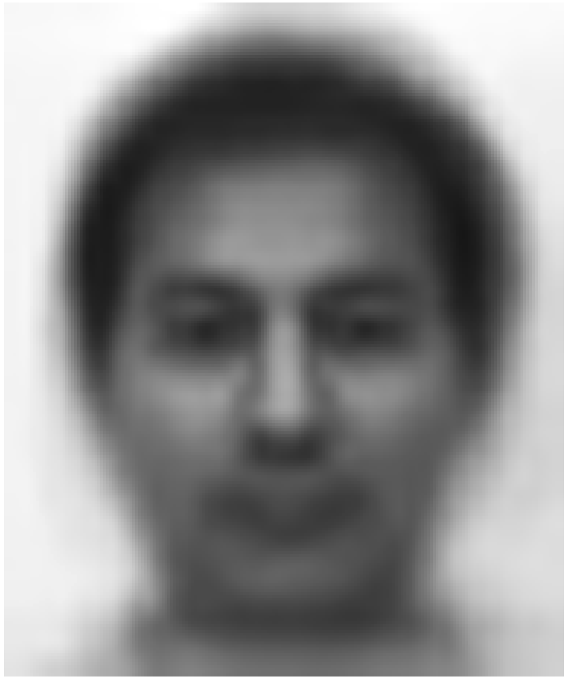
Computer Vision (TERM YEAR)

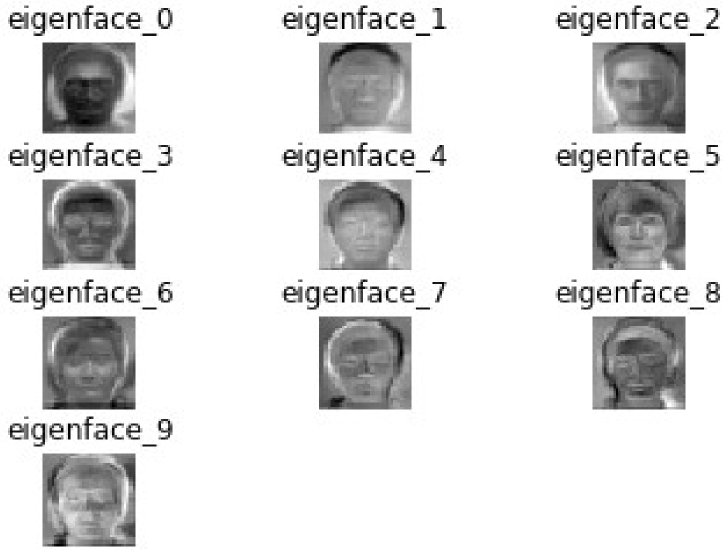
Problem Set #6

Darragh Hanley
darragh.hanley@gatech.edu

1a: Average face


ps6-1-a-1.png

1b: Eigenvectors



ps6-1-b-1.png

1c: Analysis

- Analyze the accuracy results over multiple iterations.Do these “predictions” perform better than randomly selecting a label between 1 and 15?

Below are the random accuracy results vs. the average PCA prediction over 10 iterations. You can see the the random results get nowhere near the PCA averaged results. The split % value is 0.5 and k =5 for all the below runs. For the random runs we randomly selected target predictions of 1 to 15 in each iteration.

```
------------------------------------------------------------
Part 1C : Random accuracy iter 0 : 4.85%
Part 1C : Random accuracy iter 1 : 6.67%
Part 1C : Random accuracy iter 2 : 6.67%
Part 1C : Random accuracy iter 3 : 6.67%
Part 1C : Random accuracy iter 4 : 7.88%
Part 1C : Random accuracy iter 5 : 4.85%
Part 1C : Random accuracy iter 6 : 8.48%
Part 1C : Random accuracy iter 7 : 6.67%
Part 1C : Random accuracy iter 8 : 7.88%
Part 1C : Random accuracy iter 9 : 5.45%
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
------------------------------------------------------------
PCA Good predictions =  51 Bad predictions =  32
61.45% accuracy over 1 iteration
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
63.86% accuracy over 10 iterations
```

1c: Analysis

Are there any changes in accuracy if you try low values of k ? How about high values?

