Evaluation of Eigenfaces System

Luvai Hassanali 100923952

Dec 16, 2018

Abstract

Facial recognition systems, which are appearance based, classify images based on factors such as pose, expression, and lighting. Using this method with large datasets is not efficient as there are too many possible appearance combinations, resulting in an excess time required to classify. The solution to this is to compress the faces into a low-dimensional subspace, which captures key appearance characteristics of the entire dataset. These "eigenfaces" of a dataset are the "eigenvectors of the covariance matrix of the probability distribution of vector space of human faces" [5]. These vectors are the standardized faces of the dataset and can be used in specific combinations to reconstruct another human face as a prediction for classification purposes. The eigenvectors can be efficiently computed using Principal Component Analysis (PCA) on the matrix containing the image dataset.

1 Introduction

The purpose of my project is to evaluate the eigenfaces system discussed during Lecture 19, concerning the topic of PCA. The recognition system is implemented using the process described in the lecture using the Labelled Faces in the Wild dataset. To evaluate how well this system performs, I compare it to a multilayer perceptron (MLP) and convolutional neural network (CNN), which classifies images using the raw data only. The particular factors observed are accuracy achieved and time elapsed. The primary goal of this paper is to determine the advantages and disadvantages of using eigenfaces in recognition systems.

2 Data

The data used in this project is the Labelled Faces in the Wild (LFW) dataset, a collection of face photographs designed for the task of facial recognition [1]. The dataset contains multiple images of subjects all with varying facial expressions, posture, and lighting, each with an accompanying label to identify the target. I have restricted the dataset to a minimum of 70 greyscale faces per subject, which results in a dataset of size 1288 images. Additionally, to ensure the least amount of unwanted features from the images, each face in the dataset is cropped and centered to 50x50 pixels. The dataset for each experiment is divided with a ratio of 0.25 resulting in a training set of 966 images and test set of 322.

3 Libraries

TensorFlow, Scikit-Learn, NumPy, and matplotlib are the main libraries used in this project. Scikit-Learn library has a function to load the LFW dataset and another to calculate PCA on the dataset. The Scikit-Learn library is also used to implement a secondary version of the eigenface recognition system. TensorFlow is used to implement the neural networks to compare against the eigenface recognition system. NumPy is a library which deals with the matrices that store our image data and can compute high-level mathematical functions on these matrices. The matplotlib library is used to plot data into a graphical display.

4 Implementation

The first model constructed is a basic eigenface recognition system following the process outlined in the lecture. To generate the eigenfaces of the LFW training set, the images are normalized, and the eigenvectors of the covariance matrix are extracted using Scikit-Learn decomposition function, PCA [2]. These eigenvectors represent the principal components of the training set and are ordered from most variation to least. A subset of these eigenvectors can be used to represent the images of the dataset [5]. As described in the lecture, these reduced combination eigenfaces when weighted properly can be summed together to create a prediction of a human face.

For classification, the test images and their weights are computed using the eigenfaces. The norm of all eigenfaces weights and weights of an unknown image is calculated to compute the closest distance to other training faces in the eigenspace. The process outlined in the lecture includes a step to test to see if the image is a face by implementing a threshold distance value, but since our dataset contains only faces, there were no significant threshold fluctuations. It was not necessary to implement this step. The face from the training set which the test image has the minimum distance from is the one it has the strongest correlation with, so it is the prediction of the eigenface system. To ensure that this system is working correctly, I created a model whose total functionality relies on the Scikit-Learn library, to compare it against.

The neural networks used to implement facial recognition system to compare the eigenface recognition system with is MLP and CNN. The CNN implemented first only had one channel to accommodate the greyscale pictures from our dataset. It became evident quickly that the CNN was not performing well enough, so I created a separate model with three channels for the RGB version of pictures from the LFW dataset. Since the neural network models are being trained using raw data, preliminary tests were performed to determine whether normalizing data helped classification. The results of these experiments showed that normalizing the data for the neural networks is beneficial as there is significant accuracy loss if not performed. The third model shown below is a neural network, which is implemented entirely by the Scikit-Learn library. I will discuss this in more detail later on.

