Siamese Neural Networks for Biometric Hashing

by

Matin Azadmanesh

Submitted to
the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

SABANCI UNIVERSITY

Augest 2014

Siamese Neural Networks for Biometric Hashing

APPROVED BY

Assoc. Prof. Dr. Hakan ERDOĞAN .....
(Thesis Supervisor)

Prof. Dr. Ş. İlker BİRBİL .....

Assoc. Prof. Dr. Müjdat ÇETİN .....

DATE OF APPROVAL: .....

To my family....

Acknowledgements

I want to show my sincere gratitude to my advisor Professor Hakan Erdoğan for his support and kind pieces of advice. I also want to send my special thanks to my dear brother, Masih, for his caring consulting, and a special friend, Maryam, for her patience and support during writing this thesis.

And of course I am thankful to my family in Iran, professors who I learned a lot from and friends in Sabanci University, specially VPA lab.

I would like to thank the jury members: Professor İlker Birbil and Professor Müjdat Çetin for their precious time reading my thesis and for their constructive comments.

Siamese Neural Networks for Biometric Hashing

Matin Azadmanesh

EE, M.s. Thesis, 2014

Thesis Supervisor: Hakan Erdoğan

Keywords: Siamese neural network, neural network, Biometrics, Hashing, security.

Abstract

In this project we propose a new method for biometric hashing. In our method we use a deep neural network structure to learn a similarity preserving mapping. For this purpose we train a neural network architecture that consists of two identical neural nets called Siamese neural nets where each one performs the mapping for hashing. The weights are tuned in training such that two different biometric data of a person yield a similar code but codes corresponding to different subject's images are far away. The neural network outputs a pre-hash vector which is then converted to a biometric hash vector through random projection and thresholding. We use angular distance measure to train pre-hash vectors which is more related with the Hamming distance that is used between hashes in verification time. Our experiments show that the introduced method can outperform a PCA-based baseline. We also show that a biometric hashing system which was trained using the angular distance can achieve better verification rates than another one trained with the Euclidean distance.

Matin Azadmanesh

EE, Master Tezi, 2014

Tez Danışmanı: Hakan Erdoğan

Özet

Bu proje biyometrik kıyım için yeni bir yöntem tanımlanmasını içermektedir. Önerdiğimiz metod, bir derin sinir ağı (deep neural network) yapısı aracılığı ile benzerlik koruyan bir ilişkilendirme öğrenilmesi sağlar. Bu, kıyım için ilişkilendirme yapan iki birbirine eş (ikiz) yapay sinir ağı içeren bir ağ mimarisi sayesinde mümkün kılınmıştırç. Yapay sinir ağı parametreleri öğrenme asamasında aynı ınsanın resimlerini birbirine benzer, farklı insanların resimlerini birbirinden farklı kodlara indirgeyecek şekilde ayarlanmaktadır. Yapay sinir ağından elde edilen önkıyım çıktısı, rastgele yansıtma ve sınırlandırma yöntemi ile biyometrik kıyım dizinine çevirilmektedir. Önkıyım dizinleri öğrenimi sırasında, doğrulama esnasında kullanılan Hamming uzaklık ölçütü ile ilişkili olan açısal uzaklık ölçütü kullanılmıştır. Deneylerimizin neticesinde önerdiğimiz metodun PCA temelli baz alınan yöntemlere nazaran daha iyi çalıştığı gözlemlenmiştir. Ayrıca açısal uzaklık ölçütü kullanan bir ikiz ağın Öklidyen uzaklık kullanan alternatifine göre daha iyi doğrulama başarımı elde ettiği gösterilmiştir.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Security is one of the most important concepts in modern world. One aspect of this problem is authentication which we try to improve in this project. Many systems rely on biometric data for authentication. It has many advantages like we always have the biometric property like our finger print or face features, potentially it is safe, etc.; however, there are some serious problems. One of the most important issues for such systems is that no system is safe enough to store our biological identity information. A possible solution developed to tackle the problem is called **biometric hashing**.Biometric hashing uses a pseudo-random transformation to transform biometric data into a fixed length binary hash vector that is specific to each user of the system [1]. The naive method for an authentication system is saving raw biometric data during enrollment and during authentication, comparing the input biometric with the stored one. A more secure way would be to use a hashing function applied to the raw data and save the resulting hash in the system. During authentication, the hashing function acts on the new input and the resulting hash is compared with the stored one. To improve the security of the system, we can add a random parameter to the hashing process, i.e. assign a pseudo-random vector (a key) to each user and make her hash vector depend on this secret vector. This method is sometimes called **salt hashing** (adding the random parameter to the hash vector like *salt*) and this could provide high level of security for the user since it is not easy to go back from the hash to the original biometric data without knowing the pseudo-random secret key.

The implementation of this method for face data, which is discussed in detail in [2], is based on a predefined mapping and multiplying with the pseudo-randomly generated matrix for salting.

One of the problems here is robustness of the system under different lighting conditions, different facial expressions, changes through time, facial accessories (scarf, glasses, etc.). Such invariancy is hard to obtain through a pre-defined linear mapping.

Another problem with this method is that the function is very computationally simple (a linear mapping, mostly based on principle component analysis of the data). It is easy to come up with security attacks against this method when we assume that the user's secret key is compromised. We think that even if the secret key is obtained by an attacker (without the true biometric data), it should be difficult to break system security and privacy. Currently, when an attacker obtains the secret key of a user, the performance of biohashing is not very good. So, in this thesis, we specifically address the case when the secret key is known by an attacker.

The method we propose for biometric hashing is based on salt hashing which also benefits from recent developments in deep learning. Here we train a similarity preserving mapping then salt it with a random string. Since neural networks are very capable models, in presence of enough training data we can have such a complex function that keeps the images belonging to one person close to each other and far from the other's. We show that one can obtain better verification performance from the system even if the secret key of a user is stolen by an attacker. It also makes it difficult for attackers to obtain biometric data from biometric hashes since the used transform is highly nonlinear which increases the privacy of the system.

1.2 Contributions

The main contribution in this work is to develop a biometric hashing system based on the Siamese neural network architecture. Our second contribution is to modify the **Siamese network** by using a new objective function based on angular distances which is more suitable for biometric hashing purposes. We calculate and present the gradient of the new angular distance based objective function with respect to the weight parameters of the neural network.

Here we trained a very special neural networks (Siamese architecture) with huge number of parameters for a specific application, Studding neural networks with many hidden layers is a new field of research with lots of open problems. Training such networks involves with tuning many hyper-parameters like number of hidden layers and elements in each layer, optimization algorithm and its own parameters and etc., as far as the author knows, there is no general answer to these questions and using different "techniques" to optimize these hyper-parameters was another issue we dealt with and report the results in this project.

1.3 Outline

We describe the biometric hashing problem and its solution in Chapter 2. In the Chapter 3 we give a brief introduction to neural networks and deep learning. The special neural network architecture called the "Siamese architecture" which is used to solve the biometric hashing problem is introduced in Chapter 4. We report the results of experiments on different face datasets and compare them with the existing model and then the effects of different parameters on the performance of the system is discussed in Chapter 5. Finally in Chapter 6 we give a conclusion and summary of our work.

Chapter 2

Background and Problem Statement

2.1 Biometrics

"Biometrics" means "life measurement", more often it is used as unique physiological characteristics to identify a person. Security is one of the most important applications and use of biometrics. Identifying "who" you are interacting with is an important human feature for computers as well [3]. Many biometric methods have been developed and are used for identification purposes. The idea is to use the special characteristics of a person to identify him. Here special characteristics means features like face, iris, fingerprint, signature etc. [3].

A biometric system can be seen as a pattern recognition system that identifies people by determining the authenticity of a specific physiological characteristic provided by the user [4].

Verification basically is a bi-class classification task, i.e. one person claims an identity, the system may respond "yes" or "no". On the other hand identification task can be seen as multi-class classification, in this scenario the enrolled biometrics should be classified to one class (registered person).

A biometric system can serve as :

- **Identification**: Biometrics can be used to determine a person's identity. These tasks are multi-class classification problems.
- **Verification**: Biometrics can also be used to verify a person's identity, as in the system promising secure access to a bank account at an ATM by using retinal scan.

Biometric authentication requires enrollment of a valid user where the user provides a biometric template to store in the system. This enrollment can be seen as a three-step process.

1. Capture,
2. Process,
3. Store.

Biometric data is first captured by a sensor such as camera which is followed by processing to extract a biometric feature vector. Extracted feature from the raw data is called biometric sample or biometric template. This template is stored in a database which completes the enrollment. During verification or identification, the user provides her biometric data again. After capturing the raw biometric data, it is processed to obtain the test template and then a comparison with the stored template is performed. In many applications it is important that the original biometric sample cannot be reconstructed from extracted feature [3]. The current project targets this issue in verification systems.

2.1.1 Face recognition system

Face recognition is one of the most challenging areas in the field of computer vision. Face detection as the first step for this task should be done in order to localize and to extract the face region from the background. For face detection, active contour models referred as snakes [5], or approaches based on Hough method [6] and skin-color information [7], boosted cascade of Haar-transform based features (Viola-Jones method) [8] or mixed methods are being used to detect the edges and for locating the face boundary [9].

2.1.2 Face verification

There are many different algorithms for face recognition. The called "Eigenface" [10] is one of the most popular ones. In the eigenfaces approach, we represent each face image with an N-dimensional vector $\boldsymbol{x}_i, i = 1, 2, ..., n$. The covariance matrix of all the n samples can be computed as $\boldsymbol{S} = \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$, where μ is the mean face vector. The k largest eigenvalues of $\boldsymbol{S}$ are called the principal values and the eigenvectors corresponding to them are called the principal directions of $\boldsymbol{S}$, where $k \ll n$. For each image $\boldsymbol{x}$, we obtain a feature vector $\boldsymbol{Y}$ by projecting $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{\mu}$, the mean-subtracted image, onto the subspace generated by these principal directions, that is $\boldsymbol{Y} = \boldsymbol{P}\tilde{\boldsymbol{x}}$ where the $\boldsymbol{P}$ matrix has the principal direction vectors of S in its rows. [11]

Biometric samples obtained like this are stored in the dataset as a template for each user and are used during the authentication time.

2.2 Performance metrics

By accuracy of a biometric system we mean how accurate this system can separate authentic persons from impostors. It means that the system should reject the impostors and accept genuine samples as much as possible. There are several quantities which can be used to report the accuracy of a biometric system [12].

The verification system compares an input biometric feature vector to the claimed user's stored template in the database. A score is calculated which indicates similarity (the higher the similar). Instead of a score, a distance can be calculated as well in which case a lower distance indicates more similarity. The decision is based on comparing this score or distance to a (possibly varying) threshold. If the matching score/distance is higher/lower than the threshold, the claim of identity is accepted, if not it is rejected. Based on this, there are two possible types of errors: false acceptance and false rejection.

- **False acceptance rate (FAR):** The probability that the system incorrectly accepts a false input biometric as belonging to the claimed user. It measures the likelihood of impostor inputs which are incorrectly accepted. If the person is an impostor in reality, but the matching score is higher than the threshold he will be treated as genuine.

- **False rejection rate(FRR):** The probability that the system fails to accept a genuine input. It measures the percent of valid inputs which are incorrectly rejected.

  It is clear that these two types of errors are traded off by changing the decision threshold. To observe how the errors change with respect to each other as the threshold is varied we use the following plots.

- **Receiver operating characteristic or relative operating characteristic (ROC)**: The ROC plot is a visual characterization of the trade-off between the true accept rate, which is equal to one minus false reject rate, and the false accept rate. By decreasing the threshold we will see fewer false rejects but more false accepts. Conversely, a higher threshold will reduce the FAR but increase the FRR.

- **Detection error trade-off (DET):** DET graph is another graphical plot of error rates for binary classification systems, plotting false reject rate vs. false accept rate. DET curve is equivalent to an ROC curve but directly plots two error types against each other. Generally the x- and y-axes are scaled non-linearly by their standard normal deviates.[**?**]

  If we would like to summarize the performance using a single value instead of a plot, we can use the equal error rate.

- **Equal error rate(EER):** The rate at which both acceptance and rejection errors (that is FAR and FRR) are equal. The value of the EER can be easily obtained from the ROC or DET curve. The EER is a quick way to compare the accuracy of devices with different ROC curves. In general, the device with the lowest EER is the most accurate, hence the objective of our system is to design a system with smaller EER. [12]

2.3 Biometric hashing based verification system

Biometric hashing (or biohashing) is one of the biometric template protection methods. Biohash is a binary and pseudo-random representation of a biometric template.Biometric hashing methods use two inputs:

1. Biometric template.

2. User's secret key.

A biometric feature vector is transformed into a (lower dimensional) sub-space using a pseudorandom set of orthogonal vectors which are generated from the user's secret key. Then, the result is binarized to produce a bit-string which is called the Biohash [13]. In an ideal case, the distance between the biohashes belonging to the biometric templates of the same user is expected to be relatively small. On the other hand, the distance between the biohashes belonging to different users is expected to be sufficiently high to achieve lower false acceptance rates. The desired properties of the biohashes are summarized as follows:

1. The biohash should be irreversible so that biometric template cannot be obtained from a biohash vector.

2. The biohash should be cancelable so that it can be renewed when an attacker steals it.

3. The biohash should be robust against different biometric images belonging to the same user so that the Hamming distance between the biohash vectors (i.e. generated from the same secret key but different biometric images collected at different sessions) of the same user should be small.

4. Biohash should be fragile to the biometric images which do not belong to the same user so that the Hamming distance between the biohash vectors (i.e. generated from different secret key and different biometric image) of the different users should be high [14],[15], [16].

2.4 Random projection biohashing

Random projection (RP) based biohashing scheme is proposed by Ngo et al. [17][14]. In a RP based biohashing method, there are three main phases in each stage and these phases are described as follows:

1. Feature extraction,

2. Linear Random Transformation,

3. Quantization [14],[15].

2.4.1 Generating the biohash

In traditional biohashing techniques, feature extraction is done by linear transformation of the data. Principle component analysis (PCA) [18] is the most frequent tool here. We generate a matrix $\boldsymbol{R}$ whose elements are independent and identically distributed (i.i.d) and generated from a Gaussian distribution with zero mean and unit variance, by using a Random Number Generator (RNG) with a seed derived from the user's secret key. This matrix $\boldsymbol{R}$ acts as random transformation.

In short we can summarize as:

$$\boldsymbol{B}_i = \text{sign}(\boldsymbol{R}_i(\boldsymbol{P}(\boldsymbol{x}_i - \boldsymbol{\mu}))) \tag{2.1}$$

where $\boldsymbol{x}_i$ is the vectorized image for the user i, $\boldsymbol{P}$ is the pca matrix, $\boldsymbol{R}_i$, as mentioned before, is the random matrix for the user i or her secret key, and $\boldsymbol{B}_i$ is the hash vector generated for the user i.

Quantization or binarization can simply be done by applying *sign* function to the result of the previous step.

2.4.2 Authentication using biohash

In authentication again all the operators for hashing are done on the unidentified image $\boldsymbol{x}_{new}$ to generate $\boldsymbol{B}_{new}$, then Hamming distances $\boldsymbol{B}_i$ and $\boldsymbol{B}_{new}$ are compared to a threshold η. If these two vectors are close enough, the user providing $\boldsymbol{x}_{new}$ will be verified as the claimed user i.

