Patch-based Pose Invariant Face Feature Classification

NME Mokoena

orcid.org 0000-0002-8815-0390

Dissertation submitted in fulfilment of the requirements for the degree *Master of Science in Computer and Electronic Engineering* at the North-West University

Supervisor:	Prof ASJ Helberg
Co-supervisor:	Dr HD Tsague

Graduation May 2018

Student number: 18047459

NWU®

NORTH-WEST UNIVERSITY
NOORDWES-UNIVERSITEIT
YUNIBESITI YA BOKONE-BOPHIRIMA

Tel:	+27 18 3892328
Fax:	+27 86 6068714
E-mail:	Dave.Kawadza@nwu.ac.za
Internet:	http://www.nwu.ac.za

12 February 2018

Language Editing Certificate

This serves to confirm that I have read and edited the M Sc. Dissertation of Ms NTHABISENG **MOKOENA** (Student number: **18047459**) entitled:

Patch-based Pose Invariant Face Feature Classification

The candidate corrected the language errors identified.

The document presentation is of an acceptable academic and linguistic standard.

Thank you

[illegible]
....

Dr TD Kawadza

Language Editor

Faculty of Agriculture, Science and Technology

Abstract

Real world face recognition systems are now successfully developed to recognise faces in frontal view. One of the most challenging tasks facing state-of-the-art face recognition algorithms is how to handle variations caused by the direction of the face image in terms of angles, that are between the probe and the gallery images. This research work treats the problem caused by variations in pose as a classification problem. We conduct face classification on the FERET database. Firstly, we extract the SIFT features at different scale spaces (σ); by extracting these features at different levels will help us determine which values of (σ) give a better representation of our data. Secondly, we train these features using four machine-learning algorithms: k-Nearest Neighbor (kNN), Support Vector Machine (SVM), decision trees and neural network pattern recognition. The experiments demonstrate that by increasing the blur (σ) parameter, the classification rate decreases.

Keywords: Image processing, Face recognition, Pose invariant, Feature extraction, Pose classification

Acknowledgements

I would like to express my deepest appreciation to The Almighty, for entrusting me with this project, His mercies are new to me every day.

- Greatest appreciation to my supervisor Prof. Albert Helberg, for the support and sound advice he gave me during my studies. His patience and work ethics with students were amazing. God bless Prof dearly.

- I sincerely acknowledge my co-supervisor from the CSIR Mr. Hippolyte Djonon Tsague for his patience, encouragement and eternal cheerfulness. In the same breath, I would like to acknowledge Mr. Rethabile Khutlang and Mr. Johan van der Merwe for their open door policy, great leadership from colleagues.

- I am equally grateful to the DST and the CSIR for awarding me a lifetime opportunity, a gift none will take away. Thank you.

- Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development Program Office

- I thank my colleagues and friends from MDS for their support and words of encouragement, (Mahlatse and Gugu), most of all my sincere appreciation goes to Ofetswe Lebogo, a beautiful mind, great soul, always willing to help. You are with such sound knowledge of coding and willing to share and help all the time. Am grateful to have met you, God bless you dearly.

- Lastly, I would like to thank my family, that has always been my pillar and support system. My mom Dimakatso Mokoena and dad Mohanoe Mokoena, my son Keorapetse 'Kay wa Mommy' (mommy loves you so much), my brother and his wife, together with their two daughters (Orapeleng and Moleboheng). Accept my heartfelt appreciation for your prayers and support. Thank you!

To God be the Glory.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Face recognition of frontal (relating to the front of the face) view images is a well researched area [1, 2, 3, 4], but recognition of face images in a non-frontal (relating to the side of the face) view remains a great challenge for face recognition technology (FRT) [5]. In comparison to other biometrics techniques (technological systems that use biological information of a subject), for example fingerprint recognition, FRT has the advantage of allowing subjects to be scanned without active response. For instance, if a FRT is mounted at the airport, as a means of a surveillance security system, it is expected that good FRT should be able to identify a known impostor even when the subject is uncooperative, i.e. the subject may be trying to hide their face. It would be a difficult task for a FRT to identify a known criminal using surveillance cameras at the airport if half of the subject's face were not shown. This generality of situations and environment brings major problems to a FRT [6].

Recently, researchers in the field of machine learning (ML) and computer vision; have with great effort been developing algorithms that attempt to solve the problem of face recognition in non-frontal view [6]. These algorithms are referred to as pose-invariant face recognition (PIFR) algorithms, i.e. algorithms that solve the difficulty of a FRT to identify a subject who is in a non-frontal view [5]. These algorithms seek to solve the problem of face pose. Throughout this research work and the works of other researchers in FRT [6], "pose" refers to the position of the face in the image in terms of degree angles (°) with respect to a face facing straight, e.g. -22°,+45°. According to [5], variations caused by pose bring about the problems, not limited to:

- Self-occlusion: the rotation of the head results in occlusion, which causes loss of information during recognition.

- Loss of semantic correspondence in 2D face images: This refers to the loss of interpretation of the face image due to changes in position. The position of facial image texture varies non-linearly because of variations caused by pose.

Xiaoyang et al [7] in their survey recognise the importance of the size of a face database (data sample) when solving PIFR and face recognition in general. Collecting data samples (face images) to create a database of faces can be a tedious job, in terms of time (to collect face images), storage and also processing these images. Hence, a lot of FRT algorithms are affected in the representation (e.g. features) and the size of the database, so that in most cases; many of these algorithms experience a great decline in performance when only a few samples per subject are available in the database [7]. Furthermore, having one sample per subject (one image per person) can be a great challenge both in the real world and in theory, because a FRT would have learned only one image of a person, and fail to recognise other images in different face poses. One way to make sure that less storage is used in the database, with less processing time and while maintaining satisfactory accuracy in recognition is to apply modern intelligent tools, such as pattern recognition, machine learning (ML) and computer vision.

Consequently, machine learning (ML) classification is one of the methods used in pursuit of developing PIFR algorithms so that they achieve a better performance rate, in terms of both classifying the direction of the face in the image and face recognition. Classification in statistics and ML is a problem of trying to establish into which set of categories a new instance (observation) falls, given observations whose categories are previously known (training) [8]. A classification rule or a classifier is explained as a function h which can be evaluated for any possible values of x; given the data $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, $h(x)$ will yield a similar classification $\hat{y} = h(x)$. The primary goal of a classification learner is that it should be highly accurate given a particular test data set.

The objective of this study is to implement a classification algorithm, which will classify a received input image into which pose (degree angle) it falls under. The classification model that we implemented classified the degree angle of the face, without performing face recognition.

1.1 Problem Statement and Motivation

The face recognition process would be much easier if users could be asked to stand straight facing the scanner. In that way the probe (input) image alignment would match the gallery (database) image, only if the subject of interest is already captured and stored for verification. However, in computer vision applications such as face recognition, such images may be captured from different viewpoints, and it is difficult to classify these images with existing machine learning algorithms, because the feature points from these (input and gallery) images differ. In our research work, we address the problem of PIFR by classifying under which pose angle the input image falls. The classification is performed using a still (mugshot) face image. Our approach to face classification is similar to [9], but does not include recognition of faces, it seeks to classify the pose that the input image falls under. Similar to [9], the motivation behind implementing a classification model of face images in different poses, is the realisation that pose-invariant face recognition has a wide range of applications in machine learning and computer vision, e.g. airport surveillance.

1.2 Research Questions

This research work seeks to answer the following questions:

1. How can we model a pose-invariant face image?
2. What tools (algorithms) can be used to evaluate the performance of pose-invariant features?
3. What are the conditions, such as illumination, noise and image quality, that can affect the classification (positively or negatively) of pose invariant features?

1.3 Research Goals

1. To implement a classification model that would classify a still input face image according to different poses.
2. To determine a classifier that gives high classification accuracy on patch-based face images.

1.4 Limitations

For the purposes of our study, we assume that the face images are mugshots of still faces, with no form of surgical procedure visible on the faces. Furthermore, we assume that the non-frontal images vary between degree angles of ±22,5°(horizontal) and ±45°(horizontal), to allow us to cut-off (crop) facial areas that we need on the face (nose, mouth and eyes). Additionally, the face images that are going to be used are not totally occluded, e.g. there is no form of clothing on the face, and the facial hair is only limited where one could see the eyes, nose and mouth.

1.5 Research Methodology

The research methodology of this study is based on [9]. A pose classification algorithm was used in their face normalisation model. By classifying whether a given face is in a frontal or non-frontal pose, the performance of their model was improved. In order to meet the objectives of this study, the following outline was followed:

1. Background and literature survey - In this section we investigate and survey state-of-the-art studies in pose-invariant feature extraction methods. The intention of this section is to provide a good understanding of the basement or platform of the research goal.

2. Feature extraction - After a thorough investigation of the existing work in literature, we will look at how to better represent pose invariant features for better classification.

3. Classification models - Our data will then be run through different classification models to determine a better classification of poses.

4. Results - results will be analysed and presented.

1.6 Conclusion and Project Summary

In this chapter we have introduced the concept of pose-invariant face recognition (PIRF), i.e. what is meant by pose with regards to face recognition and face classification. Also, we have discussed the problems that are caused by pose and machine learning classification. In the next chapter we will discuss in-depth the main concepts regarding PIFR and classification.

The remaining parts of this study are organised as follows:

Chapter 2 represents the background of the research work and various components for image classification. The chapter will also represent the works that has been done by other researchers, in relation to the project topic.

Chapter 3 represents the proposed to this project and the algorithms behind the proposed approach.

Chapter 4 presents the implementation and results analysis that were obtained from classification models that were tested.

Chapter 5 presents a summary of the entire research project, conclusion and recommendations for future work.

Chapter 2

Background Theory

The focus of this chapter will be on the main concepts regarding how pattern recognition and ML are used to address database representation and the classification of pose-invariant face images. Furthermore, we will look at the application of these theoretical concepts to pose-invariant face images to increase the value of our research work.

2.1 Introduction

To be able to understand how face images are represented in the database, it is important to first understand the concept of a local feature of an image in image processing. In practice, an image exists in a computer's read-only memory (RAM), as a rectangular grid of units called pixels [10]. In a pixel, brightness and intensity play an important role because they are the cause of distinctiveness between pixels. Intensity is denoted by the value of a pixel. For example, in an eight-bit (8) greyscale image, there are 256 grey levels, meaning any one pixel in a greyscale image can have a value ranging from 0 to 255 intensity. So, brightness is a relative term, the higher the intensity the brighter the pixel. Assuming we have three pixels, X,Y and Z, with brightness values $X = 25$, $Y = 74$ and $Z = 230$, then Z is brighter than X and Y is darker than Z [10]. On the other hand, a true colour image consists of 24 bits, where there are 8 bits each for Red, Green and Blue, i.e. a total of 8 x 3 = 24 bits.

Moreover, a local feature of an image is an image pixel which differs from its close neighbourhood (pixels in the same area) [11]. This local feature is commonly associated with changes in image pixel properties, such as texture, colour, brightness and intensity. A local feature can either be represented as some measurement (e.g. distance) made on an image, and transformed into a descriptor which describes an image as a whole. Or, a local feature can be represented by randomly selected small areas on an image which are called image patches, edges or even points.

By the same token, one might be interested in extracting a local feature because of the following reasons [11]:

- Local features provide a limited set of well localised and identifiable local points. The visual representation of a local feature is not necessarily important, what is important is to accurately extract the same features at a certain location over time.
- Another reason why one might be interested in extracting local features is that there is no need for image segmentation (the process of dividing an image into smaller segments). Local features offer a resilient representation of an image without taking into consideration the actual representation of the features, where the aim is to analyse their image pixel statistics not to match features on an individual basis.
- The third reason why local features are of interest is that they have a certain semantic representation in a limited context of a certain application. For example, corners of the eyes, mouth and nose tip in a face image, offer a limited representation of a face rather than of the whole face.

In the same context, Tinne et al [11] outlines the properties of good features as those which are (not limited to), firstly, repeatable given two images of the same scene (gallery and probe), the amount of features extracted in both images should be equally high for both images. Secondly, features should be local in order to minimize the possibility of occlusion (obstruction of the scene), this will allow simple model approximation of the geometric deformation between the two images. Thirdly, the quality of features extracted matters. The number of features extracted should be acceptable such that they provide a compact representation of the image. Fourthly, good features are invariant to large deformation and can be modelled mathematically if possible. Lastly, the features should be accurately located, with respect to scale and shape.

2.2 Face database, facial feature points detection and feature extraction

The challenge of detecting and extracting features arises when the features which are to be extracted are pose-variant. For instance, a surveillance camera that is mounted at a border control will capture a known criminal differently, simply because the subjects are not instructed to look at the camera when they are at the border control post. The problem will arise when features that are captured by the surveillance camera are matched with those that the law enforcement have in their database. The changes in head pose will bring differences in semantic similarity between the two images, hence it is important to detect and extract pose-invariant features for feature classification. In the following subsections we will discuss the choice of a face database and methods which are used for facial feature points detection (FFPD), and also look at methods used for feature extraction in order to attain pose-invariant features.

2.2.1 Face database

Before images can be properly classified or recognised, one needs a collection of images (data set) that are similar to the subject of the matter. For instance, for face classification at different angles (poses), it is required that images in the same pose as that being classified are pre-stored in the database. The choice of a database to be used for a specific research project should be based mainly on the problem at hand, i.e. whether the problem to be solved is recognising faces of different ages, or classifying different face poses. Again, the choice of a data set should depend on the property to be tested, i.e. how a certain algorithm behaves given a data set. It is recommended in research areas such as ML and pattern recognition to use a standard data set when comparing different algorithms. For face recognition, there are various sets of databases that are available to test different algorithms. Xiaoyang et al [7] analyse and discuss issues concerning face data collection, system evaluation and the influence of a small sample size with respect to FRT. In their work, the authors provide a descriptive direction of research in FRT and their suggested popular databases for testing a given algorithm. Their survey presents work from other researcher which shows that the most commonly used database (amongst others such as the Yale database) for changes in face appearance, and which has also achieved a high success rate in terms of performance rate is the Face Recognition Technology (FERET) database. Figure 2.1 shows a montage of some of the images found in the colour FERET database. The database consists of mugshots of both genders, males and females of different races (Blacks, Whites and Asians). Also,these images are in different face pose, e.g. -22,5°and +22,5°.

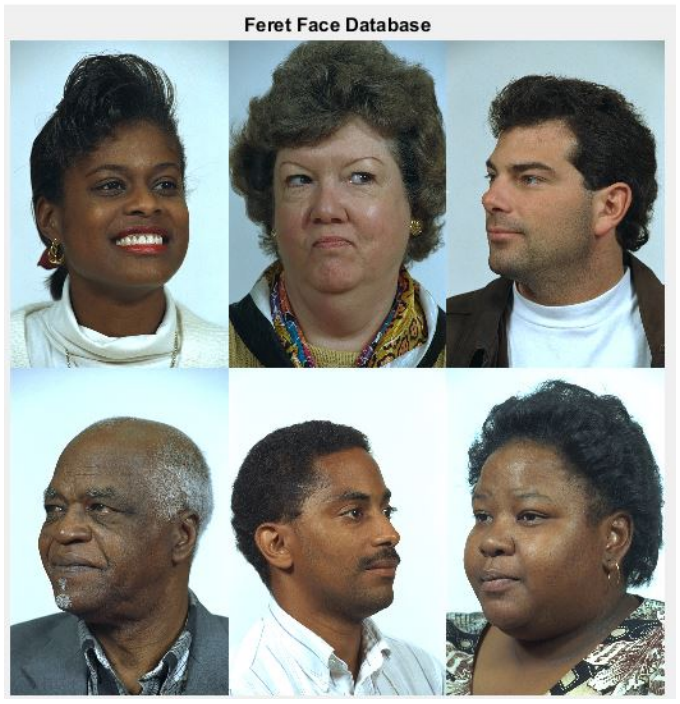


Figure 2.1: Different face poses from The FERET Database.

The Face Recognition Technology (FERET) program is a database whose main aim is to address three issues [12]:

1. Identify future areas of research in pattern recognition and machine learning.
2. To measure algorithm performance, and also
3. To assess the state-of-the-art in face recognition.

