Facial Verification using Fisher Vectors and Deep Nets

Francois Chaubard (fchaubar@stanford.edu), Mike Chrzanowski (mc2711@stanford.edu)

Abstract

Our project tackled the problem of verifying whether two facial images are of the same person or not. Specifically, we sought to replicate the results of Simonyan, et al. in their paper “Fisher Vector Faces in the Wild”, which achieves state of the art verification performance on the popular Labeled Faces in the Wild (LFW) dataset. We then sought to obtain better performance by introducing new elements into the processing pipeline such as feature augmentation and use of deep neural nets. Finally, we created a publicly-facing website for our work at fisherfaces.com that anyone can use to perform verification in real time.

1. Introduction

Teaching a computer to decipher between two faces is an unsolved task. Many companies such as microfinance firms (Lenddo, Lending Club) and social networks (Facebook), spend millions of dollars to perform user validation to answer the question “is this person who they say they are?”. One of the first steps in validation is to ensure the picture associated with a profile is unique and consistent with other pictures (i.e. passport/license pictures). For example, Lenndo is a microfinance company that delivers credit scores based on Facebook profiles. Ensuring an applicant’s I.D. corresponds with his/her profile picture(s) is essential in its verification process. The only way to do so is to compare the faces of the image of a photo-I.D. and the Facebook profile pictures. If they deem two faces to be the same incorrectly, they could very well give a loan to a phony account and lose money. Therefore, developing an algorithm that can achieve human level accuracy in deciding same face vs. different face (as demonstrated in Figure 4) would be extremely useful. The method we use to establish a baseline is that of the Simonyan, et al, paper “Fisher Vector Faces in the Wild” [0]. From there, we experiment with various parts of the algorithm to try and boost performance. We include below a schematic for the pipeline of the algorithm, although the details of how everything works is presented in Section 3.

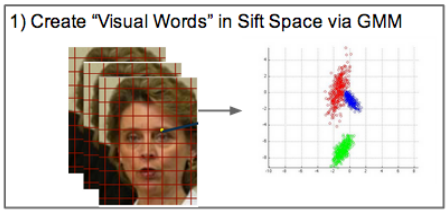

Figure 1: We create 512 “Visual Words” to describe a face by performing GMM (Gaussian Mixture Model) on a bank of SIFT Descriptors generated from face data.

2) Learn W and b

$$\arg\min_{W,b} \sum_{i,j} \max\left[1 - y_{ij}\left(b - (\phi_i - \phi_j)^T W^T W (\phi_i - \phi_j)\right), 0\right]$$

Figure 2: We create a Fisher Vector for each image and optimize the above formula to find W and b

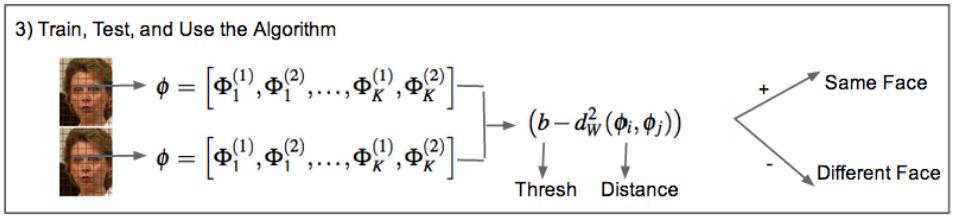

Figure 3: We use the learnt W as a Mahalanobis Distance which can be interpreted as the distance of the differences of the Fisher vectors in this subspace. We threshold this distance by another learnt parameter b and declare the Fisher vectors represent same faces if the distance is below b, and represent different faces if the distance is above b.

We uploaded the parameters from our best performing algorithm to a webserver which serves as a frontend for our work and which streamlines the process described in Figure 3, in an effort to study what weak points our algorithm has; (i.e. with regard to certain skin colors, demographics, sex, etc). Below is the output of two successful run of our algorithm on the website.

Figure 4: Two outputs of fisherfaces.com which is a website that runs our algorithm on two user fed images. On the left, we have two images of Professor Silvio Savarese and our algorithm correctly detects it as such. On the right, we have Dr. Savarese and John F. Kennedy. Note that our algorithm also correctly detects that the two faces are different.

