CSCI 1430 Final Project Report:
Methods of Facial Recognition

Laura Dayton, Dante Rousseve, Neil Sehgal, Sindura Sriram.
Brown University
May 5 2020

1. Introduction

Facial Recognition is a general class of problems that includes both face identification and face verification. Face verification involves confirming or rejecting the presence of a particular identity within an image. Face identification, on the other hand, involves identifying a person based on an image of their face. The image in question is compared against a database of registered individuals, and the identity of the person within the image is determined.[1] In this project, we focus on face identification. Several factors which vary greatly among different faces and images make this task difficult: angles, lighting, and background. In this project, we implement four different face identification methods and comparatively evaluate them using the same testing dataset, AT&T Olivetti Faces. Olivetti Faces consists of 400 images corresponding to 40 distinct people, with 10 images per person. The face identification methods we implement are based off of the facial landmarks, eigenfaces, fisherfaces, and VGG-Face models. We chose to tackle this challenge as Facial Identification is not 100% solved yet, with various methods in use around the world and remaining concerns for misidentification.

2. Related Work

While facial recognition may seem like a relatively recent phenomenon, its origins date back more than a century. In the late 1800s, after the invention of photography, police forces maintained disorganized photographic collections of suspects and convicts. However, these collections lacked an ordered indexing system. In 1879, French police officer Alphonse Bertillon invented an organizational scheme based on a series of manual, standard measurements of the unique features of suspects, such as head length. The system was first adopted in the United States in 1887 and gained widespread acceptance until the advent of fingerprint identification. In the late 1960s, Woodrow Bledsoe developed the first semi-automated Facial Recognition system, which computed distances and ratios of facial focal points that were manually located by an administrator. Further accuracy improvements continued in the next decades, but all were set back by the slow initial process of manually locating facial features. In 1988, a team of MIT and Brown researchers developed a system known as Eigenfaces. The Eigenface research demonstrated that there were fewer than 100 hundred features that, if extracted, could uniquely code any face. In 1991, two MIT researchers built off of the Eigenface system to develop the first entirely automated real-time Facial Recognition system. While the system was limited in several ways, it served to prove the feasibility of automatic Facial Recognition. In the following decade, the Defense Advanced Research Projects Agency heavily invested in creating large test databases to further the commercial Facial Recognition market. By 2002, commercial systems outperformed human participants in recognizing faces.[2]

3. Methods

Facial Landmarks

Bledsoe's semi-automated landmarks approach was the inspiration for our first method, in which we aimed to train a facial landmarks classifier and then use those predicted landmarks across images to identify similar subjects. First, a facial landmarks classifier was trained using a dataset from the University of Montreal, which consists of 7049 images. Each image contains the (x,y) real-valued coordinates in pixel space for 15 key points. We retrieved the dataset from Kaggle's Facial Keypoints Detection competition. We trained a convnet on this dataset and achieve an accuracy of .8483 on a held out validation set.

We then predict landmarks on every image in the Olivetti Faces dataset, and use these predicted landmarks to train a KNN with K=1 using varying train-test split sizes to predict identity. We find that train test splits with different random_states give varying accuracies so the average accuracy across 10 different train test splits were taken.

The following is the general structure we use for facial recognition across all four methods:

```
# Convert olivetti faces to a new representation
olivetti_vector =
Some_Model.predict(olivetti_faces)
# number of images per person to train on
percent_train_list = [1,2,3,4,5,6,7,8,9]
accuracy = np.zeros((len(percent_train_list)+1))
# for 10 random states
for r in range(10):
 # number of images to train on
 for train_amount in percent_train_list:
   X_train, X_test, y_train, y_test =
   train_test_split(olivetti_vector \
   , id_labels, test_size=(1-train_amount/10),
   random_state=r, stratify=id_labels)
   # single closest euclidean dist
   knn = KNeighborsClassifier(n_neighbors=1)
   knn.fit(X_train, y_train)
   acc = knn.score(X_test, y_test)
   accuracy[train_amount] += acc
# Take average
accuracy = accuracy/10
```

While the images in Olivetti Faces are somewhat varied (images are taken at different time with varied lighting and facial expression), the images are on the whole strongly similar with azimuth and elevation held mostly constant. We hypothesized that due to the wide similarity between images of a single person, the landmark coordinates would be somewhat similar across images of a single person. Had there been more variation of images within a single class, this method would most likely fail.

