MachineLearningFinalProject_FacialRecognition

December 13, 2021

1 Task

Facial recognition technology has become the norm in this modern-era. It has become a mundane thing, as everybody who has an iPhone can attest to. Indeed, Apple's facial recognition technology is also based on deep learning [2], furthermore, its implementation is localized on an iPhone (impressively enough).

Now Apple needed to overcome the challenges of localization due to the limitations of the iPhone (whether it be the CPU, GPU or the limited RAM) [2]. The fact still holds that based on Apple's implementation, an implementation is indeed possible, which perhaps can be improved upon with the aid of better computing power. In this particular implementation, however, we will be specifically using the Yale Facial Dataset [1] as our input and our output will classify the images based on which individual/subject is in the image, for example it will output 1 for an image of individual 1, 2 for an image with individual 2, ETC. all the way up to 15 for images of individual 15.

Deliverables Must accomplish 1. 50 percent accuracy from test data using convolution neural networks or other models we have used in class 2. Augmented data using blur and cropped techniques 3. Extract skin tone as one of the features used in the neural network

Expect to accomplish 1. Implements random sampling from data when training model 2. 65 percent prediction accuracy on test data 3. Increase Data Set by adding pictures of more individuals (for a total of 20 individuals)

Would like to accomplish 1. 80 percent prediction accuracy on test data 2. Include Gabor Filters or other image processing techniques as a part of our model 3. Increase Data Set by adding pictures of more individuals (for a total of 30 individuals)

2 Data

2.1 Approach

We will be training this dataset as a classification problem. Given a picture we would like the model to be able to predict which of the 15 people in the dataset it matches to. We plan to do our classification using a neural network model. We plan to start out by using convolutional networks and a squared loss and then try out other types of models as well to see what works best.

2.2 Original Data

Our data set is taken from the Yale Face Recognition Data base. This data base consists of 15 different subjects, and each subject has 11 different images that are different facial expressions

resulting in 165 total images. We will split up the training and testing data sets so that approximately 3/4 of the images are in the training set and 1/4 are in the testing set. So, for each subject, a randomly picked 7 of the 11 images will be the training set and the remaining 4 will be in the testing set. The resolution of each of these images is 243x320. For data augmentation, we will perform a couple different augmentations. The two main augmentations are blurring the images and mirroring the images. This allows our data set to potentially grow by 4 times, giving us more data to train the model with.

2.2.1 Display Images

[16]:
```
import numpy as np
from PIL import Image
import os
from imageprocessing_helper_functions import flatten
from sklearn.model_selection import train_test_split

#yale images
images = [f for f in os.listdir('yalefaces') if f != ".DS_Store"]
X = [Image.open("./yalefaces/" + image) for image in images]

#divide into test,dev, and train (test, train, dev)
labels = []
for image in images:
    if "subject11" in image:
        labels.append(11)
    elif "subject02" in image:
        labels.append(2)
    elif "subject03" in image:
        labels.append(3)
    elif "subject04" in image:
        labels.append(4)
    elif "subject05" in image:
        labels.append(5)
    elif "subject06" in image:
        labels.append(6)
    elif "subject07" in image:
        labels.append(7)
    elif "subject08" in image:
        labels.append(8)
    elif "subject09" in image:
        labels.append(9)
    elif "subject10" in image:
        labels.append(10)
    elif "subject15" in image:
        labels.append(15)
    elif "subject12" in image:
        labels.append(12)
```

```
    elif "subject13" in image:
        labels.append(13)
    elif "subject14" in image:
        labels.append(14)
    elif "subject01" in image:
        labels.append(1)

labels = np.array(labels)
print("Sample Images")

print("Individual %d"%(labels[0]))
display(X[0])
print("Individual %d"%(labels[1]))
display(X[1])
print("Individual %d"%(labels[2]))
display(X[2])

label = np.array(labels)
features = flatten(X)


train_features, dev_features, train_labels, dev_labels =␣
 ↪train_test_split(features, labels, test_size=0.4, random_state=42)
dev_features, test_features, dev_labels, test_labels =␣
 ↪train_test_split(dev_features, dev_labels, test_size=0.1, random_state=42)
```

```
Sample Images
Individual 8
```

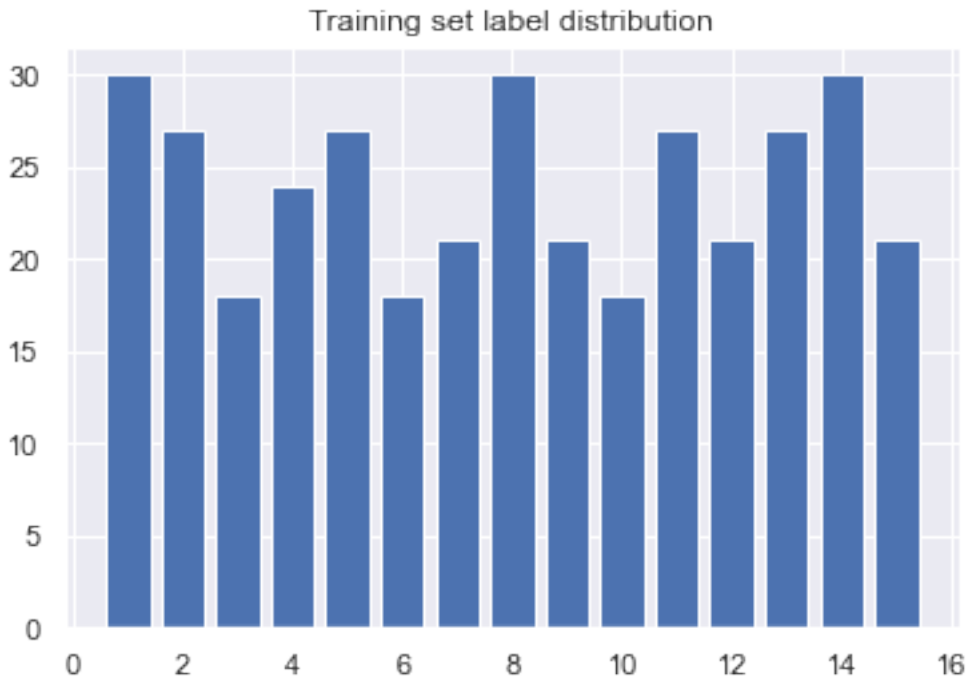
Individual 9

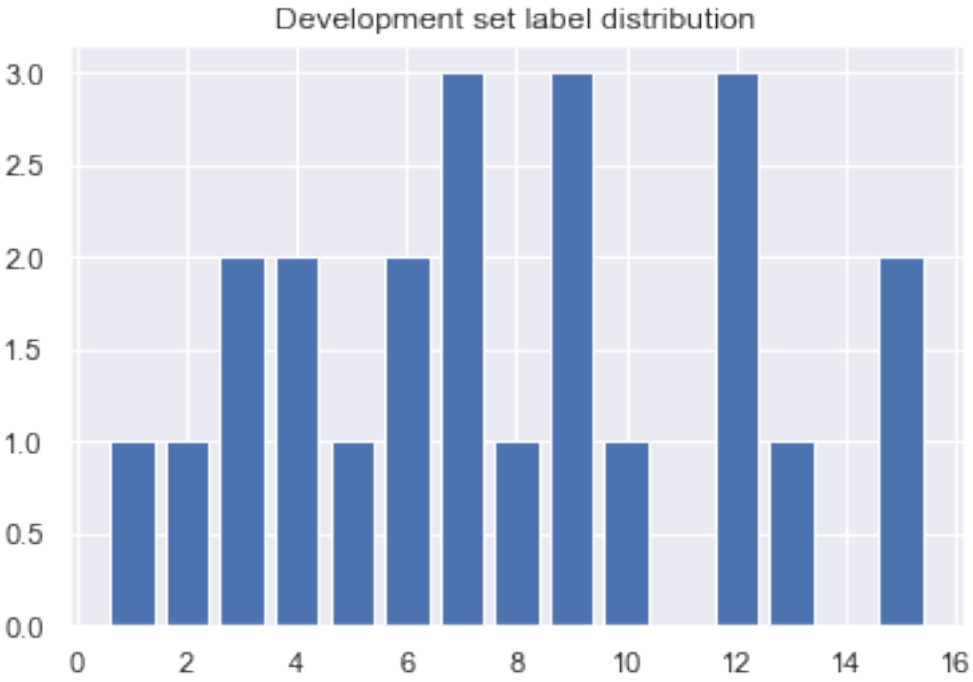
Individual 2

2.2.2 Training Set and Test Set Label Distribution

[15]:
```
# imports
from collections import defaultdict
import numpy as np
from data import load
import matplotlib.pyplot as plt
%matplotlib inline

counts = defaultdict(int)

for label in train_labels:
    counts[label] += 1

plt.figure()
plt.title("Training set label distribution")

k = counts.keys()
v = counts.values()
plt.bar(list(k), height=list(v))

dev_counts = defaultdict(int)

for label in dev_labels:
    dev_counts[label] += 1

plt.figure()
plt.title("Development set label distribution")
dk = dev_counts.keys()
dv = dev_counts.values()
plt.bar(list(dk), height=list(dv))
```

[15]:
```
<BarContainer object of 13 artists>
```





Label distribution not quite uniform so might need to look into domain adaptation techniques, but this is not implemented in this particular method/experiment

2.3 Data Augmentation

There are not enough images in the training set for a model to be train to a sufficient degree that it will be able to predict an image well with some labels having as few as 2 images, thus, the data must be augmented. In this particular case, Mirroring and Blurring operations (only on the original) are employed to increase the training dataset by 3 times (*did not blur mirrored image)

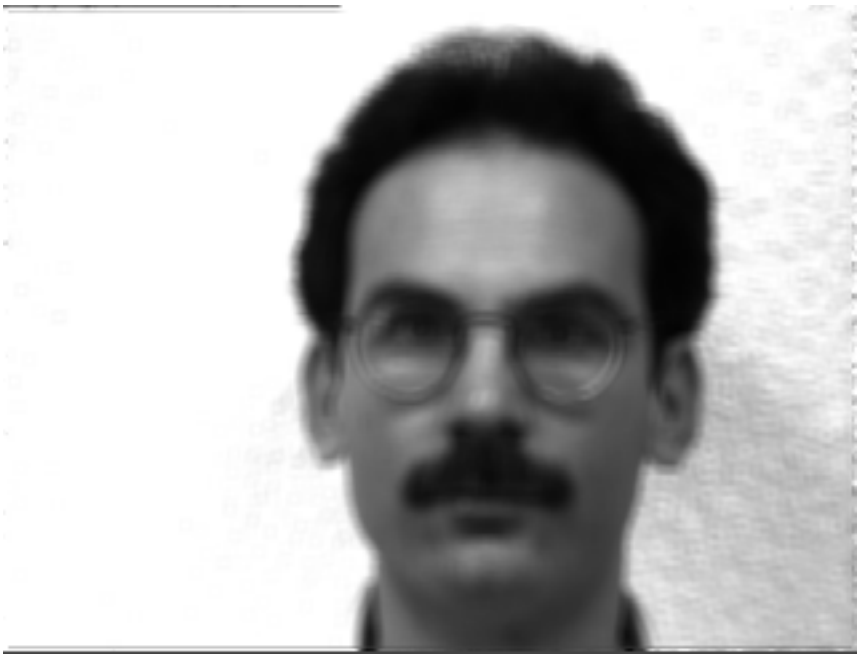
2.3.1 Blurring

[4]:
```
from imageprocessing_helper_functions import blurred, mirrored

X_blurred, label_blurred = blurred(X, labels)
label_blurred = np.array(label_blurred)

print("Sample blurred Images")

print("Blurred Individual %d"%(labels[0]))
display(X_blurred[0])
print("Blurred Individual %d"%(labels[1]))
display(X_blurred[1])
print("Blurred Individual %d"%(labels[2]))
display(X_blurred[2])
```

```
Sample blurred Images
Blurred Individual 8
```

```
Blurred Individual 9
```

Blurred Individual 2



2.3.2 Mirroring

[5]:
```
X_mirrored, label_mirrored = mirrored(X, labels)
label_mirrored = np.array(label_mirrored)

print("Sample mirrored Images")

print("Mirrored Individual %d"%(labels[0]))
display(X_mirrored[0])
print("Mirrored Individual %d"%(labels[1]))
display(X_mirrored[1])
print("Mirrored Individual %d"%(labels[2]))
display(X_mirrored[2])
```

```
Sample mirrored Images
Mirrored Individual 8
```

```
Mirrored Individual 9
```

Mirrored Individual 2

2.3.3 Eigenfaces

Each image is originally a 243 by 320 dimension images with 8 bit gray scale values. When a convolutional neural network was used to train on the data, the runtime was found out to be too

slow for any significant progress to be made.