The popularity of the FERET database is due to its being a large face database. The database contains a total of 14126 face images, which includes 1199 individuals, and 365 duplicate images. On the other hand, the Yale Database B; which is also one of the popular databases in face recognition has a total of 5760 images, for 10 individuals. The main advantage of the FERET database is that images were acquired in sets of 5 to 11 per subject, under different poses (head direction). Additionally, some of the subjects have facial hair (beard), facial expressions (smile) and also images of a subject taken under different illuminations (lighting conditions). Even though the FERET database has proven to produce good performance results, the images appear to be large in dimension, which requires some pre-processing steps (resizing and segmentation) when performing face image processing.

2.2.2 Feature detection and feature extraction

In some generic face recognition technologies (FRT), the process of facial feature detection and feature extraction are performed simultaneously [3]. Facial feature points detection (FFPD) refers to a process of using machine learning (ML) algorithms, to locate facial components [13]. These facial feature points describe the most often used areas on the face such as the corner of the eyes, the corner of the lips, the tip of the nose and the area around the face. Cootes et al [14] classify facial feature points in three categories, namely:

- points describing parts of the face with application-dependent significance, such as the corners of the lips;
- points describing application-dependent components such as the highest point along the bridge of the nose.
- points that are incorporated from points of the two categories mentioned above, like those under the chin.

Different numbers of facial feature points can be detected for different kinds of face models [13]. These models can consist of 17, 29 or 68 points; where the points selected can be combined to represent the shape of the face image x $= (x_1, ..., x_N, y_1, ..., y_N)^T$Even though more points would suggest significant information, it is time consuming for a model to detect all points. In face recognition literature, there exist a number of state-of-the-art models for feature points detection, which model the shape and appearance of the face image. Such methods include, amongst others, the statistical shape models named the Active Shape Models (ASM) [14]; Active Appearance Model (AAM) [15] which models the appearance variations of a face as a whole by minimising errors that occur from texture synthesis; and regression based models which estimate the shape from the appearance, without learning any shape or appearance model. Other methods which do not fall under any of the methods mentioned are discussed in [3, 14].

Cootes et al [15] present a model of facial appearance by combining a model of shape and texture variations. In order to build their model, they used a database of 400 face images as examples of their model, where each image is manually labelled with 68 corresponding points around the main features, i.e. around the eyes, nose and mouth, also along the face edge. They then apply algorithms which align the sets of chosen points to determine the shape of the face images. After obtaining the appearance and a rough estimate of the position, orientation and scale of the face image, they adjust the model parameters to match the images as closely as possible. The problem with their model is that the number of selected points reduce at a lower image resolution. Also the models were used for variations caused by appearance, not for variations caused by pose.

On the other hand, [16] presents statistical models of appearance that capture both shape and texture. Their models deal with variations caused by pose, on three sample data sets which consist of profile, half-profile and frontal-face images. These images then form the total data set to 234 manually labelled landmark images. Their work demonstrated that even with a small number of appearance models, a face can still be represented from a wide range of viewing angles.

One way of representing a pose-variant face image is by extracting important information from the image called features. These features can either be extracted by trained models from machine learning algorithms, or by manually designing a descriptor that will perform feature extraction. Both these methods of feature extraction help in representing a face image in a compact order, thereby saving database space and reducing processing time.

The success of a classification model depends mainly on the features extracted. Hence, feature extraction becomes an important part of machine learning (ML). The reason behind performing this step is to reduce the input data for further processing, i.e. face classification or recognition. Additionally, the assumption is that the selected features will carry relevant information from the input data-set. There are several methods of extracting viewpoint-invariant features in face images that have been published and well researched such as [11]:

- Multi-scale methods - These methods use points extracted over a range of scales as features.
- Scale-invariant methods - The methods determine both the scale and location of the feature.
- Affine-invariant methods - These are viewed as generalisations of the scale-invariant methods, with a different scaling factor in two orthogonal directions and without preserving angles.

Moreover, algorithms that develop a descriptor manually, focus on designing a representation that is naturally invariant to pose, while accurate to the identity of subject. There are two methods for pose-invariant feature extraction. The first method is by extracting engineered features in order to create a face description [5]. Engineered feature extraction methods are algorithms which re-establish the lost information in the image. The purpose of engineered feature extraction is to retain semantic correspondence (similarities) between images, which is lost due to pose variations. The process of retaining this information can be achieved by locating facial landmarks, i.e. the selected points on the face image that are used to construct features, e.g., corner of the eyes, the center of the eyes, the tip of the nose and the corners of the mouth. Retaining lost information on the face for engineered features can also be obtained by landmark-free algorithms.

Landmarks are defined as face-image regions which contain points of interest where features are to be extracted [5, 11]. For example, locating the corners of the eyes, mouth and the center of the nose, each of these components are called face landmarks. Algorithms which extract pose-invariant features by locating facial landmarks give a better explanation of the area of the image that has been located, thereby resulting in better semantic correspondence (similarities) of images. If the same image scene is located from two images which are taken in different poses (degree of angle), chances are these located areas would not contain the same texture, scale and rotation, which will result in poor comparison of images. Algorithms which locate landmarks in pose-invariant images such as [17] and others discussed in [13], seeks to restore texture, scale and rotation properties of the selected landmarks, for better similarity in images.

Using landmarks to extract pose-invariant features, Biswas et al [17] exploit low-resolution face images from a surveillance camera to correct pose. Before comparing the low-resolution probe face image, in different poses, with high-resolution frontal gallery images, facial landmarks are located, from images using a data set from the Multi-PIE database. The Multi-PIE face database was compiled by researchers at the Carnegie Mellon University. The database contains a data set with high resolution (HR) face images from different view points. Seven facial landmarks are selected, viz; the corners of the eyes, the tip of the nose and the corners of the lips. These landmarks are located by using an approach called tensor analysis, which estimates the facial landmarks and pose of the face image. These landmarks are then used to represent a face, by extracting scale-invariant descriptors discussed in the next section. Thereafter, these descriptors are combined to form one global descriptor that describes a face image. A transformation learning is performed using multidimensional scaling (MDS). The central idea of applying a transformation-based learning is to start with a simple solution to the problem, and apply transformation at each step. The largest benefit is selected to solve the problem. The learning stops when the selected transformation does not modify the data anymore, or when there are no more transformations left to be applied to the problem. MDS transforms both low-resolution probe image and high-resolution gallery image to the same image space for better comparison. In order to learn the desired transformation, the authors in [17] use the iterative majorisation algorithm, which is an optimisation method that finds the maxima or minima of a function. In their method they did not use any pose classification method, and obtained a recognition rate of 96,1% for face images kept fixed at 5°(degree angle).

On the other hand, still using high-resolution (HR) information (images with good visual information) based on pore-scale facial features for face verification, Li et al [18] proposed a landmark free method which is invariant to pose and alignment for face verification. Their method uses only one frontal image per person in the Gallery. Instead of using landmarks to detect facial components such as the eyes and nose, they use a method which detects blob-like regions (image regions that differ in image properties) called pore-scale facial features. The motivation for their work comes from a biological viewpoint, namely that different people should have similar skin quality of facial pores. These regions are likely to include fine wrinkles, hair and pores. From these regions, they extract scale-invariant face-pore keypoints. To determine feature matching of the two face images, they measure the Euclidean distance between their corresponding descriptors. Other methods include the use of learning-based features, such as [19, 20], where features are extracted using pre-trained machine learning models. These are linear models based on multi-view subspace learning [20, 21].

Extracting features for better semantic correspondence

To extract and match features across different images is a common problem in pattern recognition and computer vision. It becomes simpler when extracting features of the same scene in the same scale and rotation, but more difficult when extracting matching features of the same scene from images which differ in scale and rotation [5]. In recent years, there has been a huge interest in extracting multi-scale sampling such as Local Binary Patterns (LBP) and scale-invariant Feature Transform (SIFT). LBP and SIFT are engineered features that are largely extracted for better semantic correspondence of face images.

Lowe's [22, 23] method of feature extraction named SIFT, exhibits a powerful detection and recognition rate for objects. The main advantage of this algorithm is its accuracy, scale and rotation invariance. SIFT based descriptors have been proven to do well as local descriptors in terms of scale and rotation of the scene as well as illumination. Even though feature extraction methods such as PCA-SIFT and SURF [24] out-perform SIFT in terms of processing time (given the same parameters), SIFT has been proven to find most matches of keypoints [24].

SIFT features are not only invariant to scale but also to changes in viewpoint, rotation and illumination. For example, take an image of a person's face in a straight position; meaning, facing straight towards the camera as the original image. Then change the same image and tilt the head to 25,5°and change the image size, SIFT is able to extract and compare (match) the same features, hence scale-invariant. SIFT was first introduced by Lowe [22] as a general object-recognition algorithm. The algorithm is now widely used in face recognition to extract features that are invariant to pose. Once these kinds of features are extracted, a face can be represented in a compact descriptor which is invariant to not only scale, rotation, pose and illumination but also affine, where affine means everything that is related to the geometry of affine spaces. An example of an affine property is the average area of a random triangle chosen inside a given triangle. In the next section we will discuss in details how SIFT features are extracted and how other researchers use this algorithm to extract pose-invariant features for face classification.

2.3 SIFT

The author of the SIFT [22] algorithm illustrates the four computation stages as: scale-space extrema detection, key points localisation, orientation assignment and keypoint descriptors. For better understanding of how to extract SIFT keypoints we will divide the stages into seven main processes (which will include sub-processes), namely, creating scale space, approximating the Laplacian of Gaussian (LoG), locating keypoints, removing unreliable keypoints, keypoint orientation assignment and formulate keypoints descriptors. The algorithm is patented and can only be used for academic purposes [22, 25].

2.3.1 Creating a scale space

Generally, in image processing, scale spaces are created by generating progressively blurred out images. This is done in order to identify important information on the image (edges or corners). Blurring (blur), also called smoothing [26], is an image processing operation which is commonly used to remove noise (unwanted information) from an image. To perform a blurring operation, one needs to apply a filter, i.e. a function which modifies the pixels in an image based on some function of a local neighbourhood of each pixel. Most common filters are linear:

$$g(i,j) = \sum_{k,l} f(i+k, j+j+l)h(k,l) \tag{2.1}$$

where an output pixel's value $g(l,j)$ is determine by a weighted sum of input pixel values $f(i+k, j+l)$ and $h(k,l)$ are the coefficients of a filter, also called kernel [26]. Gaussian blur has been proven to be an effective blur estimator to remove noise [26], without introducing additional false details. A Gaussian filter alone, is able to reduce contrast and blur points at which the image brightness changes sharply (edges). In order to identify edges, two Gaussian filters can be used, and subtracted. This is the method that the SIFT algorithm uses to create a scale space.

The SIFT algorithm performs scale spaces by taking an image, generating progressive blur, then re-sizing the image to half of the original size. This step is repeated according to the number of octaves (images of the same size but different scales) selected. For an example, if one chooses three octaves, the image selected will be blurred and re-sized on three different scale spaces, this is in order to identify edges at different scale spaces. Figure 2.2 shows an image from the FERET database before creating a scale space, while Figure 2.3 depicts the same face image after applying the SIFT algorithm on 3 octaves and 3 levels, the face image loses detail at each level, and gets blurry. The number of octaves and scales entirely depends on the size of the image. However, Lowe [22] suggested that four octaves and five levels are the ideal for the original SIFT algorithm. Having explained the logical part of the algorithm, in the next section we will discuss the mathematical part of SIFT.

Figure 2.2: FERET image.

Scale space mathematics

We learn that the scale space function is formed by a particular expression applied to each pixel. This function is a convolution of a variable scale Gaussian, $G(x, y, \sigma)$, for an input image $I(x, y)$ [22, 25].

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y). \tag{2.2}$$

where

$$G(x, y, \sigma) = \frac{1}{2}\pi\sigma^{2e^{-(x^2+y^2)/(2\sigma^2)}}. \tag{2.3}$$

is the actual Gaussian blur operator in two dimension. Here,

- L represents a blurred image
- G is a Gaussian blur operator
- I is an image
- (x,y) the location coordinates on the image
- σ is the scale parameter, the value that represents the amount of blur. The larger the value, the greater the blur, which may result in edges not identified correctly because of the level of blur.
- * denotes the convolution operation in (x,y) that determines the Gaussian blur.

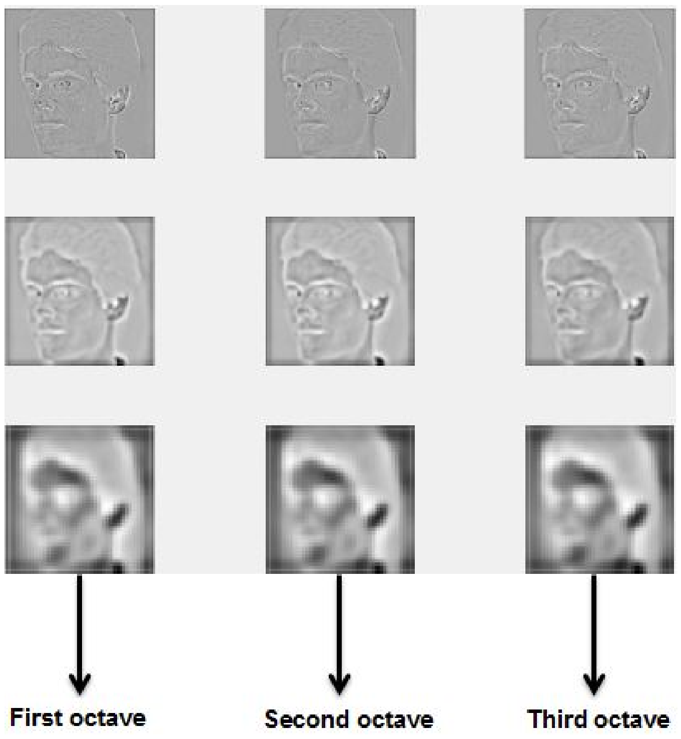


Figure 2.3: Three SIFT octaves at level three.

Laplacian of Gaussian (LoG)

After progressively creating blur images to create the scale space, the following step is to use the blurred images to generate another set of images, using a Difference of Gaussian (DoG) function. This set of images create a better platform to locate keypoints. Also, DoG is used for computational efficiency, i.e. to efficiently identify stable keypoints so as to estimate the normalised Laplacian, $\sigma^2\nabla^2$ with σ^2 factor for automatic scale selection [23, 25]. Theoretically, the Laplacian of Gaussian operation takes an image, slightly blurs it, and calculates the second-order derivative on the LoG images. The process is done so as to locate edges and corners on the image. The normalised Laplacian function after convolution is represented by:

$$O(x, y, \sigma) = \sigma^2\nabla^2 G * I(x, y). \tag{2.4}$$

where O is the normalised image and $\sigma^2\nabla^2 G$ represent the scale-invariant Laplacian of Gaussian. The scale space is used to efficiently generate Laplacian of Gaussian, by calculating the difference between two successive scale spaces. Figure 2.4 depicts how to get the scale space of images that are combined with Gaussian operators. The left side of the figure shows how two Gaussian successive images are selected, and subtracted from each other, and the following successive pair is subtracted and so on.

These Gaussian images produce the DoG, which then creates Laplacian of Gaussian approximations which are scale-invariant[23, 25].

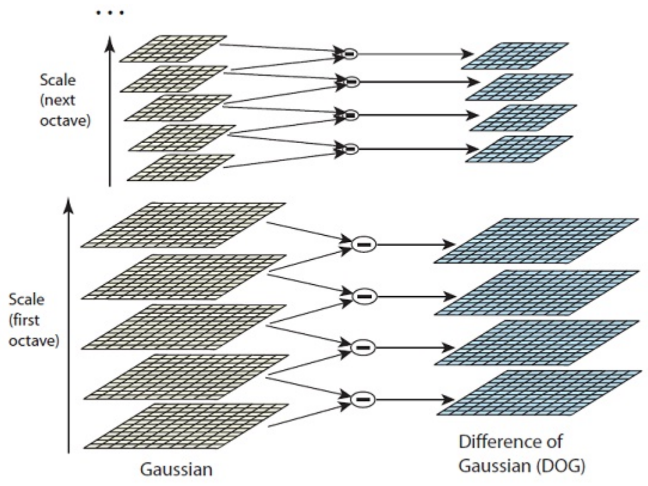


Figure 2.4: Laplacian of Gaussian (LoG) [23, 25].

