Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2019-4739418

M.Sc. Thesis

Face recognition using traditional machine learning algorithms and deep neural networks with application to face verification

Manolis Papadakis

Abstract

Biometrics authentication has been very useful and necessary nowadays due to the great developments in technology and the transaction of huge amounts of sensitive data on a daily basis. Traditionally, access to some data or service is achieved by means of some documents or a password. However, these methods are not very convenient. Alternatively, typical biometric systems can be employed that use fingerprint, iris, voice, face recognition or a combination of them. This project focuses on the task of face recognition from still images and investigates how different algorithms for face verification perform under various adverse conditions modelled by blur, salt-and-pepper noise and changes in illumination. Conventional pattern recognition algorithms are first presented. Pixel intensities, Gabor features, Local Binary Patterns (LBP) and 2D-DCT coefficients are considered as features while for classification the nearest neighbor (NNC), nearest mean (NMC), SVM classifiers, and Likelihood Ratio Tests (LRT) with Gaussian Mixture Models (GMM) are examined. Out of all these methods, Gabor features combined with the linear SVM classifier are shown to produce best results across all degradations giving an average Equal Error Rate (EER) of 0.97% using the ORL face dataset. Then, emphasis is placed on deep learning and Convolutional Neural Networks (CNN). Specifically, VGG-Face with triplet loss training for face verification is suggested. VGG-Face achieves an average EER of 2.63% when both test images of a query image pair are drawn from the same degradation conditions and an average EER of 3.80% when only one image in the given pair is degraded and the other one is derived from the clean ORL dataset. We also experimented with the extracted VGG-Face features and NNC, linear SVM and Gaussian SVM and it is seen that a linear SVM gives an average EER of 1.10% by macro-averaging the Detection Error Tradeoff (DET) curves.

Face recognition using traditional machine learning algorithms and deep neural networks with application to face verification

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Manolis Papadakis
born in Athens, Greece

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Face recognition using traditional machine learning algorithms and deep neural networks with application to face verification"** by **Manolis Papadakis** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 26/09/2019

Chairman: _____
prof.dr.ir. R. Heusdens

Advisors: _____
prof.dr.ir. R. Heusdens

_____
dr. N. Gaubitch

Committee Members: _____
dr. F.R.S.T. D.Tax

_____

Abstract

Biometrics authentication has been very useful and necessary nowadays due to the great developments in technology and the transaction of huge amounts of sensitive data on a daily basis. Traditionally, access to some data or service is achieved by means of some documents or a password. However, these methods are not very convenient. Alternatively, typical biometric systems can be employed that use fingerprint, iris, voice, face recognition or a combination of them. This project focuses on the task of face recognition from still images and investigates how different algorithms for face verification perform under various adverse conditions modelled by blur, salt-and-pepper noise and changes in illumination. Conventional pattern recognition algorithms are first presented. Pixel intensities, Gabor features, Local Binary Patterns (LBP) and 2D-DCT coefficients are considered as features while for classification the nearest neighbor (NNC), nearest mean (NMC), SVM classifiers, and Likelihood Ratio Tests (LRT) with Gaussian Mixture Models (GMM) are examined. Out of all these methods, Gabor features combined with the linear SVM classifier are shown to produce best results across all degradations giving an average Equal Error Rate (EER) of 0.97% using the ORL face dataset. Then, emphasis is placed on deep learning and Convolutional Neural Networks (CNN). Specifically, VGG-Face with triplet loss training for face verification is suggested. VGG-Face achieves an average EER of 2.63% when both test images of a query image pair are drawn from the same degradation conditions and an average EER of 3.80% when only one image in the given pair is degraded and the other one is derived from the clean ORL dataset. We also experimented with the extracted VGG-Face features and NNC, linear SVM and Gaussian SVM and it is seen that a linear SVM gives an average EER of 1.10% by macro-averaging the Detection Error Tradeoff (DET) curves.

Preface

The thesis "Face recognition using traditional machine learning algorithms and deep neural networks with application to face verification" has been written to obtain the degree of Master of Science at TU Delft. The project was done together with Pindrop, a phone anti-fraud and authentication technology company, between December 2018 and August 2019. It was supervised by prof.dr.ir. R. Heusdens from TU Delft and dr. N. Gaubitch from Pindrop.
An introduction to the problem of face recognition alongside with some definitions, performance metrics and discussion of some baseline algorithms for recognition from still images and video sequences was priorly done as part of an extra project.

Acknowledgments

First, I would like to thank my supervisors, dr. N. Gaubitch and prof.dr.ir R.Heusdens for their trust giving me the opportunity to work for my Msc Thesis in such an interesting subject. Our discussions during our meetings and their advices were very important for fulfillling this assignment. Nick was trying to give new ideas throught this period suggesting also some papers that were worth reading and contributed to this work. Also, Richard made me always ask myself about the correctness of the various approaches and helped me develop my critical thinking skills.

I would also like to thank Antoon Frehe for giving me access to the servers of the Circuits and Systems Group and helping me with his expertise on some difficulties I had.

I can't miss the opportunity to thank the Lilian Voudouri Foundation for honouring me with a scholarship that really encouraged me financially these two years.

Finally, I would like to thank my friends I met here in the Netherlands for our discussions and the good times we had together and most importantly my family who were always supporting me and encouraging me at every step of my life.

Contents

List of Figures

List of Tables

Nomenclature

γ	kernel coefficient for the Gaussian SVM classifier
λ	L_2 regularization coefficient
μ	mean value of a performance metric
σ	standard deviation of a performance metric
C	regularization parameter for the SVM classifier
K	number of images per class present in a random batch for triplet loss training
L	dimensionality of faces' (or digits') embeddings for triplet loss training
n	learning rate of an optimizer for deep learning
P	number of classes present in a random batch for triplet loss training

CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DET	Detection Error Tradeoff
DNN	Deep Neural Network
EER	Equal Error Rate
FNR	False Negative Rate
FPR	False Positive Rate
GMM	Gaussian Mixture Model
HTER	Half Total Error Rate
LBPH	Local Binary Patterns Histograms
LDA	Linear Discriminant Analysis
MLP	Multilayer Perceptron
NMC	Nearest Mean Classifier
NNC	Nearest Neighbor Classifier
PCA	Principal Component Analysis
RBF	Radial Basis Function
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TPR	True Positive Rate
UBM	Universal Background Model

1 Introduction

Person authentication (verification) has nowadays become an integral part of most technologies due to the extensive transfer of sensitive data across the Internet which necessitates finding ways to distinguish between end users. Traditionally, this has been achieved by token-based identification systems which may require the person's id or passport and knowledge-based identification systems which may require a password. However, these methods have their own drawbacks; token-based systems require the user to bring the necessary documents while password-based identification can be intrusive since the number of applications that require a password may be extremely large and therefore difficult for a person to remember all of them. Biometric systems, on the other hand, are based on measuring traits that are supposed to be unique for each person like fingerprints, voice, face images, iris etc.

Face recognition refers to the task of identifying or verifying a person from a still digital image or a sequence of image frames, that is video. It can operate in either or both two modes:

- Face verification (or authentication): This task involves a one-to-one match that compares a query face image against a template face image whose identity is being claimed.
- Face identification: This task involves one-to-many matches that compare a query face image against all the template images in the database to determine the identity of the query face.

Face recognition, nowadays, has found many applications. To begin with, some businesses let customers take a picture of themselsves in order to comfirm the payment and such applications can of course facilitate the process for users (since not any password is required) and discourage credit card thefts as well. Moreover, some consumer electronics companies, e.g. Apple, Samsung and Xiaomi Corp, have already installed technologies in their mobile phones that allow users to log in without any password but by just looking at the device. It can also be used for security reasons, for example from a company or an organization that handles sensitive data and needs to keep tight controls on who enters their facilities. Furthermore, face recongition can be used to identify criminals. For example, the FBI is using machine learning to identify suspects from their drivers licenses and in addition top US airports apply face recognition to prevent terrorist attacks. Last but not least, businesses can better meet customers needs and show specific advertisement comment by recognizing face demographics.

All the above-mentioned applications indicate why biometrics and specifically face recognition is very important nowadays and why this field has drawn such research interest lately.

1.1 Research Statement and Outline

This project investigates face authentication using traditional, hand-crafted feature based, machine learning algorithms and deep convolutional networks. This technology, as explained before, tries to identify if the shown person in a picture is the claimed identity (commonly referred to as genuine) or a different person (impostor).

Compared to other biometric systems, face recognition encounters some difficulties due to the large

number of variations in face images. Humans can have various expressions, different poses or orientations; also occluding items and different illumination conditions between enrollment and test phase may degrade the recognition performance.

State of the art results in this field are obtained with Deep Learning, where large datasets and powerful computation resources are needed([3], [4], [5], [6]). Traditional machine learning algorithms lag behind because of the need of extracting features that are as much invariant to those variabilities found in face images as possible. However, parts-based face representation with 2D DCT coefficients([7], [8]) or Gabor features([9], [10], [11]) can also give good results.

In this project, some selected traditional and deep learning based algorithms will be considered and tested against various image degradations as it is common that images are captured in different conditions at enrollment and validation times. Specifically, the significance of salt-and-pepper noise, blur and the factor of illumination will be examined and the role they play in face verification. It is deemed that the factor of illumination is the most intrusive one and that deep learning models will be more robust to such degradations due to the availabity of large datasets that help them capture useful features even in images from uncontrolled environments.

The thesis is structured as follows: Chapter 2 introduces some traditional face recognition algorithms that have been mainly used in past years (before the thrive of deep learning). Chapter 3 gives some insight in deep neural networks with more focus on architectures that have been shown great success for the face recognition problem. Chapter 4 presents some common face databases that can be used for training and evaluating face recognition systems. In chapter 5, experiments and results are shown with AT&T, VidTIMIT, and LFW benchmarks. Finally, conclusions and also some suggestions for future work are included in chapter 6.

Traditional Machine Learning Algorithms for Face Recognition

2

Since face recognition can be formulated as a classification task, various machine learning algorithms can be applied in order to construct a robust method that automatically infers the identity of the person. Every machine learning algorithm takes a dataset as input and learns from this data. After encoding every instance/sample (in our specific case, this is an image) with a feature vector, a learning algorithm goes through the data and identifies patterns.

The intrusive factors of pose, expresssion and illumination render the selection of an appropriate (rich, with many within-class variations) dataset an important task if we wish our system to be robust to these variations. In addition, two other issues are important as well; the first one is what features to use to represent a face so as to be as robust as possible to all these variations. The second one is how to classify a new face image using the chosen representation.

In this chapter, we will first give a literature review of face recognition techniques including some preprocessing steps that should be taken before feature extraction and classification algorithms take place. Then we will dive into some methods that have been widely applied and have shown good results.

2.1 Traditional Face Recognition Methods: A Review

In reality, face recognition doesn't start with extracting unique features of the face that can be used to distinguish that person from the others. First, all faces in the image have to be detected and extracted. This step is called face detection and is typically the first step of a face recognition system (see Fig. 2.1).

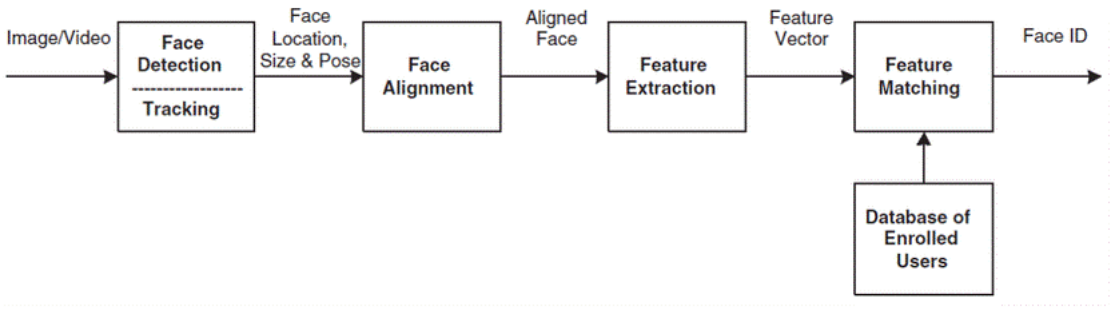


Figure 2.1: Face recognition pipeline [1]

A very popular approach for face detection that was fast enough to work in real time was introduced by Viola and Jones in 2004, when they extracted Haar features and implemented a cascade of Adaboost

classifiers [12]. The main ideas of their algorithm were the use of integral images for fast computation of Haar features, the use of boosting for selecting the most relevant features (robust feature selection) and the use of cascading to easily discard the non-face regions with simple tests. Another robust method for face detection is the Histogram of Oriented Gradients (HOG) method [13]. As the name suggests, the feature vector of a given image patch consists of the concatenation of HOGs of various subregions (cells) and this is fed to a linear SVM classifier to perform the classification face vs non-face. This HOG-SVM face detector is more accurate than Haar cascades reducing false positive rates and it is also slightly faster. Instead of Haar features, Local Binary Patterns (LBP) can also be combined with cascade boosted classifiers [14]. Currently, state-of-the-art results for face detection are obtained with deep learning. YOLO (You only look once) [15] is a deep CNN for object detection and based on its architecture some face detectors have emerged. However, such detectors may be quite slow.

Another prepreprocessing step that can boost the face recognition performance is face alignment so that key facial landmarks like the eyes and the mouth are centered and roughly in the same position of the image. Most of them are based on detecting some facial landmarks like mouth, eyes, eyebrows, nose and jaw and then rotating, translating and scaling to obtain a normalized representation of the face. A facial landmark detector that successfully detects 68 facial landmarks and therefore can be exploited for face alignment is suggested in [16]. Given some training face images annotated with the position of facial landmarks and also given the prior information of distances between these facial landmarks, an ensemble of regression trees is trained by gradient boosting to estimate the landmark positions using only the pixel intensity information. Gradient boosting in an instantiation of the idea of boosting for regression problems. Another method for aligning faces is described in [17] which uses the idea of congealing. That tries to minimize the entropy of the empirical distribution field at each pixel.

After the geometric normalization, the illumination normalization step can take place. As face images of the same person look very different under different illumination conditions, it becomes essential to compensate for them and make images more similar. Some commonly used methods are histogram equalization which is used to enhance the contrast and make the histogram more uniform and the normalization introduced by Tan and Triggs [18] that consists of three steps: gamma correction, difference of Gaussians filtering and contrast equalization.

After faces have been detected, extracted and normalized feature extraction takes place. Face recognition algorithms can be divided into three categories based on what features they extract [1]; feature-based, holistic and hybrid methods.

Feature-based approaches [19],[20] detect and extract facial landmarks like eyes, nose and mouth and measure their geometrical properties and distances between them. These features are invariant to changes in illumination and pose. However, detection of such landmarks is not that reliable reducing the efficiency of those approaches. Moreover, they ignore facial texture which could be also used to distinguish between faces.

Holistic methods consider the input face image as a whole entity and try to extract useful statistical information from the input data. Holistic methods dominated early in face recognition before hybrid methods emerged that used local-based face descriptors in one global feature vector and are next discussed in more detail. Some examples of this class of algorithms are Eigenfaces, Fisherfaces, Independent Component Analysis (ICA) and kernel methods. Both Eigenfaces [21] and Fisherfaces methods [22] are based on pixel intensity features and dimensionality reduction techniques. The first method is based on Principal Component Analysis (PCA), an unsupervised dimensionality reduction technique that preserves maximum variability and also decorrelates features. The latter method is based on Linear Discriminant Analysis (LDA), a supervised dimensionality reduction technique that finds the projection that best discriminates the classes by optimizing the Fisher's discriminant ratio. The method of Eigenfaces is very simple but it's very sensitive to the variations of pose, expression

and lighting encountered in unconstrained contexts while the method of Fisherfaces is more robust to lighting and pose variations by also using the class label information. In [23], ICA is applied for dimensionality reduction which tries to find the projection that makes features not only uncorrelated but independent as well achieving better results for face recognition than PCA. In [24], the kernel trick is applied in PCA and LDA transforms to find a better nonlinear transform yielding kernel PCA and kernel LDA methods that outperform Eigenfaces, Fisherfaces and ICA.

Hybrid methods on the other hand use local block-based features which makes them more robust to the factors of pose and illumination. Local Binary Patterns Histograms, Gabor features and 2D DCT coefficients are such examples of face descriptors.

The Local Binary Pattern (LBP) operator is a powerful texture descriptor that can be applied for face recognition since faces consist of various micropatterns [25]. It was originally designed for a fixed 3×3 scale and some extensions have been suggested in [26]. Feature vectors are then formed by concatenating histograms of such patterns from various regions.

For classification, simple classifiers like the nearest neighbor classifier, NNC (or the relevant nearest mean classifier, NMC) can be applied. These distance-based classifiers can work with Euclidean distances [21], [22], cosine similarity measures [23] or (weighted) χ^2 distances in case of histograms-based features. Support Vector Machine (SVM) classifier [27] is another more complex algorithm. Originally designed for binary classification problems, SVM aims to find the hyperplane that has the largest separation or margin between the two classes. In case data is not linearly separable, soft-margin SVM that trades-off the number of misclassifications and maximum margin can be applied or alternatively kernel SVM with custom kernels like gaussian, hyperbolic tangent, polynomial etc. can be used to find useful non-linear decision surfaces [28].

In the next section, two other local-based face descriptors that have been shown to produce great results in face recognition literature are discussed; Gabor-based features and 2D DCT coefficients.

2.2 Parts-based Face Recognition

2.2.1 Gabor features

Gabor filters have been successfully applied in many image processing tasks, such as image smoothing, texture analysis, edge detection, iris and fingerprint recognition and face recognition. Gabor filters are known to have optimal resolution in both time (space) and frequency domains. A Gabor filter is a modulated Gaussian by a complex exponential which in 2D is given by:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \exp(i(2\pi \frac{x'}{\lambda} + \psi)) \tag{2.1}$$

where $x' = xcos\theta + ysin\theta$, $y' = -xsin\theta + ycos\theta$, θ represents the orientation of the normal to the parallel stripes of the Gabor function, σ is the standard deviation of the Gaussian envelope, γ is the spatial aspect ratio that controls the ellipticity of the ellipses, λ is the wavelength of the sinusoidal factor and ψ is the phase offset.

In [9], a filterbank of odd-symmetric Gabor filters (the complex exponential in formula (2.1) reduces to a sine) of various orientations and frequencies was applied to obtain the feature vectors and classification was performed by a simple nearest mean classifier. In [10], the authors tried to combine magnitude and phase information of Gabor filtered images to obtain a richer feature representation

of the face image compared to previous research efforts that were focusing only on the magnitude information. The number of features was reduced by PCA before using SVM for the classification task.

2.2.2 Two-dimensional Discrete Cosine Transform

Two-dimensional Discrete Cosine Transform (2D-DCT) has also been applied in face recognition and preferred over DFT due to its nice compression properties. DCT features are easier to compute compared to the alternative feature representation of Gabor features while also being more robust to illumination changes when combined with polynomial coefficients also known as deltas computed from neighboring blocks [7]. After the face is detected and optionally geometrically and illumination normalized (with some preprocessing method like histogram equalization or Tan and Triggs normalization) block-based DCT can be performed for feature extraction. The face image is divided into overlapping blocks of size $M \times N$ and feature vectors are extracted in each block by keeping a small number of low-frequency DCT-II coefficients with a zig-zag technique as shown in Fig. 2.2. The DCT coefficients of a block of image $f(x, y)$ with size $M \times N$ are given by:

$$C(u,v) = a_x a_y \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) \cos(\frac{(2y+1)v\pi}{2M}) \cos(\frac{(2x+1)u\pi}{2N}) \tag{2.2}$$

where $u = 0, \ldots, N-1$, $v = 0, \ldots, M-1$, $a_x = \begin{cases} \sqrt{\frac{1}{N}} & x = 0 \\ \sqrt{\frac{2}{N}} & x > 0 \end{cases}$ and similar expression hold for a_y.

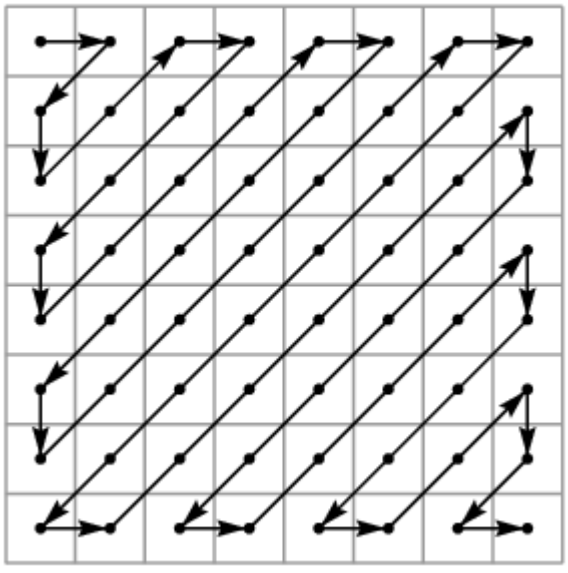


Figure 2.2: Zig-zag scanning technique for extraction of DCT coefficients

Then, the resulting feature vectors can either fit with Gaussian Mixture Models (GMMs) and likelihood ratio tests ([7],[8]) or used by common pattern recognition classifiers like NNC and SVM after a possible dimensionality reduction step for example with PCA.

2.2.3 Face verification with Gaussian Mixture Models

One interesting approach with Gaussian Mixture Models (GMMs) that contributed to our work can be found in [8] and [29]. In [29], the principle is the same with our work with the difference that it refers to speaker authentication and therefore the features used there are Mel-frequency cepstral coefficients (MFCCs), while [8] refers explicitly to the face verification task.

A face verification system, similarly to a speaker verification system, can be casted as a binary hypothesis test deciding one out of the following hypotheses:

- H_0: input face image part $\mathbf{x}$ is from the target subject S (null hypothesis)
- H_1: input face image part $\mathbf{x}$ is not from the target subject S (alternative hypothesis)

Having determined the likelihoods $p(\mathbf{x}|H_0)$ and $p(\mathbf{x}|H_1)$, classification is performed by the likelihood ratio test (LRT) as follows:

$$\frac{p(\mathbf{x}|H_0)}{p(\mathbf{x}|H_1)} = \begin{cases} \geq \theta & \text{accept } H_0 \\ < \theta & \text{reject } H_0 \end{cases} \tag{2.3}$$

Using the assumption of i.i.d. samples, a sequence (face image) $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_K})$ can be classified as "genuine" or "impostor" with respect to the target S, as follows:

$$\begin{aligned} &\log \frac{p(\mathbf{X}|H_0)}{p(\mathbf{X}|H_1)} = \log \frac{\prod_{i=1}^{K} p(\mathbf{x_i}|H_0)}{\prod_{i=1}^{K} p(\mathbf{x_i}|H_1)} = \\ &= \sum_{i=1}^{K} (\log(p(\mathbf{x_i}|H_0) - \log(p(\mathbf{x_i}|H_1))) \begin{cases} \geq \theta' & \text{accept } H_0 \\ < \theta' & \text{reject } H_0 \end{cases} \end{aligned} \tag{2.4}$$

where K is the number of (generally overlapping) blocks or feature vectors that characterize the entire image.

Gaussian mixture models have been suggested in the literature to model the class-conditional probability density functions (or likelihoods) due to their simplicity.

A GMM is given by:

$$p(\mathbf{x}) = \sum_{i=1}^{C} P_i N(\mathbf{x}|\boldsymbol{\mu}_i, \Sigma_i) \tag{2.5}$$

where C is the number of uni-modal Gaussian components, P_i are the weight coefficients (prior probabilities) of the components and $N(\boldsymbol{x}|\boldsymbol{\mu}_i, \Sigma_i)$ is the uni-modal Gaussian density with mean $\boldsymbol{\mu}_i$ and covariance Σ_i (which is usually considered to be diagonal) given by:

$$N(\mathbf{x}|\boldsymbol{\mu}_i, \Sigma_i) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i)) \tag{2.6}$$

2.2.3.1 Universal Background Model (UBM) and Mean-Only MAP adaptation

The model that describes all the alternatives to the target subject is called Universal Background Model (UBM) and the parameters $\lambda_{\text{UBM}} = \{P_i^{\text{UBM}}, \boldsymbol{\mu}_i^{\text{UBM}}, \Sigma_i^{\text{UBM}}\}_{i=1}^{C}$ are learned via Expectation-Maximization [30] which is an iterative method that finds the maximum likelihood estimates of the model parameters when data has missing values or the model has hidden (latent) variables (like in the case of GMM models). The number of mixtures C can be determined as the one than optimizes a model criterion, for example Bayesian Information Criterion (BIC) or can be determined by observing how the face recognition system performs on unseen data (validation set). Instead of independently fitting the target model λ_{target} to a number of training samples coming from S, a better idea is to adapt the well-trained UBM (in the sense that is fitted on much more data) to the target model using a Maximum A Posteriori (MAP) adaptation of the UBM. It has been shown that a mean-only adaptation that involves only the means of the UBM being adapted to the target model is effective for both speaker [29] and face verification [8].