5 Models

1. Basic implementation of eigenface system:
    - Generate covariance matrix using PCA
    - Extract eigenfaces from the covariance matrix
    - Compute normalized training set and weights
    - Given a new image, compute weights, project into eigenspace
    - Closest distance = prediction of system
2. Scikit-Learn implementation of eigenface system:
    - Extract eigenfaces using PCA
    - Project data onto the eigenspace
    - K-means classifier used to determine the closest neighbor = prediction of system
3. Scikit-Learn implementation of MLP:
    - Extract eigenfaces using PCA
    - Project data onto the eigenspace
    - MLP implemented using the Scikit-Learn function
    - Learning rate = 0.001, Activation ReLU, Adam Optimizer, 1022 hidden layers
4. TensorFlow implementation of MLP using raw data:
    - 2 hidden layer network using Leaky-ReLU for activation at each layer

- Learning rate = 0.0001, Adam Optimizer, softmax cross entropy with logits

5. TensorFlow implementation of CNN using greyscale raw data:
   - 1 layer, input data is greyscale
   - ReLU activation, Adam Optimizer, dropout applied to max pool and fully-connected layer
   - MaxPool kernel size = [1, 3, 3, 1]
   - Input → Conv Layer → Pool → Dropout → Reshape → Fully Connected → Dropout → Output
6. TensorFlow implementation of CNN using RGB raw data:
   - 1 layer, input data is RGB color
   - ReLU activation, Adam Optimizer, dropout applied to max pool and fully-connected layer
   - MaxPool kernel size = [1, 3, 3, 1]
   - Input → Conv Layer → Pool → Dropout → Reshape → Fully Connected → Dropout → Output

6 Experimentation

I began this project with the intention of calculating PCA of the training images manually. This process begins by normalizing the faces in the training set using a mean face, which is average of all images into one. The covariance matrix is calculated using the matrix of training set multiplied by its transpose, $A * A^T$ [5]. The eigenvalues are calculated to get the eigenvectors, which represent the eigenfaces of the training set. As mentioned in the implementation section, test images to be recognized are normalized and projected into eigenspace. The distance to the closest neighbor is the prediction made and since the LFW dataset has labels, the pair can be classified precisely.

In my experimentation of the recognition system, using manually calculated eigenfaces, the accuracy achieved from this model was not more than 45%. This was regardless of the amount of eigenvectors chosen to represent the dataset. Using the PCA function from the Scikit-Learn library, the predictions made using these eigenfaces had better accuracy. As stated in the lecture, typically only 20-30 eigenvectors are sufficient to represent the dataset, so this is where my experimentation started [5]. Using 20 principal components, the accuracy achieved for the system is around 45% and using 30, results in a slight increase of about 1-2 percent. To further investigate, I increased the number of principal components and observed that 50 is the minimum amount required for maximum accuracy of around 55%. Anything higher does not yield more accuracy, and any amount lower hinders it. Therefore, for the remaining tests I use 50 principal components.

Number of Principal Components	Time Elapsed (s)	Accuracy Achieved (%)
20	0.1	45
30	0.1	47
50	0.125	54
100	0.57	54

The first 12 eigenfaces or principal components of the training set calculated using Scikit-Learn PCA function are displayed in Figure 1 below. The eigenface recognition system achieved a classification score of 55% and this experiment is the base results we can compare other models with. Additionally, the time calculations in the table are the elapsed time of classifying the 322 images in the test set. The time measurements are made using Python's time.process_time function which returns the value of the system and user CPU time elapsed for classification. It does not include the amount of time required to compute PCA. Figure 2 below represents five test cases from this recognition model. The image on the left represents the image tested, the middle one is the normalized test image, and the one to the right is the predicted image determined by the recognition model.

To compare the performance of the manually created recognition system, another model was created using Scikit-Learn library. The library performed the projection of data onto the eigenspace and classified

the images using the Scikit-Learn KNeighboursClassifier [3]. This classifier uses the K-Nearest-Neighbours (KNN) algorithm to classify the test images. Using 50 principal components, the entirely dependent Scikit-Learn recognition system scored an accuracy of around 60% accuracy in the range of approximately 0.04-0.09 seconds. Additionally, with the features of the Scikit-Learn classifier, we can visualize the faces which get projected onto the eigenspace, as shown in Figure 3. Since the accuracy difference between the manually implemented recognition system and the fully-dependent one is 5%, I do not believe there are any significant mistakes made in the design of the manually implemented one.