2.5 Drawbacks of traditional biohash and cures

There are several problems for traditional biohashing procedure described in section 2.4.1.

- The system in the case of a compromised key is very fragile. The main reason is that the system is computationally very simple and all the operators till quantization are linear.
- The feature extractor which is applied in this algorithm is linear. Features extracted linearly from the raw data do not necessarily provide us invariancy under changes within a class, or generating codes with large Hamming distances between different users.

2.6 Our proposed method

To improve the biohashing system which has the discussed troubles, we introduce a deep learning based method. Several layers of nonlinearity before applying random projection, make the system harder to be attacked even with a compromised hidden key, since this system potentially has much more parameters than the model in (2.1). Figure 2.1 shows a schematic of our model.

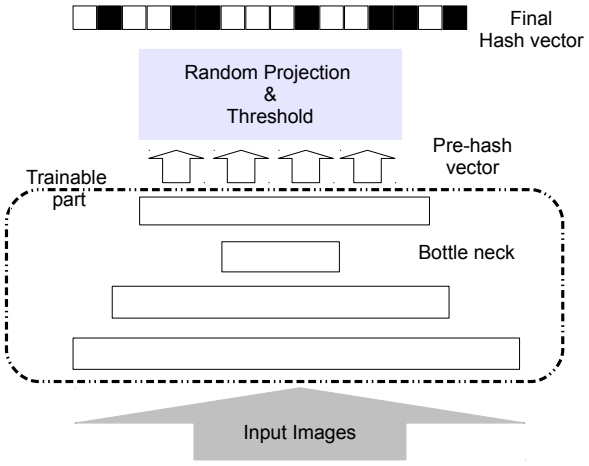

FIGURE 2.1: Deep learning based biohash model.

Deep neural networks are very rich models and by enough training they can approximate very complex functions [19]. Furthermore, first layers of DNN can be seen as feature extractors. By training a *good* structured neural network, instead of using "off-the-shelf" feature extractors that may not suite our task, we design (train) feature extractors optimized for the special task. More specifically, our suggested system consists of two identical neural networks, sharing weights. During the training both sub-networks will

be penalized if the distance between two hash vectors generated from images of the same person have large distance or hash vectors from different subjects are close.

Neural networks is a general purpose pattern recognition model. Given two person's images, they are sent to a Siamese neural network (SNN). Similarity of an image in the gallery with a probe image is computed based on SNN outputs. The structure of the SNN is shown in Figure 4.1, which is composed by two identical neural networks. During training, the two sub-networks are connected via a cost computing unit. Since in action (test phase) we use only one of the networks to compute the features, in Siamese neural networks we should have a constraint that the two sub-networks share the same weights and biases.

Chapter 3

Neural Networks and Deep Learning

Neural network is a general purpose computational model which is inspired from the human brain. The basic paradigm in neural networks is to represent a function as a nonlinear function of weighed mean of the inputs. Training is finding weights such that the entire final function generates the target output from the input, the process of finding the optimal weights is called "training".

The diagram shown in Figure 3.1 illustrates a 3 layer feedforward neural network; it consists of an output layer and two hidden layers. There are n neurons in the input layer. The output is calculated as follows: $\boldsymbol{a}^{(L)} = f(\boldsymbol{W}^{(L)}(...(f(\boldsymbol{W}^{(0)}\boldsymbol{x}_{in} + \boldsymbol{b}^{(0)})...) + \boldsymbol{b}^{(L)})$, $\boldsymbol{b}^{(i)}$ is referred to as the bias. We can set a special input for $\boldsymbol{b}$ equal to 1 and deal with it like other weights. We will call this multi-layer function as $\boldsymbol{O} = G_W(\boldsymbol{x}_{in})$ where $\boldsymbol{O} = \boldsymbol{a}^{(L)}$ is the output of the neural network and $\boldsymbol{W}$ includes all the weight parameters from all layers in it.

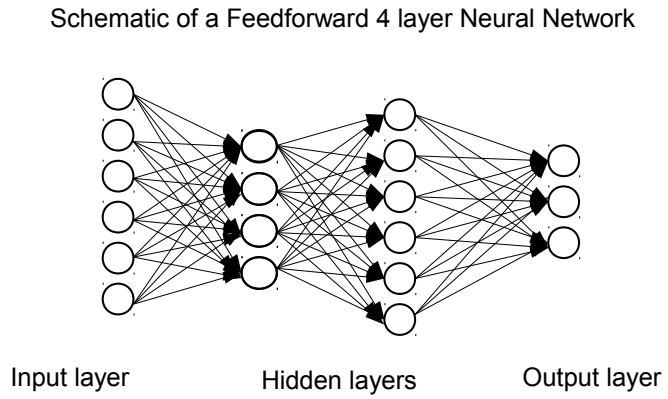

FIGURE 3.1: A schematic sample of a 3-layer neural network.

3.1 Training the neural network

There are different classes of learning methods. All learning algorithms aim to adjust the weights of the connections between units, obtain a proper output for each input, according to different update rules. Hebbian learning rule as one of the basic neural network training states that, if two units j and k are active simultaneously, their interconnection must be strengthened. If j receives input from k, [20] The following learning rule uses the difference between the actual and desired activation for adjusting the weights:

$$\Delta w_{jk} = \gamma y_j (t_k - y_k) \tag{3.1}$$

in which t_k is the desired (target) activation provided by a teacher, where γ is the learning rate parameter. This is often called the Widrow-Hoff rule or the delta rule [21]. There are different algorithms for training neural networks, the main difference in these algorithm is how they address the problem of optimization of a cost function. Gradient based optimization are the most common choice for training neural networks. In the next section we will discuss backpropagation.

3.1.1 Backpropagation

Backpropagation is a three stage algorithm:

1. **Forward Propagation**

- Passing the training data through the neural network to calculate the activations and output of the net.

2. **Backward Propagation**

   - Backward propagation of the output error (actually the gradient of the objective function) through the neural network. The gradient of the training objective which usually includes a comparison of the output of the network with a target value leads to generation of the gradient (named as the error or the δ in neural network literature) with respect to each weight parameter in the network using the chain rule.

3. **Weight Update**

   - Update the weights by subtracting a ratio γ of the gradient from the weight.

The goal of the backpropagation algorithm is to compute the partial derivative $\frac{\partial C}{\partial \boldsymbol{W}}$ where C represents the cost function (or the loss function) and $\boldsymbol{W}$ stands for any parameter in the network. An example for the cost function C can be given as the least squares cost which is given as $C = \frac{1}{2}\sum_{i=1}^{N} ||\boldsymbol{o}_i - \boldsymbol{y}_i||^2$ where $\boldsymbol{o}_i = \boldsymbol{a}_i^{(L)} = G_W(\boldsymbol{x}_i)$ is the output vector for the neural network given input $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ is the target vector for the $i'th$ training data point during training. In the below discussion we can think about the cost as the cost of a single training instance, that is $C = \frac{1}{2}||\boldsymbol{o} - \boldsymbol{y}||^2$ for simplification.

For back propagation in general we calculate the following quantities:

$$\Delta_j^{(l)} = \frac{\partial C}{\partial a_j^{(l)}} f(z_j^{(l)}). \tag{3.2}$$

Here $a_j^{(l)}$ is the activation of $j'th$ neuron of $l'th$ layer, $z_j^{(l)}$ is the input of this neuron and $\Delta_j^{(l)}$ is defined as error of neuron j in layer l.

$$\Delta^{(l)} = ((\boldsymbol{W}^{(l+1)})^T \Delta^{(l+1)}) \otimes f(\boldsymbol{x}^{(l)}). \tag{3.3}$$

Here $\otimes$ stands for Hadamard (component wise) multiplication.

The equation for the rate of change of the cost with respect to any weight in the network is:

$$\frac{\partial C}{\partial W_{jk}^{(l)}} = a_k^{(l-1)}\Delta_j^{(l)}, \frac{\partial C}{\partial b_j^{(l)}} = \Delta_j^{(l)}. \tag{3.4}$$

Backpropagation algorithm is:

1. **Input**: Set the corresponding activation $a^{(0)} = x$ (the input) for the input layer.
2. **Feedforward**: For each $l = 1, 2, \ldots, L$ compute $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$ and $\boldsymbol{a}^{(l)} = f(\boldsymbol{z}^{(l)})$.
3. **Output error** $\Delta^{(L)}$: Compute the vector $\boldsymbol{\Delta}^{(L)} = \boldsymbol{\nabla}_{a^{(L)}} C \otimes f(\boldsymbol{z}^{(L)})$.
4. **Backpropagate the error**: For each $l = L-1, L-2, \ldots, 2$ compute $\boldsymbol{\Delta}^{(l)} = (((\boldsymbol{W}^{(l+1)})^T\boldsymbol{\Delta}^{(l+1)}) \otimes f(z^{(l)})$.
5. **Output**: The gradient of the cost function is given by $\frac{\partial C}{\partial W_{jk}^{(l)}} = a_k^{(l-1)}\Delta_j^{(l)}$ and $\frac{\partial C}{\partial b_j^{(l)}} = \Delta_j^{(l)}$ [22].

3.2 Optimization algorithms for neural network learning

3.2.1 Gradient descent

We consider a cost or a loss function $C(\boldsymbol{\theta}, \boldsymbol{W})$, where $\theta = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^N$ represents the set of all inputs and outputs of the system that measures the cost of predicting. We seek the function G_W ,parametrized over vector (matrix) W, that minimizes the loss $C(\boldsymbol{\theta}, \boldsymbol{W})$ averaged on the examples. A common way to minimize an the cost function is using the gradient descent algorithm. In each iteration the weights $\boldsymbol{W}$ are updated based on the gradient of $C(\boldsymbol{\theta}, \boldsymbol{W})$,

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \gamma\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\nabla}_W C(\boldsymbol{\theta}_i, \boldsymbol{W}_t) \tag{3.5}$$

where γ is called learning rate and could be a constant or function of the parameters e.g. decreasing function of number of iterations (γ_t). Under sufficient condition,like good initialization and when the learning rate γ is sufficiently small, this algorithm convergence linearly i.e. residual error asymptotically decreases exponentially. [23].

3.2.2 Stochastic gradient descent

Stochastic gradient descent algorithm is a simplification to reduce the computational expenses of gradient descent. Here instead of computing the gradient of $C(\boldsymbol{\theta}, \boldsymbol{W})$ exactly, in each iteration we estimate this gradient by a single randomly picked example:

$$\boldsymbol{W}_{t_{+1}} = \boldsymbol{W}_t - \gamma_t \boldsymbol{\nabla}_W C(\boldsymbol{\theta_t}, \boldsymbol{W}_t) \tag{3.6}$$

In this case $\{\boldsymbol{W}_t, t = 1, \ldots\}$ is a stochastic process depends on the examples randomly picked at each iteration. We hope that Equation (3.6) behaves like its expectation Equation (3.5) despite the simplification of taking only one sample each time. Beside the speed of stochastic gradient algorithm, It can easily be performed on-line which is a big advantage for big data. γ_t has to decay in time as $\frac{1}{t}$ so that stochastic gradient descent converges in probability to the true parameter values.

3.2.3 Batch gradient descent

Another alternative for full gradient method is to approximate the gradient not only with a random sample of training data, like in stochastic gradient method. We can take a small batch of data and update the weights based on average of gradients over that small batch.

$$\boldsymbol{W}_{t_{+1}} = \boldsymbol{W}_t - \gamma \frac{1}{B} \sum_{i=1}^{B} \boldsymbol{\nabla}_W C(\boldsymbol{\theta}_i, \boldsymbol{W}_t). \tag{3.7}$$

Here B is the batch size. This method is called the mini-batch stochastic gradient method sometimes. The regular gradient descent (or steepest descent) method described in 3.2.1 is sometimes called full-batch gradient descent since it corresponds to using the whole data in the batch gradient descent. We discuss about B on performance of the system in Section 5.4.3.

3.3 Architecture of the neural net

The choice of the number of layers (deepness or shallowness of the network) for the architecture is dependent on many factors. For every specific task and data type, the number of layers and the number of nodes in each layer that would achieve good generalization is different [24][25]. Shallow architectures (structures with few hidden layer) have smaller number of parameters to be trained hence there is no need to access large dataset of training samples, on the other side such structures may not be capable to completely capture the specific information and extract proper features from the data.

In order to have high separability among data classes, the architecture should have enough parameters to handle the complexity or nonlinearity of the function to capture latent structure in the data [24].

In [26], it is stated that "From a mathematical perspective, a large number of functions can be well approximated by shallow architectures, provided they have sufficient number of elements in their hidden layers. However, the number of elements needed to estimate a highly nonlinear function can grow exponentially." [26],Neural nets with many hidden layers in some new text are referenced as "deep neural nets". These kind of networks attract a huge attention for different applications. Neural networks was a hot topic in 70s and 80s in artificial intelligence and machine learning but training deep architectures using backpropagation was not successful. The reason of the failure is called "the vanishing gradient problem" [27][28]. To overcome this problem, several methods were proposed. One is multi-level hierarchy pre-trained one level at a time through unsupervised learning and then fine-tuned through backpropagation [29]. In this method each level learns a compressed representation of the observations that is fed to the next level.

The idea is to divide the training of successive layers of a deep network in the form of small sub-networks and use unsupervised learning to minimize input reconstruction error. This will be discussed in detail later. This technique successfully eliminates the shortcomings of the gradient based learning by averting the local minima [30].

3.3.1 Pretraining and Autoencoders

Autoencoders are models that are trained in order to represent the data using a series of nonlinear transformations [24],[26]. Basically an autoencoder is a neural network with single or multiple hidden layers which is trained in an unsupervised manner, the aim of this network is to find a representation of the data. The objective of learning is to minimize the data reconstruction error. To avoid trivial cases in training, mostly a constraint like sparsity or applying smaller number of neurons than the number of input units are used. [24],[26],[31], [32] and [33]. According to the definition in [33], encoder is referred to a deterministic mapping G_W that transforms an input vector $\boldsymbol{x}$ into hidden representation $\boldsymbol{y}$, where $\boldsymbol{W}$ is the weight matrix . On the other hand a decoder maps back the hidden representation $\boldsymbol{y}$ to the reconstructed input $\boldsymbol{x}$ via $G_{W'}$. The whole process of autoencoding is to compare this reconstructed input to the original, apparently it needs an error criterion, and try to minimize this error to make the reconstructed value as close as possible to the original.

Each autoencoder is a two-layered neural network with a single nonlinear hidden layer. for input $\boldsymbol{x}$, the hidden layer representation is given as

$$h_j(x) = f(b_j^{(1)} + \sum_{i=1}^{|x|} W_{ij}^{(1)} x_i), \tag{3.8}$$

where $W_{ij}^{(1)}$ are the connection weights between input x_i and h_j, while $\boldsymbol{b}^{(1)}$ is the hidden layer bias vector. $f(.)$ is the neural network activation function. The choice of the activation function varies depending upon the statistical properties of the data [34]. The input reconstruction at the output layer is

$$\hat{x}_i = f'(b_i^{(2)} + \sum_{i=1}^{|h|} W_{ij}^{(2)} h_j(x)). \tag{3.9}$$

In (3.8) and (3.9) $W_{ij}^{(k)}$, $k = 1, 2$ are the weights corresponding to the hidden layer ($k = 1$) and output layer ($k = 2$). $f'(.)$ is the neural network activation function for the reverse path. The objective or the cost function for training autoencoder networks can be the mean square error between actual and reconstructed input,

$$C(x, \hat{x}) = ||\boldsymbol{x} - \hat{\boldsymbol{x}}||_2^2. \tag{3.10}$$

The same learning principle is applied for higher layers. The parameter optimization usually involves gradient descent using backpropagation [24][35].

