COS 314 Assignment 1 Report

Author: Jakobus Stephanus Potgieter - 18068911

Data Extraction:

The data consists of 3023 images of the faces of 62 individuals, having a ratio of 87x65 pixels. The data is immediatly masked to be less skew - since it initially highly favors G.W. Bush. A min-max scaler is also applied to standardize the data before the application of PCA.

In [1]:

```python
# required imports
import matplotlib.pyplot as plt
import numpy as np
from pickle import load
```

In [2]:

```python
people = load(open('people.pkl', 'rb'))
image_shape = people.images[0].shape
print(image_shape)
print(people.images.shape)
```

```
(87, 65)
(3023, 87, 65)
```

In [3]:

```python
# mask data - less skewed
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
  mask[np.where(people.target == target)[0][:50]] = 1

X_people = people.data[mask]
y_people = people.target[mask]

print(X_people.shape)
```

```
(2063, 5655)
```

In [4]:

```python
# standardize data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_people)
X_people_Scaled = scaler.transform(X_people)
```

Task 1 - PCA

In [5]:

```
# pca library
from sklearn.decomposition import PCA
```

In [6]:

```
# (a)
def PCA_K_Components(k, X):
    pca = PCA(n_components=k)
    pca_transform = pca.fit_transform(X)
    return pca.inverse_transform(pca_transform)

print(PCA_K_Components(15, X_people_Scaled).shape)
```

```
(2063, 5655)
```

b) Firstly we apply PCA to the dataset with a reduction of 15 principle components. Below these 15 principle components are then displayed as images in the 87x65 px format.

In [7]:

```
# (b)
pca = PCA(n_components=15)
components_b = pca.fit(X_people_Scaled).components_

components_b_display = components_b.reshape((15, 87, 65))
print(components_b_display.shape)
```

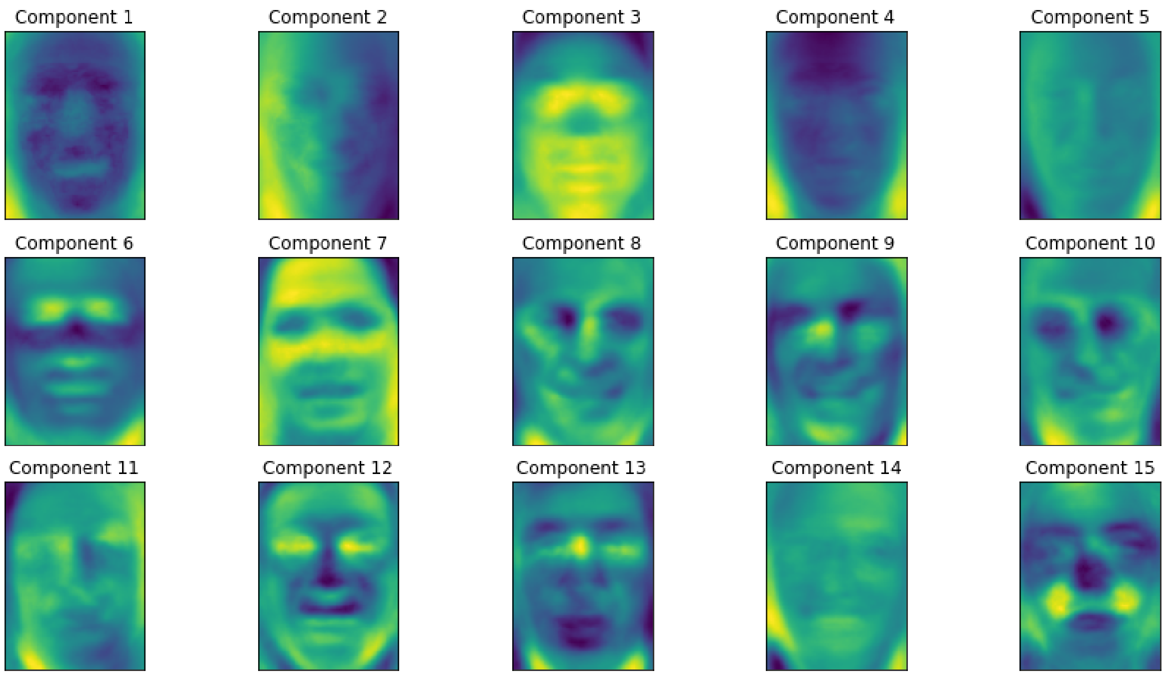
```
(15, 87, 65)
```

In [8]:

```python
# display first 15 priciple components
print('First 15 principle components of the dataset:')
fix, axes = plt.subplots(3, 5, figsize=(15, 8),subplot_kw={'xticks': (), 'yticks':

i = 1
for image, ax in zip(components_b_display, axes.ravel()):
    ax.imshow(image)
    ax.set_title("Component " + str(i))
    i = i + 1
```

```
First 15 principle components of the dataset:
```



