Homework 1: Face Detection

Part I. Implementation:

Part 1.

```python
# Begin your code (Part 1)
''' Using cv2.imread to read in all the images from the face and non-face folder.
    the star symbol (*) will read in all the files despite its specific name.     '''
image_face = [cv2.imread(file) for file in glob.glob(dataPath + '/face/*.pgm')]
image_non = [cv2.imread(file) for file in glob.glob(dataPath + '/non-face/*.pgm')]

''' Insert the images and its label to the dataset list in pairs. '''
dataset = []
for img in image_face:
  img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # convert image to gray scale
  pair = [0, 0]
  pair[0] = img_g
  pair[1] = 1
  dataset.append(pair)

for img in image_non:
  img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # convert image to gray scale
  pair = [0, 0]
  pair[0] = img_g
  pair[1] = 0
  dataset.append(pair)

# End your code (Part 1)
```

Part 2.

```python
        # Begin your code (Part 2)
        '''
        hj(xi) = 1 if fj(xi) < 0
               = 0 if fj(xi) ≥ 0
        hj(xi) = |hj(xi) - label(xi)|
        ej     = Σi (wi * hj(xi))
        errorIndex: the index of min error
        bestError: the min error value
        bestClf: the classifer which has feature of the lowest error value
        '''
        h = np.abs(np.where(featureVals < 0, 1, 0) - labels)
        e = np.sum(np.multiply(weights, h), axis=1)
        errorIndex = np.argmin(e)
        bestError = e[errorIndex]
        bestClf = WeakClassifier(features[errorIndex])

        # End your code (Part 2)
```

Part 4.

```python
# Begin your code (Part 4)

with open(dataPath) as f:
    lines = f.readlines() # Read in all the lines in the given .txt file

''' Read in the .txt file '''
files, cords = []
count, num = 0
file_index = -1
for line in lines:
    if count == num: # Read in file name and the number of faces
        sep = line.split(" ") # Separate the file name and the number
        files.append(sep[0])
        num = int(sep[1])
        count = 0
        file_index += 1
        cords.append([])
    else: # Read in all the cords
        count += 1
        sep = line[:-1].split(" ") # Neglect the '\n' sign using splice [:-1]
        c = []
        for i in range(4):
            c.append(int(sep[i])) # Append the cords into c
        cords[file_index].append(c) # Append c to cords[]
```

```python
''' Use the cords and classifer to detect faces. '''
for i in range(len(files)):
    fig, ax = plt.subplots()
    image = cv2.imread("data/detect/"+ files[i]) # Read the corresponding image
    img_g = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Convert the image to grayscale
    imgplot = plt.imshow(img_g, cmap='gray') # Show the image in grayscale

    rect = []
    for cord in cords[i]:
        cropped = img_g[cord[1]:cord[1]+cord[3], cord[0]:cord[0]+cord[2]] # Crop the image into the right coordinates
        resized = cv2.resize(cropped, (19, 19), interpolation = cv2.INTER_AREA) # Resize the cropped image to 19x19
        face = clf.classify(resized) # Classify the face, 1 is face, 0 is not
        if face: # Draw green box if it is face
            rect.append(plt.Rectangle((cord[0], cord[1]), cord[2], cord[3], linewidth=1, edgecolor='g', facecolor='none'))
        else: # Draw red box if it is not face
            rect.append(plt.Rectangle((cord[0], cord[1]), cord[2], cord[3], linewidth=1, edgecolor='r', facecolor='none'))

    for r in rect:
        plt.gca().add_patch(r) # Show the boxes on the image

    plt.show()

# End your code (Part 4)
```

Part II. Results & Analysis:

My data accuracy with no modification (T = 10, threshold = 0, polarity = 1):

