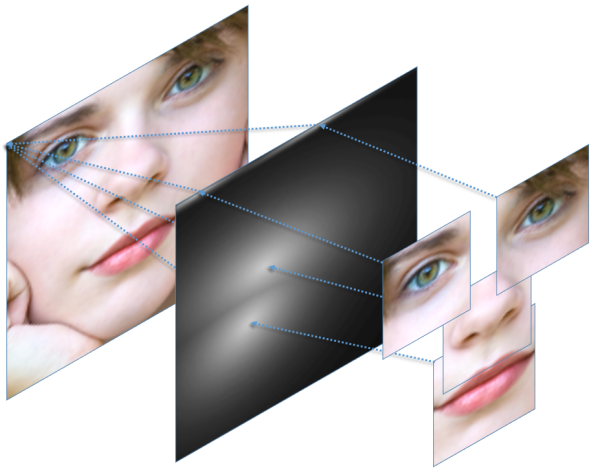


Figure 4.10: Face geometry model where white denotes the most likely placements for each part and black the least likely position for each part. The spring-cost map used in this image is a combined representation of the spring-costs of each part.

After the spring cost for a point was calculated, the resulting value was added to the response of the detection in order to calculate the final response for each part at that point. When dealing with face parts, each point has 4 values, each corresponding to a face part, and also a neuron corresponding to non-parts. Therefore the spring cost for each part was calculated, as described above, to each of the face parts and the resulting values were

added to the responses. Because the face-part CNN only has one neuron corresponding to both eyes, the value for the eye was duplicated and the spring-cost to the left eye was added to one of the responses and the spring-cost to the right eye is added to the other response.

4.2.1.1 Spring cost of non-parts

A special spring-cost for the non-part neuron was also calculated. This is because in this system the positive identifications of non-part sections are also considered when parts are competing to be proposed at a certain point in the image. The spring cost associated with the non-part response is the inverse of the spring-cost associated with face-parts. The "distance" to the ideal location of a non-part was calculated as the distance to the closest part-point subtracted from the variance in the direction of the point being considered for a non-part. This means that the closer the point gets to a part position, the further the non-part is away from ideal non-part placements. This distance is calculated mathematically as

$$Max((V_{(tmin)} + V_o - Dist_{(min)}), 0)$$

Where $V_{(tmin)}$ is the total variance to the nearest expected part, and $Dist_{(min)}$ is the distance to the closest endpoint. The total distance should always be greater than 0. This distance was then used on the sigmoid function as described above to calculate the total spring-cost of the non-part response. The value for the non-part response is added to the spring-cost of the non-part placement. The resulting responses for all the parts can then be considered for region proposals. The class with the highest response is then proposed as a localization.

4.2.2 Dealing with occlusion when detecting parts

The way that optimal part placements are calculated in the system implicitly deals with occluded regions of the face. This is because of the emphasis on the geometric likelihood of part placements using the spring-cost. Both geometric information and the visual information provided by the CNN contribute to the final detection score. This meant that even when parts are occluded and the visual component of the detection fails, the likelihood of a detection at the expected point will still be higher than the surrounding areas. This process can be seen in Figure 4.11.

4.2.3 Non-max suppression and final localization

After the values of the face-part CNN was combined with the spring-costs, a series of region proposals were generated. These region proposals were filtered using the same simple non-max suppression algorithm described in Section 4.1.1.1. The algorithm was applied to the proposals of every class individually, with the exception of the non-part class. This is

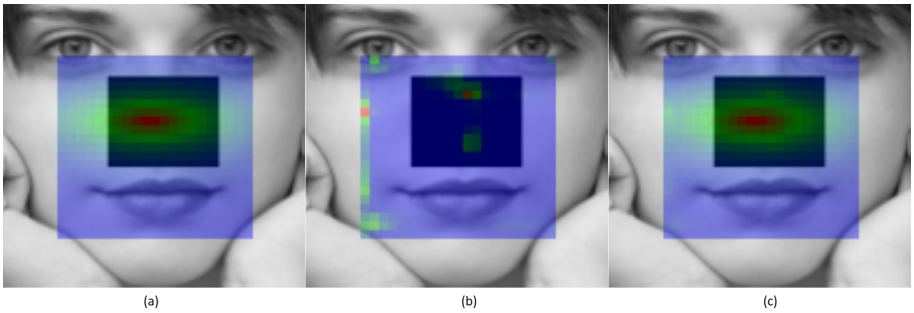


Figure 4.11: a) A map of responses based on the placement of the nose relative to the cropped face region where the nose part is occluded. Red parts are the highest responding, and blue are the lowest responding parts. This image demonstrates that the highest likelihood of placement is within the occluded part of the face based on the spring cost. b) A map of responses based completely on the nose-part responses of the face-part CNN. These responses are largely ignored because of the occlusion present. c) The sum of (a) and (b), this also shows how the response of the nose-part is adjusted by most probable placements of the part, which generates good responses in the occluded part of the image which is correct.

because the idea is not to localize non-parts but use the responses of non-parts to eliminate some of the part proposals before non-max suppression starts. The locations of the final localizations were then used to measure the accuracy of the system.

4.2.4 Dealing with rotation

In the case of face-part localization, in-plane rotation was not an issue because the assumption is that by this point in the localization process the face would already be normalized in terms of in plane rotation. The system does handle minor out-of-plane rotation as the face-part CNN is trained on images cropped from images with some degrees of pose variation and the spring-cost ellipses are trained on the placements of face parts with a certain degree of pose variation. This means that the resulting system is expected to be robust against some degree of pose variation and out-of-plane rotation.

4.3 Summary

In this chapter, the process of face-part localization from raw images to fully localized face-parts is described. The two proposed face localization methods are described. A description of the simple non-max suppression algorithm used to evaluate localization proposals and determine final localization is given. A description of the iterative alignment step is given. Then the face-part detection algorithm using the spring cost is explained with a description of how occlusion is handled. This section also included a description of how the geometric

features are used to calculate spring-cost. The next chapter focuses on how the models are trained and the system is prepared for evaluation on an unseen dataset.

Chapter 5

Experimental set up

In the previous chapter, the details of the proposed system is discussed. This chapter focuses on the training and preparation of the system for evaluation on previously unseen data. In this chapter, a description of the face dataset used for training the CNNs and calculating the geometric features is given in Section 5.1. This is followed by a description of the methods used to prepare the data for training of the CNNs in Section 5.2. Section 5.3 describes the methodology used to train and test the CNNs and also shows some of the experimental CNN architectures evaluated for the various tasks. Finally, the method for calculating the geometric features, used during localization, is given in Section 5.4.

5.1 Training dataset

In this section, the dataset used for training and testing of the proposed system is presented. An existing dataset was used for the training and testing of the face and face-part CNNs. The dataset that was used for this task was the HELEN dataset by Le et al. [2012]. The HELEN dataset is relatively constrained with low degrees of variation in pose and occlusion. A few example images from the dataset can be seen in Figure 5.1. It was collected from searches containing the word "portrait" on Flickr. The images were filtered to ensure that images of sufficient size were included. The images were hand-annotated using Amazon Mechanical Turk and further post-processing was done to ensure high-quality results [Le et al., 2012]. The produced dataset consists of 2000 training and 300 test images, each with 149 feature points with only an average of 2% occlusion. The feature points can be seen in Figure 5.2. This dataset was exclusively used to train and validate the proposed system ensuring that no subjects used in the training of the proposed system is seen during the testing. This is done in order to mimic the the use of the system in real world scenarios where all of the subjects will be unseen.



Figure 5.1: Example images from the HELEN Dataset [Le et al., 2012].



Figure 5.2: Example image from the HELEN Dataset depicting the positions of the 149 feature points [Le et al., 2012].

5.2 Data Preprocessing

In this section, the process of taking raw data from the HELEN dataset and transforming it into suitable training, testing and validation data for the CNNs is discussed. Figure 5.3 demonstrates the process of preparing raw data and splitting the resulting dataset into training, testing and validation data. The first step was to normalize the input images and using the provided feature points. The provided feature points were then used to crop the

positive samples from the normalized input images. This served as the positive samples. Negative samples were generated from the raw input images. The images were then labelled and shuffled to create a complete dataset. Thereafter the generated dataset was split into training, testing and validation datasets that were ready to be used for training of the CNNs. Although this process was used to generate data for both face and face-part CNNs, the way that positive samples were cropped and negative samples were generated differ depending on the CNN that is to be trained. For example, the face CNN requires full normalized face regions to be cropped from the whole image whereas the face-part CNN required face-parts cropped from the face region. The data used in the preparation of training data is the 2300 images from the HELEN dataset. All images are used in order to get the largest possible collection of training data. Because this dataset was exclusively used for the training of the CNNs, all training samples generated from this dataset were used as training, testing and validation samples.

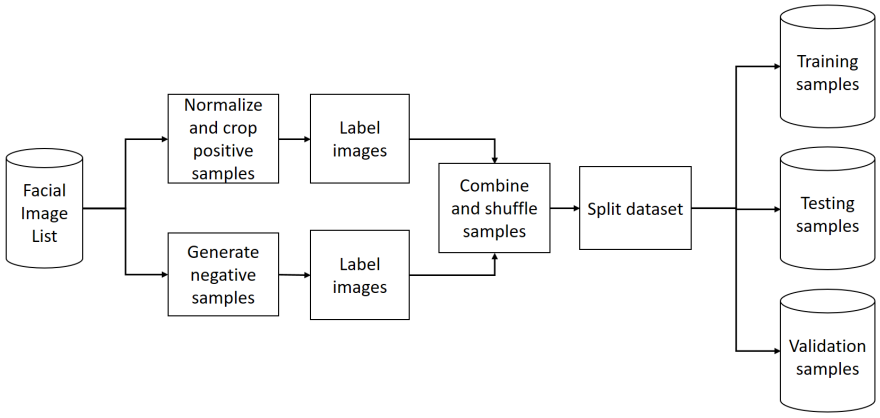


Figure 5.3: A flowchart depicting how the data is pre-processed and split into training, testing and validation data.

5.2.1 Data for face CNN

Raw images were automatically normalized by using the provided landmarks of every face to remove any in-plane rotation and to scale the images to a standard size of 128×128 pixels with the face centred in the middle of the image. In-plane rotation was removed by calculating the eye centres by averaging the two eye corners of each eye respectively. The angle between these two calculated points was determined and the entire face was rotated by the calculated angle around the centre-point between these two points in the opposite direction. This resulted in an image and a set of feature points that have both eyes on the same level, thus removing in-plane rotation. Scale was normalized by scaling the height of the face after rotation normalization, calculated by taking the y-value of the feature

point with the highest and subtracting it from the y-value of the feature point with the lowest y-value. During normalization a buffer was also added around the edges of the face as indicated by the feature points to allow the edges of the features in question to also be included in the frame after localization and cropping. This buffer was added so that after scaling the total size of the buffer is 8-pixels. This 8-pixel buffer around the edge of the image should be included in the total size of 128×128 pixels. To achieve this a scale factor value for each face was calculated by dividing the height of the face by the final scale of the image minus the 8-pixel buffer times two as the buffer is present at the top and the bottom of the image. The 8 pixels were multiplied by the scale factor to calculate a scaled up version of the buffer. The image was then cropped around the edges of the face with the scaled up buffer included. The entire cropped region was then scaled to 128×128 pixels. A raw image and a normalized version of that image, with the feature-points superimposed, can be seen in Figure 5.4. Normalization was done to minimize variation in the images and speed up training, allowing the classifier to discriminate between faces and non-faces more easily. The face regions were then cropped from the image in a square region.

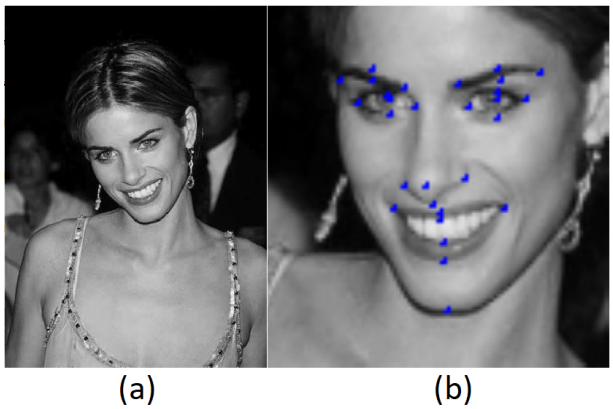


Figure 5.4: a) Original image from the COFW dataset [Burgos-Artizzu et al., 2013]. b) Normalized image with the face rotated correctly and an 8-pixel region around the edges of the face feature points.

5.2.1.1 Face data augmentation

In order to give the neural network more training data and increase the robustness of the model to slight location variance, each image was translated 15 pixels in each direction. The image was translated upwards, upwards and right, right, downwards and right, downwards, downwards and left, left, and upwards and left, resulting in 9 images. Another way of stating the above is that the image was cropped with the following coordinates as the top left corner of the image: $(x, y), (x, y+15), (x+15, y+15), (x+15, y), (x+15, y-15), (x, y-15), (x-15, y-15), (x-15, y), (x-15, y+15)$. Where x and y is the original cropping of

the image. This was done to prevent the step-size during detection from having a major impact on detection accuracy. With the augmentation of face data, the resulting dataset contains 20700 positive samples in total.

5.2.1.2 Generation of non-face samples

The second step in preparing training data was the generation of non-face samples for the face CNN. This was achieved by taking the ground truth images provided by the datasets, and randomly sampling face region sized blocks from the image. Images with more than 4 feature points visible in the generated negative image were not included in our training samples, and a new non-face sample is generated. However, this did not restrict samples that contained edges of faces which should result in a classifier that will only give positive responses where most of the face is in the frame. Figure 5.5 demonstrates some of the non-face samples generated by this technique. In total 20700 non-face samples were generated in order to match the number of face samples available for training.

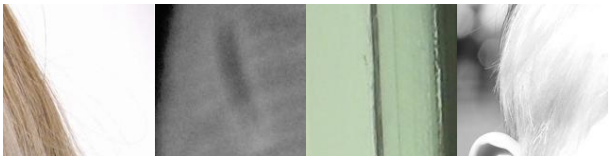


Figure 5.5: Some examples of non-face images generated from the HELEN dataset [Le et al., 2012].

5.2.2 Data for face part CNN

The normalized images extracted for the face CNN training data was used to generate the face-part samples for the training of the face-part CNN. Four image patches are extracted from the image using common feature points. The HELEN dataset that was used for training and the COFW dataset that is used for testing of the system, have only 8 data points in common and these make perfect candidates to be used to localize face-parts for the face-part classifier as the resulting part would correlate for both datasets. Four face-parts were identified that are large enough to contain discriminative features for the classifier. These parts correspond to the nose, mouth, left and right eye regions. The images extracted were centred around the average of the feature points making up those parts and are 40×40 pixels in size. Based on some initial experiments it was observed that left and right eye regions were misclassified due to their similarity. To address this problem, the architecture was changed to use only one eye class, that is differentiated during the detection algorithm. The resulting images number 9200 as the eyes, nose and mouth is cropped from all of the images. Images with occlusion of these parts were still used as the

hope is that there will still be useful information on these images and that some common occlusion patterns are modelled and understood by the face-part CNN.

5.2.2.1 Face-part data augmentation

In order to generate more training images from the HELEN dataset, and in an attempt to achieve some robustness to variation in convolution step size, the same data augmentation process described in Section 5.2.1, was used. The only difference being that the cropping was not translated by 15 pixels, but 8 pixels as the face parts were detected using smaller step sizes. This resulted in 82800 images. The resulting images were also carefully labelled to ensure that subject-independence can be guaranteed when the data is split into training, validation and testing data during training.