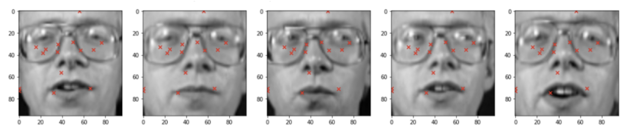


Figure 1. Predicted landmarks across 5 images of same class

Eigenfaces

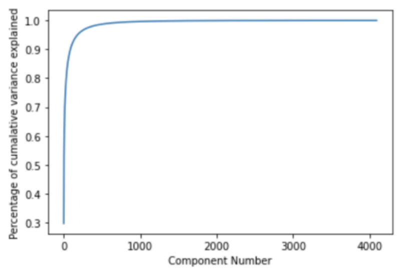
The eigenface system is based off of using Principal Component Analysis (PCA). PCA is a statistical algorithm which reduces the dimensionality of a dataset while retaining most of the variation. [3] PCA identifies the directions, known as eigenvectors or principal components, along which the data varies the most. In the eigenface system, every image in the dataset can be represented as a linear combination of these eigenvectors known as eigenfaces. Face identification is done by projecting a new image onto the subspace spanned by the eigenfaces and then is identified by matching it to the closest training image by euclidean distance. [4, 5]

We use the Labeled Faces in the Wild dataset to train PCA and determine the eigenvectors of the human face. We specifically use a variant of the dataset, LFWcrop, in which only the center portion of the image (the face) is kept and the background is omitted. This dataset consists of 13233 images. We first determine the optimal number of components to use by applying PCA with the maximum number of components.

We implement PCA using SVD using the following approach learned in CSCI1951a:

```
def pca_using_svd(data, num_components):
  # calculate mean
  mean_face = np.mean(data, axis=0)
  # center data
  data = data - mean_face
  # SVD on centered data
  U, S, V = np.linalg.svd(data)
  red_u = U[:,:num_components]
  red_s = S[:num_components]
  red_v = V[:num_components]
  return data, red_v, red_s, mean_face
```

When using PCA with the maximum number of components (equal to the height * width of the image as the number of pixels is the most the image can be represented as) we are forced to rely on scikit-learn's implementation of PCA. We had hoped to use our own implementation, but even on Google Colab with a high RAM runtime, our implementations are not optimized enough to run without crashing. Thus, we use scikit-learn's PCA initially when running PCA with the maximum number of components, and our own implementation in further steps when only needing a small number of components. After applying PCA to the LFW_crop dataset and plotting the eigenvalues we get the following graph:



```
First 5 components explain 0.5882175698363753% variance in the data
First 50 components explain 0.8493276285298338% variance in the data
First 100 components explain 0.9121692351787823% variance in the data
First 250 components explain 0.9662605206060751% variance in the data
```

We can see that just 250 components explains 96.6% of the variance in the dataset. Each image in LFWCrop is 64,64 meaning rather than representing the image using 4,096 (64*64) components, we have found a reduced way to explain most of the image in just 250 components.

For example, in Figure 2 we recreate a face using just 250 components. We also try recreating faces not in the LFWCrop dataset that the PCA was fit to. When used on images not in the original dataset, it does not perform as well (Figure 3). We find that the following 15 eigenfaces explain 71.5% of the variation within LFWcrop (Figure 4).