3.4 Deep neural network training

The training of **Deep neural networks** (DNN) involves two-stage learning of the network parameters. The first stage, i.e., pre-training, is unsupervised learning using unlabeled data and provides the initialization for the DNN. The second stage, i.e. fine-tuning stage involves supervised learning using the backpropagation algorithm.

Chapter 4

Siamese Networks

4.1 Siamese architecture, why, what, how

Our proposed method is to train a model in an unsupervised manner which generates codes for each sample and try to minimize the distances between the codes for similar examples and maximize it for distinct examples.

In applications like face recognition and face verification, the number of categories can be hundreds or thousands, with only a few examples per category. A common approach to these kind of problems is distance-based methods, which consists of computing a similarity metric between the pattern to be classified or verified and a library of stored prototypes. To apply discriminative learning techniques to this kind of application, in [36] a model is proposed which can extract information about the problem from the available data, without requiring specific information about the categories. The solution presented in [36] and then [37] and [38] is to learn a similarity metric from data. This similarity metric can later be used to compare or match new samples from previously-unseen categories (e.g. faces from people not seen during training).

Siamese networks (or metric learning) can be applied to multi-class classification problems as well may be since it will learn good mappings after which you can directly use nearest neighbor methods for classification. However this is not our goal in this thesis.

The main idea is to find a function which maps input patterns into a target space such that a simple distance in the target space (Euclidean distance or in our particular

example Hamming distance or angular distance) approximates the similarities in the input space. More precisely, given a family of functions $G_W(\boldsymbol{x})$ parameterized by $\boldsymbol{W}$, we are looking for a value of the parameter $\boldsymbol{W}$ such that the similarity metric (distance) $D_W(\boldsymbol{x}_1, \boldsymbol{x}_2) = d_S(G_W(\boldsymbol{x}_1), G_W(\boldsymbol{x}_2))$ is small if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ belong to the same class, and large if they belong to different classes. The system is trained on pairs of training samples taken from a training set.

Because the same function G with the same parameter $\boldsymbol{W}$ is used to process both inputs, the similarity metric is symmetric. This is called a *Siamese architecture* as in [39] and [36] or *hybrid architecture*, as in [37].

In our face verification system, like in [36] and [38], we first train the model to produce output vectors that are close in angular distance sense for pairs of images from the same person, and far away for pairs of images from different persons. We expect that the model would map face images of new persons that were not seen during training to vectors with proper distances. An important aspect of the proposed method is that there is infinite possibilities for the function G_W. In particular, we can use architectures which are designed to extract features that are robust to different distortions and are invariant to a class of transformations of the input. The resulting similarity metric will be robust to some differences of the pose between the pairs of images [36].

4.2 The framework for Siamese architecture

Energy-based models (EBM) assign an unnormalized energy to each configuration of parameters of a system. Prediction in such systems is performed by searching for configurations of the variables that minimize the energy [40]. A trainable similarity metric can be seen as associating an energy $E_W(s, \boldsymbol{x}_1, \boldsymbol{x}_2)$ where the binary variable s indicates whether $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ come from the same class. to labeled pairs of input patterns.

Learning is done by finding the $\boldsymbol{W}$ that minimizes a suitably designed *loss function*, evaluated over a training set [40].

The first loss function that may come to one's mind is simply minimizing $E_W(s, \boldsymbol{x}_1, \boldsymbol{x}_2)$ summed over a set of pairs of inputs from the same category, but this generally leads to a catastrophic collapse. Using such a loss function which only uses pairs from the

same category will map the whole space to a very small portion of the code space (the range of the mapping), or by setting $G_W(.)$ to a constant function a minimum for this inefficient loss function can be achieved.

A remedy for this problem is to use a **contrastive term** loss function.

4.2.1 Face verification

The task of face verification, is to accept or reject the claimed identity of a subject in an image, [36] [41]. Performance is assessed using two measures: false accepts ratio (FAR) and the false rejects ratio (FRR). A good system should minimize both measures simultaneously and have small equal error rate (EER).

In this thesis, we apply the idea of metric learning using Siamese networks to biometric hashing. The basic idea of this approach is to embed raw biometric data in a low dimensional space so that the distance between the embeddings is small if the data belongs to the same person and large otherwise [36]. This idea has been used for biometric verification before [36], [38], but has not been used for biometric hashing. Our contribution is to apply it for biohashing problems. In addition, we specifically introduce a novel loss function for Siamese networks for learning better biometric hashes.

Learning the similarity metric is realized by training a network which consists of two identical networks that share the same set of weights - a Siamese Architecture [39],[38] (see Figure 4.1).

4.3 Energy function and training the network

4.3.1 Learning the Mapping

In a famous work called “DrLim” [42] The problem of finding a function that maps input data to (binary) outputs, given neighborhood relationships between samples in input space and preserve the mutual distances between them, is formalized as follows: We are interested in the following properties:

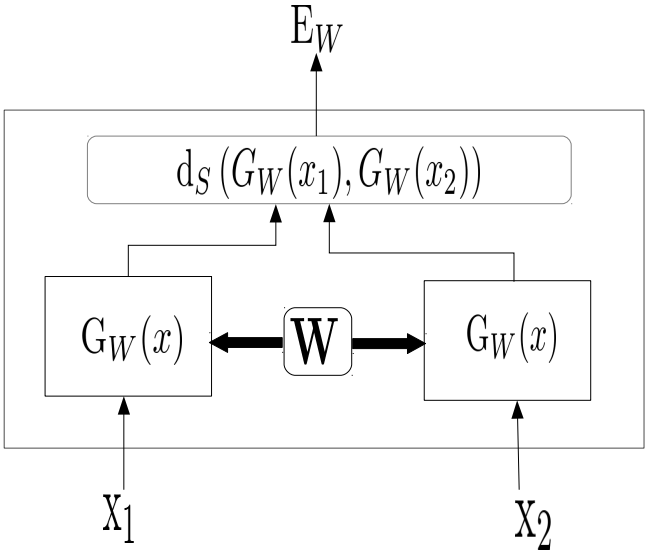

FIGURE 4.1: Schematic of the Siamese network.

1. Desired distance measures in the output space (such as Hamming distance or angular or cosine distance in this study) should approximate the neighborhood relationships in the input space.

2. The mapping should not be constrained to implementing simple distance measures in the input space and should be able to learn invariances to complex transformations.

3. It should work well even for samples whose neighborhood relationships are unknown.

We call such a mapping (function) G_W which means function G with parameter $\boldsymbol{W}$.

4.3.2 The contrastive loss function

Consider the set $\boldsymbol{X}$ of (high dimensional) training vectors. A meaningful mapping from high to low dimensional space maps similar input vectors to nearby points on the output manifold and dissimilar vectors to distant points. There are some loss functions whose minimization can produce such a function [40]. The loss function here runs over pairs of samples. Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in X$ be a pair of input vectors shown to the system. Let s be a binary label assigned to this pair: $s = 1$ for similar pairs and $s = 0$ for dissimilar

pairs. Define the parameterized distance function to be learned D_W between $\boldsymbol{x}_1, \boldsymbol{x}_2$ as the (Euclidean) distance between the outputs of the mapping (G_W). That is,

$$D_W(\boldsymbol{x}_1, \boldsymbol{x}_2) = ||G_W(\boldsymbol{x}_1) - G_W(\boldsymbol{x}_2)||_2. \tag{4.1}$$

Then the loss function (which is similar to the cost function of the Chapter 3) in its most general form is:

$$\begin{aligned} \mathcal{L}(\boldsymbol{W}) &= \sum_{i=1}^{P} L(\boldsymbol{W}, (s, \boldsymbol{x}_1, \boldsymbol{x}_2)^i), & (4.2) \\ L(\boldsymbol{W}, (s, \boldsymbol{x}_1, \boldsymbol{x}_2)) &= s \times L_S(D_W) + (1 - s) \times L_D(D_W), & (4.3) \end{aligned}$$

where D_W stands for $D_W(\boldsymbol{x}_1, \boldsymbol{x}_2)$, $(s, \boldsymbol{x}_1, \boldsymbol{x}_2)^i$ is the ith labeled sample pair, L_S is the partial loss function for a pair of similar points, L_D the partial loss function for a pair of dissimilar points, and P the number of training pairs (which may be as large as the square of the number of samples). L_S and L_D are defined such that minimizing $\mathcal{L}$ with respect to W would result in low values of D_W for similar pairs and high values of D_W for dissimilar pairs.

For our problem the loss function for a single pair of inputs and their label is the same as their energy function $E(s, \boldsymbol{x}_1, \boldsymbol{x}_2)$. So the overall loss function ends up being equal to the sum of energies over training data. For other learning problems such as multi-class classification, a loss function can be different from the energy function. For example a reasonable multi-class loss function would encourage the correct class' energy to be lower whereas wrong class' energies to be higher for a single training data [40].

There are many possible ways to define the loss function [40]. In different experiments conducted for this project the best results were obtained by the following loss function which is also used in [42],[38]and [43].

$$L(W, s, \boldsymbol{x}_1, \boldsymbol{x}_2) = s \times \frac{1}{2}(D_W)^2 + (1 - s) \times \frac{1}{2} \max\{0, \alpha - D_W\}^2, \tag{4.4}$$

where $\alpha > 0$ is a margin defined around mapped data. Dissimilar pairs contribute to the loss function only if their distance is within this radius. The contrastive term involving

dissimilar pairs, L_D, is crucial.

4.3.3 Method and gradients in detail

In this section we will describe details for learning an invariant mapping that can exploit the similarity of pairs of training examples obtained through class labels.

We construct sets based on our data collection efforts. Each set consists of two samples from training set and the label indicating if they are similar or not, $(\boldsymbol{x}_i, \boldsymbol{x}_j, s)$ as mentioned before.

4.3.3.1 Energy-based method

This approach is online (each training pair updates the system independently) and suites the large training set or large models. Moreover, we can control the balance between similar and dissimilar examples when the dataset is dominated by one or the other. DrLIM [42] is an energy-based method that is trained by stochastic gradient descent. It can be used with arbitrary nonlinear G_W. Here we define $O_i = G_W(\boldsymbol{x}_i)$.

$$\mathcal{L}(W, s, x_1, x_2) = s \times \frac{1}{2}(D_W)^2 + (1 - s) \times \frac{1}{2} \max\{0, \alpha - D_W\}^2. \tag{4.5}$$

To keep the embedding from collapsing, $\mathcal{L}$ has a contrastive component whereby dissimilar points are pulled apart through the dissimilarity loss:

$$\frac{1}{2} \max\{0, \alpha - D_W\}^2 \tag{4.6}$$

where $\alpha > 0$ is a margin which ensures that dissimilar points contribute to the loss only if they lie close-by in the embedded space, and α is fixed to a certain value [38].

The model is trained by stochastic gradient descent. The gradient of the outputs with respect to the weight matrix $\boldsymbol{W}$ given a single data point $(\boldsymbol{x}_1, \boldsymbol{x}_2, s)$ is given by:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{O}_1} \frac{\partial \boldsymbol{O}_1}{\partial \boldsymbol{W}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{O}_2} \frac{\partial \boldsymbol{O}_2}{\partial \boldsymbol{W}}, \tag{4.7}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}_1} = \begin{cases} s(\boldsymbol{O}_1 - \boldsymbol{O}_2) & \text{if } s > 0 \\ (d-\alpha)\frac{\boldsymbol{O}_1 - \boldsymbol{O}_2}{d} & \text{if } s = 0, d < \alpha \\ 0 & \text{if } s = 0, d_{ij} > \alpha, \end{cases} \tag{4.8}$$

where s indicates the similarity between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and $d = ||\boldsymbol{O}_1 - \boldsymbol{O}_2||_2$ and $\boldsymbol{O}_1 = G_W(\boldsymbol{x}_1)$. The gradient with rspect to $\boldsymbol{O}_2$ can be obtained as the negative of the gradient with respect to $\boldsymbol{O}_1$ since the loss function is symmetric.

We should notice that the equations before, are valid if only we are interested in preserving the Euclidean distance between the mapped data. The formulas for different metrics will be provided in the next section.

4.3.4 Angular distance relations

In application of our interest, obtaining similarity preserving binary hash codes; one popular way for binarization of calculated feature vectors is to threshold them, i.e. to apply sign function on all the coordinates of the feature vector.

Before going into details of the model in our specific application, we mention the following notations, definitions and theorems.

A useful distance in signal space is the angular distance between two vectors. We define angular or cosine distance between two vector $\boldsymbol{u}$ and $\boldsymbol{v}$ as follows:

$$d_S(\boldsymbol{u}, \boldsymbol{v}) := \frac{1}{\pi} \arccos \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{(||\boldsymbol{u}||||\boldsymbol{v}||)} \tag{4.9}$$

From now on, by $\mathcal{B}^K$ we refer to the binary space $\{-1, 1\}^K$.

Let's define $A : \Re^N \longrightarrow \mathcal{B}^K$ defined as follows:

$$\boldsymbol{y} = A(\boldsymbol{x}) := \text{sign}(\boldsymbol{\Phi}\boldsymbol{x}), \tag{4.10}$$

where the sign function is defined as:

$$\text{sign}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0. \end{cases} \tag{4.11}$$

The operator $A(.)$ is a mapping from $\Re^N$ to K dimensional binary space, in this definition $\boldsymbol{\Phi}$ is a matrix satisfying Restricted Isometry Principle (RIP) condition. We aim to build binary codes which are far from each other. The Hamming distance is the natural distance for counting the number of unequal bits between two hash vectors. Specifically, for $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{B}^K$ we define the normalized Hamming distance as

$$d_H(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{K} \sum_{i=1}^{K} a_i \oplus b_i, \tag{4.12}$$

where $\oplus$ is the XOR operation such that $a \oplus b$ equals 0 if $a = b$ and 1 otherwise. The distance is normalized such that $d_H \in [0, 1]$. In [44] and [45] it is shown that :

Theorem 4.1. *Let $\boldsymbol{\Phi}$ be a matrix generated as $\boldsymbol{\Phi} \sim \mathcal{N}^{K \times N}(0, 1)$, and let the mapping $A : \boldsymbol{x} \in \Re^N \longrightarrow \mathcal{B}^K$ be defined as in (4.10). Fix $0 < \epsilon < 1$. if For any $\boldsymbol{u}, \boldsymbol{v}$, we have*

$$\mathcal{P}(|d_H(A(\boldsymbol{u}), A(\boldsymbol{v})) - d_S(\boldsymbol{u}, \boldsymbol{v})| \leq \epsilon) \geq 1 - 2^{-2\epsilon^2 K}, \tag{4.13}$$

where the probability is with respect to the generation of Φ.