While researching for this project, I discovered Scikit-Learn has functions that implement neural networks, one of which include a multilayer perceptron classifier. I thought it would be valuable to collect data from a neural network, which trains using the PCA reduced data, to compare against the neural network which uses raw data only. The MLPClassifier [4] uses ReLU activation, AdamOptimizer, the learning rate of 0.001 for training. It also contains many parameters to set (Figure 4), so I tried to keep it as simple as possible, leaving as many as parameters to default settings. I did specify two parameters for the function. The first parameter is a variable for hidden_layer_sizes, which is defined in documentation as "tuple, length = n_layers − 2, default(100,), where the ith element represents the number of neurons in the ith layer." I used hidden_layer_sizes = (1024,) and the second parameter is a variable for batch size = 32. To validate the choice of batch size, different tests are performed. As an example, setting batch size to 64 (80%) scored slightly lower than 32 (82%); while 16, and 8 produced 81% accuracy each. Additionally, for the experiments of hidden layer parameter, 1024 was the middle ground where accuracy achieved was at its maximum. With a batch size of 32, this system can achieve the mean accuracy of up to 82% in around 60 epochs, which takes about 6 seconds.

For the TensorFlow implementation of MLP, that trains on raw data only, I construct my network of 2 hidden layers with 500 and 100 neurons in each layer respectively. I start with a batch size of 32, AdamOptimizer, ReLU activation, and learning rate of 0.001, similar to the MLPClassifier. This setup for the model achieves an average accuracy of around 75% after 100 epochs. To attempt to improve the model, I changed the activation to Leaky_ReLU and decreased learning rate to 0.0001, which resulted in a slight but not significant improvement. Numerous experiments performed to determine what distribution of weights for the layers would be optimal. The results of my tests indicated that the configuration of 500 and 100 hidden neurons achieved the highest accuracy possible. Overall, the model was able to predict consistently with a mean accuracy of 76% after 100 epochs, but it took around 55 seconds. We can see the benefit of training the network on PCA reduced data in the time elapsed of classification between the Scikit-Learn MLP network and my TensorFlow model. Additionally, we can see the advantage of using a neural network over the eigenface recognition system as the neural network achieves a better classification rate.

To be able to test the performance of the recognition system against another neural network, I created a convolutional neural network of one layer using TensorFlow. I had set up to the network to classify the same data used in all other examples, but noticed that my accuracy would not go over 60%. To follow up, I implemented a CNN model, which took the same images of the dataset, but in RGB color. Since the first network is not using color channels, it is not detecting features which distinctly identify different people, just the face itself. We can get a visualization of this by comparing the top 4 patches of the CNN trained with greyscale images vs. the CNN trained with RGB images, as shown in Figure 5. The added filters allowed the second of the CNN model to reach around 75-82% accuracy.

Different experiments were performed to determine optimal kernel size for max pooling layer and the weights for the layers. The model attains highest correct prediction rates when the kernel size is [1,3,3,1] and weights coefficients of [1024,7] for output layer, [25*25*50, 1024] for hidden layer, and [3, 3, 3, 50] for initial convolution layer (creates 50 3x3x3 filters). It is evident that both CNNs can achieve better accuracy than the eigenface recognition system; however, the time required is much longer. As an example, the CNN model for greyscale images reached mean accuracy of 54% in 8 minutes, while the CNN model for colored images attained 81% in close to 9 minutes, both using 20 epochs.

One of my original ideas for this project was to determine whether color affected the classification rate of a CNN. I wanted to train a CNN using black and white images and test accuracy on colored ones and vice versa, to see how accuracy was affected. From the experiments performed using the two different CNN networks it seems evident to me that this would not be possible. The lack of filters for the first representation prevents the network from learning the features that distinguish identity.