A valid comparison from past experience with dealing with image data is from HW3, using the Fashion MNIST dataset, the images in that data are 28 by 28, however, they are grayscale, similar to that of the Yale Face database.

[9]:
```
size_yale = 234 * 320
size_Fashion = 28 * 28

print("size of yale images is %d pixels, and size of fashion images is %d␣
 ↪pixels"%(size_yale, size_Fashion) )

print("yale images are %0.2f times bigger than fashion images"%(size_yale/
 ↪size_Fashion))
```

```
size of yale images is 74880 pixels, and size of fashion images is 784 pixels
yale images are 95.51 times bigger than fashion images
```

This means that one Yale Database image is equivalent in size to around 95 Fashion images meaning that training on 100 Yale Face images is equivalent to training on 9500 of the Fashion Images.

In the past HW3 training batch size is recommended to be at 100 which at this size training still took a long time for a sufficiently deep neural network which is required to achieve good result, however, the equivalent of 100 images in HW3 is demonstrate to be 1 image in this dataset size-wise, thus, it is clear that a good result can never be achieved at this size since for sure not all outcomes will be explored (15 outcomes as 15 people to be identified) (at least initially)

Thus, PCA technique (Principle Component Analysis) is used to retain only the most significant axes of variability (eigenvectors of the empirical covariance matrix). This enable us to reduce from 74480 features to only 100 which is even lesser than the features in HW3

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn import decomposition
import csv
import argparse as ap
import random
from data import load
import numpy as np



# load data
train_data, train_labels = load("yalefaces_npy", split="train")


pca = decomposition.PCA()
pca.fit(train_data)
train_data = pca.transform(train_data)
```

```
fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(pca.components_[i].reshape(243, 320), cmap='bone')
```

3 Model

Initially, approach was to solely use the convolutional neural network on the data, however, this might be a naive assumption where images are assumed to be best handled only by a neural network. Furthermore, after the PCA decomposition, data's features are reduced in a way that it might be more straightforward to work directly with the features produced from the PCA decomposition without further reformulation.

Thus, many models are employed, in an effort to find the best model for this particular dataset and in the application of facial recognition.

3.1 Convolutional Neural Network Model

Description We first attempted to train the model using the same neural network that had been best on the data in class. We also used the PCA technique here. Here is the snippet of code showing the layout of the neural net:

class BestNN(torch.nn.Module): def **init**(self, n1_channels, n1_kernel, n2_channels, n2_kernel, pool1, n3_channels, n3_kernel, n4_channels, n4_kernel, pool2, linear_features): super(BestNN, self).___init___() self.sequence1 = nn.Sequential(nn.Conv2d(1, n1_channels, n1_kernel), nn.BatchNorm2d(n1_channels), nn.ReLU(), nn.Conv2d(n1_channels, n2_channels, n2_kernel), nn.BatchNorm2d(n2_channels), nn.ReLU(),)

```
def forward(self, x):
    x = x.reshape([x.size(0), 1, 10, 12]) #reshape to 2d images
    x = self.sequence1(x)
```

```
        self.sequence2 = nn.Sequential(
            nn.MaxPool2d(x.size(2))
            )
        x = self.sequence2(x)
        x = x.reshape(x.size(0), -1) #resize to 1 vector with length of 10 for each example in bat
        return x
```

[18]:
```
!python main.py train --data-dir ./faces --log-file ./best-logs.csv␣
↪--model-save ./best.torch --model best
```

```
On step 0: Train loss 4.836712837219238 | Dev acc is 0.011342155009451797
On step 100: Train loss 0.6206836104393005 | Dev acc is 0.08506616257088846

main.py:163: DeprecationWarning: `np.float` is a deprecated alias for the
builtin `float`. To silence this warning, use `float` by itself. Doing this will
not modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return (y == y_hat).astype(np.float).mean()


On step 200: Train loss 0.303556889295578 | Dev acc is 0.0831758034026465
On step 300: Train loss 0.2471441775560379 | Dev acc is 0.07372400756143667
On step 400: Train loss 0.07811369746923447 | Dev acc is 0.07939508506616257
On step 500: Train loss 0.15538126230239868 | Dev acc is 0.08128544423440454
On step 600: Train loss 0.07546880841255188 | Dev acc is 0.0831758034026465
On step 700: Train loss 0.06938668340444565 | Dev acc is 0.06994328922495274
On step 800: Train loss 0.13126762211322784 | Dev acc is 0.08506616257088846
On step 900: Train loss 0.13665412366390228 | Dev acc is 0.07372400756143667
On step 1000: Train loss 0.051192328333854675 | Dev acc is 0.07939508506616257
On step 1100: Train loss 0.07660456746816635 | Dev acc is 0.07939508506616257
On step 1200: Train loss 0.027586642652750015 | Dev acc is 0.0888468809073724
On step 1300: Train loss 0.10922179371118546 | Dev acc is 0.07561436672967864
On step 1400: Train loss 0.04241551086306572 | Dev acc is 0.08506616257088846
On step 1500: Train loss 0.03338203579187393 | Dev acc is 0.07183364839319471
On step 1600: Train loss 0.027677560225129128 | Dev acc is 0.07939508506616257
On step 1700: Train loss 0.011385808698832989 | Dev acc is 0.07561436672967864
On step 1800: Train loss 0.01919754408299923 | Dev acc is 0.08695652173913043
On step 1900: Train loss 0.018745632842183113 | Dev acc is 0.08695652173913043
On step 2000: Train loss 0.010921601206064224 | Dev acc is 0.07939508506616257
On step 2100: Train loss 0.00676735769957304 | Dev acc is 0.07372400756143667
On step 2200: Train loss 0.02541307546198368 | Dev acc is 0.0831758034026465
On step 2300: Train loss 0.014097006991505623 | Dev acc is 0.07750472589792061
On step 2400: Train loss 0.007255523931235075 | Dev acc is 0.0831758034026465
On step 2500: Train loss 0.013189361430704594 | Dev acc is 0.08506616257088846
On step 2600: Train loss 0.012971659190952778 | Dev acc is 0.08128544423440454
On step 2700: Train loss 0.02167471870779991 | Dev acc is 0.07561436672967864
On step 2800: Train loss 0.006901147309690714 | Dev acc is 0.07939508506616257
```

```
On step 2900: Train loss 0.031697336584329605 | Dev acc is 0.08506616257088846
On step 3000: Train loss 0.013159221038222313 | Dev acc is 0.09073724007561437
On step 3100: Train loss 0.005688220728188753 | Dev acc is 0.09073724007561437
On step 3200: Train loss 0.0035058781504631042 | Dev acc is 0.08506616257088846
On step 3300: Train loss 0.004492524079978466 | Dev acc is 0.0831758034026465
On step 3400: Train loss 0.005127301439642906 | Dev acc is 0.07750472589792061
Done training. Saving model at ./best.torch
```

Description We thought that our dataset may have been too small in the first run through because the number of components is limited by the number of training images. Therefore we next tried running it on the data set including the blurred images as well as the original images.

[17]:
```
!python main.py train --data-dir ./facesmod --log-file ./best-logs.csv␣
↪--model-save ./best.torch --model best
```

```
On step 0: Train loss 3.995718479156494 | Dev acc is 0.02835538752362949

main.py:163: DeprecationWarning: `np.float` is a deprecated alias for the
builtin `float`. To silence this warning, use `float` by itself. Doing this will
not modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return (y == y_hat).astype(np.float).mean()


On step 100: Train loss 1.2741748094558716 | Dev acc is 0.0888468809073724
On step 200: Train loss 0.47337210178375244 | Dev acc is 0.08695652173913043
On step 300: Train loss 0.34404173493385315 | Dev acc is 0.08506616257088846
On step 400: Train loss 0.12064637988805771 | Dev acc is 0.08506616257088846
On step 500: Train loss 0.279146283864975 | Dev acc is 0.08128544423440454
On step 600: Train loss 0.37741485238075256 | Dev acc is 0.08128544423440454
On step 700: Train loss 0.044896405190229416 | Dev acc is 0.08506616257088846
On step 800: Train loss 0.1718878149986267 | Dev acc is 0.08506616257088846
On step 900: Train loss 0.10579290241003036 | Dev acc is 0.08695652173913043
On step 1000: Train loss 0.05681375786662102 | Dev acc is 0.08128544423440454
On step 1100: Train loss 0.22517815232276917 | Dev acc is 0.08695652173913043
On step 1200: Train loss 0.02648814581334591 | Dev acc is 0.09262759924385633
On step 1300: Train loss 0.022949127480387688 | Dev acc is 0.08506616257088846
On step 1400: Train loss 0.046089544892311096 | Dev acc is 0.08695652173913043
On step 1500: Train loss 0.31034284830093384 | Dev acc is 0.08695652173913043
On step 1600: Train loss 0.020272407680749893 | Dev acc is 0.09073724007561437
On step 1700: Train loss 0.03166045993566513 | Dev acc is 0.08695652173913043
On step 1800: Train loss 0.015138864517211914 | Dev acc is 0.09073724007561437
On step 1900: Train loss 0.023695295676589012 | Dev acc is 0.09073724007561437
On step 2000: Train loss 0.02131161279976368 | Dev acc is 0.09262759924385633
On step 2100: Train loss 0.05489877238869667 | Dev acc is 0.09073724007561437
On step 2200: Train loss 0.05578276887536049 | Dev acc is 0.08695652173913043
On step 2300: Train loss 0.035168189555540657 | Dev acc is 0.0888468809073724
```