The equations needed to perform mean-only MAP target adaptation are presented below:

$$\boldsymbol{\mu}_i^{\text{target}} = \alpha_i \boldsymbol{\Delta\mu}_i^{\text{target}} + (1 - \alpha_i)\boldsymbol{\mu}_i^{\text{UBM}} \tag{2.7}$$

$$P_i^{\text{target}} = P_i^{\text{UBM}}, \Sigma_i^{\text{target}} = \Sigma_i^{\text{UBM}} \tag{2.8}$$

where $\boldsymbol{\Delta\mu}_i^{\text{target}} = \frac{\sum_{j=1}^{N_{\text{target}}} P_{\text{UBM}}(i|\boldsymbol{x}_j)\boldsymbol{x}_j}{\sum_{j=1}^{N_{\text{target}}} P_{\text{UBM}}(i|\boldsymbol{x}_j)}$ and $\alpha_i = \frac{\sum_{j=1}^{N_{\text{target}}} P_{\text{UBM}}(i|\boldsymbol{x}_j)}{\tau + \sum_{j=1}^{N_{\text{target}}} P_{\text{UBM}}(i|\boldsymbol{x}_j)}$

Note that the weight parameters P_i and the covariance matrices Σ_i are preserved.

The fixed parameter τ affects the data-dependent mixing coefficents α_i which control the tradeoff between the new statistics $\boldsymbol{\Delta\mu}_i^{\text{target}}$ and the old ones $\boldsymbol{\mu}_i^{\text{UBM}}$. This adaptation provides a better coupling between the UBM and the target model and enables a fast computation of the log-likelihood ratio since only a few components are significant from both models for a given feature vector. It has been empirically found that $\tau = 16$ achieves good results.

This mean-only MAP adaptation of the UBM to the model of client i, can be written in compact form as:

$$\boldsymbol{s}_i = \boldsymbol{m} + \boldsymbol{d}_i \tag{2.9}$$

where $\boldsymbol{m}$ is the UBM mean supervector which is simply the concatenation of the means of the components, $\boldsymbol{m} = [\boldsymbol{\mu}_1^{\text{UBM}}, \boldsymbol{\mu}_2^{\text{UBM}}, \ldots, \boldsymbol{\mu}_C^{\text{UBM}}]^T$, $\boldsymbol{d}_i$ is the subject-dependent offset calculated as $\boldsymbol{d}_i = D\boldsymbol{z}_i$, where $D = \sqrt{\frac{\Sigma}{\tau}}$, Σ is a block diagonal matrix of the covariance matrices of the individual components, $\boldsymbol{z}_i \sim N(0, I)$ and $\boldsymbol{s}_i$ is the resulting client model.

2.2.3.2 Session variability modelling methods: Inter-Session Variability Modelling (ISV), Joint Factor Analysis (JFA) and Total Variability Modelling (TVM)

The beforementioned GMM-UBM approach does not explicitly model the inter-session (or within-client) variability that may occur due to changes in illumination, pose, expression or image acquisition

and can adversely affect the performance of a face recognition system. Two methods that try to improve robustness to session variability by explicitly modelling the within-client variation using a low-dimensional subspace and then excluding its effects to obtain more accurate client models are Inter-session variability modelling (ISV) and Joint factor analysis (JFA) which are described in [31],[32]. Their difference is basically that (JFA) also models the between-class variability hoping to obtain more robust client models in case of limited enrolment data. Details of how latent variables and low dimensional subspaces can be learned in these techniques can be found in [33].

In [31], it was shown that both ISV and JFA for face authentication capture and exclude a significant part of between-class variation. The problem of discarding useful information for classification was addressed by Total Variability Modelling (TVM) or Front-end Factor Analysis where session compensation is not performed in the high dimensional GMM mean supervector space, like in ISV and JFA, but in a low dimensional subspace called "total variability factor space" or "i-space" [34].

3 Deep Face Recognition

Deep neural networks (DNNs) [35] are based on artificial neural networks (ANNs) inspired by their biological counterparts. ANNs consist of many interconnected nodes called neurons. Neurons have some trainable parameters (synaptic weights) and when trained they can find proper representations and solve difficult classification tasks. A typical neural network of 3 layers (input layer, hidden layer, output layer) is shown in Fig. 3.1.

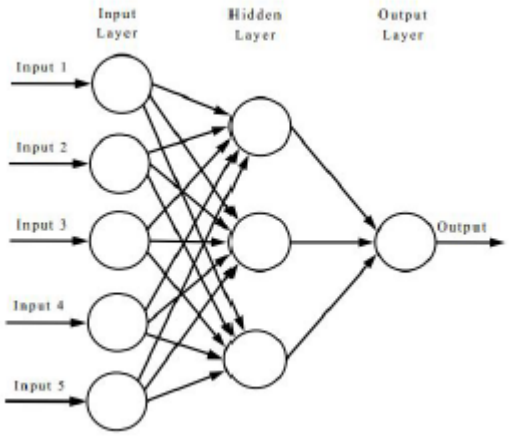


Figure 3.1: A three-layer ANN

Each neuron i in the network takes in an input vector $\boldsymbol{x}_i$, computes the inner product $\boldsymbol{z}_i = \boldsymbol{w}_i^T \boldsymbol{x}_i$, where $\boldsymbol{w}_i$ is its weight vector and finally it passes it through an activation function $h(z)$ which must be differentiable in order for the full network to be trainable with backpropagation [28], the most commonly used learning algorithm in ANN theory.

DNNs can be viewed as ANNs with many hidden layers and have found many applications in computer vision with state-of-the-art results. A feed-forward DNN can be seen as a chain of many functions applied in an hierarchical manner as

$$f(\boldsymbol{x}) = f_L(\dots f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2)\dots); \boldsymbol{w}_L) \tag{3.1}$$

where L is the total number of layers, $\boldsymbol{w}_i$ is the stacked vector of all weights of all neurons in layer i and $\boldsymbol{x}$ is the network's input. The presence of that many layers enables the model to find useful high-level feature representations that discriminate efficiently the various classes. This intrinsic feature extraction implies no need for human intervention (feature engineering can be a difficult task) and this is the reason why Deep Learning has been so useful today. Furthermore, the use of DNNs has been blessed by the large available datasets and the powerful computational resources.

The most widely used DNNs in computer vision problems are the so-called Convolutional Neural

Networks (CNNs) [35] which can be said that they are inspired by the mammalian visual cortex. Deep CNNs contain several layers of processing applying linear and non-linear operations, which are learned end-to-end to solve a particular task. This way, the important discriminative features for the final classification are learned from the abundancy of training data without any human expertise.

The learned features are formed as filters in the various convolutional layers of a CNN which try to detect and highlight specific patters of the input volume in contrast to features used in classical face recognition (like pixel intensities, binary patterns, Gabor features, 2D DCT coefficients) discussed in Chapter 2. As for the classification part, this is performed by a multilayer perceptron, MLP, with a softmax output layer which can learn arbitrarily complex non-linear decision boundaries.

CNNs are used in a variety of applications in computer vision field. From low-level problems, such as denoising, sharpening and edge detection in images, to higher-level problems, such as image classification, face/object recognition and depth estimation, CNNs are constantly finding more and more applications in recent years with great results.

In this chapter, we will describe the architecture of CNNs, some regularization methods and their training procedure and then, we will present some interesting applications of CNNs for face recognition.

3.1 Architecture of CNNs

A CNN typically consists of some non-linear layers for feature extraction and one classifier in the end in the form of a MLP that takes in the features and performs the classification.

CNNs typically consist of 3 types of layers: convolutional layers, pooling layers and fully connected layers.

- Convolutional Layer:

  The convolutional layers consist of neurons that see only a specific subset of the input space (which is called receptive field) and share their weights. As a consequence, the result of the application of that layer is the convolution of the input image with the kernel defined by the weights of a single neuron. The operation of convolution is translation invariant which is desired for many computer vision tasks like image classification. This layer has many filters, hundreds or even thousands, in order to extract many features that may be useful for the classification. The result of the convolution between the input image and one filter is added to a constant (also learnable parameter) b which is known as bias and then it passes through a non-linear activation function $\phi(\cdot)$. The final result of this procedure is called feature map and of course, there are as many feature maps as the number of different filters. Function $\phi(\cdot)$ is usually one of the following:

  1. Rectified linear, $\phi(x) = max(0, x)$
  2. Hyperbolic tangent, $\phi(x) = tanh(x)$
  3. Logistic function, $\phi(x) = \frac{1}{1+\exp(-x)}$

  and ReLU is commonly preferred since it avoids the vanishing gradient problem of the last two functions leading to much faster training while the generalization error remains similar.

- Pooling Layer:

This layer usually follows the convolutional layer and is used to reduce the feature dimensionality and control overfitting. Usually this layer slices the input image into nonoverlapping sections, and it selects the maximum value in each dropping all the others (max pooling). This max pooling is invariant to small shifts or rotations, as the max operator will select the same value under these circumstances. But also other pooling strategies can be used such as average pooling, or L_2 norm pooling.

- Fully Connected Layer:

  The convolutional and pooling layers which were discussed earlier are used for feature extraction and subsampling the feature maps respectively. The fully connected layer is placed after these layers and possibly in cascade to form a MLP (see Fig. 3.1). The MLP aims to implement the main functionality of the neural network, which is usually classification. The input of this layer is a vector of fixed dimensionality and the output, in case of classification, consists of as many neurons as the number of the different classes of the problem. The fully connected layer introduces a weight matrix and vector of biases to be learned during training.

From the above mentioned layers, only the convolutional and fully connected layers introduce adaptable parameters, weights and biases, the optimal values of which are searched during training by minimizing a suitable cost function. This cost function can be the cross-entropy loss for classification tasks or the mean-squared error for regression tasks. Overall, a typical architecture of a CNN can be seen in Fig. 3.2.

The training of CNNs is done in a very similar way as backpropagation in ANNs which is essentially a combination of gradient descent with the chain derivative rule.

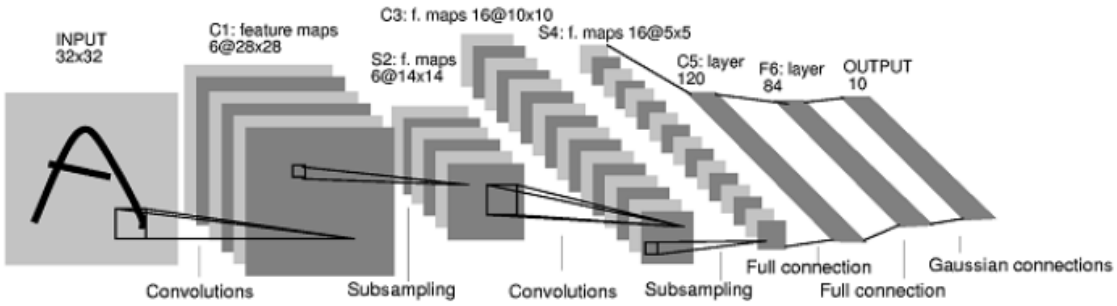


Figure 3.2: An example of a CNN architecture [2]

The hyperparameters of a CNN include the number of filters, the convolutional kernel size and the associated stride introduced by the convolutional layers and the pooling size and the associated stride introduced by the pooling layers. Apart from them, the choice of the learning rate and its schedule, the maximum number of training epochs, the number of layers (how deep the network is) are all crucial as well and their choice greatly affects the final performance of the CNN.

One systematic approach to opt for the optimal values of these hyperparameters is the k-fold cross-validation method. According to this, the training set is split into k equally sized parts (folds) and for each fold $i = 0, \ldots, k-1$ the performance J_{test} of the model is estimated after having been trained using the data from the remaining $k-1$ folds. One common choice for k is $k = 10$. From this experiment, the average test error is calculated. The same procedure is repeated for each candidate value of the relevant hyperparameter and the value that produced the best (lowest) average test error is finally kept.

The advantage of this method is that it results in a good estimate of the optimal value of the hyperparameters even with only a few data. Its disadvantage is that it requires a lot of iterations and thus time, as the network is trained k times. In case training is expensive, data can be split into training and test sets and hyperparameter values can be selected based on the test error.

3.2 Regularization for Deep Learning

Regularization is the process of adapting the training procedure of a learning algorithm in order to combat overfitting. Overfitting happens when the model learns too well the details and the noise of the training data, but it performs very bad on unseen data. Thus, some techniques can be applied to improve the generalization ability of the model with the cost of increased training error. Some common regularization techniques are as follows:

- Dropout

  This method is usually applied after the fully connected layers, which are prone to overfitting due to their large number of parameters. In dropout, at each update during training, the neurons of some fully connected layers are dropped out with probability $1 - p$, or equivalently retained with probability p (usually $p = 0.5$) and as a result a reduced network arises. All the incoming and outcoming edges of a dropped-out neuron are also removed and only the retained neurons are updated. At the next iteration, the neurons that had been previously removed, enter the network again with their previous weights. With dropout, eventually many neural networks are trained because each forward-backward propagation refers to a different network architecture. After training ends, the weights that have been learned are scaled by p, and finally the weights $\hat{w} = pw$ are used to classify a test image $\boldsymbol{x}$. This scaling is done so that the expected output of the neuron be the same as in training stages.

- Data augmentation

  Due to the deep architecture of a CNN, many parameters are introduced for training and therefore an even larger training set is necessary to avoid overfitting. If only a few training examples are availabe, then the size of the training set can be increased either by producing new examples from scratch or perturbing a little bit the already available examples to produce new ones. For example, the transforms of translation, rotation, flip or crop can be applied or even adding noise. Special care should be taken though so that the class of the new image doesn't change compared to the class of the image it was generated from.

- Weight decay

  One simple regularization method is to add to the training error J a term Ω which is a function of the weights of the network multiplied by a coefficient (regularization parameter) λ. This term is desired to be large when the network is large and small when the network is small. The parameter determines the emphasis we place on the regularization of the network: the larger the λ, the more importance is given to reducing the complexity of the network. Usually, Ω is given by the L_2 norm of the weight vector (L_2 regularization), thus:

  $$\Omega = \frac{1}{2} \sum_i w_i^2 \tag{3.2}$$

  This L_2 regularization is also known as weight decay since it makes the weights to decay towards

zero.

- Early stopping

  The general trend is that the training error keeps improving with an increase of the training epochs while the validation error first drops and then rises after some training epochs (see for example Fig. 3.3). Early stopping monitors some metric on the validation data such as the validation accuracy and when this metric stops improving, the training of the network stops. This is done because after that point, the two curves depart from each other which is a sign of overfitting.

Figure 3.3: Early stopping technique

- Max norm constraints

  Another regularization method is to impose an upper bound c on the magnitude of the weight vector of each neuron. This means, that after we update the weights in the normal way, if $\|\boldsymbol{w}\| > c$, we scale $\boldsymbol{w}$ such that $\|\boldsymbol{w}\| = c$. Usually, $c = 3$ or $c = 4$.

3.3 Training CNNs

After the hyperparameters and the type of regularization have been resolved, the training of the CNN takes place. As mentioned before, the learnable parameters are the weights and biases of the neurons in the convolutional and fully connected layers. For training, the backpropagation error algorithm [28] is used. This algorithm combines gradient descent with chain rule to efficiently compute the gradients of the loss w.r.t to all weights and biases and perform the updates. There are three ways of doing gradient descent [36]:

- Online gradient descent

  This algorithm is also called stochastic gradient descent (SGD) and uses only one example at a time to update the parameters. The stochasticity stems from the fact that training instances are usually randomly shuffled at the start of each epoch before they are presented to the network. SGD converges fast (to a local optimum) because it updates frequently the parameters and the noisy estimates of the gradients can help to escape some local minima.

- Batch gradient descent

  This algorithm uses the whole training dataset at once to update the parameters at each iteration (epoch). However, batch gradient descent results in slow convergence and in many cases is intractable since it requires a large number of computations per update and large memory specifications. On the other hand, the gradient vector of the loss with respect to the weight vector is estimated accurately.

- Mini-batch gradient descent

  This algorithm uses a number of training examples which are said to consist a mini-batch to update the model parameters. It combines the robustness and speed of SGD and the more accurate estimates of the error gradients of batch gradient descent. Moreover, this algorithm is benefited by the vectorization libraries of modern deep learning frameworks that enable fast processing of the entire mini-batch at once. This kind of gradient descent is the most popular one in the field of deep learning.

3.3.1 Optimizers for DNNs

The basic optimizer used in deep learning is SGD which can be applied using one example at a time or a minibatch. In the general case that a minibatch of m examples $\{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$ is used, the update in SGD is given by the following equation:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{n}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \tag{3.3}$$

where $\boldsymbol{\theta}$ is the parameter vector consisting of the weights and biases of the trainable layers, t is the iteration step that denotes each batch in various epochs, n is the learning rate (or the step size) that determines how much we tweak our parameters with respect to the gradients, or how much we trust the gradients and $L(\cdot)$ is the loss function.

One variant of SGD is the momentum algorithm [36]. In SGD with momentum, we also take into account previous gradients for the updates by using moving averages. That is, the update rule is given by:

$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} + \frac{n}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \tag{3.4}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{v}^{(t+1)} \tag{3.5}$$

where $\boldsymbol{v}$ is the velocity term and γ is the momentum parameter that determines how strong the effect of past accumulated gradients is on each update.

SGD with momentum can accelerate convergence while dampening oscillations especially in cases where the loss function is much steeper in one dimension than in another. The curvature of a twice differentiable function depends on the eigenvalues of its Hessian matrix and the case that the curvature in one dimension is much larger than the curvature in another can be indicated by a poorly conditioned Hessian matrix. In essence, this optimization algorithm lowers the learning rate in directions that the

loss function doesn't decrease much and conversely increases the learning rate in directions where the loss function decreases significantly.

There also some adaptive optimizers that adapt the learning rate of the model parameters [36]. For example, AdaGrad adapts the learning rates of the parameters by scaling them inversely proportional to the square root of the sum of the past squared values of the gradient. Adam is another adaptive optimizer that smooths the average and the uncentered variance of noisy gradients with exponentially weighted moving averages.

3.4 Applications of CNNs for Face Recognition

Deep face recognition is greatly affected by the breakthroughs of deep learning for the more general task of object classification yielding models that follow the same typical architectures [6].

The first approach, in 2014, was DeepFace developed by Facebook AI research [3]. DeepFace algorithm is a pipeline of 4 stages: detection, 3D alignment, representation and classification.

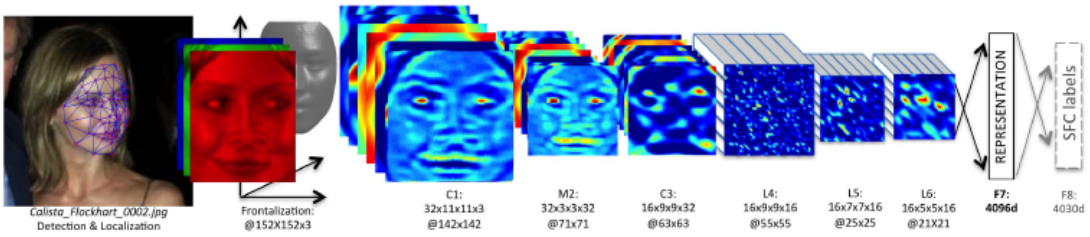


Figure 3.4: The overall architecture of DeepFace [3]

As it is shown in Fig. 3.4, after a convolutional layer with 32 filters, $C1$, a max pooling layer, $M2$ and another convolutional layer with 16 filters, $C3$, 3 locally connected layers are introduced, namely $L4$, $L5$ and $L6$. These 3 layers are similar to the conventional convolutional layers but without weight sharing, and the reason behind this design choice is that in aligned images specific areas have very different properties and larger discrimination abilities compared to others. On the other hand, this choice leads to much more trainable parameters. The last two layers, $F7$ and $F8$ are fully connected. The output of $F7$ gives the raw feature representation of the images, while the output of $F8$ is passed through the softmax function to give the class posterior probabilities and essentially performs face recognition. Finally, the features are normalized by dividing by their largest feature value followed by L_2 normalization to improve the robustness to illumination variations. DeepFace achieved a verification accuracy of 97.35% on the LFW dataset (state-of-the-art at that time) when trained on the Social Face Classification (SFC) dataset which includes 4.4 million labeled faces of 4,030 people each having from 800 to 1200 faces.

Another very successful application of CNNs for face recognition that was developed in 2015 is FaceNet [4]. FaceNet employs a CNN trained to directly measure the similarity among faces in a compact Euclidean space. At the end of the model, the structure of which is displayed in Fig. 3.5, Schroff et al. used the triplet loss with the goal that face embeddings belonging to the same person should be close together and form well-separated clusters. Triplet loss aims to enforce that an image $\boldsymbol{x}_i^a$ (anchor) of a specific person is closer to all other images $\boldsymbol{x}_i^p$ (positive) of the same person than to any image $\boldsymbol{x}_i^n$ (negative) of any other person by a margin α. In other words, the margin or gap α pushes the anchor positive pair and the anchor negative pair further away from each other (Fig. 3.6). Triplet loss uses the idea of Siamese networks in one-shot learning in the sense that the same exactly network is

applied to triplets of images to generate the face encodings (or face embeddings).

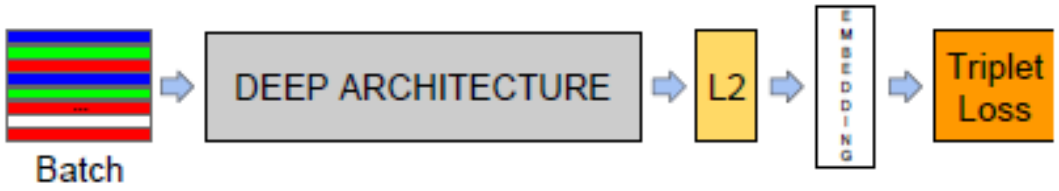


Figure 3.5: The model structure of FaceNet [4]

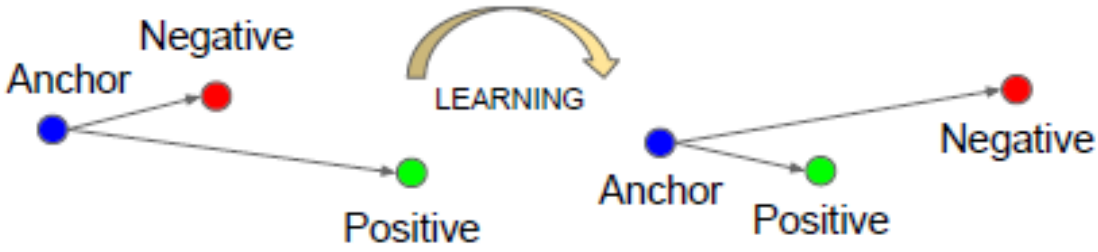


Figure 3.6: The goal of triplet loss [4]

The triplet loss is given by:

$$\text{Loss} = \sum_{i=1}^{N} \left[\|f(\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^p)\|^2 - \|f(\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^n)\| + \alpha \right]_+ \tag{3.6}$$

where N is the size of the triplet training set and the notation of the $+$ sign in the subscript means

$$\left[x\right]_+ = \begin{cases} x, & \text{if} \quad x > 0 \\ 0, & \text{else} \end{cases} \tag{3.7}$$

The selection of triplets is crucial as the performance of the model greatly depends on it. There are generally 2 ways of selecting triplets: offline and online.