2. Previous Work and Key Project Contributions

2.1 Literature Review

Since before 1987, computer scientists have been developing algorithms to perform this very task. The first few breakthrough algorithms were mostly subspace learning algorithms. The first of these, EigenFace [7], takes a basket of face images and computes a linear subspace on its data matrix using PCA (Principal Component Analysis) to discover a linear subspace that ideally spans the space of possible faces. However, since the algorithm initially treats every pixel as a feature, it has a number of weaknesses, such as non-robustness to misalignment, pose, and variation in light. Also, if the differences between the training images and the test image is large, the algorithm will perform very poorly. EigenFace achieves 60% test accuracy on the LFW dataset. FisherFace [8] is another subspace learning algorithm that finds the most discriminating subspace for a set of labeled faces via Linear Discriminant Analysis. FisherFace performs better than EigenFace in the presence of light and pose variation, but still performs poorly on test images of persons not in the training set. Most of these linear subspace learning algorithms fail to capture a good model for a face because the “face” subspace of the pixel-space is not linear, but instead rests on a high dimensional “manifold”.

Other algorithms [9] attempt to solve for this manifold directly via more recent spectral methods, performing Weighted Kernel PCA, i.e. finding a subspace of an infinite dimensional space. [10] attempts to find a multilinear subspace, representing the face image data matrix as a tensor rather than a matrix. Although

each of these ideas result in improvement, none achieve performance anywhere near “human level”, 97.5% [11]. This is, at least in part, due to the utilization of pixels directly as features. A more robust representation of a face other than pixels is required, since pixels tend to be very noisy. Modern patch descriptors such as SIFT, SURF, FREAK, etc, are much more robust to changes in illumination, rotation, and scale, and therefore, make better initial features to use. This idea brought about a new batch of algorithms that resulted in much better performance. In [18], Guillaumin, et al, utilize this representation to reach performance of 79.27% with a fairly straight forward distance metric learning algorithm. They mention that a single SIFT descriptor resulted in 74.6% accuracy just by itself!

There have been some attempts to look at the problem from an “elastic” graph perspective [12] and [19], using something similar to an Object Bank approach, in which we look for landmarks on a face, determine pose, and then attempt to either correct for this pose into a canonical form or compute a distance metric directly. However, these algorithms are computationally very expensive since we are representing the face in a 2D, or sometimes 3D, morphable model which is impossible unless we have more than one image per face and there is a rotation difference in the images. We also require that there is no occlusion in either image. These requirements make this type of model less fitting for our problem.

Finally, the most recent and popular approach to face verification utilize Neural Networks to either directly solve from pixels to classification or from some post step after one of the previously mentioned algorithms. [13] uses a 4 layer Pyramid CNN (Convolutional Neural Network) and achieves 85.8% on the LFW restricted training setting. This approach, and other flavors of it, currently achieve state of the art on the LFW dataset in the unrestricted setting but not on the restricted. We, therefore, look to merge the successful ideas of these methodologies and the current state of the art on the restricted setting [0].

2.2 Key Project Contributions

We look to leverage the image representation of the Fisher vector approach of [0], since this algorithm works the best in the restricted setting (87.47%) [11]. This suggests that this representation is the most expressive and learns the fastest, i.e. on restricted training, it outperforms all other used features.

We conduct two broad experiments on the algorithm:
(1) For pre-Fisher vector formulation, we experiment with different amounts of SIFT descriptors to extract per face image, 26k and 41k. We also attempt to add more interesting features to the SIFT descriptors. In the paper, they add location features to the descriptors (the x, y coordinates of each densely sampled SIFT descriptor). We experiment with adding not only location but color features and skin-pixel detection. Attempting to add too much here will result in multicollinearity, hence, we are careful not to add overlapping information.
(2) For post-Fisher vector formulation, the paper outlines a Mahalanobis distance learning algorithm using Fisher vectors. However, we wanted to experiment with this. First, we attempt an SVM approach to establish a baseline. We randomly sample 500,000 same face pairs and 500,000 different face pairs and compute the difference of their Fisher vectors to give us 1 million training samples where each sample is a 67,584 “Differenced Fisher vector”. We use this to train a linear SVM. We also use these samples to train a feedforward neural network (FNN). Since the dimensionality of the training samples is 67,584, we perform SVD on the data set and only use the first 500 singular vectors (representing 69.6% of the energy of the data matrix). We then perform the same SVM and FNN approach on a subspace of the data as well. The results of these experiments are demonstrated and discussed in Section 4.

