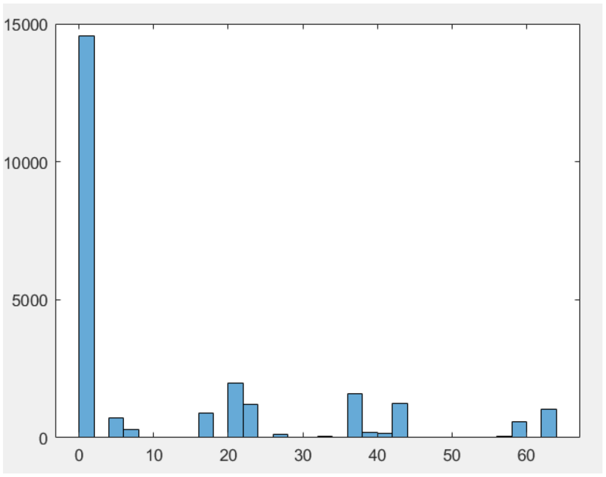
Abhiram Durgaraju

CSE 803 Homework 3

Problem 1:

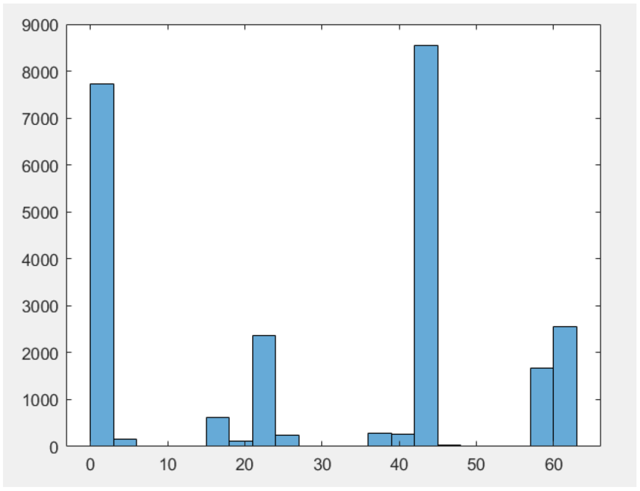
My program visionhw3_q1q2.m computes a histogram given an input image. Here are sample histograms:

‘302001.jpg’



Using gimp, we can sample several colors from the face. One example color is [214, 166, 143] in RGB. By taking the two most significant bits of each color, we get the following binary value: 111010. This is 58 in decimal. We can look at the histogram to see there is a small spike at 58, but not much. Using multiple sample pixels, we can look how big the spikes are on histogram for corresponding face colors to make judgements about the presence of a face. This image does have a face, but it is hard to determine this from just the histogram.

‘388070.jpg’



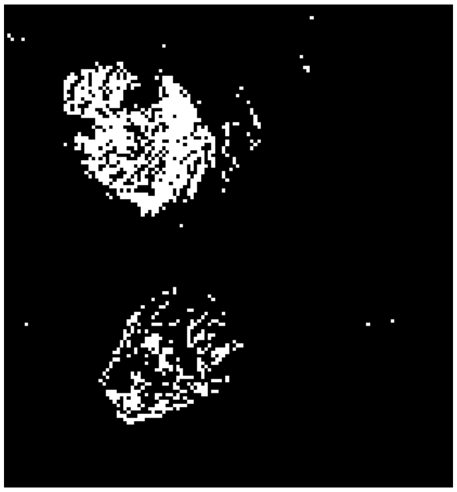
Again, using similar logic, one example color is [237, 188, 173]. Taking two most significant digits, this is again 111010, which becomes 58 in decimal. Bigger spike is seen in this image, suggesting more skin color pixels are present. Indeed, there is a face in this image. The presence of the arm skews the histogram since it has similar color distribution as the face

Abhiram Durgaraju

Problem 2:

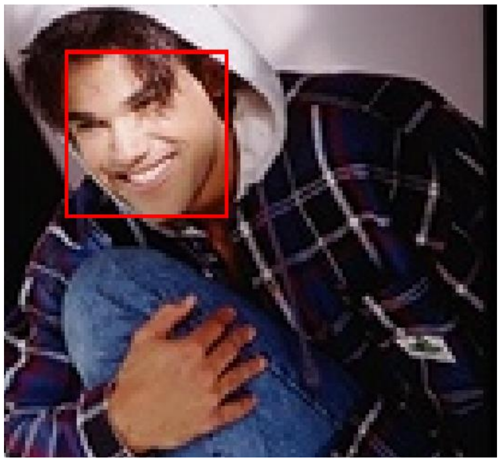
I studied various face colors using gimp and input them as cell into MATLAB. I gave an error range of 10, and within this range, I labeled all pixels that fall within these colors in the training set to 1, and the rest to 0. With this you can color segment the face to an extent, though inaccurate at times. Red bounding boxes are used when there is a higher certainty of a face. Blue bounding boxes are used when there is a lower certainty of a face. Currently this part of the program can only detect one face, but can easily be modified to detect multiple face with an extra nested for loop. My crude method (**region feature**) of identifying a face in a connected component image after foreground color segmentation is finding the component with the **largest area**. Other ideas I had were to use circularity, but the implementation of perimeter through code was challenging and for the last homework and I did this by hand so I opted to simply use area instead. This isn't very accurate but works for some images. The second half of 'visionhw3_q1q2.m' implements this problem.

I used images '302001.jpg', '388070.jpg', and '4.jpg' for training. Below is a sample binary image after color segmentation:

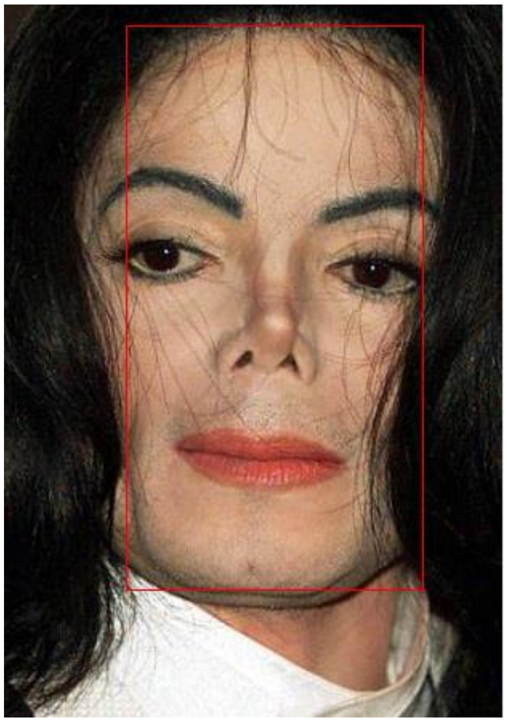


Below are 5 images after running through my program:

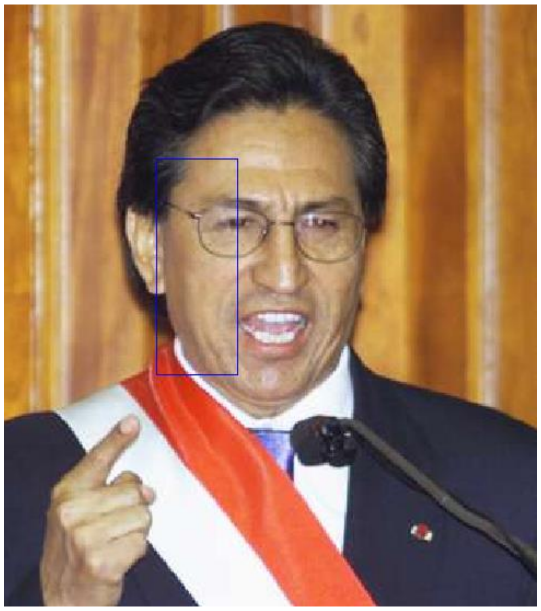
'302001.jpg'



‘302.jpg’