The easiest method is offline triplet mining. In offline mining, triplets are produced offline at the beginning of each epoch. First, embeddings of all training images are computed beforehand and triplets are formed by randomly selecting only hard ($\|f(\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^n)\|^2 < \|f\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^p)\|^2$) and semi-hard negatives ($\|f\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^p)\|^2 < \|f(\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^n)\|^2 < \|f(\boldsymbol{x}_i^a) - f(\boldsymbol{x}_i^p)\|^2 + \alpha$) that contribute to the triplet loss for each anchor positive pair (a, p). So for a given batch size B of triplets, $3B$ embeddings are required to compute the loss and backpropagate it to update the weights. This method is not that efficient though since it requires that embeddings of the whole dataset are computed to generate the triplets.

On the other hand, in online triplet mining, triplets are generated on the fly. For a given batch size of face images B, B embeddings are computed and a maximum of B^3 triplets can be generated. However, not all these triplets are valid as they don't consist of two positives and one negative. Online triplet mining is much more efficient than the offline one, since more triplets are generated with less effort.

The researchers in [4] used online triplet mining and selected randomly semi-hard negatives for every (a, p) pair in a batch. They also selected the dimensionality of the embedding to be 128 that is they represented faces with a 128-dimensional vector. Training on 100M-200M images, FaceNet achieved a new record verification accuracy of 99.63% on the LFW dataset.

A third very popular deep learning model for face recognition is VGG-Face [5]. There, the authors designed a procedure that combines a little manual effort with some automatic processing stages to create a large face dataset as it was noted that such datasets are very scarce and not publicly available. The resulting VGG-Face dataset consists of 2,622 subjects and 2.6M images (1000 images of each identity). The architecture of VGG-Face model is based on VGGNet [37], a deep neural network of 11-19 layers with constant convolutional filter sizes (3x3) and max pooling sizes (2x2) throughout. The activation functions used everywhere are the common ReLU functions, except for the last fully connected layer where a softmax function is used as it is common for classification tasks.

In these networks, the pattern of stacked convolutions with a smaller kernel size is used to approximate larger sized filters (larger receptive field) while introducing more non-linearities and thus making the decision function more discriminative and keeping the number of parameters smaller at the same time. The most common variants are VGG-16 and VGG-19 which are shown in Fig. 3.7. The architectures A, B and D of VGGNet that were examined in [5] for the problem of face recognition are presented in Table 3.1 where architecture D refers to VGG-16.

VGG-Face (with 16 weight layers) was trained on VGG-Face dataset for both tasks of face verification and face recognition. For evaluation, the standard benchmarks datasets LFW and YTF were used, employing the outside data verification protocol which enables using external training data. VGG-Face attained an accuracy of 98.95% on the LFW dataset and 97.3% on YTF.

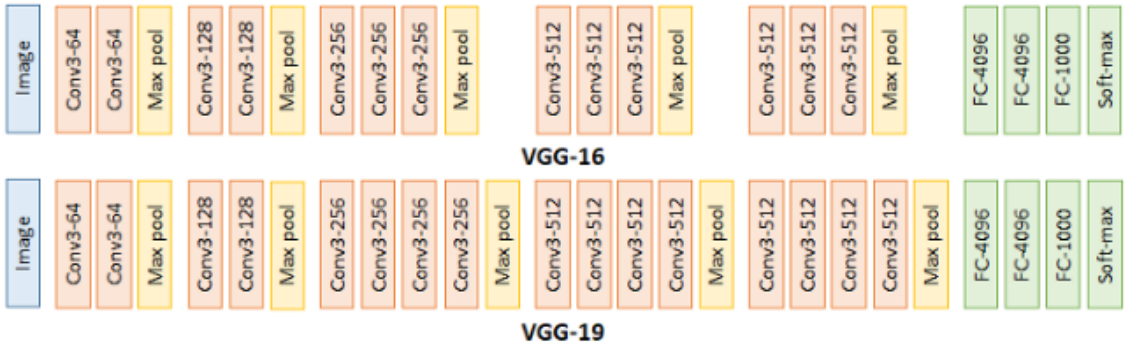


Figure 3.7: The overall architecture of VGGNet: VGG-16 and VGG-19 variants

To conclude, both FaceNet and VGG-Face learn useful face embeddings by minimizing the triplet loss that can be used to differentiate between different identities via a simple distance criterion. Both of them apply triplet networks (three instances of the same network with shared parameters) and the idea of one-shot learning which aims to learn useful features and solve classification tasks from only a few, or even one, training image. At test time, they are given two input images and they produce a similarity score between these two where clearly high values indicate that the image pair depicts the same person. In that way, problems associated with limited data for each class and a varying number of different classes (like in an online verification system) can be eliminated. Such a situation can appear when building a face recognition system for an organization where only a few images per employer are available and inevitably deep neural networks can't work.

In this project, apart from some traditional face recognition algorithms discussed in Chapter 2, we also focused on VGG-Face because of its impressive results and the availability of VGG-Face dataset

Architecture A	Architecture B	Architecture D
11 weight layers	13 weight layers	16 weight layers
input (224×224 RGB image)		
conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
Max pool		
conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
Max pool		
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
Max pool		
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
Max pool		
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
Max pool		
FC-4096		
FC-4096		
FC-1000		
softmax		

Table 3.1: Some VGGNet architectures that inspired VGG-Face

and pretrained weights as found in [5] by solving the problem of face recognition.

4 Datasets

Every machine learning model needs data, first to be trained on and later for reporting an estimate of its performance on unseen instances. Labelled datasets for face recognition should be as rich as possible covering many within class variations like changes in expression, pose, illumination etc. Especially for fitting the UBM, in case of face verification with GMMs, a large set of images of many different subjects is required to model the distribution of subject-independent facial features.

4.1 Publicly available face datasets

Some commonly used face datasets that are publicly available, are listed below:

- Color FERET Database, USA [38]

  The FERET database was created by Dr. Harry Wechsler at George Mason University and Dr. Phillips at the Army Research Laboratory (ARL) and collected in 15 sessions between August 1993 and July 1996. The database contains 1564 sets of images for a total of 14,126 images. It contains 1,199 subjects and 365 duplicate sets of images which depicted the same person but on a different day.

- Multi-Pie [39]

  The CMU Multi-PIE face database collected at Carnegie Mellon University contains 337 subjects, captured under 15 view points and 19 illumination conditions. In total, there were four recording sessions resulting in more than 750,000 images.

- Yale Face Database [40]

  This dataset contains 165 grayscale images of 15 individuals. There are 11 images per subject covering the following conditions: center-light, with glasses, happy, left-light, without glasses, normal, right-light, sad, sleepy, surprised, and wink.

- AT&T "The Database of Faces" (formerly "The ORL Database of Faces") [41]

  This dataset contains ten different images of 40 subjects under different lighting conditions, facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). All images were captured with frontalized faces having some small tolerance for side movement.

- MIT-CBCL Face Recognition Database [42]

  The MIT-CBCL face recognition database contains face images of 10 subjects providing two training sets:

  1. High resolution pictures, including frontal, half-profile and profile views

2. Synthetic images (324/subject) which were created by fitting 3D head models to the high-resolution training images. The 3D models are not included in the database.

The test set contains 200 images per subject. The creators varied the illumination, pose (up to about 30 degrees of rotation in depth) and the background.

- AR Face Database, The Ohio State University, USA [43]

  This face database was created by Aleix Martinez and Robert Benavente in the Computer Vision Center (CVC) at the U.A.B. It contains over 4,000 color images of individuals (70 men and 56 women). Images are taken from frontal view with different facial expressions, illumination conditions, and occlusions (sun glasses and scarf).

- VidTIMIT Audio-Video Dataset [44]

  The VidTIMIT dataset consists of video and corresponding audio recordings of 43 people, reciting short sentences. There were 3 sessions in total, with a space of about a week between each session. Each person recited 10 sentences, chosen from the TIMIT corpus and in addition they moved their head to the left, right, back to the center, up, then down and finally return to center. They used a broadcast quality digital video camera inside an office environment with fan noise.

- Labeled Faces in the Wild (LFW) [45]

  This dataset is one of the most commonly used datasets for evaluating face verification algorithms. It consists of 13,233 images collecting from the web concerning 5,749 different identities, with large variations in pose, expression and illumination. 1,680 individuals have two or more distinct photos in the database.

- The EURECOM Kinect Face Dataset (EURECOM KFD) [46]

  The dataset consists of multimodal facial images of 52 people (14 females, 38 males) obtained by Kinect in two sessions separated by an interval of about two weeks. In each session, 9 facial images for each person are taken according to different facial expressions, lighting and occlusion conditions. An RGB color image, a depth map and the associated 3D data are provided for all samples as well as annotations for 6 facial landmarks.

- FaceScrub - A Dataset With Over 100,000 Face Images of 530 People [47]

  FaceScrub consists of a total of 107,818 face images of 530 celebrities, with about 200 images per person. As a result, it is one of the largest public face databases. It was built based on an automatic procedure that discards those faces that do not belong to each queried person.

- VGG-Face dataset [5]

  This dataset consists of 2,622 celebrities where each identity has 1,000 images. VGG-Face dataset was created by combining small manual efforts with automated procedures.

- Youtube Faces (YTF) [48]

  This dataset contains 3,425 videos of 1,595 people from the set of subjects in LFW collected from Youtube. On average, each identity has 2.15 videos and images have much variation in resolution and poses. It's a standard benchmark for face recognition in video.

4.2 Custom Youtube Video Dataset for building UBM

As discussed before, a large face dataset including large variations is required for building the prior model, also known as UBM. One such dataset was created by following the principle of YTF [48] in [49]. For each one subject of 5,749 people found in LFW, top Youtube videos were downloaded in order to find videos of at least 10 seconds where the same face is depicted in all frames. At most 2 videos per each identity were kept, and if this search was not successful after 5 trials, the next name was considered. Faces were detected using cascade classifiers combined with LBP features [14] and differences of pixel intensities of detected face from past and future frames were compared to a threshold to infer if the same person is shown throughout the video. If this holds, for a time span of 10 seconds, the corresponding frames are stored in the database.

5 Experiments

In this chapter, experiments of some simple (baseline) algorithms are first shown using various face datasets. Then, a state-of-the art deep neural network for face verification is considered.

All experiments were implemented in Python because of its ease of use and the available toolkits that enable efficient image processing (OpenCV), computation of linear algebra operations (Numpy), implementation of machine learning algorithms (scikit-learn or sklearn) and easy definition and training of deep neural networks (Keras with TensorFlow backend).

5.1 Degradations of Face Images

Images are generally captured in different conditions at enrollment and validation times. Test faces might be noisy, blurred or differently illuminated compared to the well-controlled environment that usually involves the training (gallery) faces. In this project, we considered the effect of the nuisanace factors of salt-and-pepper noise, blur and illumination experienced by adding different pixel intensity constants and gamma correction on the performance of some traditional machine learning and deep learning algorithms. These degradations were performed for the AT&T (or ORL) dataset [41].

We first blurred our images by filtering with $3 \times 3, 5 \times 5, 7 \times 7, 9 \times 9, 11 \times 11$ and 13×13 averaging filters (Fig. 5.1). In a blurred image, the edges become less sharp which means that the shapes of the parts (for example in a face image, this could be eyes, nose, lips) are not easily recognized.

(a) original image (b) blurred with 5x5 averaging filter (c) blurred with 13x13 averaging filter

Figure 5.1: Adding blur to an image

Next, we tried adding salt-and-pepper noise with different noise probabilities $p = 0.01$, $p = 0.015$, $p = 0.02$, $p = 0.025$, $p = 0.03$ (see Fig. 5.2) which in turn correspond to the following SNR values: $SNR = 17\,dB$, $SNR = 15\,dB$, $SNR = 14\,dB$, $SNR = 13\,dB$, $SNR = 12\,dB$ where the signal-to-noise

ratio for an additive noise model (SNR) is given by:

$$SNR = 10log_{10}\frac{\sum_{i,j} I(i,j)^2}{\sum_{i,j}(I_{noisy}(i,j) - I(i,j))^2} \qquad (dB) \tag{5.1}$$

Salt-and-pepper noise, also known as speckle noise, appears itself as randomly occurring white and black pixels in the image signal and can be caused by analog-to-digital converter errors, channel transmission errors, defects in the CCD etc. The noise level is defined as $2p$, since p is the probability of observing a dark (black) pixel and usually random light (white) pixels occur with the same probability p. Typically, p is less than 0.2 [50].

(a) original image (b) noisy with salt-and-pepper probability p=0.01 (c) noisy with salt-and-pepper probability p=0.03

Figure 5.2: Adding salt-and-pepper noise to an image

Finally, we tried to change the illumination conditions of the existing images in two ways. First, we added a constant to the current intensity (modulo 256 sum) (Fig. 5.3).

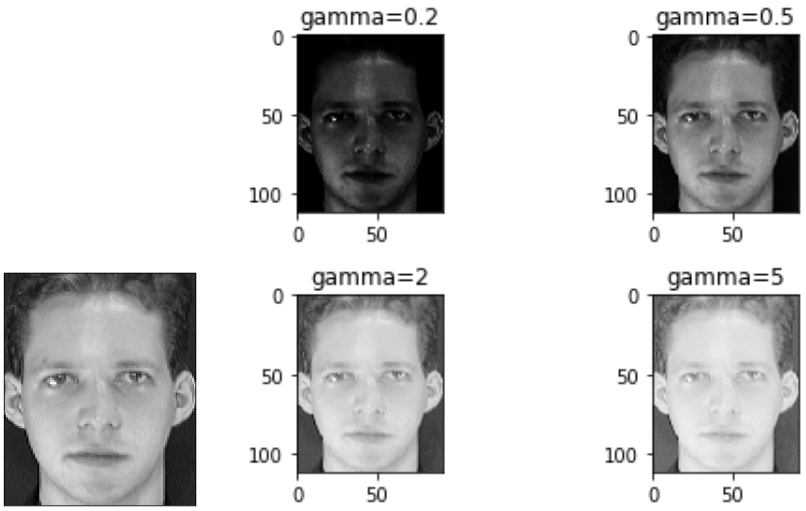
(a) original image (b) darkened image (intensity-50) (c) lightened image (intensity+50)

Figure 5.3: Adding a constant to the current pixel intensity of an image

We also examined changes in illumination modelled by gamma correction where the output image is given by:

$$I_{out} = I_{in}^{\frac{1}{\gamma}} \tag{5.2}$$

Values of γ smaller than one result in darkening the image by compressing the dynamic range in dark areas and enhancing the brightness in bright areas. The effect of values of larger than one is exactly the opposite. We considered $\gamma \in \{0.2, 0.5, 2, 5\}$ and results of this transformation for a sample image are shown in Fig. 5.4.



(a) original image (b) gamma corrected images for differnet values of γ

Figure 5.4: Effect of gamma correction on an input image for different values of γ

5.2 Traditional Machine Learning Algorithms for Face Recognition

5.2.1 Holistic Methods, LBPH and Gabor features

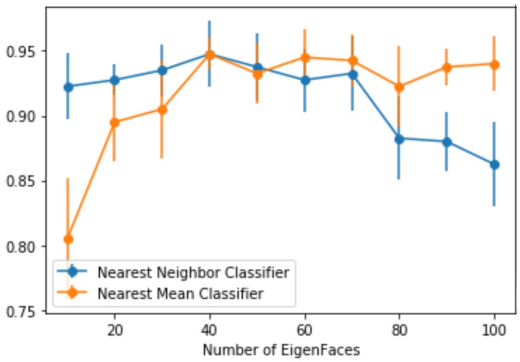
First experiments were aimed at investigating some simple algorithms that are commonly used in the field of face recognition. To that end, we considered the AT&T dataset which consists of 40 subjects each having 10 images. Since there is only a small variation in lighting, pose, expressions and facial details, it's a rather simple dataset.

As far as the algorithms are concerned, we first experimented with the Eigenfaces and Fisherfaces methods for different numbers of preserved features. For PCA, we additionally whitened our data that is after projection features are becoming uncorrelated having a unit variance, a common transform in machine learning which has been shown to produce better results. As for classification, after projecting onto the corresponding subspaces, we experimented with the NNC, NMC and SVM classifiers, both the linear and the Gaussian (RBF) one (with default parameters provided by sklearn; regularization parameter $C = 1$, Gaussian kernel width $\gamma = \frac{1}{\#features}$). All experiments were performed by splitting the whole dataset randomly into two subsets, training and test sets at a ratio $80\% - 20\%$ and repeating for 10 iterations. To report a performance measurement of our pattern recognition system, we calculated the face recognition accuracy on the test set given by:

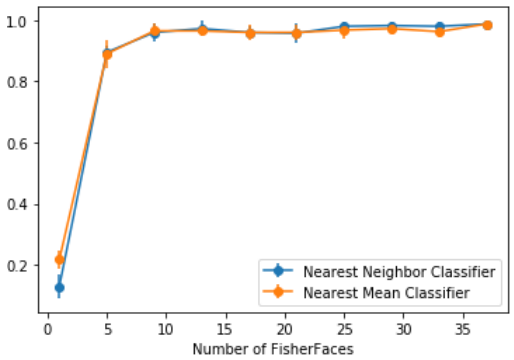
$$accuracy = \frac{correctly\ classified\ test\ set\ objects}{total\ number\ of\ test\ set\ objects} \tag{5.3}$$

The average accuracy, as well as its standard deviation, are given in the following errorbar plots of Fig. 5.5. From these plots, it can be observed that the simple Eigenfaces and Fisherfaces representations give already remarkably good results on this simple dataset, with the latter giving even better results than the former. Also, the linear SVM classifier with the LDA-based features gives the best

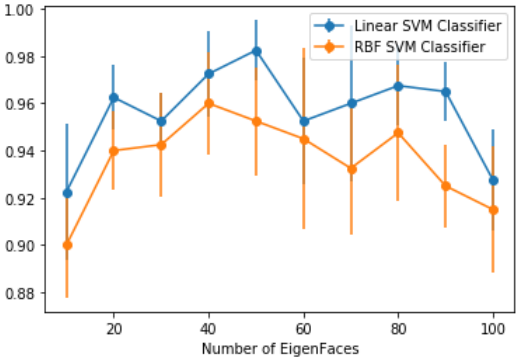
performance among all SVM and features combinations. Furthermore, using more than 5 Fisherfaces or 40 Eigenfaces, performance doesn't increase significantly (as a matter of fact, accuracy drops for algorithms based on PCA features beyond 40 Eigenfaces except for the NMC) and this is a great benefit of these dimensionality reduction algorithms as they manage to reduce the number of features from $112 \times 92 = 10304$ which is the original image size in pixels to something much less with no negative impact on the classification results. It should be also noted that the first 5 principal components account for just 48% of the total input variance while the first 100 principal components explain 89% of it when trained on the whole AT&T dataset of 400 images.



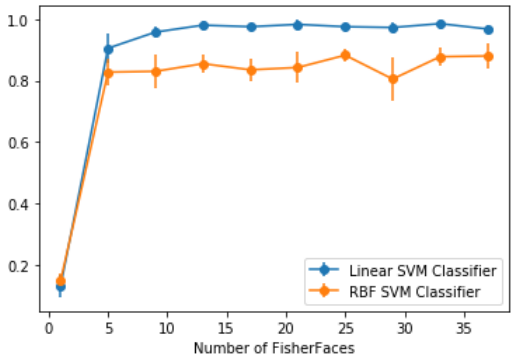
(a) Eigenfaces combined with NNC and NMC



(b) Fisherfaces combined with NNC and NMC



(c) Eigenfaces combined with Linear and Gaussian SVM



(d) Fisherfaces combined with Linear and Gaussian SVM

Figure 5.5: Errorbar plots of the recognition accuracy of the first 4 (baseline) algorithms

We also tried the LBPH method using 8 neighbors and dividing the input image to 8 cells in both horizontal and vertical directions. Classification was performed using the nearest neighbor rule on χ^2 distance feature space where the χ^2 distance between two histograms $\boldsymbol{x}$ and $\boldsymbol{\xi}$ is given by:

$$\chi^2(\boldsymbol{x}, \boldsymbol{\xi}) = \sum_{j,i} \frac{(x_{ij} - \xi_{ij})^2}{x_{ij} + \xi_{ij}} \tag{5.4}$$

We obtained the results of Table 5.1, after performing 10 iterations. LBPH algorithm also performs very good on AT&T face images with optimum results obtained with a radius of 2 pixels.

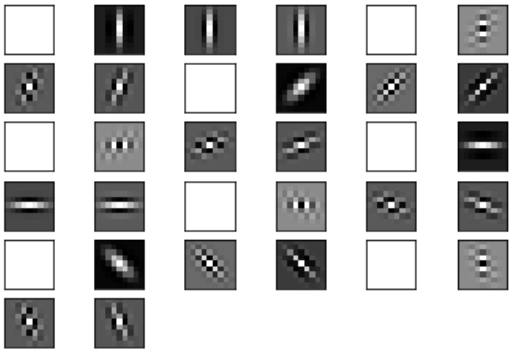
We next considered to apply a filter bank of Gabor filters in order to extract possibly interesting features. This filtering can be simply done by convolving the (histogram equalized) input image with

(radius, neighbors)	Recognition accuracy ($\mu \pm \sigma$) (%)
(1,8)	$97.62\% \pm 1.53\%$
(2,8)	$98.12\% \pm 1.28\%$
(3,8)	$97.50\% \pm 1.85\%$

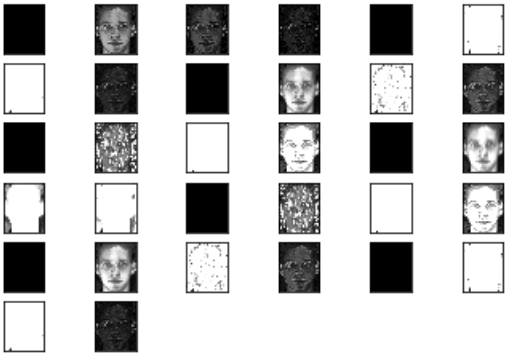
Table 5.1: Recognition accuracy of LBPH face recognizer

a set of Gabor filters. We applied a total of 32 filters with 8 different orientations (evenly spaced from 0 to π) and 4 different wavelengths (evenly spaced from 0 to π) resulting in $h \times w \times \#filters = 329728$ features per image (see Fig. 5.6). Results of the first 4 baseline algorithms combined with Gabor features are shown in Fig. 5.7. Here, the first 5 principal components explain 23% while the first 100 ones explain 63% of the total variance of Gabor features when trained on the whole At&T dataset. Even though the percentage 63% may seem less, the contribution of the 100th principal component is already small (0.1694%) rendering the inclusion of more principal components reduntant. Comparing Fig. 5.7 with Fig. 5.5, it is observed that this Gabor features representation gives slightly improved results compared to the initial pixel intensity representation.

(a) A face image (prior to histogram equalization)



(b) A filterbank of 32 Gabor filters



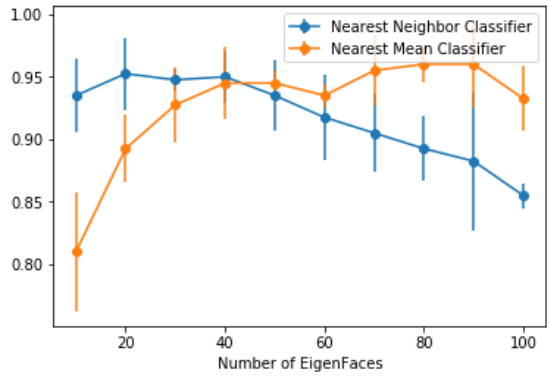
(c) Gabor features

Figure 5.6: A visualization of Gabor filters and Gabor features

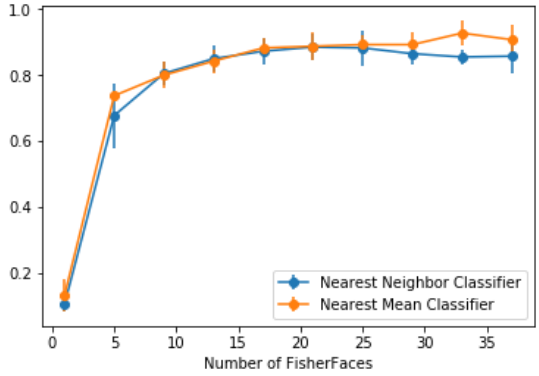
From the above results, we can conclude that all the beforementioned approaches give a very high accuracy rate on an independent test set, but this is misleading since AT&T is an easy dataset.