In words, Theorem 4.1 implies that the Hamming distance between two binary measurement vectors $A(\boldsymbol{u}), A(\boldsymbol{v})$ tends to the angle between the signals $\boldsymbol{u}$ and $\boldsymbol{v}$ as the number of output hash dimensions K increases.

4.3.4.1 Loss function using the angular distance

As discussed before, we aim to map biometric data to binary code such that samples from different classes have large mutual Hamming distance.

At the first look we can directly penalize the mapping with respect to Hamming distances between instances, but the problematic point is that Hamming distance is not differentiable.

The solution we propose is to divide the mapping into two steps:

1. Feature extraction
2. Binarization.

In **Feature extraction** step we try to find a (sub-)optimal mapping which maximizes angular distances between dissimilar samples and minimizes it between similar samples, then applying theorem 4.10, with a proper $\mathbf{\Phi}$ matrix we can obtain *good* binary codes.

To obtain the new energy function and the loss function, we just replace the Euclidean distance used in the earlier derivation with the angular distance and make no other change.

4.3.4.2 Derivatives with the respect to the angular distance

The partial derivative of angular distance with respect to one of the vectors is as follows:

$$\frac{\partial}{\partial \boldsymbol{u}} d_S(\boldsymbol{u}, \boldsymbol{v}) = (\frac{1}{2} - \frac{1}{||\boldsymbol{u}||}) \frac{\boldsymbol{v}}{||\boldsymbol{v}||}. \tag{4.14}$$

By substituting above equation into 4.7 we can obtain derivatives similar to ones in 4.8 with respect to angular distance.

The resulting derivatives are used in order to back propagate the error which is discussed in Chapter 3.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}_1} = \begin{cases} s(\frac{1}{2} - \frac{1}{||\boldsymbol{O}_1||}) \frac{\boldsymbol{O}_2}{||\boldsymbol{O}_2||} & \text{if } s > 0 \\ (d - \alpha) \frac{(\frac{1}{2} - \frac{1}{||\boldsymbol{O}_1||}) \frac{\boldsymbol{O}_2}{||\boldsymbol{O}_2||}}{d} & \text{if } s = 0, d < \alpha \\ 0 & \text{if } s = 0, d > \alpha. \end{cases} \tag{4.15}$$

Again s is the indicator for the similarity between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and $d = d_S(\boldsymbol{O}_1, \boldsymbol{O}_2)$ is the angular distance between the mapped sample and $\boldsymbol{O}_1 = G_W(\boldsymbol{x}_1)$.

Chapter 5

Experiments and Results

In this chapter, we first present our experimental setup including database preparation, next we present the experimental results followed by our discussions about certain aspects of the experimentation.

5.1 Datasets

The first dataset we used for training and testing was a relatively small dataset of 400 images from the AT&T Database of Faces [46]. The dataset contains 10 images each of 40 subjects, with variations in lighting, facial expression, accessories, and head position. Each image is 112x92 pixels, gray scale, and closely cropped to include the face only. To avoid computational difficulties we did reduce the resolution of the images to 56x46 using 4x4 subsampling. See Figure 5.1 for example face images from this dataset.

FERET dataset [47]- distributed by the National Institute of Standards and Technology, is a well known database for recognition tasks. Here we used a subset of this set containing 4 images in different poses of 60 different persons. Lighting and facial expression varies for different images and different subject. Our subset consists of 1122 images, that is, 6 images each of 187 subjects. The only preprocessing was cropping and subsampling to 56x46 pixel images.

CMU dataset is another well known database for face recognition tasks. Here we used a subset of this set containing 4 images in different poses of 68 different persons. We

cropped images into semi-elliptic frame around the center of the face and then down-sampled the images to the size 75×65 pixels.

M2VTS [48] dataset is another famous dataset for testing face recognition tasks, here we used a subset of the dataset with images of 37 people in 10 different poses where images of 27 subject used for training, 5 unseen subjects for validation and 5 for testing.

We train our model on a relatively large portion of the dataset, leave a small portion of data as validation data which is used for cross-validation to obtain parameter settings and stopping criteria and use the rest for testing.

We first shuffle the identities of the samples then make training and testing sets which consist of all images of selected identities for each set. Since train and test phase both need pairs of images, the images should be paired. If in a set (train, validation or test) C identities (people) are chosen where each one has K different samples, we will have $C \times K(K-1)$ genuine pairs and $K \times ((C-1) \times K)$ impostor pairs. In order to keep balance in training, we pick randomly a portion of impostor pairs for training.

5.2 Training protocol

Our framework as described in Chapter 4 compares two identical networks and one cost function, the input to the system is a pair of images and a similarity indicator. The images are passed through sub-networks and form two pre-hash vectors. These two vector outputs are passed to the energy (or loss) calculation routine which calculates the scalar called loss function by the Equation (4.5).

The loss function is calculated using the labels in the training data. By back-propagation the gradient of the loss function with respect to the parameters for both subnets is computed. Then each parameter is updated with the SGD method using the sum of gradients obtained at both sub-nets since the parameters of the sub-nets are tied to each other.

5.2.1 Data partitioning

In order to have a fair train and test set. we should generate a test set consisting of images of the subjects that are not seen by the learning algorithm during training, we

split the datasets into three disjoint sets, namely SET1, SET2 and SET3. Each image in each of these sets was paired up with a fixed number of other image in that set to have as many as possible genuine pairs and a reasonable number of impostor pairs. For the AT&T data, SET1 consisted of the 250 images of first 25 subjects and SET2 consisted of 50 images of last 5 subjects for validation and SET3 consisted of 50 images of last 5 subjects as test set. Training is done using only the image pairs generated from SET1 and hyper-parameters for training obtained from pairs in SET2.

For the Feret data, SET1 contains images from 50 subjects, 5 subjects for validation and again 5 person for testing. For M2VTS [48] we used a subset of the dataset with images of 37 people in 10 different poses where images of 27 subject used for training, 5 unseen subjects for validation and 5 for testing. The last dataset we used in this project is CMU [49] which has 4 images of 68 different subjects, we split them into 55, 5 and 8 for training, validation and testing respectively.

Except AT&T we down-sampled to the size 75×65 pixel (down-sampling ratio is about 2×2) then cropped all the images by an elliptic frame.

5.2.2 Network Architecture

Our sub-nets have one input layer, 4 hidden layers which have 1000, 400, 100 and 256 units in each layer respectively. The output is a real vector of length 256 which is called a pre-hash. The next step is generating hash vector, this stage can be done by multiplying the pre-hash vector with a pseudo-random matrix whose entries are generated in a pseudo-random fashion initialized by a seed from the user and satisfy a standard normal distribution.Then, we take the sign of the transformed vector to convert it into a binary hash vector. This way, we obtain the biometric hash for the given biometric data.

We used linear units for the first two layers and sigmoid transfer function for upper layers. This choice is made empirically, changing the first layer to the nonlinear decreases the performance in all different implementations. Other hyper-parameters used in the algorithm are determined empirically too. The best performance with respect to α was obtained for $\alpha = 1.5$. Another important effecting hyper-parameter is the batch size used in the optimization algorithm. Training with batch containing 20-50 training samples lead to better results. We initialize our networks with i.i.d. zero mean randomly

generated matrices for all the experiments. We compared various initializations in 5.4.4, the results there suggest using random initialization superiority to the other methods for this task.

5.3 Results

5.3.1 AT&T dataset

Figure 5.1 shows some samples from AT&T dataset.

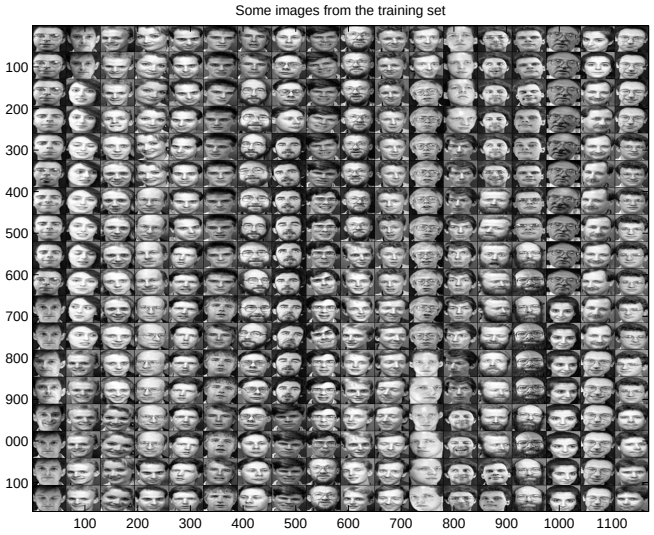

FIGURE 5.1: Some samples of AT&T dataset.

An illustration of PCA coefficients from the AT&T dataset is in Figure 5.2.

As we described before, better verification systems has DET curves, closer to the bottom-left of the plane. As we expect DET curve for training set for all methods have better performance than the same method's performance on the test set (unseen subjects). Blue curve, nearest to the origin in all the experiments is performance of our method on the training set, green curve shows the results on the test set, for the competitor method (PCA based method), training and testing curves in red and purple respectively are in farther distance from the origin compare to result of our method on the same data.

Figure 5.3 shows changing of EER for validation set and energy (mean of the energy for the batches.) in each epoch.

As we expect energy decreases through training but EER does not change dramatically, we will discuss this topic in Section 5.4.1.

Figures 5.4 and 5.5 show angular distances between pre-hash vectors for each pairs of training and testing set respectively. Here we arrange data points such that all members

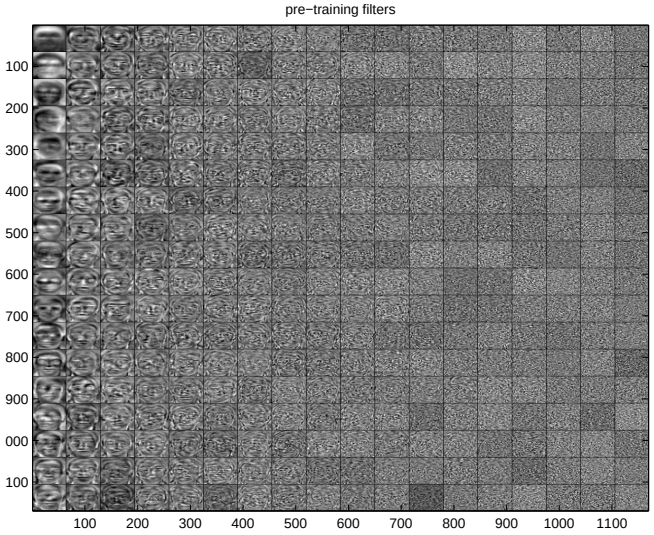

FIGURE 5.2: Some samples of PCA eigenvectors for the AT&T dataset. These filter are used for pre-training in some experiments.

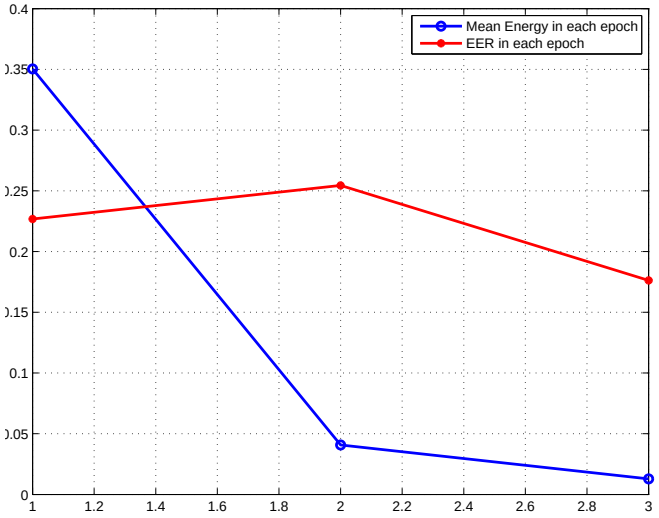

FIGURE 5.3: Changing of loss function and EER on validation data of the AT&T database.

of each classes stick together and all class have equal number of members. The DET curve show the behavior of the system with respect to pre-hash vectors and angular distance is shown in Figure 5.6.

DET curves for hash vectors of length 128, 256, 512, 1024 and 2048 are provided in Figures: 5.7 ,5.8,5.9,5.10 and 5.11 respectively.

As we use more bits for the biohash, the performance of the deep network gets better as compared to the PCA based one. As we expect he longer hash code will behave similar

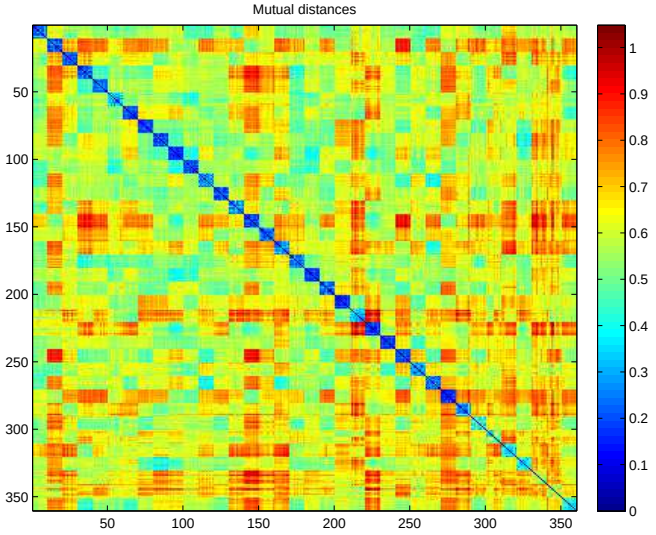

FIGURE 5.4: Mutual distances between AT&T pre-hash codes from train subset.

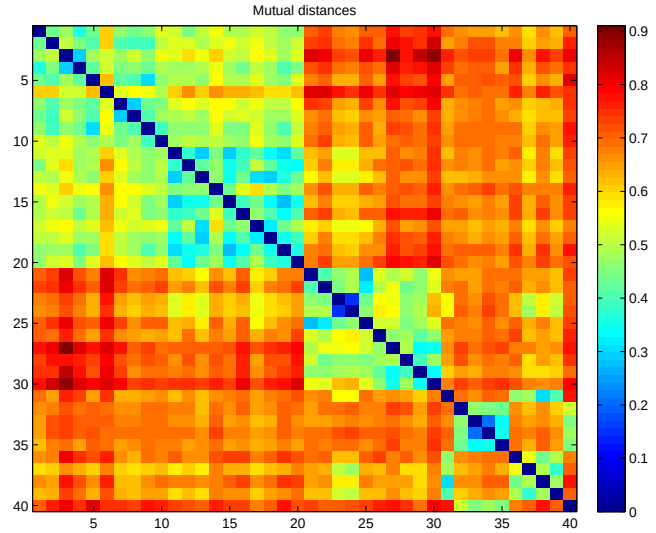

FIGURE 5.5: Mutual distances between AT&T pre-hash codes from test subset.

to pre-hash vector with respect to Angular distance.

