**a)**

A close up of a logo

Description automatically generatedA screenshot of a cell phone

Description automatically generated

*Figure 1: Plot of singular values for Figure 2: Plot of cumulative significance*

*corresponding principal components as additional principal components*

*of 190/200 neutral faces are used*

I decided to use the number of principal components that would encompass 99% of the total sum

of the magnitudes of all the singular values. Since the significance of a principal component is determined by the magnitude of its singular value (larger singular values represent larger variance of the data in the direction of the corresponding eigenvector), it is reasonable to expect that if the sum of corresponding eigenvalues for the first 54 principal components equates to 99% of the total sum of all the eigenvalues – then these 54 principal components are able to encapsulate 99% of the data required to reconstruct an image. This allows me to decrease the number of principal components by 72% while only losing 1% of the data (assuming this heuristic approach equates to this conclusion). I have no proof of the validity of this approach – it is just an intuitive, heuristic approach obtained by observing Figures 1 and 2.

**b)**

A screenshot of a cell phone

Description automatically generated

*Figure 3: Square root of MSE between*

*neutral face 4a and its reconstruction*

*vs number of principal components*

*used for the reconstruction*

The square root of the mean squared error was plotted to aid in the visualization of the plot. As the number of principal components used is increased the MSE decreases exponentially and reaches a minimum value of approximately 25 (52). This equates to approximately 1.95% of error\*, which is not far off from the 1% goal.

\* If the average difference between each pixel is 5 for pixels that range from 0 to 255, then

the average percentage of difference between the images is calculated by %.

**c)**

A screenshot of a cell phone

Description automatically generated

*Figure 4: Square root of MSE between*

*smiling face 4b and its reconstruction*

*vs number of principal components*

*used for the reconstruction*

The square root of the mean squared error was plotted to aid in the visualization of the plot. As the number of principal components used is increased the MSE decreases exponentially and, just as Figure 3 does, reaches a minimum value of approximately 25 (52). This equates to approximately 1.95% of error\*, which is not far off from the 1% goal. One noticeable difference is that the MSE hits a maximum value of approximately 45 in Figure 4, whereas the maximum value in Figure 3 is approximately 39. Also, the decrease in Figure 4 has a greater initial slope – indicating the greater impact of additional principal components on the MSE when principal component levels are below ~10. Since the principal components were created from neutral faces, it makes sense that the principal components would perform better at recreating a neutral face as opposed to a smiling one. This seems to be true when the number of principal components is less than or equal to approximately 10. After that, however, the principal components are just as successful in recreating a smiling face as they are in recreating a neutral face. However, this is only one example and it would be interesting to see if this trend holds for all pairs of neutral and smiling faces.

\* If the average difference between each pixel is 5 for pixels that range from 0 to 255, then

the average percentage of difference between the images is calculated by %.

**d)**

A screenshot of a cell phone

Description automatically generated

*Figure 5: Square root of MSE between*

*neutral face 195a and its reconstruction*

*vs number of principal components*

*used for the reconstruction*

The square root of the mean squared error was plotted to aid in the visualization of the plot. As the number of principal components used is increased the MSE decreases exponentially and reaches a minimum value of approximately 16 (42). This equates to approximately 1.56% of error\*, which is even closer to the 1% goal than previous attempts. One noticeable difference is that the MSE hits a maximum value of approximately 31 in Figure 5, whereas the maximum values were approximately 39 and 45 in Figures 3 and 4 respectively. It’s interesting that the minimum MSE values are approximately the same, if not smaller, when the image to reconstruct is not used in the initial construction of the principal components. However, it may just have been that the faces selected for recreation in Figures 4 and 5 were easy to reconstruct for the set of principal components.

\* If the average difference between each pixel is 4 for pixels that range from 0 to 255, then

the average percentage of difference between the images is calculated by %.