```
On step 2400: Train loss 0.005519387777894735 | Dev acc is 0.0888468809073724
On step 2500: Train loss 0.007317674346268177 | Dev acc is 0.08506616257088846
On step 2600: Train loss 0.005314027424901724 | Dev acc is 0.08695652173913043
On step 2700: Train loss 0.008152520284056664 | Dev acc is 0.08695652173913043
On step 2800: Train loss 0.037933025509119034 | Dev acc is 0.08506616257088846
On step 2900: Train loss 0.7185214161872864 | Dev acc is 0.08695652173913043
On step 3000: Train loss 0.02334470860660076 | Dev acc is 0.08695652173913043
On step 3100: Train loss 0.014900526031851768 | Dev acc is 0.0888468809073724
On step 3200: Train loss 0.01082578208297491 | Dev acc is 0.08506616257088846
On step 3300: Train loss 0.008521516807377338 | Dev acc is 0.08506616257088846
On step 3400: Train loss 0.009070780128240585 | Dev acc is 0.08506616257088846
Done training. Saving model at ./best.torch
```

Analysis While both classifiers performed slightly above average, and adding more data increased the accuracy slightly, neither classifier worked well at all as they both had very low accuracy. We therefore decided to move away from neural networks and to try other methods instead.

3.2 MLP Classifer

Description MLP(multilayer perceptron) classifier is a feedforward artificial neural network method in supervised learning. It maps a set of input vectors to a set of output vectors. MLP can be thought of as a directed graph consisting of multiple layers of nodes, each of which is fully connected to the next layer. Except for the input nodes, each neuron (or processing unit) is with a nonlinear activation function. A method called backpropagation algorithm is often used to train MLP. It first learns, then store data based on weights, and uses algorithms to adjust weights and reduce bias during training, where bias is the difference between actual and predicted values. The main advantage of using MLP is first, it is able to solve complex problems quickly. Second, as the improvement of perceptron, it overcomes the weakness that perceptron cannot recognize linear non-fractional data. Since this model is relatively simple, we apply it first as a benchmark.

[22]:
```
import numpy as np
from data import load
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from PIL import Image as im
from imageprocessing_helper_functions import blurred, mirrored, flatten
from sklearn.metrics import confusion_matrix



train_data, train_labels = load("yalefaces_npy", split="train")
dev_data, dev_labels = load("yalefaces_npy", split="dev")

X = []

for i in range(0, np.shape(train_data)[0]):
    array = np.reshape(train_data[i,:], (243, 320))
```

```
    data = im.fromarray(array, mode = 'L')
    X.append(data)

label = train_labels
features = train_data

#blur images
X_blurred, label_blurred = blurred(X, label)
label_blurred = np.array(label_blurred)
features_blurred = flatten(X_blurred)

#mirror images
X_mirrored, label_mirrored = mirrored(X, label)
label_mirrored = np.array(label_mirrored)
features_mirrored = flatten(X_mirrored)

#combined
features_combined = np.concatenate((features, features_blurred))
labels_combined = np.concatenate((label, label_blurred))
features_combined = np.concatenate((features_combined, features_mirrored))
labels_combined = np.concatenate((labels_combined, label_mirrored))

# Compute a PCA
n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(features_combined)

# apply PCA transformation
X_train_pca = pca.transform(features_combined)
X_test_pca = pca.transform(dev_data)

train_labels = np.ravel(labels_combined)
dev_labels = np.ravel(dev_labels)


print("Fitting the classifier to the training set")
clf = MLPClassifier(hidden_layer_sizes=(1024,), batch_size=100, verbose=True,␣
 ↪early_stopping=True).fit(X_train_pca, train_labels)

y_pred = clf.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Fitting the classifier to the training set
Iteration 1, loss = 2.78551852
Validation score: 0.194444
Iteration 2, loss = 2.18430681
Validation score: 0.361111
Iteration 3, loss = 1.80577314
Validation score: 0.416667
```

```
Iteration 4, loss = 1.53904982
Validation score: 0.388889
Iteration 5, loss = 1.35802617
Validation score: 0.388889
Iteration 6, loss = 1.23808309
Validation score: 0.416667
Iteration 7, loss = 1.14995076
Validation score: 0.444444
Iteration 8, loss = 1.08345135
Validation score: 0.444444
Iteration 9, loss = 1.02763842
Validation score: 0.444444
Iteration 10, loss = 0.98306645
Validation score: 0.472222
Iteration 11, loss = 0.94341329
Validation score: 0.527778
Iteration 12, loss = 0.90880713
Validation score: 0.527778
Iteration 13, loss = 0.87964734
Validation score: 0.527778
Iteration 14, loss = 0.85185849
Validation score: 0.527778
Iteration 15, loss = 0.82746436
Validation score: 0.527778
Iteration 16, loss = 0.80465812
Validation score: 0.555556
Iteration 17, loss = 0.78423307
Validation score: 0.555556
Iteration 18, loss = 0.76601485
Validation score: 0.555556
Iteration 19, loss = 0.74728210
Validation score: 0.527778
Iteration 20, loss = 0.73325391
Validation score: 0.527778
Iteration 21, loss = 0.71771546
Validation score: 0.527778
Iteration 22, loss = 0.70266519
Validation score: 0.555556
Iteration 23, loss = 0.68832712
Validation score: 0.555556
Iteration 24, loss = 0.67521792
Validation score: 0.583333
Iteration 25, loss = 0.66365427
Validation score: 0.583333
Iteration 26, loss = 0.65373542
Validation score: 0.583333
Iteration 27, loss = 0.64352202
Validation score: 0.583333
```

```
Iteration 28, loss = 0.63251723
Validation score: 0.583333
Iteration 29, loss = 0.62211617
Validation score: 0.583333
Iteration 30, loss = 0.61239350
Validation score: 0.583333
Iteration 31, loss = 0.60299304
Validation score: 0.583333
Iteration 32, loss = 0.59427957
Validation score: 0.583333
Iteration 33, loss = 0.58414138
Validation score: 0.583333
Iteration 34, loss = 0.57650217
Validation score: 0.583333
Iteration 35, loss = 0.56740977
Validation score: 0.583333
Validation score did not improve more than tol=0.000100 for 10 consecutive
epochs. Stopping.
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         1
         2.0       1.00      1.00      1.00         1
         3.0       1.00      1.00      1.00         2
         4.0       1.00      1.00      1.00         2
         5.0       0.00      0.00      0.00         1
         6.0       1.00      1.00      1.00         2
         7.0       1.00      1.00      1.00         3
         8.0       0.33      1.00      0.50         1
         9.0       1.00      0.67      0.80         3
        10.0       1.00      1.00      1.00         1
        12.0       1.00      1.00      1.00         3
        13.0       1.00      1.00      1.00         1
        15.0       1.00      1.00      1.00         2

    accuracy                           0.91        23
   macro avg       0.87      0.90      0.87        23
weighted avg       0.93      0.91      0.91        23
```

- Accuracy = (TP + TN)/(TP + TN + FP + FN)
- Precision (a.k.a. Sensitivity)= TP/(TP + FP)
- Recall (a.k.a. True Positive Rate) = TP/(TP + FN)
- Specificity = TN/(TN + FP)
- False Positive Rate = FP/(FP + TN)
- F1 Score = 2*(Precision* Recall)/(Precision + Recall)

- Balanced Accuracy = (Sensitivity + Specificity)/2

HyperParameter Search To attempt to achieve higher performance, we performed hyperparameter tuning on the following hyperparameters for the MLP classifier: activation function, alpha, and beta_1. This was done by first viewing the default values for these parameters in the sklearn model, and then performing GridSearchCV on values surrounding those default values for each hyperparameter.

[23]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import svm
import warnings
warnings.filterwarnings('ignore')

print("Current Parameters are: \n")
print(clf.get_params())

print("Hyperparameters to tune: activation, alpha, beta_1\n")
param_grid = {'activation': ['relu', 'tanh'],
              'alpha': [1, 0.1, 0.01, 0.001, .0001],
              'beta_1': [0.9, .92, .94, .96, .98, 1],
             }

#clf_new = MLPClassifier(max_iter=100)

grid_search = GridSearchCV(estimator = clf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train_pca, train_labels)
print(grid_search.best_params_)


y_pred = grid_search.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Current Parameters are:

{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 100, 'beta_1': 0.9,
'beta_2': 0.999, 'early_stopping': True, 'epsilon': 1e-08, 'hidden_layer_sizes':
(1024,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun':
15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change': 10,
'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': None, 'shuffle':
True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose':
True, 'warm_start': False}
Hyperparameters to tune: activation, alpha, beta_1

Fitting 3 folds for each of 60 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    6.2s
```

```
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   29.2s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:   33.7s finished

Iteration 1, loss = 4.02610395
Validation score: 0.250000
Iteration 2, loss = 3.26058359
Validation score: 0.277778
Iteration 3, loss = 2.80661104
Validation score: 0.305556
Iteration 4, loss = 2.50879020
Validation score: 0.305556
Iteration 5, loss = 2.32620904
Validation score: 0.333333
Iteration 6, loss = 2.19963440
Validation score: 0.333333
Iteration 7, loss = 2.10855824
Validation score: 0.333333
Iteration 8, loss = 2.03387639
Validation score: 0.388889
Iteration 9, loss = 1.97025924
Validation score: 0.388889
Iteration 10, loss = 1.91111925
Validation score: 0.388889
Iteration 11, loss = 1.85788602
Validation score: 0.388889
Iteration 12, loss = 1.80462389
Validation score: 0.388889
Iteration 13, loss = 1.75334286
Validation score: 0.388889
Iteration 14, loss = 1.70222526
Validation score: 0.444444
Iteration 15, loss = 1.65372981
Validation score: 0.444444
Iteration 16, loss = 1.60723015
Validation score: 0.444444
Iteration 17, loss = 1.56227189
Validation score: 0.444444
Iteration 18, loss = 1.51966758
Validation score: 0.444444
Iteration 19, loss = 1.48000383
Validation score: 0.444444
Iteration 20, loss = 1.44204592
Validation score: 0.416667
Iteration 21, loss = 1.40584043
Validation score: 0.416667
Iteration 22, loss = 1.37196243
Validation score: 0.388889
Iteration 23, loss = 1.33920341
```

```
Validation score: 0.361111
Iteration 24, loss = 1.30887334
Validation score: 0.361111
Iteration 25, loss = 1.28087531
Validation score: 0.388889
Validation score did not improve more than tol=0.000100 for 10 consecutive
epochs. Stopping.
{'activation': 'tanh', 'alpha': 1, 'beta_1': 0.98}
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         1
         2.0       1.00      1.00      1.00         1
         3.0       1.00      1.00      1.00         2
         4.0       1.00      1.00      1.00         2
         5.0       1.00      1.00      1.00         1
         6.0       1.00      1.00      1.00         2
         7.0       0.75      1.00      0.86         3
         8.0       0.50      1.00      0.67         1
         9.0       1.00      0.67      0.80         3
        10.0       1.00      1.00      1.00         1
        12.0       1.00      1.00      1.00         3
        13.0       1.00      1.00      1.00         1
        15.0       1.00      0.50      0.67         2

    accuracy                           0.91        23
   macro avg       0.94      0.94      0.92        23
weighted avg       0.95      0.91      0.91        23
```

3.3 SVM Classifier

Description SVM is a supervised learning model that works in both classification and regression analysis problems. Given a set of observation, each training instance is marked as belonging to one or another of the two categories. SVM trains to classify a new instance to one of two categories of models, making it to be a binary linear classifier. The SVM model is to represent instances as points in space, so that the mapping separates instances of separate categories by the widest apparent interval possible, and then map new instances into the same space and predict the category based on which side of the interval they fall on. More importantly, if the raw data is not linearly seperable, kernel funciton is able to map non-linear separable data into a higher dimensional space which could make our data linear separable.