We next considered to evaluate these algorithms on the different degradations discussed in Section 5.1. For the following experiments, we fixed the number of principal components to 100 and the number of Fisherfaces to 39 (the maximum allowable) as these values yielded good results.

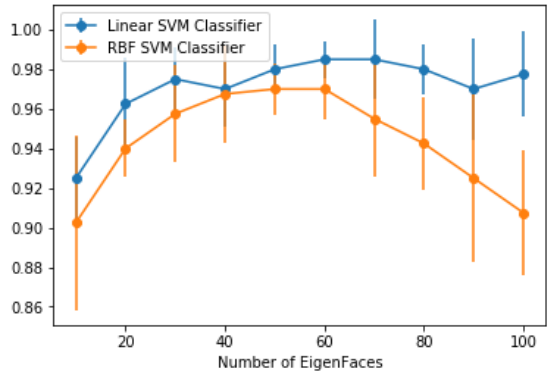
Training on the clean dataset and testing on the blurred dataset, we obtained the results of Fig. 5.8.



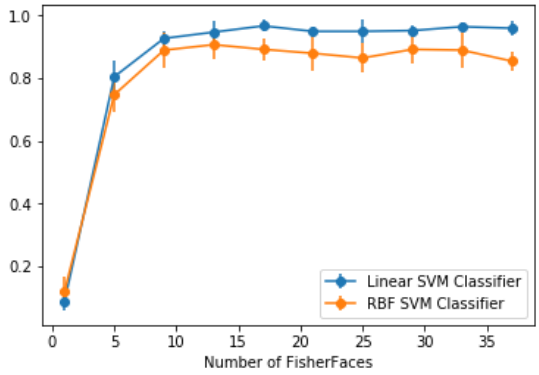
(a) Eigenfaces (Gabor features) combined with NNC and NMC



(b) Fisherfaces (Gabor features) combined with NNC and NMC



(c) Eigenfaces (Gabor features) combined with Linear and Gaussian SVM



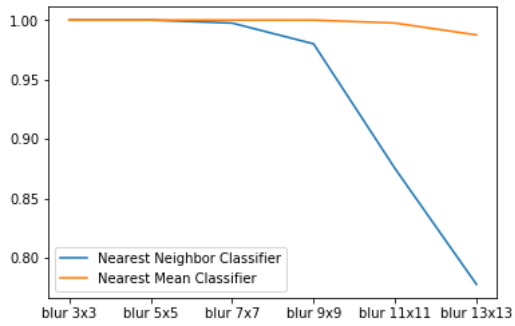
(d) Fisherfaces (Gabor features) combined with Linear and Gaussian SVM

Figure 5.7: Errorbar plots of the recognition accuracy of the first 4 (baseline) algorithms on Gabor feature space

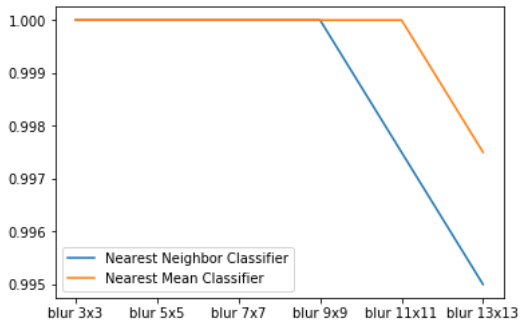
As we see, the NMC is more robust than the NNC for both representations. Furthermore, Linear SVM systematically outperforms RBF SVM. The Fisherfaces representation gives better results than the Eigenfaces representation except for the case of the linear SVM classifier. Moreover, as expected, if we add more blur, the recognition results become less accurate.

Training on the clean dataset and testing on the noisy dataset, we obtained the results of Fig. 5.9. All algorithms give perfect predictions even for increasing noise levels.

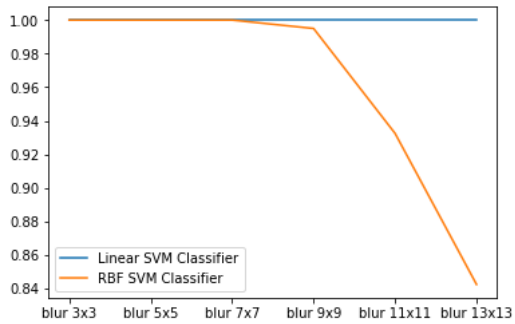
Results of the first 4 baseline algorithms for the test cases determined by different adding constants are presented in Fig. 5.10 while the corresponding results for the different gamma corrections are presented in Fig. 5.11. Eigenfaces combined with either NNC/NMC or Linear/Gaussian SVM classifier performed very good only for bright test images (adding constant ≥ -20) while breaking down for darker images. Same conclusions hold for Fisherfaces combined with NNC/NMC with the difference that they perform poorly also for very bright images (adding constant > 30). Furthermore, Fisherfaces combined with SVM classifiers give poor results for a much broader range of different illumination conditions. For the test case of images with gamma correction, all algorithms give best results for moderately different images ($\gamma \in (0.5, 2)$) where LDA-based features produce much worse results compared to PCA-based features. This implies that the use of the class label information in finding



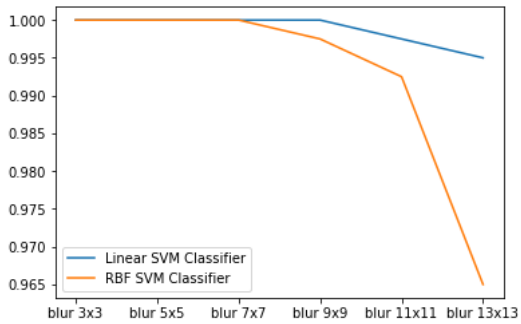
(a) Eigenfaces combined with NNC and NMC on blurred images



(b) Fisherfaces combined with NNC and NMC on blurred images



(c) Eigenfaces combined with Linear and Gaussian SVM on blurred images



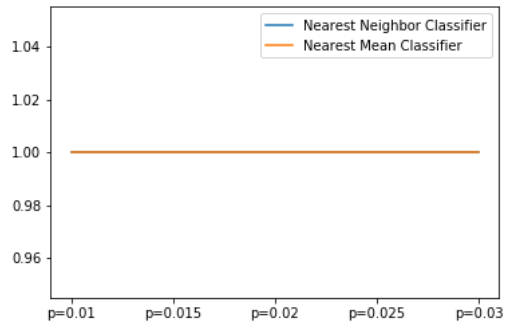
(d) Fisherfaces combined with Linear and Gaussian SVM on blurred images

Figure 5.8: Accuracy results of the first 4 baseline algorithms on the blurred images

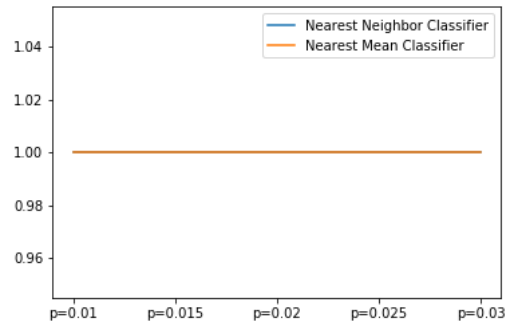
an appropriate subspace from the classification point of view is not optimal when illumination is much different at test time.

To finally conclude which algorithm gives more robust results for face verification against all these degradations we computed the Equal Error Rate (EER) in each separate test case from the macro-average Receiver Operating Characteristic (ROC) curve or alternatively the macro-average Detection Error Tradeoff (DET) Curve. The ROC curve which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for different classification thresholds is typically used to measure biometric performance for verification tasks [51]. Alternatively, the same information can be visualized by a DET curve [52] which plots the False Negative Rate (FNR) or False Rejection Rate (FRR) against the FPR or False Acceptance Rate (FAR). These is a a tradeoff between these two types of errors and the point where FPR=FNR gives the EER. The macro-average ROC curve can be simply constructed by averaging ROC curves for each separate one versus all classification problems defined for each class-subject. Results are presented in Table 5.2.

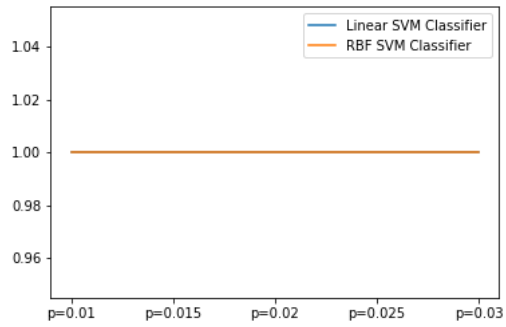
Based on the table, we can say that Gabor features combined with PCA and linear SVM result in the most robust verification system achieving an average EER of 0.97% . We can also immediately notice that all these methods suffer when the illumination conditions change drastically except for the LBPH (which as expected is invariant to monotonic gray-level changes) pointing out that these illumination changes are much more annoying compared to noise and blur.



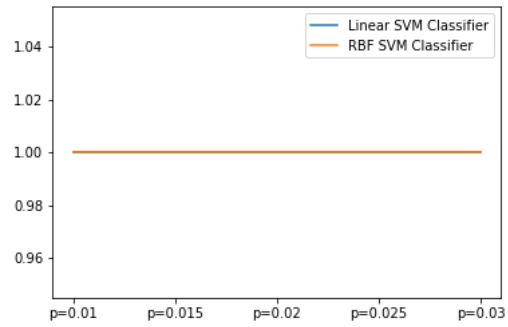
(a) Eigenfaces combined with NNC and NMC on noisy images



(b) Fisherfaces combined with NNC and NMC on noisy images



(c) Eigenfaces combined with Linear and Gaussian SVM on noisy images



(d) Fisherfaces combined with Linear and Gaussian SVM on noisy images

Figure 5.9: Accuracy results of the first 4 baseline algorithms on the noisy images

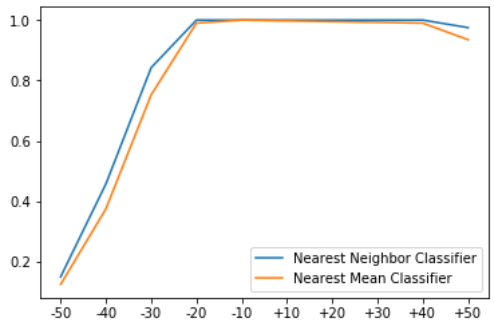
5.2.2 DCT block-based Face Recognition

For our experiments with the GMM-UBM model combined with local block-based DCT features, we considered the VidTIMIT database [44] and first 200 videos from the Youtube video dataset described in Section 4.2.

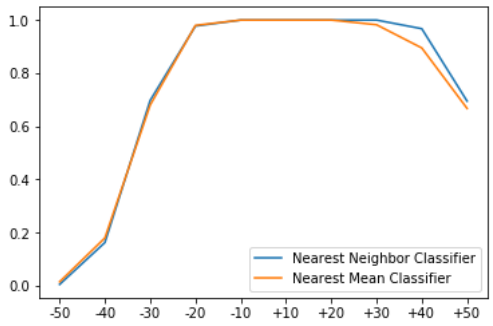
Frames were selected every 5 instances from the Youtube dataset and every 10 instances from the VidTIMIT dataset. In this way, we tried to take advantage of the possible variety of poses, expressions, lighting conditions etc. that may exist in a video sequence to obtain a more robust face verification system for still images.

The detected faces from the input images (using the Viola and Jones Haar cascade face detector) were cropped and resized to 80×80 and (optionally) photometrically normalized using the Tan & Triggs preprocessing chain. Then, the image is divided into a number of $B \times B$ sized blocks with an overlap of 50% in both horizontal and vertical directions to ensure a good representation of the image. Obviously larger values for the overlap lead to better robustness with respect to errors in alignments but also to an increase in the number of blocks that arise and thus to an increase in the computational complexity.

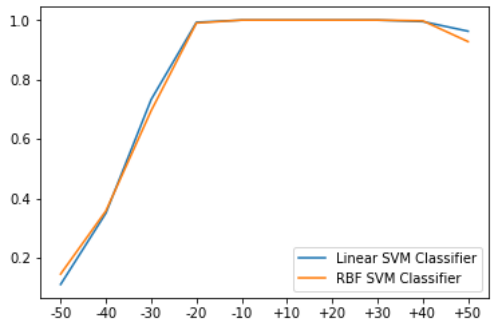
Each block was normalized to zero mean and unit variance before extracting the M+1 lowest-frequency 2D discrete cosine transform (2D-DCT) coefficients using a zig-zag technique. After the normalization of the block, the lowest frequency is redundant (always equal to zero) so it can be excluded. The feature vectors that arise from each image are mean and variance normalized in each dimension. This



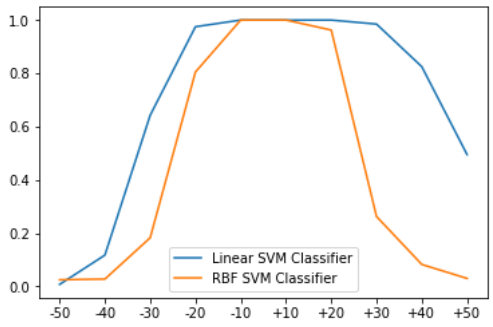
(a) Eigenfaces combined with NNC and NMC on differently illuminated images



(b) Fisherfaces combined with NNC and NMC on differently illuminated images



(c) Eigenfaces combined with Linear and Gaussian SVM on differently illuminated images



(d) Fisherfaces combined with Linear and Gaussian SVM on differently illuminated images

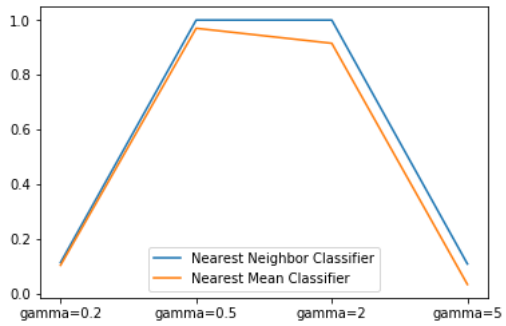
Figure 5.10: Accuracy results of the first 4 baseline algorithms on the differently illuminated images

normalization can boost the performance of the end-classifier as it gives equal significance to all DCT frequency dimensions. Finally, each image contains a set of K feature vectors, $O = \{\boldsymbol{o}^1, \boldsymbol{o}^2, \ldots, \boldsymbol{o}^K\}$, where the feature vector dimensionality is M.

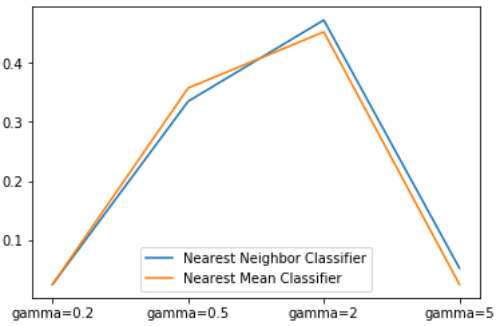
The VidTIMIT dataset was divided into 3 separate sets: training, development and test sets. The training set contained the videos from the 1st session, the development set contained the videos from the 2nd session and lastly the videos from the 3rd session were included in the test set.

The training set was used to build the client models by performing mean-only MAP adaptation of the UBM which was built using the EM algorithm on the separate set of face images extracted from the 200 Youtube videos. The maximum number of iterations for the EM algorithm was set to 100 and the convergence threshold was set to 10^{-5}. The initialization of the model parameters for the EM algorithm was done using the k-means clustering algorithm. The development set was used to tune the hyperparameters such as the block size B, the number of DCT coefficients M kept in each block and the number of components C. In addition, the development set was used to derive the decision threshold τ^* that sets FPR=FNR producing the EER. Finally, the test set was used to find the Half Total Error Rate (HTER) which is defined as the mean of the FPR and FNR at the threshold τ^*. The relevance factor was set equal to 16 as this value has been shown to produce good results in the literature [8], [29].

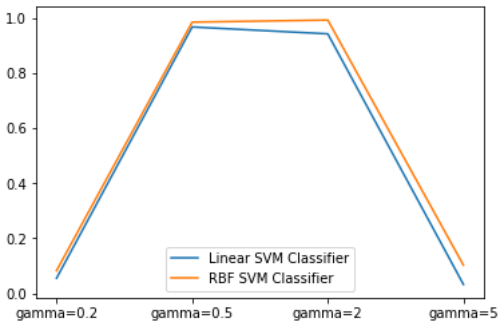
Preliminary experiments showed that $B = 16$, $M = 66$ is a good choice in our case (see Table 5.3).



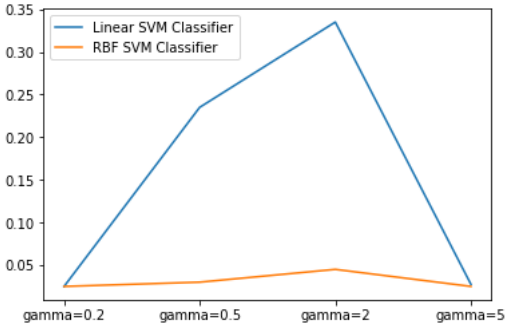
(a) Eigenfaces combined with NNC and NMC on differently illuminated images



(b) Fisherfaces combined with NNC and NMC on differently illuminated images



(c) Eigenfaces combined with Linear and Gaussian SVM on differently illuminated images



(d) Fisherfaces combined with Linear and Gaussian SVM on differently illuminated images

Figure 5.11: Accuracy results of the first 4 baseline algorithms on images with gamma correction

So, we next focused on trying different numbers of Gaussian components for modelling the UBM and therefore the client models as well, with fixed block size $B = 16$ and keeping $M = 66$ DCT coefficients per block.

Results are shown in Tables 5.4 and 5.5, where in Table 5.4 Tan & Triggs preprocessing was applied while for Table 5.5 this illumination normalization was not taken into consideration.

We can see that results are better without Tan & Triggs preprocessing and this can be explained by the low resolution of the cropped images and the small variation in the lighting conditions. Furthermore, using more Gaussian components, the distribution of the DCT coefficients becomes more accurate improving results but the gain becomes small after $C = 64$ components.

The ROC and DET curves of the face verification system with GMM-UBM modelling, $B = 16$, $M = 66$ and $C = 128$, with and without Tan & Triggs preprocessing of the cropped face images, are presented in Fig. 5.12 and 5.13 correspondingly.

We also experimented with the same features but with different classifiers from the LRT in the GMM-UBM approach. Specifically, we considered NNC, linear SVM and Gaussian SVM classifiers for different number of principal components $(50, 100, 200, 400)$ corresponding to 57%, 71%, 83% and 92% of the total variability in the input training data respectively. The parameters for the SVM classifiers (C which is the regularization parameter and γ which is the inverse of the variance of the gaussian

	Pixel Intenisty Features				Gabor Features				
	Eigenfaces	Fisherfaces	SVM+Eigenfaces	SVM + Fisherfaces	Eigenfaces	Fisherfaces	**SVM+Eigenfaces**	SVM+Fisherfaces	LBPH
3×3	0	0	0	0	0.0050	0.0148	**0**	0.0067	0
5×5	0	0	0	0	0.0315	0.0742	**0**	0.0102	0.0274
7×7	0.0026	0	0	0	0.1091	0.1308	**0**	0.0127	0.0930
9×9	0.0244	0	0	0	0.2248	0.1673	**0**	0.0203	0.1544
11×11	0.1031	0.0026	0	0	0.3032	0.2192	**0.0025**	0.0195	0.2142
13×13	0.1870	0.0051	0	0	0.3551	0.2572	**0.0038**	0.0241	0.2767
p=0.01 (17 dB)	0	0	0	0	0	0	**0**	0.0076	0
p=0.015 (15 dB)	0	0	0	0	0	0	**0**	0.0116	0
p=0.02 (14 dB)	0	0	0	0	0	0	**0**	0.0125	0
p=0.025 (13 dB)	0	0	0	0	0	0	**0**	0.0076	0
p=0.03 (12 dB)	0	0	0	0	0	0.0050	**0**	0.0084	0
Intensity-50	0.4642	0.5058	0.5848	0.6125	0.1659	0.4244	**0.1306**	0.4599	0
Intensity-40	0.3616	0.4620	0.3608	0.3822	0.0763	0.3491	**0.0709**	0.3297	0
Intensity-30	0.1364	0.2336	0.1137	0.1184	0	0.1939	**0.0063**	0.1848	0
Intensity-20	0	0.0220	0	0	0	0.0268	**0**	0.0566	0
Intensity-10	0	0	0	0	0	0	**0**	0.0114	0
Intensity+10	0	0	0	0	0	0	**0**	0.0120	0
Intensity+20	0	0	0	0	0	0	**0**	0.0120	0
Intensity+30	0	0	0	0	0	0	**0**	0.0096	0
Intensity+40	0	0.0315	0	0	0	0	**0**	0.0119	0
Intensity+50	0.0244	0.2338	0.0092	0.0076	0	0.0050	**0**	0.0270	0
$\gamma = 0.2$	0.4749	0.5	0.0114	0.0206	0.1091	0.0408	**0.0291**	0.0408	0
$\gamma = 0.5$	0	0.4042	0	0	0	0	**0**	0	0
$\gamma = 2$	0	0.3454	0	0	0	0	**0**	0	0
$\gamma = 5$	0.4650	0.4924	0	0.0244	0	0	**0**	0	0
Averaged EER	0.0897	0.1295	0.0432	0.0466	0.0552	0.0763	**0.0097**	0.0519	0.0306

Table 5.2: Equal Error Rates (EERs) of selected baseline algorithms on different degradation conditions

B	M	C	EER % (development set)	τ^*	HTER % (test set)
8	28	16	12.21	0.1532	14.19
12	45	16	5.49	0.1749	6.63
16	**66**	**16**	**4.35**	**0.6177**	**5.76**
20	66	16	4.32	0.6659	4.80
24	91	16	3.41	1.2094	4.06

Table 5.3: Results of the GMM-UBM model with 2D DCT features on the VidTIMIT dataset without Tan & Triggs preprocessing for various approaches during feature extraction

kernel) were tuned on the validation (development) set. The parameter values that were examined were $C \in \{1, 10, 100, 1000\}$ and $\gamma \in \{0.0001, 0.001, 0.01\}$. Results are presented in tables 5.6 and 5.7 with Tan & Triggs and without Tan & Triggs preprocessing respectively and in parentheses the optimal values of the parameters are shown.

We can notice again that results with no Tan & Triggs preprocessing are generally better than those obtained with this illumination normalization into consideration. Moreover, an increase of the principal components kept does not necessarily lead to an increase in the performance as we see in the case of NNC. The best results (based on the development set) for each classifier are highlighted in bold.

Comparing the results of Tables 5.6 and 5.7 to those of Tables 5.4 and 5.5, we can conclude that the GMM-UBM system is clearly superior to the simple NNC, but lags behind the optimal linear and RBF SVM classifiers which can be explained by the discriminative training approach of the latter classifiers. The best results on the test set, for both cases of including Tan & Triggs preprocessing and not, are met using 400 principal components and a linear SVM classifier.

As a final experiment with block-based DCT features, we tried to see how they behave on the same degradations of ORL images as before.

B	M	C	EER % (development set)	τ^*	HTER % (test set)
16	66	16	7.36	0.5227	8.86
16	66	32	4.32	0.4434	5.18
16	66	64	3.33	0.2987	4.38
16	**66**	**128**	**2.84**	**0.5970**	**4.06**

Table 5.4: Results of the GMM-UBM model with 2D DCT features on the VidTIMIT dataset with Tan & Triggs preprocessing

B	M	C	EER % (development set)	τ^*	HTER % (test set)
16	66	16	4.35	0.6177	5.76
16	66	32	3.68	0.4866	4.48
16	66	64	2.80	0.6880	3.89
16	**66**	**128**	**2.75**	**0.7495**	**3.22**

Table 5.5: Results of the GMM-UBM model with 2D DCT features on the VidTIMIT dataset without Tan & Triggs preprocessing

We first experimented with the block size B and the number of DCT coefficients M kept from each block. The original ORL images were divided into two sets, training and test, at a ratio $80\% - 20\%$. Projecting the resulting features onto 200 principal components which account for 87% of the total variability of input data and using the nearest neighbor classification rule, we obtained the recognition accuracy results of Table 5.8. The optimal values for the block size and the number of DCT coefficients per block were $B = 20$ and $M = 66$ correspondingly.