2.3.2 Locating keypoints

Finding keypoints is divided into two parts in the SIFT algorithm, namely, locating maxima and minima in DoG images and locating subpixel maxima and minima. The maxima/minima of DoG of images are detected by comparing a pixel to its neighbours at the current scale, the scale below and the scale above it. An approximated sample point is selected if the scale is smaller than the neighbours or if the scale is larger than the neighbours [23]. The maxima and minima of scales are 'approximated' because they are not always located exactly on a pixel, they can be located between the pixels, and the information between the pixels cannot be accessed, hence the location of the subpixel. Mathematically, subpixels are generated as [25]:

$$D(x) = D + \frac{\partial D^T}{\partial x}\frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{2.5}$$

where D and its derivatives are evaluated at a sample point $x = (x, y, \sigma)^T$. The above is the Taylor series (expansion of a function into an infinite sum of terms) of the image calculated at the approximated keypoint. Upon solving the equation, we obtain the location of subpixel keypoints, which increases the chances of algorithm stability as well as the chances of matching images.

2.3.3 Removing unreliable keypoints

Some of the keypoints selected above are of no use, viz., keypoints that lie at the edge of the image and those which are low in contrast. The solution is to discard them. Firstly, the low contrast keypoints are removed by the magnitude of the intensity. That is, if the magnitude of the current pixel's intensity (of the DoG image) is less than a certain threshold, then the keypoint is rejected. Secondly, removing keypoints which lie along the edges requires calculating two gradients at a given keypoint. As such, the image around which keypoints lie can either be a corner (meaning both gradients will be big), or an edge (where the perpendicular gradient will be big and the gradient along the edge will be small), or the image can be a flat region where, both gradients are small. Corners make good keypoints, so the idea is to extract corners. To check if a point is a corner or not, Lowe [23, 25] computes something similar to Harris corner detector [27], the Hessian matrix at a given location and scale. Efficiency here is increased by calculating the ration of two values [23, 25].

$$Tr(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta \tag{2.6}$$

$$Det(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \tag{2.7}$$

where H is a 2 x 2 Hessian matrix that computes the principal curvatures (maximum and minimum of the normal curvature at a given point on a surface) at the location and scale of a keypoint, α the eigenvalues with the largest magnitude and β gives the eigenvalues with the smaller magnitude.

2.4 Keypoint orientation

Now that strong keypoints have been selected, and they have been tested to be scale-invariant, the next step is to assign an orientation to these keypoints. The main idea of this stage is to collect the magnitude and the gradient direction of each keypoint. Therefore, the size of the orientation region depends on the scale, where orientation region is the region on the image that provides rotation invariance. The bigger the scale the bigger the orientation region [25]. Let L be the Gaussian smooth image, where $m(x, y)$ denotes the magnitude and $\theta(x, y)$ the orientation of each image sample $L(x, y)$. Then, orientation and gradient magnitude are calculated using:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}. \quad (2.8)$$

$$\theta(x,y) = \tan^{-}1((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))). \quad (2.9)$$

2.5 Generating keypoint descriptor

The idea behind this step is to generate a unique description for each keypoint, which is easy to calculate and to compare to other keypoints. The first step in constructing a keypoint is to create a 16 x 16 window around the keypoint, then further divide this window into sixteen 4 x 4 windows. Figure 2.5 shows the second step, which is a 4 x 4 window in which gradient and magnitude are calculated. Then from this 4 x 4 window a histogram of 8 bins is generated. Each bin corresponds to 0-44°, 45-89°and so on [23]. Finally, the gradient orientation of 4 x 4 block is put into the 8 bins to create a normalised 128 dimension descriptor.

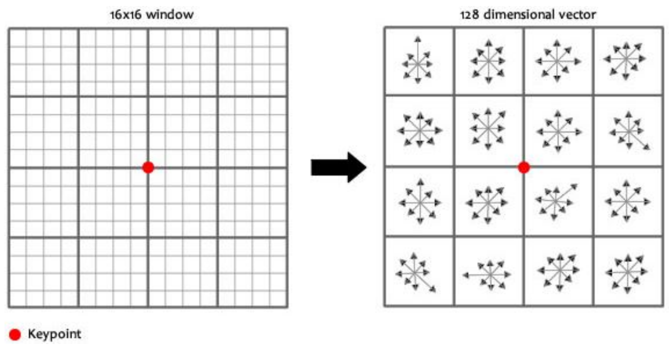


Figure 2.5: Image gradient and keypoint descriptor [23, 25].

Our view from the above discussed is that the SIFT algorithm is computationally expensive, some of the calculations are intense, such as creating scale space and locating keypoints. Nonetheless, the algorithm has been proven to be scale, orientation and viewpoint-invariant.

2.6 Supervised machine learning

Supervised machine learning involves classification algorithms that reason from given class labels (corresponding outputs), to produce general hypotheses in order to make predictions on future instances [8]. The resulting classifier from these supervised methods is then tested by assigning class labels, where the values of the predictor features are known, but the class label values are unknown. The classifier's performance evaluation is most of the time based on the prediction accuracy [8, 28]. Prediction accuracy is calculated by the percentage of correct predictions divided by the total number of predictions. Supervised classifiers can be trained using one of three techniques, namely:

- Using all data for training or no validation method, then compute the error rate on the same data. This method of validation can be computationally expensive, but can come in handy when the most accurate error rate is required. Also, the estimated error rate for this technique can be unrealistically low.
- Another technique is to split the training set into two portions, two thirds are used for training and the other third will be used for testing and performance estimation.
- In cross validation, the training set is divided into equal portions and mutually exclusive subsets. For each subset, the classifier is then trained on the union of all the other subsets. The error of the classifier is then estimated by the average error rate of each subset.

2.6.1 Supervised machine learning algorithm selection

The option of choosing a classifier is mainly based on the prediction accuracy, which is explained as the percentage of correct predictions divided by the total number of predictions [8]. If a supervised learning algorithm's error rate evaluation is unsatisfactory, a number of factors that are affecting the error rate must be investigated, possibly [8]:

- A large training set is needed;
- the selected algorithm is unsuitable for the kind of data used;
- parameter tuning is needed;
- the dimensionality of the problem is too high.

There are two ways in which one may select a supervised ML algorithm: Firstly, if there is adequate data supply, a number of training sets of size N can be sampled, then run the selected algorithms on each of the training sets. Thereafter estimate the difference in accuracy for each classifier on a large data set, such that the mean(average) total of these differences is an estimate of the expected difference in generalization error across all possible training sets of size N, and the variance is an estimate of the classifier in the total set. Secondly, in common cases, a supervised ML algorithm can be selected by performing statistical comparison of the accuracies of the trained classifiers on a selected data set [8].

One of the disadvantages of supervised ML algorithms is that these techniques are costly and time consuming in terms of data labelling [29]. However, the principal advantage of these techniques is that an operator can detect errors and correct them where necessary.

The notion behind Support Vector Machines (SVM's) is the distance from the decision surface to the closest data point which is known as a margin, i.e. two data classes are separated by a hyperplane. An upper bound of the expected generalisation of data has been proven to maximise the margin, also creating the largest possible distance between the separating hyperplane and the the instance on both sides of the plane [8]. Figure 2.6 shows that the function of the SVM is to search for a decision surface that is maximally far away from the data points. The margin of the classifier is determined by the distance from the decision surface to the closest point. The decision function of the SVM is defined by a subset (data points) of the data which defines the location of the separator. The six red and blue data points which are on the separator are called support vectors, the rest of the data points take no part in determining the decision surface [8]. It is for this reason that we chose the SVM as one of the supervised method for this research work.

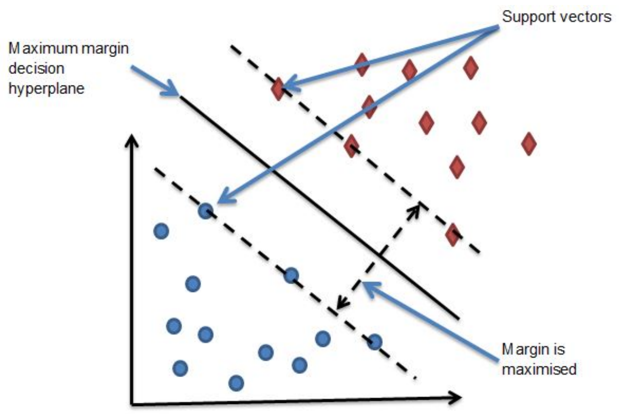


Figure 2.6: Support Vectors and margin for a simple problem.

Furthermore, the training data set is said to be linearly separable when a pair $(\mathbf{w}, b)$ exists such that [8]:

$$\begin{aligned} \mathbf{w}^{\intercal}\mathbf{x}_i + b \geq 1, \qquad &\textbf{for all} \qquad \mathbf{x}_i \in \rho \\ \mathbf{w}^{\intercal}\mathbf{x}_i + b \leq -1, \qquad &\textbf{for all} \qquad \mathbf{x}_i \in \nu \end{aligned} \tag{2.10}$$

where $\mathbf{w}$ is the weight vector (quantity with magnitude and direction), b is the bias (decision hyperplane) or $-b$ the threshold and x_i is the i-th component of x.

Moreover, SVM's are able to learn tasks where the number of features is larger than the number of training instances. Thus the complexity of the SVM algorithm is unaffected by the number of features encountered in the training data [8].

2.6.2 SVM kernel functions

Most of the real-world problems consist of non-separable data, for which there is no hyperplane that can successfully separate the negative from the positive data [29]. Kernel functions are a class of functions which are responsible for solving such problems. These classes of functions map the input data to a higher-dimensional space which is called the transformed feature space. The kernel functions calculate the inner products directly from the feature space. Selecting a kernel function is a crucial step, as it defines the transformed feature space into which the training set will be classified. The SVM will still operate accurately as long as the kernel function is working properly [8]. The process of selecting a kernel setting is closely similar to choosing hidden nodes in neural networks, which we will look at in the succeeding sections. Some of the kernels are Gaussian (also known as radial basis) functions and polynomial functions.

2.7 k-Nearest neighbour (k-NN)

k-Nearest Neighbor (k-NN) also called lazy learning, is instant-based supervised learning. The instant-based algorithms are the kinds of learning methods which delay the process of generalisation until classification is performed. Informally, k-NN is defined as an instant-based learning (IBL) algorithm that stores the entire training set into memory, such that when a test set is presented, k-NN selects the k nearest training instances (data points) to the particular test instance. From these k nearest instance that have been selected, k-NN predicts the dominating class of the k instances as the class which falls under the test instance [30]. In other words, the instance within the data set will closely be approximated to other instances with similar properties [8, 31]. The distance is determined by a distance metric. Table 2.1 shows some of kNN distance metrics:

Euclidean: $D(x,y) = \left(\sum_{i=1}^{m} \lvert x_i - y_i \rvert^r \right)^{1/r}$
Camberra: $D(x,y) = \sum_{i=1}^{m} \frac{\lvert x_i - y_i \rvert}{\lvert x_i + y_i \rvert}$
Manhattan: $D(x,y) = \sum_{i=1}^{m} \lvert x_i - y_i \rvert$

Table 2.1: Approaches for distance metrics [8].

where r is 1 or 2, and where $\mathbf{x} = (x_1, x_2, ..., x_m)$ and $\mathbf{y} = (y_1, y_2, ..., y_m)$ are vectors. Some of the distance metrics that are used in k-NN are tabulated in Table 2.1. The sole purpose of a distance metric is to minimise the distance between similar classes and to maximise the distance between different classes [8]. Moreover, the choice of k affects the performance of a k-NN algorithm, since incorrect classification of query instances is possible because of the following reasons [8]:

- The region defining the class or some part of the class is so small, that the instance of that particular class overpowers the region. This problem can be solved by choosing a smaller k.
- Noise present on the query instance also causes misclassification, because the noisy instance overpowers the information to be classified. A higher k can solve this problem.

To support the above two facts, Okamoto and Yugami [30] present their analyses using instance-based learning (IBL) algorithms. The authors used a k-NN classifier to learn the algorithm's behaviour when handling noise. The tests were done on three different types of noise, namely, irrelevant attribute noise, relevant attribute noise and class attribute noise. These attributes are the information sources that the quality of data can be classified under. Where relevant and irrelevant attributes noise are Boolean-attributes noise. Their analyses were presented on both noisy and no noise domains in order to analyse the learning behaviour of the k-NN algorithm. In their analyses, they deal with m-of-n concepts, (where m is threshold value in target concepts and n is number of relevant attributes), whose positives are defined by having m or more than n relevant attributes. They also include the size of training set, the number of irrelevant and relevant attributes, the probability x of attribute, the threshold value and the value of k. In order to analyse the effects of noise, they plot the predicted behaviour of k-NN in each domain. The effect of each domain characteristic on the expected accuracy of k-NN was predicted. In addition the required number of instances to achieve a particular accuracy was also predicted. Lastly they presented the most favourable value of k against the training set. Their study shows that the expected accuracy of the k-NN algorithm noticeably decreases as each value of k increases where noise is present.

2.8 Decision trees

Decision trees are supervised classifiers that learn simple decision rules from the data points, in order to predict the value of the target. The goal of these models is to sort instances based on features for classification [8, 32]. These learning methods select which variables are important for classification instances, as well as indicate to which class a particular instance belong.

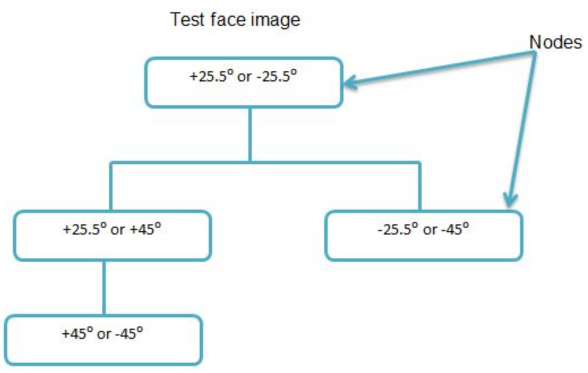


Figure 2.7: Decision tree classifying from a set of attributes.

Figure 2.7 is a decision tree that classifies from a set of attributes (poses). Each level on the decision tree divides the data according to different specified attributes. The main aim of a decision tree is to achieve a perfect classification score with the minimal number of decisions to be made. Therefore, the feature that best classifies the training data is said to be the root node of the tree [8]. Even though there is no one method that is known to be best in finding the feature that best divides data points, information gain is proven to be one of the popular methods that is used to find the feature that best divides the data points [8]. In machine learning the concept of information gain is used to define the measure of impurity in a data set. Information gain increases with the average purity of a particular subset, i.e., it involves partitioning data into subsets which contain instances of similar values. Further advantages of decision trees are:

- Decision trees are simple to understand.
- They are able to handle categorical and numerical data.
- It is possible to validate a model of a decision tree using statistical data.
- Decision trees require little data preparation.

A decision tree can be interpreted into a set of different rules. This is achieved by creating a separate rule for each path. Rules are created from the root to the leaf of the tree [8]. In classification, each class is represented by a disjunctive normal form (DNF). A DNF means, any Boolean condition that can be used to classify the data and can be expressed in $\vee$ "or" or $\wedge$ "and". Therefore, a k-DNF expression is of the form: $(X_1 \wedge X_2 \wedge ... \wedge X_2 n) \vee (X_{m+1} \wedge X_{n+2} \wedge ... X_{2n}) \vee ... \vee (X_{(k-1)^{n+1}} \wedge X_{(k-1)^{n+2}} \wedge ... \wedge X_{kn})$, where k represents the number of disjunctions (joining of two statements with the connector $\vee$ "OR"), n the number of conjunctions (joining of two statements with the connector $\wedge$ "AND")in every disjunction and X_n is defined over alphabet $X_1, X_2, ..., X_j \cup \sim X_1, \sim X_2, ..., \sim X_j$ [8]. The goal of learning a particular set of rules is to build the smallest rule set that matches the training data.