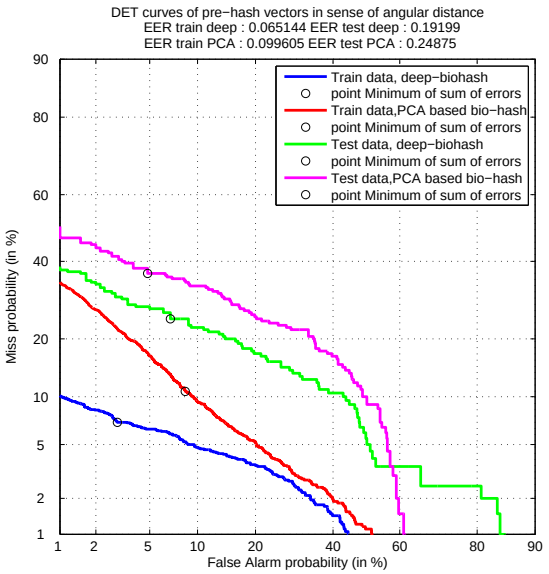

FIGURE 5.6: DET curve for AT&T dataset of pre-hash with angular distance .

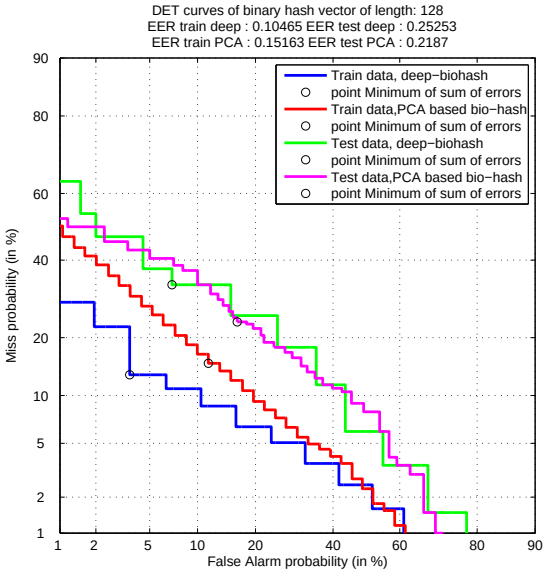

FIGURE 5.7: DET curve for AT&T dataset, hash vector of length 128.

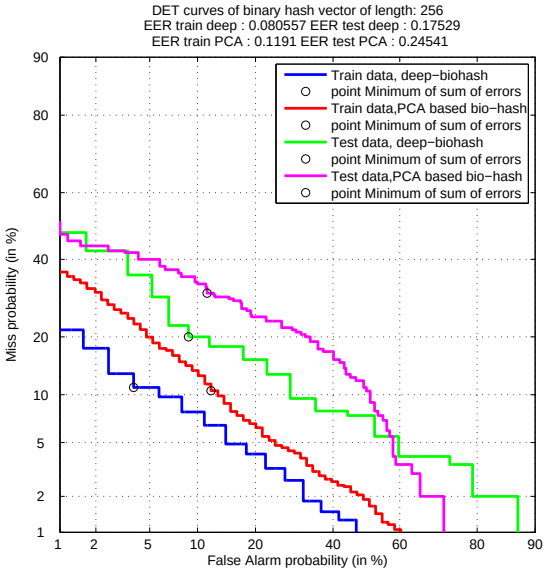

FIGURE 5.8: DET curve for AT&T dataset, hash vector of length 256.

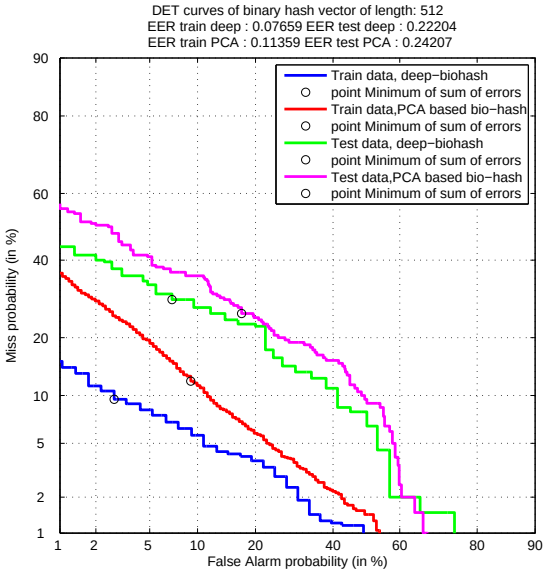

FIGURE 5.9: DET curve for AT&T dataset, hash vector of length 512.

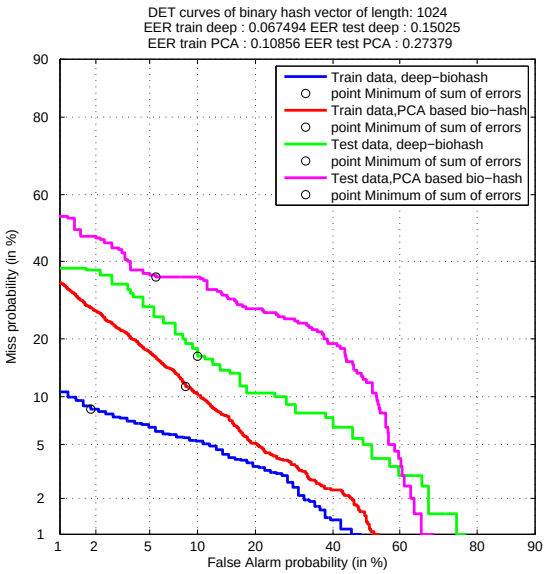

FIGURE 5.10: DET curve for AT&T dataset, hash vector of length 1024.

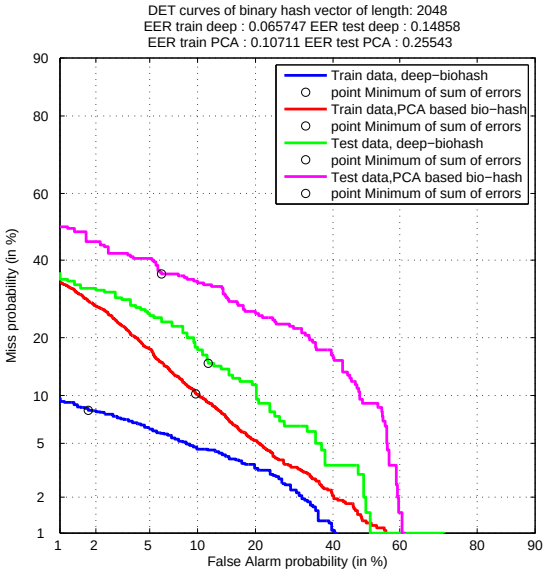

FIGURE 5.11: DET curve for AT&T dataset, hash vector of length 2048.

5.3.2 FERET dataset

For this task we crop FERET dataset images into semi-elliptic frame around the center of the face and then down-sampled the images to the size 75×65 pixels. Figure 5.12 shows some some samples of this dataset.

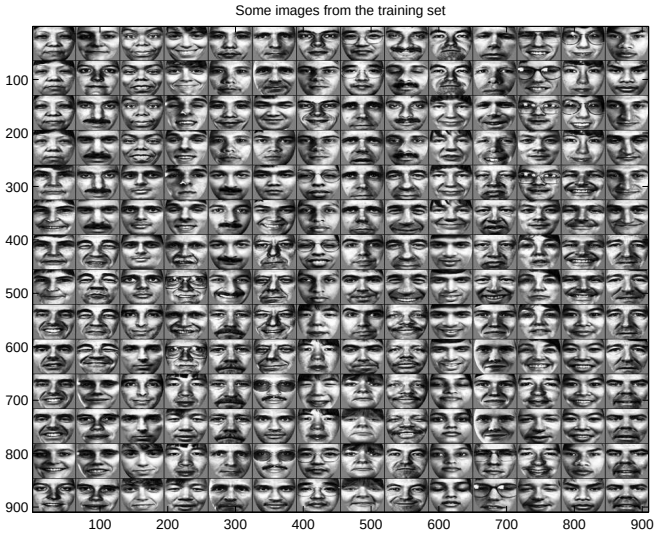

FIGURE 5.12: Some samples of FERET dataset.

Figure 5.13 shows changing of EER for validation set and Energy (mean of the energy for the batches.) in each epoch.

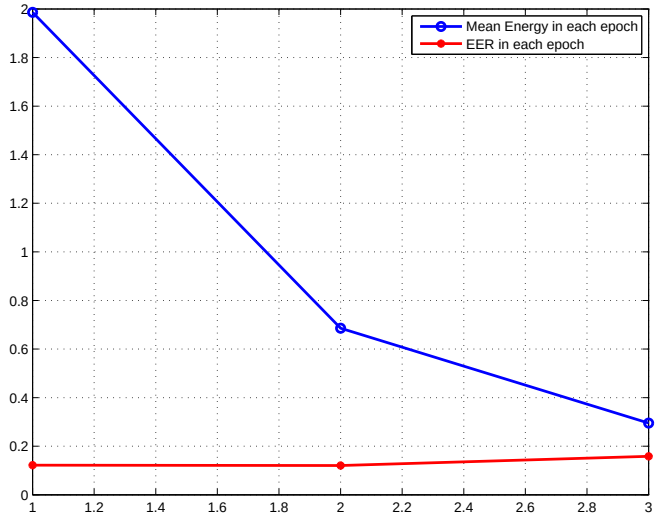

FIGURE 5.13: Changing of loss function and EER on validation data of the FERET database.

Figures 5.14 and 5.15 show angular distances between pre-hash vectors for each pairs of training and testing set respectively. Here we arrange data points such that all members of each class stick together and all class have equal number of members.

The plot of Detection Error Trade-off (DET) curve of PCA pre-hash vectors in comparison with pre-hash vectors obtained from deep learning method is shown in Figure5.16.

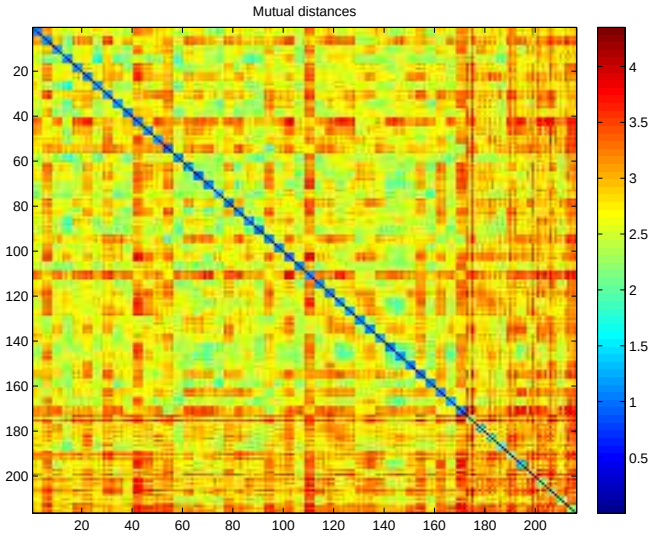

FIGURE 5.14: Mutual distances between FERET pre-hash codes from train subset.

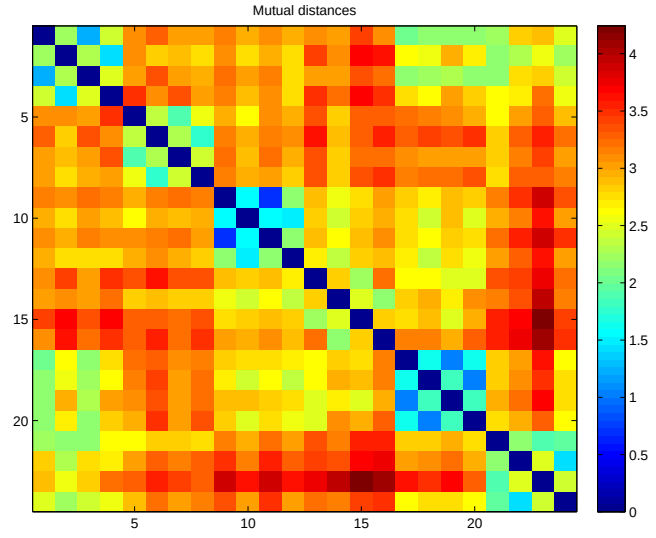

FIGURE 5.15: Mutual distances between FERET pre-hash codes from test subset.

DET curves for hash vectors of length 128, 256, 512, 1024 and 2048 are provided in Figures: 5.17 ,5.18,5.19,5.20 and 5.21 respectively.

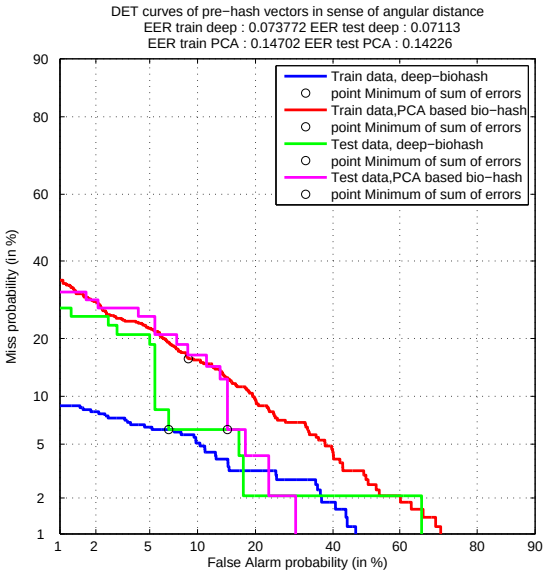

FIGURE 5.16: DET curve for FERET dataset of pre-hash with angular distance .

As we expect for this dataset too, using more bits for the biohash, the performance of the deep network gets better as compared to the PCA based one.

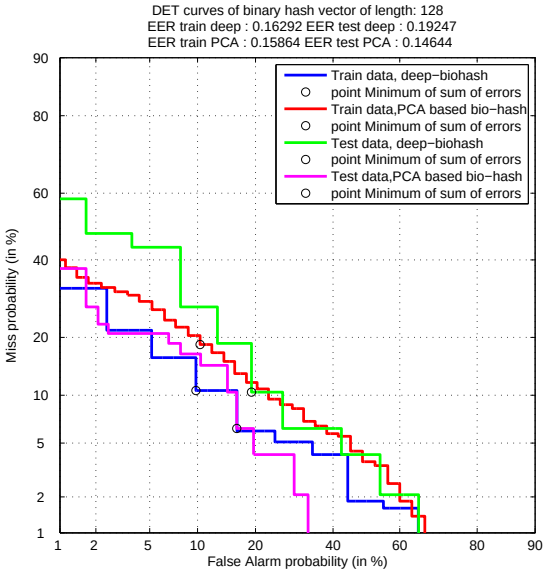

FIGURE 5.17: DET curve for FERET dataset, hash vector of length 128.

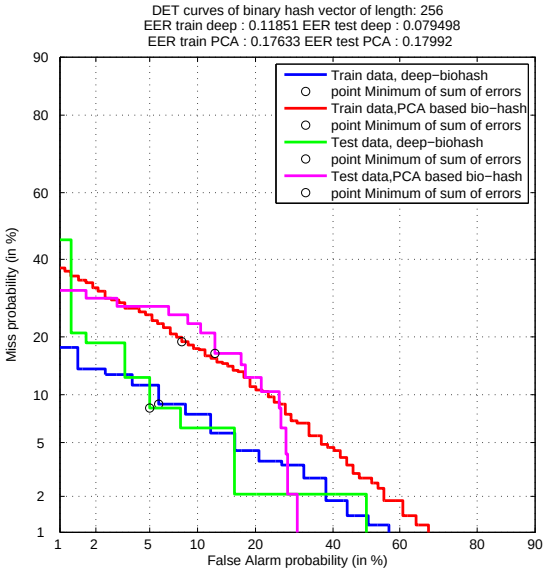

FIGURE 5.18: DET curve for FERET dataset, hash vector of length 256.