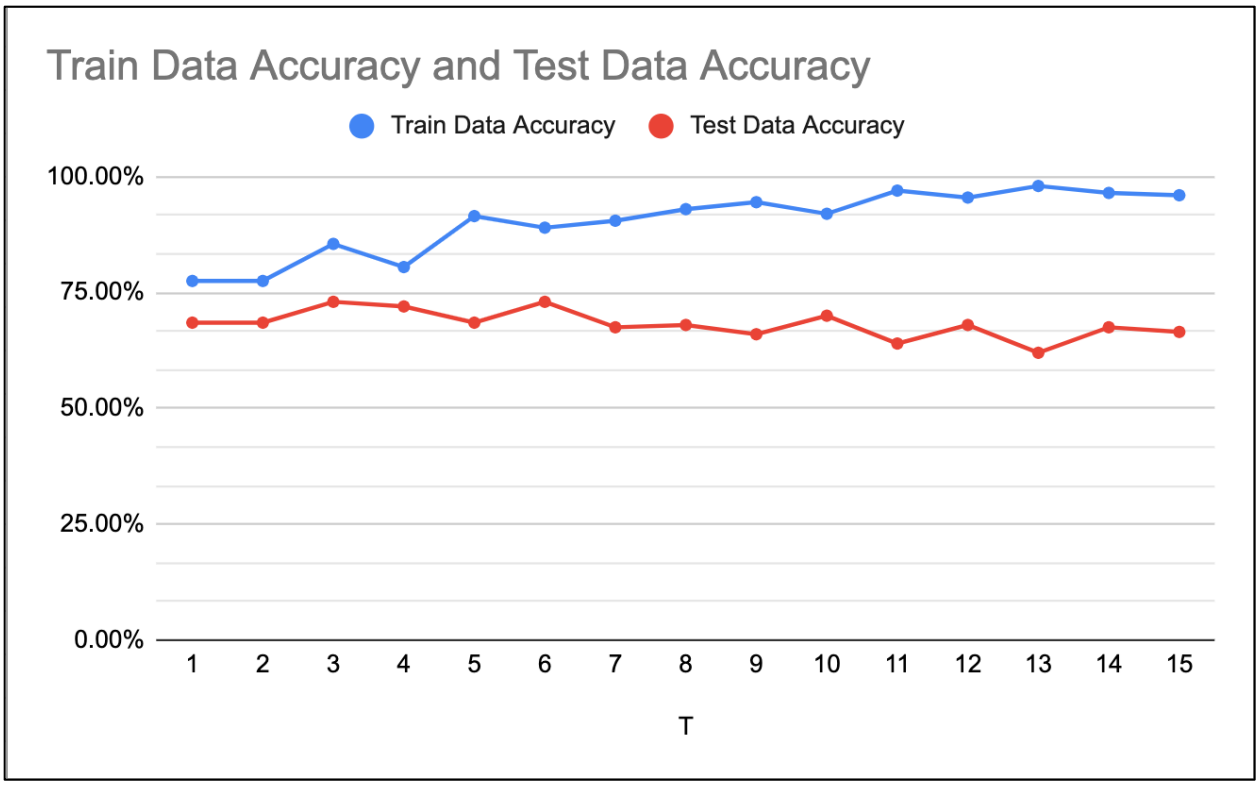
```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(7, 1,
 1, 17)], negative regions=[RectangleRegion(6, 1, 1, 17)]) with accuracy: 146.000000 and alpha: 0.997055
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 7,
 2, 4)], negative regions=[RectangleRegion(4, 11, 2, 4)]) with accuracy: 150.000000 and alpha: 0.806344

Evaluate your classifier with training dataset
False Positive Rate: 0/100 (0.000000)
False Negative Rate: 16/100 (0.160000)
Accuracy: 184/200 (0.920000)

Evaluate your classifier with test dataset
False Positive Rate: 38/100 (0.380000)
False Negative Rate: 22/100 (0.220000)
Accuracy: 140/200 (0.700000)
```