3. Technical Details

3.1 Summary

We partition the processing pipeline into pre-processing, training, and testing stages, which we briefly describe and in which we'll go into detail in 3.2. The pre-processing step consists of first creating all dense SIFT descriptors for all images and then selecting a subset of them to fit the parameters for a Gaussian Mixture Model (GMM). We then use the GMM parameters to create Fisher vector representations for each image, which we store for later use. The training step involves learning a projection matrix W and a threshold b such that, for two Fisher vectors φ_i, φ_j of images of the same person:

$$f(\varphi_i - \varphi_j) = b - (\varphi_i - \varphi_j)^T W^T W (\varphi_i - \varphi_j)$$

is very positive whereas for two images of different people, the same function produces very negative output. Testing then consists of creating Fisher vector representations of two images and then observing the result of the function above.

Because we sought to replicate their method (and hopefully improve on it), our algorithmic analysis hews very closely to the analysis Simonyan, et al. present. Where we differ is chiefly in providing intuition and clarification for various steps of the algorithm that confused us *a priori* as well as identifying the implementation tricks that were required to make the algorithm work and which were **not** detailed in the paper. We also discuss differences of approaches between our implementation and theirs, when applicable. We then discuss the work involved to make fisherfaces.com operational and also talk briefly about augmentations we tried to improve on Simonyan, et al's algorithm, which saw no significant performance boost.

3.2 Details

3.2.1 Things that Work: The Current Pipeline

3.2.1.1 Preprocessing

In the 'Unrestricted' setting, the creators of the LFW dataset have created 10 splits of the approximately 13,000 images in the corpus. The training set then consists of all images in 9 of the splits, and validation is then composed of all images in the left-out split.

The first step is to first run the (MATLAB-implemented) Viola Jones detector on an image to create a bounding box for the face in the image. The image is then cropped to this bounding box and is resized so that it is 160 x 125 pixels. As is the standard, the color space is converted from RGB to grayscale.

We then fit a GMM model to the training data. For this, we iterate through each image in the training set and compute its dense SIFT descriptor representation. Simonyan, et al. advocate using a stride of 1 pixel with 5 bin sizes that are scaled $\sqrt{2}$ apart to generate about 26,000 128-dimensional descriptors per image. We actually found that using slightly smaller bin sizes to generate 41,000 descriptors works slightly better and produces better results. We use the dense SIFT implementation provided in `vlfeat` [20] for this task.

We note that each image consists of 160x125x3 pixels and, if we represent each pixel with 4 bytes, we get about 234 KB. After the SIFT descriptor generation, each image representation now requires 128 * 41,000 * 4 bytes ~ 20 MB. As we have around 12,000 images in the training set, storing all 429 million descriptors for the images would take about 234 GB. Although we have access to machines with this much RAM

(specifically, Amazon Web Service's cr1.8xlarge EC2 instance has 244 GB and 32 CPUs), we found the runtime of the GMM process on so many samples is unacceptably long. After consultation with Karen Simonyan, himself, we found that in practice, a decent GMM fit can be achieved if we down sample 1 million descriptors, uniformly. So, our process generates the descriptors per image and then samples 100 of them for storage in memory. The rest are discarded.

At this point, Simonyan, et al. advocate calculating the RootSIFT representations of each descriptor. The key idea here is that using the Hellinger kernel, which is defined as $H(x,y) = \sum_{i=1}^{n} \sqrt{x_i}\sqrt{y_i}$, to compare histograms has been found to result in better matches than using Euclidean distance [1]. The steps involved to use this kernel instead of Euclidean distance are straightforward: L1 normalize each descriptor and then take the element-wise square root. However, we found that using RootSIFT resulted in little to no performance impact on our final numbers, and so to simplify our processing pipeline, we excluded this step for most of our experiments.