Implementation and Results

[27]:
```
from sklearn import svm
import numpy as np
from data import load
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
```

```
from PIL import Image as im
from imageprocessing_helper_functions import blurred, mirrored, flatten
from sklearn.metrics import confusion_matrix



train_data, train_labels = load("yalefaces_npy", split="train")
dev_data, dev_labels = load("yalefaces_npy", split="dev")

X = []

for i in range(0, np.shape(train_data)[0]):
    array = np.reshape(train_data[i,:], (243, 320))
    data = im.fromarray(array, mode = 'L')
    X.append(data)

label = train_labels
features = train_data

#blur images
X_blurred, label_blurred = blurred(X, label)
label_blurred = np.array(label_blurred)
features_blurred = flatten(X_blurred)

#mirror images
X_mirrored, label_mirrored = mirrored(X, label)
label_mirrored = np.array(label_mirrored)
features_mirrored = flatten(X_mirrored)

#combined
features_combined = np.concatenate((features, features_blurred))
labels_combined = np.concatenate((label, label_blurred))
features_combined = np.concatenate((features_combined, features_mirrored))
labels_combined = np.concatenate((labels_combined, label_mirrored))

# Compute a PCA
n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(features_combined)

# apply PCA transformation
X_train_pca = pca.transform(features_combined)
X_test_pca = pca.transform(dev_data)

train_labels = np.ravel(labels_combined)
dev_labels = np.ravel(dev_labels)
```

```
print("Fitting the classifier to the training set")
clf = svm.SVC(C=5, gamma=0.001).fit(X_train_pca, train_labels)

y_pred = clf.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Fitting the classifier to the training set
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         1
         2.0       1.00      1.00      1.00         1
         3.0       1.00      1.00      1.00         2
         4.0       1.00      1.00      1.00         2
         5.0       1.00      1.00      1.00         1
         6.0       1.00      1.00      1.00         2
         7.0       1.00      1.00      1.00         3
         8.0       0.33      1.00      0.50         1
         9.0       1.00      0.67      0.80         3
        10.0       1.00      1.00      1.00         1
        12.0       1.00      1.00      1.00         3
        13.0       1.00      1.00      1.00         1
        15.0       1.00      0.50      0.67         2

    accuracy                           0.91        23
   macro avg       0.95      0.94      0.92        23
weighted avg       0.97      0.91      0.92        23
```

- Accuracy = (TP + TN)/(TP + TN + FP + FN)
- Precision (a.k.a. Sensitivity)= TP/(TP + FP)
- Recall (a.k.a. True Positive Rate) = TP/(TP + FN)
- Specificity = TN/(TN + FP)
- False Positive Rate = FP/(FP + TN)
- F1 Score = 2*(Precision* Recall)/(Precision + Recall)
- Balanced Accuracy = (Sensitivity + Specificity)/2

HyperParameter Search To attempt to achieve higher performance, we performed hyperparameter tuning on the following hyperparameters for the SVM classifier: C, gamma. This was done by first viewing the default values for these parameters in the sklearn model, and then performing GridSearchCV on values surrounding those default values for each hyperparameter.

[28]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

print("Current Parameters are: \n")
```

```
print(clf.get_params())

print("Hyperparameters to tune: \n")
param_grid = {'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'gamma': [.1, .01, .005, .001, .0001],
              'kernel': ['rbf']}

#clf_new = svm.SVC()

grid_search = GridSearchCV(estimator = clf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train_pca, train_labels)
print(grid_search.best_params_)

y_pred = grid_search.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Current Parameters are:

{'C': 5, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0':
0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 0.001, 'kernel':
'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking':
True, 'tol': 0.001, 'verbose': False}
Hyperparameters to tune:

Fitting 3 folds for each of 50 candidates, totalling 150 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  92 tasks      | elapsed:    0.9s

{'C': 8, 'gamma': 0.001, 'kernel': 'rbf'}
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         1
         2.0       1.00      1.00      1.00         1
         3.0       1.00      1.00      1.00         2
         4.0       1.00      1.00      1.00         2
         5.0       1.00      1.00      1.00         1
         6.0       1.00      1.00      1.00         2
         7.0       1.00      1.00      1.00         3
         8.0       0.33      1.00      0.50         1
         9.0       1.00      0.67      0.80         3
        10.0       1.00      1.00      1.00         1
        12.0       1.00      1.00      1.00         3
        13.0       1.00      1.00      1.00         1
        15.0       1.00      0.50      0.67         2

    accuracy                           0.91        23
```

```
   macro avg       0.95      0.94      0.92        23
weighted avg       0.97      0.91      0.92        23


[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:    1.3s finished
```

3.4 Random Forest Classifier

Description The random forest classifier combines multiple decision trees into one classifier. This allows a very simple classification method to be mode more effective. Random forest classifiers often work well as built within existing libraries, therefore they are simple to test on data, and avoid having to spend alot of time tailoring a model to your specific needs. In previous studies Random Forest models have been shown to perform up to as good as SVM classifiers.

Implementation and Results

[37]:
```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from data import load
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
from PIL import Image as im
from imageprocessing_helper_functions import blurred, mirrored, flatten
from sklearn.metrics import confusion_matrix



train_data, train_labels = load("yalefaces_npy", split="train")
dev_data, dev_labels = load("yalefaces_npy", split="dev")

X = []

for i in range(0, np.shape(train_data)[0]):
    array = np.reshape(train_data[i,:], (243, 320))
    data = im.fromarray(array, mode = 'L')
    X.append(data)

label = train_labels
features = train_data

#blur images
X_blurred, label_blurred = blurred(X, label)
label_blurred = np.array(label_blurred)
features_blurred = flatten(X_blurred)

#mirror images
X_mirrored, label_mirrored = mirrored(X, label)
label_mirrored = np.array(label_mirrored)
features_mirrored = flatten(X_mirrored)
```

```
#combined
features_combined = np.concatenate((features, features_blurred))
labels_combined = np.concatenate((label, label_blurred))
features_combined = np.concatenate((features_combined, features_mirrored))
labels_combined = np.concatenate((labels_combined, label_mirrored))

# Compute a PCA
n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(features_combined)

# apply PCA transformation
X_train_pca = pca.transform(features_combined)
X_test_pca = pca.transform(dev_data)

train_labels = np.ravel(labels_combined)
dev_labels = np.ravel(dev_labels)


print("Fitting the classifier to the training set")
clf = RandomForestClassifier().fit(X_train_pca, train_labels)

y_pred = clf.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Fitting the classifier to the training set
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         1
         2.0       1.00      1.00      1.00         1
         3.0       0.50      0.50      0.50         2
         4.0       1.00      1.00      1.00         2
         5.0       0.00      0.00      0.00         1
         6.0       0.67      1.00      0.80         2
         7.0       1.00      1.00      1.00         3
         8.0       0.25      1.00      0.40         1
         9.0       1.00      0.67      0.80         3
        10.0       1.00      1.00      1.00         1
        12.0       1.00      0.33      0.50         3
        13.0       0.00      0.00      0.00         1
        14.0       0.00      0.00      0.00         0
        15.0       1.00      1.00      1.00         2

    accuracy                           0.74        23
   macro avg       0.67      0.68      0.64        23
weighted avg       0.81      0.74      0.73        23
```

HyperParameter Search To attempt to achieve higher performance, we performed hyperparameter tuning on the following hyperparameters for the random forest classifier: max_depth, max_features, min_samples_leaf, n_estimators. Note that this model has many more hyperparameters so the search was slightly larger than the previous models. This was done by first viewing the default values for these parameters in the sklearn model, and then performing GridSearchCV on values surrounding those default values for each hyperparameter.

[38]:
```
from sklearn.model_selection import GridSearchCV

print("Current Parameters are: \n")
print(clf.get_params())

print("Hyperparameters to tune: \n")
param_grid = {
    'bootstrap': [True],
    'max_depth': [None, 50, 70, 90, 110],
    'max_features': ['auto', 3],
    'min_samples_leaf': [1, 2, 3],
    'n_estimators': [100, 200, 300]
}


grid_search = GridSearchCV(estimator = clf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train_pca, train_labels)
print(grid_search.best_params_)

y_pred = grid_search.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Current Parameters are:

{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None,
'verbose': 0, 'warm_start': False}
Hyperparameters to tune:

Fitting 3 folds for each of 90 candidates, totalling 270 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    7.9s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   34.5s
[Parallel(n_jobs=-1)]: Done 270 out of 270 | elapsed:  1.0min finished

{'bootstrap': True, 'max_depth': 70, 'max_features': 3, 'min_samples_leaf': 3,
```

```
'n_estimators': 100}
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         1
         2.0       1.00      1.00      1.00         1
         3.0       1.00      0.50      0.67         2
         4.0       1.00      0.50      0.67         2
         5.0       0.00      0.00      0.00         1
         6.0       1.00      1.00      1.00         2
         7.0       1.00      0.67      0.80         3
         8.0       0.50      1.00      0.67         1
         9.0       1.00      0.67      0.80         3
        10.0       1.00      1.00      1.00         1
        11.0       0.00      0.00      0.00         0
        12.0       1.00      0.67      0.80         3
        13.0       0.33      1.00      0.50         1
        14.0       0.00      0.00      0.00         0
        15.0       0.50      0.50      0.50         2

    accuracy                           0.70        23
   macro avg       0.69      0.63      0.63        23
weighted avg       0.86      0.70      0.74        23
```

3.5 KNeighbors Regressor

Description K-nearest-neighbor method is a nonparametric statistical method used for classification and regression. In both cases, the input contains K closest training samples in the Feature Space. In KNeighbors classification, the output is a classification label. The class of an object is determined by a majority vote of its neighbors. The label that appears most among k nearest neighbors (k is a positive integer, usually is not large) determines the class assigned to the object. It is useful to measure the weight of neighbors, whether by classification or regression, so that the weight of the neighbor closed to the object is more important than the weight of the neighbor that relatively far away. K-nearest neighbor is also one of the simplest model in machine learning.

[39]:
```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from data import load
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
from PIL import Image as im
from imageprocessing_helper_functions import blurred, mirrored, flatten
from sklearn.metrics import confusion_matrix

train_data, train_labels = load("yalefaces_npy", split="train")
dev_data, dev_labels = load("yalefaces_npy", split="dev")

X = []
```

```
for i in range(0, np.shape(train_data)[0]):
    array = np.reshape(train_data[i,:], (243, 320))
    data = im.fromarray(array, mode = 'L')
    X.append(data)

label = train_labels
features = train_data

#blur images
X_blurred, label_blurred = blurred(X, label)
label_blurred = np.array(label_blurred)
features_blurred = flatten(X_blurred)

#mirror images
X_mirrored, label_mirrored = mirrored(X, label)
label_mirrored = np.array(label_mirrored)
features_mirrored = flatten(X_mirrored)

#combined
features_combined = np.concatenate((features, features_blurred))
labels_combined = np.concatenate((label, label_blurred))
features_combined = np.concatenate((features_combined, features_mirrored))
labels_combined = np.concatenate((labels_combined, label_mirrored))