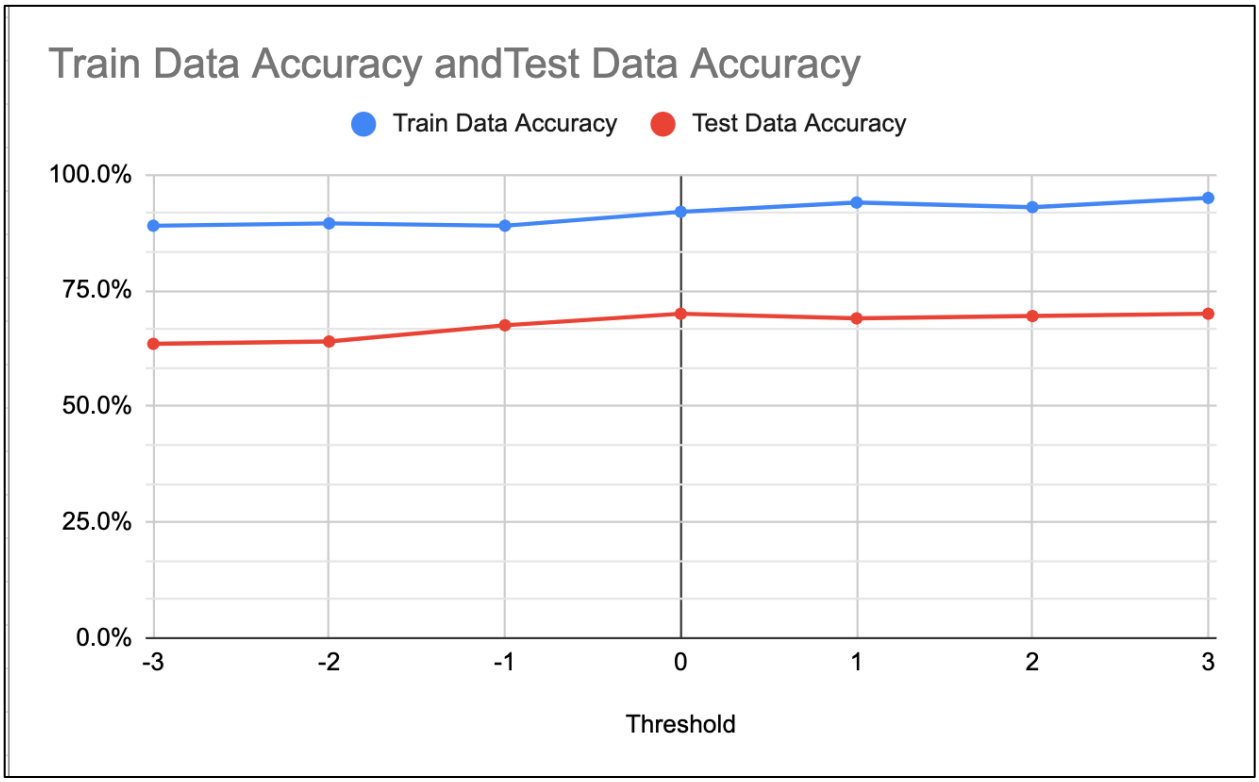
Hold polarity and threshold as a constant (thr=0, pol=1) and vary T from 1 to 15. Train Data Accuracy seems to have a positive relation with T while Test Data Accuracy doesn't seem to be affected. Number of faces correctly detected doesn't seem to be affected either. (Number of green boxes/ Total number of boxes)

Method 1 200 Pictures	Train Data Accuracy	Test Data Accuracy	Detect Image 1	Detect Image 2	Own Image 1	Own Image 2	Own Image 3
T=1, thr=0, pol=1	77.50%	68.50%	3/4	4/15	1/2	3/5	6/9
T=2, thr=0, pol=1	77.50%	68.50%	3/4	4/15	1/2	3/5	6/9
T=3, thr=0, pol=1	85.50%	73.00%	3/4	11/15	1/2	5/5	6/9
T=4, thr=0, pol=1	80.50%	72.00%	2/4	6/15	1/2	5/5	6/9
T=5, thr=0, pol=1	91.5%	68.5%	3/4	11/15	2/2	5/5	7/9
T=6, thr=0, pol=1	89.0%	73.0%	2/4	8/15	1/2	5/5	6/9
T=7, thr=0, pol=1	90.5%	67.5%	2/4	10/15	2/2	4/5	7/9
T=8, thr=0, pol=1	93.0%	68.0%	2/4	11/15	2/2	5/5	9/9
T=9, thr=0, pol=1	94.5%	66.0%	3/4	11/15	2/2	4/5	9/9
T=10, thr=0, pol=1	92.0%	70.0%	3/4	8/15	2/2	4/5	7/9
T=11, thr=0, pol=1	97.0%	64.0%	3/4	11/15	2/2	4/5	9/9
T=12, thr=0, pol=1	95.5%	68.0%	3/4	10/15	2/2	4/5	7/9
T=13, thr=0, pol=1	98.0%	62.0%	3/4	11/15	2/2	4/5	9/9
T=14, thr=0, pol=1	96.5%	67.5%	3/4	10/15	2/2	4/5	7/9
T=15, thr=0, pol=1	96.0%	66.5%	3/4	10/15	2/2	4/5	7/9



Hold T and polarity as a constant and vary the threshold from -3 to 3. Train Data Accuracy seems to be affected slightly by the threshold. The higher the threshold, the higher the accuracy. Test Data Accuracy seems to have an even less but still noticeable relation with the threshold. Still, the higher the threshold, the higher the accuracy. However, it seems that as the threshold gets lower, more green boxes are recognized in the detection of real images. It may just be that more false positives are happening so that anything will be detected as a face.