Prior to running GMM, the authors also advocate running PCA to reduce the dimensionality of the the descriptors so as to reduce the amount of noise introduced when fitting the means and covariances of the GMM clusters. This is extremely important as these will be used as a dictionary in the rest of the pipeline. We find the 64 most significant principal components to project the sampled SIFT descriptors onto and then serialize this matrix for later use.

At this point, for each post-PCA descriptor, we add spatial information regarding its keypoint, which is not captured by SIFT descriptors *a priori*. Specifically, we add the values $[\frac{x}{w} - \frac{1}{2},\ \frac{y}{h} - \frac{1}{2}]$, where (x, y) is the center of the patch of image data that this particular descriptor belongs to, w is the image width (125 for us), and h is the image height (160). Thus, after these steps, we have a descriptor matrix of dimensionality 66x1 million. Finally, we fit 512 clusters to this data using `vl_gmm`, a parallelized implementation available in `vlfeat`. To further prevent overfitting, we constrain the cluster covariance matrices to be diagonal. This process takes about half an hour and results in 512 means, covariances, and priors that are then serialized for later use.

Having fit our GMM parameters, we then create a Fisher vector for each image. A Fisher vector for an image is a collection of partial derivatives of the log-likelihood of the descriptors being generated from these fit parameters [2]. Formally, for image z that is represented with SIFT descriptors $X = \{x_{z1}, ..., x_{zN}\}$, we solve the following optimization problem:

$$maximize\ L(X|\theta) = \sum_{n=1}^{N} log(p(x_n|\theta))$$

$$subject\ to \sum_{k=1}^{K} w_k = 1$$

Where:

$$p(x_n|\theta) = \sum_{k=1}^{K} w_k p(x_n|\theta_k)$$

$$p(x_n|\theta_k) = \frac{exp(-\frac{1}{2}(x_n-\mu_k)^T\Sigma_k^{-1}(x_n-\mu_k))}{(2\pi^{D/2})|\Sigma_k|^{\frac{1}{2}}}$$

N = the number of descriptors (ie, 41,000)

$\mu_k =$ fitted mean of cluster k, $\Sigma_k =$ fitted diagonal covariance matrix of cluster k

D = dimensionality of the the means and descriptors (ie, 66), K = the number of clusters (ie, 512)

And we define each element j=1,...,D of $\frac{\partial L(X|\theta)}{\partial \mu_k}$ and $\frac{\partial L(X|\theta)}{\partial \Sigma_k}$ as [3]:

$$\Phi_{kj}^{(1)} = \frac{\partial L(X|\theta)}{\partial \mu_k}_j = \frac{1}{N\sqrt{w_k}} \sum_{n=1}^{N} q_{nk} \frac{(x_{n_j} - \mu_{k_j})}{\Sigma_{k_{j,j}}}$$

$$\Phi_{kj}^{(2)} = \frac{\partial L(X|\theta)}{\partial \Sigma_k}_j = \frac{1}{N\sqrt{2w_k}} \sum_{n=1}^{N} q_{nk} \left(\frac{(x_{n_j} - \mu_{k_j})^2}{\Sigma_{k_{j,j}}^2} - 1 \right)$$

where,

$$q_{nk} = \frac{w_k p(x_n|\theta_k)}{\sum_{j=1}^{K} w_j p(x_n|\theta_j)}$$

For image z, we define Fisher vector φ_z as:

$$\varphi_z = [\Phi_1^{(1)}, \Phi_1^{(2)}, ..., \Phi_K^{(1)}, \Phi_K^{(2)}]$$

We then store each Fisher vector for later use in the training step.