**e)**

A close up of a device

Description automatically generatedA screenshot of a cell phone

Description automatically generated

*Figure 6: Reconstruction of the Tesla Figure 7: Reconstruction of the Tesla*

*Cybertruck using all principal components Cybertruck using 15 principal components*

I tested the ability for the principal components created by using neutral faces to reconstruct a Tesla Cybertruck. The reconstruction in Figure 6 is quite good. Although some of the sharper detail is lost, one can clearly tell that it is a car. Just to see what would happen, I attempted a reconstruction of the truck using only 15 principal components. You can see a blurry car in the background accompanied by a pair of creepy eyes. I’m surprised at how good the reconstruction is when using all the principal components – I was expecting the results in Figure 6 to be about as good as they are in Figure 7.

**f)**

A close up of a mans face

Description automatically generatedA screenshot of a cell phone

Description automatically generated

*Figure 8: Reconstruction of neutral face Figure 9: Reconstruction of neutral face*

*4a rotated 45 degrees using all principal 4a rotated 45 degrees using 15 principal components components*

A close up of a mans face

Description automatically generatedA close up of a mans face

Description automatically generated

*Figure 10: Reconstruction of neutral face Figure 11: Reconstruction of neutral face*

*4a rotated 180 degrees using all principal 4a rotated 180 degrees using 15 principal components components*

I tested the ability for the principal components created from neutral faces to reconstruct image 4a (an image in the set of faces used to create the principal components) after being rotated. In Figures 8 and 10 the recreation is very good – there is just a bit of detail lost in the finer features of the face. When only using 15 principal components the creepy eyes come back, but one can begin to see the eyes and part of the nose of the original image (in the correct orientation). I’m also impressed with the reconstruction results of Figure 8, more so than the reconstruction of Figure 10. I had assumed the greatest variation would be in the vertical direction and the second greatest variation to be in the horizontal direction (simply from observing the organization of features on the faces). Since Figure 10 keeps a vertical and horizontal organization of features I expected the reconstruction to be fine. In Figure 8, however, the orientation of features is now in a diagonal direction and I expected this to hinder the ability for the principal components to reconstruct the image to the point that Figure 8 would look like Figure 9.

**Appendix**

*Code:*

1 *##################################*

2 *# Joseph Bell #*

3 *# ECE 269 PCA Eigenfaces Project #*

4 *# 12/1/2019 #*

5 *##################################*

6

7 **import** **matplotlib.pyplot** **as** **plt**

8 **import** **matplotlib.image** **as** **mpltimg**

9 **import** **os**

10 **import** **numpy** **as** **np**

11 **import** **math**

12

13

14 *########################################################*

15 *# input: absolute path to folder containing face images#*

16 *# returns: m \* L \* N array, where m = number of face #*

17 *# images, L rows in image, N cols in image #*

18 *########################################################*

19 **def** vectorize\_face\_folder(face\_folder\_path): *#converts face images to matrix*

20 face\_images = os.listdir(face\_folder\_path)

21 first\_face\_path = os.path.join(face\_folder\_path,face\_images[0]) *# Used to initialize np.array to store all faces - assumes all faces are same size as first image*

22 num\_of\_faces = len(face\_images) *# which is true in this case as they've all been cropped*

23 first\_face = mpltimg.imread(first\_face\_path)

24 faces\_vector = np.zeros((num\_of\_faces, first\_face.shape[0], first\_face.shape[1]))

25

26 **for** i **in** range(num\_of\_faces):

27 face\_path = os.path.join(face\_folder\_path, face\_images[i])

28 face\_read = mpltimg.imread(face\_path)

29 faces\_vector[i,:,:] = face\_read

30

31 **return** faces\_vector

32

33 *########################################################*

34 *# input: m\*L\*N array, where m = number of face #*

35 *# images, L rows in image, N cols in image #*

36 *# returns: L\*N array which is the mean of m face #*

37 *# images #*

38 *########################################################*

39 **def** calculate\_mean\_face(faces\_vector): *#calculates mean of all matrices that represent faces*

40 num\_of\_faces = faces\_vector.shape[0]

41 face\_summation = np.zeros(faces\_vector[0].shape)

42 **for** i **in** range(num\_of\_faces):

43 face\_summation = face\_summation + faces\_vector[i,:,:]