2.9 Neural Networks

Neural Networks (NN) machine learning algorithms are based on the notion of perceptrons. A perceptron can be seen as a computerised machine, which is built to represent a human brain in order to recognise objects. They are an attempt of modelling the data capability of a nervous system [33]. Figure 2.8 depicts a single layered artificial neuron model. If x_1 up to x_n are input feature values, w_1 up to w_n are prediction vectors or connection weights, and t is the adjustable threshold, then the sum of weighted inputs is computed using perceptron:

$$\begin{aligned} if \sum_i w_i x_i \geq t \quad & \text{then} \quad y = 1 \\ else \quad (if \sum_i w_i x_i < t) \quad & \text{then} \quad y = 0 \end{aligned} \tag{2.11}$$

The common way used for a single-layered perceptron algorithm to learn a linearly separable pattern, is to run the algorithm over and over through a given training set, until it finds the prediction vector which is correct on all of the training set. The prediction rule will then be used to predict the test

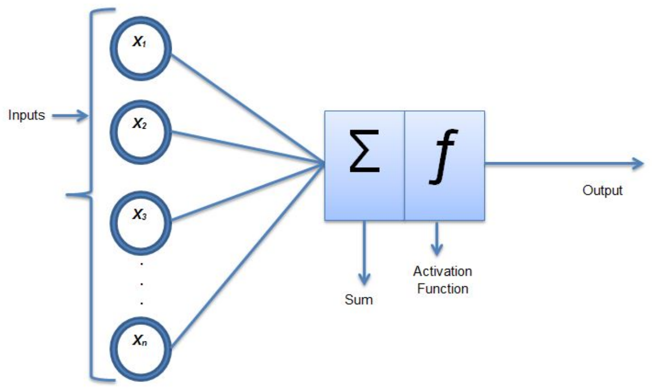


Figure 2.8: Single layered artificial neuron model.

set [8]. However, if the instances are not linearly separable, Artificial Neural Networks (ANN) are used to try and solve the problem of inseparability. ANN also known as multilayered perceptrons, are neural networks of a large number of neurons (units) that are joined in a pattern of connection. These neurons are divided into [8, 34]:

- input nodes - whose function is to receive information that is to be processed.
- output nodes - these are neurons which are the results of processing.
- hidden nodes - these are neurons in between input and output.

2.9.1 Feed-forward artificial neural network

The feed-forward ANN is a multilayered perceptron which allows the signals to travel from input to output [8]. Figure 2.9 illustrates a feed-forward ANN where input-output mapping is determined by training a network on a set of paired data. The connection weights between neurons are fixed, then the network is used to determine classification of new instances [8, 34]. Mathematically, let the activation function (output) be a^i_j of the j^{th} neuron in the i^{th} layer, where the input vector is determined by the j^{th} element of a^1_j.

Moreover, to determine the activation value at all the output neurons, the signal travels through the network during the classification process [8]. Every input node consists of an activation value, which represents a feature external to the net. These inputs then send the activation value to every hidden node which is connected, where each of the hidden nodes will calculate its own activation value. Then the signal is passed on to the output. The activation value for each receiving node is

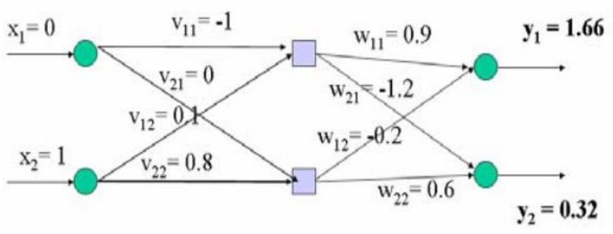


Figure 2.9: Feed-forward ANN [8].

calculated according to its activation function. Thereafter, the function calculates the sum of all the contributions of the sending nodes, where the contribution of a node is defined as the weight of the connection between the sending and the receiving node x the sending nodes activation value [8]:

The sum is further modified by adjusting the activation sum to a value between 0 and 1, and/or setting the activation value to 0, except if a threshold is reached for that particular sum.

Additionally, equation 2.12 is used to establish a relation between the next layer's input and its previous layer [8]:

$$a^i_j = \sigma\left(\sum_k (w^i_{jk}.a^{i-1}_k) + b^i_j\right) \tag{2.12}$$

where:

- σ is the activation function,
- w^i_{jk} is the weight from the k^{th} neuron in the $(i-1)^{th}$ layer to the j^{th} neuron in the i^{th} layer,
- b^i_j represents the bias of the j^{th} neuron of the i^{th},
- a^i_j is the activation value of the i^{th} layer.

In order to determine how well the network performs with respect to its given training samples and expected inputs a cost function is used. The cost function C is a single-valued number and not a vector, it is of the form:

$$C(W, B, S^r, E^r) \tag{2.13}$$

where W is the neural network weights, B is the network's biases, S^r is the network's input of a single training sample and E^r is the desired output of the training sample.

2.10 Performance evaluation metrics

There exist different types of metrics that are implemented in order to evaluate algorithm performance. In this section we look at proposed methods for evaluation of performance of the classification models to be trained.

2.10.1 Receiver Operating Characteristic (ROC) curve

The Receiver Operating Characteristic (ROC) curve was initially developed by an engineer during the World War II; its main purpose was to detect enemy objects in the battlefields [35]. Over the years, this method has been widely implemented in fields such as data-mining and machine learning, as a graphical representation of the performance of a particular classification algorithm. The aim of the ROC curve is to [35, 36].

- Compare the efficiency of two or more classification algorithms;
- Study the inter-observer variability when two or more observers measure the same continuous instances;
- Find an optimal cut-off point to least misclassify the two group subjects, and
- Evaluate the discriminatory ability of a continuous marker to correctly assign into a two group classification.

2.10.2 Confusion Matrix

Confusion Matrix is one of the evaluation measures that can be extracted from a ROC curve. The confusion matrix is the evaluation measure for binary classification problem, i.e. a classification problem which has two classes, positive or negative. Positive and negative classes can be further characterised as true positive, false negative and false positive, true negative respectively.

Table 2.2 tabulates a confusion matrix with the following evaluation measures:

	Predicted Class	
True Class	Positive	Negative
Positive	True Positive	False Negative
Negative	False Positive	True Negative

Table 2.2: Confusion Matrix.

- True positive (TP) - These are instances correctly predicted in relation to the positive class.
- False negative (FN) - Instances predicted as negative, whereas their true class is positive.
- False positive (FP) - Instances predicted as positive, when they are actually of the negative class.
- True negative (TN) - Instances predicted correctly as negative class.

$$
\begin{aligned}
Se &= \frac{|TP|}{|TP| + |FN|} \\
Spe &= \frac{|TN|}{|FP| + |TN|}
\end{aligned}
\tag{2.14}
$$

where Se and Spe are sensitivity and specificity of a classification model respectively. These are statistical measures in ML models to determine the usefulness of a test [35]. In terms of face pose, Se is the probability that a subject's face image will be correctly classified under its rightful pose class, and Sp is the probability that an incorrect face pose will be truly classified as incorrect based on the trained classes.

2.11 Prior art

It has been proven that variation caused by pose forms one of the factors that affect the full functionality of an FRT [3]. In fact Abiantun et al [37] in their work, give as reason that the problem of pose remains overlooked in real-world applications. This section looks at some of the research work which attempts to learn pose-invariant face features, in order to achieve the full functionality of an FRT system [38]. The motivation behind the selection of prior art in this section is to learn about the methods that are used in order to solve the problem of pose. Additionally, to learn the different degrees of angle (°) that pose variation images have been tested on, the different databases that are used to solve variations caused by pose, and also how pose-invariant images are being represented in different methods. Yan et al [39] proposed a novel learning framework which uses Multi-task Learning approach (MTL). MTL approaches in classification/regression, aims at solving multiple task at the same time. These methods simultaneously learn classification models for a given task, for example methods which classify head pose. For a given head pose (45°), their method learns appearance across partitions which operates on a grid. Additionally, it also learns partition-specific appearance variations. To select the appropriate partitioning of the head pose, they use two graphs from a trained appearance similarity model, one which uses grid partitions based on camera geometry and the other which uses head pose classes, to efficiently cluster appearance-wise grid partitions. The difference of this approach from our approach is that [39] employs the MTL head-pose classification method under target motion.

On the other hand, one study has shown that pose-invariant features can be achieved by Principal Components. Kumar et al [40] in their study use Principal Components as features. Taking a large set of features, Principal Components involve a smaller number of dissimilar features. They use the eigen vectors of the covariance matrix as their basis vectors for these Principal Components. In order for them to learn similarities, learn dissimilarities and reduce the dimensionality of these Principal Components features, they make use of PCA. Furthermore, the significance of a feature is determined by the eigen vectors that have the most energy as interpreted from their corresponding eigen values. The images are then projected into a reduced vector space, viz. the vector space that is formed only by the selected significant eigen vectors. Finally, their feature vector is formed by the projection coefficients. Only then do they estimate a linear transformation function from the feature vector which will generate a frontal-face feature from a non-frontal face. The work that is done by [40] uses the Indian Institute of Technology Kanpur (IITK) Database, where each subject has images in poses: 0° (being a frontal pose), ±15° and as well as ±30°. The IITK database is a multimodal project which is sponsored by the Ministry of Communication and Information Technology, New Delhi. In this project IITK develops other biometric systems such as fingerprint and iris recognition, including face recognition. Finally, [40] uses 50 images to learn the transformation function and 59 for testing.

To learn pose-invariant features, Zhang et al [41] use a high-level feature learning scheme, which incorporates random faces (RF) and sparse many-to-one encoders (SME). Autoencoders are methods which are used for unsupervised learning in artificial neural networks (ANN), for efficiency in coding in order to learn representations (features) on a particular data set. They first select facial landmarks on face images at ±45°and ±75°. Then according to the similarities defined by the landmarks, they use an engineered feature extraction method to extract pose-invariant facial features. Their method of feature extraction is based on a single-hidden-layer neural network, which they use to extract discriminative features, and train an input facial image that is in different poses (many). In contrast to the traditional feed-forward neural network, where the hidden layer is the output of a weight function followed by an activation function, their model uses the hidden layer as pose-invariant high-level feature representation, assuming that images of the same subject in different poses share the same high-level feature representation. They then use these features to compare the input to gallery of face images which are in the frontal pose, to confirm the identity of the received probe image. A similar method of extracting engineered features was earlier used in [42].

Ensuring better semantic correspondence across poses in face images is achieved by combining landmark-level and component-level features in [43], through extracting Multi-Directional Multi-Level Dual-Cross Patterns (MDML-DCPs). Using the first derivative of the Gaussian operator, MDML-DCP computes the face image by reducing the impact of illumination. DCP's are descriptors that are used to represent a face image. These descriptors are based on the textural structure of a human face. In order to build a pose-invariant face representation they follow three steps, namely, image filtering, local sampling and pattern encoding. In order to perform local sampling and pattern encoding, they use Dual-Cross Patterns (DCPs). These patterns are formed by landmarks from facial components, i.e. eyebrows, eyes, nose, mouth and the forehead. To encode the patterns that will form a face representation, they combine textural information and DCPs belonging to the same facial component. Furthermore, to construct a pose-invariant face representation, they use MDML-DCPs to convert grey-scale face images into multi-directional gradients that are invariant to illumination. The conversion is performed by the first derivative of the Gaussian operator. The next step is to build pose-invariant descriptors by normalising the face images using geometric rectification based on the similarity and affine transformation. The reason they use a similarity transformation is that it restores the original information of facial components and; facial contours; and also retains their configuration, while the use of affine transformation reduces differences in appearances caused by pose variations [43]. Their method proves that by combining both component-level and holistic-level features the representation generated is complementary and appropriate, thereby promoting robustness to variations. The reason is that component-level features are independent of changes in face pose; for they focus on a single facial component, whereas even though holistic-level is sensitive to changes in pose, it captures complete information on both facial contour and facial components.

2.12 Conclusion

From the methods discussed in this chapter, our method performs classification of input face images as the first step towards correcting the pose of the face image. This chapter looked at the fundamental terminology and calculations regarding the extraction of features and feature classification learners. Also, it looked at some of the work done by researchers to extract pose-invariant features. In the next chapter we will go through the methods that were used in this research work in order to achieve a representation for pose-invariant face images.

Chapter 3

Methodology

3.1 Introduction

This chapter specifies the formal steps leading to image pre-processing, feature extraction, feature normalisation, supervised classification and comparison and evaluation of classification models. The methodology of this study is based on the model of Ho et al [9]; for face pose classification. Unlike [9], the purpose of this study is not to use Markov random fields for matching face images, but to implement a classification model that will classify a still face image according to different poses. The chapter further explains corresponding architectural illustrations. Our approach focuses on creating a pose-estimation model by classifying images in different poses. Figure 3.1 illustrates a flowchart that shows the proposed methodology of this research study. First an image dataset will be constructed from the FERET database. These images will be in different poses and randomly selected. Secondly, image pre-processing will be performed to improve and enhance image data, in order to remove unwanted distortions, such as noise. The next step will be to extract features in order to create representation of the face images that are going to be used in this research work. Following the features extraction will be the normalisation step, where the range of the extracted features will be standardised. After normalisation, the normalised features will be divided into training data and test data, then a supervised classification model will be prepared which is required to make predictions (of poses). Finally, from these trained models, a decision will be made on which model best classifies our data. The model that best classifies our data will be selected to be the pose classification model.

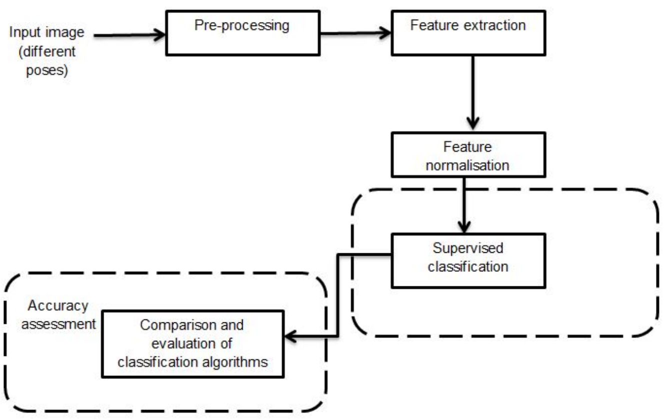

Input image (different poses)
Pre-processing
Feature extraction
Feature normalisation
Supervised classification
Accuracy assessment
Comparison and evaluation of classification algorithms

Figure 3.1: Proposed approach.

3.2 Pre-processing

Pre-processing is a process that includes several functions. The main purpose of pre-processing is to improve or enhance the image (in most cases presented in pixels) data that cause distortion, for both input and output, and it is also important for further processing of the images [29]. Figure 2.1 in the previous chapter, presents some of the FERET images that are used in this research work. One of the main advantages of the FERET dataset is that it is available as open source, and although the images are not of high resolution (HR), there are a reasonable number of face images (14126) that address the problem of face pose. For this research work, a hundred images from each of the four different poses (P1 - P4) are selected to build a classifier, giving a total of 400 images for both training and testing. The computation time is directly proportional to the number of images used. The images range from qr (+22,5°) quarter right, ql (-22,5°)quarter left, random images where subject faces to photographer's left, rd (-45°) and random images where the subject faces to the photographer's right, ra (+45°). Furthermore, rd and ra are random but consistent images whose precise angles are unknown. For example, both the first and the second images Figure 2.1 are of the same pose, qr. The images are in colour and of size 768 x 512 pixels. These images form the basis of our dataset.

3.2.1 Patch cropping

As seen from the montage in Figure 2.1, and from the size of the images mentioned in the previous paragraph, these images are relatively large (mugshots, showing the shoulder area) to be directly processed, hence we do not use the whole face area (global), but select only the local landmarks (eyes, nose and mouth). Also, because of the chosen face pose (non-frontal), we could not automatically detect the landmarks using the Viola-Jones algorithm [44], the first object detection framework proposed by Paul Viola and Michael Jones, that provided competitive object detection rates in real life. Besides object detection, this algorithm is also used for frontal face detection. As a result, we used a snipping tool [45], to capture clips of an image. This tool helps to manually select the patches that are desired (eyes, nose and mouth). Our patch size was not fixed, the size varied from patch to patch, i.e. one subject's patch size of the mouth is not the same size as the next subject's patch size. The idea is to find a rectangle that covers the area of the chosen landmark, as seen in Figure 3.2, without cropping any of the landmarks around it. Algorithm 1 shows the described procedure of cropping the patches from the image

Algorithm 1 Patch cropping

```
$face\ image \leftarrow PatchSize$
1: if $PatchSize = m$ x $n$ matrix (width $\times$ height) then $imageCropping = save\ patch$
2: else $imageCropping = discard\ patch$
3: end if
```

We then used the image set function from Matrix Laboratory (MATLAB). This function is for loading the patches on to MATLAB, where they will be processed as a collection of image datasets rather than individually. MATLAB is a tool that scientists and engineers use to analyse and design systems. This tool has been used in different research areas including machine learning, computer vision, image processing and robotics. The main reason MATLAB was selected to pursue this research work is that it provides a number of useful tools with regard to image processing. Furthermore, MATLAB provides a platform that is interactive, i.e. one is able to monitor images by means of visual display. In addition to image processing tools, MATLAB offers statistical as well as classification tools. When loaded on to MATLAB, these patches are then converted to matrices, the format in which MATLAB reads images.