3.2.1.2. Training

Given the Fisher vector representations of the images in the training set, we seek to learn a matrix W and threshold b such that $b - (\varphi_i - \varphi_j)^T W^T W (\varphi_i - \varphi_j)$ is very positive when images i and j are of the same person and very negative when they are not. Simonyan, et al. choose to go about achieving this goal with the following steps:

1). imposing a margin be learnt:

$$y_{ij}(b - (\varphi_i - \varphi_j)^T W^T W (\varphi_i - \varphi_j)) > 1$$

where y_{ij} is +1 if i and j are of the same person and -1 otherwise

2): using a hinge-loss cost function:

$$argmin_{W,b} \sum_{i,j} max[1 - y_{ij}(b - (\varphi_i - \varphi_j)^T W^T W (\varphi_i - \varphi_j)),\ 0]$$

As this cost function is convex in b but not in W, training is done with stochastic subgradient descent where, for each iteration, two Fisher vectors are chosen, with an equal probability of the two images being of the same person or not. An update at iteration t is then only performed on W and b if the correct classification was made with insufficient margin or if the classification was incorrect:

$$W_{t+1} = W_t - \gamma_W y_{ij} W_t (\varphi_i - \varphi_j)(\varphi_i - \varphi_j)^T$$

$$b_{t+1} = b + \gamma_b y_{ij}$$

Note that after consultation with the authors, we chose $\gamma_W = 0.5$ and $\gamma_b = 10$. Finally, as the problem is non-convex, the authors choose to initialize W by creating a matrix of all the Fisher vectors, performing PCA on it, and then using the top 128 principal components as the rows of W.

3.2.1.3. Testing

Testing whether two images are of the same person simply involves using the learnt parameters from the preprocessing stage to construct Fisher vectors. The learnt W and b parameters are then used to see whether the distance is larger or smaller than b, and a decision is then made. We note that, once all parameters are unserialized, the most expensive step is the Fisher vector creation process: this tends to run for about a minute on an AWS m1.small machine. As matrix operations are highly optimized, reaching a decision for a pair of Fisher vector is extremely fast.

3.2.1.4. fisherfaces.com

fisherfaces.com is currently pointed at a single AWS m1.small machine with 1.7GB of RAM and one single-threaded CPU. It was built with the Flask web application micro-framework available for Python. Users are allowed to upload two reasonably-sized images to the website, which are then sent through a method that exactly implements the testing process mentioned in 3.2.1.2. Parameters are learnt on a much more

capable machine that has 4 cores and 16GB of RAM and are then copied remotely to the webserver so that they can be used for on-demand verification. The training, testing, and on-demand verification code is all in Python and can co-exist nicely with the website code. However, all of the preprocessing steps must be written in MATLAB as we heavily leverage `vlfeat`. So we have to make a new call to instantiate MATLAB every time a new request comes in, which is very problematic as the startup time for the program is relatively slow (ie, around 5-8 seconds). Although several packages exist that try to run MATLAB as a server, we found these applications are not currently maintained and very bug-ridden. This startup overhead, combined with the slow, single-threaded CPU and low RAM available means that requests typically take about a minute to process. We plan on migrating to a larger machine once we can get grant funds from Amazon for this. In the meantime, the website is not available to serve multiple requests, and so the website backend has been coded in such a way as to only allow one job to be processed at any one point in time.

3.2.2 Things that Didn't Work

3.2.2.1. Using a Denoising Autoencoder before Initializing W

One of our suspicions when we began this project was that the introduction of a layer that captured non-linear similarities between Fisher vectors and which occurred right before the initialization of W might result in better generalization performance. The intuition here was training was very susceptible to the W initialization step, and performing PCA is just equivalent to performing a linear transformation of the training data. So, it stood to reason that a non-linear transformation would capture similarities that were previously overlooked. To test this idea out, we tried to train a single-layer denoising autoencoder (DA) from the `Theano` Python library with the Fisher vectors as the input data.

Briefly, a DA is similar to a feed-forward autoencoder in that it is given a set of training instances, each of size n, and tries to reconstruct a particular training instance using weights learnt for its k hidden units, where k < n. That is, for each training instance x, we have from [4]:

$$y = \sigma(Wx + b)$$

where y is the compressed representation of x, $\sigma(\cdot)$ is the sigmoid function, W is the weight matrix to learn, and b is the learnt bias.

The graphical model then tries to reconstruct the training instance from the compressed representation:

$$x' = \sigma(W'y + b')$$

where W' and b' are a new set of learnt parameters.

