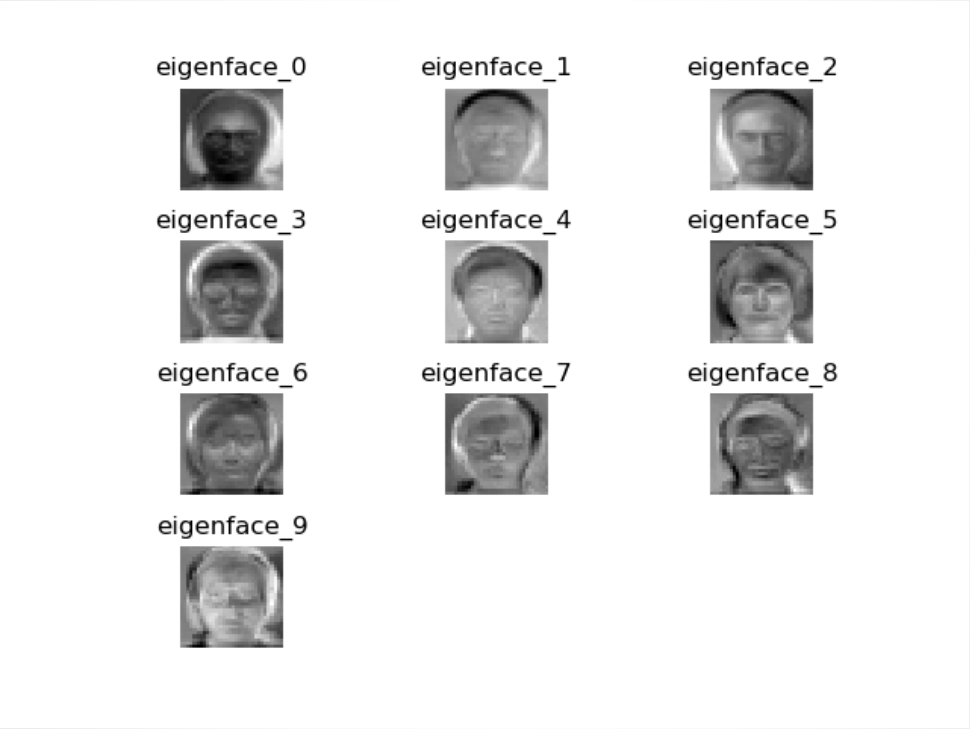
Computer Vision (Spring 2019) Problem Set #6

Alon Amar
aamar32@gatech.edu

1a: Average face

ps6-1-a-1

1b: Eigenvectors



ps6-1-b-1

1c: Analysis

Analyze the accuracy results over multiple iterations. Do these "predictions" perform better than randomly selecting a label between 1 and 15? Are there any changes in accuracy if you try low values of k? How about high values? Does this algorithm improve changing the split percentage p?

The average score over 5 iterations is 68.43%, but the highest was 74.7% and the lowest was 61.45%. Since our dataset is randomly selected, we might miss some of the faces completely (all facial expression of the same face) or not have enough representation of them. That means, that it will be hard for our classifier to identify a certain face and cause a low score.
Randomly selecting a label generates a score of an average of 6% - which make sense since the probability to be right for all images with random selection is $\left(\frac{1}{15}\right)^{83}$ (83 is the number to images to recognize).
As we increase k, our score goes up since we are basically increasing the resolution of our PCA, using more eigenvectors that represents our space more tightly. Lowering k will cause the opposite effect – we are loosing information.
Lower p increase our chances to miss some faces completely, thus, lowering the final score. Increasing p above 0.5 on the other hand, doesn't necessarily mean we have a better score, since we are covering more than half of our dataset. As p get closer to 1, we might even get lower score, since we are limiting our test pool to an handful of images – if we look at the extreme (only 1 image in test set), we can get only 0% or 100%.

Initial Scores
68.67%
74.70%
67.47%
61.45%
69.88%

2a: Average accuracy

Report the average accuracy over 5 iterations. In each iteration, load and split the dataset, instantiate a Boosting object and obtain its accuracy.