The smallest patch that is loaded in the image set is of size 39 x 164 x 3 pixel unit8, where:

- 39 x 164 (width x height) pixel represents the size of the patch.
- x 3 represents the RGB colour space, i.e. the patches are first loaded in true colour defined by the three chromaticities of red, green and blue primaries.
- unit8 is a MATLAB data type, which stands for unsigned integers, i.e. the values contained in a particular matrix are all positive.

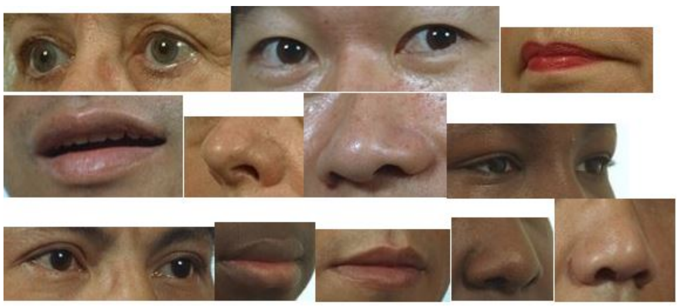


Figure 3.2: Eye, mouth and nose cropped patches.

3.2.2 Pixel brightness transformation

When programming an image, we need to consider that pixels have width and height therefore we use the formula:

- Assume an image with a given WIDTH and HEIGHT, measured in pixels.
- We know that the pixel array has a total number of elements equalling WIDTH * HEIGHT.
- For any given X, Y point in the image window, the location in our pixel array is: LOCATION = X + Y * WIDTH

Furthermore, from the pixel properties discussed in the previous chapter, we further learned that image enhancement covers pixel brightness transformation, where pixel brightness has two classes, namely, brightness corrections and grey-scale transformation. Brightness correction changes the brightness of the image without taking into consideration the position of the image pixel, and also retains the luminance of the image. Grey-scale simply applies the irreversible conversion (brightness) of a photo, rather than applying adjustment (changing the photo completely). The main aim of grey-scaling the images is to attain images with equally distributed brightness levels over the whole brightness scale. The distinction between black and white (BW) and grey-scale images is that BW have only two pixel colours, i.e. black and white pixels; whereas, grey-scale intensity is stored as an 8-bit integer, i.e. 256 possible shades of grey from black and white. The transformation function is given as:

$$s = T(r) \tag{3.1}$$

where s is the pixel value of the output image, r is the pixel value of the input image and T is a transformation function that maps each value of r to each value of s.

In this study, we used the linear grey level transformation. For the linear transform we used a simple identity transformation function, so as to enhance our images in order to provide more detail about the image, and also for better contrast. We had initially selected the power - law transformation, but later observed that we did not extract as many strong keypoints as we would desire to, for better classification, i.e. some of the patches contained a lot of 0's thereby resulting in less information about the patches. Nonetheless, in identity transition the mapping is direct for each value of the input image to the output image, resulting in the same input image and output image. Figure 3.4 shows a representation of a true colour and a grey-scaling image. The two images are still the same in position and scale even after grey-scale, the difference is the brightness of the images.

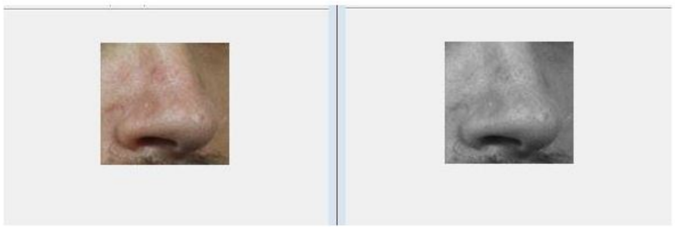


Figure 3.3: Representation of the original and grey-scale images.

3.2.3 Image filtering using guided filter

Guided image filtering has been proven to be efficient and also effective in a larger diversity of machine learning and computer vision applications [46]. We chose the guided filter as a method of filtering because it allows edges to be preserved on the patches, considering that in our feature extraction method, we are required to compute image edges. Also this method of image filtering has proven to have better behaviours near edges, and it can transfer the structure of the guidance image (be it the input, output or a different image). The guided filter uses the contents of the guidance image to influence the filtering, by creating a less smoothed and more structured output image. In this research work, the filtering output patch at a pixel i is a weighted average of:

$$q_i = \sum_j W_{xy}(I)\rho_j \tag{3.2}$$

where x and y are the pixels at a given point, and I is the guidance image whose function is a filter kernel W_{ij} and is independent of ρ.

3.3 Feature extraction

For this work, we selected the SIFT algorithm to convert the face patches into keypoints for better database representation. The reason we chose this algorithm is that we are not only getting the parameters x and y (the position of a pixel) and t which is the scale of the image, but SIFT finds keypoints which are rotation, viewpoint and illumination invariant. Algorithm 2 shows a step-by-step method of how this algorithm was used in this work to create a descriptor.

While the original scale parameter (σ value) of SIFT is zero (0), we extract the SIFT keypoints at three different scale parameters; $\sigma = 3$, $\sigma = 6$ and $\sigma = 9$. This is done so that we observe the value of σ where we are able to extract enough keypoints for accurate classification of face pose, i.e. the level of blur that can give us better classification of face images at different poses.

Algorithm 2 Compute SIFT descriptor

```
SET octave
SET level
SET $\sigma$
procedure –CREATE SCALE SPACE
    for each octave do
        Create Gaussian blur intervals
        Create DoG intervals
        Compute edges for each interval
        procedure –FINDING KEYPOINTS
            Search each octave for stable extrema
            Create keypoints at dominant orientations of extrema
            procedure –GENERATE A DESCRIPTOR
                for each keypoint do
                    Rotate sample grid to match keypoint orientation
                    Sample the rotated region and create descriptor

                    Save descriptor
```

The goal of the above process is to determine the value of σ that will be suitable for our dataset, which will assist in extracting meaningful descriptors for each patch. Again in chapter 2 we learned that to remove unwanted keypoints, i.e. those which are low in contrast and those which lie at the edge of the image, a certain threshold has to be set that will determine the magnitude of the current pixel. We will again regulate the threshold value for better localisation of keypoints.

Lastly, to create a 128-dimension descriptor, we keep the original window at 16 x 16 around the keypoint and divide the window into sixteen 4 x 4 windows. Therefore, the magnitude ($m(x_i, y_i)$) and orientation ($\theta(x_i, y_i)$) of each patch sample (pixel) will be calculated as shown in equations 2.8 and

2.9 for all classes ($\pm 22, 5°, \pm 45°$). The objective of this feature extraction method is to extract pose invariant features, by locating meaningful keypoints and constructing significant descriptors which can also be used for texture classification in future research work.

3.4 Feature Normalisation

Feature normalisation is a step which is performed during the image processing stage. This method is used to standardise the data from the extracted features. Each feature component is normalised to a range of [0 1]. There are various normalisation methods used which are scale invariant such as residual data, mean data and standard deviation data. The normalisation procedure used in this research work is the Linear scaling to unit variance. The main purpose of choosing this procedure over others is to transform the feature component x to a random variable with mean 0 and unit variance. The procedure taken to achieve scaled features is described in subsequent sections:

3.4.1 Calculating the Mean (μ)

Calculating the Mean (μ) means we calculate the average of all the feature vectors extracted from each patch. We have 100 patches per pose, i.e. P1 - P4, the mean is calculated per pose as the average of features (100 x 128), 100 images each having 128 feature vectors. We then use the formula:

$$\bar{X_j} = \frac{\sum_{i=1}^{n} X_i j}{n} \tag{3.3}$$

where:

- n is the keypoint of all the patches, $n = 100$;
- i is each keypoint in a feature vector per patch (column), such that $i = 1$, ..., n;
- j is each keypoint in a feature vector per patch (row), $j = 1$, ..., 128;
- X_{ij} is the extracted keypoint at position $_{ij}$.

In other words, μ of a pose class will be a feature vector of 128 elements representing the mean of any given class of the four classes we have. The purpose of this step is to find the average per pose class. The next step is to calculate the spread of features per class.

3.4.2 Calculate the Standard Deviation (σ)

The definition of Standard Deviation is given as, the average distance from the mean of the data set to a point [47]. Here we calculate how spread out our features are, by calculating the squares of the distance from each data point, to the mean of the set. Thereafter, we added all the calculated squares, divided by $n - 1$, then took the positive square root. The formula used is:

$$s_j = \sqrt{\frac{\sum(X_{ij} - \bar{X_{ij}})^2}{n-1}} \tag{3.4}$$

Where:

- n is the keypoint of all the patches, $n = 100$
- i is each keypoint in a feature vector per patch (column), such that $i = 1, ..., n$
- j is each keypoint in a feature vector per patch (row), $j = 1, ..., 128$
- s_j is the keypoint at the j_th position
- X_{ij} is the raw feature
- $\bar{X}_{ij}$ is the mean at a position

Similar to the mean, the σ of a pose will be a 128-dimensional feature vector representing the Standard Deviation of any given class of the four classes we have. The next step following the two discussed subsections, is to normalise our features vector for each patch by using equation 3.5.

After determining the mean and standard deviation of the four poses, we then calculate the normalised feature vector for each patch:

$$\tilde{x_i} = \frac{|x_i - \mu_j|}{\sigma_j} \tag{3.5}$$

In the above equation, we let:

- x_i be the extracted keypoint at the i^{th}position;
- μ_j be the calculated mean at the j^{th} position;

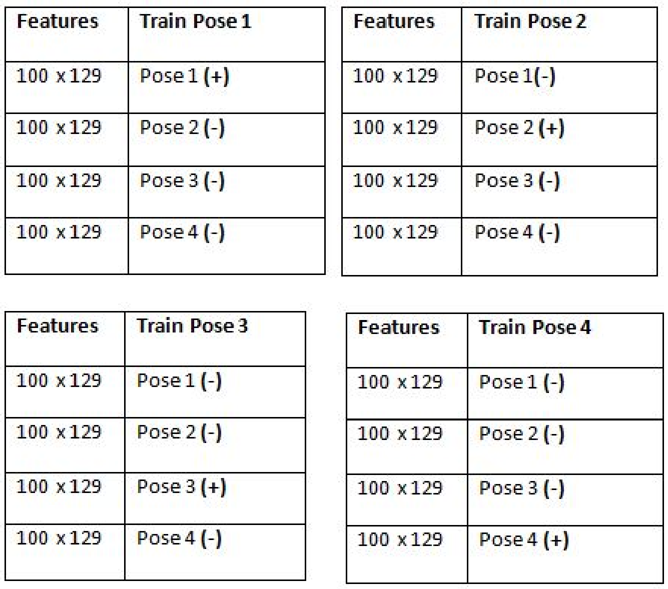
- σ_j be the calculated standard deviation at the j^{th} position;
- $\tilde{x_i}$ be the normalised keypoint at the i^{th} position.

3.5 Supervised Classification

We use a multiclass classification method for training our models. In a multiclass classification, the given inputs X represent a matrix of predictors, and a set of n training observations, $\vec{x_i}$, which consist of p-dimensional vector of features each, where each column represents one predictor and each row represents one observation. For each training observation, there exist a corresponding class label, $y_i \in \{1, -1\}$, represented by Y which is an array of class labels (P1 - P4) where each row corresponds to the value of corresponding x_i. Therefore, n normalised SIFT feature vectors of dimension 128 that contain training observations of $(\vec{x_i}, y_i)$ which represent the feature at position i and the corresponding class label at y_i. Furthermore, test observations which will be used at a later stage in order to test the classification performance of our model are represented by $\vec{x}^* = (x_1^*, ..., x_p^*)$. With this information, our goal was to classify into which pose (°) the input image falls.

3.5.1 Support Vector Machine (SVM)

Essentially, there are two techniques in which one may train a multiclass classifier, namely, One-against-All (1AA) and One-Against-One (1A1), where 1A1 is a classifier that performs pairwise comparison between n classes, and each classifier trained on two of these classes, giving a total of $\frac{N(N\text{-}1)}{2}$ models. On the other hand, 1AA builds model with N different binary classifiers [48]. Upon testing our data on both these techniques, we opted to use the 1AA technique. Our reason for choosing this approach was not only based on the data, but that it was the most commonly used approach [48]. In addition we had observed that there was no significant difference in validation accuracy percentage when we trained our data on both models. Hence we opted to train the data using the One-Against-All approach. Figure 3.4; tabulates how each model is trained and how a class is tested against other classes. For instance, when Pose 1 is trained, all other poses become negative and Pose 1 becomes positive, and so on. Furthermore, we used an adjustable kernel in order to be able to adjust the kernel scale to improve the model learning.



Features	Train Pose 1
100 x 129	Pose 1 (+)
100 x 129	Pose 2 (-)
100 x 129	Pose 3 (-)
100 x 129	Pose 4 (-)

Features	Train Pose 2
100 x 129	Pose 1(-)
100 x 129	Pose 2 (+)
100 x 129	Pose 3 (-)
100 x 129	Pose 4 (-)

Features	Train Pose 3
100 x 129	Pose 1 (-)
100 x 129	Pose 2 (-)
100 x 129	Pose 3 (+)
100 x 129	Pose 4 (-)

Features	Train Pose 4
100 x 129	Pose 1 (-)
100 x 129	Pose 2 (-)
100 x 129	Pose 3 (-)
100 x 129	Pose 4 (+)

Figure 3.4: Model training.

3.5.2 k-Nearest Neighbours

We used k-Nearest Neighbours (k-NN) to solve a classification problem. The first step will be to determine the value of k. In Chapter 2; we learned that with an increase in the k value, the separating boundary becomes smoother; the intention is to increase the value of k until we get a smoother boundary. Also, from the same value of k; we can determine the two parameters which are crucial for our research work, namely, the validation error rate and the training error rate. These error rates together with the training curve will help us determine a suitable value of k.

3.5.3 Decision trees

Considering that our data are categorical, i.e. either ”true” or ”false”, the aim is to use a decision tree algorithm that performs binary splits. The decision tree that is going to be trained will set apart the descriptor to be trained based on all the four poses that are chosen. Figure 2.7 in the previous chapter depicts the structure of the decision-tree model for this research work. The normalised SIFT descriptors will be the root node of the entire population on the tree. Thus, our root node will have 400 samples of dimension 128. This will then be further divided into four nodes (±22,5°and ±45°) for training, and be labeled accordingly.

The first step will be to set limitations on the tree size to ensure that there is no over-fitting while modelling a decision tree. The over-fitting problem can be eliminated by using various parameters which are used to define a decision tree. Parameters that we have to consider include:

- Minimum required number of observations to split a node. Minimising the observations will also help in avoiding the problem of over-fitting our model.
- Controlling the depth of the tree. Using K-fold validation, we would determine at which value of K we obtain better classification without over-fitting our model. Also considering that higher values of K will allow our model to learn similarities of face pose.
- Maximum number of surrogates per node. Considering that in a binary tree, a depth of n; would produce a maximum of 2^n nodes, this will help to determine the maximum number of surrogates per node.
- Maximum features to consider when splitting the tree.

3.5.4 Neural network pattern recognition

In order to train the extracted features using neural networks, we chose a two-layer feed-forward neural network. The reason we chose this classification algorithm is that it can perform the classification of vectors with less computational complexity, yet achieve satisfactory classification accuracy, if the network is given enough neurons in its hidden layer. Figure 3.5 depicts our network architecture. The network will consist of an input layer; its inputs are SIFT descriptors from the four different poses that we chose, i.e. $\pm$ 22,5 ° and $\pm$ 45°. The data will then be passed through the next layer which is a hidden layer that consists of a sigmoid activation function. The sigmoid activation function takes the mathematical form of $\theta(x) = \frac{1}{(1+e^{-x})}$ which makes learning of connectors in a neural network easier. The sigmoid function is used in neural networks and machine learning because it uses less computational power to process the neurons in a network. In our work, the sigmoid function is incorporated with softmax output neurons, which is a function used in the final layer of our neural network classifier. The figure also shows the weights denoted by (W). These weights determine connection from the current neuron to the next layer's neuron, each connection carrying different weights. There is no connection among neurons of the same layer. To ensure better performance of our network, we plan to increase the number of neurons during the training phase.