It then tries to minimize the reconstruction error $\sum_i ||x^{(i)} - x'^{(i)}||_2$ through back propagation. DAs also add Gaussian noise to the input data in order to learn a more robust mapping from the input to the reconstruction. Unfortunately, we found that training DAs takes an extremely long time - over 24 hours for a particular training/test split using very powerful machines. To put this in perspective, we can perform the entire 3.2.1.2 training pipeline on the same machines in about 5 hours. Also, we could not come up with a choice of parameters that resulted in better verification accuracy than when the DA step was completely left out. We discuss these results in greater detail in Section 4.

3.2.2.2. Using a Feedforward Neural Net for Training

More ambitiously, we were interested in seeing how well we could approximate the current pipeline by using a simple, multi-layered neural net. That is, we removed the training and testing steps outlined in 3.2.1.2 and 3.2.1.3 and instead, used a trained neural net to classify pairs of images. The visible layer was $\varphi_i - \varphi_j$ for two images i and j. This first layer was of dimensionality 67,584, the first hidden unit layer was initially around dimensionality 1024, the second layer 128, and the final layer had 2 units that output the likelihood of this pair originating from the same person or not. For simplicity and runtime consideration, we did not pre-train the layers: if initial results looked promising, we would then implement this additional (in terms of runtime) step.

We tried each node having its own W and b parameters as well as having each node in a layer share them. Unfortunately, we found this method was not competitive with the current pipeline in terms of memory, runtime, or verification performance. Moreover, increasing the dimensionality of the hidden layers meant storing larger and larger dense weight matrices, and using more training data also meant storing the differences of many Fisher vectors in memory. We attempted a second layer of 9024 and the training time required was a week and a day. Although it achieved better performance, the training time was a deal breaker. Although we noted that a higher dimension second layer would probably result in huge gains, we eventually could not allocate additional memory resources to complete training of our neural net no matter which framework we used: PyBrain, Theano, FANN (Fast Artificial Neural Network), or Matlab.

3.2.2.3. Learning a Separating Hyperplane Rather than a Projection Matrix

We were interested in seeing what performance we were capable of achieving using a simple separating hyperplane rather than a projection matrix. That is, instead of using the training and test steps outlined in 3.2.1.2 and 3.2.1.3, we trained a L2-regularized SVM with a linear kernel using `liblinear` [5].Formally, and as provided in the `liblinear` documentation, this classifier solves the following primal problem:

$$min_w w^T w + C \sum_i max[1 - y_i w^T x_i,\ 0]$$

Where: $x_i = \varphi_j - \varphi_k$ for two different images j and k

$y_i = 1$ if j and k come from the same person and -1 otherwise.

As with the W learning in 3.2.1.2, the labels of the training set are evenly split between same and different person pairs. We chose 500,000 same pairs and 500,000 different pairs giving us 1 million Fisher vector differences to use for training samples. However, after training the SVM with many different regularization parameters, we never reached a performance over 50%. It is clear that, because it has an order of magnitude fewer parameters to learn, the SVM's performance would be a lower bound on the training performance compared to learning a projection matrix, and it would almost assuredly be a lower bound on validation performance as well. Thusly, we attempted to perform SVD on the data matrix to use only the top 500 left singular vectors, $D_{sub}=U_{sub} * \mathbf{\Sigma}_{sub}$ in which D_{sub} is 1 million by 500. Indeed, despite our attempts to tune the SVM's hyperparameters and form subspaces for our data matrix, we could not achieve results better than 50% test accuracy.

3.2.2.4. Adding Additional Features During Fisher Vector Generation

Another suspicion of ours at the beginning of the project was that the pipeline presented by Simonyan, et al. didn't capture a host a features that might prove useful for verification performance. For instance, we imagined that counting the number of skin pixels within the image area described by a specific SIFT descriptor might be very valuable information to use, as might information about the background in certain images. We primarily explored the inclusion of the fraction of skin pixels in the post-PCA SIFT descriptors. That is, for a given image, we ran an implementation of the "gray world algorithm" that has been optimized to detect skin pixels [6]. After the SIFT descriptors for a given image have been run through PCA, we then add a feature that contains the percentage of skin pixels that are included in the region of a particular SIFT descriptor, as determined by its corresponding keypoint. Unfortunately, we found that the inclusion of these features slightly hurt validation performance. It seems like the Viola Jones detector is already capturing skin information, and so the explicit inclusion of pixel data in the descriptors hurt generalization performance. Rather than explore this further, we were more interested in dealing with the challenges of successfully training the neural nets on our dataset.