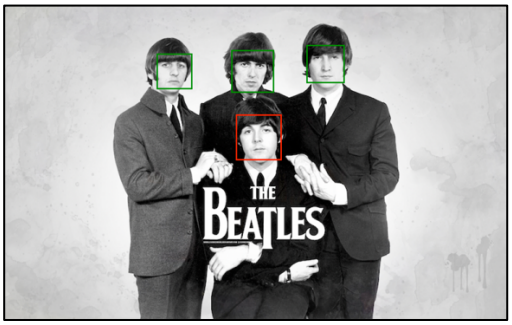
Method 1 200 Pictures	Train Data Accuracy	Test Data Accuracy	Detect Image 1	Detect Image 2	Own Image 1	Own Image 2	Own Image 3
T=10, thr=3, pol=1	95.0%	70.0%	2/4	6/15	0/2	4/5	2/9
T=10, thr=2, pol=1	93.0%	69.5%	2/4	8/15	1/2	3/5	3/9
T=10, thr=1, pol=1	94.0%	69.0%	3/4	7/15	2/2	4/5	4/9
T=10, thr=0, pol=1	92.0%	70.0%	3/4	8/15	2/2	4/5	7/9
T=10, thr=-1, pol=1	89.0%	67.5%	3/4	11/15	2/2	5/5	7/9
T=10, thr=-2, pol=1	89.5%	64.0%	3/4	13/15	2/2	5/5	8/9
T=10, thr=-3, pol=1	89.0%	63.5%	3/4	12/15	2/2	5/5	8/9



Hold T and threshold as a constant and vary the polarity with 1 or -1. It does not seem to have a positive impact.

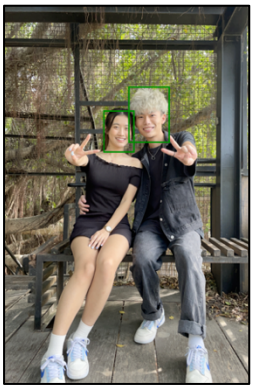
Method 1	Train Data Accuracy	Test Data Accuracy	Detect Image 1	Detect Image 2	Own Image 1	Own Image 2	Own Image 3
T=10, thr=0, pol=1	92.0%	70.0%	3/4	8/15	2/2	4/5	7/9
T=10, thr=0, pol=-1	21.0%	31.5%	1/4	11/15	1/2	2/5	3/9

The image result of the T=10, threshold=0, polarity=1:

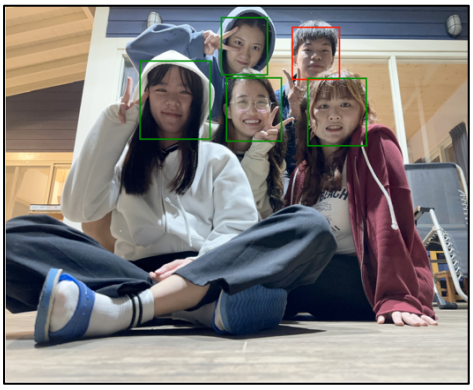


Detect Image 1


Detect Image 2



Own Image 1



Own Image 2

Own Image 3

For the bonus part, I used the sigmoid function for h(x) instead of the original one. The rest remains the same. Both accuracy of the train and test data seems to go down slightly. However, the difference is so little that the face detected in the real images made no difference.

```
            bonus: method 2 --> sigmoid function
            hj(xi) = 1/(1+e^(-xi))
            hj(xi) = |hj(xi) - label(xi)|
            ej     = Σi (wi * hj(xi))
            errorIndex: the index of min error
            bestError: the min error value
            bestClf: the classifer which has feature of the lowest error value
            '''
            h = np.abs(1/(1 + np.exp(featureVals)) - labels)
            e = np.sum(np.multiply(weights, h), axis=1)
            errorIndex = np.argmin(e)
            bestError = e[errorIndex]
            bestClf = WeakClassifier(features[errorIndex])
```