5.2.2.2 Generation of negative face-part samples

The approach to the generation of negative samples of face parts varies from that of face samples in that the search for face parts happen in much more constrained conditions than that of faces. This is because the search for face parts occurs in a pre-localized face region, meaning that all the negative samples will also be parts of a face. Therefore negative samples were generated by randomly selecting parts of a cropped face, and once again limiting the amount of overlap allowed with face parts. In this work, the number of feature points that were allowed to overlap was limited to 2 key feature points, as many of the parts in the face can be represented with as little as 4 key feature points. More training data was expected to be necessary as the similarity of part and non-part images was expected to be higher than that of face and non-face images. Examples of negative face parts are shown in Figure 5.6. The number of negative face-part images generated was 20700 as this represents one quarter of the generated face-part images, and this would allow negative samples to be represented equally in the resulting dataset. The resulting dataset contains a total of 103800 images.

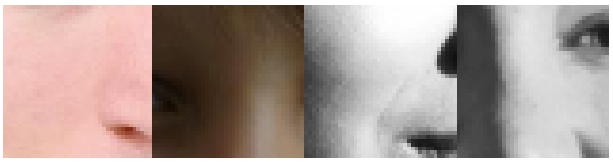


Figure 5.6: Some examples of negative face part images generated from the HELEN dataset [Le et al., 2012].

5.2.3 Training data for geometric features

The geometric features for face and face-part based localization were generated using the feature points provided in the dataset. The training data for this feature was generated

by using the same process of normalization described in Section 5.2.1, meaning that the feature points were rotated to remove any in-plane rotation, and scaled in order to fit in the centre of a square area of 128×128 pixels that includes an 8-pixel border. These features were represented as a file with a numbered list of x and y coordinates each representing a feature point in the face. There were a total of 2300 of these files.

5.3 CNN architecture and training

This section deals with the process used to train both CNNs as well as the experimental CNN architectures explored in order to find well performing CNNs for both the face and face-part localization tasks.

5.3.1 Training process

Before training starts, all the training and validation samples were loaded into memory. These samples were then shuffled to ensure that the CNN is shown samples from each class at random. After all the samples have been loaded into memory the CNN was initialized according to the architecture specified before the training process started and all the weights between neurons were initialized to random values. After initialization, an iterative process was followed to progressively adjust the weights between neurons in the CNN until it was able to reliably discriminate between the classes. This adjustment was based on the error produced by looking at a sample and determining by how much the CNN misclassified the sample. In this work, training makes use of a standard back-propagation algorithm to train the CNN. This training methodology also made use of a technique called "batch" training and allowed the algorithm to make adjustments to the weights of the CNN based on the error generated from a group of samples called a "batch". In this work, batches each consisted of 100 images, meaning that 100 images are used at a time to determine the classification error and calculate the adjustments to be made to the weights of the CNN. The training steps are depicted in Figure 5.7.

Each training iteration consisted of the same steps. Each iteration started by taking a batch of 100 training samples from the training data. The batch of images are then fed through the network and an average training error was calculated. The average error in classification was recorded and is known as the loss. The error for each sample was also recorded and used to calculate how much the weights should be adjusted in order to improve the classification accuracy of the CNN. The CNN weights were adjusted according to values calculated by the back-propagation algorithm and the CNN with all of the calculated weights being persisted to the system. This meant that a snapshot of the CNN at this point in training was taken and persisted to the hard drive of the computer it was being trained on in order to make later comparisons. The accuracy of the CNN on the

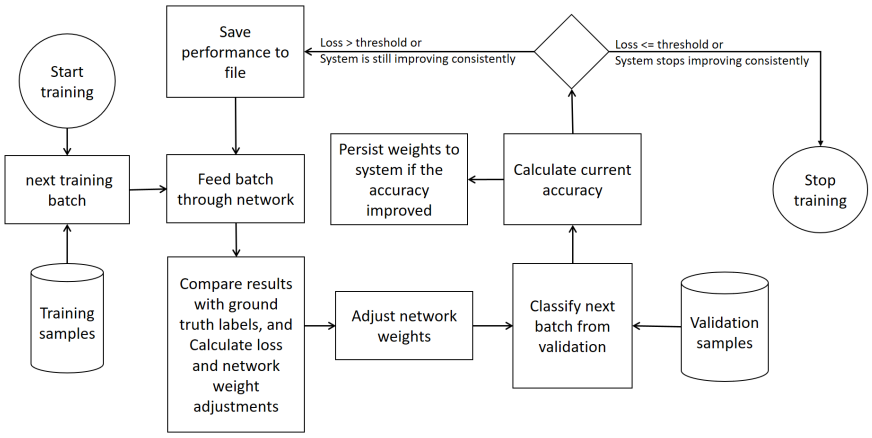


Figure 5.7: A flowchart depicting the process of training a CNN.

training data was recorded.

A new batch of 100 images was taken from the validation set and was fed through the newly adjusted CNN. The classification accuracy of the CNN on these images was recorded. The accuracy and the loss of the CNN was recorded for each iteration to allow analysis on the training of the CNN retrospectively. This process was monitored carefully to ensure that the performance was increasing consistently. This process was repeated until the CNN loss drops below $0,001$ or stops improving for many iterations. This indicated that the CNN has converged or will not be able to converge given the current architecture.

After the CNN has converged and achieved a satisfactory accuracy on the training and the validation dataset, or if the accuracy of the CNN has stopped improving, the network was tested on a dataset of unseen testing samples. The network achievement on the unseen data is recorded as the CNN accuracy. In order to get more insight into the accuracy achieved on the testing dataset, a confusion matrix was created. A confusion matrix is a matrix whereby the actual class of an image is one dimension of the matrix and the second is the class as calculated by the classification algorithm. This gave insight into which classes the algorithm often misclassified or "confuses" with other classes. This process is depicted in Figure 5.8.

5.3.2 Training, validation, and testing datasets

Training and testing was performed using separate training, validation and testing sets. This means that 70% of the data was used to train the system, 20% of the data was used

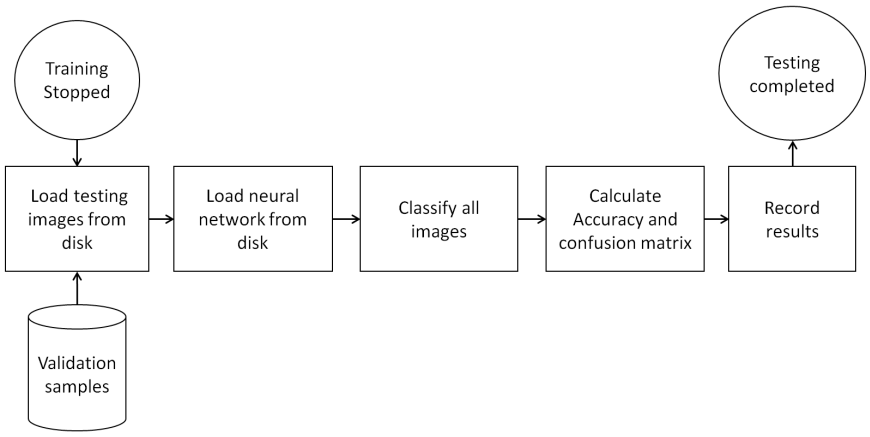


Figure 5.8: A flowchart depicting the process of testing the resulting CNN on validation data after training.

as a validation set to measure network performance during training and the remaining 10% was used to test the accuracy of the network on unseen data after the network has been trained. Care was taken not to use the augmented data for a single sample in more than one of the training, validation or testing datasets as this would have violated the subject independent classification that the CNN was expected to achieve. This meant that an image with all of the augmented data generated from that image should have been in only one of the datasets. This was done by carefully labelling the data during preprocessing and perform the 70-20-10 split based on the original images rather than the entire dataset at random.

5.3.3 ANN performance metrics

In order to test the performance of machine learning algorithms there are many metrics that are often employed [Sokolova and Lapalme, 2009]. One of the most common ways to record the test results of a classification algorithm is in a confusion matrix. A confusion matrix is calculated by recording the number of true-positive (tp), true-negative (tn), false-positive (fp) and false-negative (fn) classifications. The confusion matrix can be used to calculate various other metrics. Some of the most common metrics for binary classification tasks such as face detection is accuracy and precision. Accuracy is known as the overall effectiveness of a classifier and is calculated by dividing the correctly classified samples by the total number of samples [Sokolova and Lapalme, 2009]. This is described mathematically as

$$\frac{tp + tn}{tp + tn + fp + fn} \tag{5.1}$$

Precision is known as the class agreement with the positive labels [Sokolova and La-

palme, 2009]. In other words, it is how well the system performed at detecting true positive cases. This can be described mathematically by

$$\frac{tp}{tp + fp} \tag{5.2}$$

5.3.4 Face CNN architecture and training

This section describes the methodology followed in the training of the face CNN on the face and non-face samples generated in Section 5.2.1, as well as some of the architectures explored in an attempt to produce a robust face classifier.

Minor variations on architectures from other authors that are known to perform well on similar tasks were used in order to create a face CNN that fits the model proposed in this work well. The four architectures that were investigated was the modified LeNet architecture by the Theano Development Team [2018] that is referred to as the LeNet architecture, the architecture by Garcia and Delakis [2004] that is referred to as the Garcia architecture, a custom architecture, and finally, a modified version of the architecture by Garcia and Delakis [2004] that is referred to as the Modified Garcia architecture. Each architecture described in this section was trained on the same datasets. The dataset contained a total of 41400 images, where 20400 were face images and 20400 were negative samples. The dataset was split into 28980 training images, 8280 testing images, and 4140 validation images. Each CNN was trained using a Stochastic Gradient Descent (SGD) training algorithm with a learning rate of 0,007, L1 decay of 0,001, L2 decay of 0,0005 and a momentum of 0,5. These values were inspired by the best practices outlined by Hinton [2012], however, some initial experimentation suggested that the selected values offer consistent results during training for the proposed architectures.

The final performance of each architecture was measured by taking the neural net with the highest test data performance and classifying 1000 images, 500 from each class. The resulting classifications were used to create confusion matrices for each architecture. These confusion matrices were used in order to calculate accuracy and precision metrics using the methods described in 5.3.3. These are all recorded in Table 5.5 at the end of this section for easy comparison.

The first architecture that was tested was a variation on the LeNet5 architecture by the Theano Development Team [2018]. This architecture was chosen to serve as a baseline for other architectures to be compared with, as the LeNet architecture has been used extensively for various applications in the past. The architecture can be seen in Figure 5.9. The architecture was slightly modified by inputting a slightly larger image than the original and reducing the 3 input channels to 1. This changes the size of the input layer from $28 \times 28 \times 3$ to $42 \times 42 \times 1$. This was done because the generated input images are grey-scale and do

not have separate red, green and blue channels and the size is adjusted in order to perform classification on a higher resolution image. The second modification that was made was the use of 2 output neurons for faces and non-faces rather than the 10 neurons used in the original architecture for hand-written digit classification. The loss during training was mapped out in Figure 5.17. The classification accuracy of the training and validation datasets during training is mapped out in Figure 5.10. This architecture achieved a minimum loss of 0,03012 on training data, with an accuracy of 96,1 and precision of 98,53 and a confusion matrix of the performance on validation data can be seen in Table 5.1.

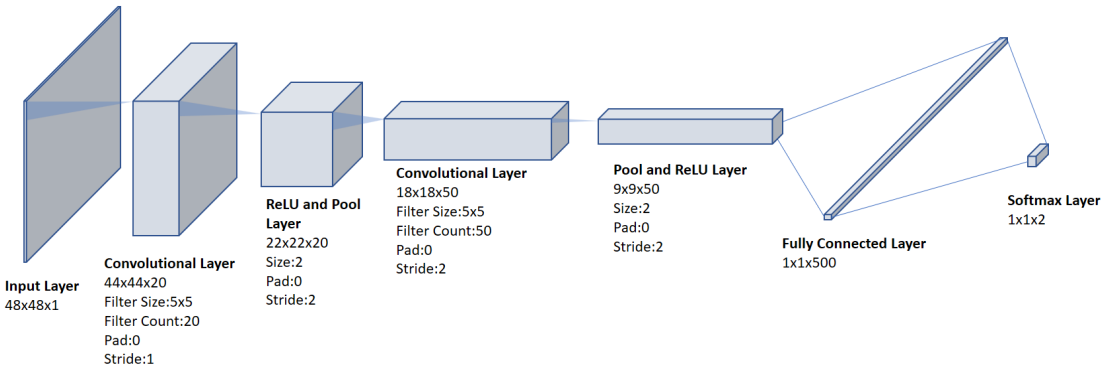


Figure 5.9: Proposed architecture for a face CNN based on a variation on the LeNet5 architecture as described by the Theano Development Team [2018].

	Face	Non-Face
Face (500)	468	32
Non-Face (500)	7	493

Table 5.1: Confusion matrix for the face CNN based on the work from the Theano Development Team [2018] on validation data.

The second architecture that was tested was an architecture suggested by Garcia and Delakis [2004], and is a CNN especially designed for face detection application. This architecture has been shown to perform well in literature. In this architecture, there was a dramatic reduction in the number of filters specified compared to the LeNet architecture. The LeNet5 architecture has 50 filters in the last convolutional layer, where this architecture only has 20 filters. This could speak to the more focused application of this neural network as opposed to CNNs with more general applications or more classes to classify. The layers tend to be larger than that of the LeNet architecture meaning that dimensionality is not reduced as drastically in the early layers as in the LeNet architecture. The architecture was modified from the original as a slightly larger input layer was used. Rather than the

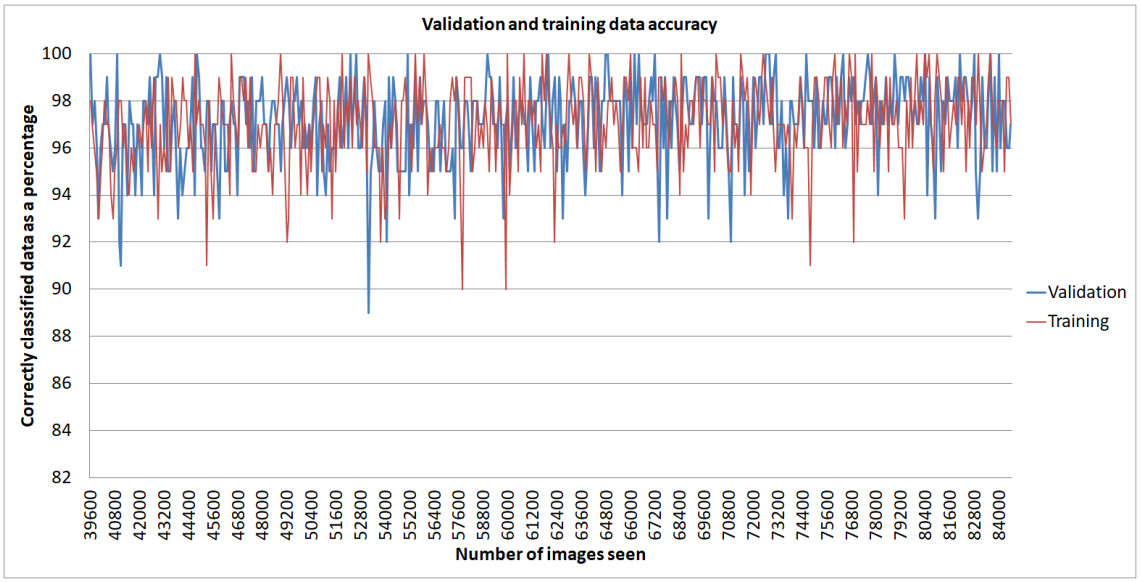


Figure 5.10: Accuracy of classification of samples from training and validation datasets as training progressed for the architecture proposed by the Theano Development Team [2018].