4. Results

4.1 Evaluation Criteria

As mentioned in Section 3, we used the 'Unrestricted' version of the LFW dataset in which the 13,000+ images are partitioned into 10 splits. A training procedure would then use the images in 9 of the 10 splits to generate as many training samples as required, and the last split would be used for evaluation. Specifically, the LFW creators provide 600 specific image pairs to be evaluated per split, where 300 of these are of image pairs of the same person and 300 are of different people. Our key metrics to judge the performance of our pipeline was the Accuracy, Precision and Recall obtained on these 600 samples per evaluation split. We defined a False Positive as our method deciding two images are from the same person when they were not, and a False Negative was the opposite event occurring.

4.2 Summary of Findings

The following table provides the results of a subset of our experiments:

Experiment	Accuracy	Precision	Recall
Mahalanobis Metric - 26k – Non-DF - Non-Root	76.7	68.7	81.7
Mahalanobis Metric - 26k – Non-DF - RootSift	80.7	76.0	83.8
Mahalanobis Metric - 26k – DF - RootSift	81.5	73.0	88.0
Mahalanobis Metric - 26k – DF - Non-Root	82.5	75.0	88.2
Mahalanobis Metric - 41k – DF - Non-Root	83.8	78.7	87.7
Mahalanobis Metric - 41k – Non-DF - RootSift - VJ	**86.0**	**77.3**	**93.5**
Mahalanobis Metric - 41k – Non-DF - Non-Root - VJ	**86.2**	**78.3**	**92.9**
Mahalanobis Metric - 41k – Non-DF - Non-Root - VJ - SKIN	**85.7**	**76.7**	**93.5**
Linear SVM – C = 1 - 41k – DF - Non-Root	50.0	50.0	50.1
Neural Network [67584,1024,128,2] - 41k – DF - Non-Root	73.5	74.2	73.1
Neural Network [67584,9024,128,2] - 41k – DF - Non-Root	78.5	79.2	78.1

We now describe the 'Experiment' categorization:

- If the experiment learned a Mahalanobis Metric, then the basic categorization is 'Mahalanobis Metric - # of SIFT descriptors generated per image - Deep-Funneled or Normal LFW dataset - RootSift used (or not)'. Additional fields are 'VJ', which means the Viola Jones detector was used, 'SKIN', which means skin pixels were used as features.
- If the experiment learned a separating hyperplane, then the categorization is 'Linear SVM - C hyperparameter value - # of SIFT descriptors generated per image - DF or Normal LFW dataset - RootSift used (or not)'.
- Finally, if the experiment concerned is a FNN, then it's of the form 'Neural Network [visible units, size of hidden layer 1, size of hidden layer 2, size of output layer] - # of SIFT descriptors generated per image - DF or Normal LFW dataset - RootSift used (or not)'.

Where we note that the 'Deep Funneled' LFW dataset [17] is a slightly modified version of the normal LFW dataset where a canonical template of the images has been learnt via a deep network and each image has been transformed to fit that template. We achieve the best performance when learning a Mahalanobis

Distance on the LFW dataset with 41,000 generated SIFT descriptors per image and also using the Viola Jones landmark detector. Surprisingly, the same experiment, when repeated on the Deep-Funneled dataset, did slightly worse: this might just be due to choosing unlucky descriptors to compute the GMM parameters. As we discussed previously, the addition of skin parameters causes slightly worse generalization performance, as does the use of RootSift descriptors. Finally, as we noted, experiments that used SVMs and Neural Nets generate performance numbers that are not competitive. We note that our optimal results do not achieve the 93% accuracy reported by Simonyan, et al. We attribute this to a variety of factors, including the fact that we only use 1M SIFT descriptors to perform GMM fitting as well as, most importantly, the fact that the authors use outside training data to perform face alignment to aid in training.