# Compute a PCA
n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(features_combined)

# apply PCA transformation
X_train_pca = pca.transform(features_combined)
X_test_pca = pca.transform(dev_data)

train_labels = np.ravel(labels_combined)
dev_labels = np.ravel(dev_labels)


print("Fitting the classifier to the training set")
clf = KNeighborsClassifier().fit(X_train_pca, train_labels)

y_pred = clf.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Fitting the classifier to the training set
             precision    recall  f1-score   support

        1.0       0.00      0.00      0.00         1
        2.0       1.00      1.00      1.00         1
```

```
         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00         2
         5.0       0.00      0.00      0.00         1
         6.0       1.00      1.00      1.00         2
         7.0       0.25      0.33      0.29         3
         8.0       0.00      0.00      0.00         1
         9.0       0.00      0.00      0.00         3
        10.0       1.00      1.00      1.00         1
        12.0       0.17      0.33      0.22         3
        13.0       1.00      1.00      1.00         1
        15.0       0.00      0.00      0.00         2

    accuracy                           0.30        23
   macro avg       0.34      0.36      0.35        23
weighted avg       0.27      0.30      0.28        23
```

HyperParameter Search To attempt to achieve higher performance, we performed hyperparameter tuning on the following hyperparameters for the KNN classifier: leaf_size, n_neighbors, and metric. This was done by first viewing the default values for these parameters in the sklearn model, and then performing GridSearchCV on values surrounding those default values for each hyperparameter.

[40]:
```
from sklearn.model_selection import GridSearchCV

print("Current Parameters are: \n")
print(clf.get_params())

print("Hyperparameters to tune: \n")
param_grid = {
    'leaf_size': [1, 10, 30, 50, 70, 90],
    'n_neighbors': [1, 2, 3, 4, 5, 6],
    'metric': ['minkowski', 'euclidean']
}



grid_search = GridSearchCV(estimator = clf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train_pca, train_labels)
print(grid_search.best_params_)

y_pred = grid_search.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Current Parameters are:
```

```
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
Hyperparameters to tune:

Fitting 3 folds for each of 72 candidates, totalling 216 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  58 tasks      | elapsed:    1.4s

{'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 3}
              precision    recall  f1-score   support

         1.0       0.00      0.00      0.00         1
         2.0       1.00      1.00      1.00         1
         3.0       0.00      0.00      0.00         2
         4.0       1.00      0.50      0.67         2
         5.0       1.00      1.00      1.00         1
         6.0       1.00      1.00      1.00         2
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
         9.0       0.00      0.00      0.00         3
        10.0       1.00      1.00      1.00         1
        12.0       0.17      0.33      0.22         3
        13.0       0.00      0.00      0.00         1
        15.0       0.00      0.00      0.00         2

    accuracy                           0.30        23
   macro avg       0.40      0.37      0.38        23
weighted avg       0.33      0.30      0.30        23


[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:    2.3s finished
```

3.6 Resiliency

3.6.1 Original + 5 More Individuals

[4]:
```
import numpy as np
from PIL import Image, ImageOps
import os
from imageprocessing_helper_functions import flatten

def FiveMoreImages():

    #images convert to grayscale and resized
    images = [f for f in os.listdir('Augmented_With_5_More') if f != ".
 ↪DS_Store"]
    X = [Image.open("./Augmented_With_5_More/" + image) for image in images]
    for i in range(0, len(X)):
```

```
        X[i] = ImageOps.grayscale(X[i])
        X[i] = X[i].resize((243,320))



    label = []
    for image in images:
        if "putin" in image: #subject 16 is Putin
            label.append(16)
        elif "obama" in image: #subject 17 is Obama
            label.append(17)
        elif "prayuth" in image: #subject 18 is General Prayuth
            label.append(18)
        elif "Depp" in image: #subject 19 is Johnny Depp
            label.append(19)
        elif "kim" in image: #subject 20 is Kim Jong Un
            label.append(20)

    label = np.array(label)
    features = flatten(X)

    print("Sample Images")

    print("Individual %d"%(label[0]))
    display(X[0])
    print("Individual %d"%(label[1]))
    display(X[1])
    print("Individual %d"%(label[3]))
    display(X[3])

FiveMoreImages()
```

```
Sample Images
Individual 19
```

Individual 18

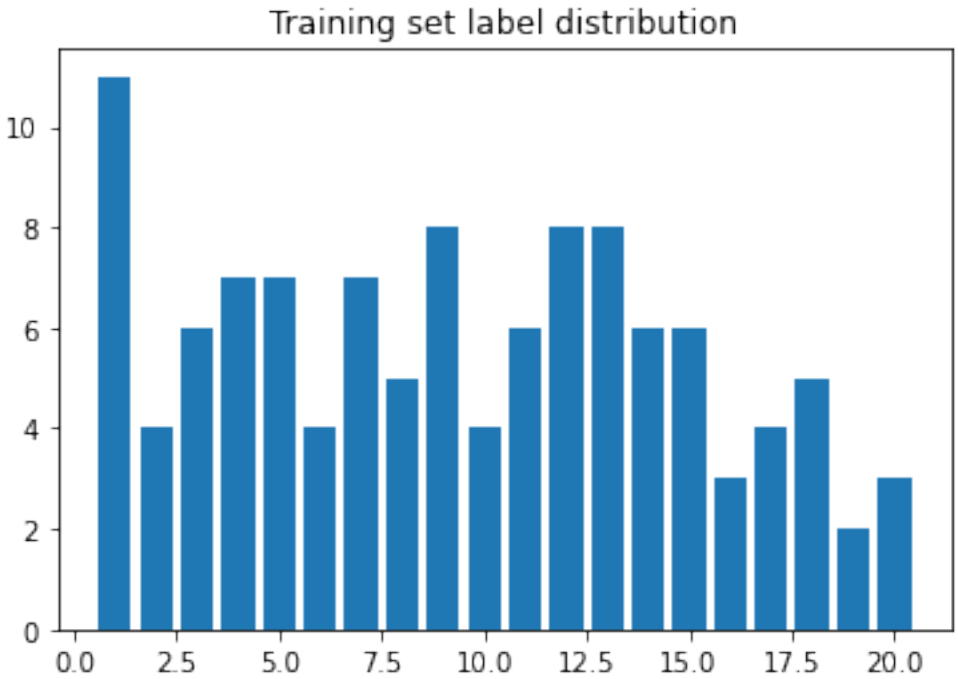
Individual 16

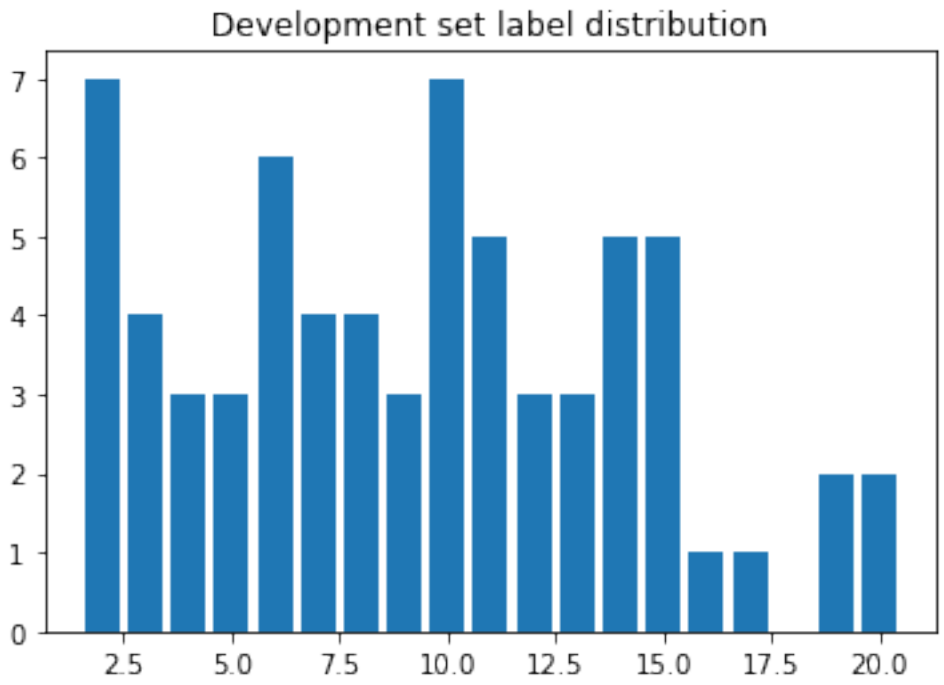
[5]:
```
import numpy as np
from PIL import Image
import os
from imageprocessing_helper_functions import flatten
from sklearn.model_selection import train_test_split
from DifferentFormatImages import FiveMoreImages

#yale images
images = [f for f in os.listdir('yalefaces') if f != ".DS_Store"]
X = [Image.open("./yalefaces/" + image) for image in images]

#divide into test,dev, and train (test, train, dev)
label = []
for image in images:
    if "subject11" in image:
        label.append(11)
    elif "subject02" in image:
        label.append(2)
    elif "subject03" in image:
        label.append(3)
```

```
    elif "subject04" in image:
        label.append(4)
    elif "subject05" in image:
        label.append(5)
    elif "subject06" in image:
        label.append(6)
    elif "subject07" in image:
        label.append(7)
    elif "subject08" in image:
        label.append(8)
    elif "subject09" in image:
        label.append(9)
    elif "subject10" in image:
        label.append(10)
    elif "subject15" in image:
        label.append(15)
    elif "subject12" in image:
        label.append(12)
    elif "subject13" in image:
        label.append(13)
    elif "subject14" in image:
        label.append(14)
    elif "subject01" in image:
        label.append(1)

label = np.array(label)
features = flatten(X)

features_fivemore, label_fivemore = FiveMoreImages()

#combined
features_combined = np.concatenate((features, features_fivemore))
labels_combined = np.concatenate((label, label_fivemore))


train_features, dev_features, train_labels, dev_labels =␣
 ↪train_test_split(features_combined, labels_combined, test_size=0.4,␣
 ↪random_state=42)
dev_features, test_features, dev_labels, test_labels =␣
 ↪train_test_split(dev_features, dev_labels, test_size=0.1, random_state=42)

from collections import defaultdict
import numpy as np
from data import load
import matplotlib.pyplot as plt
%matplotlib inline
```

```
counts = defaultdict(int)

for label in train_labels:
    counts[label] += 1

plt.figure()
plt.title("Training set label distribution")

k = counts.keys()
v = counts.values()
plt.bar(list(k), height=list(v))

dev_counts = defaultdict(int)

for label in dev_labels:
    dev_counts[label] += 1

plt.figure()
plt.title("Development set label distribution")
dk = dev_counts.keys()
dv = dev_counts.values()
plt.bar(list(dk), height=list(dv))
```

[5]: <BarContainer object of 18 artists>





Most images are converted to grayscale from RGB and to the same dimension as that of the Yale Face Database images just so that the new info can be easily integrated in the same methodology as before, however, as can be observed the integrity of the height to width ratio is not conserved!

Label distribution is again not quite uniform so might need to look into domain adaptation techniques, but this is not implemented in this particular method/experiment

Best Model (MLP Classifier) On This Dataset

[11]:
```
import numpy as np
from data import load
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from PIL import Image as im
from imageprocessing_helper_functions import blurred, mirrored, flatten
from sklearn.metrics import confusion_matrix



train_data, train_labels = load("yalefaces_npy_fivemore", split="train")
dev_data, dev_labels = load("yalefaces_npy_fivemore", split="dev")