$32 \times 36 \times 1$ input layer, this version uses an input of $42 \times 42 \times 1$. This slight increase in size accounts for the larger regions used in the proposed system to detect faces. This architecture can be seen in Figure 5.11. The loss during training is mapped out in Figure 5.17. The classification accuracy of the training and validation datasets during training is mapped out in Figure 5.12. The minimum loss this CNN was able to achieve was 0,00112, the total accuracy was 98% and the precision was 99,79%. A confusion matrix based on the validation data can be seen in Table 5.2.

	Face	Non-Face
Face (500)	482	19
Non-Face (500)	1	498

Table 5.2: Confusion matrix for the face CNN based on the work from Garcia and Delakis [2004] as tested on validation data.

Next, a custom architecture was proposed in an attempt to produce a simplified architecture that achieves comparable performance. The idea was that dimensionality could be drastically reduced in the first convolution as was seen in the architecture by the Theano Development Team [2018], and a few large filters, as was used by Garcia and Delakis [2004], could be used to effectively model simple face sections in one layer and then combined in the fully connected layer. The suggested architecture can be seen in Figure 5.13. The loss

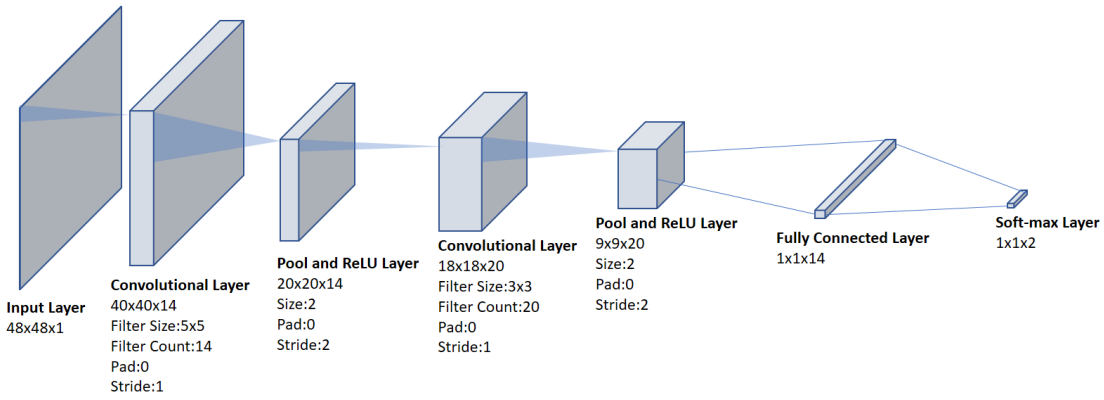


Figure 5.11: Proposed architecture for a face CNN based on the architecture described by Garcia and Delakis [2004].

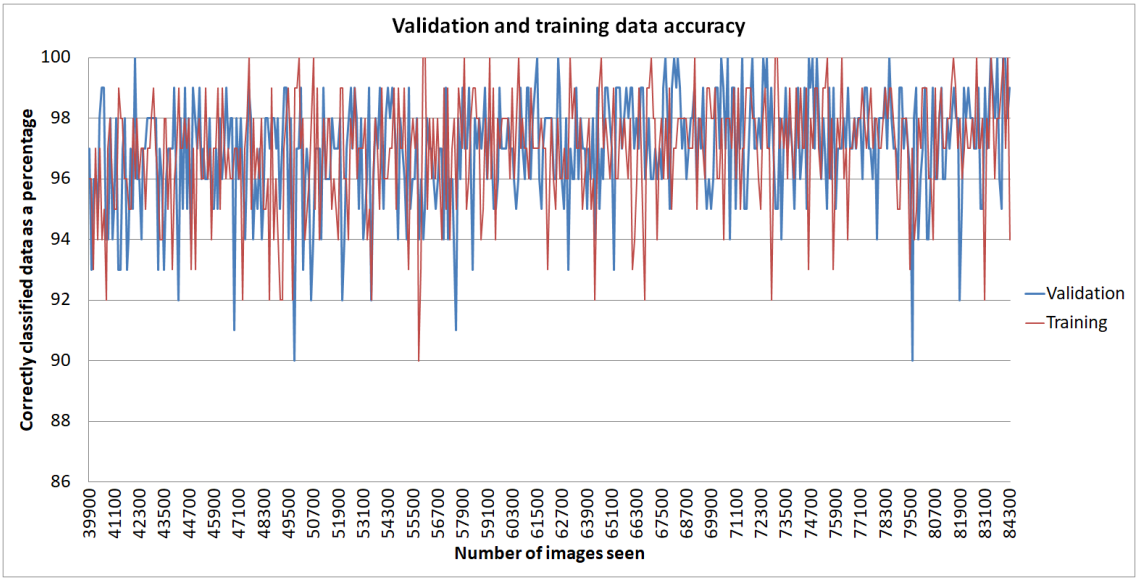


Figure 5.12: Accuracy of classification of samples from training and validation datasets as training progressed for the architecture proposed by Garcia and Delakis [2004].

during training is mapped out in Figure 5.17. It is interesting to note that even though this smaller neural net managed to learn much quicker, it simply was not able to achieve the accuracy seen on other architectures. The quicker learning was probably because of the reduced number of parameters, however since this architecture only has one convolutional layer, it does not effectively exploit the property of CNNs to combine simple features into more complex features and does not reduce the dimensionality effectively. The classifica-

tion accuracy of the training and validation datasets during training was mapped out in Figure 5.14. The confusion matrix for the resulting neural net based on validation data can be seen in Table 5.3. The minimum loss value was 0,03602, the accuracy achieved was 94,6% and the precision was 93,91%.

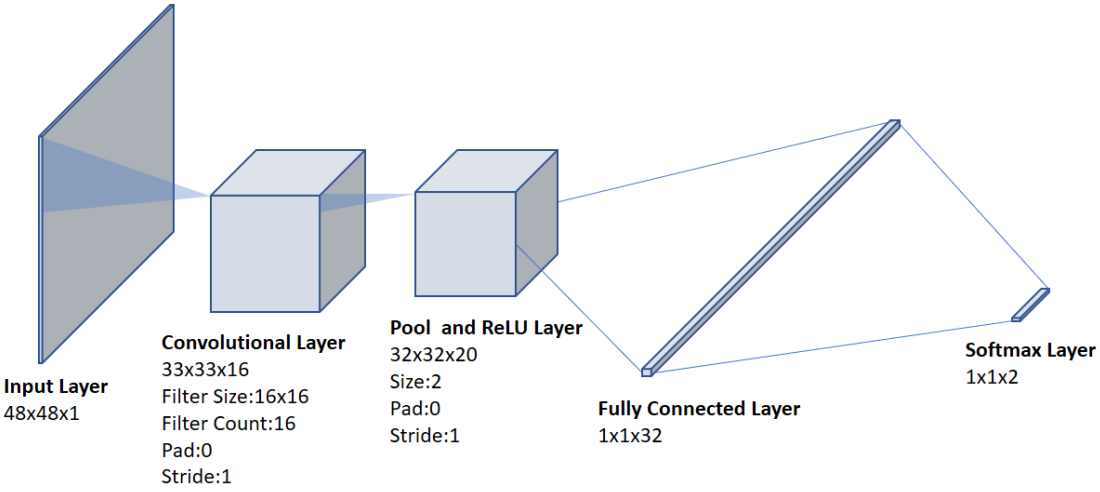


Figure 5.13: Proposed custom architecture for a face CNN.

	Face	Non-Face
Face (500)	478	23
Non-Face (500)	31	468

Table 5.3: Confusion matrix for a proposed face CNN tested on validation data

The final architecture used was based on the architecture suggested by Garcia and Delakis [2004] however a few changes were made to the original architecture. In order to cater for the larger input image, an extra pair of convolution and pooling layers where added in order to down-sample the image right at the beginning of the neural network. Also, more neurons were added to the fully connected layer in order to deal with more variation in the face region. This CNN achieved results very similar to that of the original by Garcia and Delakis [2004] tested earlier, however, it did manage to slightly improve performance. The classification accuracy of the training and validation datasets during training was mapped out in Figure 5.16. The confusion matrix for the resulting neural net based on validation data can be seen in Table 5.4. The minimum loss value that the network was able to achieve was 0,00591, the total accuracy was 98,1% and the precision was 99,18%.

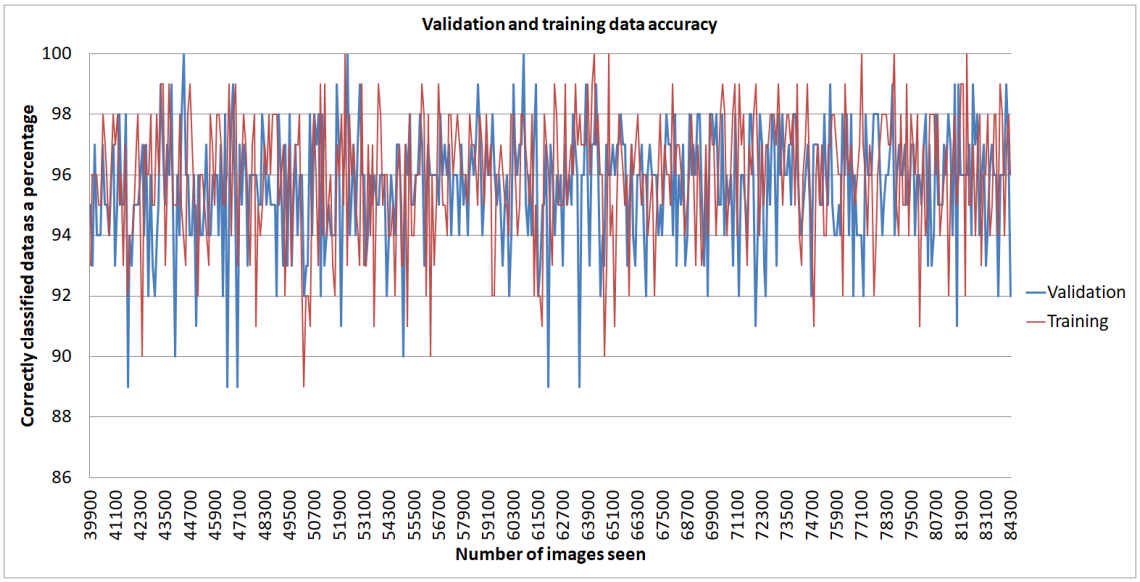


Figure 5.14: Accuracy of classification of samples from training and validation datasets as training progressed for the custom architecture.

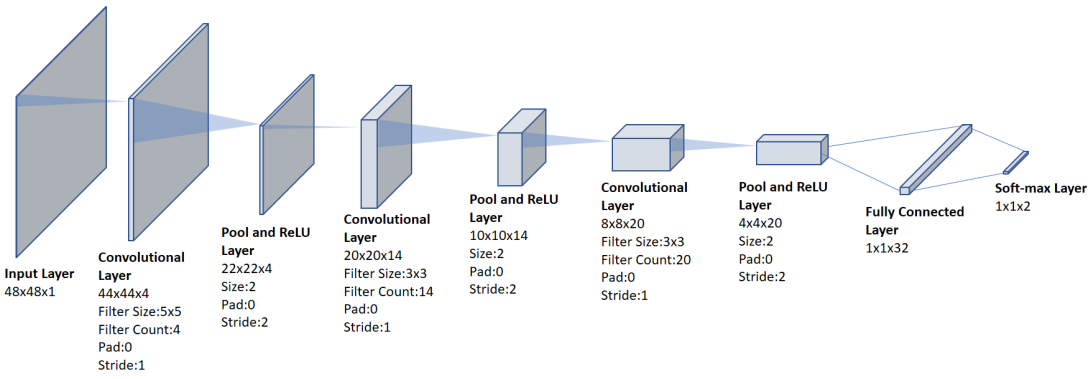


Figure 5.15: A modified version of the architecture by Garcia and Delakis [2004] for a face CNN

Table 5.5 shows that the modified version of the architecture by Garcia and Delakis [2004] had the best overall performance and was then selected as the final face CNN and was used in the final experiments.

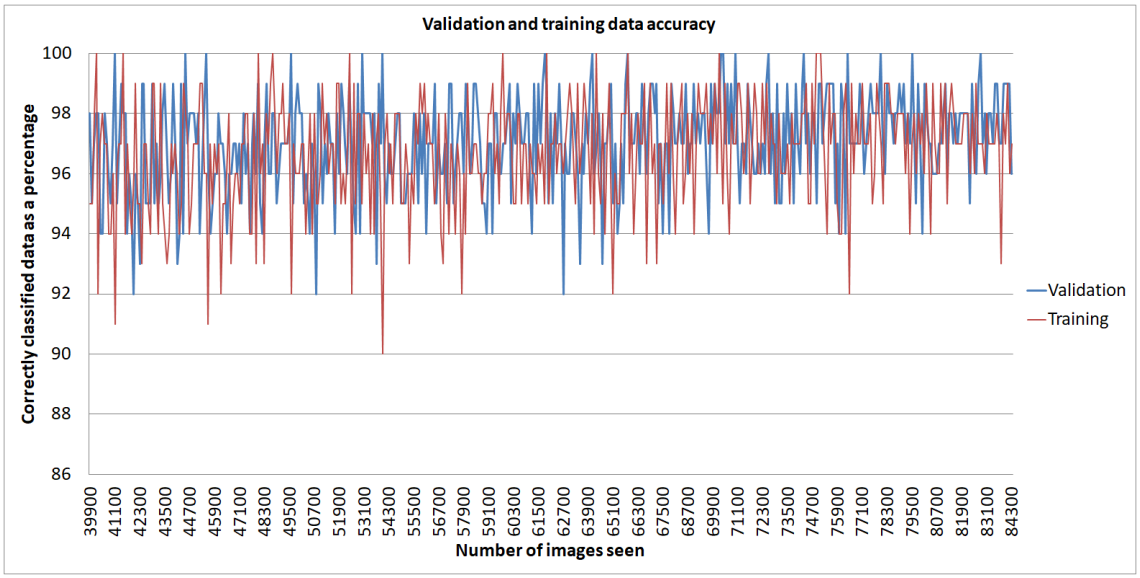


Figure 5.16: Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on Garcia and Delakis [2004].

