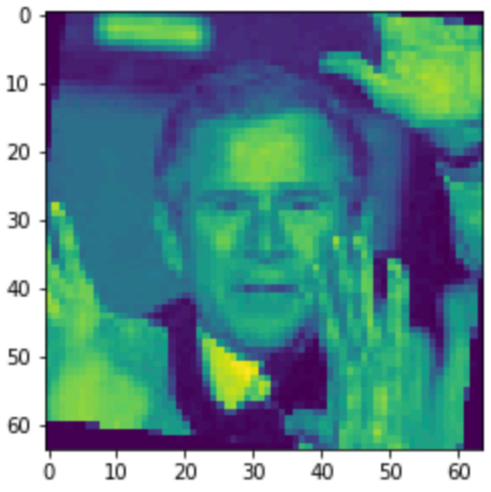
0. Background and Objective

In this project, we seek to analyze different types of popular machine learning models given the task of performing binary image classification. Specifically, we create various models which predict the presence or absence of George Bush and Serena Williams in images. This project report includes a comprehensive overview of all project phases with relevant background and intermediate results. We end with a number of takeaways learned from carrying out this project.

While we do not include every line of code in this report, a full version will be available at https://github.com/philiphossu within a few days of the report being handed in. Lastly, please note that while the third person tense is utilized throughout the report, this project was performed only by myself, Philip Hossu.

0.1 Provided Data

We were provided a dataset containing 13233, 64 x 64 pixel greyscale images. These images were likely taken from the *Labeled Faces In The Wild* dataset available from The University of Massachusetts at Amherst. This dataset contains 530 true labeled instances of George Bush and 52 true labeled instances of Serena Williams. The low number of labels present for Williams will be a large factor for the rest of the project. A sample image of a True instance of Bush can be seen below.



0.2 Metrics

Several different metrics exist which attempt to quantify how “good” a machine learning model is. For this project, we utilize the F1 metric. Given the sample confusion matrix below, we show how F1 is calculated. Note that in this binary classification scenario, true and false denote the presence or absence of the individual in the image, respectively. The F1 value ranges from 0 to 1, with a higher value signifying a “better” model in terms of its predictions.

$$\begin{array}{ccc} & & \text{Predict} \\ & & \begin{array}{cc} T & F \end{array} \\ \text{Actual} & \begin{array}{c} T \\ F \end{array} & \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \end{array}$$

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

We note that there is a distinct difference between Train F1 and Test F1. Train F1 denotes the F1 value which was achieved on the dataset which the model used to learn the behavior of the data. The Test F1 value tells us how well the model performed when trying to classify labeled data it has never seen, and was not trained on.

0.3 Caveats

We test several kinds of models in the following sections and compare them primarily based on the Test F1 result. There is no golden rule for determining what parameters work better than others within the context of the various models, so a large amount of trial and error was performed throughout this project. It is possible that certain models may perform better given a more precisely tuned parameter set, so the results in this report do not reflect the general power of the models. Additionally, the dataset made it difficult to get solid results for the Williams case, as we had such few true instances of her present in the dataset.

0.4 Variables

In this report, we include a number of brief code snipets to illustrate exactly what was tested. Below is a table of commonly used variables.

Variable Name	Contents
X	The provided image dataset. A list of array-formatted 64x64 greyscale images.
y_bush	An array of labels, one for each image, where 0/1 correspond to Bush being/not being in the image.
y_williams	An array of labels, one for each image, where 0/1 correspond to Williams being/not being in the image.

1. KNN and SVMs (Phase 1)

1.1 About KNN

K-Nearest Neighbors (KNN) is a class of supervised learning algorithms which work based on the idea of neighbor voting. Given a labeled dataset, the algorithms will attempt to classify new instances based on distances to known, labeled, points.

We utilize the KNN implementation directly from the Scikit-Learn package in Python. The most fundamental parameter which this KNN algorithm takes is that of n_neighbors. This parameter denotes the number of nearby points which the algorithm will compare the test point with, in order to predict its class.

KNN has a number of advantages, namely that it is easy to understand and use. It can be an effective tool, depending on the kind of data you are analyzing. However, it also has a number of disadvantages. It is computationally expensive and was one of the slowest algorithms tested in the entire project.