From our image depictions of the first 15 principle components above, we see these images to form general and rough structures of human faces. This is to be expected, as PCA identifies the most important and distinguishable components in the dataset, and since our images are of human faces these components will identify important features of the given faces. We also see the components to roughly identify different specific individuals, which is also expected since out dataset consists of multiple (up to 50) images of the same individuals. Thus, unique features of individuals will also be extracted and contained in the first 15 principle components.

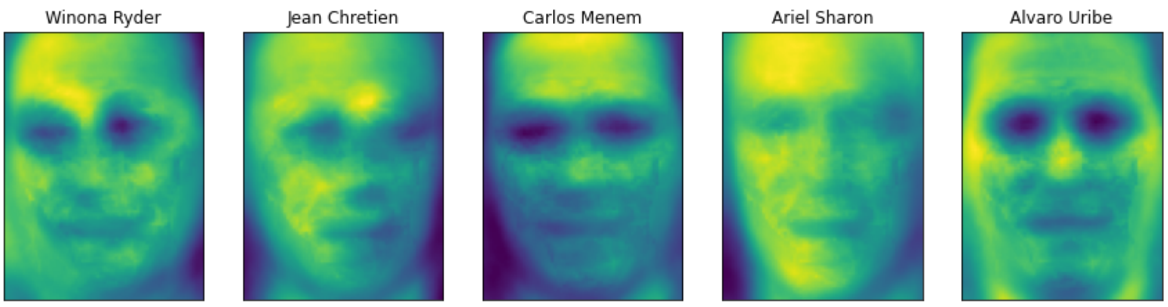
c) Next we again apply PCA to our dataset, this time for 10,30,50,100,500 principle components. We then reconstruct the images of our dataset with the respective amount of componets - displaying only the first 5 images in our dataset.

In [17]:

```
# (c)
# 10
components_10 = PCA_K_Components(10, X_people_Scaled)
components_10_display = components_10.reshape((2063, 87, 65))

print('First 5 images with 10 principle component pixels:')
fix, axes = plt.subplots(1, 5, figsize=(15, 8),subplot_kw={'xticks': (), 'yticks':
for target, image, ax in zip(people.target, components_10_display, axes.ravel()):
  ax.imshow(image)
  ax.set_title(people.target_names[target])
```

```
First 5 images with 10 principle component pixels:
```

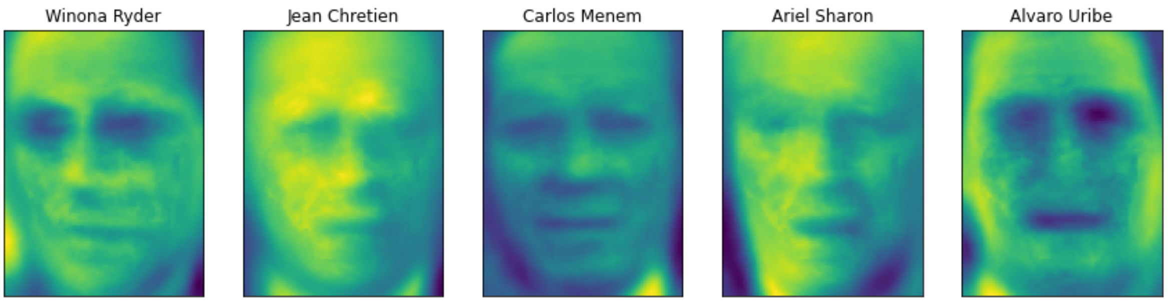


In [16]:

```
# 30
components_30 = PCA_K_Components(30, X_people_Scaled)
components_30_display = components_30.reshape((2063, 87, 65))

print('First 5 images with 30 principle component pixels:')
fix, axes = plt.subplots(1, 5, figsize=(15, 8),subplot_kw={'xticks': (), 'yticks':
for target, image, ax in zip(people.target, components_30_display, axes.ravel()):
  ax.imshow(image)
  ax.set_title(people.target_names[target])
```

```
First 5 images with 30 principle component pixels:
```