44

45 mean\_face = face\_summation/num\_of\_faces

46

47 **return** mean\_face

48

49 *##################################################################*

50 *# input (faces\_vector): m\*L\*N array, where m = number of face #*

51 *# images, L rows in image, N cols in image #*

52 *# input (mean\_face): L\*N array of mean face of m face images #*

53 *# returns: m\*L\*n array of faces\_vector[i,:,:] - mean\_face #*

54 *# #*

55 *##################################################################*

56 **def** calculate\_mean\_adjusted\_faces(faces\_vector, mean\_face):

57 mean\_adjusted\_faces = np.zeros(faces\_vector.shape)

58 num\_of\_faces = faces\_vector.shape[0]

59 **for** i **in** range(num\_of\_faces):

60 mean\_adjusted\_faces[i,:,:] = faces\_vector[i,:,:] - mean\_face

61

62 **return** mean\_adjusted\_faces

63

64 *##################################################################*

65 *# input: m\*L\*N array, where m = number of face #*

66 *# images, L rows in image, N cols in image #*

67 *# returns: L\*L array which is the covariance matrix #*

68 *##################################################################*

69 **def** calculate\_covariance\_matrix(mean\_adjusted\_faces):

70 num\_of\_faces = mean\_adjusted\_faces.shape[0]

71 first\_face = mean\_adjusted\_faces[0] *#used for initializing AAT\_face\_summation shape*

72 AAT = np.zeros((first\_face.shape[0], first\_face.shape[0])) *# where A is a mean adjusted face matrix*

73

74 *# Used 3D arrays as the computation time is almost identical this way, however I did read through the quicker method in the paper they used*

75 *# (they used 2D arrays) and understand it (i.e. why it's quicker and how it results in the same eigenvalues and scaled eigenvectors.*

76 *# Also, had already implemented 3D array method in all my code and it would have been a hassle to change everything*

77 *# to work with 2D dimensions that also would've been of minor benefit to computation time (and less intuitive for me - 3D arrays make more*

78 *# sense to me when dealing with images.*

79

80 **for** i **in** range(num\_of\_faces):

81 face = mean\_adjusted\_faces[i,:,:]

82 AAT = AAT + np.matmul(face, face.T) *#if face is 193\*162 then AAT\_face\_summation is 193\*193*

83

84 **return** AAT

85

86 *##################################################################*

87 *# input: L\*L array which is the covariance matrix #*

88 *# returns: L, L\*L array of eigenvalues and eigenvectors #*

89 *##################################################################*

90 **def** calculate\_eigenpairs(covariance\_matrix):

91 eigenvalues, eigenvectors = np.linalg.eig(covariance\_matrix)

92

93 **return** eigenvalues, eigenvectors

94

95 *########################################################################*

96 *# input: L array of eigenvalues #*

97 *# input: L\*L array of eigenvectors #*

98 *# input: num\_of\_faces is number of PCs for face space #*

99 *# returns: L\*L, L array of eigenvectors and eigenvalues for face space #*

100 *########################################################################*

101 **def** select\_face\_space(eigenvalues, eigenvectors, num\_of\_faces):

102 sorted\_eigenvalues = sorted(eigenvalues, reverse=True)

103 face\_space\_eigenvectors = np.zeros((eigenvectors.shape[0], num\_of\_faces)) *#each column is an eigenvector*

104 face\_space\_eigenvalues = np.zeros(num\_of\_faces)

105 **for** i **in** range(num\_of\_faces):

106 eigenvalue = sorted\_eigenvalues[i]

107 corresponding\_eigenvector\_index = np.where(eigenvalues == eigenvalue)[0][0]

108

109 corresponding\_eigenvector = eigenvectors[:, corresponding\_eigenvector\_index]

110

111 face\_space\_eigenvectors[:,i] = corresponding\_eigenvector

112 face\_space\_eigenvalues[i] = eigenvalue

113

114 **return** face\_space\_eigenvectors, face\_space\_eigenvalues

115

116 *########################################################################*

117 *# input: L array of eigenvalues for face space #*

118 *# return: no return, just creates bar plot for singular values #*

119 *########################################################################*

120 **def** create\_principal\_component\_bar\_plot(face\_space\_eigenvalues):

121 number\_of\_eigenvalues = face\_space\_eigenvalues.shape[0]

122 singular\_values = np.sqrt(face\_space\_eigenvalues)