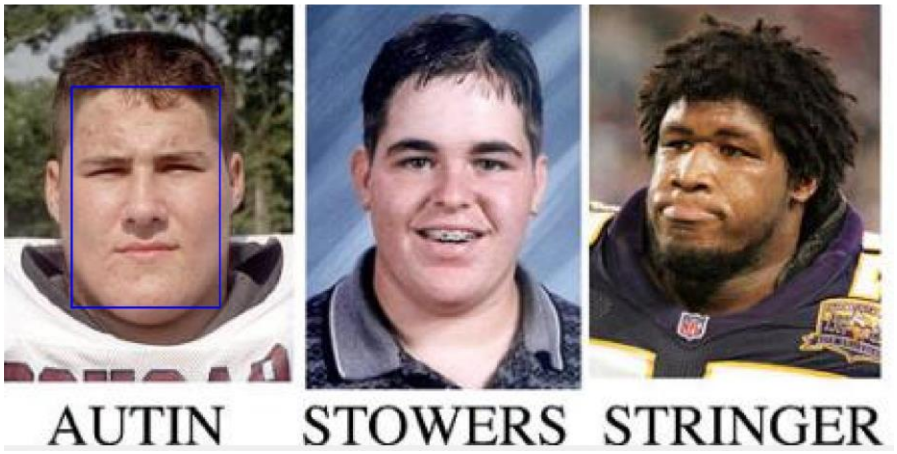


’10.jpg’



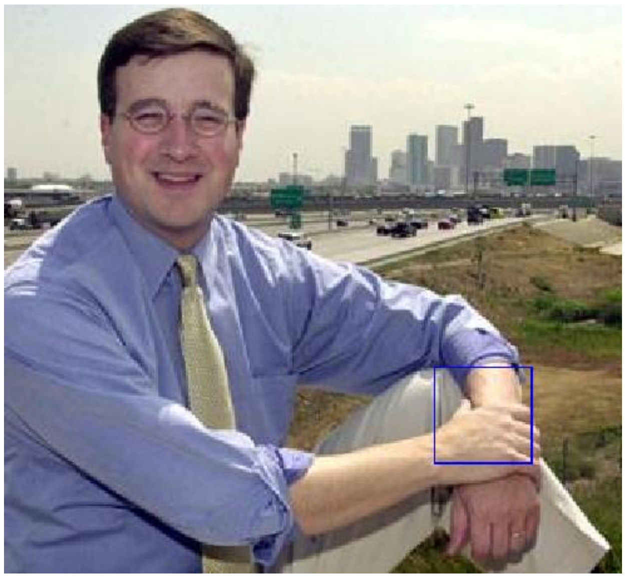
This is an example of an image with a blue bounding box, i.e low certainty. Notice that the pitfall of using largest area is that if the components are not connected adequately for the entire face due to small training set, this can happen, and does happen frequently.

’11.jpg’



Certainty was calculated by diving area of labeled pixels of interest over the area of the bounding box. If it was over 0.4, it was considered high probability of face and the box was set to red color. So, even when the bounding box was fairly accurate, the predictive value is not very helpful and more adjustment may be needed. 0.4 was the arbitrary value chosen from various test samples, but more thorough testing needs to be done. Also, as seen above, this can detect one face, since it is set to detecting max area currently.

’14.jpg’



Images where skin colored pixels besides the face are present pose a challenge to this method of face detection. Since more area of connected components are covered by the hands in this picture, the program isolates this part of the image. More diverse criteria than just area are needed to get weed out these false-positives, such as possibly circularity, bounding box ratio etc.

Problem 3:

Unfortunately, I was unable to implement a fully working Viola-Jones face detector using MATLAB in the last couple of weeks. Though, I will outline everything I was able to achieve below. I implemented most of it, but due to a problem with Adaboost returning the same weak learner after every round of boosting, I could not progress any further in my implementation.

Firstly, I used the training images that were supplied to us, and used the 'bbox.txt' files to crop out the face out of every image. I did this for every face on every training image and saved it into a known directory, which led to a total of 585 positive (face) training samples. I made sure to turn these images to greyscale and resize these to 24 x 24 beforehand to make it easier with application of the hear features.

To get a negative training set, the idea I had was to use a random x_left and y_top coordinate, with the same bounding box size as the face for a particular image and create an image using 'imcrop' in MATLAB. I then checked whether or not the resulting image had any overlap with the face of the image (IOU or bboxOverlapRatio). If this confirmed that indeed this 'imcrop' had no overlap with face, I considered this a negative sample and used 'imwrite' to write these images to another directory, after turning them to grayscale and resizing them 24 x 24. In the end, I only had a total of 217 negative (non-face) training samples.

My file 'sample_gathering.m' was responsible for getting all of these training samples. The coding for this was quite tricky and challenging since file IO in MATLAB isn't very intuitive. In hindsight, I should have come up with a better method for picking the negative samples since 217 is a very small sample size and the final obstacle in my homework may have been a result of this. I realized too late that it is better to have a larger negative training set since most of the 24x24 windows in a given picture will actually be non-face. The training images (positive and negative) are in the zip file.