We then project the Olivetti Faces down to the subspace defined by the LFWCrop's eigenfaces and perform facial recognition using KNN in the same manner as defined by the facial landmarks setup. One difference is that we vary

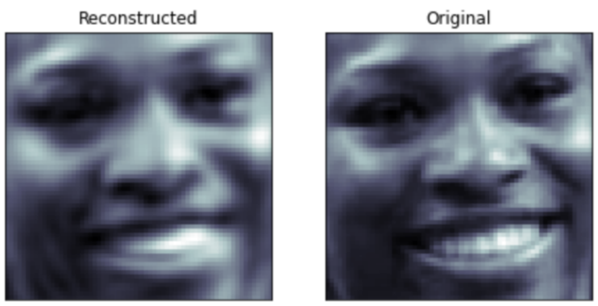


Figure 2. Reconstruction using 250 components

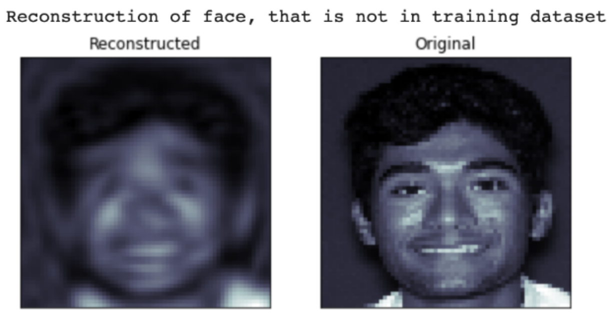


Figure 3. Reconstructing a Face outside of the Training Data

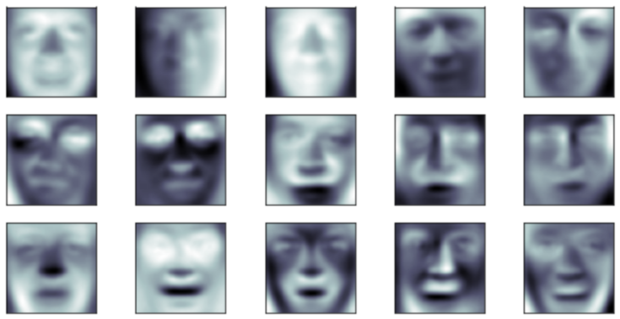


Figure 4. The 15 Eigenfaces

the number of components used to train the KNN to see how few we can use to still achieve a high accuracy.

Fisherfaces

The Fisherfaces system, similar to the Eigenfaces system, aims to describe images by projecting them down to a lower dimensional space. The Fisherfaces system alters the traditional method by representing the images in a subspace that maximizes the difference between features, rather than attempting to cover the most variance with the features created. This means that the vectors that result from the Fisherfaces method are not meant to represent the key parts of the images, but rather to represent the features that represent the greatest difference between images in the data set. To simplify, the system tries to have similar classes cluster closely together and different classes be separate from one another. We do this by looking at the “scatter-within”, the scatter distance between images of the same class, and the “scatter-between”, the scatter distance between images of different classes, and maximizing the ratio of “scatter-between” to “scatter-within”. [6]

So in Fisherfaces, we use PCA to get the eigenvectors for the dataset as usual, but once we have these, we use Linear Discriminant Analysis to perform that ratio maximization we discussed previously.

For our Fisherfaces model, we used the LDA described above, as well as the PCA described previously in the eigenfaces model. First we perform PCA on the data, then we perform Linear Discriminant Analysis, passing in the projection of the data onto the PCA eigenvectors. Our fisherfaces vectors are then going to be the dot product of eigenvectors from our PCA and the eigenvectors from our LDA. [7]

```
def fisherfaces(data, labels, num_components):
  n, m = data.shape
  num_classes = len(np.unique(labels))
  # use PCA to get eigenvectors
  data_pca, eigenvecs_pca, eigenvals_pca,
  mean_pca = pca_using_svd(data, num_components)
  eigenvecs_pca = np.transpose(eigenvecs_pca)
  #use LDA on data projected into eigen-space
  data_lda, eigenvecs_lda, eigenvals_lda =
  lda(np.dot(data_pca, eigenvecs_pca),
  label, num_components)
  # vectors for fisherfaces are dot product
  # of the vectors from both methods
  eigenvectors =
  np.dot(eigenvecs_pca, eigenvecs_lda)
  return eigenvectors.T, eigenvals_lda, mean_pca
```

The 15 fisherfaces that represent the most variation between the different classes are shown below in Figure 4.