For this phase of the project, we additionally utilized a combination of the cross_validate() and StratifiedKFold() functions available in Scikit-Learn. These functions enabled us to split our data into more chunks and get a more accurate average of Train/Test F1 values. More detailed information on how KNeighborsClassifier(), cross_validate(), and StratifiedKFold() operate is available at https://scikit-learn.org/stable/documentation.html.

1.2 Bush and Williams KNN Testing Results

The code snipets below show what we ran in order to test the KNN classifier on the George Bush subset of the data. The code for the Williams testing looks identical to the code above with the appropriate names changed. Note that the random_state parameter was set to 7000 in order to obtain consistent results between trials and splits.

```
knn_bush = KNeighborsClassifier(n_neighbors=1)

cv_results_bush = cross_validate(knn_bush, X, y_bush,
cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=7000),
scoring=('precision', 'recall', 'f1'), return_train_score=False, n_jobs=-1)
```

Below is a table which summarizes the results for the KNN Bush test using 1, 3, and 5 n_neighbors. The optimal F1 appears to come from n_neighbors=1. While this is the case, the F1 values themselves are very low.

n_neighbors	Mean Test F1 (Bush)	Mean Test F1 (Williams)
1	0.1367123867	0.1739130467
3	0.06449348	0
5	0.03172825667	0

The Williams dataset too appears to have benefited from a low n_neighbors. Here we saw 3 and 5 n_neighbors returning a 0 mean F1. A number of potential factors may have caused this. It's important to remember how few True labels we have for Williams in the first place. Perhaps the splitting of the data 3 times dispersed the True labels too far and the model was unable to accurately learn.

1.3 About SVMs

Support Vector Machines (SVMs) are another class of supervised learning methods. They can be used not just for classification, but also regression and outlier detection. SVMs are effective in high dimensional situations, and images fall comfortably into this category. SVMs seek to find a boundary between positive and negative training examples in a high-dimensional space.

We again utilize the Scikit-Learn package for an implementation of SVMs in Python, specifically using the SVC class of functions. These take a very wide variety of parameters, including the penalty (C), as well as the kernel function (linear, polynomial, rbf, sigmoid, etc.), and several parameters specific to each kernel function. The presence of these kernel functions allows us to seek out decision boundaries which are more complex by transforming the data into a new space. Much more information can be found on Scikit-Learn's SVC function at https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

1.4 Bush and Williams SVM Testing Results

For both sets of labels, we tested several different combinations of parameters, utilizing various options for kernel function including linear, polynomial, sigmoid, and rbf. Listing all combinations would not be an efficient use of space, so we will focus only on the parameter combinations which returned the optimal F1 scores, shown below. The sheer number of parameter combinations for the SVC model make it very difficult to effectively search this possible space.

```
svc_model_bush = SVC(C=10000, gamma=0.0001, kernel='rbf')
svc_model_williams = SVC(C=0.04, kernel='linear')
```

For Bush, the optimal SVC model which we were able to locate utilized the rbf kernel, a penalty value of 10000, and a gamma value of 0.0001. For Williams, the optimal SVC model which we were able to locate utilized the linear kernel with a penalty of 0.04. Upon testing these models using cross_validate(), we obtained the below mean Test F1 values.

Model & Parameters	Mean Test F1
Bush SVC: Kernel = rbf, C = 10000, gamma = 0.0001	0.6422265067
Williams SVC: Kernel = linear, C = 0.04	0.54095238

It was not surprising that the optimal Bush model was not the same as the optimal Williams model given the great discrepancy in number of positive labels. It may be for this reason that the Williams dataset reacted better to the simple linear model.

2. Applying PCA (Phase 2)

For the second phase of the project, we applied Principal Components Analysis (PCA) on the dataset to reduce the number of dimensions and see how this would impact the KNN and SVM models previously discussed.

2.1 About PCA

The goal of PCA in this context is to reduce the number of dimensions in the dataset. To do this, it attempts to find the dimensions which account for the highest variance, and then reduce the dimensions which do not fall into this category. One of the ways to perform PCA is through Singular Value Decomposition, a matrix process.