	Face	Non-Face
Face (500)	485	15
Non-Face (500)	4	496

Table 5.4: Confusion matrix for the modified face CNN by Garcia and Delakis [2004] as tested on validation data

Architecture	Accuracy	Precision	Loss
LeNet [the Theano Development Team, 2018]	96,1	98,53	0,03012
Garcia [Garcia and Delakis, 2004]	98	**99,79**	0,00112
Custom	94,6	93,91	0,03602
Modified Garcia	**98,1**	99,18	**0,00591**

Table 5.5: A table where accuracy of the neural networks are compared

5.3.5 Face-part CNN architecture and training

In this section, various architectures proposed for the face-part CNN is described. In this section, 4 architectures are tested. The first is the modified LeNet architecture by the Theano Development Team [2018]. This architecture is referred to as the LeNetParts architecture. The second is the architecture by Garcia and Delakis [2004], and is referred to as the GarciaParts architecture. The third architecture is a modified version of the the

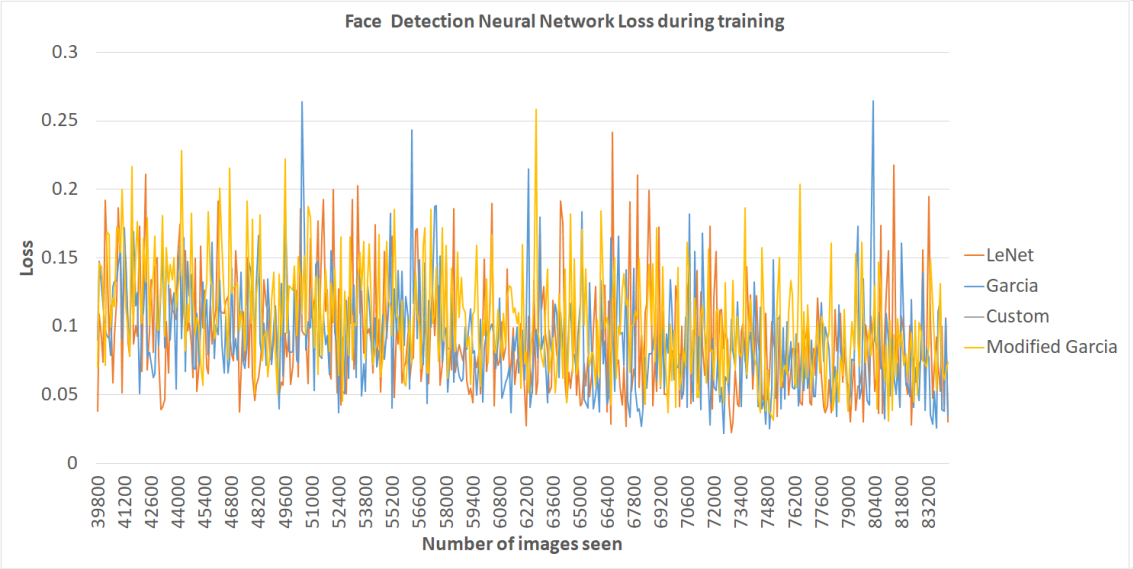


Figure 5.17: Face CNN architecture training loss comparison

Theano Development Team [2018] architecture, and is referred to as the Modified LeNet-Parts architecture. Finally, a modified version of the architecture proposed by Garcia and Delakis [2004] is presented and is referred to as the Modified GarciaParts architecture. Each architecture was trained on the dataset generated in Section 5.2.2, consisting of a total of 103800 images with 41520 for eyes, 20760 for nose, 20760 for mouth, and 20760 for non-parts. The dataset was split into 72660 training images, 20760 testing images, and 10380 validation images. The training methodology was the same as that described in Section 5.3. Each net was trained using a SGD training algorithm with a learning rate of 0,03, L2 decay of 0,001 and a momentum of 0,8.

The final performance of each architecture was measured by taking the neural net with the highest test data performance and classifying 1000 images, 250 from each class. The resulting classifications were used to create confusion matrices for each architecture. These confusion matrices were used in order to calculate accuracy metrics using the methods described in Section 5.3.3. These are all recorded in Table 5.10 at the end of this section for easy comparison.

The first architecture that was tested for performance on part detection was a variation on the LeNet5 architecture by the Theano Development Team [2018]. Once again the only changes that were made to this architecture was the use of a slightly larger input layer with only one dimension of $42 \times 42 \times 1$ versus the original with $28 \times 28 \times 3$. The other change

that was made was the use of 4 output neurons instead of the 10 used in the original. The final architecture for this CNN can be seen in Figure 5.18. The classification accuracy of the training and validation datasets during training is mapped out in Figure 5.19. The loss during training is mapped out in Figure 5.26. This architecture achieved a minimum loss of 1,24174 and a confusion matrix based on the classification of validation data can be seen in Table 5.1. It is important to note that these experiments where run multiple times using the same architecture and the results reported are the final results from these experiments. This means that these poor results are representative of the system performance and not due to an implementation or testing error. The total accuracy this architecture was able to achieve was 25%. It is evident that this network was not able to handle the complexity and variability of the classification of face parts as well as effectively reduce the dimensionality of the larger input layer.

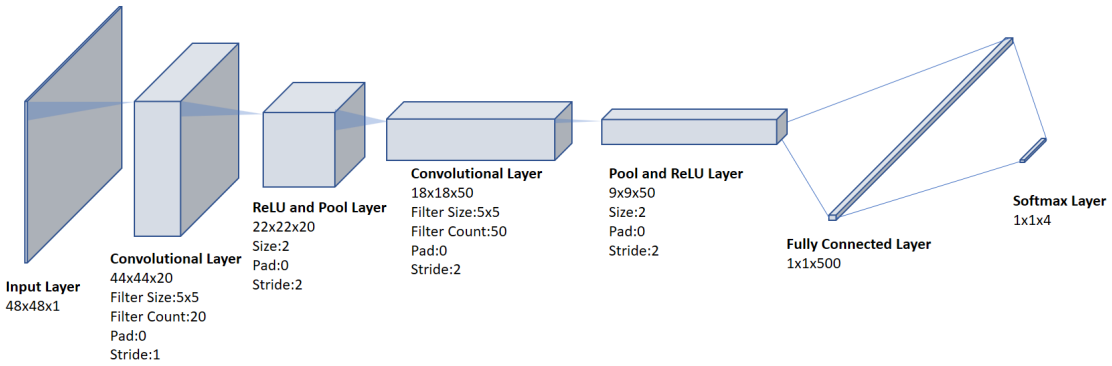


Figure 5.18: Proposed architecture for a face-part CNN based on the modified LeNet architecture described by the Theano Development Team [2018]

	Eye	Nose	Mouth	Random
Eye (250)	0	250	0	0
Nose (250)	0	250	0	0
Mouth (250)	0	250	0	0
Random (250)	0	250	0	0

Table 5.6: Confusion matrix for a part detector based on the work from the Theano Development Team [2018] tested on 1000 validation images

The second architecture that was tested was the architecture suggested by Garcia and Delakis [2004] applied to face part detection. The architecture was once again modified to use and $42 \times 42 \times 1$ input layer rather than an input layer of size $32 \times 36 \times 1$. The second modification is the use of 4 output neurons over the single neuron described in

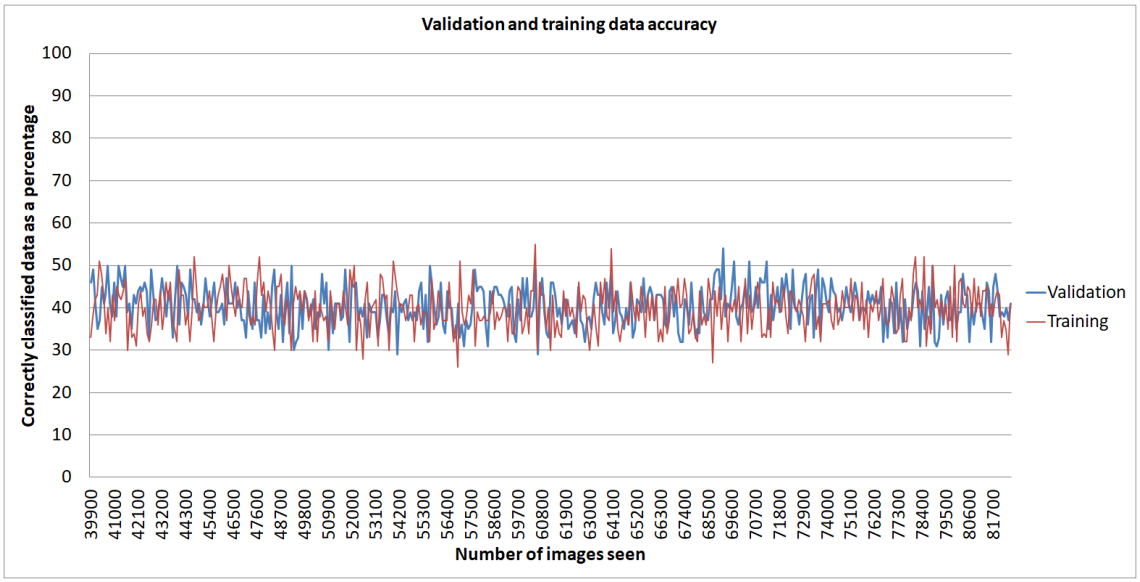


Figure 5.19: Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on the modified LeNet architecture by the Theano Development Team [2018].

literature. This architecture can be seen in Figure 5.20. The classification accuracy of the training and validation datasets during training is mapped out in Figure 5.21. The loss during training is mapped out in Figure 5.26. The minimum loss this CNN was able to achieve was 0,82124 and a confusion matrix based on the validation dataset can be seen in Table 5.7. The accuracy that this architecture was able to achieve was 49,3%. Although this CNN did manage to train better than the LeNet based architecture, it still did not manage to generalize effectively. A likely cause for this is the small number of highly connected neurons in the fully connected layer. These are responsible for combining the features extracted during the convolutions and making an accurate classification, and the high variability in the classes that are being detected could account for the failure of the network to classify these effectively.

The next architecture was based on the modified LeNet architecture by the Theano Development Team [2018], however, more layers were added to deal with the increased number of categories to be classified. This should lead to sufficient dimensionality reduction of the input, as well as, allow for sufficient features to be extracted to be effectively combined and classified in the final layer. The final architecture can be seen in Figure 5.22. The classification accuracy of the training and validation datasets during training is mapped out in Figure 5.23. The loss during training is mapped out in Figure 5.26. The minimum loss this CNN was able to achieve was 0,55120 and a confusion matrix of the classifica-

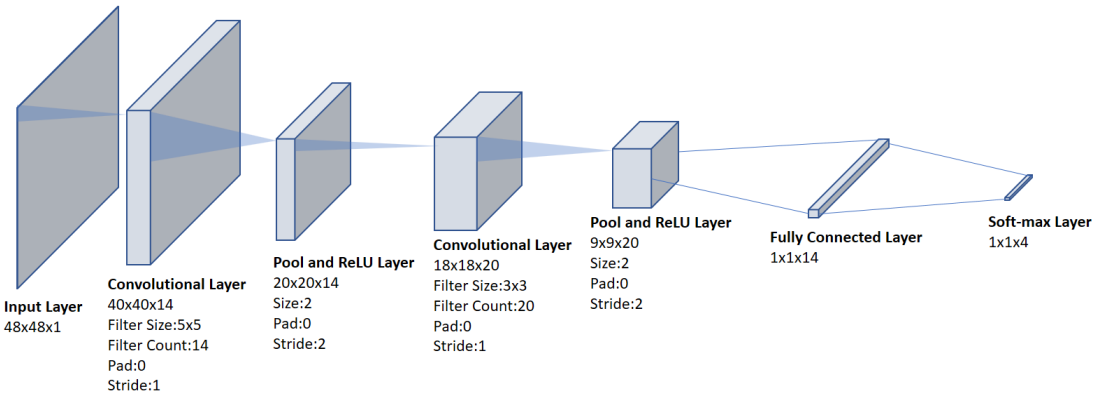


Figure 5.20: Proposed architecture for face-part CNN based on the architecture described by Garcia and Delakis [2004]

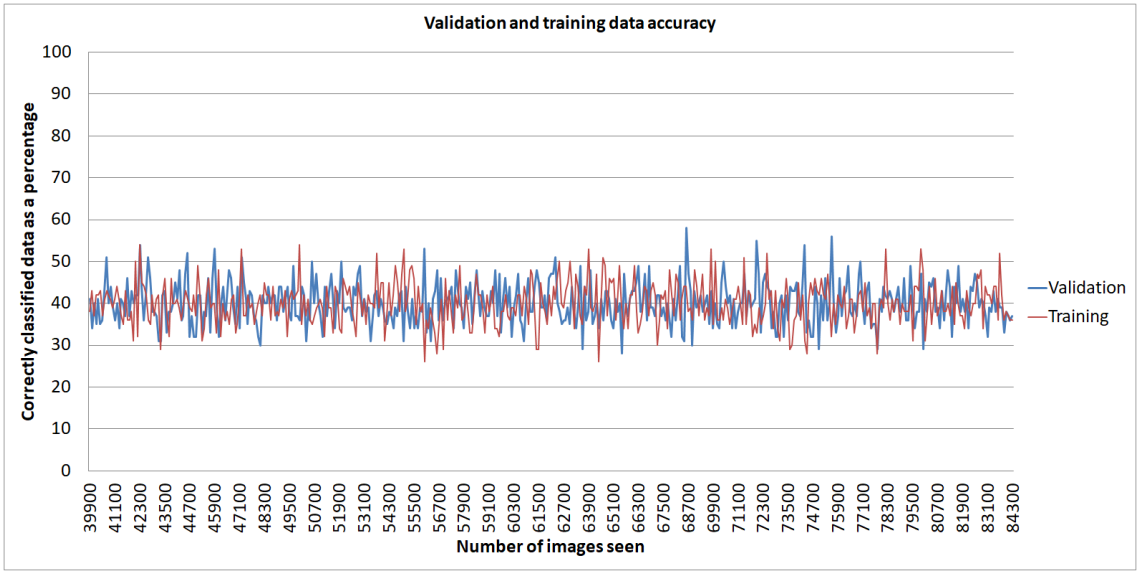


Figure 5.21: Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on Garcia and Delakis [2004].

tion of validation data can be seen in Table 5.8. This architecture managed to achieve an accuracy of 49,4%. This was a significant improvement over the previous version of this architecture however the drastic reduction in layer size in the initial convolution layer could have caused some important information not to be modelled effectively.

The final architecture used was a modified version of the architecture suggested by Garcia

	Eye	Nose	Mouth	Random
Eye (250)	31	74	106	39
Nose (250)	0	226	18	6
Mouth (250)	1	48	101	100
Random (250)	6	24	85	135

Table 5.7: Confusion matrix for the face-part CNN based on the work from Garcia and Delakis [2004] on 1000 validation images

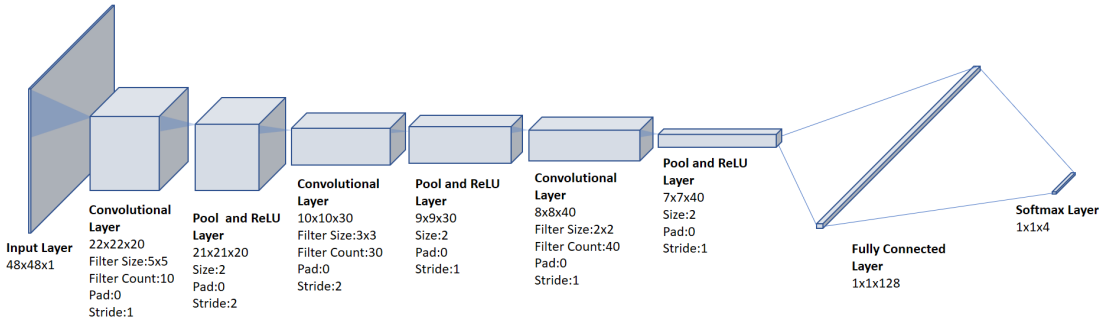


Figure 5.22: Proposed architecture for face-part CNN based on a modified version of the LeNet architecture proposed by the Theano Development Team [2018].