Below the results for in increasing values of k. It can be seen that very low values perform significantly worse; however the results perform close to optumal at around 15 and above.

```
------------------------------------------------------------
PCA k: 1, 15.66% accuracy
PCA k: 2, 45.78% accuracy
PCA k: 3, 53.01% accuracy
PCA k: 4, 62.65% accuracy
PCA k: 6, 65.06% accuracy
PCA k: 8, 67.47% accuracy
PCA k: 10, 78.31% accuracy
PCA k: 15, 79.52% accuracy
PCA k: 20, 79.52% accuracy
PCA k: 30, 81.93% accuracy
PCA k: 100, 80.72% accuracy
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

1c: Analysis

Does this algorithm improve changing the split percentage p ?

Below the results for in increasing values of p. It can be seen that very low values perform significantly worse; however the results perform close to optumal at around 0.4 and above.

```
------------------------------------------------------------
Split p: 0.1, 46.31% accuracy
Split p: 0.2, 59.85% accuracy
Split p: 0.3, 62.07% accuracy
Split p: 0.4, 72.73% accuracy
Split p: 0.5, 67.47% accuracy
Split p: 0.6, 77.27% accuracy
Split p: 0.7, 74.00% accuracy
Split p: 0.8, 66.67% accuracy
Split p: 0.9, 70.59% accuracy
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

2a: Average Accuracy

Below can be seen the accuracy for the Random, Weak and Boosting in both Train and Test. We see with a split of 0.8 and 5 random runs, and with 5 boosted iterations the Training and Testing accuracy for each is very close. However the random significantly underperforms. The Boosting seems to do about 2% points better than the Weak classifier in both train and test.

```
p,split; Iters       ;Trn Rand; Trn Weak; Trn Boost; Tst Rand; Tst Weak; Tst Boost
[  0.8       5.       49.95305  87.48044  90.20344   50.5      87.       89.25   ]
```

2a: Analysis

Now we test the boosting accuracy over a different number of iterations, averaging the results each time over 5 runs.
We see that the number of iterations has a large impacts on the test and train accuracy. The more iterations the better.