Recall that our images are 64 x 64 pixels meaning that each image has 4096 dimensions. Reducing the number of dimensions in our dataset has the immediate effect of speeding up processing time. However, reducing too many dimensions means that we lose the details in our images, and operate only on a general understanding of what regions and aspects are most different.

We use the PCA implementation from Scikit-Learn. The primary parameter which we changed was the number of dimensions to keep. Details on the others can be found at https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

2.2 Bush and Williams PCA Testing Results

We experimented with a wide range for the number of remaining components, from 32 to 64, 128, 256, 512, 1024, 2048, and finally 3072. An example of the code which we used to perform the PCA reduction is as follows.

```
pca_64 = PCA(n_components=64)
pca_64.fit(X)
X_pca_64 = pca_64.transform(X)
```

From the perspective of reducing processing time, all of the iterations of PCA testing were successful. Processing time drastically reduced as the number of dimensions were reduced from the dataset.

In regard to maintaining (or improving) average Test F1, the results were more ambiguous. When testing KNN, both models still performed optimally under n_neighbors = 1. In fact, both models improved their mean Test F1 under the reduced dataset. However, the number of components to achieve this optimal KNN test differed, 64 for Bush and 256 for Williams. For the SVC Bush case, the previously optimal model maintained the best average Test F1, though this value decreased slightly. For the Williams set, the best model in terms of F1 slightly changed, and the overall value maintained very similar. These results are summarized in the following table.

Model	Pre-PCA Mean Test F1	Post-PCA Mean Test F1
Bush KNN: n_neighbors = 1	0.1367123867	0.1505991767 (using 64 n_components)
Williams KNN: n_neighbors = 1	0.1739130467	0.2232167833 (using 256 n_components)
Bush SVC: Kernel = rbf, C = 10000, gamma = 0.0001	0.6422265067	0.63538346 (using 3072 n_components)
Williams SVC: Kernel = linear, C = 0.04	0.54095238	N/A
Williams SVC: Kernel = linear, C = 0.125	N/A	0.54187192 (using 2048 n_components)

It's notable that despite reducing the number of dimensions, we were able to locate models which very nearly matched or slightly exceeded the previous best results.

3. CNNs (Phase 3)

3.1 About CNNs

Convolutional Neural Networks (CNNs) are certainly the most complex of the supervised learning methods so far addressed in this project report. We will abstain from describing the underlying constructs in detail, but rather give a general motivation for how they work.

As with neural networks in general, CNNs are made up of an input layer, an output layer, and hidden layers. Every layer is made up by nodes, which have several properties. Perhaps most important of these are the activation function which calculates the value to be passed on, and the weight which this value is multiplied by. The hidden layers in a CNN are typically convolution layers and max pool layers. A convolution is a patch of pixels on the image. These patches move around the image in a user-defined manner to learn the details present. Pooling layers combine convolutional information together.

For this project, we utilize the Keras package in conjunction with TensorFlow (TF), available for Python. These allow us to easily create CNNs to be tested. Unlike previous sections, we utilize a simple Test/Train split methodology for the sake of processing time, using Scikit-Learn's train_test_split() function. All of these packages can be learned about at their respective documentation sites.

We additionally have to compute the train and test F1 metrics using our own code now, as Keras and TF do not optimize this value automatically. We use the following code to compute the confusion matrix, precision, recall, and F1.

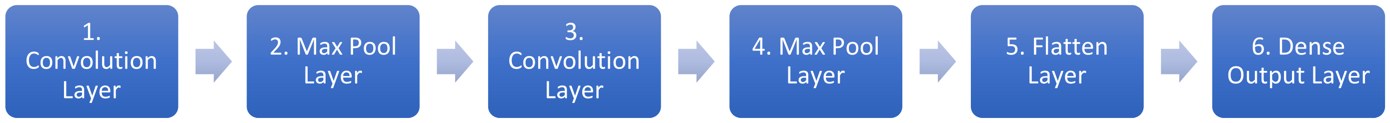
```
model_predicts = np.round(CNN_model.predict(X_test_person))

TP = 0; TN_b 0; FP = 0; FN_b 0
for i in range(0,len(model_predicts)):
    if(y_test_person[i] == 1 and model_predicts[i] == 1):
        TP += 1
    if(y_test_person[i] == 0 and model_predicts[i] == 1):
        FP += 1
    if(y_test_person[i] == 1 and model_predicts[i] == 0):
        FN += 1
    else:
        TN += 1

precision = (TP)/(TP+FP)
recall = (TP)/(TP+FN)
f1 = (2*precision*recall)/(precision+recall)
```

The first line is perhaps the most notable. Here we use the fitted CNN_model's predict feature to feed it the test data of one of the individuals. From there, we go through these predictions and compare them to the true y_test labels of the corresponding person.