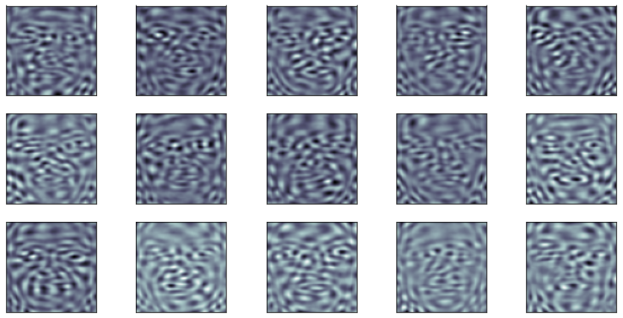


Figure 5. The 15 Fisherfaces

VGG-Face

To utilize the VGG-Face model, we used pretrained VGG-Face weights acquired from Lakshmi Narayana Santha [8].

To download these weights from google drive, it was necessary to create a new virtual environment to install 'gdown'. The requirements.txt for the new environment can be found within our code handin. To download the weights the line included in the top comments of our main.py file may be run through the GCP terminal. We preprocessed the Olivetti Faces dataset by resizing the images to 224x224, and converting them to color in order to pass them through the VGG CNN. After applying the VGG-Face weights, the images are passed through the VGG 16 CNN (Figure 5). Next, after preprocessing and model use, it was necessary to create a reference dataset from within the Olivetti dataset to act as the "training", or reference data. This reference dataset is required as the use of pretrained weights does not call for a training dataset, yet an accuracy benchmark is still necessary. To measure accuracy, we use the KNN approach described in the three previous methods.

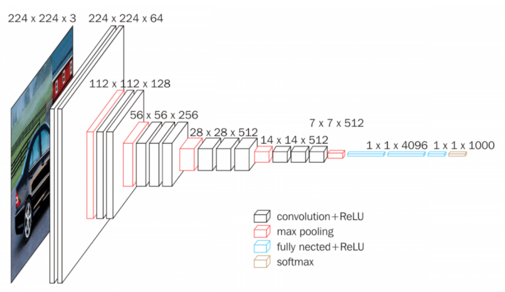


Figure 6. The VGG 16 Convolutional Network [9]

4. Results

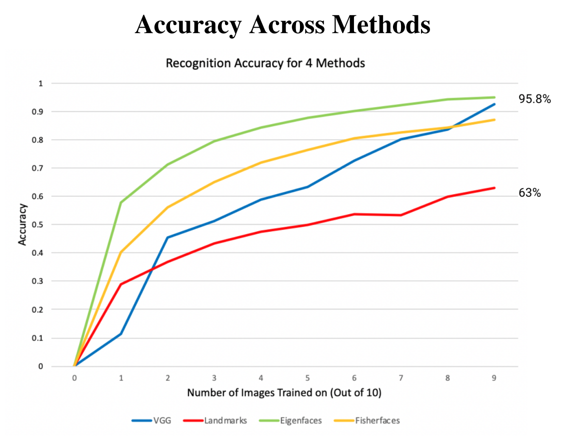


Figure 7. Each method achieved its highest accuracy when training on 9 images per person and testing on 1 per person. The highest accuracy achieved on Facial Landmarks is 63%. The highest accuracy achieved on Eigenfaces is 95.75%. The highest accuracy on Fisherfaces is 87.25%.The highest accuracy achieved on VGG-Face is 92.5%. In the Appendix, please see a graph comparing the number of components against accuracy for Eigenfaces and FisherFaces. We find that very few components can yield high accuracy.

4.1. Discussion

Our project raises questions regarding what characterizes the optimal method of facial recognition, and what the optimal arrangement of training and testing data is. While the highest accuracies are achieved with large training data sets, are they feasible to work with in real-world applications? The highest accuracies our methods were able to achieve were 63%, 95.8%, 87.25%, and 92.5%. These results provoke thought regarding the accuracy threshold that should be accepted as successful in facial recognition applications today. What would be the motives for using each of the different facial recognition techniques? Would it be desirable in some situations to use the less successful recognition methods, and why would this be? The process of completing recognition on very unique training and testing datasets has exemplified the most effective qualities in facial images for recognition. Testing on cartoon images like Simpsons characters, which are very 2-dimensional and side-profile focused, highlight the importance of depth, and positioning in facial recognition dataset images.