X = []

for i in range(0, np.shape(train_data)[0]):
```

```
    array = np.reshape(train_data[i,:], (243, 320))
    data = im.fromarray(array, mode = 'L')
    X.append(data)

label = train_labels
features = train_data

#blur images
X_blurred, label_blurred = blurred(X, label)
label_blurred = np.array(label_blurred)
features_blurred = flatten(X_blurred)

#mirror images
X_mirrored, label_mirrored = mirrored(X, label)
label_mirrored = np.array(label_mirrored)
features_mirrored = flatten(X_mirrored)

#combined
features_combined = np.concatenate((features, features_blurred))
labels_combined = np.concatenate((label, label_blurred))
features_combined = np.concatenate((features_combined, features_mirrored))
labels_combined = np.concatenate((labels_combined, label_mirrored))

# Compute a PCA
n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(features_combined)

# apply PCA transformation
X_train_pca = pca.transform(features_combined)
X_test_pca = pca.transform(dev_data)

train_labels = np.ravel(labels_combined)
dev_labels = np.ravel(dev_labels)


print("Fitting the classifier to the training set")
clf = MLPClassifier(hidden_layer_sizes=(1024,), batch_size=100,␣
 ↪early_stopping=True).fit(X_train_pca, train_labels)

y_pred = clf.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Fitting the classifier to the training set
             precision    recall  f1-score   support

          1       0.00      0.00      0.00         0
          2       0.75      0.43      0.55         7
          3       1.00      0.50      0.67         4
```

```
              4       1.00      0.67      0.80         3
              5       0.33      0.33      0.33         3
              6       1.00      0.83      0.91         6
              7       0.00      0.00      0.00         4
              8       0.75      0.75      0.75         4
              9       0.50      0.67      0.57         3
             10       1.00      0.57      0.73         7
             11       0.50      0.60      0.55         5
             12       0.50      1.00      0.67         3
             13       1.00      0.33      0.50         3
             14       1.00      0.80      0.89         5
             15       0.80      0.80      0.80         5
             16       0.00      0.00      0.00         1
             17       0.00      0.00      0.00         1
             18       0.00      0.00      0.00         0
             19       0.00      0.00      0.00         2
             20       0.50      1.00      0.67         2

    accuracy                             0.57        68
   macro avg       0.53      0.46      0.47        68
weighted avg       0.70      0.57      0.61        68


/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[12]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

print("Current Parameters are: \n")
print(clf.get_params())

print("Hyperparameters to tune: \n")
param_grid = {'activation': ['relu', 'tanh'],
              'alpha': [1, 0.1, 0.01, 0.001, .0001],
              'beta_1': [0.9, .92, .94, .96, .98, 1],
             }

clf_new = MLPClassifier(max_iter=100)

grid_search = GridSearchCV(estimator = clf_new, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train_pca, train_labels)
print(grid_search.best_params_)


y_pred = grid_search.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Current Parameters are:

{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 100, 'beta_1': 0.9,
'beta_2': 0.999, 'early_stopping': True, 'epsilon': 1e-08, 'hidden_layer_sizes':
(1024,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun':
15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change': 10,
'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': None, 'shuffle':
True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose':
False, 'warm_start': False}
Hyperparameters to tune:

Fitting 3 folds for each of 60 candidates, totalling 180 fits

/opt/anaconda3/lib/python3.8/site-
packages/sklearn/model_selection/_search.py:918: UserWarning: One or more of the
test scores are non-finite: [0.05555556 0.04385965 0.05847953 0.06140351
0.04678363        nan
 0.05263158 0.08479532 0.04093567 0.06432749 0.04678363        nan
 0.09064327 0.04678363 0.04093567 0.04678363 0.06432749        nan
```

```
 0.02339181 0.08187135 0.07017544 0.02631579 0.04093567        nan
 0.05847953 0.0380117  0.0380117  0.0877193  0.07894737        nan
 0.04093567 0.08479532 0.06432749 0.05555556 0.08187135        nan
 0.02631579 0.02339181 0.04678363 0.04093567 0.04385965        nan
 0.04678363 0.0497076  0.06140351 0.0380117  0.05263158        nan
 0.03216374 0.0380117  0.02631579 0.0497076  0.06140351        nan
 0.02339181 0.0380117  0.04678363 0.07017544 0.06725146        nan]
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

{'activation': 'relu', 'alpha': 0.01, 'beta_1': 0.9}
              precision    recall  f1-score   support

           2       1.00      0.86      0.92         7
           3       1.00      0.50      0.67         4
           4       1.00      1.00      1.00         3
           5       0.75      1.00      0.86         3
           6       1.00      0.83      0.91         6
           7       0.80      1.00      0.89         4
           8       1.00      0.75      0.86         4
           9       0.60      1.00      0.75         3
          10       0.88      1.00      0.93         7
          11       1.00      1.00      1.00         5
          12       0.50      1.00      0.67         3
          13       0.50      0.67      0.57         3
          14       1.00      1.00      1.00         5
          15       1.00      1.00      1.00         5
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         1
          19       0.00      0.00      0.00         2
          20       0.00      0.00      0.00         2

    accuracy                           0.82        68
   macro avg       0.67      0.70      0.67        68
weighted avg       0.81      0.82      0.80        68


/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

3.6.2 Original + 15 More Individuals

[13]:
```
def TenMoreImages():

    #images convert to grayscale and resized
    images = [f for f in os.listdir('Augmented_With_10_More') if f != ".
↪DS_Store"]
    X = [Image.open("./Augmented_With_10_More/" + image) for image in images]
    for i in range(0, len(X)):
        X[i] = ImageOps.grayscale(X[i])
        X[i] = X[i].resize((243,320))



    label = []
    for image in images:
        if "berbatov" in image: #subject 21 is berbatov
            label.append(21)
        elif "churchill" in image: #subject 22 is churchill
            label.append(22)
        elif "cruise" in image: #subject 23 is tom cruise
            label.append(23)
        elif "dimaria" in image: #subject 24 is dimaria
            label.append(24)
        elif "drogba" in image: #subject 25 is drogba
            label.append(25)
        elif "klopp" in image: #subject 26 is klopp
            label.append(26)
        elif "mj" in image: #subject 27 is michael jackson
            label.append(27)
        elif "ronaldinho" in image: #subject 28 is ronaldinho
            label.append(28)
        elif "rooney" in image: #subject 29 is rooney
            label.append(29)
        elif "xi" in image: #subject 30 is xi
            label.append(30)

    label = np.array(label)
    features = flatten(X)
```

```
    print("Sample Images")

    print("Individual %d"%(label[0]))
    display(X[0])
    print("Individual %d"%(label[1]))
    display(X[1])
    print("Individual %d"%(label[3]))
    display(X[3])

TenMoreImages()
```

```
Sample Images
Individual 27
```

```
Individual 24
```