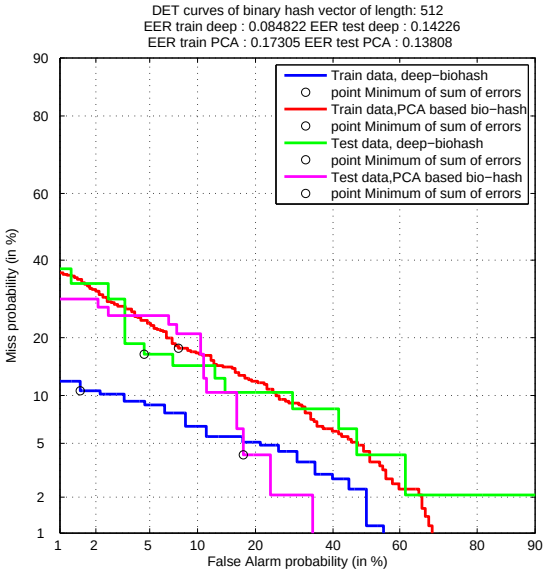

FIGURE 5.19: DET curve for FERET dataset, hash vector of length 512.

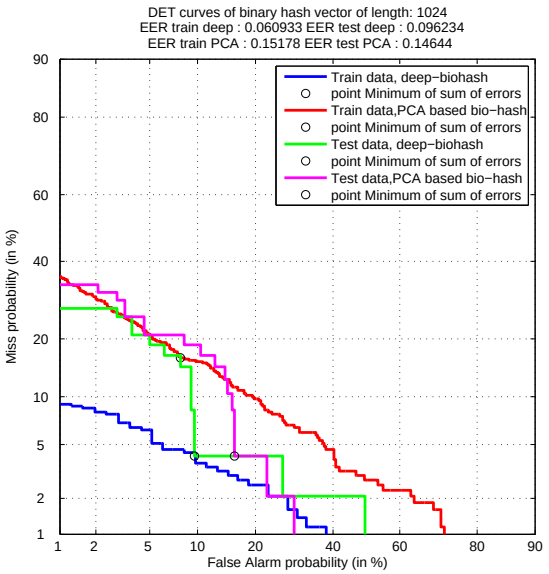

FIGURE 5.20: DET curve for FERET dataset, hash vector of length 1024.

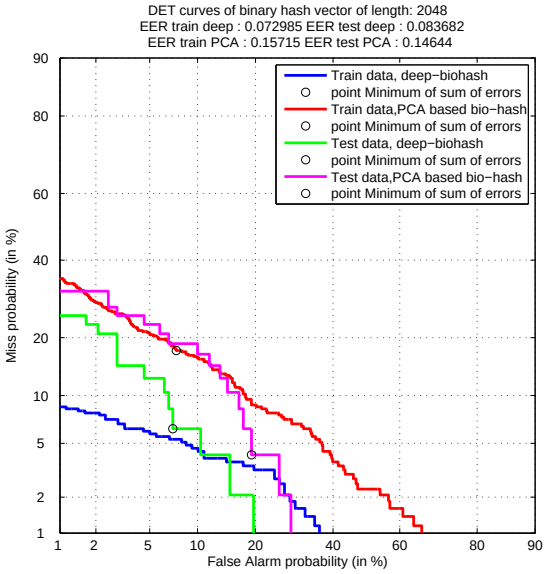

FIGURE 5.21: DET curve for FERET dataset, hash vector of length 2048.

5.3.3 CMU dataset

Figure 5.22 shows some samples of this dataset.

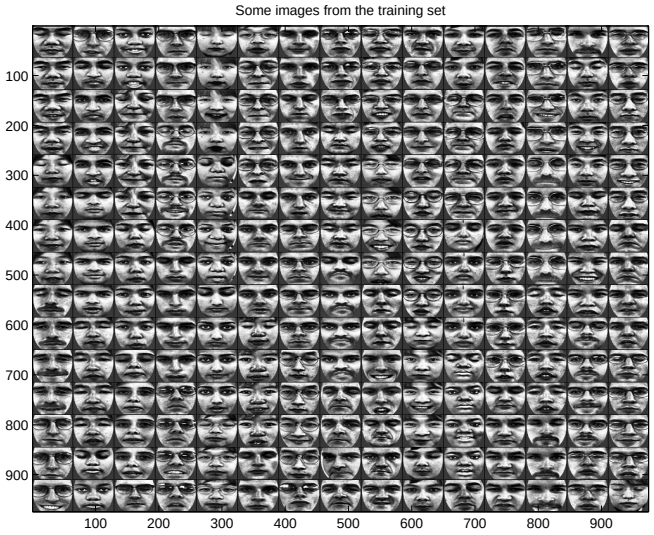

FIGURE 5.22: Some samples of CMU dataset.

The plot of Detection Error Trade-off (DET) curve of PCA pre-hash vectors in comparison with pre-hash vectors obtained from deep learning method is shown in Figure5.23.

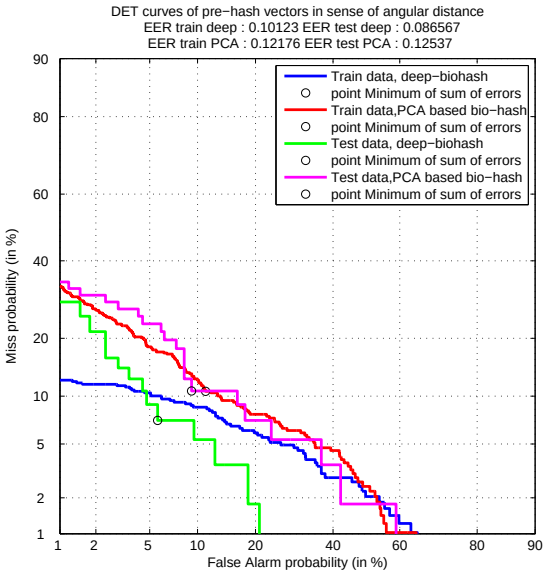

FIGURE 5.23: DET curve for cmu dataset of pre-hash with angular distance .

Figure 5.24 shows changing of EER for validation set and Energy (mean of the energy for the batches.) in each epoch.

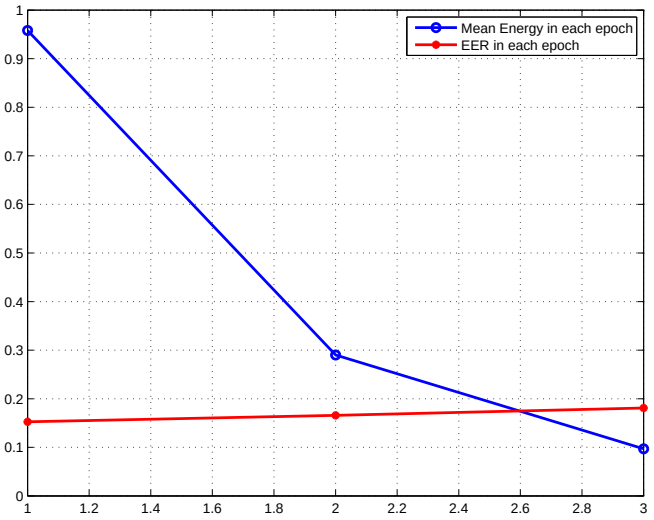

FIGURE 5.24: Changing of Energy and EER through learning over validation set of CMU dataset.

DET curves for hash vectors of length 128, 256, 512, 1024 and 2048 are provided in Figures: 5.25 ,5.26,5.27,5.28 and 5.29 respectively.

Here again, using more bits for the biohash, the performance of the deep network gets better as compared to the PCA based one.

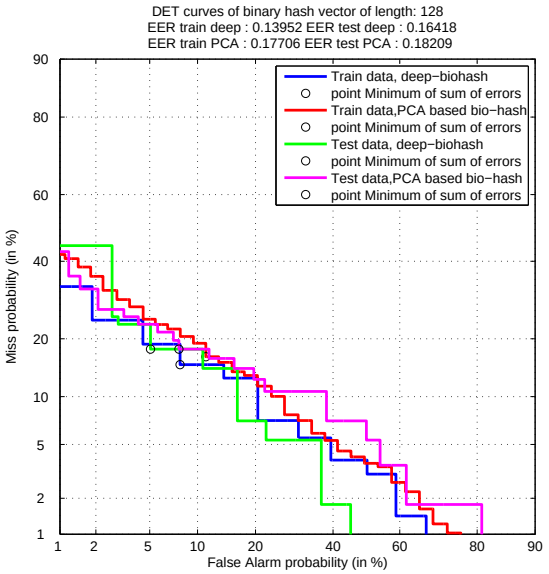

FIGURE 5.25: DET curve for CMU dataset, hash vector of length 128.

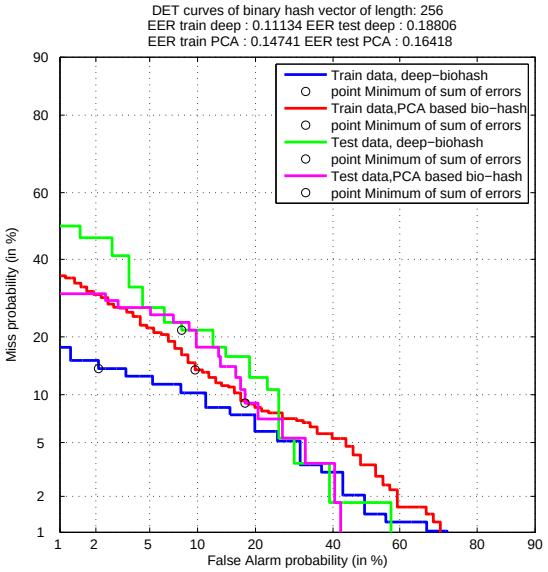

FIGURE 5.26: DET curve for CMU dataset, hash vector of length 256.

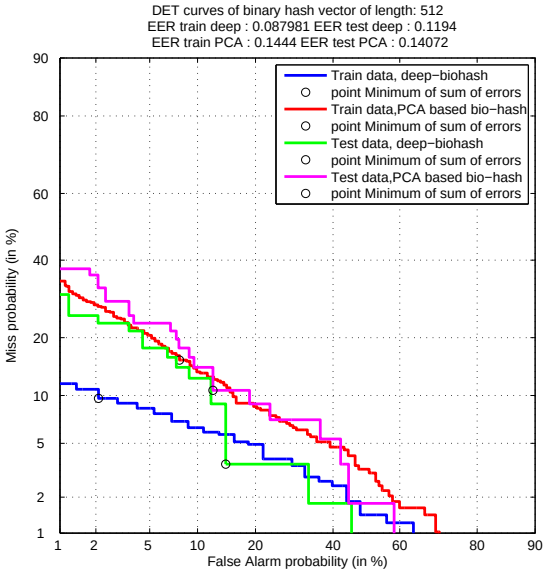

FIGURE 5.27: DET curve for CMU dataset, hash vector of length 512.

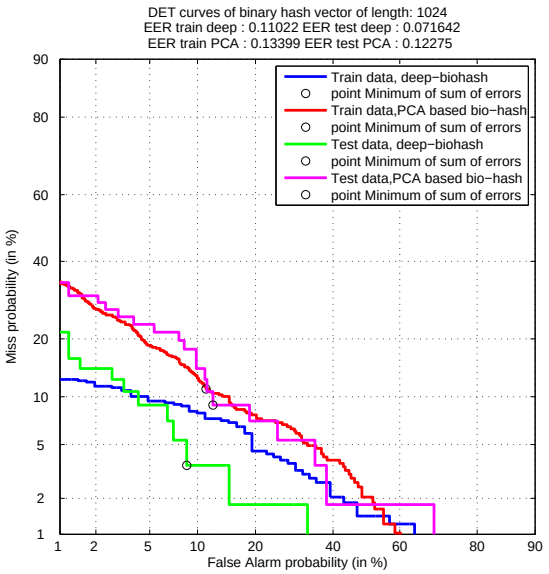

FIGURE 5.28: DET curve for CMU dataset, hash vector of length 1024.

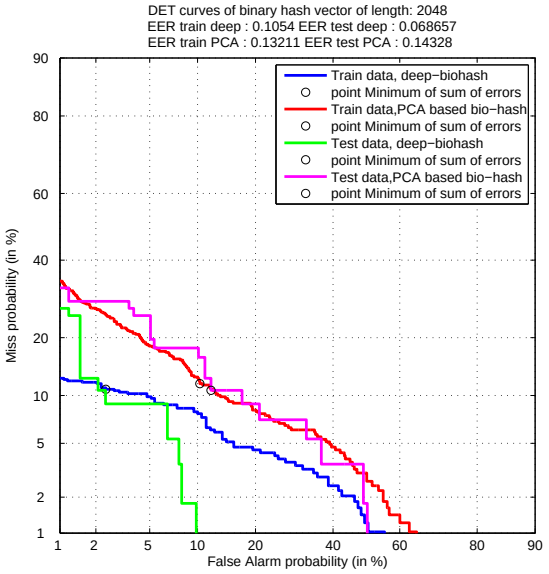

FIGURE 5.29: DET curve for CMU dataset, hash vector of length 2048.

5.3.4 M2VTS dataset

Figure 5.30 shows some some samples of this dataset.

Our method on this dataset is not as good as the results on previous datasets, although our purposed method still outperforms PCA based method on this dataset too. The reason may lay on visual similarity between the subjects.

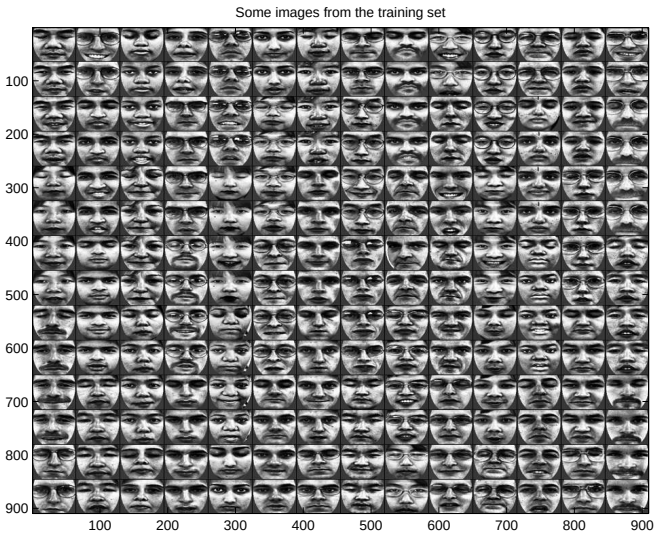

FIGURE 5.30: Some samples of M2VTS dataset.

Figure 5.31 shows changing of EER for validation set and Energy (mean of the energy for the batches) in each epoch. The plot of Detection Error Trade-off (DET) curve of PCA pre-hash vectors in comparison with pre-hash vectors obtained from deep learning method. According to theorem 4.1 using *Angular* distance give better measure to compare deep-learning-based and pca-based method 5.32.

Here again, using more bits for the biohash, the performance of the deep network gets better as compared to the PCA based one.