It's difficult to compare how well our systems do against current commercial systems. For one, our evaluation dataset, Olivetti, is very limited and would not generalize to real 'in the wild' use cases. For example, we recognize that our facial landmarks system would fail miserably on images at different angles. In addition, state of the art systems are improving quickly. Our best model, eigenfaces, is based on decades-old research. Most face recognition algorithms in 2018 outperformed the most accurate algorithms from late 2013. For example, in 2018 a government department found their system to have a failure rate of 0.2% compared with a 4% failure rate in 2014. [10]

5. Conclusion

Facial Recognition holds great promise in the security and safety of today's globalized world. It can confirm witness identifications, identify terrorists, and lead to the arrest of violent criminals. It is quicker and cheaper than past biometrics, and when coupled with other technologies such as surveillance cameras, its capabilities are vastly extended. In short, it can make the world safer. Yet, it also has clear drawbacks. Depending on the training dataset, it can be biased against certain races.[11]There's a lack of standards regarding the technology's use. Indeed, there are no federal laws governing its usage in law enforcement. And it can be a threat to liberty and justice. For example, facial recognition is a key tool used in China's ongoing genocide against the Uighur people. [2] Thus, if used correctly, with proper oversight, facial identification technology can have great impact.

References

[1] S. Marcel, "Sebastien marcel lab: Face verification / face authentication." 1

[2] N. Sehgal, *Facial Reckoning: The New Age of Digital Surveillance and Facial Recognition*. 1, 4

[3] M. Ringner, "What is principal component analysis?." 2

[4] L. Paul, "Face recognition using principal component analysis method." 2

[5] M. Turk and A. Pentland, "Face recognition using eigenfaces," *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2

[6] A. Martinez, "Fisherfaces," *Scholarpedia*, vol. 6, no. 2, p. 4282, 2011. revision #91266. 3

[7] P. Wagner, "Fisherfaces," *Bytefish*, 2012. 3

[8] L. Santha, "Face recognition with vgg-face in keras.," *Medium*, 2019. 3

[9] M. Hassan, "Vgg 16: Convolutional network for classification and detection," *Neurohive*, 2018. 4

[10] F. R. in 2020, "Facial recognition: top 7 trends." 4

[11] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Proceedings of the 1st Conference on Fairness, Accountability and Transparency* (S. A. Friedler and C. Wilson, eds.), vol. 81 of *Proceedings of Machine Learning Research*, (New York, NY, USA), pp. 77–91, PMLR, 23–24 Feb 2018. 4

Appendix

Team contributions

Laura Dayton I worked with Sindura on the VGG-Face method, which involved image preprocessing and CNN implementation. To aquire the VGG-Face pretrained weights, I found it necessary to create a new virtual environment. Sindura and I completed much evaluation of the VGG-Face performance at various random states to determine which provided optimal accuracy.

Dante Rousseve I implemented the Fisherfaces facial identification system. This was comprised of building up from the PCA used in eigenfaces by coding a Linear Discriminant Analysis method to produce the fisherfaces. I also worked with a Simpsons Dataset to test the cross-validity of facial recognition from 2D to 3D, and worked with the Yale Face Database to test the robustness of the system to illumination changes. Those results can be seen in the Additional Work section.

Neil Sehgal I implemented two of the facial identification systems: the Facial Landmarks and Eigenface based systems. This involved training a facial landmarks classifier and implementing and experimenting with PCA on two human face datasets: LFWCrop and the University of Montreal's Facial Landmarks Kaggle dataset. In addition, I experimented with PCA on Google's cartoon data set, looking at how explained variance differed in the cartoon space differed from the face space. I also used extended the eigenfaces system to build Face Detector seen in the Additional Work section.

Sindura Sriram I implemented the VGG-Face model for Deep Facial Recognition with Laura. This involved building our model architecture, curating diverse datasets for training and validation, and tuning hyperparameters.