	Eye	Nose	Mouth	Random
Eye (250)	127	78	20	25
Nose (250)	18	219	2	11
Mouth (250)	95	44	90	21
Random (250)	65	66	61	58

Table 5.8: Confusion matrix for a proposed face-part CNN

and Delakis [2004]. In the previous version, the network was not able to learn to consistently classify all 4 classes. In order to improve on this network the number of filters was increased to allow for more features to be learned. An extra convolution layer was also added to assist in dimensionality reduction of the slightly larger input. Finally, the number of neurons in the fully connected layer was increased to add more possible classifications to the network. In this architecture the number of filters where increased to cater for the increased number of categories the network was expected to cope with. The final architecture can be seen in Figure 5.24. The classification accuracy of the training and validation datasets during training is mapped out in Figure 5.25. This architecture managed to achieve a minimum loss of 0,26313 and the confusion matrix of the validation data for the resulting neural net can be seen in Table 5.9. This architecture managed to

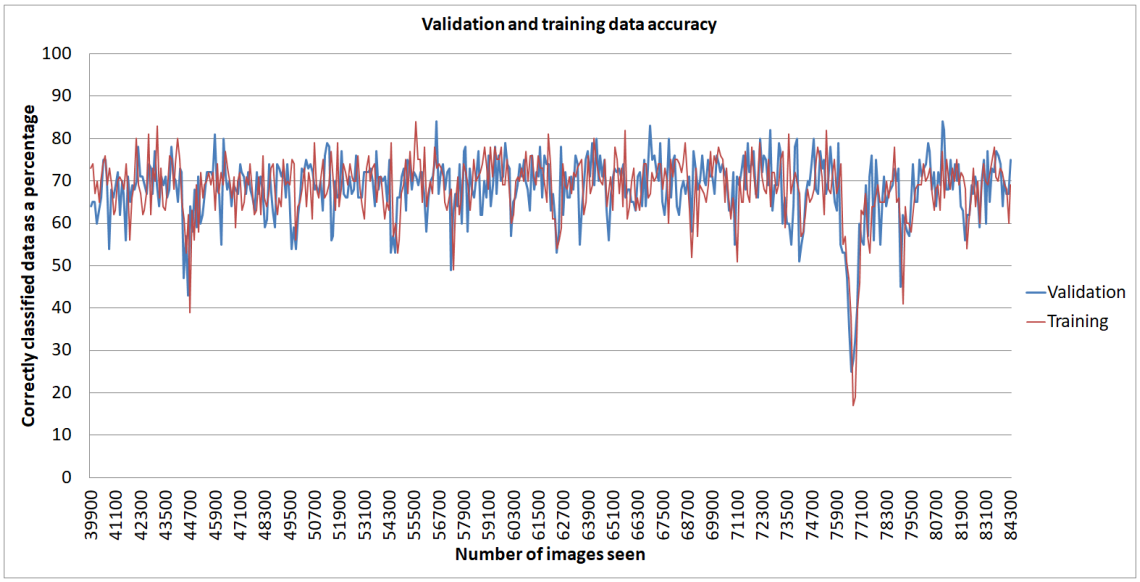


Figure 5.23: Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on the Theano Development Team [2018].

achieve a total accuracy of 75,3%. This is a significant improvement over any of the other architectures that were tested.

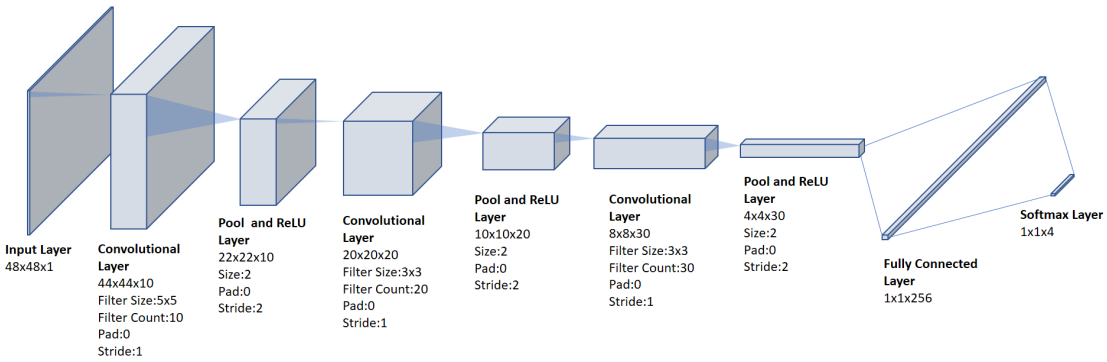


Figure 5.24: Proposed architecture for Part detector neural network

From the final comparison in Table 5.10 it is quite evident that the final architecture that was proposed significantly outperformed the other proposed architectures on validation data. This final architecture was then selected as the final face-part CNN and was used in the final experiments.

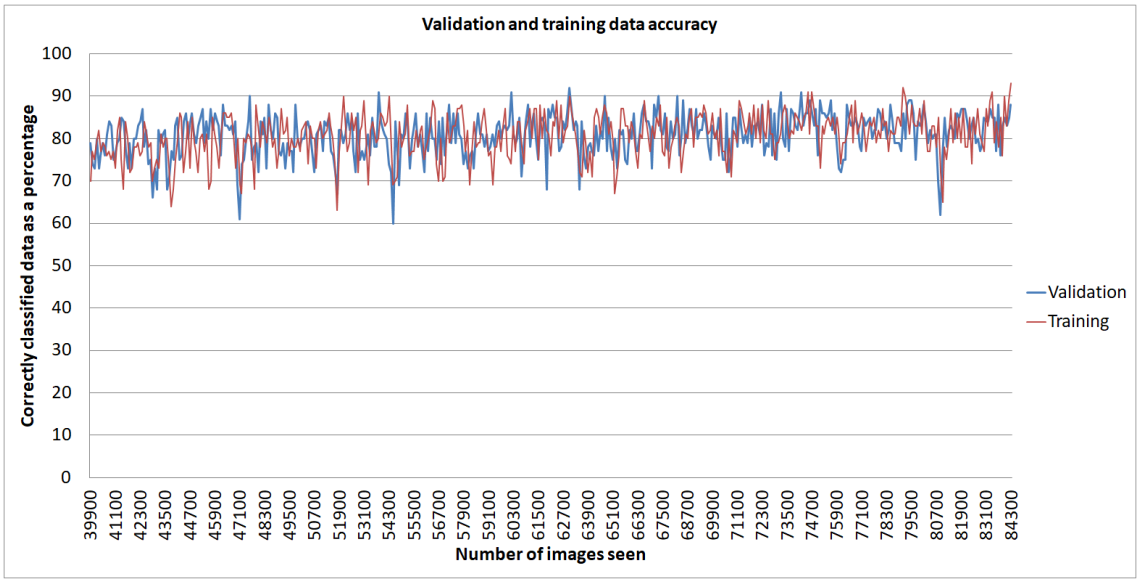


Figure 5.25: Accuracy of classification of samples from training and validation datasets as training progressed for the modified architecture based on Garcia and Delakis [2004].

	Eye	Nose	Mouth	Random
Eye (250)	179	18	25	28
Nose (250)	33	211	0	6
Mouth (250)	33	20	167	30
Random (250)	24	14	16	196

Table 5.9: Confusion matrix for final face-part CNN as tested on 1000 validation samples.

	Accuracy	Loss
LeNet [the Theano Development Team, 2018]	25	1,24174
Garcia [Garcia and Delakis, 2004]	49,3	0,82124
Modified LeNet Figure 5.22	49,4	0,55120
Modified Garcia Figure 5.24	**79,3**	**0,26313**

Table 5.10: A table where accuracy of the face-part detector neural networks are compared

5.4 Geometric feature training

In this section, the method of calculating the geometric features, used to calculate the spring cost, is discussed. Training is performed on the geometric data generated in Section 5.2.3. The constraints between the patches were trained statistically from the provided feature points of the normalized images. For every face-part, the corresponding face-part from

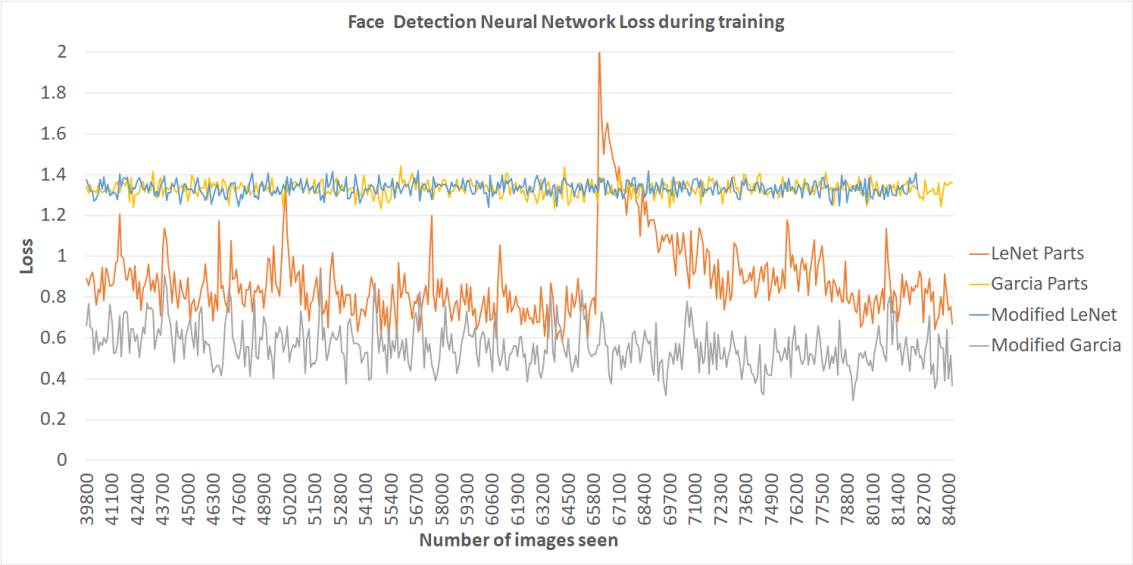


Figure 5.26: Face-part CNN architecture training loss comparison

every image was used to calculate the median point of the face-part. In other words, all the feature points associated with each part were summed up and divided by the number of feature points that make up the part in question. This produced the average placement for the part in a certain image. The average location for each part in the training images was then used to calculate the median placements of each part for the entire training dataset. These placements could then be used as the locations that the face-part CNN considers when determining the presence or absence of a part. Using the median point of the part makes the generalization less sensitive to outliers. The second statistical method that was used to describe the geometric aspects of the selected point is the use of a 95% covariance error ellipse around the most statistically significant points. This allows the system to describe the variance in the position as well as the direction of the greatest variance in 2D space. A visualization of the end result can be seen in Figure 5.27.

A covariance error ellipse or confidence ellipse is a way to describe normally distributed 2-dimensional data by means of an ellipse that contains some percentage of all of the observed points. The reason that this method is more advantageous than fitting a multi-variate Gaussian is because it allows more fine grained control over the probabilities generated for points falling within the range of plausible points. The first step of calculating an ellipse was to calculate the covariance matrix of the data [Spruyt, 2014]. In order to calculate an ellipse with an arbitrary rotation, the angle of the ellipse has to be determined. This was done by calculating the eigenvectors and values of the covariance matrix. The direction

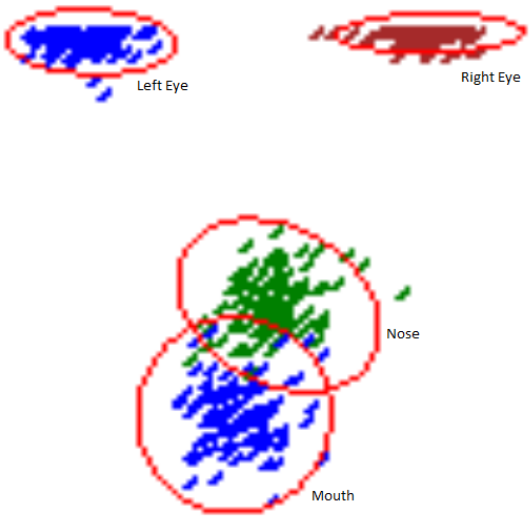


Figure 5.27: Confidence Ellipses calculated from observed points.

of the greatest variance in the observed points is described by the eigenvectors, and the variance is described by the corresponding eigenvalue [Spruyt, 2014]. When the covariance calculated in the covariance-matrix equals zero, it means that the data is not correlated and thus the spread is aligned to one of the axes, and the variance in an axis would simply be equal to the spread of the data in the x and y directions [Spruyt, 2014]. In this special case, the ellipse can be described by the equation for a standard ellipse where s is defined by the cumulative Chi-Square distribution at the degree of certainty one would like to describe the data. In this case, s was calculated with a 95% confidence with 2 degrees of freedom and was equal to $5,991$.

$$\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2 = s \tag{5.3}$$

An axis aligned ellipse could then, be calculated as an ellipse with the major axis calculated as $2\sigma_x\sqrt{5,991}$, and the minor axis could be calculated as $2\sigma_y\sqrt{5,991}$ where σ_x is the standard deviation in the x-axis and σ_y is the standard deviation in the y-axis.

In order to calculate arbitrary ellipses, that are not axis-aligned, the major and minor axis of the ellipse can be calculated using the eigenvalues calculated from the covariance matrix [Spruyt, 2014]. The major axis was calculated as $2\sqrt{5,991\lambda_1}$, and the minor axis was calculated as $2\sqrt{5,991\lambda_2}$ where λ_1 and λ_2 are the eigenvalues calculated from the covariance matrix.

The rotation of the ellipse was calculated simply as the angle between the largest eigenvector and the x-axis [Spruyt, 2014].

$$\alpha = \arctan \frac{v_1(y)}{v_1(x)} \tag{5.4}$$

where v_1 is the largest eigenvector calculated from the covariance matrix. This provided all the parameters to sufficiently describe the part placements as the coordinates of the median points, the heights and widths, and the angles of all the part ellipses was now known.

The output matrix was then combined with the face geometry, calculated during training. The face geometry is represented as four ellipses, one for each face part, each described by a centre point, variance in the x- and y-axes, rotation. This corresponds to the shapes described in Section 4.2.1. Each of the part responses was then multiplied in the output matrix with a function describing the spring-cost of the part, which was a function of the position of the detection and the parameters of the face geometry for the part. This was done with each of the responses that correlate to every face-part which does not have the "negative" label, as this is simply ignored during spring cost calculation. The resulting spring-cost value was then added to the response from the neural network. This results in a value that represents both visual and geometric likelihood of a positive response. This process can be seen in Figure 4.8. The final layer of the network uses a sigmoid activation function, producing a value between 0 and 1. The spring cost that is calculated produces a value from 0 to 1 based on the distance and direction from the ellipse, where 1 is the response correlating to the highest likelihood of a positive placement. This makes it easy for visual and geometric information contribute equal amounts to the final detection response as both of these features produce values between 0 and 1.

5.5 Summary

In this chapter, a description of the dataset used during training, and preparation thereof is given. This chapter also explains how the face and face-part CNN is trained and evaluated on training data. Finally, the process of calculating the geometric features used during localization is shown. Now that the system is trained and the geometric features are calculated,the next chapter evaluates performance of the CNNs as well as the entire system on previously unseen data from the COFW dataset.