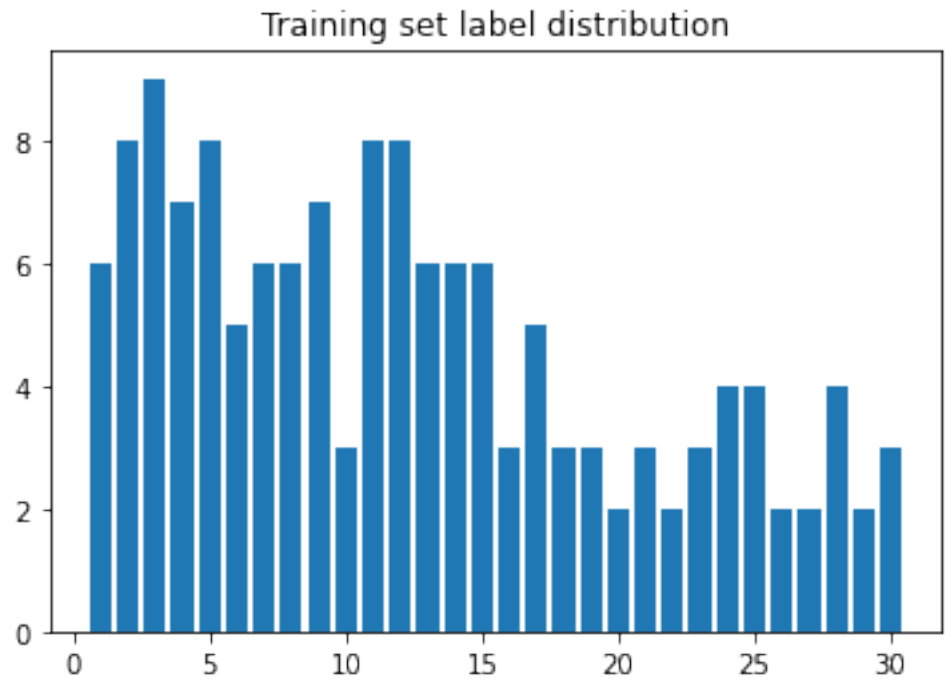
Individual 21

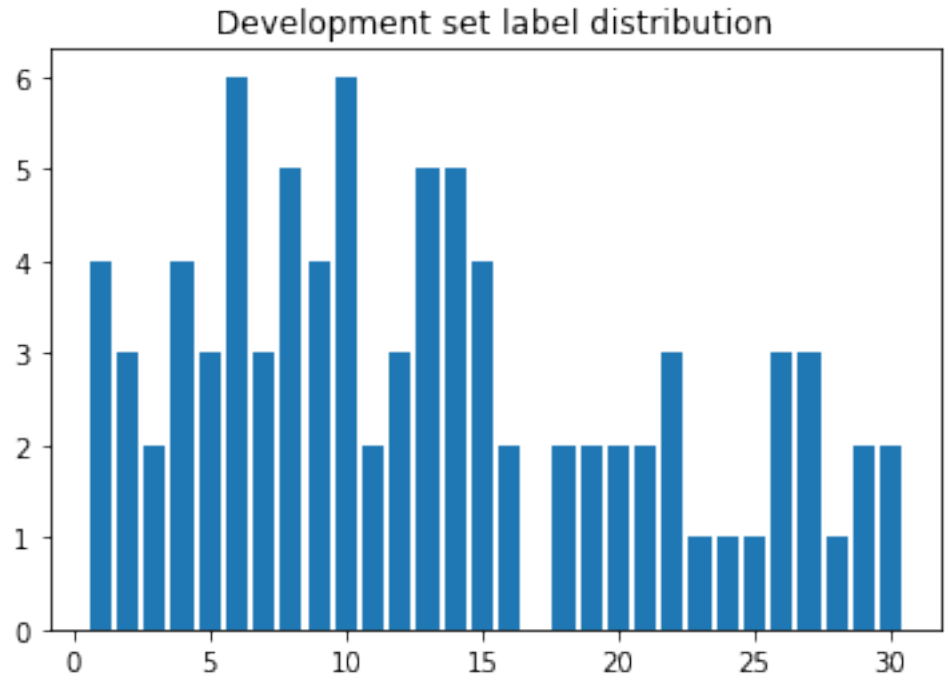
[14]:
```
import numpy as np
from PIL import Image
import os
from imageprocessing_helper_functions import flatten
from sklearn.model_selection import train_test_split
from DifferentFormatImages import FiveMoreImages, TenMoreImages

#yale images
images = [f for f in os.listdir('yalefaces') if f != ".DS_Store"]
X = [Image.open("./yalefaces/" + image) for image in images]

#divide into test,dev, and train (test, train, dev)
label = []
for image in images:
    if "subject11" in image:
        label.append(11)
    elif "subject02" in image:
        label.append(2)
    elif "subject03" in image:
        label.append(3)
```

```
    elif "subject04" in image:
        label.append(4)
    elif "subject05" in image:
        label.append(5)
    elif "subject06" in image:
        label.append(6)
    elif "subject07" in image:
        label.append(7)
    elif "subject08" in image:
        label.append(8)
    elif "subject09" in image:
        label.append(9)
    elif "subject10" in image:
        label.append(10)
    elif "subject15" in image:
        label.append(15)
    elif "subject12" in image:
        label.append(12)
    elif "subject13" in image:
        label.append(13)
    elif "subject14" in image:
        label.append(14)
    elif "subject01" in image:
        label.append(1)

label = np.array(label)
features = flatten(X)

features_fivemore, label_fivemore = FiveMoreImages()
features_tenmore, label_tenmore = TenMoreImages()


#combined
features_combined = np.concatenate((features, features_fivemore))
labels_combined = np.concatenate((label, label_fivemore))

features_combined = np.concatenate((features_combined, features_tenmore))
labels_combined = np.concatenate((labels_combined, label_tenmore))


train_features, dev_features, train_labels, dev_labels =␣
 ↪train_test_split(features_combined, labels_combined, test_size=0.4,␣
 ↪random_state=42)
dev_features, test_features, dev_labels, test_labels =␣
 ↪train_test_split(dev_features, dev_labels, test_size=0.1, random_state=42)

from collections import defaultdict
```

```
import numpy as np
from data import load
import matplotlib.pyplot as plt
%matplotlib inline

counts = defaultdict(int)

for label in train_labels:
    counts[label] += 1

plt.figure()
plt.title("Training set label distribution")

k = counts.keys()
v = counts.values()
plt.bar(list(k), height=list(v))

dev_counts = defaultdict(int)

for label in dev_labels:
    dev_counts[label] += 1

plt.figure()
plt.title("Development set label distribution")
dk = dev_counts.keys()
dv = dev_counts.values()
plt.bar(list(dk), height=list(dv))
```

[14]: <BarContainer object of 29 artists>





Label distribution is again not quite uniform so might need to look into domain adaptation techniques, but this is not implemented in this particular method/experiment

Best Model (MLP Classifier) On This Dataset

[15]:
```
import numpy as np
from data import load
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from PIL import Image as im
from imageprocessing_helper_functions import blurred, mirrored, flatten
from sklearn.metrics import confusion_matrix



train_data, train_labels = load("yalefaces_npy_fifteenmore", split="train")
dev_data, dev_labels = load("yalefaces_npy_fifteenmore", split="dev")

X = []

for i in range(0, np.shape(train_data)[0]):
    array = np.reshape(train_data[i,:], (243, 320))
    data = im.fromarray(array, mode = 'L')
    X.append(data)

label = train_labels
features = train_data

#blur images
X_blurred, label_blurred = blurred(X, label)
label_blurred = np.array(label_blurred)
features_blurred = flatten(X_blurred)

#mirror images
X_mirrored, label_mirrored = mirrored(X, label)
label_mirrored = np.array(label_mirrored)
features_mirrored = flatten(X_mirrored)

#combined
features_combined = np.concatenate((features, features_blurred))
labels_combined = np.concatenate((label, label_blurred))
features_combined = np.concatenate((features_combined, features_mirrored))
labels_combined = np.concatenate((labels_combined, label_mirrored))

# Compute a PCA
n_components = 100
pca = PCA(n_components=n_components, whiten=True).fit(features_combined)

# apply PCA transformation
X_train_pca = pca.transform(features_combined)
```

```
X_test_pca = pca.transform(dev_data)

train_labels = np.ravel(labels_combined)
dev_labels = np.ravel(dev_labels)


print("Fitting the classifier to the training set")
clf = MLPClassifier(hidden_layer_sizes=(1024,), batch_size=100,␣
 ↪early_stopping=True).fit(X_train_pca, train_labels)

y_pred = clf.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Fitting the classifier to the training set
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         3
           3       0.40      1.00      0.57         2
           4       0.80      1.00      0.89         4
           5       1.00      1.00      1.00         3
           6       0.86      1.00      0.92         6
           7       1.00      1.00      1.00         3
           8       1.00      0.80      0.89         5
           9       1.00      1.00      1.00         4
          10       1.00      1.00      1.00         6
          11       1.00      1.00      1.00         2
          12       0.75      1.00      0.86         3
          13       0.83      1.00      0.91         5
          14       1.00      1.00      1.00         5
          15       1.00      1.00      1.00         4
          16       0.00      0.00      0.00         2
          17       0.00      0.00      0.00         0
          18       0.00      0.00      0.00         2
          19       0.00      0.00      0.00         2
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         3
          23       0.00      0.00      0.00         1
          24       0.00      0.00      0.00         1
          25       0.20      1.00      0.33         1
          26       0.00      0.00      0.00         3
          27       1.00      0.33      0.50         3
          28       0.00      0.00      0.00         1
          29       0.00      0.00      0.00         2
          30       0.00      0.00      0.00         2

    accuracy                           0.70        86
```

```
   macro avg       0.49      0.54      0.50        86
weighted avg       0.67      0.70      0.67        86
```

```
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[17]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

print("Current Parameters are: \n")
print(clf.get_params())

print("Hyperparameters to tune: \n")
param_grid = {'activation': ['relu', 'tanh'],
              'alpha': [1, 0.1, 0.01, 0.001, .0001],
              'beta_1': [0.9, .92, .94, .96, .98, 1],
             }

clf_new = MLPClassifier(max_iter=100)
```

```
grid_search = GridSearchCV(estimator = clf_new, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train_pca, train_labels)
print(grid_search.best_params_)


y_pred = grid_search.predict(X_test_pca)
print(classification_report(dev_labels, y_pred))
```

```
Current Parameters are:

{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 100, 'beta_1': 0.9,
'beta_2': 0.999, 'early_stopping': True, 'epsilon': 1e-08, 'hidden_layer_sizes':
(1024,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun':
15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change': 10,
'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': None, 'shuffle':
True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose':
False, 'warm_start': False}
Hyperparameters to tune:

Fitting 3 folds for each of 60 candidates, totalling 180 fits

/opt/anaconda3/lib/python3.8/site-
packages/sklearn/model_selection/_search.py:918: UserWarning: One or more of the
test scores are non-finite: [0.03009259 0.03472222 0.02777778 0.04398148
0.03009259        nan
 0.04166667 0.03240741 0.03935185 0.03009259 0.03935185        nan
 0.04166667 0.04861111 0.04398148 0.04166667 0.01157407        nan
 0.00925926 0.04166667 0.03472222 0.02546296 0.03009259        nan
 0.03935185 0.03240741 0.04398148 0.04861111 0.03009259        nan
 0.04398148 0.0162037  0.01851852 0.01157407 0.04166667        nan
 0.03240741 0.00925926 0.03009259 0.02777778 0.02083333        nan
 0.01157407 0.01388889 0.03009259 0.02777778 0.02314815        nan
 0.01388889 0.01851852 0.03009259 0.02314815 0.02083333        nan
 0.02777778 0.02083333 0.01851852 0.02546296 0.01388889        nan]
  warnings.warn(

{'activation': 'relu', 'alpha': 0.01, 'beta_1': 0.92}
              precision    recall  f1-score   support

           1       0.67      1.00      0.80         4
           2       0.50      0.33      0.40         3
           3       0.33      1.00      0.50         2
           4       0.67      1.00      0.80         4
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         6
           7       0.75      1.00      0.86         3
```

```
              8       1.00      1.00      1.00         5
              9       1.00      0.75      0.86         4
             10       1.00      0.83      0.91         6
             11       0.67      1.00      0.80         2
             12       1.00      1.00      1.00         3
             13       0.83      1.00      0.91         5
             14       1.00      0.80      0.89         5
             15       1.00      0.75      0.86         4
             16       0.00      0.00      0.00         2
             17       0.00      0.00      0.00         0
             18       0.00      0.00      0.00         2
             19       0.00      0.00      0.00         2
             20       0.00      0.00      0.00         2
             21       0.00      0.00      0.00         2
             22       0.00      0.00      0.00         3
             23       0.00      0.00      0.00         1
             24       0.00      0.00      0.00         1
             25       0.00      0.00      0.00         1
             26       0.00      0.00      0.00         3
             27       0.00      0.00      0.00         3
             28       0.00      0.00      0.00         1
             29       0.00      0.00      0.00         2
             30       0.00      0.00      0.00         2

    accuracy                           0.62        86
   macro avg       0.41      0.45      0.42        86
weighted avg       0.60      0.62      0.59        86


/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

3.6.3 Conclusion Regarding Resiliency

As can be seen by the results above the model develop failed to predict any/almost all of the added images. The new added images have many factors that differed from the Yale Face Database Images

Firstly, height to width ratio is distored in the added images in such a way that the dimensions could be the same as that of the Yale Face Database

Secondly, the images added have componenets such as torso, arms, more vivid backgrounds ETC. and did not only concentrate on the face like that of the Yale Face Database

In conclusion, further processing is needed for the new images such that only the face portion is retained and fed into the model. However, at this moment, this model doesn't display resilience to other types of images (that are not face close-up), however, on the bright side the first 15 images (from the Yale Database) are able to be classified correctly even with the added "junk" images this displayed resiliency in a sense for these images to added

4 Analysis and Conclusion

Throughout our exploration of the Yale Faces data set we found that the Support Vector Machine and Multilayer Perceptron models performed the best. We believe that the MLP model performed one on our dataset as this model is better than a linear perceptron because it can classify data that is not perfectly linearly separable and image data is most likely only separable by very complicated functions. Also, the MLP classifier allows us to train the data using a neural network with as many layers as we want without having to form the structure of the layers by hand. Since neural networks are typically a good model for image classification problems and increasing the number of layers is often correlated with model accuracy, it seems logical that it would work well. SVM allows for non-linear classification. SVM is a good model for analyzing images as it aims to maximize the margin between different decision surfaces. This allows the model to not overfit to the data, and to make decisions based on how different two images' properties may be and not just on whether they are an exact match to the training data set. Paired with PCA, Support Vector machines are able to make decisions in this way only on properties of the data that are hopefully the most

relevant. The properties that are most relevant are decided by the PCA algorithm. We were able to get testing accuracy of around 90% for each of these models. Despite our original predictions we found that neural networks were difficult to work with for our dataset as they did not work well as an out of the box tool, and the models that worked well on the dataset form class performed very poorly on our data set. We were able to meet our final test accuracy goal of 80%, however some of our other original deliverables did not align with the direction we took the project. We did not end up implementing random sampling, since we ended up not having enough data instead of having too much, and we focused on other preprocessing techniques such as PCA instead of Gabor filters. We also decided that extracting skin tone as a feature would not be beneficial as our images were black and white. While we did not accomplish many of our original deliverables, we instead worked on different types of models and data processing techniques that we had not planned at the beginning of the project as we moved away from neural networks. Moving forward, an area in which our models could be greatly improved is resiliency. When tested against outside images the model failed. One idea of how we could improve in this area is to include a preprocessing filter that would isolate the face from a photo before analyzing it.

5 Citation

[1] "UCSD Computer Vision." Yale Face Database, http://vision.ucsd.edu/content/yale-face-database.

[2] "An on-Device Deep Neural Network for Face Detection." Apple Machine Learning Research, https://machinelearning.apple.com/research/face- detection.

5.1 Code Used

For blurring, mirroring, and flattening

[ ]:
```
import numpy as np
from PIL import Image, ImageFilter, ImageOps
from sklearn.model_selection import train_test_split
import numpy as np