Then, I created a function to calculate the integral image as this is the essential to speeding up the computation of the haar features. This was fairly straightforward, and my program 'integral_image.m' accomplishes this. It uses a recursive method to calculate the sum while checking to see if the image coordinates are within bounds.

Then, to implement the actual haar features, I created 4 separate nested for loops to implement each separate haar feature type. I used two 2-Rectangle features, one 3-Rectangle feature, and one 4-Rectangle haar feature. Computing the sum of pixels using an integral image for each feature depends on the size and type of feature. For 2-Rectangle features, you only need 6 pixel coordinates. For 3-Rectangle features, you need 8 pixel coordinates. For 4-rectangle features, you need 10 pixel coordinates. Of course, my program also needed to check if the feature bounds are within in the actual image. This was challenging to implement and took quite some time especially because I got array out of bound error for a long time during this process. The final feature array for both positive and negative training data were computed along with their [row,column,width,height] values saved in cell of 1x5 for each feature. This program is highlighted in 'haar.m'

This was done for every image and saved using fileIO in matlab, as shown in 'compute_haar.m'. The resulting array was saved as 'positive_training_set_all_features.mat' and 'negative_training_set_all_features.mat', and are all attached in the zip file.

The next step was starting to implement Adaboost. This required first initialize the weights of positive training images to 1/2m and negative images to 1/2l. So I set them to 1/585 and 1/217 respectively. Then, I normalized the weights according to the formula written in the paper so that it was a probability distribution and that they added up to 1. The challenging part was figuring out how to train an classifier by finding the optimal threshold which results with lowest error for each feature. Multiple methods can be employed to solve this, but I chose to compare the distribution of means of the positive and negative training samples. Then, I took the absolute value of the difference between the two distributions with the range of x as the two means. I took the minimum value of this distribution to arrive at the optimal threshold for each feature. Then, I set the polarity to +1 if the mean of the positive images was greater than the mean of the negative images and polarity to -1 if it was not. I saved the results of these efforts to 'threshold_array_set.mat' and 'polarity_features_set.mat'. My code for this inside 'viola_implementation.m'. I have commented out most of the code so it is easier to segment and read.

Then I trained the classifier by comparing the threshold and polarity to the computed feature values for every single feature and gave it a value of 1 if it classified it correctly and a value of 0 if misclassified. This was stored in an array (585+217) x 95k. This is seen in 'classifier_array_set.mat'.

Then, I calculated the error for each feature, by multiplying the weights with the absolute value of the classifier minus image_type (0 for positive training sample, 1 for negative training sample). This was then stored in the first row of 'error_rates_set.mat'. I chose the feature with the minimum error as my first weak classifier. Adaboost told me that feature 12036 (out of 95348) was the feature with the least weighted error of 0.2388. This was my first weak classifier.

Here is where I ran in problems:

Obviously, a strong classifier is composed of multiple weak classifiers, and therefore I needed to run Adaboost again. I updated the weighted errors by calculating beta and followed the equation under step 4) as outlined in the Viola Jones paper. Yet, Adaboost still gives feature 12036 as the best weak classifer in round 2 of boosting (it now has an error of 0.093). I tried removing feature12036 from the pool of features so see if Adaboost is working adequately. It was able to give me a different feature for round 2, but from round 3 onwards, multiple features ended up having the same lowest error, which caused further issues in picking the best weak classifiers.

Though the Adaboost was unsuccessful in giving me meaningful features past round 2, the core idea that multiple weak classifiers resulting in a stronger classifier with a lower error rate was shown clearly. One weak classifier had an error rate of 0.2388. Two weak classifiers combined into a strong classifier for an error of 0.093.

So, this is where I got stuck in the implementation. If I was able to solve this issue, the last portion (testing) of the problem is trivial. All I would have to do is write a code to run a scanning 24x24 window across an image and test only the features that were selected by the strong classifier. Only a window which satisfied this weak classifier would then draw a bounding box around the region.

I apologize for being unsuccessful in fully implementing this algorithm, though I put a lot of effort into coding what I had and stored all my variables and code in the zip for you to see. I am new to image processing and MATLAB so I am still learning. Please be generous with grading the report!