5. Conclusions

Creating a successful facial verification algorithm is extremely challenging: image representations are very high dimensional, and one must be sure the algorithm is expressive enough to capture important similarities while preventing overfitting. Similarly, training neural nets is an extremely challenging process that requires very high-resource machines running for very long periods of time. We successfully implemented Simonyan ,et al's work as a baseline, and we are still very confident in the ability of neural nets to increase verification performance on the LFW dataset provided enough time and computation power. We hope to explore this further in subsequent work.

6. References

[0] Simonyan, Karen, et al. *Fisher Vector Faces in the Wild*. <http://www.robots.ox.ac.uk/~vedaldi/assets/pubs/simonyan13fisher.pdf>.

[1] Arandjelovic, Relja, and Andrew Zisserman. *Three things everyone should know to improve object retrieval*. <http://www.robots.ox.ac.uk/~vgg/publications/2012/Arandjelovic12/arandjelovic12.pdf>

[2] Garg, Vinay, Siddhartha Chandra, and C. V. Jawahar. *Sparse Discriminative Fisher Vectors in Visual Classification*. <http://delivery.acm.org/10.1145/2430000/2425388/a55-garg.pdf>.

[3] *Fisher vector fundamentals*. VLFeat.org. <http://www.vlfeat.org/api/fisher-fundamentals.html>.

[4] *Denoising Autoencoders (dA)*. deeplearning.net. <http://deeplearning.net/tutorial/dA.html>.

[5] Fan, Rong-En, et al. "LIBLINEAR: A Library for Large Linear Classification." *Journal of Machine Learning Research* 9 (2008): 1871-74. <http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf>.

[6] Jain, Guarav. *Skin Detection*. MATLAB Central. <http://www.mathworks.com/matlabcentral/fileexchange/28565-skin-detection>.

[7] Sirovich, Lawrence, and Michael Kirby. "Low-dimensional procedure for the characterization of human faces." *JOSA A* 4.3 (1987): 519-524.

[8] Belhumeur, Peter N., João P. Hespanha, and David Kriegman. "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.7 (1997): 711-720.

[9] Yang, Ming-Hsuan, Narendra Ahuja, and David Kriegman. "Face recognition using kernel eigenfaces." *Image processing, 2000. proceedings. 2000 international conference on*. Vol. 1. IEEE, 2000.

[10] Chang, Hong, et al. "Spectral range selection for face recognition under various illuminations." *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*. IEEE, 2008.

[11] "Results." *LFW : Labeled Faces in the Wild*. University of Massachusetts, n.d. Web. 19 Mar. 2014.

[12] Zhu, Xiangxin, and Deva Ramanan. "Face detection, pose estimation, and landmark localization in the wild." *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012.

[13] Fan, Haoqiang, et al. "Learning Deep Face Representation." *arXiv preprint arXiv:1403.2802* (2014).

[14] Pedregosa, Fabian, et al. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research 12. *Scikit-learn: Machine Learning in Python*. <http://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>.

[15] Bergstra, J, et al. "Theano: A CPU and GPU Math Expression Compiler." *Proceedings of the Python for Scientific Computing Conference* (2010). <http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf>.

[16] Huang, Gary, et al. "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments." <http://vis-www.cs.umass.edu/lfw/lfw.pdf>.

[17] *Advances in Neural Information Processing Systems (NIPS)* (2012). *Learning to Align from Scratch*. Web. 19 Mar. 2014. <http://vis-www.cs.umass.edu/papers/nips2012_deep_congealing.pdf>.

[18] Guillaumin, Matthieu, Jakob Verbeek, and Cordelia Schmid. "Is that you? Metric learning approaches for face identification." *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009.

[19] V. Blanz and T. Vetter. Face recognition based on fitting a 3d morphable model. IEEE TPAMI, 2003.

[20] Vedaldi, Andrea. *VLFeat. VLFeat.org*. <http://www.vlfeat.org/>.