Eigenface and Fisherface Results with Varying Components

Eigenface Accuracy

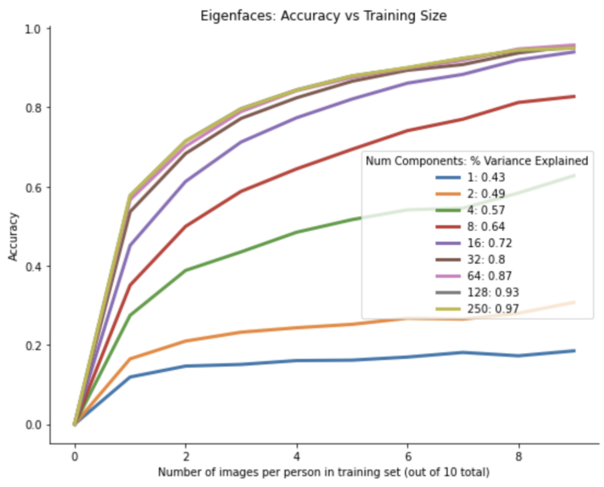


Figure 8. Here we see that only 16 components, explaining 72% of the variance seen in the LFWCrop human face dataset, are needed to achieve a high level of accuracy. Increases beyond 16 do very little.

ADDITIONAL WORK

Face Detection and Cartoon Set

We also extended the eigenfaces implementation into a simple Face Detection model, or a Face or not Face classifer. When using eigenfaces for face identification, we projected each test image into the sub-space spanned by the eigenfaces, the face space, which was determined by fitting PCA to the LFWCrop dataset. We then determined the test images identity by comparing its position in the face space with the positions of known individuals. Thus, all images of human faces should lie somewhat together in this face space, describing images that are face-like. Therefore, if we project a non-face image into the face space, it should lie further away from all face images. We utilize this knowledge to create a simple face detector, testing it on a combined dataset including images from Olivetti Faces and Google's Cartoon Set. Cartoon set is a large database of colored cartoon faces. For

Fisherface Accuracy

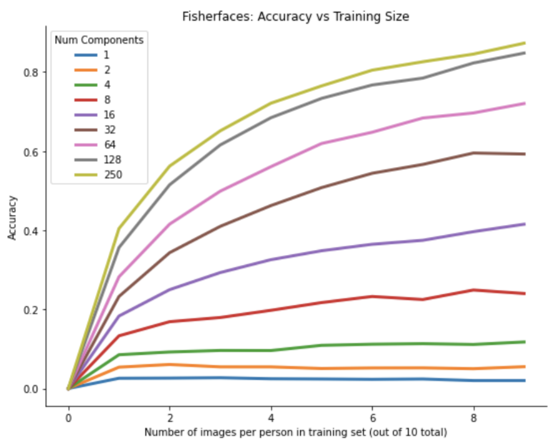
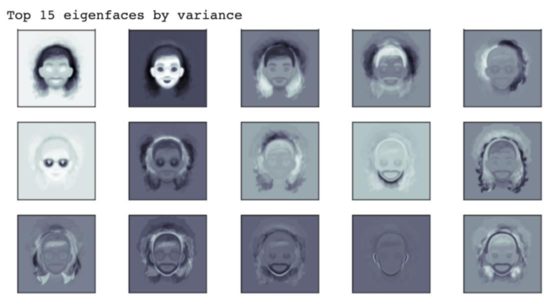


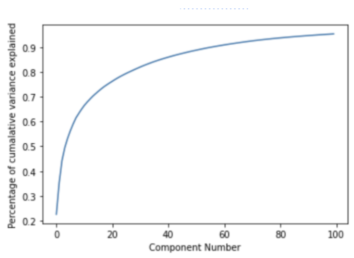
Figure 9. Here we see that more componenets are needed than in Eigenfaces

simplicity's sake, we convert all images to greyscale (we find that we are unable to fit PCA on RGB images of Cartoon Set as this crashes even in Google Colab). The following are 15 of the images in the set:

Applying PCA to this dataset, we see that the eigenfaces of the cartoon set are substantially different than the eigenfaces of LFWCrop.