Training on the whole ORL dataset of clean images and testing on the various difficult scenarios described before, we obtained the EERs of Table 5.9 by macro-averaging the DET curves to be consistent with the comparison with the previous algorithms of Eigenfaces, Fisherfaces, Gabor Eigenfaces, Gabor Fisherfaces all of which were combined with NNC and linear SVM and finally LBPH.

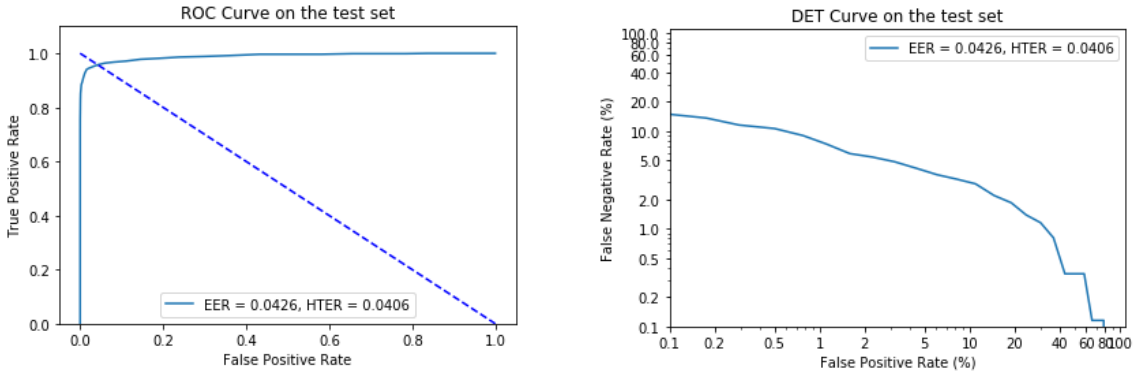
We can notice again that the Tan & Triggs preprocessing has generally a negative impact on face verification performance especially in case of blur. This can be explained by the fact that the negative effect of blurring is highlighted by the Gaussian blurring step in the preprocessing chain. However, it is beneficial in case of great illumination changes modelled by a large negative additive constant as it is explicitly aimed at mitigating some effects of the illumination variation.

Best results are obtained using a Gaussian SVM classifier (with $C = 10$, $\gamma = 0.001$) on top of DCT features with no Tan & Triggs preprocessing. This model achieves an EER of 2.65% averaged across all degradations. Comparing Tables 5.2 and 5.9 we can conclude that the optimal results are obtained with a linear SVM classifier on Gabor features reduced by PCA.

5.3 Deep Face Recognition

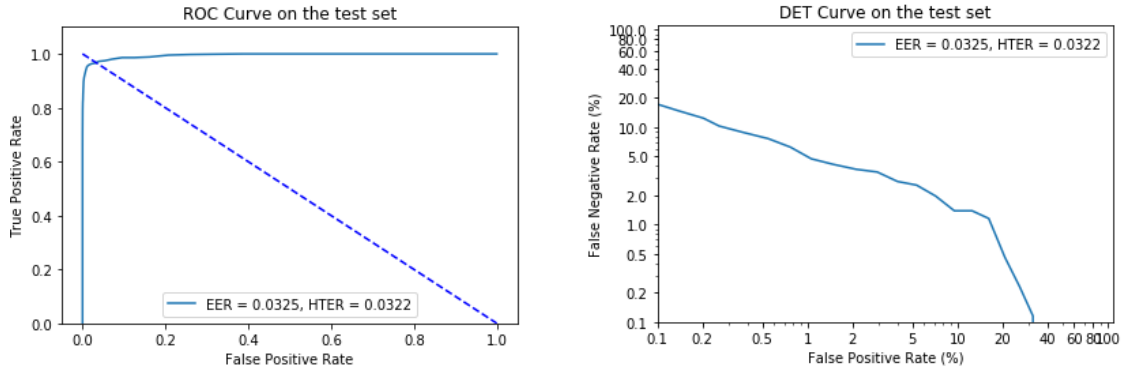
Inspired by the great success of VGG networks and also VGG-Face which is based on their architecture to solve the problem of face recognition using deep learning techniques, we tried to implement ourselves the experiments done in [5].

For our purposes, we used a subset of the VGG-Face dataset due to lack of powerful computational



(a) (b)

Figure 5.12: ROC and DET curves of the GMM-UBM based face verification system, with $B = 16$, $M = 66$, $C = 128$ and Tan & Triggs preprocessing, on the test set



(a) (b)

Figure 5.13: ROC and DET curves of the GMM-UBM based face verification system, with $B = 16$, $M = 66$, $C = 128$ and no Tan & Triggs preprocessing, on the test set

resources (GPUs) and the expensive training of deep neural networks. The authors provide the URLs of the training images of the famous subjects of the dataset alongside with the bounding boxes of the detected faces. Some links were dead, so we ended up with a training set of 1591 subjects and 1,033,160 images, so on average there are 649 images per subject.

The architecture of VGG-Face which consists of 16 weight layers is shown in the last column of Table 3.1 with the notice that the last fully connected layer doesn't consist of 1000 units (like VGG networks that were used in ImageNet challenge to recognize 1000 classes) but the number of different subjects in the face dataset (2622 units in [5]). Two other variants of VGG networks were also considered in [5]. For our initial experiments for face recognition, we also considered configuration A from VGG networks which is 11 weight layers deep and shown in the first column of Table 3.1.

	NNC			Linear SVM			RBF SVM		
n (PCA)	EER %	τ^*	HTER %	EER %	τ^*	HTER %	EER %	τ^*	HTER %
50	13.58	0.3154	9.68	(C=1) 8.45	0.0095	10.68	(C=1, γ=0.01) 5.62	0.0097	6.68
100	**12.10**	**0.3207**	**7.38**	(C=1) 5.37	0.0097	7.16	(C=1, γ=0.001) 3.59	0.0099	3.86
200	12.54	0.3191	7.91	(C=1) 3.02	0.0099	4.44	(C=10,γ=0.001) 2.58	0.0099	2.86
400	22.41	0.2841	14.76	**(C=1) 1.92**	**0.01**	**2.29**	**(C=10, γ=0.0001) 2.02**	**0.0099**	**2.38**

Table 5.6: Results of various classifiers employing 2D DCT features on the VidTIMIT dataset with Tan & Triggs preprocessing

	NNC			Linear SVM			RBF SVM		
n (PCA)	EER %	τ^*	HTER %	EER %	τ^*	HTER %	EER %	τ^*	HTER %
50	11.83	0.3223	8.09	(C=1) 7.51	0.0095	9.94	(C=1,γ=0.01) 5.70	0.0097	6.52
100	**7.81**	**0.3359**	**5.14**	(C=1) 4.75	0.0098	6.28	(C=1,γ=0.001) 3.02	0.0099	3.33
200	11.74	0.3220	7.38	(C=1) 2.25	0.01	3.46	(C=1,γ=0.001) 2.79	0.0099	2.82
400	24.38	0.2771	15.76	**(C=1) 2.13**	**0.0099**	**2.38**	**(C=10, γ=0.0001) 2.24**	**0.0099**	**2.66**

Table 5.7: Results of various classifiers employing 2D DCT features on the VidTIMIT dataset without Tan & Triggs preprocessing

5.3.1 Training for face recognition

Initially, the CNN was used for face recognition, so the cost function used was the categorical cross-entropy loss given by

$$L = -\frac{1}{M}\sum_{t=1}^{M}\sum_{i=1}^{N} y_{ti} log(\hat{y}_{ti}) \tag{5.5}$$

where M is the mini-batch size, N is the number of different classes or subjects, $\boldsymbol{y_t}$ is the true one-hot label vector of face image l_t and $\hat{\boldsymbol{y}}_t$ is the output of the softmax layer at the end of the CNN.

The algorithm used to minimize the cost function was the mini-batch gradient descent (SGD) with momentum where the updates are given by equations 3.4 and 3.5.

According to [5], the batch size M and the momentum coefficient γ were set to 64 and 0.9 respectively. The learning rate was initially set to 10^{-2} and then it was decreasing by 10 every time that the validation accuracy stopped increasing but other values for the initial learning rate were also considered (see Table 5.10). The weights of VGG-Face A were initialized from a Gaussian distribution with $\mu = 0$ and $\sigma = 10^{-2}$.

The CNN was also regularized to combat overfitting. Specifically, it was trained with weight decay, the coefficient of which was explored among 0, $5 \cdot 10^{-8}$, $5 \cdot 10^{-4}$ and $5 \cdot 10^{-2}$ and after the first 2 fully connected layer dropout was also applied with probability 0.5. Moreover, since this deep convolutional network introduces many trainable parameters, it requires a very large training set. To obtain a richer training dataset, online data augmentation was applied (a very common strategy in deep learning) by flipping on the fly the training images of a mini batch left to right with 50% probability. Also, the images were mean normalized by subtracting per channel the average red, green and blue components as found from the training set to speed up learning. This was the only preprocessing done and it and was also applied later for triplet loss training. In each iteration random 224×224 crops of scaled images (the smaller dimension is set equal to 256) were feed to the network to make the model position and scale invariant.

An example of online data augmentation for a face in the dataset is shown in Fig. 5.14 where different

B	M	Recognition Accuracy % (test set)
8	28	56.25
12	45	53.75
16	66	58.75
20	**66**	**58.75**
24	91	57.5

Table 5.8: Results of NNC with 2D DCT features on a subset of the ORL dataset with no Tan & Triggs preprocessing for various approaches during feature extraction

	No Tan & Triggs preprocessing			Tan & Triggs preprocessing		
	1NN	Linear SVM (C=1)	**RBF SVM (C=10, $\gamma = 0.001$)**	1NN	Linear SVM (C=1)	RBF SVM (C=100, $\gamma = 0.0001$)
3×3	0	0	**0**	0	0	0
5×5	0	0	**0**	0	0	0.0019
7×7	0.0522	0.0212	**0.0201**	0.3068	0.1616	0.1630
9×9	0.2853	0.0860	**0.0815**	0.4073	0.3456	0.3468
11×11	0.3681	0.0691	**0.0661**	0.4690	0.2731	0.2729
13×13	0.3725	0.0684	**0.0667**	0.4768	0.2671	0.2653
p=0.01 (17 dB)	0	0	**0**	0	0	0
p=0.015 (15 dB)	0	0	**0**	0	0.0038	0.0038
p=0.02 (14 dB)	0	0	**0**	0	0.0063	0.0063
p=0.025 (13 dB)	0	0	**0**	0.0026	0.0161	0.0295
p=0.03 (12 dB)	0	0	**0**	0.0074	0.0227	0.0235
Intensity-50	0.1993	0.2100	**0.1959**	0.0719	0.1034	0.0995
Intensity-40	0.0588	0.0593	**0.0522**	0.0172	0.0177	0.0165
Intensity-30	0	0.0017	**0**	0	0.0245	0
Intensity-20	0	0	**0**	0	0.0245	0
Intensity-10	0	0	**0**	0	0.0245	0
Intensity+10	0	0	**0**	0	0	0
Intensity+20	0	0	**0**	0	0	0.0245
Intensity+30	0	0	**0**	0	0	0
Intensity+40	0	0	**0**	0	0	0
Intensity+50	0	0.0021	**0**	0	0	0
$\gamma = 0.2$	0.1558	0.1774	**0.1800**	0.1667	0.1950	0.1950
$\gamma = 0.5$	0	0	**0**	0	0	0
$\gamma = 2$	0	0	**0**	0	0	0
$\gamma = 5$	0	0	**0**	0	0	0
Averaged EER	0.0597	0.0278	**0.0265**	0.0770	0.0596	0.0579

Table 5.9: Equal Error Rates (EERs) of selected algorithms on DCT features for different degradation conditions

versions of randomly flipped and cropped images are presented.

Training for 70 epochs, and randomly splitting the images of the first 3 subjects into training and validation sets at a ratio $80\% - 20\%$, we obtained the following results for the training and validation accuracy at the end of the training (see Table 5.10).

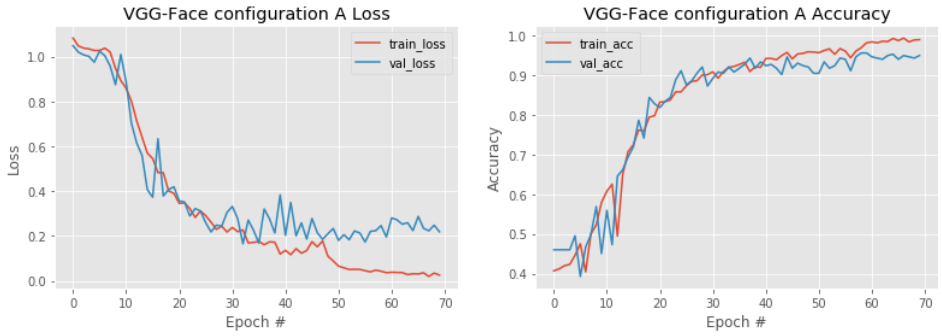
The optimal values for the initial learning rate, the patience (number of epochs in which validation accuracy is not increasing before reducing the learning rate by a scale of 10) and L_2 regularization coefficient are 10^{-2}, 10 and $5 \cdot 10{-}8$ respectively. In this case, validation accuracy is 94.89% and the plots of the loss, accuracy and learning rate as a function of the epoch index are given in Fig. 5.15.

From Fig. 5.15, we observe that there is only a small gap between the training and validation accuracy which means that the model is well-fitted. Moreover, the training accuracy has reached a plateau meaning that more training epochs are not needed.

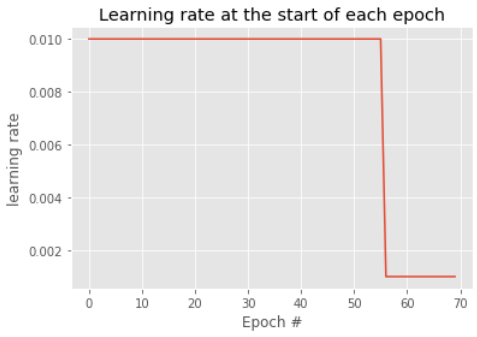
We then examined the deeper VGG-Face D configuration. The weights of all layers were initialized using the Glorot uniform (also known as Xavier uniform) method [53] as suggested in [37]. We set

(a) (b)

Figure 5.14: Online data augmentation for one example image from the VGG-Face dataset



(a) (b)



(c)

Figure 5.15: Some curves that visualize the training history of the best performing VGG-Face configuration A (learning rate $n = 10^{-2}$, *patience* $= 10$, L_2 regularization coeff. $\lambda = 5 \cdot 10^{-8}$). (a) Loss vs epoch index. (b) Accuracy vs epoch index. (c) Leaning rate vs epoch index.

Initial learning rate	Patience	L_2 regularization	Training accuracy (%)	Validation accuracy (%)
10^{-2}	10	0	98.32	94.57
$\mathbf{10^{-2}}$	**10**	$\mathbf{5 \cdot 10^{-8}}$	**98.88**	**94.89**
10^{-2}	10	$5 \cdot 10^{-4}$	98.72	93.61
10^{-2}	10	$5 \cdot 10^{-2}$	41.99	46.01
10^{-3}	10	$5 \cdot 10^{-8}$	42.01	46.01
10^{-1}	10	$5 \cdot 10^{-8}$	19.80	39.30
10^{-2}	0	$5 \cdot 10^{-4}$	42.01	46.01
10^{-2}	0	$5 \cdot 10^{-8}$	42.01	46.01
10^{-2}	20	0	98.72	94.25
10^{-2}	20	$5 \cdot 10^{-8}$	98.00	94.57
10^{-2}	20	$5 \cdot 10^{-4}$	96.63	91.05
10^{-2}	20	$5 \cdot 10^{-2}$	41.99	46.01

Table 5.10: Recognition results of VGG-Face A using 70 training epochs

the patience to 10 epochs as this value gave optimal results for the shallow A configuration and experimented with different L_2 regularization coefficients. The results are shown in Table 5.11.

Initial learning rate	Patience	L_2 regularization	Training accuracy (%)	Validation accuracy (%)
10^{-2}	10	0	98.48	93.93
10^{-2}	10	$5 \cdot 10^{-8}$	91.19	89.78
$\mathbf{10^{-2}}$	**10**	$\mathbf{5 \cdot 10^{-4}}$	**99.60**	**94.25**
10^{-2}	10	$5 \cdot 10^{-2}$	45.59	47.28

Table 5.11: Recognition results of VGG-Face D using 70 training epochs

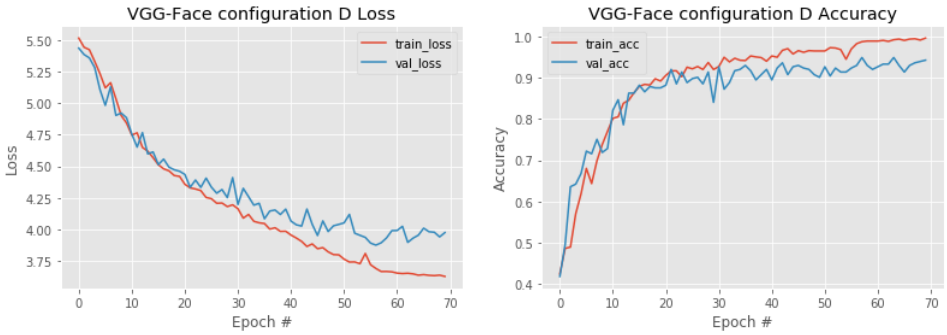
It is observed that the optimal L_2 regularization coefficient is equal to $5 \cdot 10^{-4}$ yielding a validation accuracy equal to 94.25%. Comparing Tables 5.10 and 5.11, we can see that with VGG-Face D the optimal validation accuracy rate is smaller which can be explained by the fact that a deeper architecture like this requires a larger training set to benefit from. The plots of the loss, accuracy and learning rate as a function of the epoch index are given in Fig. 5.16.

Again from Fig. 5.16 (b), we observe that the model learns to classify validation images pretty accurately as it does for the training images, so there is no overfitting and more regularization is not needed.

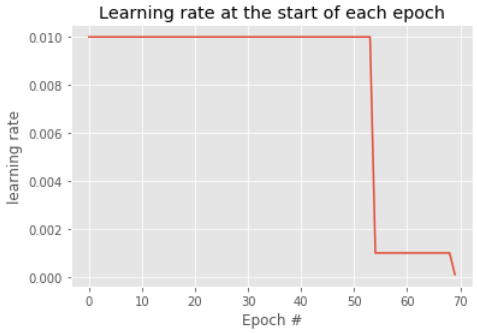
Using the publicly available pretrained weights of VGG-Face on the whole VGG-Face dataset, we tried to visualize some feature maps at different layers to get an insight of the features that this deep network learns (see Fig. 5.17). As expected, deeper layers learn more complex, higher level, representations of raw image pixel data which may not be intuitive to humans but are optimal for the final classification task. On the other hand, convolutional layers closer to the input of the model capture low level features like edges, simple curves or color.

5.3.2 Training for face verification (triplet loss)

For solving the face verification task, it was suggested in [5] to learn the face embeddings by taking advantage of the pretrained VGG-Face for face recognition. That is, we exploited the fact that face recognition and face verification are similar tasks and we didn't train from scratch. Learned facial features for face recognition can be useful for face verification as well. Specifically, we used the convo-



(a) (b)



(c)

Figure 5.16: Some curves that visualize the training history of the best performing VGG-Face configuration D (learning rate $n = 10^{-2}$, *patience* $= 10$, L_2 regularization coeff. $\lambda = 5 \cdot 10^{-4}$). (a) Loss vs epoch index. (b) Accuracy vs epoch index. (c) Leaning rate vs epoch index.

lutional and first two fully connected layers with their learned fixed associated weights by minimizing the cross-entropy loss and tweaked only the classifier part performed by the fully connected layer in the end. This strategy is known as transfer learning.

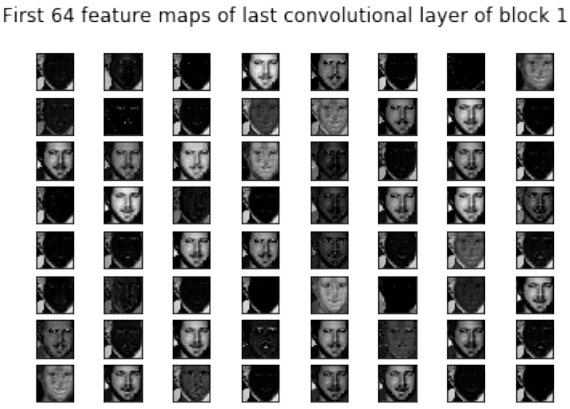
Initially, we tried triplet loss training using the weights of VGG-Face learned by our experiments and then we also tried to exploit the pretrained weights on the whole VGG-Face dataset. The output softmax layer was replaced by a fully connected layer with $L = 1024$ neurons and linear activation that learns the discriminative projection. During this stage, the rest of the pretrained network remains frozen and only this last fully connected layer is learned by minimizing the empirical triplet loss, given by:

$$E(W^{'}) = \frac{1}{|T|} \sum_{(a,p,n) \in T} max\{0, \alpha - \|\boldsymbol{x_a} - \boldsymbol{x_n}\|_2^2 + \|\boldsymbol{x_a} - \boldsymbol{x_p}\|_2^2\}, \quad \boldsymbol{x_i} = W' \frac{\boldsymbol{\phi}(l_i)}{\|\boldsymbol{\phi}(l_i)\|_2} \tag{5.6}$$

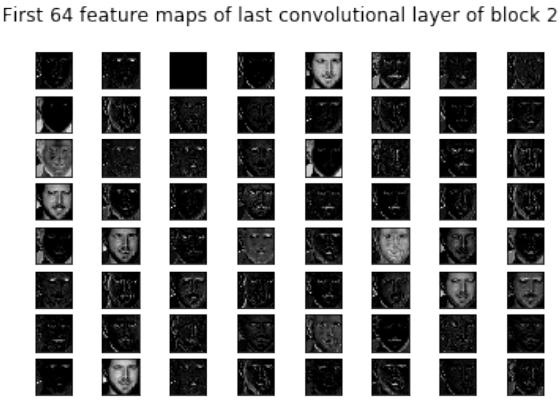
where α is the learning margin, T is the triplet training set that includes all anchor and positive pairs $(\boldsymbol{x_a}, \boldsymbol{x_p})$ that belong to the same person, $\boldsymbol{\phi}(l_i)$ is the output of the penultimate layer of the Siamese network and $\boldsymbol{x_n}$ is randomly selected among all negatives from T that violate the triplet loss margin.