The EER of different subsets of dataset in this experiment have larger value compared to the previous datasets obtained by our purposed method, The reason for that may lay in small number of images for each subject and comparably large number of subjects.

Different Hash vector length can be achieved by changing the dimension of the random projection matrix R. Here we have DET curves for hash vectors of length 128, 256, 512, 1024 and 2048 are provided in Figures: 5.33 ,5.34,5.35,5.36 and 5.37 respectively.

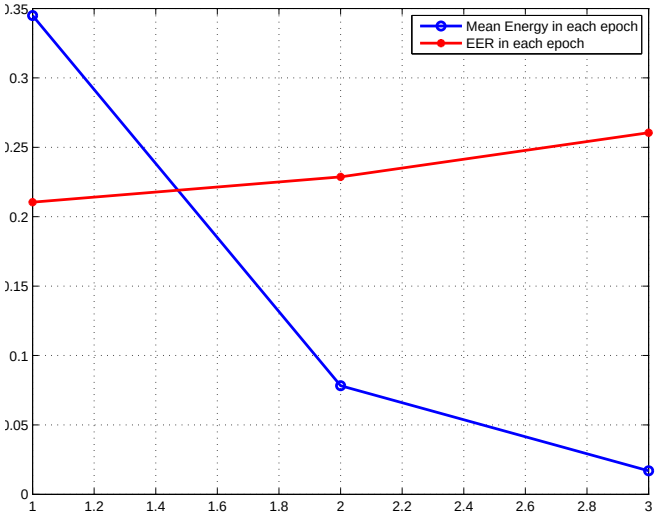

FIGURE 5.31: Changing of Energy and EER through learning over validation set of M2VTS dataset.

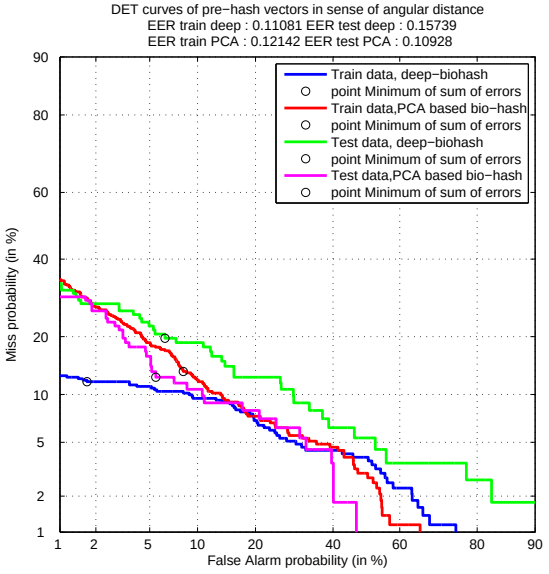

FIGURE 5.32: DET curve for M2VTS dataset of pre-hash with angular distance .

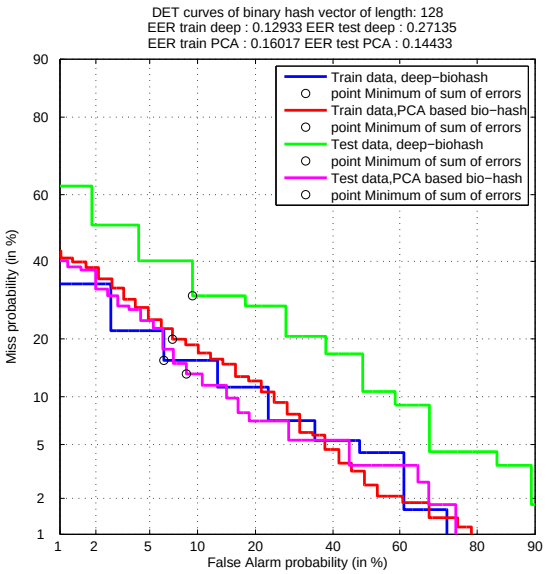

FIGURE 5.33: DET curve for M2VTS dataset, hash vector of length 128.

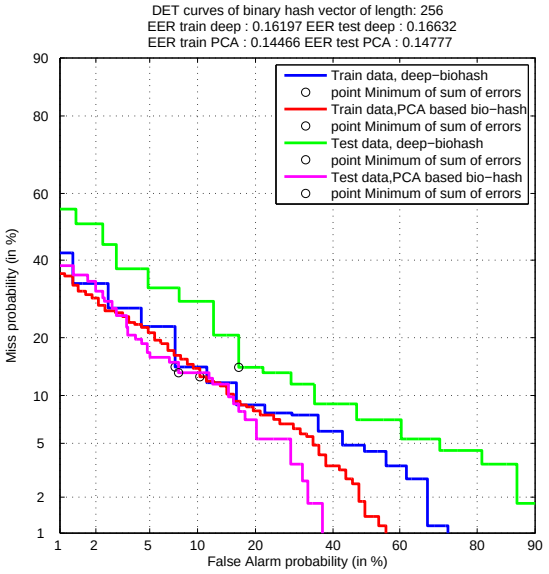

FIGURE 5.34: DET curve for M2VTS dataset, hash vector of length 256.

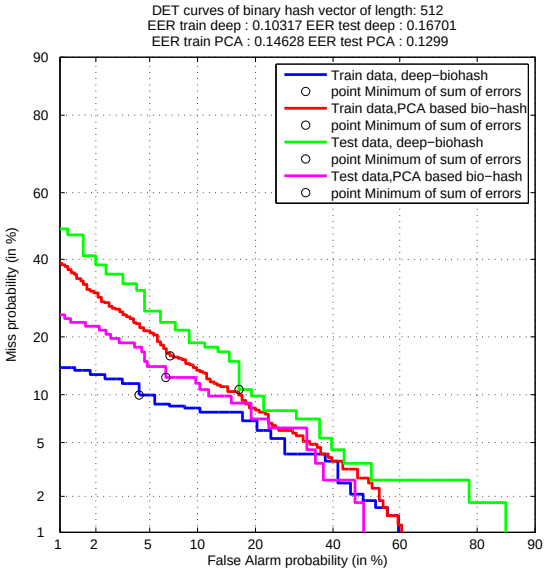

FIGURE 5.35: DET curve for M2VTS dataset, hash vector of length 512.

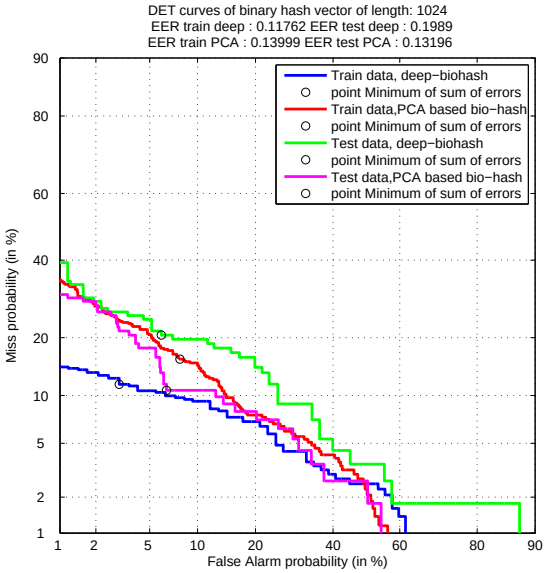

FIGURE 5.36: DET curve for M2VTS dataset, hash vector of length 1024.

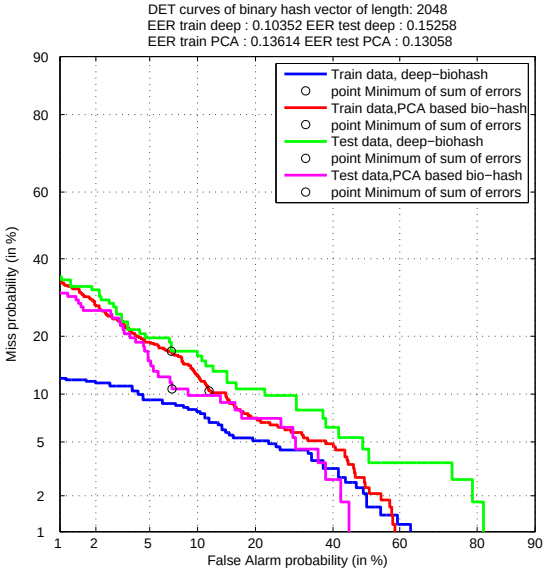

FIGURE 5.37: DET curve for M2VTS dataset, hash vector of length 2048.

5.4 Discussion

In this section, we discuss various parameters for Siamese neural network learning and their affects on the performance of the system. Since our models deals with several millions of parameter to work well, training procedure is quite important.

5.4.1 Number of Epochs

One interesting phenomena we witnessed in this research was the effect of number of epochs on performance of the network. We saw that the networks performance decrease by increasing number of epoch and training time. There is a discussion in [50] under the label **Early stopping** which claims that :

"Early stopping can be seen as restricting the optimization procedure to a relatively small volume of parameter space that corresponds to a local basin of attraction of the supervised cost function, by constraining the optimization procedure to a region of the parameter space that is close to the initial configuration of parameters."

The effect of different number of iteration for training the network on loss and performance (EER)of the network in one experiment using AT&T dataset 5.38 confirms the claim above.
Here the loss values are obtained over training data and EER is computed on validation set. In order to find the best point stop, we halt the algorithm whenever EER value starts to increase.

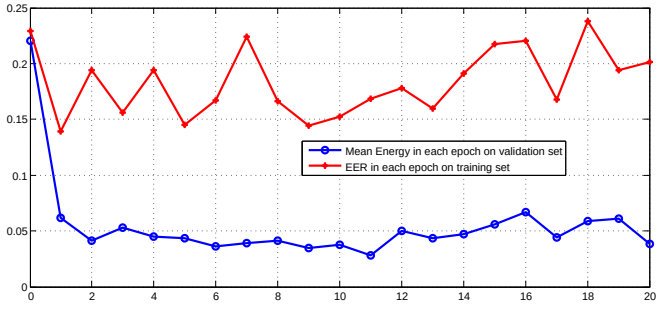

FIGURE 5.38: Effect of number of iteration on EER and loss.

5.4.2 Effect of the parameter α

In section 4.3.2 we discussed about the properties of the energy function. One of the properties obtained in [38] and [36] empirically is that there should be a bound for moving apart dissimilar pairs, i.e. if a dissimilar pair mapped to vectors are farther than a fixed threshold (called α) then it is necessary to penalize the network for this particular pair. Figure 5.39 show effect of different values for α EER of the network on test set and both train and test set.

The experiment results suggest that larger bounds, (α) will improve the performance of the system which may be expected, since when using smaller values of α the algorithm will stop updating its weights as soon as images belonging to different subjects map to vectors farther than α, which increases the possibility of error.

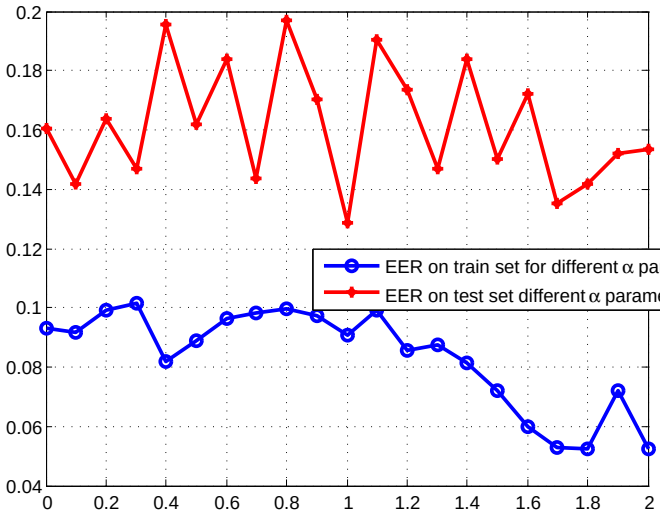

FIGURE 5.39: Effect of of α on EER on test and train set.

5.4.3 Effect of Batch size

As we said before in Section 3.2, mini-batch stochastic gradient method benefits from speed of calculation relative to the full batch method and also higher accuracy compared to the stochastic gradient method. We can think about batch stochastic gradient method with a fixed batch-size as a point on a spectrum with extremes on full gradient (offline method) and stochastic gradient (each batch only have one data point). In our experiments using smaller batches in training improves the performance of the network 5.40.

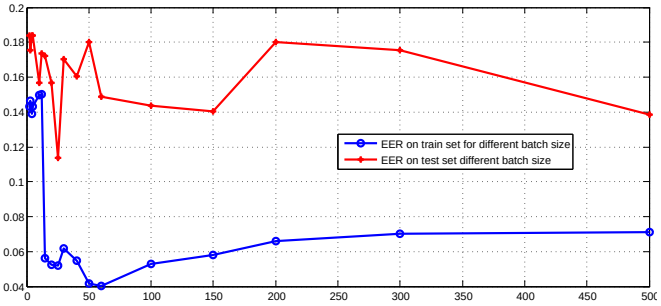

FIGURE 5.40: Effect of Batch size on EER and Energy.

5.4.4 Pretraining effect

Here are the results of three different methods for pretraining of the neural net, Random initialization, PCA coefficient for the first layer and initialized using stacked autoencoder. In contrary to some other machine learning task pretraining does not improves the performance of the networks and surprisingly randomly initialized net show better results.

This experiment is conducted over AT&T dataset. Here we had a 5-layer neural network of [300,100,40,100,256] unit in each layer respectively For all experiments the training stops after 10 iterations. The results are shown in Figures 5.41, 5.42.

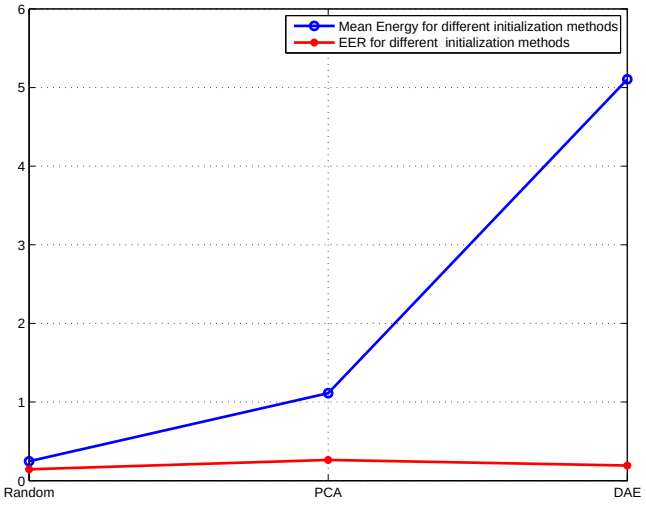

FIGURE 5.41: Energy and EER for randomly initialized, with PCA coefficients and deep auto encoder.