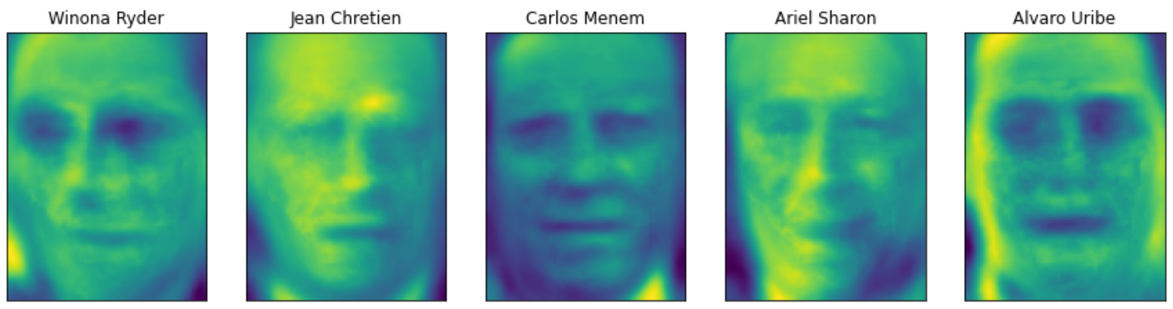


In [15]:

```
# 50
components_50 = PCA_K_Components(50, X_people_Scaled)
components_50_display = components_50.reshape((2063, 87, 65))

print('First 5 images with 50 principle component pixels:')
fix, axes = plt.subplots(1, 5, figsize=(15, 8),subplot_kw={'xticks': (), 'yticks':
for target, image, ax in zip(people.target, components_50_display, axes.ravel()):
  ax.imshow(image)
  ax.set_title(people.target_names[target])
```

```
First 5 images with 50 principle component pixels:
```

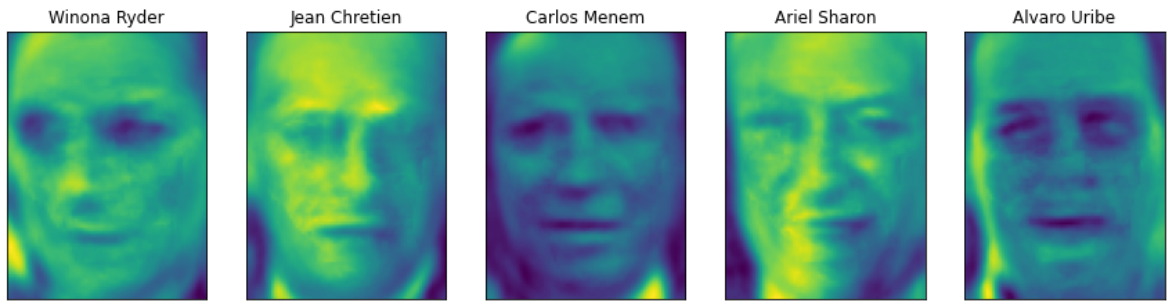


In [14]:

```
# 100
components_100 = PCA_K_Components(100, X_people_Scaled)
components_100_display = components_100.reshape((2063, 87, 65))

print('First 5 images with 100 principle component pixels:')
fix, axes = plt.subplots(1, 5, figsize=(15, 8),subplot_kw={'xticks': (), 'yticks':
for target, image, ax in zip(people.target, components_100_display, axes.ravel()):
  ax.imshow(image)
  ax.set_title(people.target_names[target])
```

```
First 5 images with 100 principle component pixels:
```