123 eigenvalue\_index\_array = np.linspace(1, number\_of\_eigenvalues, number\_of\_eigenvalues)

124 plt.bar(eigenvalue\_index\_array, singular\_values)

125 plt.xlabel('Singular Value')

126 plt.ylabel('Singlular Value Magnitude')

127 plt.show()

128

129 *#########################################################################*

130 *# input (face\_to\_reconstruct): L\*N image to reconstruct using face space#*

131 *# input (face\_space\_eigenvectors): L\*L array of face space eigenvectors #*

132 *# input (mean\_face): L\*N mean face (constructed from 190 neutral faces) #*

133 *# returns: L\*N reconstructed image #*

134 *#########################################################################*

135 **def** principal\_component\_image\_reconstruction(face\_to\_reconstruct, face\_space\_eigenvectors, mean\_face): *#takes in a single image and reconstructs it by using principal components*

136 number\_of\_faces = face\_space\_eigenvectors.shape[1]

137 reconstructed\_image = np.zeros(face\_to\_reconstruct.shape)

138 mean\_adjusted\_face = face\_to\_reconstruct - mean\_face

139 **for** i **in** range(number\_of\_faces):

140 eigenvector = face\_space\_eigenvectors[:,i].reshape((face\_space\_eigenvectors[:,i].shape[0],1))

141 w = np.matmul(eigenvector.T, mean\_adjusted\_face)

142 w\_u = np.matmul(eigenvector, w)

143 reconstructed\_image = reconstructed\_image + w\_u

144

145 reconstructed\_image = reconstructed\_image + mean\_face

146 **return** reconstructed\_image

147

148 *#########################################################################*

149 *# input (image\_reconstruction): L\*N reconstructed image #*

150 *# input (face\_to\_reconstruct): original image used for reconstruction #*

151 *# returns: Mean Squared Error of inputs #*

152 *#########################################################################*

153 **def** calculate\_MSE(image\_reconstruction, face\_to\_reconstruct):

154 L = image\_reconstruction.shape[0]

155 N = image\_reconstruction.shape[1]

156

157 MSE = np.sum(np.square(image\_reconstruction - face\_to\_reconstruct))/(L\*N)

158

159 **return** MSE

160

161 *#########################################################################*

162 *# input: L\*N rgb image #*

163 *# returns: L\*N gray image #*

164 *#########################################################################*

165 **def** rgb2gray(image):

166 gray\_image = np.zeros((image.shape[0], image.shape[1]))

167 gray\_image[:,:] = image[:,:,0]\*0.2989 + image[:,:,1]\*0.5870 + image[:,:,2]\*0.1140

168

169 **return** gray\_image

170

171 *#########################################################################*

172 *# input: L\*L array of eigenvectors #*

173 *# returns: L array of cumulative significance used for plotting #*

174 *#########################################################################*

175 **def** calculate\_significance\_of\_eigenvalues(eigenvalues):

176 total\_sum = sum(eigenvalues)

177 cumulative\_significance = np.zeros(eigenvalues.shape[0])

178 **for** i **in** range(eigenvalues.shape[0]):

179 cumulative\_significance[i] = sum(eigenvalues[:i])/total\_sum

180 **return** cumulative\_significance

181

182 *# Folder used to create Principal Components*

183 face\_folder = 'neutral\_faces'

184

185 *# Folder used to access images to reconstruct*

186 face\_folder\_reconstruct = 'neutral\_faces'

187 *#face\_folder\_reconstruct = 'smiling\_faces'*

188

189

190 *# Creating absolute paths*

191 face\_folder\_path = os.path.join(os.getcwd(), face\_folder)

192 face\_folder\_path\_reconstruct = os.path.join(os.getcwd(), face\_folder\_reconstruct)

193

194 *#### Below are all used one at a time for different reconstructions ###*

195 *#face\_to\_reconstruct = mpltimg.imread(os.path.join(face\_folder\_path\_reconstruct, '4b.jpg'))*

196 *#face\_to\_reconstruct = mpltimg.imread(os.path.join(face\_folder\_path\_reconstruct, '195a.jpg')) #to be used later to test MSE of image reconstruction*