	Run 1	Run 2	Run 3	Run 4	Run 5
Random Training	51.02%	48.67%	49.30%	50.23%	52.27%
Weak Training	87.48%	88.89%	85.60%	87.48%	87.95%
Boosting Training	90.45%	90.14%	88.89%	92.49%	89.36%
Random Test	54.38%	50.62%	51.25%	49.38%	49.38%
Weak Test	86.25%	82.50%	93.75%	86.25%	86.88%
Boosting Test	90.00%	83.12%	96.25%	86.88%	91.25%

2a: Analysis

Analyze your results. How do the Random, Weak Classifier, and Boosting perform? Is there any improvement when using Boosting? How do your results change when selecting different values for num_iterations? Does it matter the percentage of data you select for training and testing (explain your answers showing how each accuracy changes).

We can clearly see how Boosting is the best out of the three, and it is an improvement of the weak classifier. Random is way behind. As boosting use more than one weak classifier, we are increasing the resolution of our solution, improving on some images that a single weak classifier couldn't identify.

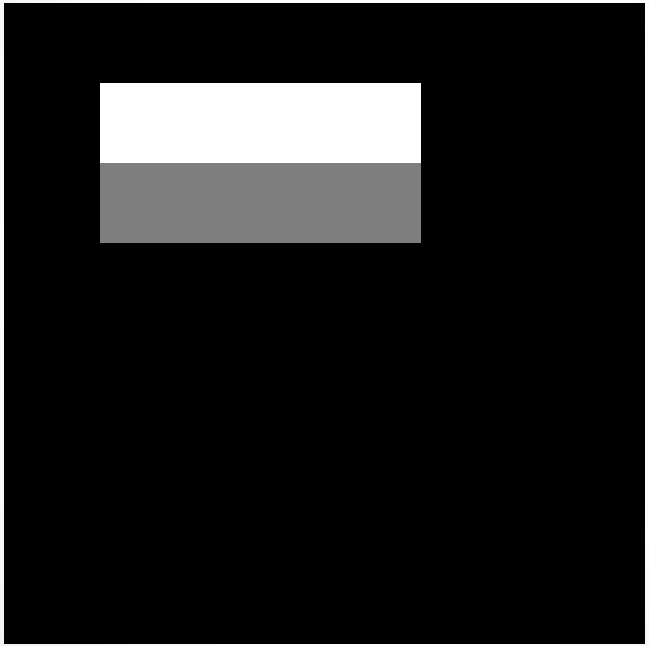
Our number of iteration determine our number of classifiers we use, thus increasing the number of iterations, increase our number of classifiers, improving our resolution and giving more weights for the wrong labels. Higher num_iterations increase the score to near 100% (num_iterations = 20).

num_iterations	5	10	15	20
score	87.50%	90.62%	96.88%	99.38%

The value of p doesn't really change the score, unless it is really small (0.05). In that value we don't feed our classifiers with enough information about the world.

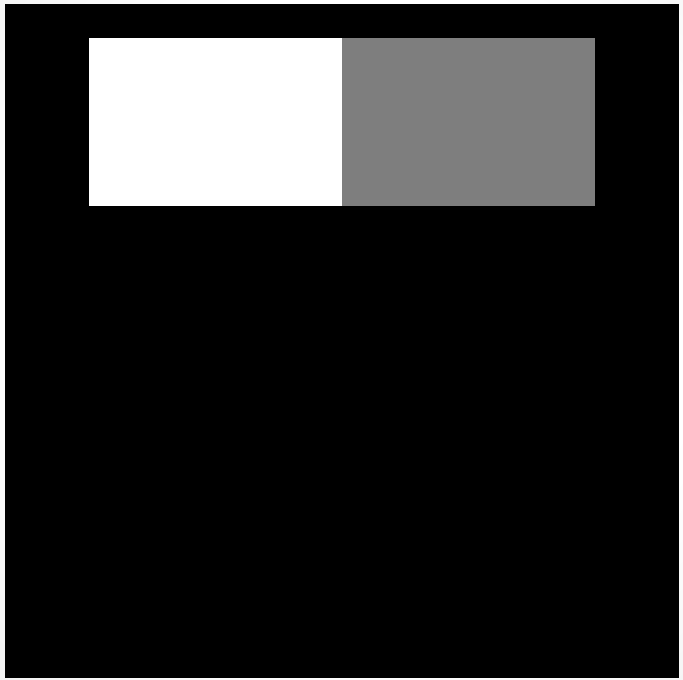
p	0.05	0.2	0.4	0.6	0.8
score	81.95%	88.26%	88.31%	90.00%	89.38%