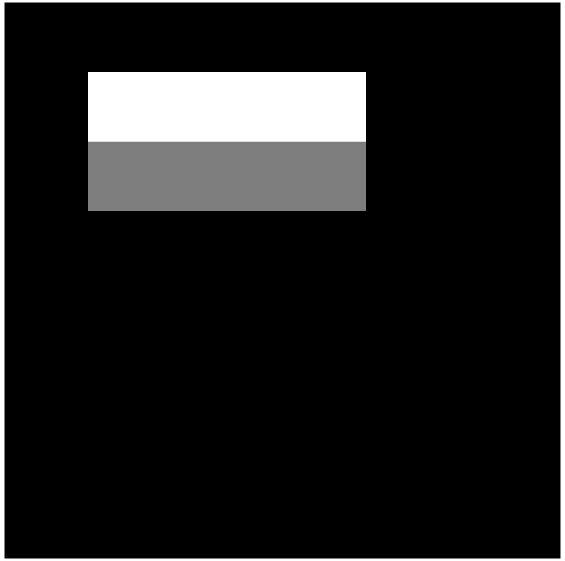
```
------------------------------------------------------------
p,split; Iters       ;Trn Rand; Trn Weak; Trn Boost; Tst Rand; Tst Weak; Tst Boost
[  0.8       2.       49.95305  87.48044  87.48044  50.5      87.       87.      ]
[  0.8       5.       49.95305  87.48044  90.20344  50.5      87.       89.25    ]
[  0.8      10.       49.95305  87.48044  94.99218  50.5      87.       92.5     ]
[  0.8      20.       49.95305  87.48044  99.31142  50.5      87.       99.25    ]
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

2a: Analysis

Now we test the boosting accuracy over a different splits, each time using 10 boostings iterations, averaging the results each time over 5 runs.
We see that the number of iterations has little impacts on the test and train accuracy. It seems to be only random change.
This is interesting as, p of 0.2 is significantly less image data as the rest of the runs. It seems the algorithms are robust to lower data sets.

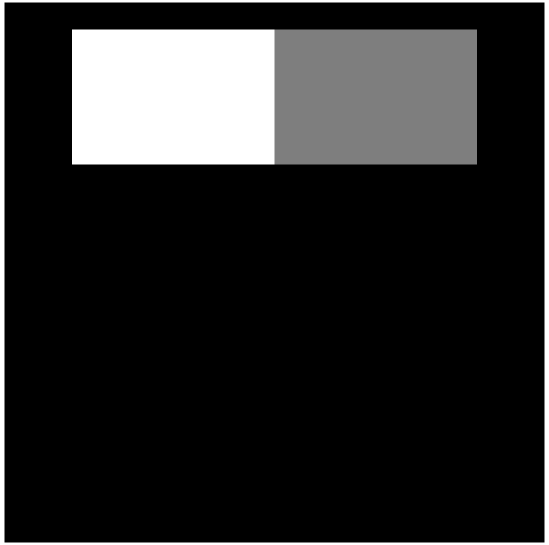
```
------------------------------------------------------------
p,split; Iters       ;Trn Rand; Trn Weak; Trn Boost; Tst Rand; Tst Weak; Tst Boost
[  0.2      10.       49.68553  87.67296  96.98113  50.09375  85.28125   93.3125 ]
[  0.4      10.       51.72414  87.7116   95.98746  49.70833  87.08333   93.5    ]
[  0.6      10.       48.51775  87.80793  95.36534  50.125    86.3125    93.5    ]
[  0.8      10.       49.95305  87.48044  94.99218  50.5      87.        92.5    ]
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