The learnable parameters $W^{'}$ were to be learned on a subset of the VGG-Face dataset. Our initial experiments were using a small dataset of first three people in alphabetical order and then extended

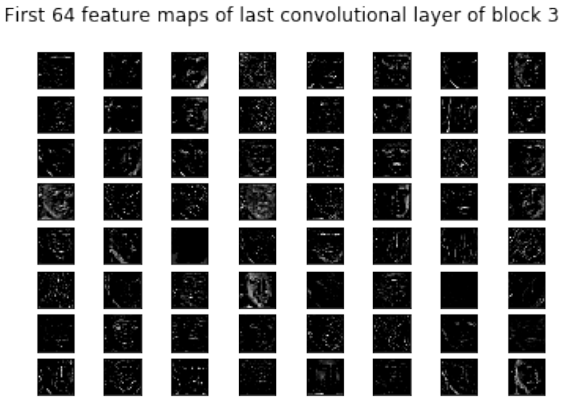
(a)



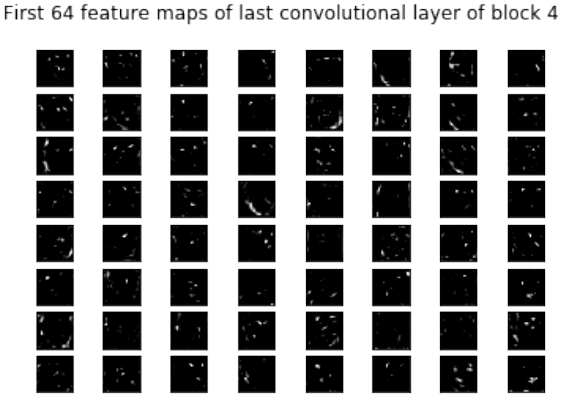
(b)



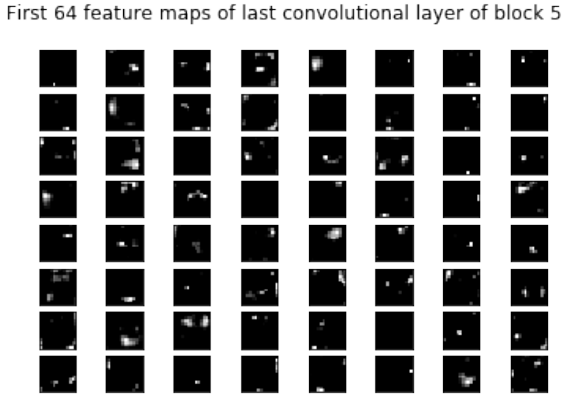
(c)



(d)



(e)



(f)

Figure 5.17: Some selected feature maps of the last convolutional layer of all blocks referring to VGG-Face D architecture and input image shown in (a)

to include images from 30, 600 and 1591 people.

First experiments with images from three people were all using 10 epochs and vanilla SGD for the optimization. For triplet mining, we randomly picked among negatives that violate the triplet loss margin for every anchor-positive pair from random batches of $P = 3$ people and $K = 20$ images per person. That is, each batch consists of 60 images and each epoch was defined to consist of 5 batches at the end of which the VGG-Face model is evaluated on LFW pairs.

For evaluation, the standard LFW benchmark was used, employing the outside data verification protocol which enables us to use external training data. Since alignment of face images is beneficial at test time as noted in [5], we focused on the funneled variant of the dataset [17].

At test time, the network was given pairs of still images l_1 and l_2 and decided if these two instances encoded by the feature vectors $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ given by Eq. 5.6 belong to the same identity or not. The decision was based on whether the distance $\|\boldsymbol{x_1} - \boldsymbol{x_2}\|$ was smaller than a threshold τ where τ is tuned on some validation data to maximize verification accuracy. Specifically for the case of LFW, we are given 6000 pairs of same and different pairs equally divided into 10 folds. As a result, each fold contains 300 pairs of images belonging to the same person (genuine pairs) and 300 pairs of images coming from different people (impostor pairs). The optimal decision threshold τ was found independently for each fold, by using the remaining 9 training splits and verification accuracy results were averaged over the 10 folds. The metrics used to evaluate and possibly compare the performance of VGG-Face with other state-of-the-art models were verification accuracy that simply computes the rate of the correctly classified test pairs as same or different and EER.

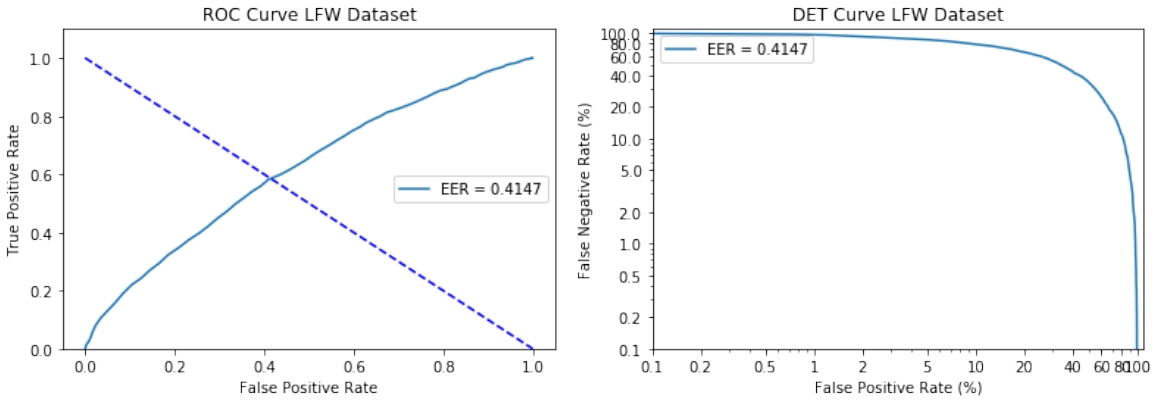
All subsequent results refer to VGG-Face D architecture (VGG-Face for simplicity).

We started our experiments by utilizing the weights found by our training procedure with the subset of 3 identities. The triplet loss margin was first set to $\alpha = 1$ and the learning rate was $n = 0.25$. After fitting the weights of the last projection layer, we obtained the following evaluation plots on the LFW dataset (Fig. 5.18). Verification accuracy was found to be $58.63\% \pm 2.20\%$ and the threshold which produced the $EER = 41.47\%$ was 1.0165. These numbers are really off the numbers that are mentioned in [5]($acc. = 98.95\%, EER = 0.87\%$) and this can be explained by the fact that only a small subset of the VGG-Face dataset was used for both tasks of face recognition and face verification causing the learned features of the frozen part of the face embedding model to be not that rich and the face verification model to collapse.

Then, we repeated exactly the same setting but now using the pretrained weights offered by the Visual Geometry Group of the university of Oxford. Results are shown in Fig. 5.19. Verification accuracy was found to be $87.07\% \pm 0.75\%$ and the threshold which produced the $EER = 15.11\%$ was 0.9329. These numbers are much better than the ones obtained before and this highlights the need of use of a large training set for the task of face recognition (prior to training for face verification) in order for the model to learn rich discriminative facial features. However, results are still worse than those mentioned in [5] since a much smaller training dataset was used for learning the face embeddings.

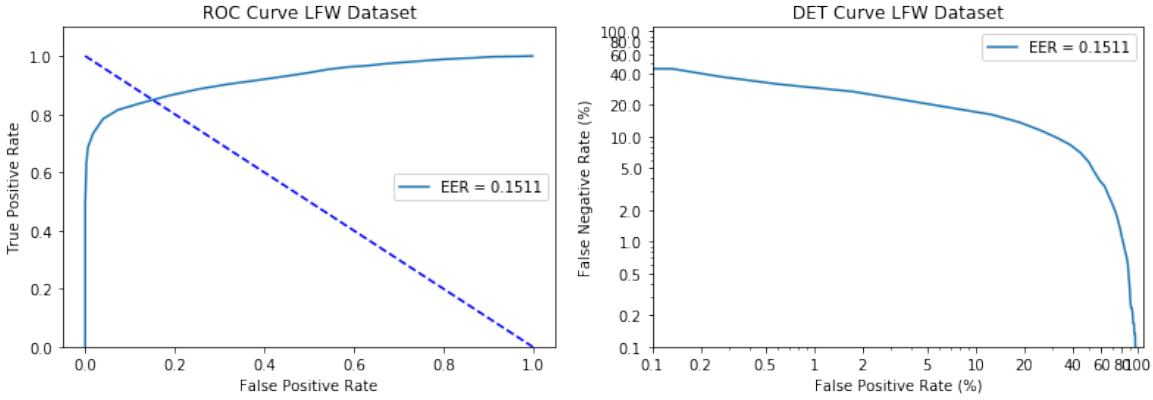
We then considered to see the influence of the triplet loss margin on the produced results. To that end, we repeated the training of VGG-Face with triplet loss using the pretrained weights from the VGG and setting $\alpha = 0.2$, value which was used in FaceNet [4]. Results are shown in Fig. 5.20. Verification accuracy was found to be $93.58\% \pm 1.20\%$ and the threshold which produced the $EER = 6.61\%$ was 0.8183. That is, reducing the triplet loss margin to 0.2 produced an even larger verification accuracy. This means that for the face verification problem at hand, enforcing the face embeddings of different people to be not that far away from each other provides better results.

We also experimented with the learning rate since this is often claimed as one of the most important hyperparameters in deep learning. As the learning rate $n = 0.25$ might be a bit high causing the



(a) (b)

Figure 5.18: Results of VGG-Face on LFW, with weights found by ourselves for face recognition, using triplet loss training, $\alpha = 1$, SGD with $n = 0.25$ and a training set of 3 identities from VGG-Face dataset. (a) ROC curve. (b) DET curve.

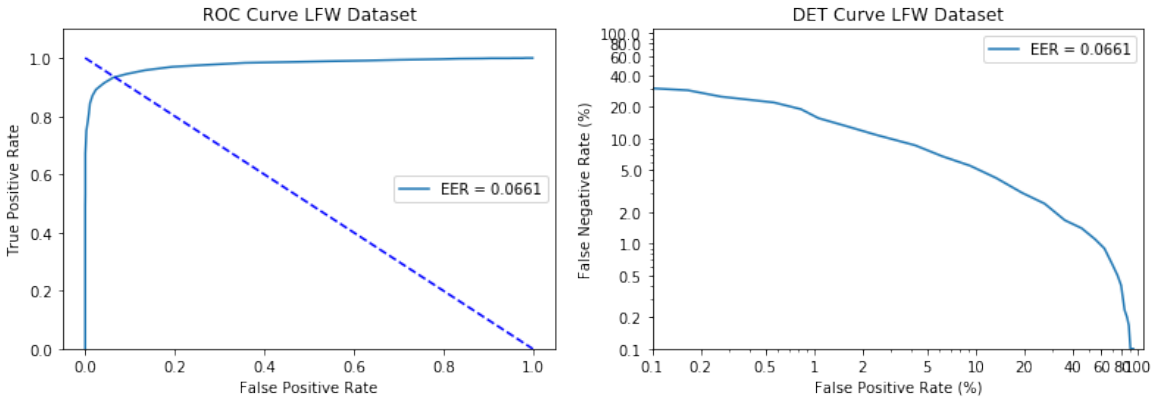


(a) (b)

Figure 5.19: Results of VGG-Face on LFW, with pretrained weights for face recognition, using triplet loss training, $\alpha = 1$, SGD with $n = 0.25$ and a training set of 3 identities from VGG-Face dataset. (a) ROC curve. (b) DET curve.

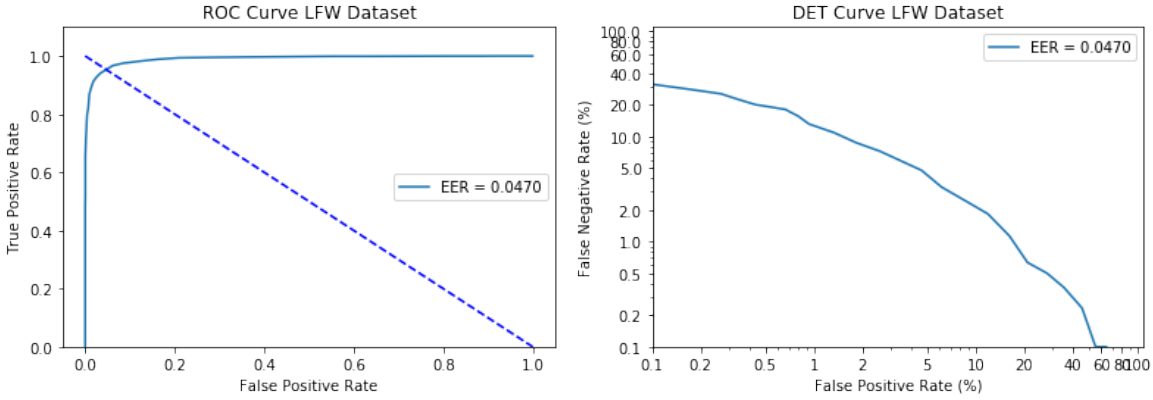
optimization algorithm to oscillate and not converge to a local optimum, we also tried minimizing the triplet loss setting $n = 0.01$. The corresponding ROC and DET curves showing the model performance on LFW images are shown in Fig. 5.21. The verification accuracy was found to be $95.07\% \pm 1.10\%$ and the threshold which produced the $EER = 4.70\%$ was 0.7898. That is, with $\alpha = 0.2$ and $n = 0.01$ we obtained the best results after the $10th$ epoch and we can further proceed with training for more epochs as the model may have not converged yet (and in fact it has not since the number of learnable parameters is $4096 * 1024 = 4,194,304$ which needs a reasonable amount of training).

As a last experiment, we attempted to see what the face verification performance is in case that no triplet loss is utilized and face features are extracted directly from the second to last fully connected layer of VGG-Face (4096 features per face image) followed by L2 normalization. Corresponding results are shown in Fig. 5.22. Verification accuracy was found to be $95.90\% \pm 0.77\%$ and the threshold which



(a) (b)

Figure 5.20: Results of VGG-Face on LFW, with pretrained weights for face recognition, using triplet loss training, $\alpha = 0.2$, SGD with $n = 0.25$ and a training set of 3 identities from VGG-Face dataset. (a) ROC curve. (b) DET curve.

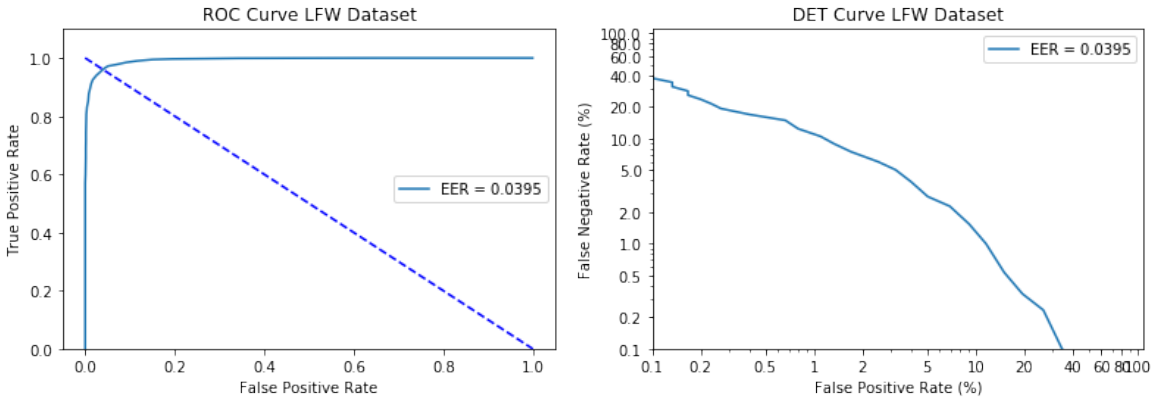


(a) (b)

Figure 5.21: Results of VGG-Face on LFW, with pretrained weights for face recognition, using triplet loss training, $\alpha = 0.2$, SGD with $n = 0.01$ and a training set of 3 identities from VGG-Face dataset. (a) ROC curve. (b) DET curve.

produced the $EER = 3.95\%$ was 1.2405. Consequently, results were even better with no triplet loss training and this also indicates that the face verification model trained with triplet loss and a training set of images from 3 subjects has not converged to a useful configuration with the beforementioned hyperparameter settings.

We next tried to see whether including a larger training set of 30 subjects from the VGG-Face dataset can further improve results. We experimented with both vanilla SGD and SGD with momentum optimizers, different learning rates n (0.25 and 0.01) and also including random flips and crops of size 224×224 when generating triplets from the training set. In the last case, to compute face embeddings at test time, an average of the feature vectors obtained from the four corners and the center with an additional horizontal flip (10 in total for each test scale) across 3 different scales of 256, 384 and 512 was also tried. Different scales were examined both separately and jointly. Results of the validation set



(a) (b)

Figure 5.22: Results of VGG-Face on LFW, with pretrained weights for face recognition, with no triplet loss training. (a) ROC curve. (b) DET curve.

(LFW) verification accuracy are shown in Table 5.12, where 10 training epochs were used. From Table 5.12, we see that SGD with $n = 0.01$ and data augmentation in the training set gives best results. We can also observe that taking a larger training set into consideration for triplet loss training improves performance as expected. Multi-scale augmentation at test time deteriorated results in contrast to findings in [37] which suggests that data augmentation is beneficial only in the training phase. This could be attributed to the fact that the test scale is quite different from the fixed training scale $S = 256$.

The training triplet loss and LFW verification accuracy as a function of the epoch index are shown in Fig. 5.23. The corresponding evaluation ROC and DET curves are presented next in Fig. 5.24. From Fig. 5.23, we see that both training triplet loss and mean accuracy on LFW fluctuate which indicate that a larger number of training epochs is necessary to see convergence and perhaps some amount of regularization. We examined both cases in the following by training for 30 epochs, first with no regularization at all and next with L_2 regularization with coefficient $\lambda = 5 \cdot 10^{-4}$. Results of verification accuracy and EER on LFW dataset are shown in Table 5.13 and it can be said that L_2 regularization is beneficial for learning face embeddings.

All above results are not satisfactory in the sense that learning embeddings for verification does not seem to improve verification accuracy compared to 95.90% accuracy we get without triplet loss training.

We first considered to increase the number of subjects in the training set to 600 and increase the number of training epochs to 150. In each batch, we sampled 5 images per subject from 300 people in total and we used the SGD optimizer with initial learning rate $n = 0.01$ which was dropped by 10 after every 30 epochs. We also used weight decay with $\lambda = 0.0005$ and data augmentation at training time. Verification accuracy was found to be $95.73\% \pm 1.16\%$ and EER was 4.32% while the associated threshold was 0.7819 .

We finally considered to further increase the training set so as to include face images from 1,591 subjects and experiment with the different optimizers among SGD, SGD with momentum, Adagrad and Adam. Learning rate decay was performed only in cases of SGD and SGD with momentum, while the adaptive optimizers were used with their default parameters (Adagrad with $n = 0.01$ and Adam with $n = 0.001$). For the case of SGD, we also experimented with a fixed learning rate $n = 0.01$ (no learning rate decay) combined with weight decay and learning rate decay with no weight decay. All the remaining parameters are kept the same as before. Results are shown in Table 5.14. We can notice that vanilla SGD gives the best results while the adaptive optimizers fall behind. Also, learning rate

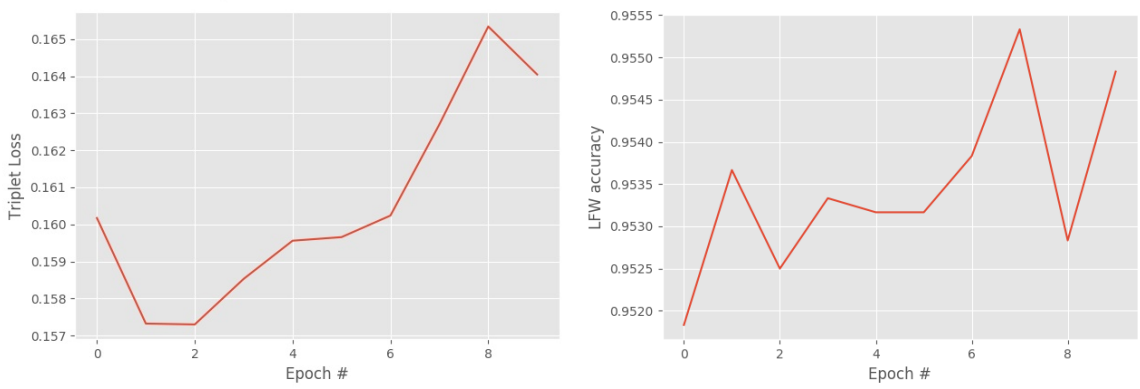
Optimizer	Random flips and crops (training set)	Flips and crops (test set)	LFW verification accuracy $(\mu \pm \sigma)(\%)$
Vanilla SGD $(n = 0.25)$	No	No	$93.77\% \pm 1.23\%$
Vanilla SGD $(n = 0.01)$	No	No	$95.57\% \pm 1.14\%$
SGD with momentum $(n = 0.25, \gamma = 0.9)$	No	No	$92.80\% \pm 1.22\%$
SGD with momentum $(n = 0.01, \gamma = 0.9)$	No	No	$94.65\% \pm 1.32\%$
Vanilla SGD $(n = 0.25)$	Yes	No	$93.82\% \pm 1.25\%$
Vanilla SGD $(n = 0.01)$	**Yes**	**No**	$\mathbf{95.62\% \pm 1.19\%}$
Vanilla SGD $(n = 0.25)$	Yes	Yes	Scale:256 $87.85\% \pm 1.44\%$ Scale:384 $71.52\% \pm 0.60\%$ Scale:512 $66.00\% \pm 1.20\%$ Multiscale: $81.35\% \pm 1.17\%$
Vanilla SGD $(n = 0.01)$	Yes	Yes	Scale:256 $87.32\% \pm 1.16\%$ Scale:384 $70.90\% \pm 0.68\%$ Scale:512 $66.18\% \pm 1.82\%$ Multiscale: $83.03\% \pm 1.59\%$

Table 5.12: Results of VGG-Face on LFW after training with triplet loss for 10 epochs using a training set of 30 subjects

L_2 regularization	Verification Accuracy $(\mu \pm \sigma)(\%)$	EER (%)
0	$94.85\% \pm 1.22\%$	4.81%
$5 \cdot 10^{-4}$	$95.12\% \pm 1.30\%$	4.98%

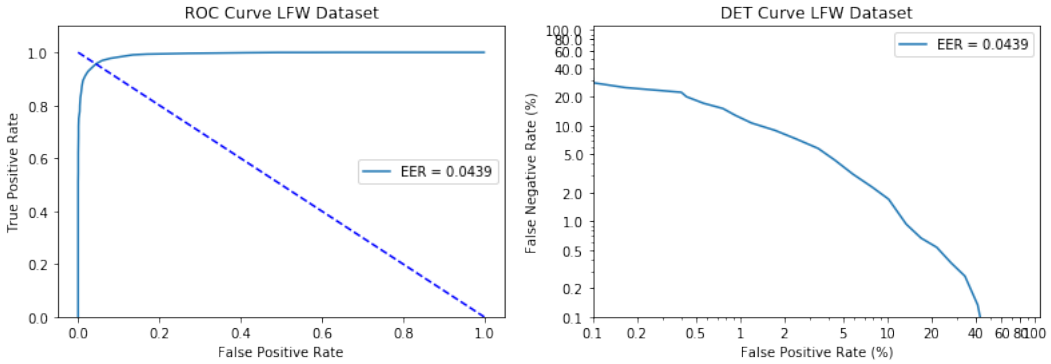
Table 5.13: Results of VGG-Face on LFW after triplet loss training for 30 epochs, with $\alpha = 0.2$, SGD with $n = 0.01$ and different L_2 regularization coefficients

decay and L2 regularization are shown to improve performance. However, results are still worse than those obtained with no triplet loss training. This can be attributed to some hyperparameters (like the triplet loss margin α, the leaning rate n and its associate schedule etc.) that are not optimally configured (see also Subsection 5.3.4).



(a) (b)

Figure 5.23: Some curves that visualize the training history of VGG-Face for triplet loss training, using a training set of 30 subjects, $\alpha = 0.2$, SGD with $n = 0.01$ and data augmentation during training. (a) Training triplet loss. (b) (Mean) LFW verification accuracy.



(a) (b)

Figure 5.24: Results of VGG-Face on LFW, with pretrained weights for face recognition, using triplet loss training, a training set of 30 subjects, $\alpha = 0.2$, SGD with $n = 0.01$ and data augmentation during training. (a) ROC curve. (b) DET curve.

5.3.3 Performance of VGG-Face on various image degradations

As we did for the traditional face recognition algorithms, we also desired to check how VGG-Face behaves in adverse conditions. To that end, we considered the same perturbed dataset we used earlier in our assignment, concerning the ORL face images and the degradations of blur, noise and illumination.