Model Type	Avg. Time Elapsed (s)	Avg. Accuracy Achieved (%)
Eigenface System	0.125	55
Scikit-Learn Eigenface System	0.06	60
TensorFlow MLP	55	75
TensorFlow CNN (greyscale input)	510	50
TensorFlow CNN (RGB input)	535	80
Scikit-Learn MLP	6	80

7 Conclusion

Eigenface recognition systems do not compare to a neural network trained on raw data. The chart above provides a summary of the results achieved throughout experimentation. With the eigenface recognition system, the maximum accuracy achieved was around 55-60%, while networks trained on raw data had accuracy results upwards of 75%. Although there is a significant difference in time elapsed between the two types of classification models, I believe the accuracy achieved from the neural networks is worth the extra time required. Additionally, the experiment using Scikit-Learn MLPClassifier, shows that a neural network trained on reduced data scores the same as neural network trained on raw data, but in much less time. This fact can be used to design new superior systems or to implement improvements to existing facial recognition systems which are time-consuming.

Figures

Figure 1: The first twelve eigenfaces computed using PCA (components=50)

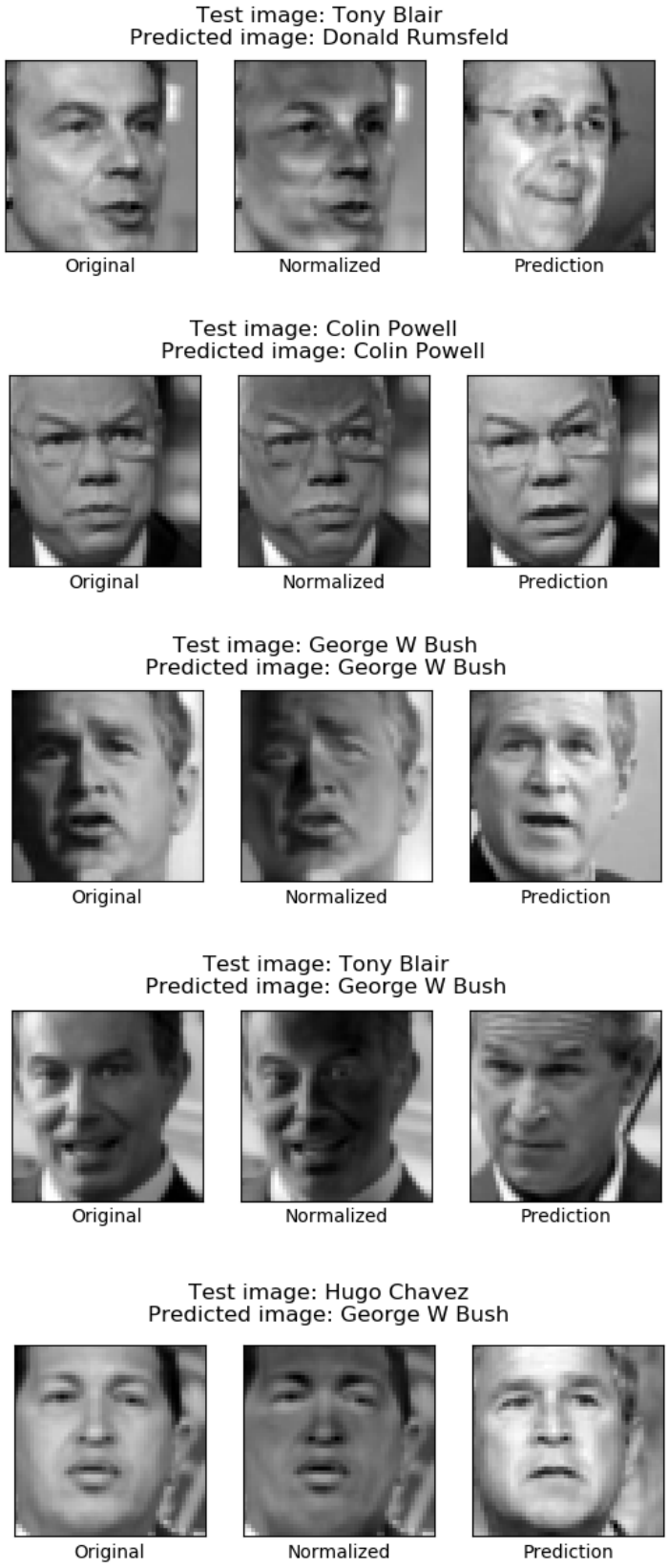

Figure 2: Basic eigenface system prediction examples

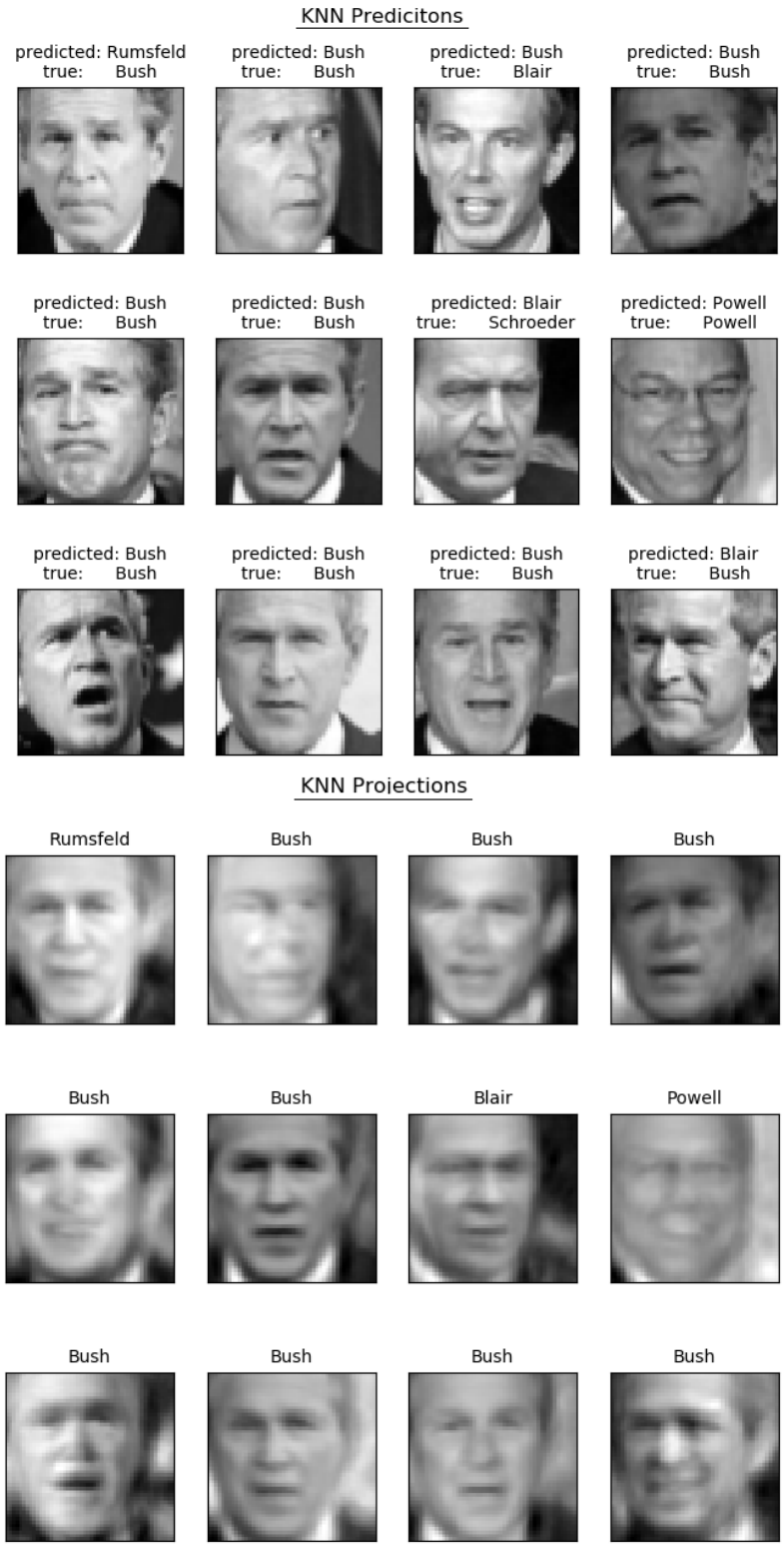

Figure 3: Scikit-Learn eigenface system predictions and projections using KNN algorithm

class `sklearn.neural_network.` **`MLPClassifier`** (*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10*) ¶ [source]