3.2 Bush and Williams CNN Testing Results

As previously stated, while CNN models are relatively easy to set up using Keras/TF, it is extremely difficult to determine what to test due to the vast number of parameters. One can modify the structure of the network by adding or removing layers in any pattern they choose, with many types of layers, and every layer having several adjustable parameters. Given the time constraints, we were able to test many different network structures and parameters, and the following yielded the best results.

Beginning with Bush, we found the best test F1 value to come from a CNN containing two pairs of convolution & max pooling, followed by a flatten layer and a sigmoid output layer. For all CNN models we include a diagram and explanation, and for this first one we also include the code.

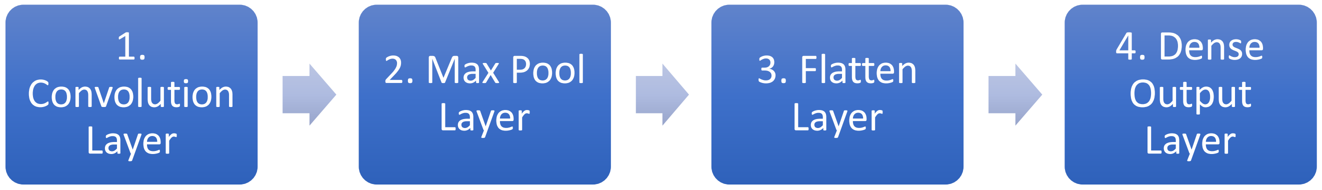


```
CNN_model_bush = Sequential()

CNN_model_bush.add(Conv2D(128, kernel_size=2, activation="tanh",
input_shape=(64, 64, 1)))
CNN_model_bush.add(MaxPooling2D(pool_size=2))

CNN_model_bush.add(Conv2D(64, kernel_size=4, activation="tanh"))
CNN_model_bush.add(MaxPooling2D(pool_size=4))

CNN_model_bush.add(Flatten())
CNN_model_bush.add(Dense(1, activation="sigmoid"))

CNN_model_bush.compile(optimizer='adam',loss='binary_crossentropy',
metrics=['accuracy'])
```

In summary, the Bush CNN consists of the following layers: Layer 1 is a 2D convolution layer. It takes the input shape of the image (64, 64, 1) to denote our 64x64 image in greyscale. The kernel size is 2, activation is tanh, and 128 for the filters (dimensionality of the outputs pace / number of output filters in the convolution). Layer 2 is a max pool layer with a pool size of 2. Layer 3 is another 2D convolution layer. This one also uses a tanh activation function but now has only 64 filters, and a kernel size of 4. Layer 4 is another max pool layer, pool size of 4. Layer 5 is a flatten layer to flatten the input. Layer 6 is an output layer using the sigmoid activation function.

Upon fitting this network over several epochs, we were able to use the model to predict labels on the test data. The results in terms of Test F1 are shown, along with the Williams result, at the end of this section.

In terms of the Williams network, a simpler structure appeared optimal. However, regardless of the number of epochs used to fit this simpler model, the test F1 varied wildly.



A summary of the layers present is as follows: Layer 1 is a 2D convolution layer taking the input shape of the image (64x64, greyscale), the relu activation function, a kernel size of 3, and 32 filter size. Layer 2 is a max pool layer with a pool size of 2. Layer 3 is a flatten layer. Layer 4 is a dense output layer using the sigmoid activation function.

As with the Bush model, we must fit the Williams CNN to the training data. Even within the same network structure, parameters, and number of epochs, fitting multiple times yielded very different train and test F1 values, ranging from absolute 0 to the 0.56 reported below.