3a: Haar Features



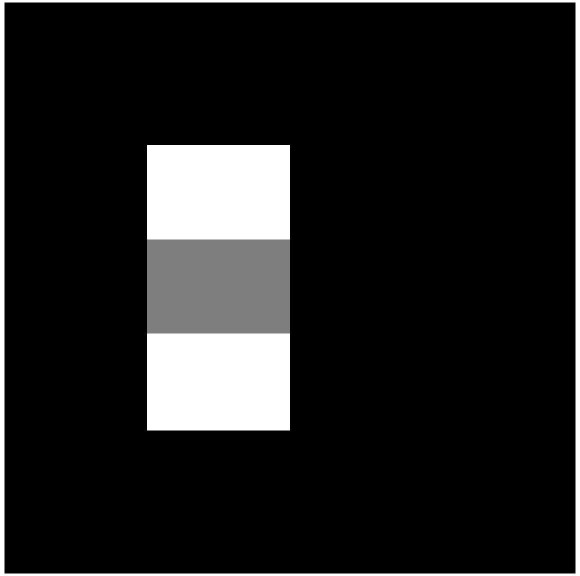
ps6-3-a-1.png

3a: Haar Features



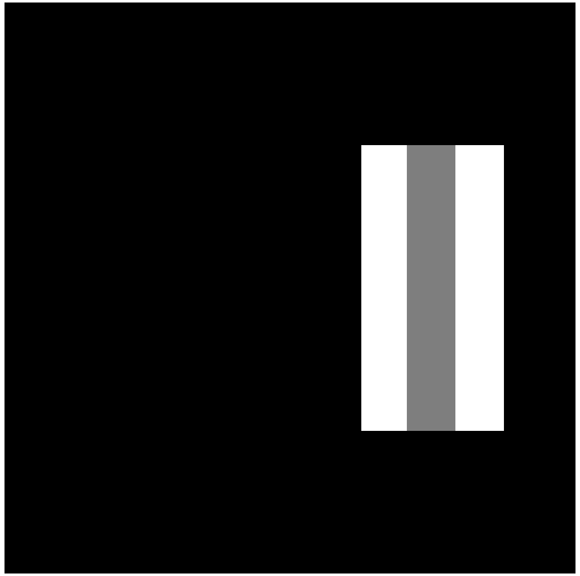
ps6-3-a-2.png

3a: Haar Features



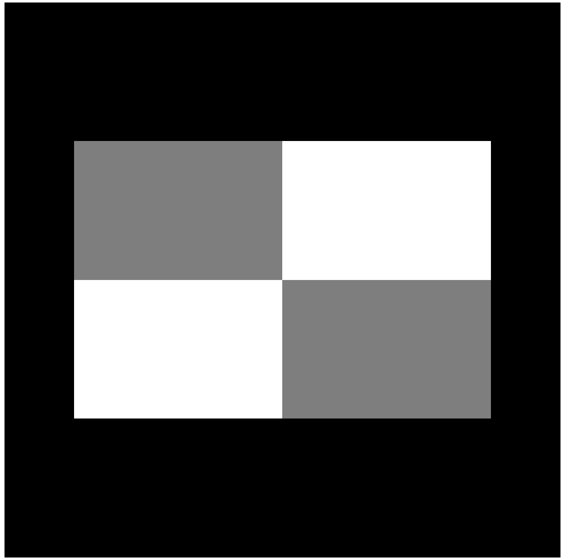
ps6-3-a-3.png

3a: Haar Features



ps6-3-a-4.png

3a: Haar Features

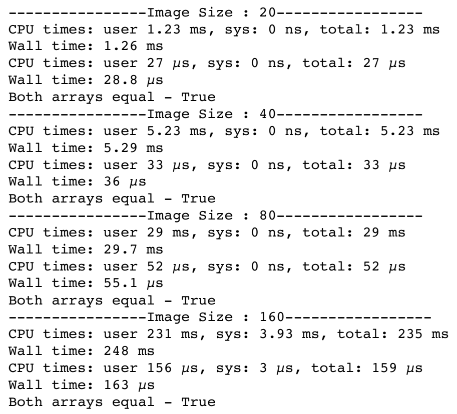


ps6-3-a-5.png

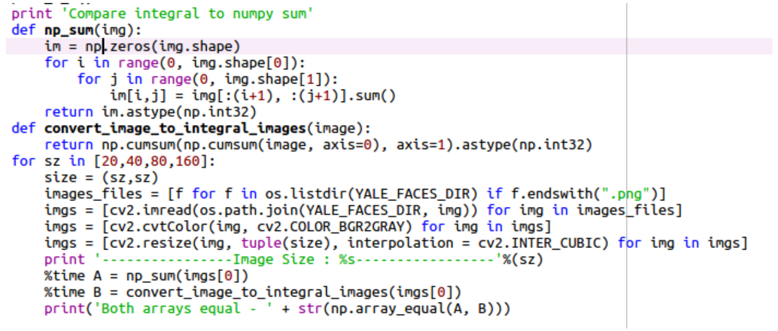
3c: Analysis

Text Answer: How does working with integral images help with computation time? Give some examples comparing this method and np.sum.

Numpy sum was seen to take a lot longer that integral arrays. Numpy recalculates the values for each position in the array over the respective axis; however integral arrays are calculated once and the values are stored in memory.
With integral arrays we can take advantage of numpy’s cumsum using just the points of the slice of the array.
Below can be seen time results of numpy sum vs integral array along with the code.