T=10, thr=0, pol =1	Train Data Accuracy	Test Data Accuracy	Detect Image 1	Detect Image 2	Own Image 1	Own Image 2	Own Image 3
Method 1	92.0%	70.0%	3/4	8/15	2/2	4/5	7/9
Method 2	91.5%	67.5%	3/4	8/15	2/2	4/5	7/9

Part III. Answer the questions:

1. Please describe a problem you encountered and how you solved it.

   A problem I encountered is that the success rate of the program was so low to the point of concerning. In part 1, I originally labeled the face pictures '1' and non-face pictures '0' as stated in the sides. However, after I finished part 2 and ran the program, the success rate of test data was always around 55% to 60%. It's really no better than guessing. Thus, I made a wild choice and switched the label hoping it would make a difference, and it sure did! Although it is not that big of a difference, but the results became more a more accept rate of 65% to 70%.

   I'm not really sure what made the difference since the labeling of '1' or '0' should not necessarily make a change in the decision making of the program. If professor Chen or the TA's have any idea, it would be fun to share it to the class!

   Another point I would like to mention isn't really a problem but an amazement at how numpy multiplication works. In part 2, I tried to multiply a 2D array with an 1D array when I was calculating the h(x) function. I didn't think much of it when I ran it the first time, but as it was running, I realized that they weren't the same dimension, so the multiplication shouldn't work. However, it finished running and everything seems to be fine. After a little bit of Googling, I found out that numpy will automatically elongate the smaller dimension array to fit the need of the bigger one. How convenient!

2. What are the limitations of the **Viola-Jones' algorithm**?
   a) Limited to frontal views: The viola-Jones algorithm is designed to work best on frontal views of faces. It may not work as well on faces that are tilted or turned away from the camera.
   b) Sensitive to light conditions: The algorithm relies heavily on contrast differences between light and dark regions of an image. This means that it may not perform well in situations with extreme lighting conditions. One way to avoid this problem is to apply a filter to the training and testing dataset so that the contrast is more evident.
   c) False positives: The algorithm can sometimes produce false positive detections, where it detects an object that is not actually present in the image. This can happen if there are features in the image that resemble the patterns the algorithm is looking for.
   d) Training data biases: The accuracy of the algorithm depends heavily on the quality and diversity of the training data used to train it. If the training data is biased towards certain demographics or environmental conditions, the algorithm may perform poorly on other datasets. In this homework, only 100 training data pictures were given. This may be the reason to the poor performance.

3. Based on **Viola-Jones' algorithm**, how to improve the accuracy except changing the training dataset and parameter T?

   As mentioned above, image preprocessing may help the accuracy rate. Image preprocessing techniques can be used to enhance the quality of the input images before they are fed into the algorithm. This can help to improve the accuracy of the algorithm by reducing noise, improving contrast, and enhancing the relevant features in the image.

   Some of the techniques are:

   a) Image normalization: This technique is used to adjust the brightness and contrast of an image to improve its visibility. Normalization can be done using histogram equalization or contrast stretching.
   b) Edge detection: Edge detection algorithms can be used to identify the edges of objects in an image. This can help to highlight the important features of an object and reduce the influence of noise in the image.
   c) Smoothing: Smoothing techniques, such as Gaussian blur or median filtering, can be used to reduce noise in the image. This can help to improve the accuracy of the algorithm by reducing the influence of noise on the detection process.

4. Other than **Viola-Jones' algorithm**, please propose another possible **face detection** method (no matter how good or bad, please come up with an idea). Please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm.

   Another face detection method could use the help of deep learning. Compared to the Adaboost algorithm, deep learning approach should have much higher accuracy, and are robust to variations. Modern technology in deep learning is capable of high accuracy in the realm of face detection. Moreover, they can have more variation in lighting, pose, angle, and scale of the tested images as they can learn relevant features directly from the data.

   However, there are some downsides to the deep learning approach. Compared to the Adaboost algorithm, it requires extremely high computational power and long training time to achieve its high accuracy rate. This makes it less practical for real-time face detection applications. It also requires a large amount of labeled training data. Collecting and labeling large amount of data can be time consuming and costly as mentioned in class. Lastly, it is difficult to interpret as the features learned by the model may not be easily understandable by humans. This makes it hard for humans to understand and troubleshoot the model.