VGG-Face for face verification is expected to take in two face images and compare the distance of their embeddings to a threshold. To find the optimal decision threshold τ^* for the comparison of two face embeddings, we focused on the set of the original (not perturbed) face images. This set was divided into two subsets; 80% of the images were included in the training set while the rest were included in the test set. Out of all different VGG-Face models we tried in Subsection 5.3.2, we focused on

Optimizer	Verification Accuracy ($\mu \pm \sigma$) (%)	EER (%)
SGD	$95.75\% \pm 0.96\%$	4.35%
SGD with no learning rate decay	$94.75\% \pm 1.12\%$	5.35%
SGD with $\lambda = 0$	$95.65\% \pm 0.86\%$	4.28%
SGD with momentum	$94.33\% \pm 0.99\%$	5.59%
Adagrad	$92.25\% \pm 1.21\%$	7.86%
Adam	$91.60\% \pm 0.84\%$	8.53%

Table 5.14: Results of VGG-Face on LFW after triplet loss training for 150 epochs, using various optimizers

VGG-Face without triplet loss training since this configuration gave the best results on LFW in our experiments.

The face verification performance was evaluated on a number of different decision thresholds. More specifically, we examined 100 evenly spaced points between the minimum and maximum distance of face embeddings pairs from the training set. At each threshold value, all possible training face embedding pairs were classified as same or different identity and by comparison with the ground truth, the accuracy and F1 scores were obtained. Since the number of negative pairs is much larger than the number of positive pairs (for a database of P people each having K images, $\#negative\ pairs = \binom{P}{2}K^2 = \frac{P(P-1)K^2}{2}$, $\#positive\ pairs = P\binom{K}{2} = \frac{PK(K-1)}{2}$), the F1 score was used as the evaluation metric to pick the optimal threshold as it is more appropriate for skewed classification problems. Finally, using this threshold, all possible test face embedding pairs were classified as same or different and the test set verification accuracy was reported.

We next present the evaluation scores as a function of the threshold τ (Fig. 5.25), the histograms of the distance distributions of the positive and negative pairs (Fig. 5.26) and the ROC and DET curves of VGG-Face evaluated on the test set (Fig. 5.27).

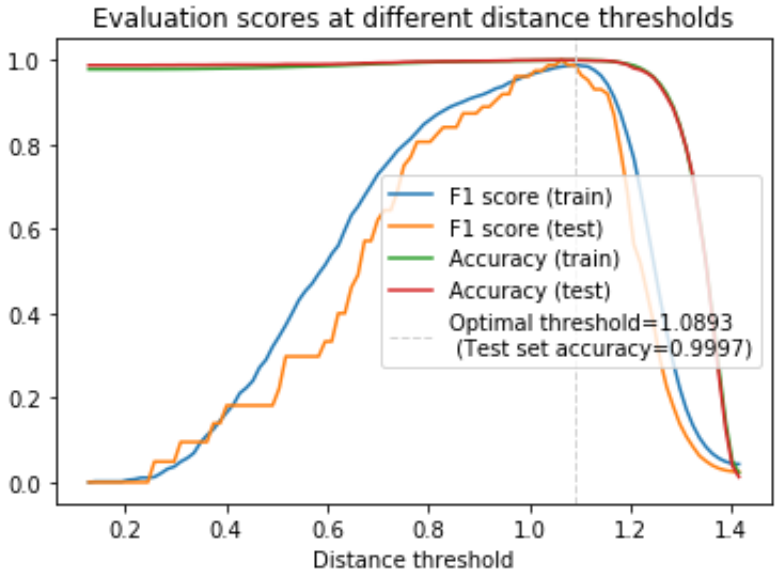


Figure 5.25: F1 and accuracy scores for both training and test sets as a function of the distance threshold τ

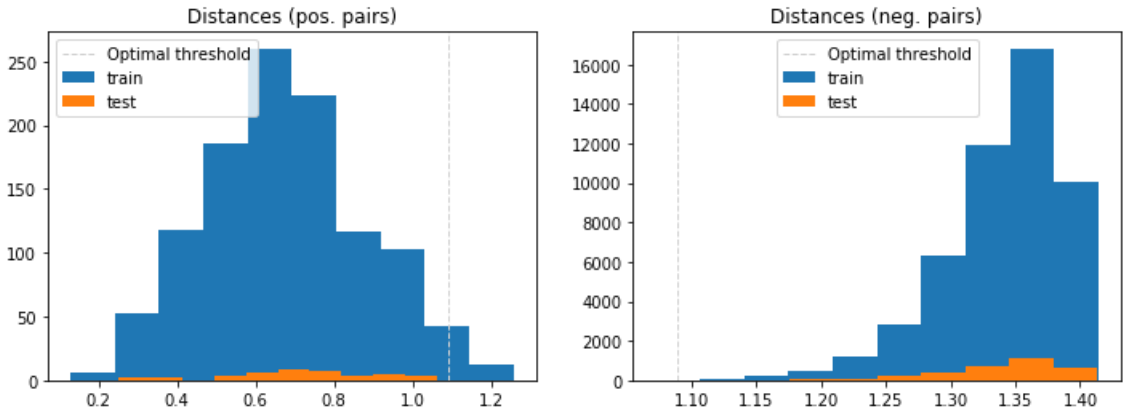
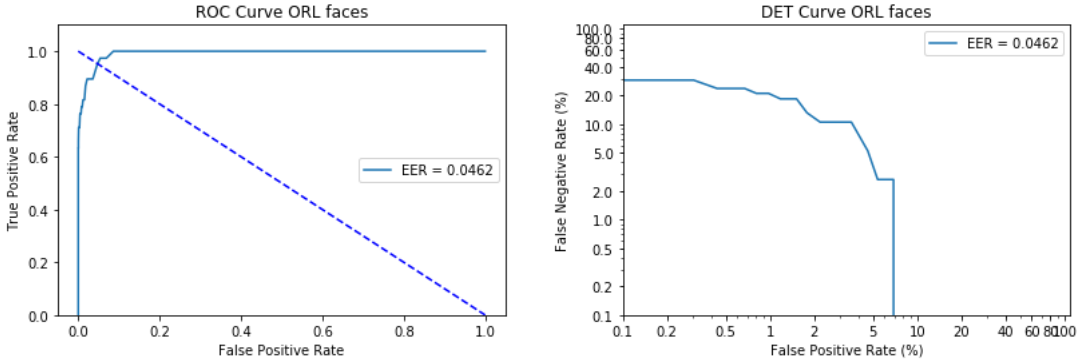


Figure 5.26: Histograms of the distribution of distances of positive and negative pairs



(a) (b)

Figure 5.27: Results of VGG-Face on ORL faces, using the optimal decision threshold $\tau^* = 1.0893$ as found from the training set. (a) ROC curve. (b) DET curve.

From Fig. 5.26, we can observe that there is a significant distinction between the distributions of distances of positive and negative test pairs at the optimal decision threshold which explains the excellent result of face verification accuracy being equal to 99.97%. However, it should be noted that even a baseline algorithm that always predicts different identity has an accuracy equal to 98.73% since there are 3120 negative and only 40 positive pairs in the test set.

Using this optimal distance threshold $\tau^* = 1.0893$, we calculated the verification accuracy of VGG-Face in two cases; where the input (query) pairs are drawn from various adverse conditions and where only the probe image of a test pair is degraded while the gallery image is derived from the original clean ORL dataset (see Table 5.15). We also calculated the EERs in both cases which are displayed in the DET curves of Fig. 5.28, 5.29, 5.30, 5.31 and 5.32, 5.33, 5.34, 5.35 correspondingly.

We see that as expected, verification performance drops when the query pairs' acquisition conditions are much degraded compared to the clean images of the original ORL faces. For small degradations of illumination (adding up to ± 50 to each pixel intensity), gamma corrections with $\gamma \in \{0.5, 2, 5\}$, blurs (using up to 7×7 averaging kernels) and noise levels (up to $p = 0.01 \implies SNR = 17\ dB$), performance is almost equal to that obtained on clean image pairs, but this is not the case for higher levels of degradation especially in the case of blur and noise. Verification accuracy and EERs are similar

Test case	Blur									
	3×3	5×5	7×7	9×9	11×11	13×13	NA			
Both degraded	0.9993	0.9986	0.9942	0.9245	0.4474	0.0410	NA			
Only one degraded	0.9993	0.9987	0.9934	0.9788	0.9750	0.9750	NA			
	Salt-and-pepper noise									
	17dB	15dB	14dB	13dB	12dB	NA				
Both degraded	0.9946	0.9882	0.9627	0.8741	0.6818	NA				
Only one degraded	0.9959	0.9907	0.9837	0.9789	0.9764	NA				
	Illumination; adding constant									
	-50	-40	-30	-20	-10	10	20	30	40	50
Both degraded	0.9969	0.9984	0.9990	0.9990	0.9991	0.9994	0.9994	0.9993	0.9983	0.9969
Only one degraded	0.9940	0.9976	0.9988	0.9992	0.9993	0.9994	0.9994	0.9994	0.9986	0.9972
	Illumination; gamma correction									
	$\gamma = 0.2$	$\gamma = 0.5$	$\gamma = 2$	$\gamma = 5$	NA					
Both degraded	0.9679	0.9993	0.9993	0.9983	NA					
Only one degraded	0.9889	0.9991	0.9993	0.9984	NA					

Table 5.15: Verification accuracy rates of VGG-Face on ORL images for different scenarios and two test cases; both degraded where both images of a test pair are degraded and only one degraded where only the probe image of a test pair is degraded

in both test cases and in all scenarios except for large blurs (9×9, 11×11 and 13×13 averaging filters), higher levels of salt-and-pepper noise (SNR = 14 dB, SNR = 13 dB and SNR = 12 dB) and gamma correction with $\gamma = 0.2$. Verification accuracy is generally higher in the latter case and this can be attributed to the fact that the associated decision threshold is optimized on clean images. On the other hand, EERs are generally worse when only one image of a pair is of lower quality especially for increasing noise levels and blurs implying that these conditions require at least that the gallery image which a probe image is to be compared to is not degraded.

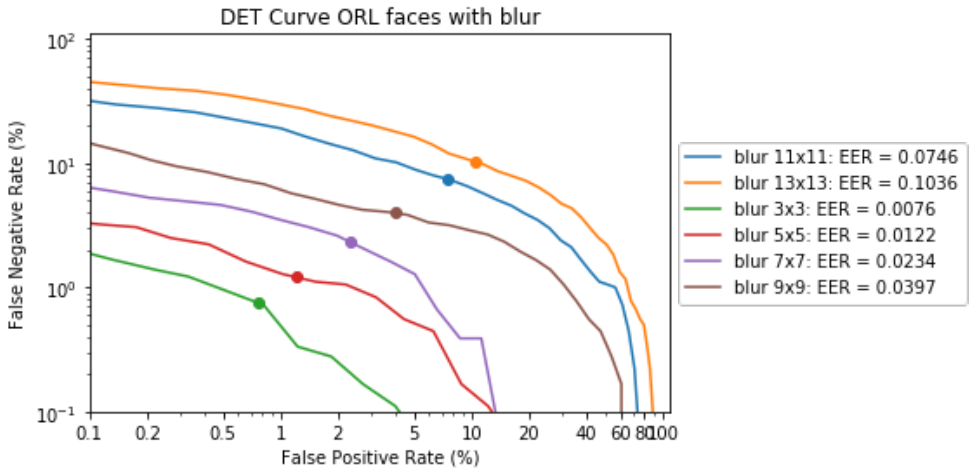


Figure 5.28: DET Curve of VGG-Face on ORL face images pairs for different averaging blurring filters (both degraded test case)

To be consistent with the comparison of VGG-Face features against the traditional machine learning algorithms of Section 5.2, we used the same classifiers on top of these deep learning based features. The original ORL images were again divided into two sets, training and testing, at a ratio 80% - 20% so as to derive the optimal parameters for the SVM classifiers, $C \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$ and $\gamma \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, based on the test set recognition accuracy. We experimented

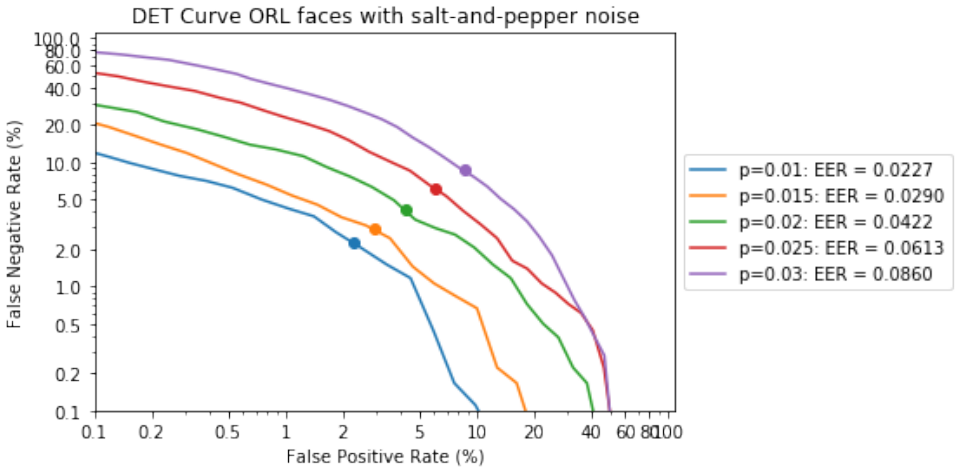


Figure 5.29: DET Curve of VGG-Face on ORL face images pairs for different salt-and-pepper noise probabilities (both degraded test case)

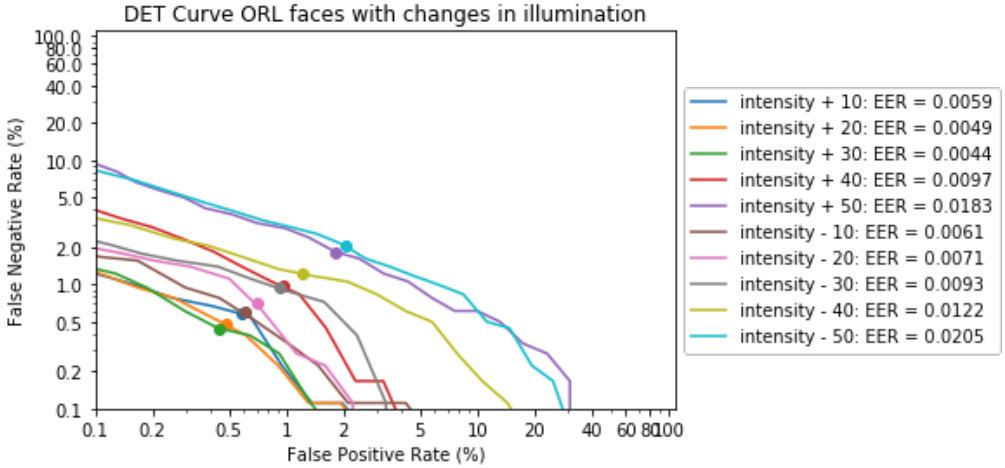


Figure 5.30: DET Curve of VGG-Face on ORL face images pairs for different adding illumination constants (both degraded test case)

with both full 4096 features and first 200, extracted by PCA features which account for 97% of the input variance. Training on the whole ORL dataset of clean images and testing on the same adverse conditions as before, we obtained the EERs of Table 5.16. We can see that considering the first 200 principal components gives slightly better results in general but differences are very small. All algorithms suffer mainly in the cases of noise and blur but are quite robust to changes in illumination conditions which implies that the learned VGG-Face features are rather invariant to them. Best results are obtained with a linear SVM classifier (with $C = 0.001$) using first 200 principal components which gives an average EER of 1.10% which is higher than the optimal one (0.97%) obtained before using a linear SVM on reduced by PCA Gabor features. This could be explained by the fact that although VGG-Face features are quite general and learned from a large variety of faces found in the VGG-Face dataset, they are not optimized to capture the differences between the faces of the specific ORL face dataset.

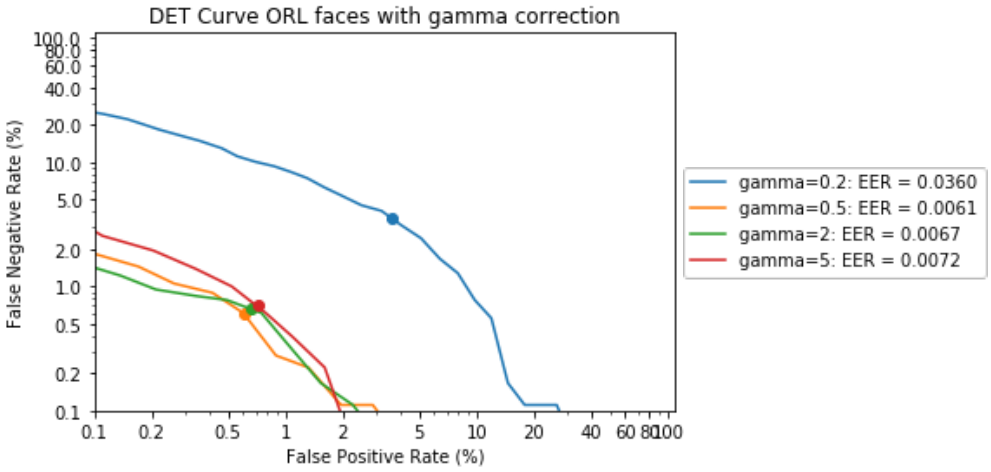


Figure 5.31: DET Curve of VGG-Face on ORL face images pairs for different gamma corrections (both degraded test case)

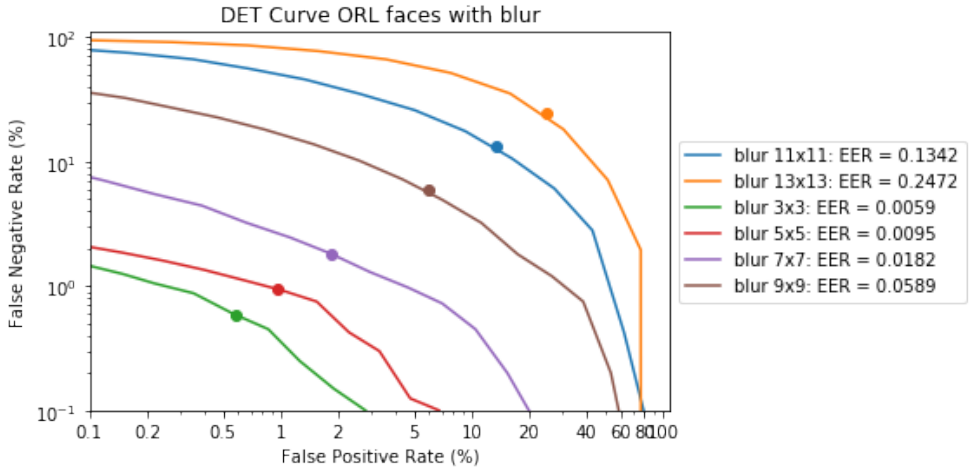


Figure 5.32: DET Curve of VGG-Face on ORL face images pairs for different averaging blurring filters (one degraded test case)

5.3.4 Training with Triplet Loss on MNIST Dataset

Since results of our experiments with VGG-Face trained with triplet loss are not satisfying, we considered a simpler, smaller-scale problem. Specifically, we used the MNIST dataset of handwritten images which comes with a training set of 60,000 grayscale images and a test set of 10,000 grayscale images. The image size is 28×28 pixels.

We first designed a simple CNN for digit classification. Its architecture is shown in Fig. 5.36 where dropout with probability $p = 0.2$ was applied on the fully connected layer prior to the output softmax layer. It was trained for 10 epochs, using the Adam optimizer and in the end it accomplished a 99.9% training and 98.48% test set accuracy.

The embedding model was created by first removing the output layer, L_2-normalizing the output of

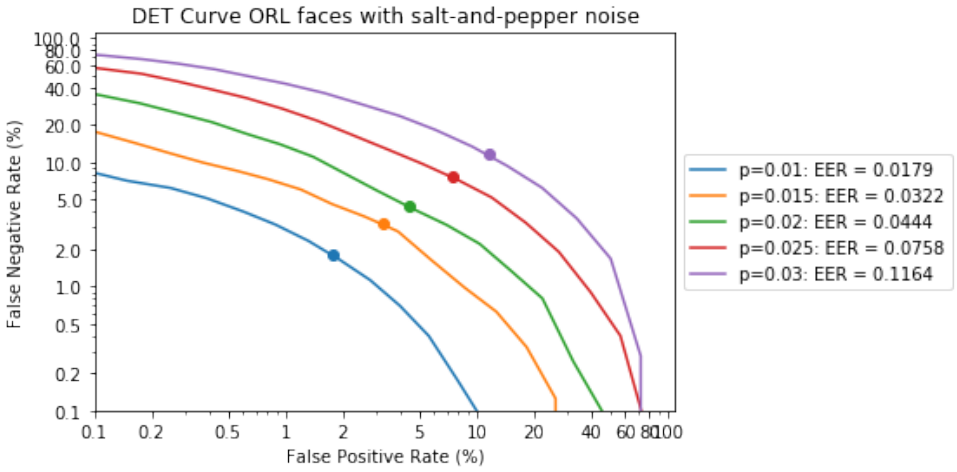


Figure 5.33: DET Curve of VGG-Face on ORL face images pairs for different salt-and-pepper noise probabilities (one degraded test case)

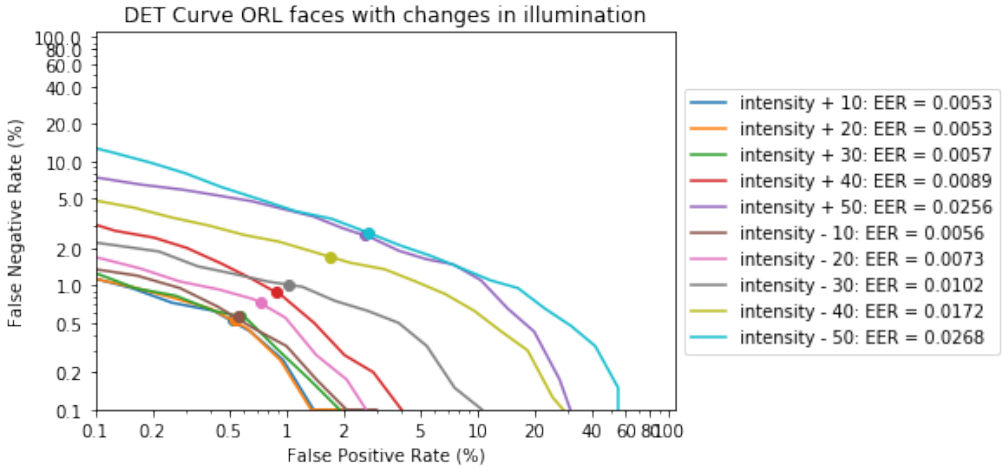


Figure 5.34: DET Curve of VGG-Face on ORL face images pairs for different adding illumination constants (one degraded test case)

the hidden layer with the 128 units and adding a fully connected layer with as many neurons as the desired size of digits' embeddings L. Again, the number of training epochs was set to 150, and in every epoch K random samples per digit from a number of P digits were selected 5 times in total to find the triplets to train on. For evaluation, a 10-fold cross validation scheme was performed, like in the case of LFW benchmark, where each fold consists of 300 same and 300 different pairs and the first image of the pairs shows the same digit in the whole fold.