Chapter 6

Results

In this chapter, the dataset used for testing the system performance on unseen images is discussed in Section 6.1. This is followed by a discussion on how the data from the testing dataset was prepared for testing in Section 6.2 and following that, the performance of the final CNN architectures on testing data is presented in Section 6.3. This is followed by a discussion on how system performance is measured in Section 6.4.1. Then the performance of the proposed part detection techniques is presented in Section 6.4.2 as well as the performance of the face-part localization based on pre-cropped images in Section 6.4.3. This is followed by an examination of the results where examples of some of the resulting localizations are presented in Section 6.5 as well as an analysis of the effect of occlusion and out-of-plane rotation on localization error. Finally the results of the proposed methods are compared to other similar systems in Section 6.6.

6.1 Testing dataset

In order to simulate testing the system in the wild, a completely new dataset was used to evaluate system performance. This ensures that none of the samples used in the training dataset would be found in the testing data with which the system is presented in this section. This also validates that the system did not learn any dataset specific features that could aid in the classification of faces or face-parts.

The COFW dataset was specifically chosen to measure system performance under higher degrees of occlusion and pose variation. This dataset was created specifically to benchmark face landmark algorithms in challenging conditions [Burgos-Artizzu et al., 2013]. The images from the COFW dataset are significantly more occluded on average than those of the HELEN dataset, with 23% of all feature points occluded [Burgos-Artizzu et al., 2013]. A few example images from the dataset can be seen in Figure 6.1. The dataset consists of 507 images each with 28 facial feature points localized on the face and the occluded points are

explicitly labelled as occluded. The feature points can be seen in Figure 6.2. The proposed system was not trained on any of the images in this dataset in order to ensure subject independence. By using this dataset exclusively to measure system performance it replicates real world scenarios where the subjects are unseen and the data capture techniques could vary drastically.



Figure 6.1: Example images from the COFW Dataset [Burgos-Artizzu et al., 2013].

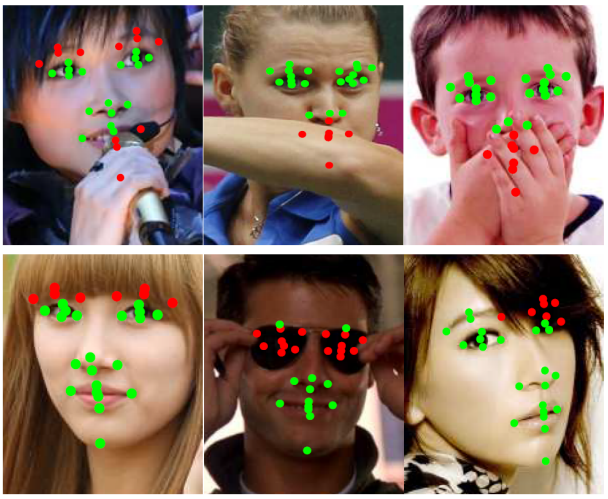


Figure 6.2: Example feature point locations on images from the COFW Dataset [Burgos-Artizzu et al., 2013]. Red points are occluded and green points are visible

6.2 Preparation of test data

In this section, the method used to prepare normalized test data is presented. For testing of face-part localization as well as CNN performance, 3 datasets were generated. The first was a dataset of images with the face normalized, the second was a dataset of cropped faces, and finally a dataset of face-parts.

In order to generate a set of images containing normalized faces, the raw data from the COFW dataset was used to generate the 507 normalized images. In order to avoid the use of a pyramid of features approach or an explicit search over possible in-plane rotation during localization, the testing data was normalized with regards to scale and rotation. This was done following the same process as outlined in Section 5.2.1, with a slight difference in that the normalized faces were not cropped from the larger image but rather translated into the middle of the image. This was to prevent errors from occurring when faces are localized close to the edge of the image. This resulted in 507 normalized images for the use in testing system performance. Samples from the produced dataset can be seen in Figure 6.3. It is important to note that the resulting images do not have the same size, however, the faces contained within the images do have a consistent size and rotation.



Figure 6.3: Samples from the normalized COFW dataset that were used to test the performance of the proposed system.

A second dataset was a dataset of 507 cropped faces generated following the full normalization process as described in Section 5.2.1. This dataset of pre-cropped faces was used to test the performance of the face-part localization without the added complexity of dealing with face localization first. This data was also used to test the performance of the face CNN on unseen testing data. A sample of pre-cropped face data from the COFW can be seen in Figure 6.4.

Finally a dataset of face-parts was constructed from the COFW dataset following the methodology described in Section 5.2.2. This was used to test the performance of the



Figure 6.4: Samples from the normalized COFW dataset that were used to test the performance of the proposed system.

face-part CNN on unseen data.

6.3 Face and face-part CNN performance

In this section, the performance of the final face CNN and the face-part CNN on unseen testing data is presented. The face CNN was tested on 1000 prepared face images from the COFW dataset with both face and non-face classes represented equally. The architecture was able to achieve an accuracy of 93,8% and a precision of 90,28% on unseen data. This is comparable to the accuracy achieved on validation data during testing and from the confusion matrix shown in Table 6.1 it looks like a relatively low number of false positives are generated.

	Face	Non-Face
Face (500)	492	9
Non-Face (500)	53	446

Table 6.1: Confusion matrix for the final trained face CNN architecture on unseen testing data.

The face-part CNN was tested on 1000 face part images generated from the COFW dataset. The images were selected at random, however, the images were balanced to ensure that 250 samples from each class were included in the testing of the face-part CNN. This CNN architecture was able to achieve an accuracy of 70,2% on unseen data. The confusion matrix produced by the face-part CNN can be seen in Table 6.2. It is interesting to note the high number of failures in the "Random" class.

6.4 System performance

In this section, a description of how performance of the proposed DPM model was calculated is given. This is followed by the resulting performance of the system. The first set

	Eye	Nose	Mouth	Random
Eye (250)	177	30	28	15
Nose (250)	30	220	0	0
Mouth (250)	56	4	190	0
Random (250)	26	52	57	115

Table 6.2: Confusion matrix for the final part CNN as tested on 1000 unseen testing images.

of results shows the base performance of the system as it is intended to be applied where face and part detection takes place simultaneously. The second set of results show the performance of the part localization algorithm when face detection was ignored. The average distance from each part placement to the ideal placements is plotted as a percentage of the distance between the centres of the eyes. This means that an error of greater than 100% is possible when the detection is off by an average distance greater than the distance between the eyes.

6.4.1 Measurement metrics

In order to measure the accuracy of DPMs and similar systems, a very simple formula is used to calculate comparable results. The system was tested on a set of testing data and the detected points were recorded. The euclidean distances from the detected points to the ground-truth points were then calculated and these error distances were added together. This value was then divided by the number of feature points to get the average error distance. Mathematically the average error can be calculated by:

$$averageTrueError = \frac{\sum \delta_i}{numberOfParts} \tag{6.1}$$

Where δ_i is the euclidean distance between the detected part i and the ground truth location of part i. This error was divided by the distance between the eyes and multiplied by 100 in order to get the average error as a percentage of the distance between the eyes. As noted by Zhu and Ramanan [2012], this system of measurement assumes that both eyes are visible and assumes near-frontal faces. This can be written as

$$normalizedError = \frac{averageTrueError}{\delta_{eye}} \tag{6.2}$$

Where δ_{eye} is the euclidean distance between the two eye centres.

The final accuracy is often described using the average of the normalized error of each detection. Outliers are ignored by adding the constraint that every detection with an error of greater than a cut-off percentage is considered a failed detection and is not included in the calculation of the average error. The in literature, the cut-off error is often set to

10% [Burgos-Artizzu et al., 2013, Sun et al., 2013, Dong and Wu, 2015]. These values are recorded as errors and the goal is to reduce this number and reduce the average error. It is also common to plot the normalized error values of the dataset on a cumulative error distribution curve for easy visualization. This curve has the normalized error on the x-axis and the percentage of the test dataset on the y-axis. The resulting curve would describe the number of images with an error smaller than the error measured on the x-axis. This means that, given normally distributed data, the graph should represent an S-curve with y values ranging from 0% to 100%.

6.4.2 Face and face-part localization results

The first experiment was performed on 100 random normalized images generated using the methodology described in Section 6.2. Figure 6.5 shows the cumulative error distribution of the 100 images sampled. Table 6.3 shows the average error, median error as well as the standard deviation of the various techniques. It also documents the percentage of instances where the algorithm did not manage to localize any face in the image. These cases are recorded as failure cases as all of the test images contain exactly one face.

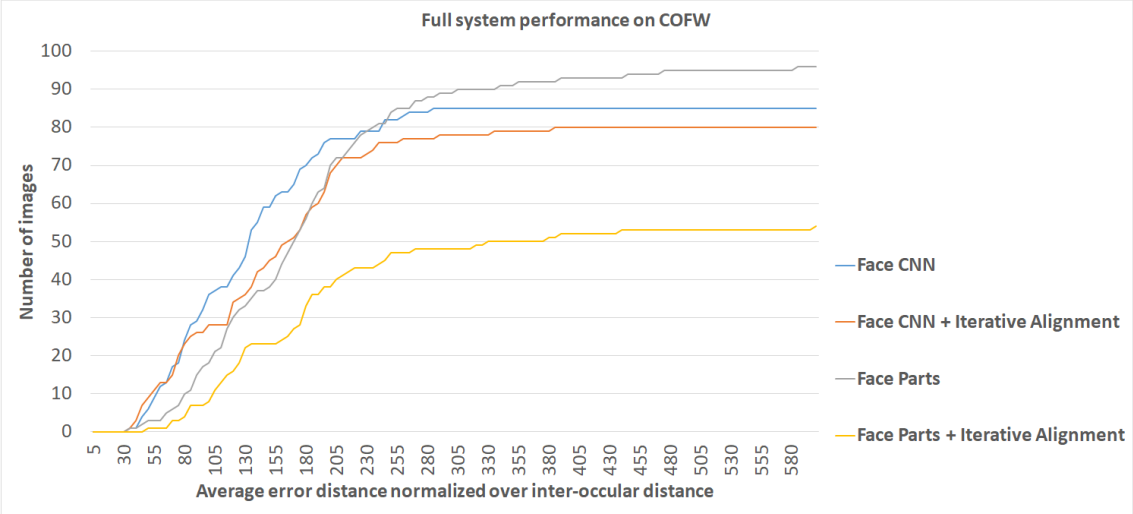


Figure 6.5: Cumulative error graph for face part localization performance on a sample of 100 images from the COFW dataset.

6.4.3 Face-part localization

In order to test the performance of the face-part localization step in isolation, the experiment was performed on pre-cropped face images. Figure 6.6 shows the cumulative error

Method	Average Error	Median Error	Standard Deviation	Failures
Face CNN	**125,11%**	**122,25%**	**58,86%**	14%
Face CNN + Iterative Alignment	139,00%	138,16%	71,28%	19%
Face-part CNN	174,84%	168,84%	90,96%	**3%**
Face-part CNN + Iterative Alignment	180,41%	171,97%	101,51%	45%

Table 6.3: Statistics around the results. Error is measured as a percentage of the distance between the eyes.

distribution of the 100 images sampled for part localization given a correctly cropped face. Table 6.4 shows the average error, median error as well as the standard deviation and the percentage of failure cases for each face part.

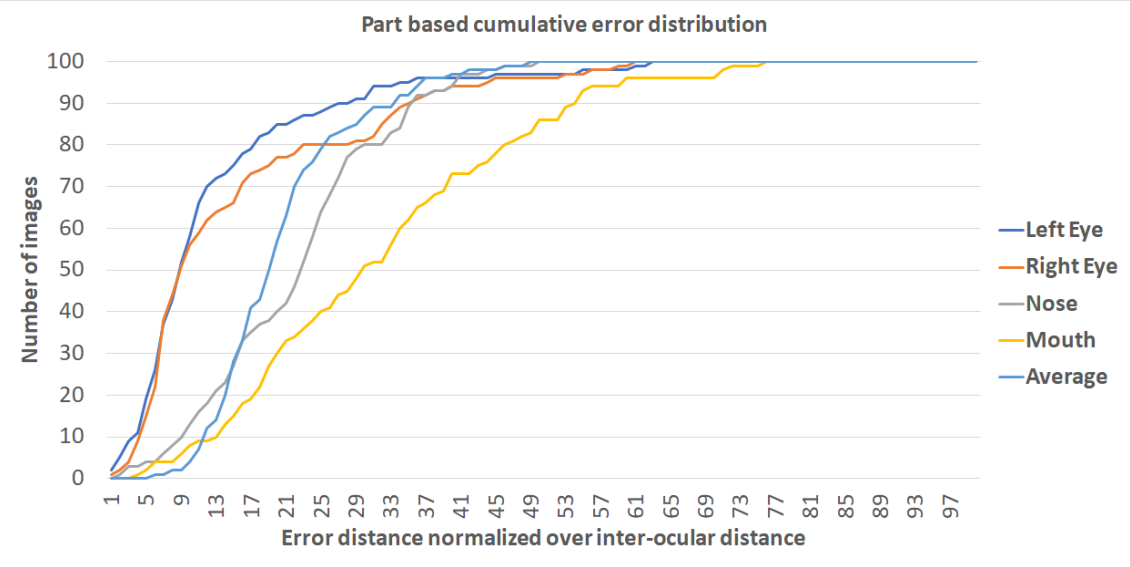


Figure 6.6: Cumulative error graph for part localization performance on a sample of 100 normalized face images from the COFW dataset.

From Table 6.4 it can be seen that there are no failure cases. This is because of the way that the spring-cost model would force a detection even if the CNN does not manage to make any detection. This behaviour is expected and is particularly useful when dealing with occlusion in the face region. It can also be seen from Figure 6.4 that the results for the two eyes are different. It is possible for the results for the two eyes to differ, as images with out of plane rotation would have increased spring-cost for one, or both, of the eyes. In some of the images used for testing, the two eye regions could also look quite different because of out-of-plane rotation or occlusion. It is also evident from the graph that the system had

Method	Average Error	Median Error	Standard Deviation	Failures
Left Eye	12,51 %	8,89 %	11,75 %	0
Right Eye	14,99 %	8,70 %	13,64 %	0
Nose	22,10 %	22,62 %	10,28 %	0
Mouth	31,60 %	29,64 %	16,35 %	0
Combined	20,30 %	18,96 %	8,21 %	0

Table 6.4: Face part detection results

some difficulty reliably localizing the mouth region. This is somewhat unsurprising as the mouth region has the most variability in terms of shape and appearance.

6.5 Discussion

In this section, examples for the best and worst localizations on images for each variation on the proposed algorithm are shown, in order to get some insight as to where the algorithm performs well and fails as well as examining the overall system performance. This section also includes an analysis of the impact of CNN training on face detection in Section 6.5.3 as well as the impact of occlusion in Section 6.5.4 and the impact of out-of-plane rotation in Section 6.5.5.

6.5.1 Face and face-part localization

The performance of the combined face and face-part localization is discussed in this section and possible reasons for the observed results is given.

6.5.1.1 Face part localization using a face CNN

The results shown in Figure 6.5 show quite definitively that this method produced more accurate localizations than the other proposed methods with an average error of 125,11 % of the inter-ocular distance. It did, however, have a high error rate of 14% meaning that the in 14% of the cases, the algorithm was unable to detect a face in the input image.