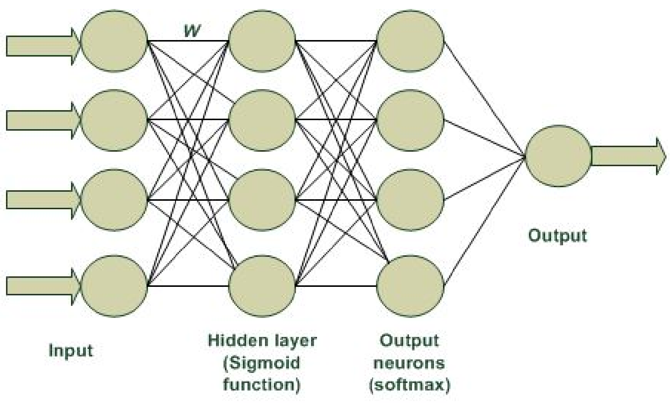


Figure 3.5: Model training.

3.6 Conclusion

In this chapter we looked at mathematical equations and algorithms which forms part of the methodology for this research work. In the following chapter we will look at the performance of our data, when methods discussed above are applied, these results will also be analysed.

Chapter 4

Analysis of results

4.1 Introduction

This chapter discusses the experiments that were performed to determine which angle class the input face falls. The main purpose of our experiments was twofold, namely:

- To determine the angle of the input; thereby accomplishing the research objective (1).
- To determine the classifier that gave the highest classification accuracy on patch-based face images; thereby accomplishing the research objective (2).

Given the type of problem we are addressing (face pose), model evaluation was not only done to learn how well a specific model could predict a given pose, but also to learn how well to choose between different models, by tuning different parameters. Furthermore, to quantify the model's performance, by using both classification accuracy and a given metric.

4.2 Pre-processing

The dataset used for experiments, which were conducted in this study are randomly chosen mugshots from the FERET database. The database is distributed by the National Institute of Standards and Technology (NIST), in order for researchers to develop better automated face recognition algorithms, technology and techniques. Also, this database is used to test and evaluate different face recognition algorithms. The release agreement for the download of the FERET database can be downloaded at the following link: https://www.nist.gov/itl/iad/image-group/color-feret-database. The mugshots were selected by going through the images in the database, and selecting the images that have all required poses per subject. The subject's set of images had to have all four poses used to create training classes for this research work, i.e. ql, qr, ra and rd face poses. The mugshots that were excluded are those which had some or none of the poses we selected for the training set. The selected mugshots are from subjects of different races, i.e. Black, White and Asian. The subjects have wrinkles, are bearded (facial hair) and some of the subject's' eyes are half open. Essentially, these subjects are in four different poses, ±22,5°and ± 45°, i.e. P1 = -22,5°, P2 = +22,5°, P3 = +45°and P4 = -45°.

Upon obtaining the patches using Algorithm 1 in Chapter 3, pre-processing was performed. Firstly, the patches were set to the same size (200 x 200 (wxh)), in order to be able to combine them. The re-sizing was further useful when we extracted features, as any size lower than 200 x 200 was not sufficient for extraction of features for some patches. After performing a guided filter by applying equation 3.2, we obtain an image as shown in Figure 4.1. This image is a patch for one subject in which features will be extracted in order to form a representation.

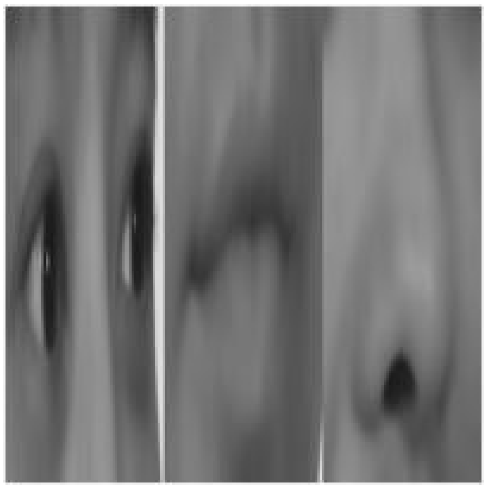


Figure 4.1: Guided filtered image.

4.3 Feature extraction

Initially, using the equation 2.2 discussed in Chapter 2, $L(x_i, y_i, \sigma) = G(x_i, y_i, \sigma) * I(x_i, y_i)$ to create a scale space and Gaussian blurs, we set octave and level to the original Lowe's values, which are 4 and 5 respectively. The problem was that when keypoints were extracted, there were more fields of the

128-descriptor that contained '$0's$'. As a matter of fact, we realised that the classification percentage drops drastically when classification is performed, and also the algorithm takes long to execute, giving an average of 14.41 seconds elapsed time at run-time. We then decreased both values (octave and level) to 3 at each σ value (3,6,9). This change of the original SIFT algorithm gave us better representation of $L(x, y, \sigma)$ also giving us the average run-time of 1.14 seconds, thereby computing Difference of Gaussian (DoG) images and locating keypoints more efficiently.

Following the procedure in Algorithm 2, the SIFT keypoints used in this research work were extracted at different scale parameters (σ), as a result, creating different scale spaces, i.e. scale space are created at ($\sigma = 3$, $\sigma = 6$ and $\sigma = 9$) for each pose. The choice of different scale parameters was performed so as to identify the ideal scale space for our dataset. For instance, in Figure 4.2 σ is set to 9 to extract features for Pose 1, 2, 3 and 4, thereby obtaining the patches in Figure 4.2, where $\sigma = 9$, i.e. $G(x, y, \sigma)$ Difference of Gaussian patches. These patches are a result of both equations 2.2 and 2.3, where a Gaussian blur operator $G(x, y, \sigma)$ is applied to obtain the image in Figure 4.2.

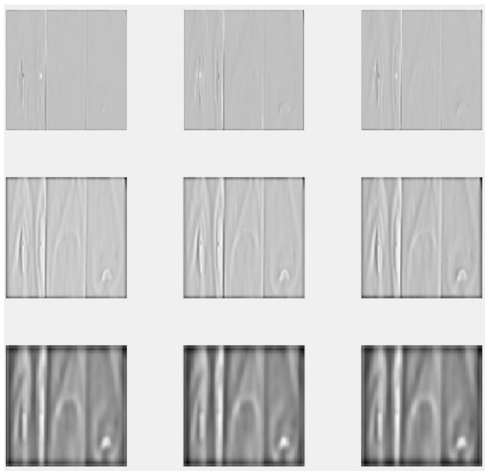


Figure 4.2: The DoG patches at $\sigma = 9$, $G(x, y, \sigma)$.

Furthermore, the next step is to locate keypoints by using the DoG patches obtained in Figure 4.2. We apply equation 2.5 in order to locate the subpixel keypoints for each patch, and also apply equation 2.6 and 2.7 for accurate localisation of keypoints on the DoG patches. This process of localisation of keypoint was depicted using the $G(x, y, \sigma)$ patches for clear visualisation in Figures 4.3 (a) and (b). Figure 4.3 (a) shows the low contrast keypoints that were initially located in green colour. Some of these keypoints are either poorly localised or low in contrast therefore we needed to eliminate them. So, looking closely at Figure 4.3 (b) we observe that some of the keypoints have fallen off, only the ones that are accurately localised (in blue colour) remain. These keypoints are the ones which were used to construct a pose invariant descriptor for better classification of face images. Moreover, the threshold value for accurate localisation of meaningful keypoints is set to be 0,1, thereby allowing elimination of poorly constructed keypoints or those with low contrast. These keypoints were useful for construction of a descriptor. The reason we chose 0,1 was that we were unable to construct a descriptor at a higher threshold value because of our patch size.

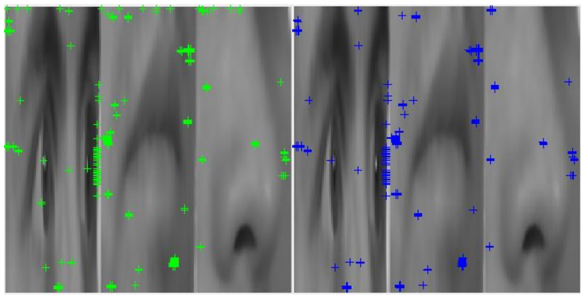

(a) Low contrast (b) Accurate keypoints

Figure 4.3: Keypoints localisation.

Even though the SIFT descriptor that we extracted was normalised, it had uneven keypoints, i.e. some keypoints were slightly if not much higher than others, i.e. negative values were extracted; as a result, the positive keypoints appeared to be more significant in weight than negatives. Initially, we took raw SIFT keypoints and put them in a scatter plot; we obtained a scatter plot as depicted in Figure 4.4. The horizontal values represent the x-axis, while the vertical values represent the y-axis of the raw SIFT descriptor. As seen in the figure, the values of the descriptors are well between 0 and 1, meaning they are normalised. These features were trained using a random SVM classifier, with pre-defined parameters. We obtained a percentage of 29,8% classification accuracy.

As a consequence of the above classification accuracy, we then applied linear scaling to unit variance normalisation method discussed in the previous chapter. Upon applying this method, we obtained data as shown on the scatter plot in Figure 4.5. We then took these descriptors and trained them with the same SVM classifier the raw descriptors were trained with. The classification percentage obtained with the same pre-defined parameters of the classifier was 54,3%. Our observation regarding this increase in the classification percentage is that, the classifier learns better with scattered keypoints, as opposed to when the keypoints are clustered in one place as in the raw keypoints, where keypoints are clustered just below 1 and most of the extracted keypoints were '0'. The objective of the normalisation process was to scale the keypoints to have an equal mean and standard deviation using equation 3.3. By normalising the descriptor, we ensure that each keypoint that was extracted and accurately localised, is equally significant for better classification of patches.

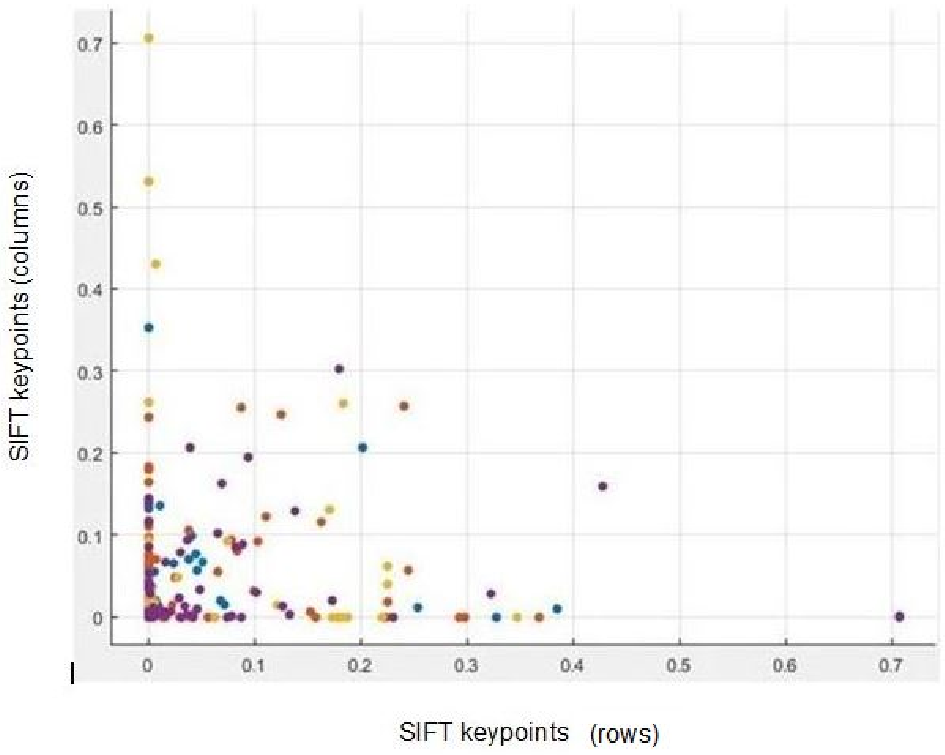


Figure 4.4: Raw SIFT Descriptor.

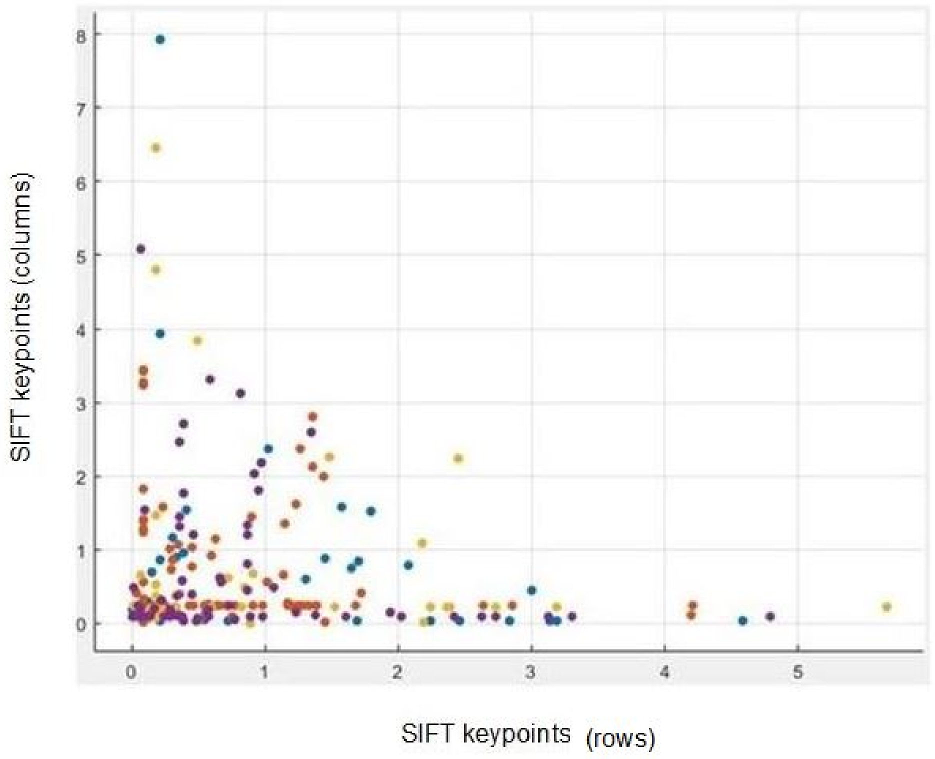


Figure 4.5: SIFT Descriptors normalised using linear scaling to unit variance method.

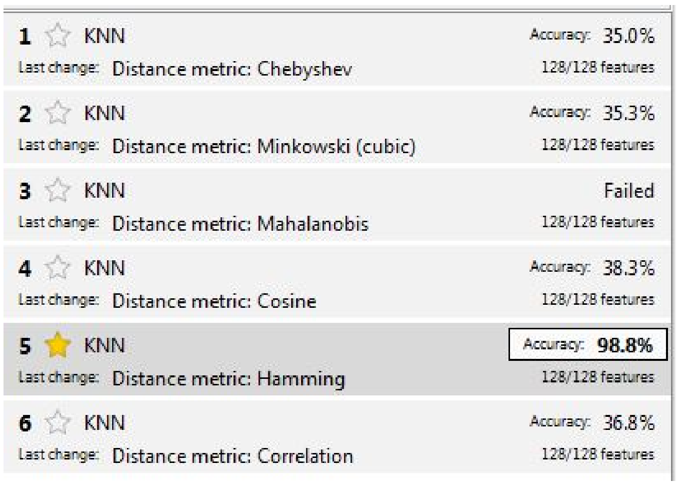
4.4 Supervised classification, Comparison

This section and the proceeding sub-sections will present pose classification results of the experiments performed in this study. We will present different results for different classifiers, obtained from extracting SIFT features at different scale spaces, i.e. σ (3, 6 and 9). Firstly, the training data sets are given as inputs to the classification algorithms in order to develop a classification model. To obtain the training and test set, we used a K-fold cross validation method for all data sets, because when using this method it does not matter how the data gets to be divided, a data point gets in the training set (k-1) times, also we can obtain an indication of how well the learner will perform when given new inputs to predict. As a result, we used 10-fold of the total data to form a training set across all data sets. The remaining set was used for validation. Initially, all four classes (face poses) were trained on all different types of classifiers, then we chose a classifier that performed better given a specific pose. The choice of the classifier depends mainly on the performance of a trained classification model.