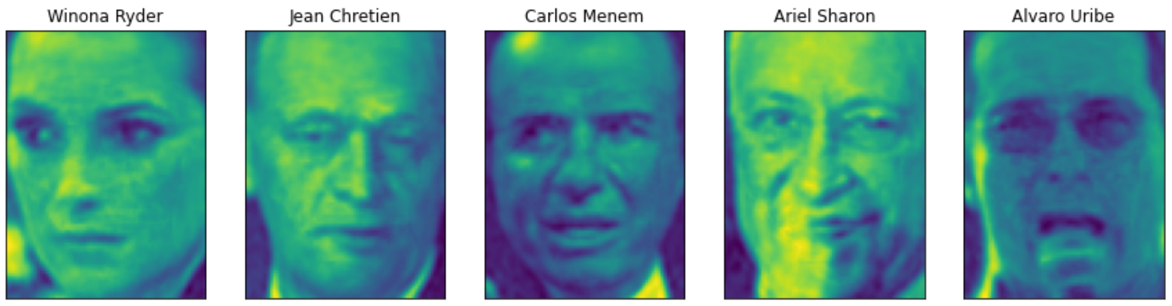


In [13]:

```
# 500
components_500 = PCA_K_Components(500, X_people_Scaled)
components_500_display = components_500.reshape((2063, 87, 65))

print('First 5 images with 500 principle component pixels:')
fix, axes = plt.subplots(1, 5, figsize=(15, 8),subplot_kw={'xticks': (), 'yticks':
for target, image, ax in zip(people.target, components_500_display, axes.ravel()):
  ax.imshow(image)
  ax.set_title(people.target_names[target])
```

```
First 5 images with 500 principle component pixels:
```



From our displayed images above we see that the clarity/quality of the images increase with the increase in the number of principle components. This is to be expected, as more components will directly contain more principle pixel data of the images. We also see that about 500 components is enough to produce images with enough quality to be able to clearly identify the individuals and around 100 and less the images start to become unclear. This serves as motivation for the use of 500 principle components of the data in the neural network of part 2.

Task 2 - NN Classification

Read Me:

The Neural Network model for Part 2 can be found in the 'NN Classifier Model' folder. The following specifications were used in the model:

- PCA on input data with 500 components
- Tested for multiple number of hidden layer nodes (presented in results)
- Learning rate $\alpha = 0.01$
- Activation functions: Layer 1: tanh(x); Layer 2: sigmoid(x)

Please supply the input data parameters in the following order:

1) Training set 2) Training labels 3) Test set 4) Test labels

Data extraction and split:

Below we firstly extract the data of the 10 specified individuals. The data is also split into a 70/30 ratio of training/testing sets, which was used for testing purposes. I now note that this was not a stratified split, but was mostly sufficient for testing. The included npy files in the folder are the given sets that were uploaded on ClickUp.

In [23]:

```
import matplotlib.pyplot as plt
import numpy as np
```

In [5]:

```
# extract data of the 10 given people + construct multi classification label matrix
target_names_new_index = [21,5,35,53,58,14,10,15,12,25]
target_names_new = []
for i in range(0,len(target_names_new_index)):
    target_names_new.append(people.target_names[target_names_new_index[i]])

X_people_new = []
y_people_new = []

for i in range(0, len(y_people)):
    if y_people[i] in target_names_new_index:
        index = target_names_new_index.index(y_people[i])
        y_people_new.append(index)
        X_people_new.append(X_people[i])

print(len(y_people_new))
print(len(X_people_new))
```

```
500
500
```

In [7]:

```
# split data into training:test sets in ratio 70:30
X_people_new_train = []
X_people_new_test = []
y_people_new_train = []
y_people_new_test = []
for i in range(0,len(y_people_new)):
    if i < (len(y_people_new)*0.7):
        X_people_new_train.append(X_people_new[i])
        y_people_new_train.append(y_people_new[i])
    else:
        X_people_new_test.append(X_people_new[i])
        y_people_new_test.append(y_people_new[i])

print(len(y_people_new_train))
print(len(y_people_new_test))
```

```
350
150
```

In [23]:

```
# store in .npy files for NN model parameters
training_arr = np.array(X_people_new_pca_train)
np.save('training_arr.npy', training_arr)

test_arr = np.array(X_people_new_pca_test)
np.save('test_arr.npy', test_arr)

training_label = np.array(y_people_new_train)
np.save('training_label.npy', training_label)

test_label = np.array(y_people_new_test)
np.save('test_label.npy', test_label)
```

```
(350, 500)
(150, 500)
(350,)
(150,)
```

Results and Observations:

The datasets used for the training and testing leading to the following results, were the supplied sets uploaded on ClickUp.

The neural network model was trained and tested for 1,5,10,20,30,40 nodes in the hidden layer - with 1 node being very similar to plain logistic regression.

Only the confusion matrices of the 30 hidden layers test will be displayed - as it obtained the best accuracy on the test set. A comparison of the obtained accuracy on the train and test set with a change in the number of hidden layer nodes will also be graphically shown.

The requested output data of the tests are also provided in identified folders.