Figure 6.7 shows the best localization for this technique. An error distance of 34,79% is still very high when compared to other techniques. There were quite a few strong false positive responses that did influence the final localization. There was also a very strong response on the face region itself, however, this point would only include part of the face and it is unclear why this region would cause such a strong response. There were no strong detections close to the actual face region and the detections seem to have clustered around darker parts of the image. This could be because the face CNN has not been exposed to negative samples that were sampled from the normalized images. It is also important to

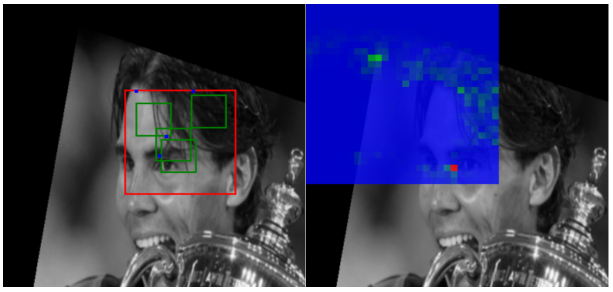


Figure 6.7: a) An example of the most accurate localization using a face CNN with an average error of 34,79 %. b) The face CNN response over the image that was used to calculate the face region.

note that the face region is heavily rotated and occluded. The average of the false positive samples did end up averaging to somewhere close to the actual face region. Even though this is reported as the most accurate localization, it is evident that this is only due to the dimensions of the image and not the accuracy of the algorithm.

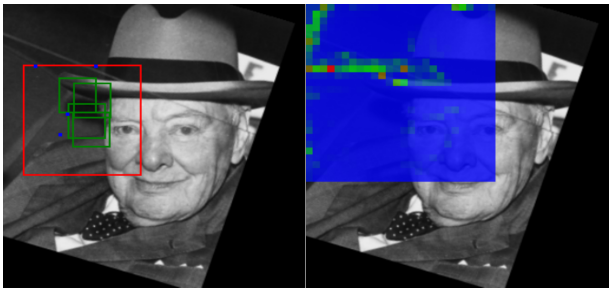


Figure 6.8: a) An example of one of the least accurate localizations using a face CNN with an average error of 284,91%. b) The face CNN response over the image that was used to calculate the face region.

Figure 6.8 depicts the worst localization for this technique. From the response map of the face CNN, it appears that the network has a tendency to favour straight lines and partial facial regions whilst not producing any strong responses on the face itself. This might be a symptom of an over-trained network.

6.5.1.2 Face part localization using a face CNN with iterative alignment

The results shown in Figure 6.5 shows quite definitively that the addition of the iterative alignment step led to a net decrease of accuracy as well as an increase in errors. Errors increased from 14% to 19% of the total images, and the average localization error increased with 13,89% to 139,00% of the inter-ocular distance. The failure of the iterative align-

ment step might be due to the fact that initial face region localization is not accurate. It could also be because of the high confusion seen in the part detector that would cause a false positive response of a part, moving the image further from the true location of the face.

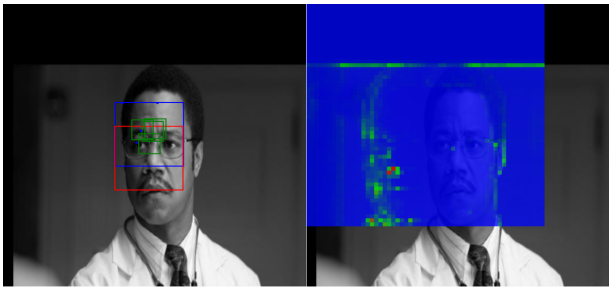


Figure 6.9: a) An example an accurate localization using a face CNN combined with an iterative alignment step, resulting in an average error of 32,21% of the inter-ocular distance. The red square shows the initial localization for the face region, and the blue square shows the localization after the alignment step has been performed. b) The face CNN response for faces over the image that was used to calculate the face region.

Figure 6.9 depicts the best localization for this technique. It is also evident that the alignment step actually moved the final localization further from the optimal location. It is also interesting to note that when the response of the face detection neural network is considered, it highlights the networks affinity to horizontal lines, as well as the edges of faces. This might be caused by anomalies in the training data or the fact that this structure was never explicitly seen in any of the negative samples. In spite of the many false positives, the average position does correlate with the actual face region.

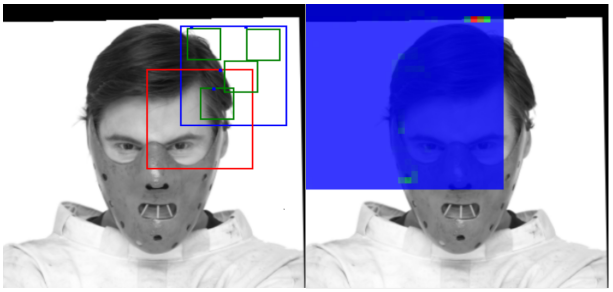


Figure 6.10: a) An example of one of the least accurate localizations using a face CNN combined with an iterative alignment step, resulting in an average error of 384,19% of the inter-ocular distance. b) The CNN response over the image that was used to calculate the face region.

Figure 6.10 depicts the worst localization for this technique. Once again the network responded very well to the horizontal lines at the top of the image, instead of the responses

on the face regions itself. There was one response close to the true face region, which is surprising, considering the high amount of occlusion present in the true face region. The iterative adjustment also caused the localization to deviate from the ideal region even further. This could be because of a very strong response for the "mouth" part in the hair in this image, causing the algorithm to align the parts in order to better match the mouth part. This could be addressed in future work by improving the criteria by which this matching is done.

6.5.1.3 Face part localization using a part CNN

Overall, face-part CNN localization performed much worse than the face CNN detection algorithms, with an average accuracy of 174,84% of the inter-ocular distance. This might be because of a huge increase in false positives produced by this method. This could be expected as the algorithm does attempt to be more lenient to noise or non-face-parts in the image. It is interesting to note that the overall errors with this technique dramatically decreased with only a 3% error rate. This is also expected as the algorithm would generate more region proposals because of the leniency towards noise, meaning that the chance of not locating any faces within an image is greatly reduced.



Figure 6.11: a) An example an accurate localization using a part CNN with an average error of 34,34%. b) A map of responses calculated by the face-part CNN algorithm.

Figure 6.11 depicts the best localization for this technique. Once again it is interesting to note the high number of false positives in the region connected to the black line at the top of the region and the remarkable absence of positive responses in the face region. The lack of positive detections in the face region is likely because of the fact that all the negative samples that where shown to the face-part CNN was sampled from the face region. This would cause the CNN to be very discriminative in that region. This is also supported by the confusion matrix presented in Table 6.2, where the class with the highest number of misclassifications was the "Random" class, making it very likely that non-face parts in the image gets confused for actual face parts. The types of images sampled from the rest of the image, would have been unseen to the CNN and would cause a great deal of confusion. The

average localization does happen to coincide with the ground truth facial region, however, this is not intentional.



Figure 6.12: a) An example of one of the least accurate localizations using a face-part CNN with an average error of 580,85%. b) A map of responses calculated by the part-based detector.

Figure 6.12 depicts the worst localization for this technique. It is once again interesting to note the high number of false positives on the black region at the top of the image. The rest of the positive responses surround the face region, but no strong responses on the face region itself. A possible reason for the high number of proposals in the black region at the top of the image could be due to some of the training data labelled as positive examples of parts is occluded or very dark, causing the face-part CNN to identify these areas as valid face parts. The high error is also due to the large size of the image.

6.5.1.4 Face part localization using a part CNN with iterative alignment

In this section, the effect of iterative alignment on the final localization of the face-part CNN localization method is examined. The overall accuracy dropped by 5,57% and errors increased dramatically bringing it up to 45% of the total number of images, making it the most error-prone of all the algorithms tested so far. One reason for the increase in errors could be due to the initial error in localization and the subsequent part detection step failing to localize any viable points for adjustment. This would not affect the previous algorithm as the spring cost forces some points to be placed based only on geometry.

Figure 6.13 depicts the best localization for this technique. The same affinity towards the horizontal line at the top of the image is observed. The reason this image had the smallest error is because of the small size of the image. The addition of the iterative alignment exacerbated the problem as it had no good landmarks to use to improve the alignment.

Figure 6.14 depicts the worst localization for this technique. The reasons for the inaccurate localization is similar to the causes observed on the results of other methods.



Figure 6.13: a) An example an accurate localization using a face-part CNN with iterative alignment with an average error of 49,86% of the inter-ocular distance. b) A map of responses calculated by the face-part CNN.



Figure 6.14: a) An example an inaccurate localization using a face-part CNN with iterative alignment with an average error of 599,83% of the inter-ocular distance. b) A map of responses calculated by the part-based detector.

6.5.2 Face-part localization

The following images are based on the localization of the face parts given a successful cropping of the face region. Possible explanations for the results is discussed.
Figure 6.15 depicts the best localization for this technique. This image has a frontal view, meaning that the spring-cost would naturally align well with the ground truth positions of the face parts being detected. It is also important to note that the nose location is detected well despite the spring-models expectation of where the point should be located. This is also expected as the nose is one of the parts least likely to be misclassified as demonstrated by the confusion matrix depicted in Table 6.2. This is a good example of the confidence ellipses effectiveness in describing placements of facial feature points.

Figure 6.16 depicts the worst localization for this technique. This image shows some definite occlusion over the left eye, however, the constraints enforced by the spring-model were not strong enough to place the part at the most likely placement. This image also depicts a smile with teeth, and it is possible that the proposed network has not generalized this

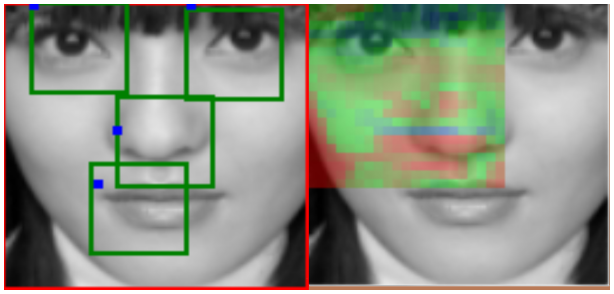


Figure 6.15: a) An example an accurate localization using a face-part detection CNN with an average error of 5,75%. b) The responses of the face-part CNN combined with the spring-cost that was used to calculate the part locations.

feature successfully.

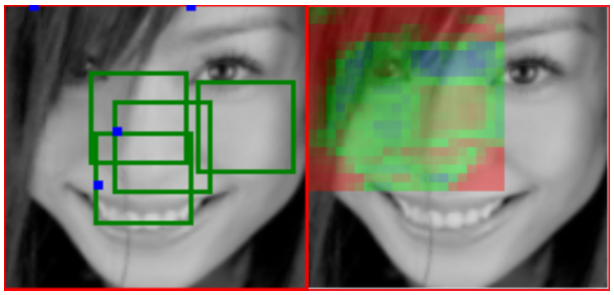


Figure 6.16: a) An example of one of the least accurate localizations using a face-part detection CNN with an average error of 48,11 %. b) The responses of the face-part classifier neural network combined with the spring-cost that was used to calculate the part locations.

6.5.3 Impact of neural network performance on localization

In this section, the impact of neural network training on the face detection responses used for localization is investigated. Face CNNs from different parts in the training process were taken and used to detect face regions on an image in order to get more insight into what the network is responding to as training continues and the network becomes better at discriminating between faces and non-faces. The hope is that over- or under-training of the network could be identified as the reason for the poor performance of the face CNN during detection. In this section the CNN described in Figure 5.15 is used. The loss values used in this section was selected such that relative evenly spaced loss intervals could be sampled from the CNNs persisted during the training of the actual CNN that was used for the experiments. The lowest loss value of the CNNs sampled was that of the CNN that achieved the lowest loss value during actual training. Then 5 other samples are selected

backwards from this loss value in roughly even intervals. The responses are shown in Figure 6.17. When looking at the responses it is clear that in this particular image,the CNN does get more discriminative as training progresses. The network does seem to have an affinity for hard lines. This could be because of the data used as negative samples are too similar or have too much overlap with positive samples of parts. It is also quite interesting that the face region is quickly discarded as a valid face, whilst sections with edges of the face seem to cause the network to respond quite well. From early in the training steps the dark line on the right of the image causes a strong response in the CNN and this affinity remains throughout the training process although it does get less. One possible explanation is that there is an anomaly in the training data that causes the CNN to train to recognize these lines as positive samples.



Figure 6.17: Face region proposals for face CNNs with a loss of: a) 0,29992 b) 0,25329 c) 0,19691 d) 0,15731 e) 0,09852 f) 0,00591

Two aspects of of the training of these networks are likely to blame for the strong responses in incorrect parts of the image. The first is that two different datasets were used for training and testing datasets. Because these two datasets were created using different methods, it is possible that the networks learned some features that were specific to the training dataset that was not present in the dataset that was used for testing. The second, and very likely cause of the strong responses to the dark lines, is the fact that the negative samples for face detection was not taken from normalized images, meaning that the dark lines were not present in the negative samples that the network was presented with. Without that knowledge, it is very likely that the network started to associate dark lines with the edges of faces, which caused a lot of inaccurate localizations.

6.5.4 Impact of occlusion on face and face-part localization

When looking at the combined results in Section 6.4.2 not much is learned about how the system is affected by occlusion in the face region. The COFW dataset has 29 hand-annotated feature points, where not only the position is given, but also whether or not the feature point is occluded. These points were used to distinguish between heavily occluded and lightly occluded images and look at the performance impact of occlusion on the system performance. The COFW has an average of 23% occlusion of all feature points [Burgos-Artizzu et al., 2013]. In this analysis the average error of all images with occlusion

above the average of 23% will be compared to the average error of all the images with below average occlusion. This is done for all 4 proposed techniques, as well as the part localization on pre-cropped faces and can be seen in Table 6.5. The dataset in total had an average occlusion of 23,81% and 100 randomly selected images. In total there where 48 images with light occlusion, and 52 images with heavy occlusion. The lightly occluded images had an average occlusion of 10,41% and the heavily occluded images had an average occlusion of 36,14%.

Method	Light Occlusion	Heavy Occlusion	Average
Face CNN	129,18%	121,49%	125,11%
Face CNN + Iterative Alignment	143,94%	134,53%	139,00%
Face-part CNN	187,14%	163,52%	174,84%
Face-part CNN + Iterative Alignment	182,14%	178,81%	180,41%
Part only localization	17,92%	22,25%	20,30%

Table 6.5: A comparison between the error for images with light occlusion and images with heavy occlusion. Error is measured as a percentage of the distance between the eyes.

From Table 6.5 it is evident that the Face CNN achieved the best results on both heavily occluded and lightly occluded images, however it did seem to perform slightly better on occluded images. It is possible that this is just because of the samples in this dataset. When Iterative alignment was added, there was a reduction in performance on both heavy and lightly occluded images, however it did seem to perform slightly better as occlusion rose. When looking at the face-part CNN based techniques, overall performance was negatively affected however, in comparison to the lightly occluded faces, there was quite an increase in performance as occlusion was higher. This could possibly point to the fact that the face-part CNN based techniques are not discriminative enough when it comes to detecting faces, however, when the number of face parts are reduced, there are fewer areas that are likely to generate false positive detections. Part detection when the face is pre-cropped, performed better on lightly occluded faces than on heavily occluded faces. This is in line with what is expected given that the locations of the occluded face parts have to be estimated by the system.