The results were analysed according to different levels of σ values as shown in Table 4.1; together with the kernels and kernel parameters that obtained the highest classification results. As well as their respective performance metrics, evaluating the sensitivity (the ability of a classifier to correctly identify those which belong to a class) and specificity (the ability of a classifier to correctly classify those elements which do not belong to the group).

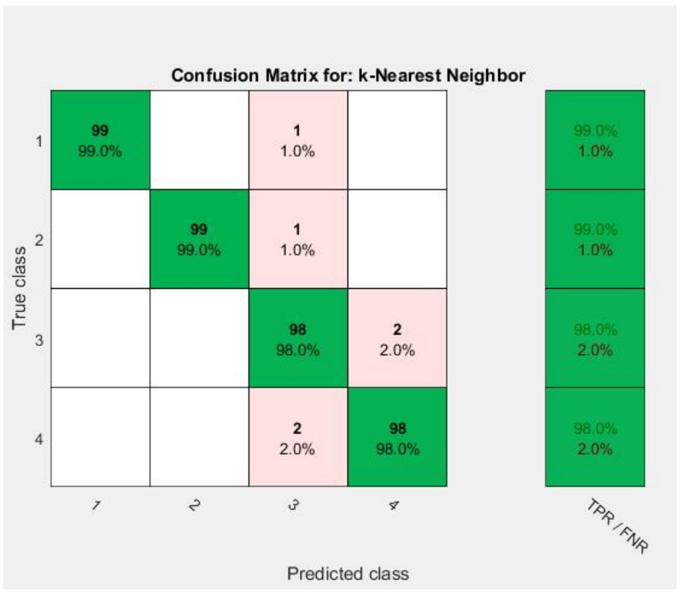
4.4.1 Classification at $\sigma = 3$

Features were extracted and trained at $\sigma = 3$, normalised using linear scaling to unit variance. These features were trained using distance-based measures shown in Figure 4.6. The Hamming distance metric shows a better classification rate as compared to other distance metrics that were trained, obtaining a 98,8% classification rate, this means the likelihood that a face pose will correctly be classified for the trained pose is high. However, the classification rate on its own does not communicate much about the data that was classified. We then used a 4 x 4 confusion matrix (because there are four classes) as shown in Figure 4.7. The confusion matrix shows that for both Pose 1 and 2 the classifier predicted 99 features (in green) correctly. These are True Positives which correspond to 99% accuracy, whereas one image (in red) was misclassified (False Negative) as pose three for both classes. Similarly, the confusion matrix also shows that 98 features from Pose 3 and 4 were correctly classified, corresponding to the 98% and two of the features were misclassified, corresponding to the 2% of the data.



1 ☆ KNN	Last change: Distance metric: Chebyshev	Accuracy: 35.0%	128/128 features
2 ☆ KNN	Last change: Distance metric: Minkowski (cubic)	Accuracy: 35.3%	128/128 features
3 ☆ KNN	Last change: Distance metric: Mahalanobis	Failed	128/128 features
4 ☆ KNN	Last change: Distance metric: Cosine	Accuracy: 38.3%	128/128 features
5 ★ KNN	Last change: Distance metric: Hamming	Accuracy: 98.8%	128/128 features
6 ☆ KNN	Last change: Distance metric: Correlation	Accuracy: 36.8%	128/128 features

Figure 4.6: kNN Trained Models.



True class \ Predicted class	1	2	3	4	TPR / FNR
1	99 99.0%		1 1.0%		99.0% 1.0%
2		99 99.0%	1 1.0%		99.0% 1.0%
3			98 98.0%	2 2.0%	98.0% 2.0%
4			2 2.0%	98 98.0%	98.0% 2.0%

Figure 4.7: Confusion Matrix at $\sigma = 3$.

4.4.2 Classification at $\sigma = 6$

Features which were extracted at $\sigma = 6$ were classified using decision trees. With maximum of 10 splits and 10 surrogates, we obtained a classification rate of 97%. The observation made is that, even though we increased the number of splits, we still obtained the same classification percentage. Figure 4.8 shows a plot of the ROC curve for the trained tree, where we plot the TP (y-axis) against the FP (x-axis). The interpretation is that the closer the curve falls to the y-axis and to the top line of the ROC space, the better classification accuracy the trained classifier gives. The Confusion matrix shown in Figure 4.9 shows that for Pose 1, 95 features were correctly classified, which corresponds to the 95%. However, three of the 100 features were misclassified under Pose 3 and two (2) were misclassified under Pose 4. For Pose 2, 99 features were correctly classified, and one feature was misclassified under Pose 1. Three features which belong in Pose 3 and two features which belong in Pose 4 were also misclassified under Pose 1.

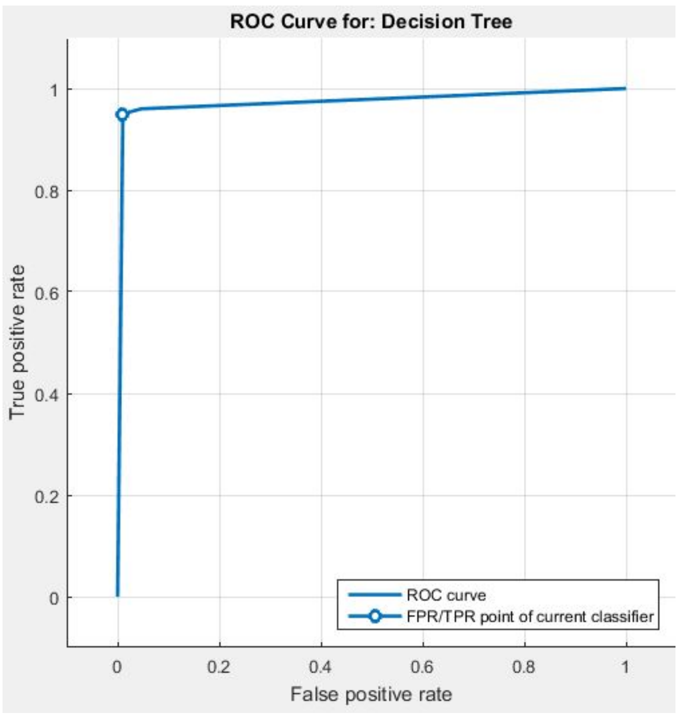


Figure 4.8: ROC Curve at $\sigma = 6$.

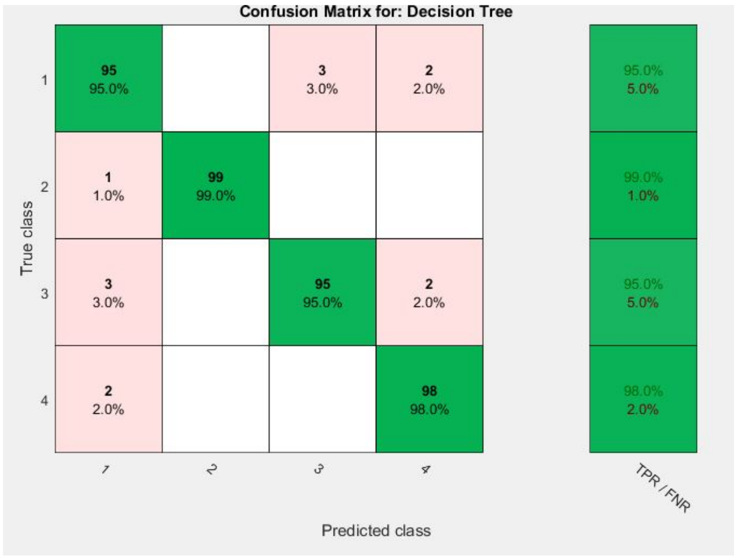


Figure 4.9: Confusion Matrix at $\sigma = 6$.

4.4.3 Classification at $\sigma = 9$

An SVM classifier was used to train features which were extracted at $\sigma = 9$. The kernel scale of the classifier was adjusted depending on the classifier's performance. At kernel scale 4 we obtained a classification rate of 72%, any number above caused a decrease in the classification rate. Even though we used 25-fold of validation division, the plot from the obtained ROC curve (Figure 4.10); shows that the model was not learning very well, as the graph is not as smooth as the previously discussed data, where the sigma value was low. At the same time, the confusion matrix (Figure 4.11) also shows (in red blocks) a lot of misclassified classes, which proves that at $\sigma = 9$, our data set is not performing well, even after training the data set with the previous classifiers (decision tree and k-NN).

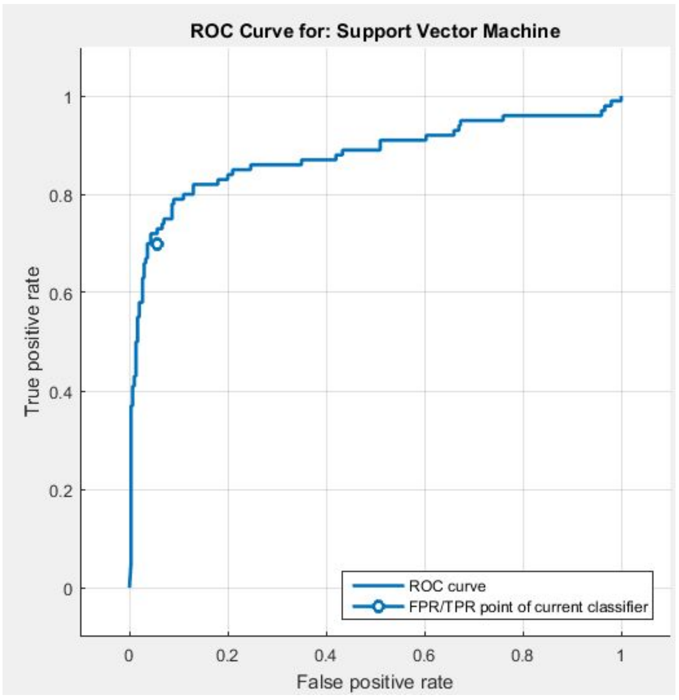
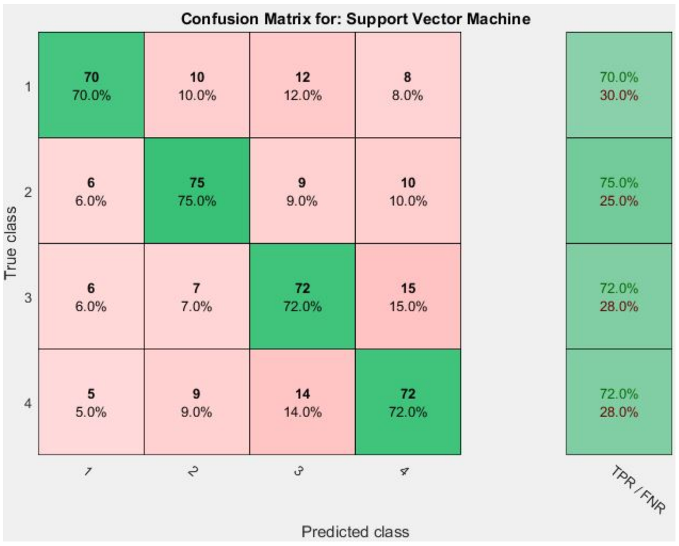


Figure 4.10: SVM ROC curve at $\sigma = 9$.

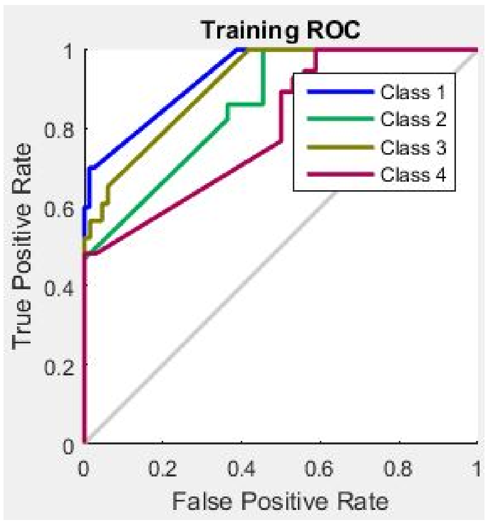

Confusion Matrix for: Support Vector Machine

True class \ Predicted class	1	2	3	4	TPR / FNR
1	70 70.0%	10 10.0%	12 12.0%	8 8.0%	70.0% 30.0%
2	6 6.0%	75 75.0%	9 9.0%	10 10.0%	75.0% 25.0%
3	6 6.0%	7 7.0%	72 72.0%	15 15.0%	72.0% 28.0%
4	5 5.0%	9 9.0%	14 14.0%	72 72.0%	72.0% 28.0%

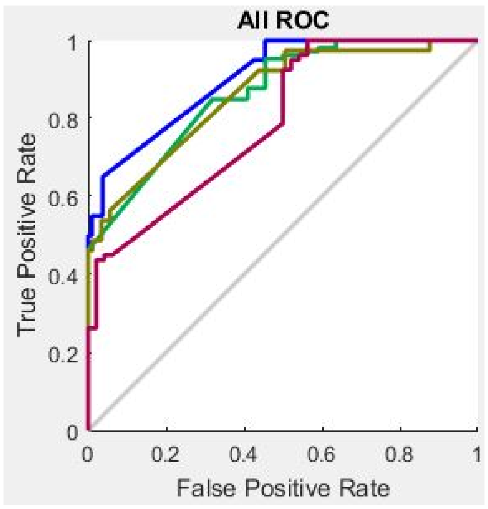
Figure 4.11: SVM Confusion Matrix at $\sigma = 9$.

4.4.4 Neural Network Pattern Recognition

Next, as an alternative to SVM and decision trees, the normalised SIFT features were trained using a two-layer feed-forward neural network. The input data set for each consists of a 400 x 128 matrix, which is 400 subjects by 128 SIFT feature vector for each patch and the target is a 4 x 128 matrix where each column consists of 0's and 1's. The presence of the 1 in a row indicates which of the poses the patch falls under. We initially trained all the poses per σ value. Features which were extracted at $\sigma = 3$ gave better results irrespective of a small training data set. Below are two ROC curve graphs which show how well the model learned. Figure (a) shows the blue, green, yellow and red lines representing the four classes that we trained. It shows how the model learns better on pose 1 and 3 ($\pm 45°$), as it covers over 0.7 (70%) of the area under the curve (AUC). When the AUC is at 1 (one) it means the model is learning well. The further the lines move away from the left border, the more difficult it gets for the model to learn. Furthermore, the curves also show that the probability of the model to misclassify a face pose is high, but the curves are not below the diagonal line, which may have suggested that the model is performing poorly.



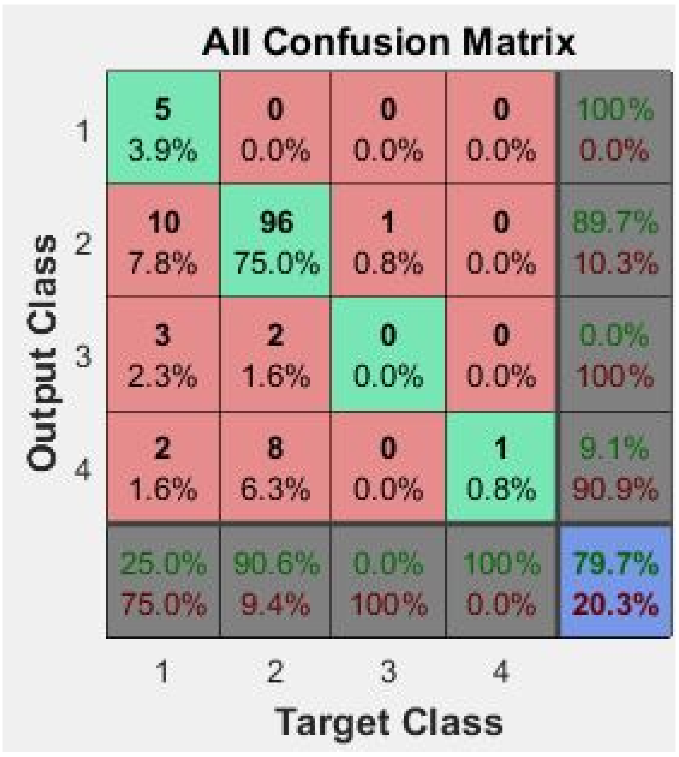
(a) ROC curve for Training set.