3a: Haar Features



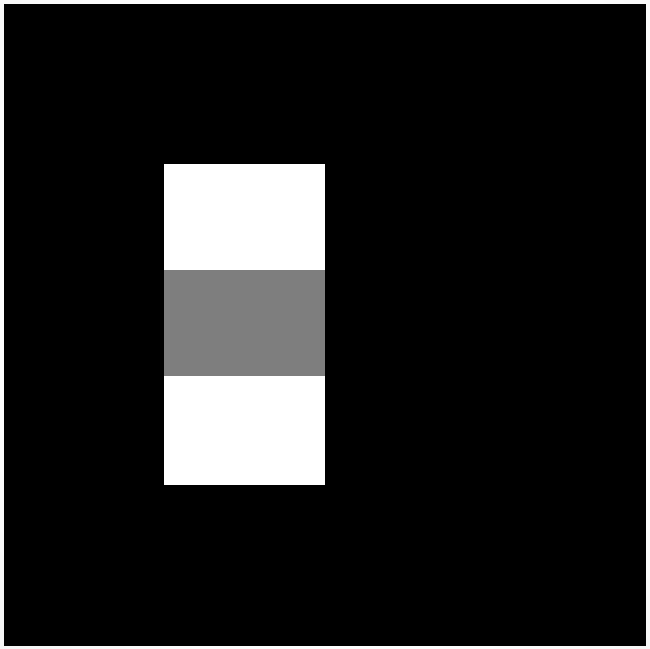
ps6-3-a-1

3a: Haar Features



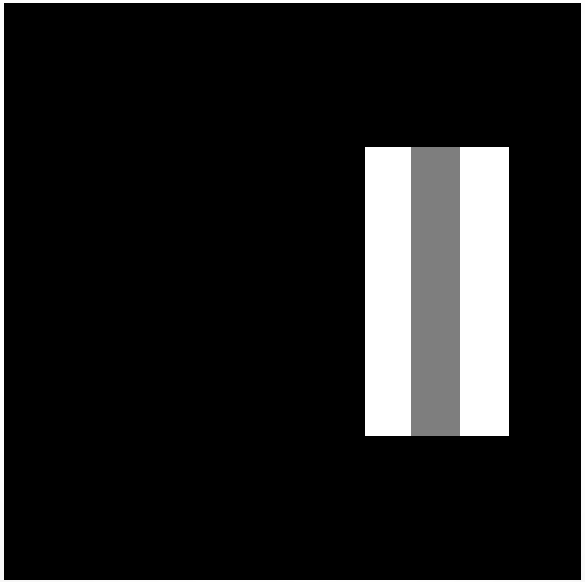
ps6-3-a-2

3a: Haar Features



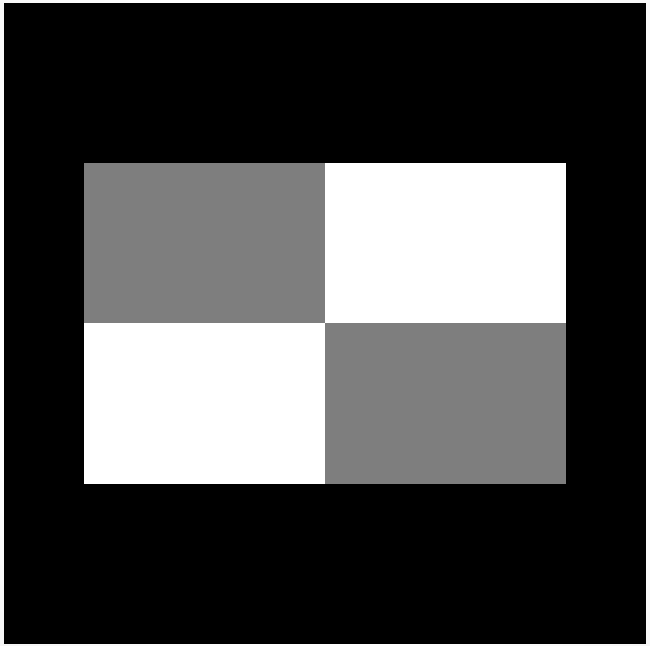
ps6-3-a-3

3a: Haar Features



ps6-3-a-4

3a: Haar Features



ps6-3-a-5

3c: Analysis

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

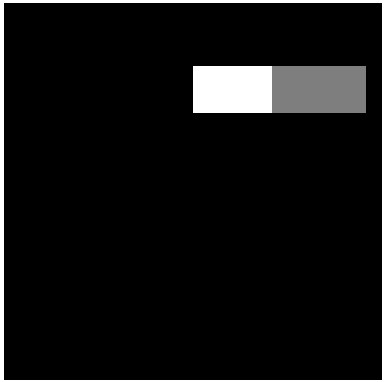
Integral images decrease computational time -> The only operations that we need to preform are 4 array accesses, 2 additions and 2 subtraction (Not including preprocessing).
If we would have used np.sum, what happens behind the curtain, is a summations of N*N array. That includes N*N array accesses and additions.
Since we are doing this calculation for each Haar feature per image, the time adds up.

If we want to describe the 2 methods in time complexity, integral image is O(1), while np.sum is O(N^2) – N is the size of the window.
If we add up the total use of that function, assuming the number of our images is M, and number of feature is F, we get:
O(M*F) vs. O(M*F*N^2), which is critical. (Not including the extra calculation for each gray and white patch)

4b: Viola Jones Features

ps6-4-b-1

4b: Viola Jones Features



ps6-4-b-2

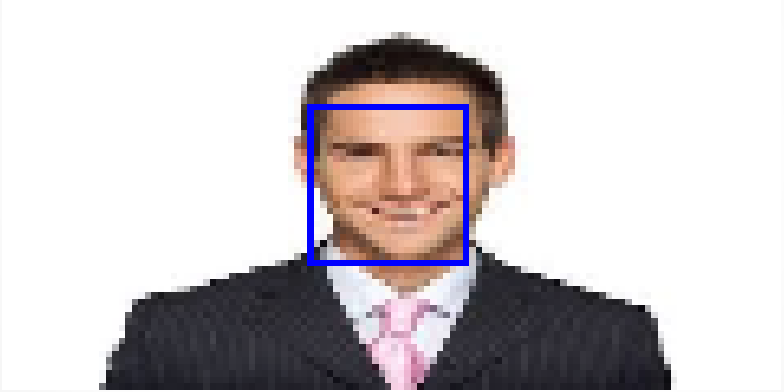
4b: Analysis

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: 98.57%
Prediction accuracy on testing: 77.14%

The selected Haar features are the features each weak classifier (VJ_Classifier) picked with the lowest error. Each feature represent each classifier decision process. They are the best features that can discriminate between faces and non-faces.
Each feature gives us information on changes in the image (like a big gradient).
In our case, it seems like the first feature (ps6-4-b-1) might describe the left part of the face the includes the right eye of a person – the gray patch is the eye.
The second feature (ps6-4-b-2) seems like the left eye, only is a lower resolution.
Besides them, we have additional 3 feature (5 classifiers) that try to describe a face in a unique way.

4c: Viola Jones Face Recognition



ps6-4-c-1