As we wanted to see the influence of the number of the different classes seen in training with triplet loss, we experimented with different subsets of the training set of the MNIST dataset, considering first two, three, six and all ten digits (dataset breadth). This could be considered similar to our previous training of VGG-Face with triplet loss where we didn't use the whole original VGG-Face dataset. Digit verification accuracy results are shown in Table 5.17. Without triplet loss training, validation verification accuracy was found to be equal to $86.60\% \pm 4.10\%$. This means that triplet loss indeed makes digit embeddings more distinct and validation verification accuracy increases using at

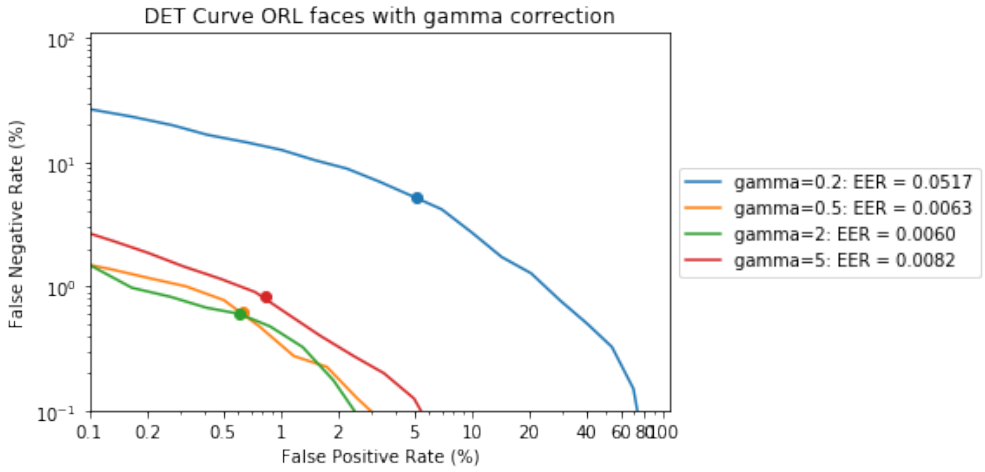


Figure 5.35: DET Curve of VGG-Face on ORL face images pairs for different gamma corrections (one degraded test case)

	No dimensionality reduction			PCA with 200 principal components		
	1NN	Linear SVM (C=0.001)	RBF SVM (C=0.001, $\gamma = 0.001$)	1NN	**Linear SVM (C=0.001)**	RBF SVM (C=0.001, $\gamma = 0.001$)
3×3	0	0	0	0	**0**	0
5×5	0	0	0	0	**0**	0
7×7	0	0.0019	0.0019	0	**0.0020**	0.0019
9×9	0.0454	0.0145	0.0147	0.0522	**0.0140**	0.0145
11×11	0.2298	0.0580	0.0620	0.2146	**0.0551**	0.0561
13×13	0.3966	0.1092	0.1152	0.3994	**0.1071**	0.1139
p=0.01 (17 dB)	0	0	0	0	**0**	0
p=0.015 (15 dB)	0	0.0025	0.0038	0.0026	**0.0025**	0.0038
p=0.02 (14 dB)	0.0148	0.0063	0.0076	0.0148	**0.0063**	0.0076
p=0.025 (13 dB)	0.0868	0.0222	0.0215	0.0805	**0.0218**	0.0219
p=0.03 (12 dB)	0.2144	0.0408	0.0418	0.1985	**0.0428**	0.0420
Intensity-50	0.0123	0.0025	0.0038	0.0099	**0.0025**	0.0042
Intensity-40	0	0	0	0	**0**	0
Intensity-30	0	0	0	0	**0**	0
Intensity-20	0	0	0	0	**0**	0
Intensity-10	0	0	0	0	**0**	0
Intensity+10	0	0	0	0	**0**	0
Intensity+20	0	0	0	0	**0**	0
Intensity+30	0	0	0	0	**0**	0
Intensity+40	0.0026	0	0	0.0026	**0**	0
Intensity+50	0.0196	0.0069	0.0073	0.0220	**0.0072**	0.0082
$\gamma = 0.2$	0.0123	0.0127	0.0131	0.0148	**0.0127**	0.0139
$\gamma = 0.5$	0	0	0	0	**0**	0
$\gamma = 2$	0	0	0	0	**0**	0
$\gamma = 5$	0	0	0	0	**0**	0
Averaged EER	0.0414	0.0111	0.0117	0.0405	**0.0110**	0.0115

Table 5.16: Equal Error Rates (EERs) of selected algorithms on VGG-Face features for different degradation conditions

least 4 or more digits in training. When the embedding size is $32 \times 1024 = 4,194,304$, the number of the trainable parameters is $4,194,304$ as in VGG-Face with 1024-D embeddings. Moreover, when $P = 10, K = 25$ the number of random triplets is 3000 as in VGG-Face with $K = 5$, $P = 300$. It is also observed that increasing the embedding size L generally improves performance with a great boost when going from 3 to 128 dimensions. The effect of the batch size in weight updates is not that critical except when all digits are presented in the training set where noisy updates with online gradient descent are better than mini-batch gradient descent updates. Adam optimizer consistently gives less accurately results compared to SGD but with faster convergence. Finally, the large standard deviation is explained by the fact that certain digits are ignored during training with triplet loss and as a result their embeddings are not optimized.

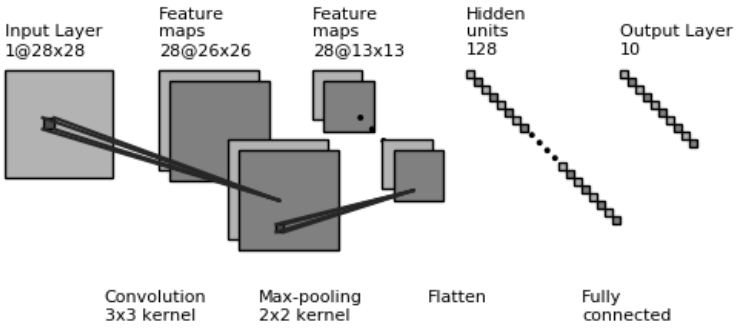


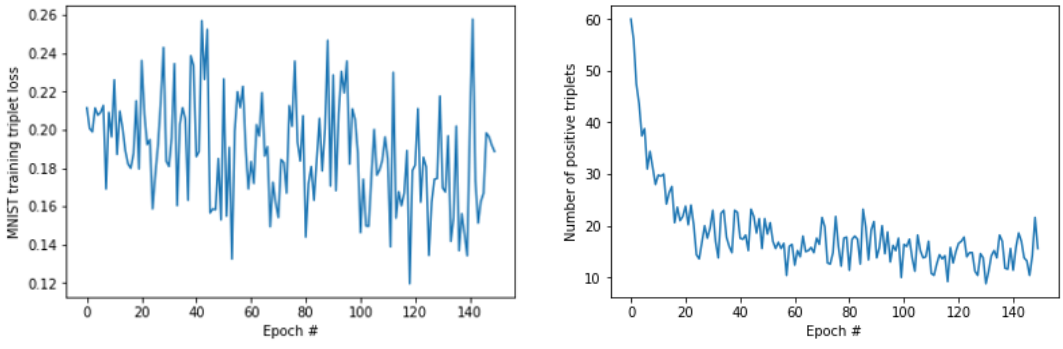
Figure 5.36: A shallow CNN for digit classification

Number of classes in training set	Optimizer (learning rate)	P	K	L	Batch size (for weight updates in optimization)	Verification Accuracy(%)
2	SGD (0.01)	2	5	3	32	$68.38\% \pm 11.56\%$
	SGD (0.01)	2	5	128	32	$85.68\% \pm 5.20\%$
	SGD (0.01)	2	5	1024	32	$87.00\% \pm 5.11\%$
	SGD (0.01)	2	5	128	1	$86.33\% \pm 5.03\%$
	SGD (0.01)	**2**	**25**	**32×1024**	**32**	**$87.12\% \pm 5.91\%$**
	SGD (0.01)	1	5	128	1	NA
	Adam (0.001)	2	5	128	1	$85.37\% \pm 4.45\%$
3	SGD (0.01)	3	5	3	32	$74.70\% \pm 12.59\%$
	SGD (0.01)	3	5	128	32	$87.15\% \pm 6.67\%$
	SGD (0.01)	**3**	**5**	**1024**	**32**	**$88.30\% \pm 6.28\%$**
	SGD (0.01)	3	5	128	1	$86.87\% \pm 6.91\%$
	SGD (0.01)	3	25	32×1024	32	$87.52\% \pm 7.31\%$
	SGD (0.01)	2	5	128	1	$86.98\% \pm 6.49\%$
	Adam (0.001)	3	5	128	1	$85.83\% \pm 7.45\%$
6	SGD (0.01)	6	5	3	32	$85.83\% \pm 7.07\%$
	SGD (0.01)	6	5	128	32	$91.23\% \pm 4.39\%$
	SGD (0.01)	6	5	1024	32	$91.80\% \pm 4.27\%$
	SGD (0.01)	6	5	128	1	$91.75\% \pm 5.71\%$
	SGD (0.01)	**6**	**25**	**32×1024**	**32**	**$92.60\% \pm 5.19\%$**
	SGD (0.01)	3	5	128	1	$91.57\% \pm 5.04\%$
	Adam (0.001)	6	5	128	1	$90.80\% \pm 5.64\%$
10	SGD (0.01)	10	5	3	32	$90.65\% \pm 3.32\%$
	SGD (0.01)	10	5	128	32	$94.88\% \pm 1.76\%$
	SGD (0.01)	10	5	1024	32	$94.83\% \pm 1.92\%$
	SGD (0.01)	**10**	**5**	**128**	**1**	**$97.30\% \pm 0.96\%$**
	SGD (0.01)	10	25	32×1024	32	$97.20\% \pm 0.95\%$
	SGD (0.01)	3	5	128	1	$96.07\% \pm 1.21\%$
	Adam (0.001)	10	5	128	1	$97.08\% \pm 1.14\%$

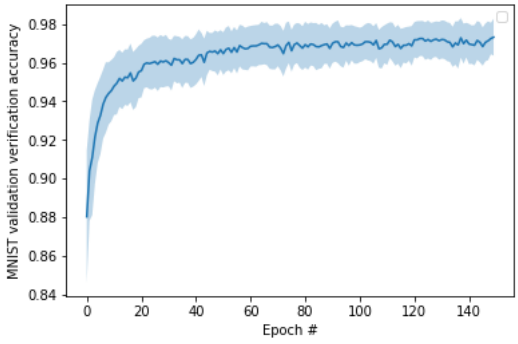
Table 5.17: Effect of the number of different classes (digits) seen in training (dataset breadth)

The evolution of the training triplet loss, the number of positive triplets (that is triplets that contribute to the triplet loss) in a random batch of PK images and the validation verification accuracy against the number of epochs is shown in Fig. 5.37 for the best performing network using all 10 digits during training. It is observed again that the triplet loss doesn't converge but this is normal as the training

set triplet loss in trained on is not fixed but instead random batches and associated triplets are fed to the network. A more appropriate plot to check convergence could be the number of positive triplets present in the random batches of PK images against the number of epochs. On the other hand, the validation verification accuracy is the metric to be tracked when training with triplet loss and it is observed that the validation verification accuracy converges after about 100 epochs.



(a) (b)



(c)

Figure 5.37: Some curves that visualize the training history of the best performing shallow CNN for digit verification. (a) Training triplet loss. (b) Number of positive triplets in a random batch of PK images. (c) Errorbar plot of the validation verification accuracy.

We also experimented with the number N of training examples from each class using all 10 digits in training. All classes were equally represented yielding a training set of $10N$ images. The training details were the same as the ones that yielded the optimal verification accuracy for the case of using all 10 digits during training from Table 5.17. The results are shown in Table 5.18. As expected, increasing the dataset depth generally improves results. However, using more than 1000 examples from each digit class doen't give much gain. Also, comparing Tables 5.17 and 5.18, we can infer that dataset depth is more crucial than dataset breadth for triplet loss training, conclusion which can also be extended for the case of VGG-Face.

The benefit of triplet loss training can also be understood by viewing the t-SNE embeddings of the raw images (before triplet loss training) and their embeddings (after triplet loss training). T-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique, particularly useful for visualizing high-dimensional data. This is achieved by minimizing the Kullback-Leibler divergence between two probability distributions that model similarity in the original high-dimensional and low-dimensional spaces. T-SNE was developed by Laurens van der Maaten and Geoffrey Hinton

Total Number of Training Samples: $10N$	Verification Accuracy (%)
100	$91.73\% \pm 1.91\%$
1000	$96.67\% \pm 1.20\%$
2000	$96.88\% \pm 0.86\%$
10000	$97.02\% \pm 0.96\%$
20000	$97.12\% \pm 0.74\%$
30000	$97.38\% \pm 0.91\%$
40000	$97.08\% \pm 0.97\%$
50000	$96.93\% \pm 0.92\%$
60000	$97.03\% \pm 1.16\%$

Table 5.18: Effect of the number of different examples (images) seen in training (dataset depth)

in 2008 [54] and unlike PCA which is also a dimensionality reduction technique, but linear, it is preferred in cases where the data is organized in non-linear manifolds. Moreover, since PCA maximizes variance, it preserves large pairwise distances while t-SNE preserves small pairwise distances.

The t-SNE embeddings of some training images before triplet loss training (raw data images) and after triplet loss training are shown in Fig. 5.38, where we can clearly see that after triplet loss training, images of the same digit form tight clusters and are well separated from images of different digits.

5.4 Online application of face verification

The assignment was concluded with an online application that takes in two face images and infers if they belong to the same person or not. Images are captured using a webcamera at a resolution of 640×480 pixels. Faces were detected and cropped from captured images using a pretrained (on a subset of LFW images) SVM linear classifier based on HOG features. For face alignment, 68 facial landmarks were detected in the cropped faces according to [16] and faces were wrapped so as specific landmarks (like the outer eyes and the tip of the nose in our case) to be placed at fixed positions. The face images of a given test pair were normalized by subtracting the per-channel means of the VGG-Face dataset and then they were fed to VGG-Face to extract facial features. The distance threshold used for the decision between same or different subjects was set to 1.2405 as it was found to yield the EER on the LFW benchmark.

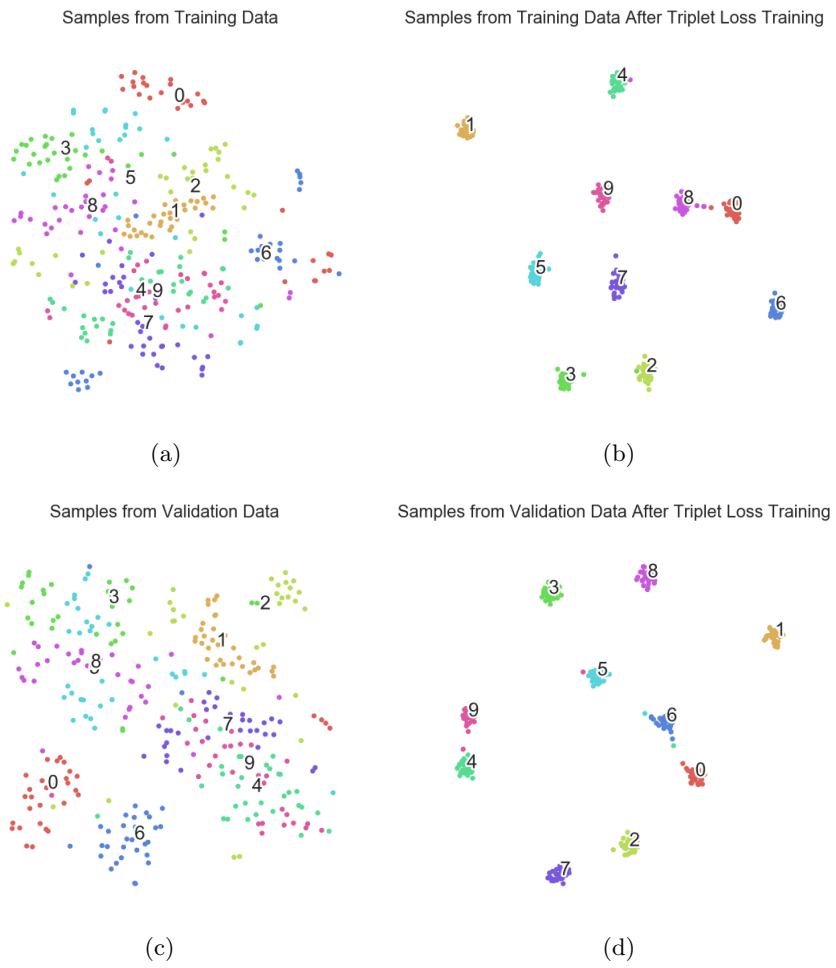


Figure 5.38: t-SNE embeddings of raw data (left) and after triplet loss training (right) with 128-D embeddings, 10 digits per batch, 5 images per digit and updating weights example by example. (top) Training images. (bottom) Validation images.

6 Conclusions and Future Work

Several traditional face recognition algorithms and VGG-Face, a state-of-the-art deep CNN have been studied in this project. Specifically, we were interested to see how robust these algorithms are under the adverse conditions of salt-and-pepper noise, blur and changes in illumination modeled by additive intensity constants and gamma correction.

As far as traditional face recognition is concerned, we experimented with various features and classifiers. For features, pixel intensities, local binary patterns, Gabor-based features and 2D DCT coeffiecients were considered. Dimenisionality reduction was also tried on the abovementioned features by PCA and LDA. As for the classification part, the simple distance-based NNC and NMC classifiers and the more advanced Linear SVM and Gaussian SVM classifiers were used for pixel intensities, Gabor-based and DCT features while for DCT also Gaussian Mixture Models with likelihood ratio test were examined. The UBM in the GMM approach was fit on Youtube videos which is a good source due to the variability of poses, expressions and illumination they include. Although these algorithms are expected to perform well in controlled environments (as shown with the ORL dataset), they are not that robust against various degradations especially those modelled by illumination changes. For face verification under adverse conditions, it was concluded that Gabor features combined with PCA and Linear SVM give an average (over all degradations) EER of 0.97% using the original ORL face dataset for training.

We also examined VGG-Face which is a 16 weight layer deep CNN that produced state-of-the-art results [5]. There, the authors showed that training VGG-Face with triplet loss instead of cross-entropy loss further improves face verification performance, but this was not observed in our experiments. The reason behind this is the non optimal tuning of some hyperparameters like the learning rate n, the L_2 regularization coefficient λ etc. as the exact same implementation of the triplet loss with the setting of random batches for training was shown to improve verification results for the simpler problem of digit verification. However, training VGG-Face with triplet loss was computationally and timely expensive and more experiments were not feasible without GPUs. VGG-Face model without triplet loss training was found to produce an EER of 3.95% on the LFW benchmark. The same model was used to see how it behaves when two face images are coming from the same adverse conditions and when one degraded image is compared to a not degraded one. It attained an average EER of 2.63% in the first case and an average EER of 3.80% in the latter case using again the ORL dataset for training (finding the optimal threshold). The higher EER in the second case can be explained by the fact that face embeddings are not so robust against different noise levels and blurs in contrast to illumination changes and as a result, in such cases, they are not close to face embeddings of clean images from the same subjects in the Euclidean space. We also experimented with the extracted VGG-Face features and NNC, Linear SVM and Gaussian SVM and it was seen that these features combined with PCA and Linear SVM give an average EER of 1.10%.

Further work is first needed to find the optimal values for the hyperparameters that enable VGG-Face to have a better perfromance when trained with triplet loss. Also, integrating other biometrics as well into the recognition procedure (like voice) may improve performance especially in cases where the probe images are captured in environments and conditions much different that those pertaining the gallery or training images. This combination of modes (feature-level or score-level fusion) is worth to be examined.

Bibliography

[1] S. Z. Li and A. K. Jain, *Handbook of Face Recognition*. Springer Publishing Company, Incorporated, 2nd ed., 2011.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.

[4] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

[5] O. M. Parkhi, A. Vedaldi, A. Zisserman, *et al.*, "Deep face recognition.," in *bmvc*, vol. 1, p. 6, 2015.

[6] M. Wang and W. Deng, "Deep face recognition: A survey," *arXiv preprint arXiv:1804.06655*, 2018.

[7] C. Sanderson and K. K. Paliwal, "Polynomial features for robust face authentication," in *Proceedings. International Conference on Image Processing*, vol. 3, pp. 997–1000, IEEE, 2002.

[8] S. Lucey and T. Chen, "A gmm parts based face representation for improved verification through relevance adaptation," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, pp. II–II, IEEE, 2004.

[9] T. Barbu, "Gabor filter-based face recognition technique," *Proceedings of the Romanian Academy*, vol. 11, no. 3, pp. 277–283, 2010.

[10] F. Bellakhdhar, K. Loukil, and M. Abid, "Face recognition approach using gabor wavelets, pca and svm," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 2, p. 201, 2013.

[11] L. Shen and L. Bai, "A review on gabor wavelets for face recognition," *Pattern analysis and applications*, vol. 9, no. 2-3, pp. 273–292, 2006.

[12] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[13] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005.

[14] C. Atanasoaei, "Multivariate boosting with look-up tables for face processing," tech. rep., EPFL, 2012.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[16] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1867–1874, 2014.

[17] G. B. Huang, V. Jain, and E. Learned-Miller, "Unsupervised joint alignment of complex images," in *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, IEEE, 2007.

[18] X. Tan and W. Triggs, "Enhanced local texture feature sets for face recognition under difficult lighting conditions," *IEEE transactions on image processing*, vol. 19, no. 6, pp. 1635–1650, 2010.

[19] R. Brunelli and T. Poggio, "Face recognition: Features versus templates," *IEEE transactions on pattern analysis and machine intelligence*, vol. 15, no. 10, pp. 1042–1052, 1993.

[20] A. Samal and P. A. Iyengar, "Automatic recognition and analysis of human faces and facial expressions: A survey," *Pattern recognition*, vol. 25, no. 1, pp. 65–77, 1992.

[21] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 586–591, IEEE, 1991.

[22] K. Etemad and R. Chellappa, "Discriminant analysis for recognition of human face images," *Josa a*, vol. 14, no. 8, pp. 1724–1733, 1997.

[23] M. S. Bartlett, J. R. Movellan, and T. J. Sejnowski, "Face recognition by independent component analysis," *IEEE Transactions on neural networks*, vol. 13, no. 6, pp. 1450–1464, 2002.

[24] M.-H. Yang, "Kernel eigenfaces vs. kernel fisherfaces: Face recognition using kernel methods.," in *Fgr*, vol. 2, p. 215, 2002.

[25] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 12, pp. 2037–2041, 2006.

[26] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 7, pp. 971–987, 2002.

[27] G. Guo, S. Z. Li, and K. Chan, "Face recognition by support vector machines," in *Proceedings fourth IEEE international conference on automatic face and gesture recognition (cat. no. PR00580)*, pp. 196–201, IEEE, 2000.

[28] S. S. Haykin *et al.*, *Neural networks and learning machines/Simon Haykin.* New York: Prentice Hall,, 2009.

[29] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted gaussian mixture models," *Digital signal processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

[30] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[31] C. McCool, R. Wallace, M. McLaren, L. El Shafey, and S. Marcel, "Session variability modelling for face authentication," *IET biometrics*, vol. 2, no. 3, pp. 117–129, 2013.

[32] R. Wallace, M. McLaren, C. McCool, and S. Marcel, "Intersession variability modelling and joint factor analysis for face authentication," 2011.

[33] R. Vogt and S. Sridharan, "Explicit modelling of session variability for speaker verification," *Computer Speech & Language*, vol. 22, no. 1, pp. 17–38, 2008.

[34] R. Wallace and M. McLaren, "Total variability modelling for face verification," *IET biometrics*, vol. 1, no. 4, pp. 188–199, 2012.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

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

[38] "Color feret database." https://www.nist.gov/itl/iad/image-group/color-feret-database.

[39] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "Multi-pie," *Image and Vision Computing*, vol. 28, no. 5, pp. 807–813, 2010.

[40] "Yale face database." http://vision.ucsd.edu/content/yale-face-database.

[41] "At&t database of faces." https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[42] "Mit-cbcl face recognition database." http://cbcl.mit.edu/software-datasets/heisele/facerecognition-database.html.

[43] A. M. Martinez and R. Benavente, "The ar face database," *Tech. Rep. 24 CVC Technical Report*, 01 1998.

[44] C. Sanderson and B. C. Lovell, "Multi-region probabilistic histograms for robust and scalable identity inference," in *International conference on biometrics*, pp. 199–208, Springer, 2009.

[45] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," 2008.

[46] R. Min, N. Kose, and J.-L. Dugelay, "Kinectfacedb: A kinect database for face recognition," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 11, pp. 1534–1548, 2014.

[47] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 343–347, IEEE, 2014.

[48] L. Wolf, T. Hassner, and I. Maoz, *Face recognition in unconstrained videos with matched background similarity*. IEEE, 2011.

[49] L. Montesinos García, "Audio-visual authentication for mobile devices," 2018.

[50] V. Rani, "A brief study of various noise model and filtering techniques," *Journal of global research in computer science*, vol. 4, no. 4, pp. 166–171, 2013.

[51] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[52] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki, "The det curve in assessment of detection task performance," tech. rep., National Inst of Standards and Technology Gaithersburg MD, 1997.

[53] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[54] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.