(b) ROC curve for overall network.

An equally significant aspect is the confusion matrix on Figure 4.12 which shows that after 23 iterations the performance of our model stood at 80%. While pose 2 obtained a good classification rate, there was no pose that was correctly matched for pose 3, having a misclassification rate of 2,3% in red.

Lastly, the performance plot in Figure 4.13 shows the error against the number of iterations for training, validation and test data, this error is based on the total system performance rate in Figure 4.12. It also show how the network's mean square error drops continually while it learns. The lines (blue, red and green) shows the decreasing error as the network learns. Training and validation converge at a point, indicating how well the network will generalise to new data.



All Confusion Matrix

Output Class \ Target Class	1	2	3	4	
1	**5** 3.9%	**0** 0.0%	**0** 0.0%	**0** 0.0%	100% 0.0%
2	**10** 7.8%	**96** 75.0%	**1** 0.8%	**0** 0.0%	89.7% 10.3%
3	**3** 2.3%	**2** 1.6%	**0** 0.0%	**0** 0.0%	0.0% 100%
4	**2** 1.6%	**8** 6.3%	**0** 0.0%	**1** 0.8%	9.1% 90.9%
	25.0% 75.0%	90.6% 9.4%	0.0% 100%	100% 0.0%	**79.7%** **20.3%**

Figure 4.12: Neural Network Confusion Matrix.

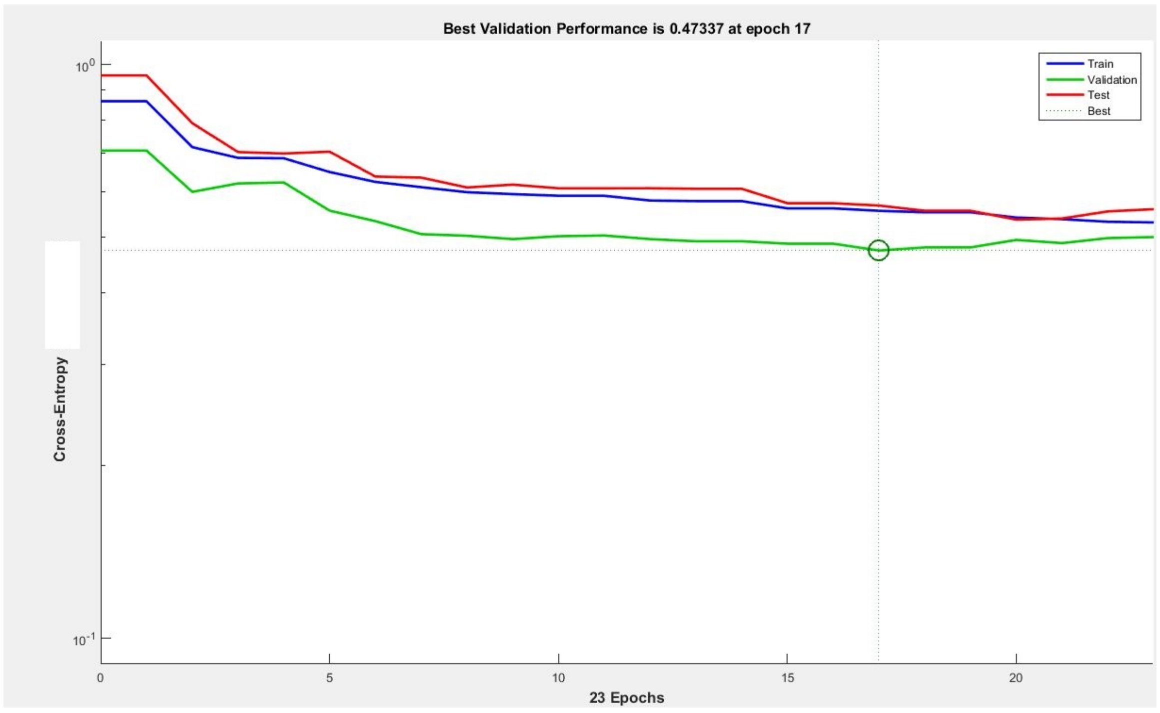


Figure 4.13: NN validation Performance Plot at $\sigma = 3$.

4.5 Evaluation of algorithms

The Confusion matrix used above is helpful and useful to understand the performance of our classifier, given a set of features. However, it is not a model evaluation metric, we need to be able to choose a better performing model between models that were trained, and determine which model classifies our features better. In order to evaluate the performance of the trained models, we used Sensitivity (True Positive rate) and Specificity (True Negative rate). These performance evaluation metrics were calculated using the formula 2.14 in Chapter 2. The aim is to get both rates high, in that way we knew that the model would correctly classify the pose.

Firstly, evaluation of Specificity and Sensitivity was performed for each class, i.e. one against all. Using the information from each class, the second step was to calculate the overall performance of a given classification model, using real numbers not percentages. At $\sigma = 3$, Sensitivity was 0.98 and Spe = 0.98. Whereas, $\sigma = 6$ obtained Se = 0.94 and Spe = 0.97. Lastly, $\sigma = 9$ obtained Se = 0.64 and Spe = 0.64. From these evaluations we concluded that kNN performs better where $\sigma = 3$. While on the other hand, SVM at $\sigma = 9$, gave us undesirable classification rate, with the features that were extracted. Table 4.1 evidently shows a summary of the results obtained, where using kNN a classification rate of 98% was obtained as the better performing classifier.

Classification Learners					
SVM	Sigma (σ)	Validation Division	Kernel function	Kernel Scale	Classification rate
	9	25 folds	Quadratic	4	72%
Decision Tree	Sigma (σ)	Validation Division	Split method	Max. splits	Classification rate
	6	10 folds	Gini's diversity index	10	96,8%
*k*NN	Sigma (σ)	Validation Division	Distance Metric	Num. of Neighbors	Classification rate
	3	10 folds	Hamming distance	3	98,8%
NN Pattern Recognition	Sigma (σ)	Validation Division	Activation Function	Epoch	Classification rate
	3	90 samples training, 19 samples validation and testing respectively	Sigmoid and softmax	23 iterations	79%

Table 4.1: Overall Classification Performance

4.6 Summary

This chapter presented different steps of image processing that were performed in this work. Firstly, we manually created the patches by selecting the eyes, nose and mouth. These patches were then combined to form one image; this was to ensure that we had sufficient information to describe a face at a specific pose. Following this step pre-processing was performed, to remove interfering noise on the image. Secondly, features were extracted with different scale values using the SIFT algorithm. The extraction of these features at different scale values was done to determine the scale value that gives us better classification of our features. Our observation on feature extraction was that we could not extract enough features with the original octave and level of 4 and 5 respectively. So, we changed the octave and level values to 3. Most of the features with low contrast and features along the edges were eliminated. Even though SIFT extracts normalised features, we obtained a low classification accuracy. The next step was to normalise the features using linear scaling to unit variance.

Upon feature normalisation, the features were trained using SVM, decision trees, kNN and neural network pattern recognition. At sigma $(\sigma) = 3$, for both kNN and neural network pattern recognition a classification accuracy percentage of 98,8% and 78% was obtained respectively. This means our features performed a better classification at $\sigma = 3$ on these classifiers. Furthermore, $\sigma = 6$ decision trees appeared to give us a better classification accuracy, as we obtained 96.8% classification accuracy. Lastly, at the higher sigma value, where $\sigma = 9$, the SVM obtained a 72% classification rate, which was the highest classification rate we obtained at a high sigma. Having trained all the features, we observed that the classifiers perform better at a lower sigma value, in our case where $\sigma = 3$.

The next section will give the summary of our study, suggestions for future work and concludes the research done.

Chapter 5

Summary, future work and conclusion

This section gives the overall summary of the study conducted, followed by possible future study, and lastly a brief conclusion.

5.1 Summary

Firstly, we reiterate that the main goals of this research work were:

1. To implement a classification model that would classify a still input face image according to different poses.
2. To determine a classifier that gives high classification accuracy on patch-based face images.

Having considered our research goals, which ultimately lead to solving research questions which posed in the first chapter, we trained four classification models. Below is a summary of the accomplishment of our research goals, starting with the first research goal:

- Firstly, we manually collected face images in four different poses ($\pm$ 22,5° and $\pm$ 45°). From these images we manually selected landmarks (eyes, nose and mouth) which we called the patches. These patches were combined to form one image of a single subject (automated), in this way we would obtain a compact representation with less computational power.

- The next steps were all automated, with pre-processing of patches being performed so as to remove noise and to rescale the patches to the same patch size.
- Thereafter, the SIFT features were extracted from these patches at different scale spaces (3, 6, 9) so as to observe the performance of our classifiers.
- The next step was to normalise the features to ensure standardised feature vectors.
- Four machine learning models based on SVM, k-NN, decision trees and neural network pattern recognition were trained to test the accuracy of the extracted features.
- The overall classification results obtained by using the Hamming distance metric in the kNN classifier, obtained a classification rate of 98,8%. We then conclude that k-NN is the suitable machine learning algorithm that can accurately classify a face pose at $\pm$ 22,5° and $\pm$ 45°.

The classification results in [9] were not fully discussed, but it was mentioned that the classification algorithm that was used improved the performance of their normalisation model. In the literature discussed in chapter 2, there was no face classification model implemented before recognition of faces was done. We therefore believe that pose classification models are an important models to be integrated in face recognition models.

5.2 Future work

The following is recommended for further study based on the findings from this study:

1. Lateral pose variations as opposed to vertical pose variation and combinations.
2. Effects of lighting, resolution and age.
3. To further explore the features that were extracted to determine a function that will transform a non-frontal face image into a frontal face image for face recognition in different viewpoints.

5.3 Conclusion

The results of this research work show that all the classifiers performed differently, and also for some classifiers performance is highly dependent on the training data set. For example, a NN with more data set performs better classification. Nonetheless, k-NN gave good results irrespective of the size of the data set. In this research work, we demonstrated that with k-NN at $\sigma = 3$ we achieved a better face classification rate of 98% for pose invariant features. At $\sigma = 9$ SVM performed worst obtaining a classification rate of 72%. Our conclusion is that pose invariant feature extraction for the images used on this research work gives better classification at $\sigma = 3$ using kNN to train the features. These findings will help towards recognition of faces in different poses for real world applications. The ability of an FRT to classify images in different poses will also help improve the accuracy in classification.

Bibliography

[1] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–2041, 2006.

[3] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.

[4] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2): 210–227, 2009.

[5] Changxing Ding and Dacheng Tao. A comprehensive survey on pose-invariant face recognition. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(3):37, 2016.

[6] Xiaozheng Zhang and Yongsheng Gao. Face recognition across pose: A review. *Pattern Recognition*, 42(11):2876–2896, 2009.

[7] Xiaoyang Tan, Songcan Chen, Zhi-Hua Zhou, and Fuyan Zhang. Face recognition from a single image per person: A survey. *Pattern recognition*, 39(9):1725–1745, 2006.

[8] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[9] Huy Tho Ho and Rama Chellappa. Pose-invariant face recognition using markov random fields. *IEEE transactions on image processing*, 22(4):1573–1584, 2013.

[10] Michael P Ekstrom. *Digital image processing techniques*, volume 2. Academic Press, 2012.

[11] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision*, 3(3):177–280, 2008.

[12] P Jonathon Phillips, Hyeonjoon Moon, Syed A Rizvi, and Patrick J Rauss. The feret evaluation methodology for face-recognition algorithms. *IEEE Transactions on pattern analysis and machine intelligence*, 22(10):1090–1104, 2000.

[13] Nannan Wang, Xinbo Gao, Dacheng Tao, Heng Yang, and Xuelong Li. Facial feature point detection: A comprehensive survey. *Neurocomputing*, 2017.

[14] Timothy F Cootes, Christopher J Taylor, David H Cooper, and Jim Graham. Active shape models-their training and application. *Computer vision and image understanding*, 61(1):38–59, 1995.

[15] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.

[16] Timothy F Cootes, Gavin V Wheeler, Kevin N Walker, and Christopher J Taylor. View-based active appearance models. *Image and vision computing*, 20(9):657–664, 2002.

[17] Soma Biswas, Gaurav Aggarwal, Patrick J Flynn, and Kevin W Bowyer. Pose-robust recognition of low-resolution face images. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):3037–3049, 2013.

[18] Dong Li, Huiling Zhou, and Kin-Man Lam. High-resolution face verification using pore-scale facial features. *IEEE transactions on image processing*, 24(8):2317–2327, 2015.

[19] Zhenyao Zhu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning identity-preserving face space. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 113–120, 2013.

[20] Meina Kan, Shiguang Shan, Hong Chang, and Xilin Chen. Stacked progressive auto-encoders (spae) for face recognition across poses. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1883–1890, 2014.

[21] Simon Prince, Peng Li, Yun Fu, Umar Mohammed, and James Elder. Probabilistic models for

inference about identity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34 (1):144–157, 2012.

[22] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[23] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[24] Luo Juan and Oubong Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009.

[25] Cong Geng and Xudong Jiang. Sift features for face recognition. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 598–602. IEEE, 2009.

[26] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[27] Konstantinos G Derpanis. The harris corner detector. *York University*, 2004.

[28] Xiaojin Zhu. Semi-supervised learning literature survey. 2005.

[29] Milan Sonka, Vaclav Hlavac, and Roger Boyle. Image pre-processing. In *Image Processing, Analysis and Machine Vision*, pages 56–111. Springer, 1993.

[30] Seishi Okamoto and Nobuhiro Yugami. Effects of domain characteristics on instance-based learning algorithms. *Theoretical Computer Science*, 298(1):207–233, 2003.

[31] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[32] Yongheng Zhao and Yanxia Zhang. Comparison of decision tree methods for finding active objects. *Advances in Space Research*, 41(12):1955–1959, 2008.

[33] Raúl Rojas. *Neural networks: a systematic introduction.* Springer Science & Business Media, 2013.

[34] Guoqiang Peter Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.

[35] Luzia Gonçalves, Ana Subtil, M Rosário Oliveira, and PZ Bermudez. Roc curve estimation: An overview. *REVSTAT–Statistical Journal*, 12(1):1–20, 2014.

[36] JV Candy and EF Breitfeller. Receiver operating characteristic (roc) curves: An analysis tool for detection performance. *LLNL Report: LLNL-TR-642693*, 2013.

[37] Ramzi Abiantun, Utsav Prabhu, and Marios Savvides. Sparse feature extraction for pose-tolerant face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 36(10):2061–2073, 2014.

[38] Soma Biswas, Gaurav Aggarwal, Patrick J Flynn, and Kevin W Bowyer. Pose-robust recognition of low-resolution face images. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):3037–3049, 2013.

[39] Yan Yan, Elisa Ricci, Ramanathan Subramanian, Oswald Lanz, and Nicu Sebe. No matter where you are: Flexible graph-guided multi-task learning for multi-view head pose classification under target motion. In *Proceedings of the IEEE international conference on computer vision*, pages 1177–1184, 2013.

[40] Jeet Kumar, Aditya Nigam, Surya Prakash, and Phalguni Gupta. *Pose Invariant Face Recognition*. PhD thesis, Indian Institute of Technology Kanpur, 2011.

[41] Yizhe Zhang, Ming Shao, Edward K Wong, and Yun Fu. Random faces guided sparse many-to-one encoder for pose-invariant face recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2416–2423, 2013.

[42] Simon JD Prince, James H Elder, Jonathan Warrell, and Fatima M Felisberti. Tied factor analysis for face recognition across large pose differences. *IEEE Transactions on pattern analysis and machine intelligence*, 30(6):970–984, 2008.

[43] Changxing Ding, Jonghyun Choi, Dacheng Tao, and Larry S Davis. Multi-directional multi-level dual-cross patterns for robust face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 38(3):518–531, 2016.

[44] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[45] M Milton Joe and Dr B Ramakrishnan. A survey of various security issues in online social networks. *International Journal of Computer Networks and Applications*, 1(1):11–14, 2014.

[46] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1397–1409, 2013.

[47] Lindsay I Smith et al. A tutorial on principal components analysis. *Cornell University, USA*, 51 (52):65, 2002.

[48] Ryan Rifkin. Multiclass classification. *Lecture Notes, Spring08. MIT, USA*, page 59, 2008.