Confusion matrices of test with 30 nodes in hidden layer:

In [21]:

```
# 30 hidden layers confusion matrices
train_confusion = np.loadtxt("Tests/Test_30/TrainConfusion.txt", int)
print('Training set confusion matrix:')
print(train_confusion)
print("Accuracy: 100%")
```

```
Training set confusion matrix:
[[36  0  0  0  0  0  0  0  0  0]
 [ 0 29  0  0  0  0  0  0  0  0]
 [ 0  0 37  0  0  0  0  0  0  0]
 [ 0  0  0 30  0  0  0  0  0  0]
 [ 0  0  0  0 40  0  0  0  0  0]
 [ 0  0  0  0  0 35  0  0  0  0]
 [ 0  0  0  0  0  0 33  0  0  0]
 [ 0  0  0  0  0  0  0 35  0  0]
 [ 0  0  0  0  0  0  0  0 38  0]
 [ 0  0  0  0  0  0  0  0  0 37]]
Accuracy: 100%
```

In [23]:

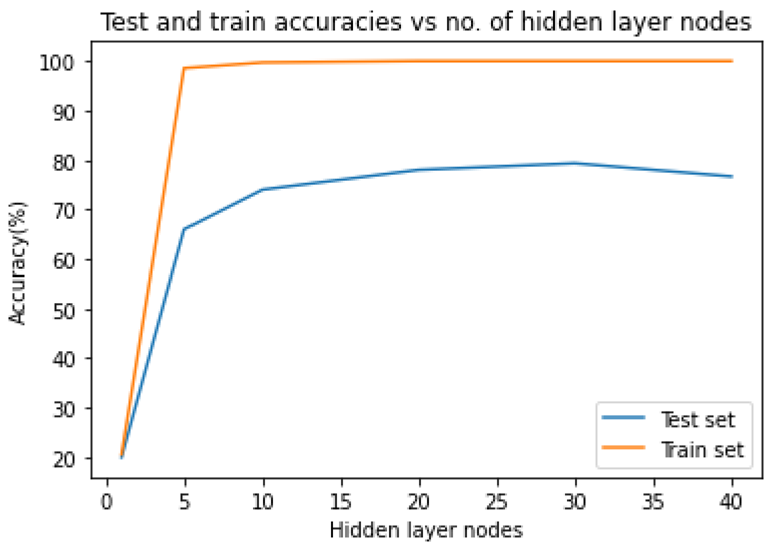
```
test_confusion = np.loadtxt("Tests/Test_30/TestConfusion.txt", int)
print('Test set confusion matrix:')
print(test_confusion)
print("Accuracy: 79.33%")
```

```
Test set confusion matrix:
[[11  0  0  0  0  1  2  1  0  0]
 [ 0 17  0  2  0  0  0  0  0  1]
 [ 0  1 13  2  0  1  0  1  0  0]
 [ 1  1  0 14  0  0  1  1  0  0]
 [ 1  1  0  0 10  0  2  0  1  0]
 [ 0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  2  0  0 10  0  0  1]
 [ 1  0  0  0  0  0  0 10  0  0]
 [ 0  1  0  0  0  0  2  0 11  1]
 [ 0  0  0  0  0  0  0  2  0 10]]
Accuracy: 79.33%
```

In [37]:

```
# train/test accuracy vs number of hidden layers
layers = [1,5,10,20,30,40]
test_accuracy = [20.00,66.00,74.00,78.00,79.33,76.67]
train_accuracy = [20.57,98.57,99.71,100,100,100]

plt.plot(layers,test_accuracy, label='Test set')
plt.plot(layers,train_accuracy, label='Train set')
plt.ylabel('Accuracy(%)')
plt.xlabel('Hidden layer nodes')
plt.title('Test and train accuracies vs no. of hidden layer nodes')
plt.legend()
plt.show()
```



From the above results we can observe that only having a single node in the layer is very uneffective in learning as a model in the context of our tests and data. An increase to only 5 nodes makes the model a lot more effective in learning and predicting test data - showing the improvement that the introduction of the hidden layer supplies to the model. We see a maximum accuracy around 30 nodes, with a slight decrease in accuracy with a further increase in nodes. This is most likely due to the datasets being used being quite small, meaning there is not enough data to properly train a too large amount of weights for unseen test data. This also shows the importance of choosing the correct amount of nodes for a neural network model, as it can directly affect the accuracy of the model's predictions on test data.

In [ ]:

```

```