197 *#face\_to\_reconstruct = rgb2gray(mpltimg.imread(os.path.join(os.getcwd(),'tesla\_cybertruck.jpg')))*

198 face\_to\_reconstruct = mpltimg.imread(os.path.join(os.getcwd(),'neutral\_rotate\_45.jpg'))

199

200 *###### Code to calculate Principal Components ######*

201 faces\_vector = vectorize\_face\_folder(face\_folder\_path=face\_folder\_path)

202 mean\_face = calculate\_mean\_face(faces\_vector=faces\_vector)

203

204 mean\_adjusted\_faces = calculate\_mean\_adjusted\_faces(faces\_vector=faces\_vector, mean\_face=mean\_face)

205 covariance\_matrix = calculate\_covariance\_matrix(mean\_adjusted\_faces=mean\_adjusted\_faces)

206 eigenvalues, eigenvectors = calculate\_eigenpairs(covariance\_matrix=covariance\_matrix) *#eigenvectors have euclidean norm of 1*

207

208 *#99% significance occurs at 54*

209 num\_of\_PC = 54

210 face\_space\_eigenvectors, face\_space\_eigenvalues = select\_face\_space(eigenvalues=eigenvalues, eigenvectors=eigenvectors, num\_of\_faces=num\_of\_PC) *#max value for num\_of\_faces is total number of eigenvectors, number is chosen heuristically*

211 create\_principal\_component\_bar\_plot(face\_space\_eigenvalues=face\_space\_eigenvalues)

212 cumulative\_significance = calculate\_significance\_of\_eigenvalues(eigenvalues=face\_space\_eigenvalues)

213 *###############################################################################################################3*

214

215 *# Uncomment code to plot cumulative significance chart - only required once to determine how many principal components to used*

216 *# for the project. 99% of the sum of magnitudes of all eigenvalues is obtained at eigenvalue 54. Therefore, 54 principal components*

217 *# are used.*

218

219 indices = np.linspace(1, cumulative\_significance.shape[0], cumulative\_significance.shape[0])

220 plt.plot(indices, cumulative\_significance)

221 plt.plot(54,0.99,'or')

222 plt.xlabel('Principal Component')

223 plt.ylabel('Cumulative Significance')

224 plt.show()

225

226 *# Reconstruct image*

227 image\_reconstruction = principal\_component\_image\_reconstruction(face\_to\_reconstruct=face\_to\_reconstruct, face\_space\_eigenvectors=face\_space\_eigenvectors, mean\_face=mean\_face)

228

229 *# Plotting original and reconstructed image next to each other*

230 f = plt.figure()

231 f.add\_subplot(1,2,1)

232 plt.imshow(face\_to\_reconstruct, cmap='gray')

233 plt.xlabel('Original')

234 f.add\_subplot(1,2,2)

235 plt.imshow(image\_reconstruction, cmap='gray')

236 plt.xlabel('Reconstructed')

237 plt.show()

238

239 *###### Uncomment code to calculate MSE and create plots of MSE vs number of principal components #######*

240

241 MSE = calculate\_MSE(image\_reconstruction=image\_reconstruction, face\_to\_reconstruct=face\_to\_reconstruct)

242 PC\_indices = np.linspace(1,num\_of\_PC,num\_of\_PC)

243 MSE\_array = np.zeros(num\_of\_PC)

244

245 **for** i **in** range(num\_of\_PC):

246 face\_space\_eigenvectors, face\_space\_eigenvalues = select\_face\_space(eigenvalues=eigenvalues, eigenvectors=eigenvectors, num\_of\_faces=i)

247 image\_reconstruction = principal\_component\_image\_reconstruction(face\_to\_reconstruct=face\_to\_reconstruct, face\_space\_eigenvectors=face\_space\_eigenvectors, mean\_face=mean\_face)

248 MSE = calculate\_MSE(image\_reconstruction=image\_reconstruction, face\_to\_reconstruct=face\_to\_reconstruct)

249 MSE\_array[i] = MSE

250

251 plt.plot(PC\_indices, np.sqrt(MSE\_array))

252 plt.xlabel('Number of Principal Components')

253 plt.ylabel('Square root of MSE')

254 plt.show()