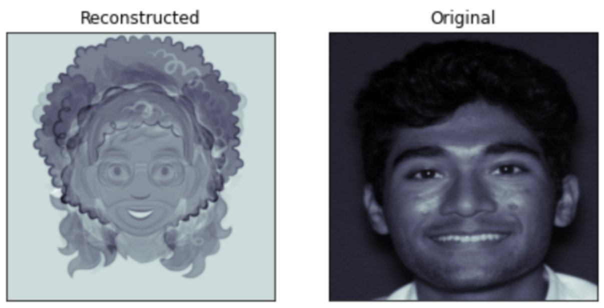


Interestingly, the number of components explaining % variance tracks very similar to the number of components in LFWCrop. For example, 5 components in LFWCrop explains 58.5% of the variance and 5 components in cartoon set explains 56.3% of the variance seen.



```
First 5 components explain 0.5633734962952642% variance in the data
First 20 components explain 0.763184357779579% variance in the data
First 35 components explain 0.8426886897025663% variance in the data
First 50 components explain 0.8886731606047453% variance in the data
Note that you need 50 components to explain .847 of variance in lfw crop dataset
```

Lastly, we see qualitatively, that reconstructing a human face image is not possible using the cartoon set eigenfaces:

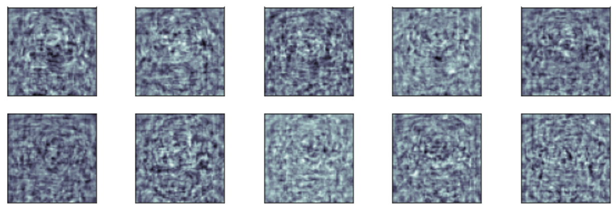


We then implement a simple face detector by fitting PCA to LFWCrop and projecting 400 images from each dataset, Olivetti Faces and Cartoon Set, down to the face space defined by LFWCrop. We then train a binary classifier (KNN with K=1 and train/test split of .33) to detect between human faces and cartoon faces. Accuracy on the test set is 100%.

Simpsons Dataset and Reconstruction

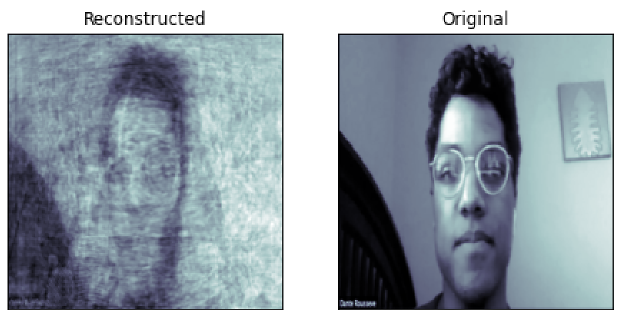
With Fisherfaces, we also trained and tested the model on a labelled Simpsons dataset. We performed two experiments: training on the Simpsons Dataset, then testing on Olivetti, and training on LFW, then testing on the Simpsons Dataset.

The fisherfaces that represent the Simpsons Dataset are as follows:

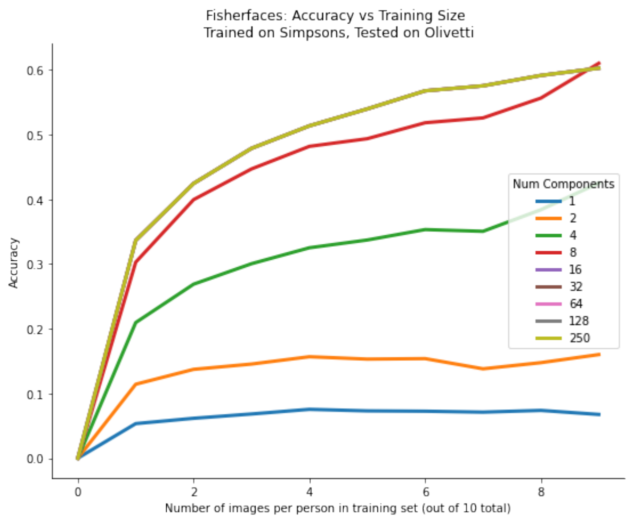


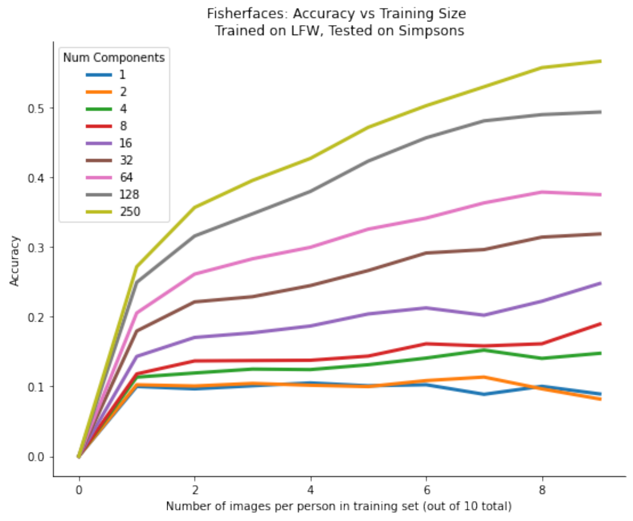
We can see that by training on the Simpsons, we only need about 8 components to achieve the maximum accuracy of the model. The accuracy is still low, likely because of the dimensionality difference.

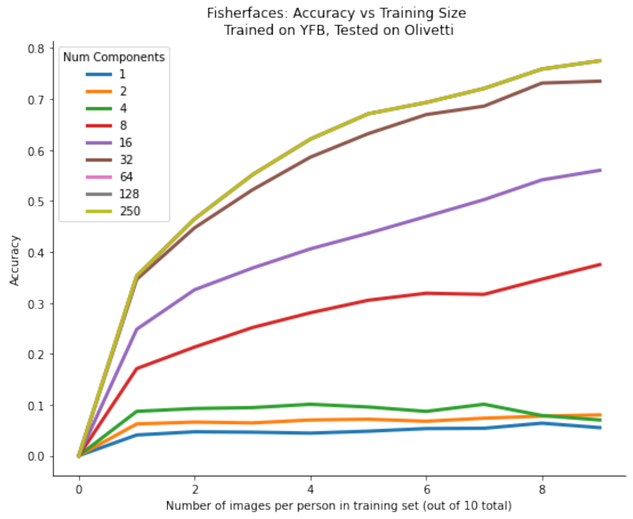
We also attempted to perform reconstruction using the Simpsons Dataset, which required the use of eigenfaces. Below is a reconstruction of one of our group members.

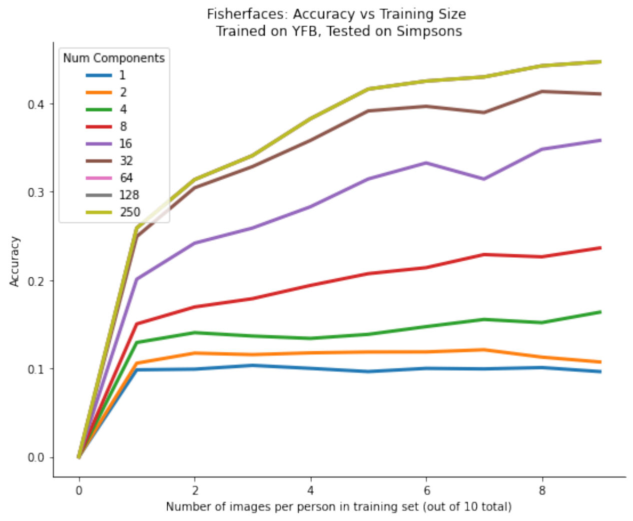


The features that describe Simpsons characters don’t completely translate over to the real world, resulting in a poor reconstruction.


Fisherfaces: Accuracy vs Training Size
Trained on Simpsons, Tested on Olivetti
Accuracy
Number of images per person in training set (out of 10 total)
Num Components


Fisherfaces: Accuracy vs Training Size
Trained on LFW, Tested on Simpsons
Accuracy
Number of images per person in training set (out of 10 total)
Num Components


Fisherfaces: Accuracy vs Training Size
Trained on YFB, Tested on Olivetti
Accuracy
Number of images per person in training set (out of 10 total)
Num Components


Fisherfaces: Accuracy vs Training Size
Trained on YFB, Tested on Simpsons
Accuracy
Number of images per person in training set (out of 10 total)
Num Components