Model	Train F1	Test F1
Bush CNN (6 Layer, 5 epochs)	0.9048913043478262	0.8277777777777778
Williams (4 Layer, 4 epochs)	0.8717948717948717	0.5789473684210527

The 6 layer CNN structure performed very well for the Bush data. Despite the long fitting process, the Test F1 value was significantly higher than any previous model. As for the Williams data, the same cannot be said. The results were sporadic and often very low. While this model barely surpasses the SVM result, it's difficult to confidently say that it is better.

4. Transfer Learning (Phase 4)

In the final phase of the project, we attempt a simple form of transfer learning. Transfer learning describes the process of fitting a model to a related but different dataset and then, without clearing the weights learned, re-fit the model to the Bush/Williams dataset. Perhaps by fitting the model to a prior face-related dataset, it would enable us to achieve a higher Test F1 value on the target dataset.

4.1 Data and Processing

To pre-train the model, we first need a relevant dataset. We selected *The Extended Yale Faces Database B (Original Images)* to pursue. This dataset is available for download at http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html. Here we find approximately 16,000 images of 28 subjects under various poses and illumination conditions. While this set needed to be heavily processed to fit the objectives of this project, we thought it would be good to train on faces which are clearly visible in different lighting conditions, and see how this model would help us in the Bush/Williams classification.

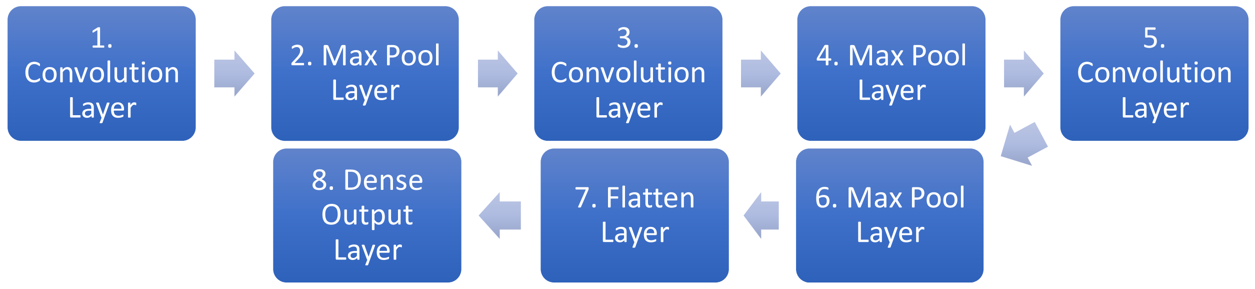
After downloading the dataset, we processed it using the Python Image Library (PIL) to manipulate the images themselves, a package called glob to help go through directories, and numpy to convert images to arrays. We decided to select individual number 18 to be our True individual, and labeled all the other individuals with False. The whole data processing loop can be found below, with comments included to explain what each piece is doing.

```
X = []; y_18 = [];
i = 0
folderslist = glob.glob("ExtendedYaleB/*/")

for i in range(0,len(folderslist)):
    for filename in glob.glob(folderslist[i]+'*.pgm'):
        # Open image
        im = Image.open(filename)
        # Crop image to square
        width, height = im.size
        left = (width - 480)/2
        top = (height - 480)/2
        right = (width + 480)/2
        bottom = (height + 480)/2
        im = im.crop((left, top, right, bottom))
        # Resize image to 64x64
        im = im.resize((64,64), Image.ANTIALIAS)
        # Convert to array for pickling purposes
        im = np.array(im)
        # Add to images list
        X.append(im)
        # If person 18, label in y_18
        if(folderslist[i] == 'ExtendedYaleB/yaleB18/'):
            y_18.append(1)
        else:
            y_18.append(0)
    # Increment folders counter
    i = i + 1
```

From here, the data was easily pickle'd and stored for easy access in future loads.

4.1 Pre-Training CNNs on Yale Dataset

Another difficult decision comes when considering what model to pre-train to the Yale Dataset. We attempted to train several different kinds of models and apply them both to the Bush and Williams datasets with varying levels of success. Ultimately, we chose to use the following CNN model to transfer to both Bush and Williams.