#X is list of image object
#label is list of integer label s
#return blurred image list X_blurred, and its labels label_blurred
def blurred(X, label):
    X_blurred = []
    label_blurred = []
    for i in range(0, len(X)):
        X_blurred.append(X[i].filter(ImageFilter.BLUR))
        label_blurred.append(label[i])
    return X_blurred, label_blurred

#X is list of image object
#label is list of integer label s
#return mirror image list X_mirrored, and its labels label_mirrored
def mirrored(X, label):
```

```
    X_mirrored = []
    label_mirrored = []
    for i in range(0, len(X)):
        X_mirrored.append(ImageOps.mirror(X[i]))
        label_mirrored.append(label[i])
    return X_mirrored, label_mirrored

#X is list of image object
#return flatten images numpy array of dimension (lenght of X, 77760)
def flatten(X):
    features = np.zeros((len(X),77760))
    i = 0
    for currentImage in X:
        data1 = np.asarray(currentImage)
        data1 = data1.reshape((1, 243*320))
        features[i,:] = data1
        i += 1
    return features
```

To Process More Images (To Yale Face Database)

[ ]:
```
import numpy as np
from PIL import Image, ImageOps
import os
from imageprocessing_helper_functions import flatten


def FiveMoreImages():

    #images convert to grayscale and resized
    images = [f for f in os.listdir('Augmented_With_5_More') if f != ".
↪DS_Store"]
    X = [Image.open("./Augmented_With_5_More/" + image) for image in images]
    for i in range(0, len(X)):
        X[i] = ImageOps.grayscale(X[i])
        X[i] = X[i].resize((243,320))



    label = []
    for image in images:
        if "putin" in image: #subject 16 is Putin
            label.append(16)
        elif "obama" in image: #subject 17 is Obama
            label.append(17)
        elif "prayuth" in image: #subject 18 is General Prayuth
            label.append(18)
```

```
        elif "Depp" in image: #subject 19 is Johnny Depp
            label.append(19)
        elif "kim" in image: #subject 20 is Kim Jong Un
            label.append(20)

    label = np.array(label)
    features = flatten(X)
    return features, label

def TenMoreImages():

    #images convert to grayscale and resized
    images = [f for f in os.listdir('Augmented_With_10_More') if f != ".
↪DS_Store"]
    X = [Image.open("./Augmented_With_10_More/" + image) for image in images]
    for i in range(0, len(X)):
        X[i] = ImageOps.grayscale(X[i])
        X[i] = X[i].resize((243,320))



    label = []
    for image in images:
        if "berbatov" in image: #subject 21 is berbatov
            label.append(21)
        elif "churchill" in image: #subject 22 is churchill
            label.append(22)
        elif "cruise" in image: #subject 23 is tom cruise
            label.append(23)
        elif "dimaria" in image: #subject 24 is dimaria
            label.append(24)
        elif "drogba" in image: #subject 25 is drogba
            label.append(25)
        elif "klopp" in image: #subject 26 is klopp
            label.append(26)
        elif "mj" in image: #subject 27 is michael jackson
            label.append(27)
        elif "ronaldinho" in image: #subject 28 is ronaldinho
            label.append(28)
        elif "rooney" in image: #subject 29 is rooney
            label.append(29)
        elif "xi" in image: #subject 30 is xi
            label.append(30)

    label = np.array(label)
    features = flatten(X)
    return features, label
```

To Convert To NPY Files (Original)

[ ]:
```
import numpy as np
from PIL import Image
import os
from imageprocessing_helper_functions import flatten
from sklearn.model_selection import train_test_split

#yale images
images = [f for f in os.listdir('yalefaces') if f != ".DS_Store"]
X = [Image.open("./yalefaces/" + image) for image in images]

#divide into test,dev, and train (test, train, dev)
label = []
for image in images:
    if "subject11" in image:
        label.append(11)
    elif "subject02" in image:
        label.append(2)
    elif "subject03" in image:
        label.append(3)
    elif "subject04" in image:
        label.append(4)
    elif "subject05" in image:
        label.append(5)
    elif "subject06" in image:
        label.append(6)
    elif "subject07" in image:
        label.append(7)
    elif "subject08" in image:
        label.append(8)
    elif "subject09" in image:
        label.append(9)
    elif "subject10" in image:
        label.append(10)
    elif "subject15" in image:
        label.append(15)
    elif "subject12" in image:
        label.append(12)
    elif "subject13" in image:
        label.append(13)
    elif "subject14" in image:
        label.append(14)
    elif "subject01" in image:
        label.append(1)

label = np.array(label)
```

```
features = flatten(X)


train_features, dev_features, train_labels, dev_labels =␣
 ↪train_test_split(features, label, test_size=0.4, random_state=42)
dev_features, test_features, dev_labels, test_labels =␣
 ↪train_test_split(dev_features, dev_labels, test_size=0.1, random_state=42)



np.save(os.path.join("yalefaces_npy","dev.feats"), dev_features)
np.save(os.path.join("yalefaces_npy","test.feats"), test_features)
np.save(os.path.join("yalefaces_npy","train.feats"), train_features)
np.save(os.path.join("yalefaces_npy","dev.labels"), dev_labels)
np.save(os.path.join("yalefaces_npy","test.labels"), test_labels)
np.save(os.path.join("yalefaces_npy","train.labels"), train_labels)
```

To Convert To NPY Files (20 Images)

[ ]:
```
import numpy as np
from PIL import Image
import os
from imageprocessing_helper_functions import flatten
from sklearn.model_selection import train_test_split
from DifferentFormatImages import FiveMoreImages

#yale images
images = [f for f in os.listdir('yalefaces') if f != ".DS_Store"]
X = [Image.open("./yalefaces/" + image) for image in images]

#divide into test,dev, and train (test, train, dev)
label = []
for image in images:
    if "subject11" in image:
        label.append(11)
    elif "subject02" in image:
        label.append(2)
    elif "subject03" in image:
        label.append(3)
    elif "subject04" in image:
        label.append(4)
    elif "subject05" in image:
        label.append(5)
    elif "subject06" in image:
        label.append(6)
    elif "subject07" in image:
        label.append(7)
```

```
    elif "subject08" in image:
        label.append(8)
    elif "subject09" in image:
        label.append(9)
    elif "subject10" in image:
        label.append(10)
    elif "subject15" in image:
        label.append(15)
    elif "subject12" in image:
        label.append(12)
    elif "subject13" in image:
        label.append(13)
    elif "subject14" in image:
        label.append(14)
    elif "subject01" in image:
        label.append(1)

label = np.array(label)
features = flatten(X)

features_fivemore, label_fivemore = FiveMoreImages()

#combined
features_combined = np.concatenate((features, features_fivemore))
labels_combined = np.concatenate((label, label_fivemore))


train_features, dev_features, train_labels, dev_labels =␣
 ↪train_test_split(features_combined, labels_combined, test_size=0.4,␣
 ↪random_state=42)
dev_features, test_features, dev_labels, test_labels =␣
 ↪train_test_split(dev_features, dev_labels, test_size=0.1, random_state=42)



np.save(os.path.join("yalefaces_npy_fivemore","dev.feats"), dev_features)
np.save(os.path.join("yalefaces_npy_fivemore","test.feats"), test_features)
np.save(os.path.join("yalefaces_npy_fivemore","train.feats"), train_features)
np.save(os.path.join("yalefaces_npy_fivemore","dev.labels"), dev_labels)
np.save(os.path.join("yalefaces_npy_fivemore","test.labels"), test_labels)
np.save(os.path.join("yalefaces_npy_fivemore","train.labels"), train_labels)
```

To Convert To NPY Files (30 Images)

[ ]:
```
import numpy as np
from PIL import Image
import os
```

```
from imageprocessing_helper_functions import flatten
from sklearn.model_selection import train_test_split
from DifferentFormatImages import FiveMoreImages, TenMoreImages

#yale images
images = [f for f in os.listdir('yalefaces') if f != ".DS_Store"]
X = [Image.open("./yalefaces/" + image) for image in images]

#divide into test,dev, and train (test, train, dev)
label = []
for image in images:
    if "subject11" in image:
        label.append(11)
    elif "subject02" in image:
        label.append(2)
    elif "subject03" in image:
        label.append(3)
    elif "subject04" in image:
        label.append(4)
    elif "subject05" in image:
        label.append(5)
    elif "subject06" in image:
        label.append(6)
    elif "subject07" in image:
        label.append(7)
    elif "subject08" in image:
        label.append(8)
    elif "subject09" in image:
        label.append(9)
    elif "subject10" in image:
        label.append(10)
    elif "subject15" in image:
        label.append(15)
    elif "subject12" in image:
        label.append(12)
    elif "subject13" in image:
        label.append(13)
    elif "subject14" in image:
        label.append(14)
    elif "subject01" in image:
        label.append(1)

label = np.array(label)
features = flatten(X)

features_fivemore, label_fivemore = FiveMoreImages()
features_tenmore, label_tenmore = TenMoreImages()
```

```
#combined
features_combined = np.concatenate((features, features_fivemore))
labels_combined = np.concatenate((label, label_fivemore))

features_combined = np.concatenate((features_combined, features_tenmore))
labels_combined = np.concatenate((labels_combined, label_tenmore))


train_features, dev_features, train_labels, dev_labels =␣
↪train_test_split(features_combined, labels_combined, test_size=0.4,␣
↪random_state=42)
dev_features, test_features, dev_labels, test_labels =␣
↪train_test_split(dev_features, dev_labels, test_size=0.1, random_state=42)



np.save(os.path.join("yalefaces_npy_fifteenmore","dev.feats"), dev_features)
np.save(os.path.join("yalefaces_npy_fifteenmore","test.feats"), test_features)
np.save(os.path.join("yalefaces_npy_fifteenmore","train.feats"), train_features)
np.save(os.path.join("yalefaces_npy_fifteenmore","dev.labels"), dev_labels)
np.save(os.path.join("yalefaces_npy_fifteenmore","test.labels"), test_labels)
np.save(os.path.join("yalefaces_npy_fifteenmore","train.labels"), train_labels)
```

To Load Data From NPY Files

[ ]:
```
import numpy as np

import random
from scipy import ndarray
import skimage as sk
from skimage import transform
from skimage import util

import os

def load(path, split, load_labels=True):
    if load_labels:
        labels = np.load(f'{path}/{split}.labels.npy')
    else:
        labels = None
    data = np.load(f'{path}/{split}.feats.npy')
    return data, labels
```