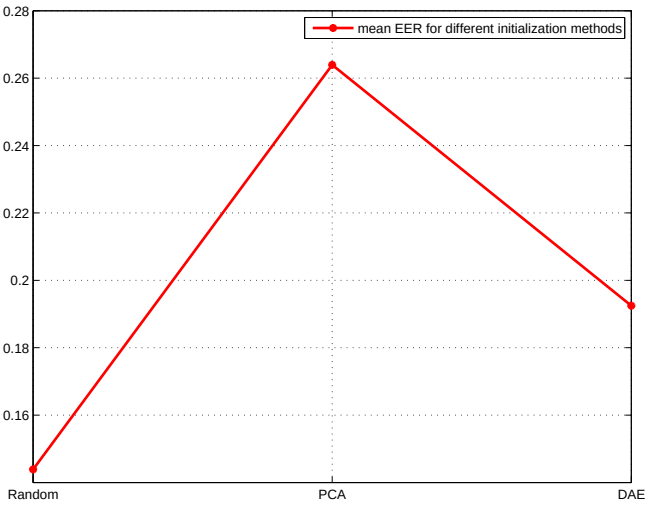

FIGURE 5.42: EER for randomly initilized, with PCA coefficients and deep auto encoder.

5.4.5 Importance of specialized loss using the angular distance

According to Theorem 4.1 angular distance approximates the Hamming distance after quantization properly, so we expect that the hash vectors obtained from networks with angular distance-base objective function outperform hash codes obtained with the same structure but objective function of different metric like in this experiment Euclidean distance.

This experiment is conducted over AT&T dataset. Here we had a 4-layer neural network of [1000,400,100,256] unit in each layer respectively. 5.43, 5.44,5.45 and 5.46

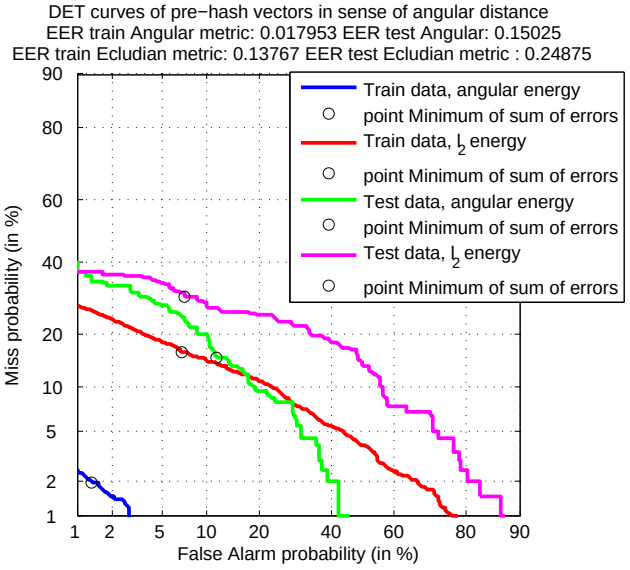

FIGURE 5.43: DET curves for pre-hash codes obtained from networks trying to decrease Euclidean distance versus angular distance.

As we expect, DET curves after applying binarization follows the result for pre-hash data.

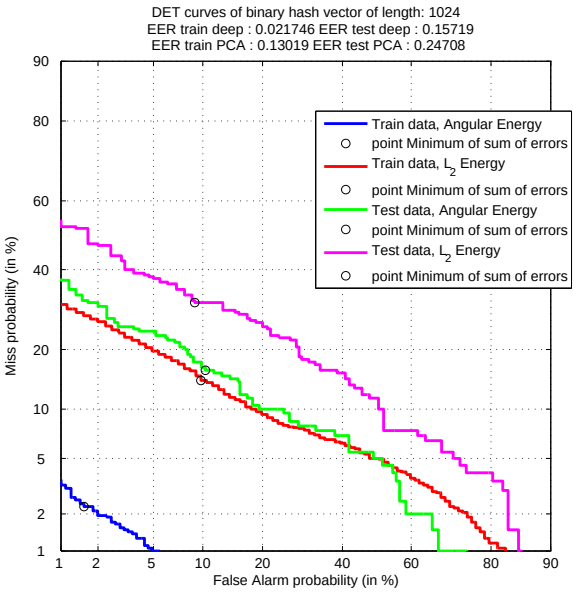

FIGURE 5.44: DET curves for Hash codes of length 1024 obtained from networks trying to decrease Euclidean distance versus angular distance.

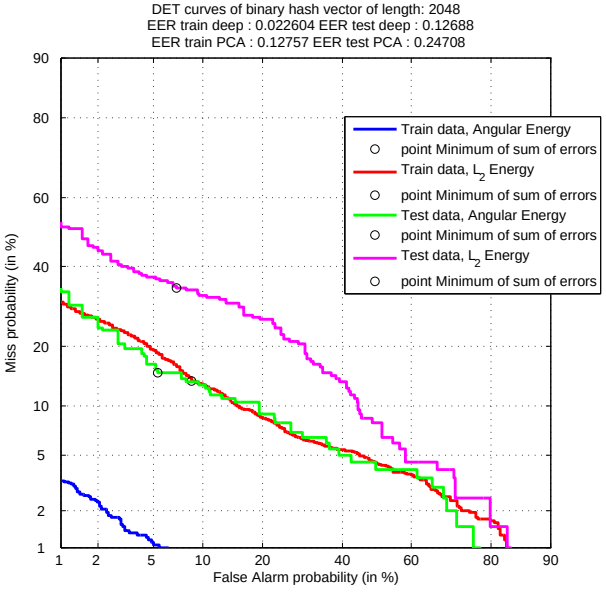

FIGURE 5.45: DET curves for Hash codes of length 2048 obtained from networks trying to decrease Euclidean distance versus angular distance.

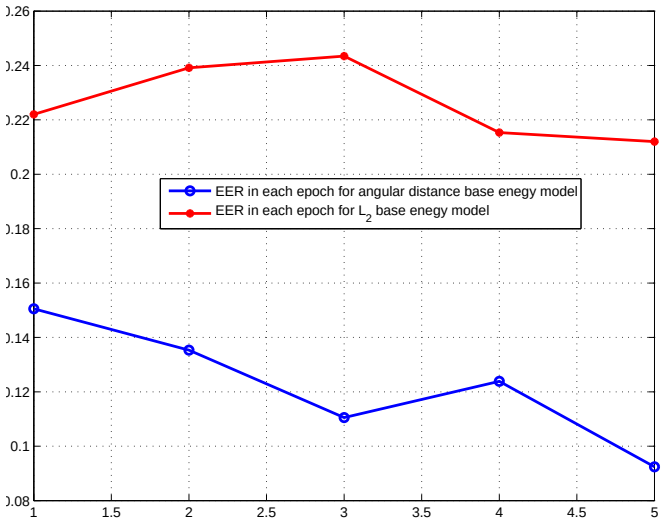

FIGURE 5.46: Energy and EER for networks trying to decrease Euclidean distance and angular distance on validation set of AT&T dataset.

Chapter 6

Conclusion and summary

Our experiments and results suggests that Siamese neural networks can lead to good results on biometric hashing and for face images with proper setting and training can outperform an existing method such as PCA based method. The manifold learned by Siamese model can learn invariances in complex classes and exclude dissimilar samples even in a complex case like face images.

Choosing a good metric can improve the result dramatically. In this case *angular distance* improves the results.

Since our model potentially has huge number of elements, regularization plays important role in training. In our special case, it seems that the error surface is highly nonlinear with lots of local minima(based on the results by trying to escape shallow local minima we may be trapped in even shallower local minima.). Early stopping can help improve the results, we can guess the number of required iteration by cross-validation.

Good initialization is another vital parameter. Despite the popularity of autoencoder initialization in some pattern recognition tasks, this initialization does not improve the result in this case and random initialization leads to better results.

6.1 Future work

The system we introduced here for biometric hashing can be also used for different biometric data, like voice, signature, etc.. These areas could be explored in future

research.

Another important challenge in this area of research is to find better energy functions for purposes like the ones we had in this project.

Bibliography

[1] Ovie Carroll and Mark Krotoski, "Using 'digital fingerprints' (or hash values) for investigations and cases involving electronic evidence," *United States Attorneys Bulletin*, vol. 62, no. 3, pp. 44–82, 2014.

[2] David CL Ngo; Andrew BJ Teoh and Alwyn Goh, "Biometric hash: high-confidence face recognition," *Circuits and Systems for Video Technology, IEEE Transactions*, vol. 16, pp. 771–775, 2006.

[3] P. Gupta; A. Rattani; H. Mehrotra and A. K. Kaushik, "Multimodal biometrics system for efficient human recognition," in *Defense and Security Symposium*, International Society for Optics and Photonics, 2006, pp. 62020Y–62020Y.

[4] A. Ross and A. K. Jain, "Information fusion in biometrics," *Pattern Recognition Letters*, vol. 24, no. 13, pp. 2115–2125, 2003.

[5] M. Kass; A. Witkin and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision 1.4*, pp. 321–331, 1988.

[6] P. Hough, "Methods and means for recognizing complex patterns," 1962.

[7] B. Topcu; H. Erdogan; C. Karabat and B. Yanikoglu, "Biohashing with fingerprint spectral minutiae," in *BIOSIG*, Darmstadt, September 2013.

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

[9] R. Bolle; A. K. Jain; and S. Pankanti (Eds.), *Biometrics: Personal Identification in Networked Society*, Kluwer Academic Publishers, 2000.

[10] M. Turk and A. Pentland, "Eigenfaces for recognition. journal of cognitive neuroscience," *Journal of Cognitive Neuroscience*, vol. 1, no. 1, pp. 7186, 1991.

[11] Wang; Yunhong; Tieniu Tan and Anil K. Jain, "Combining face and iris biometrics for identity verification," in *Audio-and Video-Based Biometric Person Authentication*, pp. 805–813. Springer Berlin Heidelberg, 2003.

[12] Soyuj Kumar; Tarun Choubisa Sahoo and S. R. Prasanna, "Multimodal biometric person authentication: A review," *IETE Technical Review*, vol. 29, no. 1, 2012.

[13] M. J. Jones and J. M. Rehg, "Statistical color models with application to skin detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1999, vol. 1, pp. 274–280.

[14] Andrew B. J. Teoh; Alwyn Goh and David C. L. Ngo, "Random multispace quantization as an analytic mechanism for biohashing of biometric and random identity inputs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 18921901, 2006.

[15] C. Karabat and H. Erdogan, "Error-correcting output codes guided quantization for biometric hashing," *IEICE Transactions on Information and Systems*, vol. E95-D, no. 6, pp. 1707–1712, Jun. 2012.

[16] C. Karabat and H. Erdogan, "Discriminative projection selection based face image hashing," *IEICE Transactions on Information and Systems*, vol. E95-D, no. 5, pp. 1547–1551.

[17] David Ngo Chek Ling; Andrew Teoh Beng Jin and Alwyn Goh, "Eigenspace-based face hashing," in *ICBA*, 2004, p. 195199.

[18] Svante Wold; Kim Esbensen and Paul Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1, pp. 37–52, 1987.

[19] Y. Bengio; A. Courville and P. Vincent, "Representation learning: A review and new perspectives," *Pattern Analysis and Machine Intelligence, IEEE Transactions*, vol. 35, pp. 1798 – 1828, 2013.

[20] Randall C O'Reilly and Yuko Munakata, *Computational Explorations in Cognitive Neuroscience*, The MIT Press, 2000.

[21] Ben Krose and Patrick Van der Smaft, *An introduction to neural networks*, University of Amsterdam, 1996.

[22] Michael A. Nielsen, *Neural Networks and Deep Learning*, Determination Press,, 2014.

[23] Léon Bottou, "Stochastic gradient descent tricks.," *In Neural Networks: Tricks of the Trade Springer Berlin Heidelberg.*, pp. 421–436, 2012.

[24] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, vol. 2, pp. 1–127, 2009.

[25] H. Larochelle; D. Erhan; A. Courville; J. Bergstra and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," *Proc. ICML*, pp. 473–480, 2007.

[26] H. Larochelle; Y. Bengio; J. Louradour and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[27] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen*, Ph.D. thesis, f. Informatik, Technische Univ., Munich, 1991.

[28] S. Hochreiter; Y. Bengio; P. Frasconi and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[29] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, vol. 4, pp. 234242, 1992, 2009.

[30] G. Hinton; S. Osindero and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[31] GE Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, pp. 185234, 1989.

[32] Paul E Utgo and David J Stracuzzi, "Many-layered learning," *Neural computation*, vol. 14, no. 10, pp. 2497–529, 2002.

[33] Pascal Vincent; H Larochelle; I Lajoie; Y Bengio and P Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, no. 10, pp. 3371–3408, 2010.

[34] G. Hinton, "Learning multiple layers of representation," *Trends in Cognitive Science*, vol. 11, pp. 428–434, 2007.

[35] Y. Bengio P. Vincent, H. Larochelle and P. Manzagol, "Extracting and composing fobust features with denoising a autoencoders," *Proc. ICML*, pp. 1096–1103, 2008.

[36] S. Chopra; R. Hadsell and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," *Computer Vision and Pattern Recognition*, vol. 1, pp. 539–546, 2005.

[37] Sun Yi; Xiaogang Wang and Xiaoou Tang, "Hybrid deep learning for face verification," *Computer Vision (ICCV)IEEE International Conference*, pp. 1489–1496, 2013.

[38] G. W. Taylor; I. Spiro; Ch. Bregler and R. Fergus, "Learning invariance through imitation," in *IEEE Computer Society*, Washington, DC, USA, 2011.

[39] J. Bromley; I. Guyon; Y. LeCun; E. Sackinger and R. Shah, "Signature verification using a siamese time delay neural network," *Advances in Neural Information Processing Systems*, pp. 737–744, 1993.

[40] Y. LeCun; S. Chopra; R. Hadsell; M. Ranzato and F. Huang, "A tutorial on energy-based learning," *Predicting structured data*, 2006.

[41] S. A. Rizvi; H. Moon and P. J. Phillips, "The feret verification testing protocol for face recognition algorithms," *Automatic Face and Gesture Recognition, Third IEEE International Conference*, pp. 48–53, 1998.

[42] R. Hadsell; S. Chopra and Yann LeCun, "Dimensionality reduction by learning an invariant mapping," *Computer vision and pattern recognition*, vol. 2, pp. 1735–1742, 2006.

[43] H. Mobahi; R. Collobert and J. Weston, "Deep learning from temporal coherence in video.," *In Proceedings of the 26th Annual International Conference on Machine Learning,ACM*, pp. 737–744, 2009.

[44] Jason Noah Laska, *Regime Change: Sampling Rate vs. Bit-Depth in Compressive Sensing*, Ph.D. thesis, RICE UNIVERSITY, Munich, 1991.

[45] Jacques; Laurent; Jason N. Laska; Petros T. Boufounos and Richard G. Baraniuk, "Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors," *arXiv preprint arXiv:1104.3160*, 2011.

[46] Ferdinando Samaria and Andy Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of 2nd IEEE Workshop on Applications of Computer Vision*, Sarasota FL, 1994.

[47] P. Jonathon; Hyeonjoon Moon; Syed A. Rizvi Phillips and Patrick J. Rauss, "The feret evaluation methodology for face-recognition algorithms," *Pattern Analysis and Machine Intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000.

[48] Juergen Luettin and Gilbert Matre, "Evaluation protocol for the extended m2vts database (xm2vtsdb)," Tech. Rep. EPFL-REPORT-82488, IDIAP.

[49] Ole Bernsen, "4.4 cmu pose, illumination, and expression (pie) database," Tech. Rep. 98, Deliverable status Public Contractual date of delivery.

[50] D. Erhan; Y. Bengio; A. courville; P. Manzagol and P. Vincent, "Why does unsupervised pre-training help deep learning?," *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.