6.5.5 Impact of rotation on face and face-part localization

Similar to the analysis of the impact of occlusion on the detection accuracy in Section 6.5.4, this section investigates the impact of out-of-pane rotation on the detection accuracy of the described methods. In order to measure this effect, the rotation of the faces had to be calculated because the COFW dataset that was used for testing system performance had no rotation labels. The rotation was calculated as the absolute distance between the center of the two eye centres and the center of the nose. This difference was normalized by

the interocular distance in order to express rotation as the distance of the nose from the face center as a percentage of the interocular distance. This only accounts for rotation in the y-axis, meaning that it describes the amount by which the subject is looking to the left or right. Unfortunately there was no reliable way to estimate by how much the subject is looking up or down using only the available data points. The rotation metric was used to calculate the average rotation of the subjects used during testing. The average rotation as described by the difference in location of the nose points from the center-point between the eyes was calculated as 12,98% of the interocular distance. In total there were 46 images with light rotation, and 54 images with heavy rotation. Images with a low degrees of rotation had an average of 4,62% offset from the face centre while images with high degrees of rotation had an offset of 20,11% from the centre of the face. The average error for all images with low rotation is compared to the average error of all images with high degrees of rotation. This is done for all 4 proposed techniques, as well as the part localization on pre-cropped faces and can be seen in Table 6.6.

Method	Light Rotation	Heavy Rotation	Average
Face CNN	131,08%	114,71%	125,11%
Face CNN + Iterative Alignment	148,04%	123,11%	139,00%
Face-part CNN	172,47%	177,07%	174,84%
Face-part CNN + Iterative Alignment	185,31%	171,40%	180,41%
Part only localization	21,09%	19,63%	20,30%

Table 6.6: A comparison between the error for images with low out-of-plane rotation and images with heavy out-of-plane rotation. Error is measured as a percentage of the distance between the eyes.

Similar to the results shown in Table 6.5, there does not seem to be a strong correlation between the presence of rotation and performance for most of the algorithms that attempt to do face localization. This is likely due to the high error in localization due to other factors as described in Section 6.5.4. When looking at the face-part localization on pre-cropped faces it is shown that the results tend not to correlate to the presence of out-of-plane rotation. This is in contrast to that of the presence of occlusion as seen in Table 6.5 for the same method. This is in contrast with expected performance, however it is possible that this is merely because of the data used and not because of properties of the method itself. This apparent indifference to rotation could be explained by the fact as long as face-parts are visible, the system would have a good chance of accurately localizing these parts given that the geometric features were calculated using the placements of face-parts on many rotated images, as well as the face-part CNN being trained on these images. It is important to note that an average rotation of 12,98% is not extremely high and most face parts should still be visible to the localization algorithm.

6.6 Comparison to other work

In this section our results are compared with that of some of the state-of-the-art methods. All cases where faces are not detected or where face parts in a region could not be located are considered as failures. This differs from the work that we are comparing to, where all localizations with an error of more than some threshold is considered a failure.

The first comparison is with the Adaptive Cascade algorithm by Dong and Wu [2015]. These results are measured on the LFPW dataset which also includes faces in the wild with a small degree of occlusion as well as sharing the same landmark points as COFW, making it a decent comparison to our work in terms of general performance. In their work all localizations with an average error of more than 10% of the inter-ocular distance are counted as failures.

The second work we compare our results with is the work done on Robust Cascaded Pose regression by Burgos-Artizzu et al. [2013]. This work is also tested using the COFW dataset and results are also normalized over inter-ocular distance, making this a very good result to compare our performance to. It is also important to note that any localization with an error of more than 10% of the inter-ocular distance is counted as a failure case.

Another method that the system is compared to is the occlusion coherence method proposed by Ghiasi and Fowlkes [2014]. This work was trained on the LFPW dataset and is also tested on the COFW dataset. It is important to note that the tests are performed on pre-cropped face regions making it more comparable to the Part only localization results presented in this work. The error rate is calculated as any localization greater than 0,1%.

The proposed system is also compared to the work done by Ren et al. [2014] on regressing binary features. This work was also tested on the LFPW dataset, however, no error results are given.

The final comparison that is made is with the work on CNN cascades by Sun et al. [2013]. Their results are also based on their system's performance on the LFPW dataset and considering all detections with an error greater than 10% of the inter-ocular distance a failure.

From Table 6.7 it is quite evident that our proposed system fails at achieving results comparable to that of other state-of-the-art systems, both in average error as well as error rate. It is important to note that across the board performance on the COFW dataset was worse than that of work done on the LFPW dataset. This is because of the challenging nature of the COFW dataset compared to the LFPW dataset [Belhumeur et al., 2013b, Burgos-Artizzu et al., 2013].

Method	Dataset	Average Error	Failures
Face CNN	COFW	125,11 %	14%
Face CNN + Iterative alignment	COFW	139,00 %	19%
Face-part CNN	COFW	174,84 %	3%
Face-part CNN + Iterative alignment	COFW	180,41 %	45%
Part only localization	COFW	20,30 %	0%
Adaptive Cascade [Dong and Wu, 2015]	LFPW	2,215%	2%
RCPR [Burgos-Artizzu et al., 2013]	COFW	8,5 %	2 %
Occlusion coherence [Ghiasi and Fowlkes, 2014]	COFW	7,46%	13,24%
Regressing local binary features [Ren et al., 2014]	LFPW	3,35%	-
CNN cascade [Sun et al., 2013]	LFPW	2%	1%

Table 6.7: Statistics around the results

6.7 Summary

In this chapter, COFW dataset is described. This is followed the preprocessing steps used to prepare the COFW data for the testing of the system. Then the trained final face and face-part CNN architectures are tested on unseen testing data from the COFW dataset. Following CNN testing, the system performance on test data was presented. An investigation into the best and worst performing localizations was made and possible causes for the results are discussed. This included analyses of the impact of CNN performance, occlusion, and out-of-plane rotation. Finally a comparison to other similar methods was made and discussed.

Chapter 7

Conclusion

In conclusion, a system using DPMs with CNN features was proposed with the goal of being robust to various forms of occlusion. A new CNN architecture was proposed for the classification of faces, and a new CNN architecture was proposed that was capable of classifying 4 face parts. An architecture for a face detector that is able to ignore occlusion was proposed, as well a method of statistically modelling the face as a combination of ellipses, each denoting a part with plausible placements relative to the root node. A method for the calculation of spring-cost, allowing for constraints between localized parts to be enforced was proposed. Finally, a simple non-max suppression algorithm was proposed.

The system was trained on normalized images from the HELEN dataset and tested on normalized images from the COFW dataset. The results were documented, explored and compared to that of state-of-the-art methods. The resulting system was outperformed by all comparable methods. This is mostly due to inaccuracies in classifications from the proposed CNNs and the simplicity of the non-max suppression algorithm used. This highlights the dependence on accurately trained CNNs when it is used as both a feature extraction and a classification method in a DPM system. The importance of the use of an effective method of ignoring false positive detections as well as a robust region proposal algorithm was highlighted.

7.1 Future work

In future work, an improved neural net training methodology could be used to improve detection and remove the high rate of false positives seen in this work. This will include ensuring that negative training samples for the face CNN are generated from normalized samples as the normalization process introduces new artefacts that have a dramatic effect on the classification ability of the neural networks. The face-part CNN method would have a good chance of performing better if the face-part CNN is trained on negative parts that are not within the face region, as it is important that the network remains discriminative

and general enough for face localization. Another addition to the face-part CNN face localization algorithm, would be to search over a larger area for each part. This would make it more robust to pose variation and could potentially be accurate enough to allow simultaneous face and face-part localization. The introduction of more complex non-max suppression algorithms and region proposal techniques could also be expected to improve results. Using the standard intersection-over-union non-max suppression that is proven to produce good results would be a better choice, and it would allow for more than one face to be detected in a single image. The advent of new classification techniques such as capsule networks open up new possibilities of applying these new networks to the problem of face alignment [Sabour et al., 2017]. Finally, the use of multi-purpose networks as described by Ranjan et al. [2017] could possibly provide the basis of classifiers that are more accurate using fewer training samples.

Bibliography

T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–2041, 2006.

P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. Localizing parts of faces using a consensus of exemplars. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):2930–2940, 2013a.

P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. Localizing parts of faces using a consensus of exemplars. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(12):2930–2940, 2013b.

Y. Bengio, Y. LeCun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden markov models. In *Advances in neural information processing systems*, pages 937–944, 1994.

Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2559–2566. IEEE, 2010.

X. P. Burgos-Artizzu, P. Perona, and D. P. Robust face landmark estimation under occlusion. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1513–1520. IEEE, 2013.

D. Chen, S. Ren, Y. Wei, X. Cao, and J. Sun. Joint cascade face detection and alignment. In *European Conference on Computer Vision*, pages 109–122. Springer, 2014.

T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *Computer VisionECCV98*, pages 484–498. Springer, 1998.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

J. Cottrill. Students understanding of the concept of chain rule in first year calculus and the relation to their understanding of composition of functions. *Unpublished doctoral dissertation, Purdue University, West Lafayette*, 1999.

G. E. Dahl, T. N. Sainath, and G. E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *European conference on computer vision*, pages 428–441. Springer, 2006.

C. Darwin and P. Prodger. *The expression of the emotions in man and animals*. Oxford University Press, USA, 1872.

H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan. *Neural network design*. Martin Hagan, 2014.

Y. Dong and Y. Wu. Adaptive cascade deep convolutional neural networks for face alignment. *Computer standards & interfaces*, 42:105–112, 2015.

V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

P. Ekman and K. Dachter. Are facial expressions universal?, 2014. URL `http://greatergood.berkeley.edu/article/item/are_facial_expressions_universal`. Accessed: 2015-08-07.

P. Ekman and W. Friesen. Facial action coding system: A technique for the measurement of facial movement. *Palo Alto, CA: Consulting Psychological Press*, 1978.

P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

P. F. Felzenszwalb and D. P. Huttenlocher. Efficient matching of pictorial structures. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 66–73. IEEE, 2000.

P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, 100(1):67–92, 1973.

K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

C. Garcia and M. Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 26(11):1408–1423, 2004.

C. Gershenson. Artificial neural networks for beginners. *arXiv preprint cs/0308031*, 2003.

G. Ghiasi and C. C. Fowlkes. Occlusion coherence: Localizing occluded faces with a hierarchical deformable part model. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

R. Gross, I. Matthews, and S. Baker. Active appearance models with occlusion. *Image and Vision Computing*, 24(6):593–604, 2006.

Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, 2009.

R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

B. Heisele, M. Pontil, et al. Face detection in still gray images. Technical report, Massachusetts institute of technology Cambridge MA Center for Biological and Computational Learning, 2000.

G. E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

L. Huang, A. Shimizu, and H. Kobatake. Robust face detection using gabor filter features. *Pattern Recognition Letters*, 26(11):1641–1649, 2005.

D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

M. S. Islam and S. Auwatanamongkol. A novel feature extraction technique for facial expression recognition. *International Journal of Computer Science Issues*, 10(3):9–14, 2013.

P. Ivan. Active appearance models for face recognition. *Vrije Universiteit Amsterdam, Amsterdam*, 2007.

C. E. Izard. Four systems for emotion activation: Cognitive and noncognitive processes. *Psychological Review*, 100(1):68–90, 1993.

A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

V. Jain and E. Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009*, 2(7):8, 2010.

H. Jiang and E. Learned-Miller. Face detection with the faster r-cnn. In *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on*, pages 650–657. IEEE, 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Q. V. Le et al. A tutorial on deep learning part 2: autoencoders, convolutional neural networks and recurrent neural networks. *Google Brain*, pages 1–20, 2015.

V. Le, J. Brandt, Z. Lin, L. Bourdev, and T. Huang. Interactive facial feature localization. *Computer Vision–ECCV 2012*, pages 679–692, 2012.

Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

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

Y. LeCun, K. Kavukcuoglu, C. Farabet, et al. Convolutional networks and applications in vision. In *ISCAS*, volume 2010, pages 253–256, 2010.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

O. Matan, R. Kiang, C. Stenard, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jackel, and Y. Le Cun. Handwritten character recognition using neural network architectures. In *Proceedings of the 4th USPS advanced technology conference*, pages 1003–1011, 1990.

M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555–559, 2003.

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

D. Meyer and F. T. Wien. Support vector machines. *R News*, 1(3):23–26, 2001.

R. Min, A. Hadid, and J. Dugelay. Efficient detection of occlusion prior to robust face recognition. *Scientific World Journal*, 2014.

A. W. Moore. Support vector machines. *Tutorial. School of Computer Science of the Carnegie Mellon University. Available at http://www.cs.cmu.edu/˜awm/tutorials. [Accessed August 16, 2009]*, 2001.

S. Nass, J. Steuer, and E. R. Tauber, editors. *Computers are social actors*, 1994. ACM.

D. Nguyen, V. Nguyen, M.-T. Tran, and A. Yoshitaka. Deep convolutional neural network in deformable part models for face detection. In *Pacific-Rim Symposium on Image and Video Technology*, pages 669–681. Springer, 2015.

M. A. Nielsen. Neural networks and deep learning. *Determination Press*, 2015.

R. W. Picard. Affective computing. *M.I.T Media Laboratory Perceptual Computing Section Technical Report*, 1995.

R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

S. Ren, X. Cao, Y. Wei, and J. Sun. Face alignment at 3000 fps via regressing local binary features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1685–1692, 2014.

J. Roth, Y. Tong, and X. Liu. Adaptive 3d face reconstruction from unconstrained photo collections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4197–4206, 2016.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

P. Savalle, S. Tsogkas, G. Papandreou, and I. Kokkinos. Deformable part models with cnn features. In *European Conference on Computer Vision, Parts and Attributes Workshop*, 2014.

D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN 2010*, pages 92–101, 2010.

V. Sharma, S. Rai, and A. Dev. A comprehensive study of artificial neural networks. *International Journal of Advanced research in computer science and software engineering*, 2(10), 2012.

M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

V. Spruyt. How to draw an error ellipse representing the covariance matrix? `http://www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-matrix/`, 2014. Accessed: 2018-03-18.

M. Storer, P. M. Roth, M. Urschler, H. Bischof, and J. A. Birchbauer. Active appearance model fitting under occlusion using fast-robust pca. In *VISAPP (1)*, pages 130–137, 2009.

Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3476–3483, 2013.

the Theano Development Team. Convolutional neural networks (lenet). `http://deeplearning.net/tutorial/lenet.html`, 2018. Accessed: 2018-01-20.

A. Thomas. Advantures in machine learning neural networks tutorial a pathway to deep learning. `http://adventuresinmachinelearning.com/neural-networks-tutorial/`, 2017. Accessed: 2018-08-20.

Y. Tian, T. Kanade, and J. F. Cohn. *Facial Expression Recognition*, chapter Chapter 19 - Facial Expxression Recognition, page 487. Springer Science and Business Media, 2011.

D. Triantafyllidou, P. Nousi, and A. Tefas. Fast deep convolutional face detection in the wild exploiting hard sample mining. *Big Data Research*, 2017.

J. Whitehill, M. Bartlett, and J. Movellan. Automatic facial expression recognition for intelligent tutoring systems. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–6. IEEE, 2008.

B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

S. Yang, P. Luo, C.-C. Loy, and X. Tang. From facial parts responses to face detection: A deep learning approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3676–3684, 2015.

A. L. Yuille, D. S. Cohen, and P. W. Hallinan. Feature extraction from faces using deformable templates. In *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR'89., IEEE Computer Society Conference on*, pages 104–109. IEEE, 1989.

K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

W. Zhang, G. Zelinsky, and D. Samaras. Real-time accurate object detection using multiple resolutions. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

L. Zhu, Y. Chen, and A. Yuille. Learning a hierarchical deformable template for rapid deformable object parsing. *IEEE transactions on pattern analysis and machine intelligence*, 32(6):1029–1043, 2010.

X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE, 2012.