This model has three layers of convolution and max pooling. Specifically, Layer 1 is a convolution layer with 128 filters, kernel size 2, tanh activation function, and an input shape of the resized greyscale images (64, 64, 1). Layer 2 is a max pool layer with a pool size of 2. Layer 3 is a convolution layer with 64 filters, kernel size of 4, and a tanh activation function. Layer 4 is a max pool layer with pool size of 4. And finally, layers 5 and 6 are duplicates of layers 1 and 2. The network ends with Layer 7 being a flatten layer, and layer 8 being the output layer with the sigmoid activation function. We fitted this network over a series of 5 epochs.

This transfer CNN achieved a perfect 1.0 Test F1 on its respective Yale testing data. The complex nature of the network gives it a lot of predicting power when paired with a big training dataset and plenty of true labels to learn from. Additionally, the images in the Yale Dataset are fairly “nice” in the sense that most contain only one individual, usually clearly lit, with no obstructions covering parts of the face.

4.2 Bush and Williams Transfer Testing Results

Using this pre-trained model, we re-fitted it to the Bush and Williams datasets (on two separate, non-reinitialized copies of the pre-trained model) for the purpose of observing the new train and test F1 values. The resulting Train/Test F1 values are found below.

Model	Train F1	Test F1
Bush 8 Layer Transfer (10 epochs)	1.0	0.9285714285714285
Williams 8 Layer Transfer (5 epochs)	0.90625	0.5833333333333334

The transfer model achieved the best Test F1 value across the entire project for the Bush and Williams datasets. The Bush value is now very high, and the raw code results show that over the testing data, only 3 predictions were false positives and 21 were false negatives, compared with 156 true positives and 4390 true negatives – a fairly remarkable result. The Williams results were still fairly poor, and this can certainly be attributed to the same reasons as before – not enough labels to learn accurately from.

It’s worth briefly noting that the models used here do differ slightly from those in Phase 3. Namely, they have an extra convolution and max pool layer. As a crude preliminary comparison, we ran the structure defined in section 4.1 in a non-transfer setting directly on the Bush/Williams datasets. Their Train/Test F1 results are shown below.

Model	Train F1	Test F1
Bush 8 Layer NO Transfer (10 epochs)	1.0	0.9085365853658538
Williams 8 Layer NO Transfer (5 epochs)	0.6346153846153846	0.46153846153846156

Given this, we can draw a preliminary conclusion that the transfer process did aid the models in terms of F1.

5. Conclusion & Takeaways

In this project, we worked with several different kinds of prominent machine learning models to classify image data. Specifically, we sought to compare average F1 values of KNN, SVMs, and CNNs in terms of their ability to perform binary classification of the presence of George Bush and Serena Williams in 64x64 pixel greyscale images. In this process, we also considered the impact that PCA has on KNN and SVMs, as well as the potential benefit gained from a simple implementation of transfer learning.

From all tested models, the best F1 result for the Bush dataset (~0.929) came from the model structure in Section 4.1, pre-trained on the Yale Extended Dataset B over 5 epochs, then to the Bush data over 10 epochs. Similarly, the best F1 result for the Williams dataset (~0.583) came from the model structure in Section 4.1, pre-trained on the Yale Extended Dataset B over 5 epochs, then trained on the Williams data over 5 epochs. However, stepping away from the perspective of F1, these CNN models certainly took some of the longest time to run and find optimal parameters for.

There are a number of more general takeaways learned from carrying out this project.

1. If you don't have enough true labels, it's difficult for any of the algorithms we tested to predict well. Exactly what counts as "enough" will likely depend on the contents and specifics of the images themselves. We observed this trend throughout the project, as the Williams predictions maintained an average F1 regardless of the model. Similarly, many models end up predicting all testing data points as false.
2. Randomness is very difficult to overcome. Even when running the same model using the same parameters multiple times, results often varied. This made it very difficult to concretely determine the power of a certain network structure, for example.
3. There must be more efficient ways to search for parameter combinations in simpler models, if you have enough time. The models we tested take so many user defined parameters that it's difficult to know what will and will not work well. Only through more practice and learning will a better intuition arise as to where a good place to start is.
4. It would have been wise to make more plots and figures along the way. We would have liked to give more reasoning behind various decisions, but it was too late to go back and re-run hours of tests to gain data in order to compare.