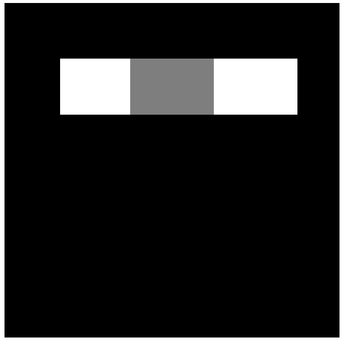


```
----------------Image Size : 20-----------------
CPU times: user 1.23 ms, sys: 0 ns, total: 1.23 ms
Wall time: 1.26 ms
CPU times: user 27 µs, sys: 0 ns, total: 27 µs
Wall time: 28.8 µs
Both arrays equal - True
----------------Image Size : 40-----------------
CPU times: user 5.23 ms, sys: 0 ns, total: 5.23 ms
Wall time: 5.29 ms
CPU times: user 33 µs, sys: 0 ns, total: 33 µs
Wall time: 36 µs
Both arrays equal - True
----------------Image Size : 80-----------------
CPU times: user 29 ms, sys: 0 ns, total: 29 ms
Wall time: 29.7 ms
CPU times: user 52 µs, sys: 0 ns, total: 52 µs
Wall time: 55.1 µs
Both arrays equal - True
----------------Image Size : 160-----------------
CPU times: user 231 ms, sys: 3.93 ms, total: 235 ms
Wall time: 248 ms
CPU times: user 156 µs, sys: 3 µs, total: 159 µs
Wall time: 163 µs
Both arrays equal - True
```

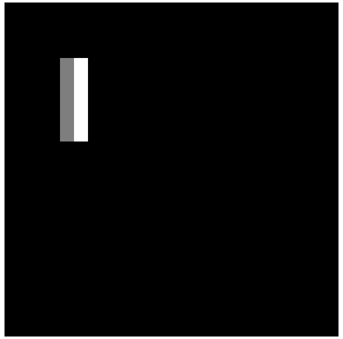


```
print 'Compare integral to numpy sum'
def np_sum(img):
    im = np.zeros(img.shape)
    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            im[i,j] = img[:(i+1), :(j+1)].sum()
    return im.astype(np.int32)
def convert_image_to_integral_images(image):
    return np.cumsum(np.cumsum(image, axis=0), axis=1).astype(np.int32)
for sz in [20,40,80,160]:
    size = (sz,sz)
    images_files = [f for f in os.listdir(YALE_FACES_DIR) if f.endswith(".png")]
    imgs = [cv2.imread(os.path.join(YALE_FACES_DIR, img)) for img in images_files]
    imgs = [cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in imgs]
    imgs = [cv2.resize(img, tuple(size), interpolation = cv2.INTER_CUBIC) for img in imgs]
    print '----------------Image Size : %s-----------------'%(sz)
    %time A = np_sum(imgs[0])
    %time B = convert_image_to_integral_images(imgs[0])
    print('Both arrays equal - ' + str(np.array_equal(A, B)))
```

4b: Viola Jones Features


ps6-4-b-1.png

4b: Viola Jones Features



ps6-4-b-2.png

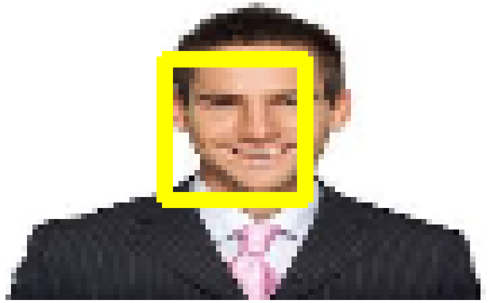
4b: Analysis

Report the classifier accuracy both the training and test sets with a number of classifiers set to 5. What do the selected Haar features mean? How do they contribute in identifying faces in an image?

Prediction accuracy on training: 94.29%
Prediction accuracy on testing: 68.57%

The Haar features represent filters which can represent the properties of the face, or the most relevant features. These features, when used give the minimum error rate for classifying the faces.
In the detection phase of the Viola–Jones algorithm, a window is moved over the input image, and for each subsection of the image the Haar-like feature is calculated. This difference is then compared to a learned threshold that separates non-objects from objects. Because Haar-like feature is only a weak learner or classifier a large number of Haar-like features are necessary to accurately describe an object. In the Viola–Jones object detection, the Haar features are organized in a classifier cascade to form a strong learner or classifier.
In recent years Haar features have been mostly replaced in object detection by CNN methods such as RetinaNet, RCNN or Yolo – there build up dynamic feature bank, where each feature adapts over training to best classify the objects in data inputted.

4c: Viola Jones Face Recognition



ps6-4-c-1.png