Figure 4: Scikit-Learn MLP Classifier function declaration

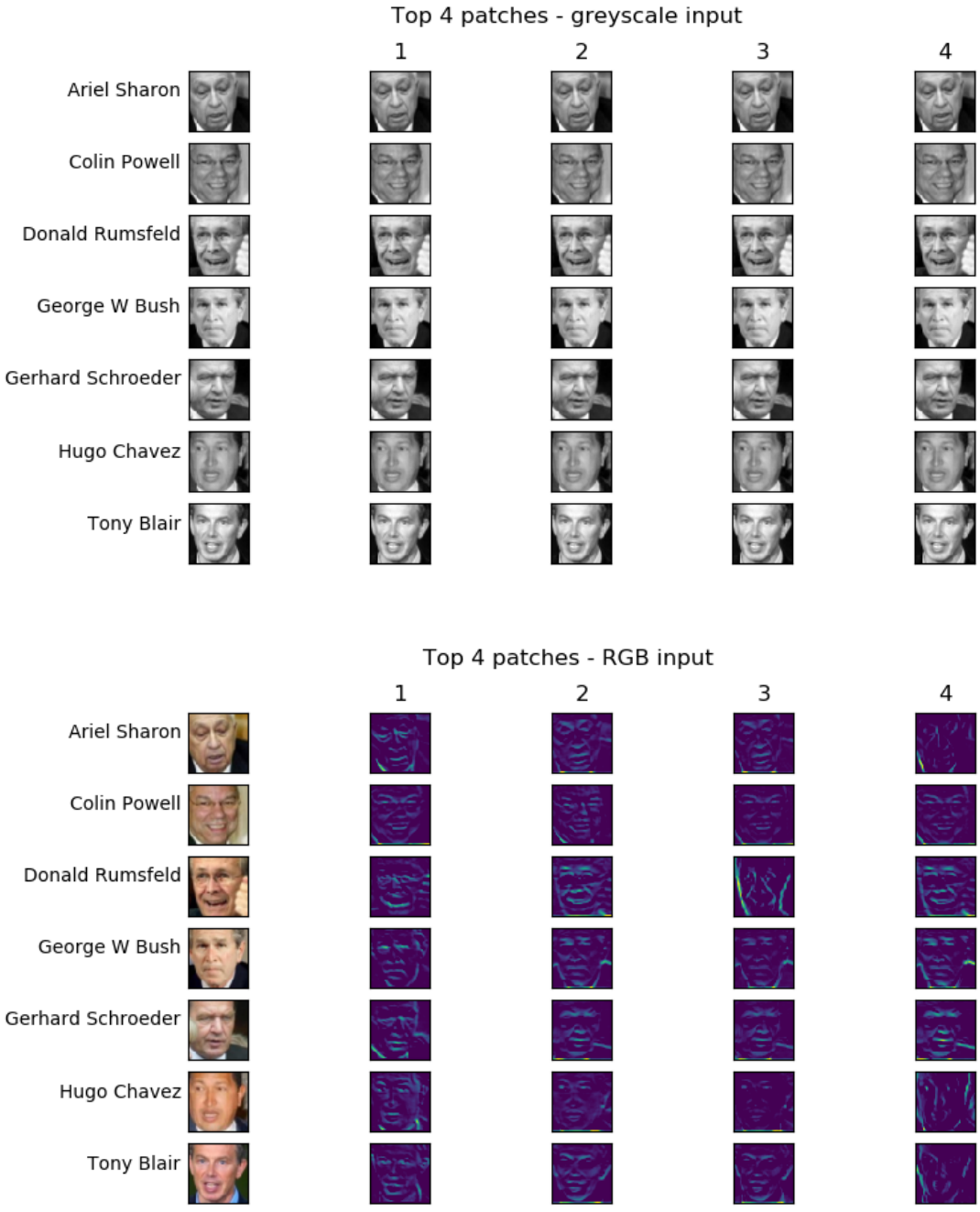

Figure 5: Top 4 patches displayed for both variation of CNNs after 20 epochs of training

References

[1] Labeled Faces in the Wild Home., vis-www.cs.umass.edu/lfw/.

[2] "Sklearn.decomposition.PCA. 1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation, scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[3] "Sklearn.neighbors.KNeighborsClassifier. 1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation, scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[4] "Sklearn.neural_network.MLPClassifier. 1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation, scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[5] White, Dr. A. "Principal Component Analysis (PCA)", 2018. Lecture 19: PowerPoint presentation, https://sikaman.dyndns.org:8443/WebSite/rest/site/courses/4107/index.html