

Assignment 4: Face Detection

Before you start:

Download the assignment's files [here](#).

The project is graded **1/3** written assignment and **2/3** code assignment.

Written Assignment:

Once you have finished the coding assignment (described next), you'll need to finish the written assignment. The written assignment is in the file "Written Assignment 4.pdf".

Coding Assignment:

For this assignment you'll be implementing a face detector. There are three main steps: computing features, learning a classifier and detecting faces in a query image. Feel free to reuse any code from previous assignments. For background, I would recommend reading the paper "Robust Real-time Object Detection", Viola and Jones, Int. Journal of Computer Vision, 2001 (it's included in the homework download called "ViolaJonesIJCV2001.pdf".)

In the project, you should only have to update one file "Project4.cpp. " The buttons in the UI will call the corresponding functions in "Project4.cpp. "

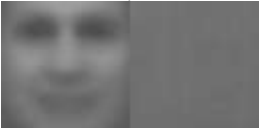
What to turn in:

To receive credit for the project, you need to turn in the completed file "Project4.cpp", any additional files needed for compiling, the requested images and files for each task, and the written assignment.

Tasks:

Step 1: For the first step, you'll compute features for the training dataset. Follow these steps:

1. Implement DisplayAverageFace. This function should display the average over all positive and negative training instances. That is, compute the average image over all face images (positive instances) and background images (negative instances.) The results should look like this:



The average face is on the left and the average background is on the right. For this assignment you don't need to worry about color, you can just use the green channel for all questions.

2. Implement IntegrallImage. This computes the integral image as described in class. That is, every pixel should be the sum of the pixels to its upper left. To test your results, open image "IntegralTest.png" and compare it to "IntegralAnswer.bmp".

3. Implement ComputeFeatures, and its two helper functions SumBox and BilinearInterpolation. ComputeFeatures takes as input the randomly initialized weak classifiers. To see how the weak classifiers are initialized, please see InitializeFeatures. Each weak classifier contains 2 or 3 boxes (specified by m_NumBoxes) defined by the member variable m_Box. Multiply the sum of pixels within each box by m_BoxSign and add them together to find the corresponding feature response. Store the resulting values in the array "features. " Compute the sum of the pixels within a box using the helper function SumBox. SumBox in turn uses the helper function BilinearInterpolation to handle fractional pixel values.

Hint: The values of m_Box store the upper left-hand corner and lower right-hand corner of the box in NORMALIZED coordinates ((0,0) is upper left-hand corner and (1,1) is the lower right-hand corner.) To compute the corners of the box in the image you'll need to multiply by the box size and add the offset. For example, the "x" coordinate of the upper left-hand corner is computed using $m_Box[i][0][0] * size + c0$, and the "y" coordinate is computed using $m_Box[i][0][1] * size + r0$.

Required: Open the dataset "TrainingDataSmall.txt" and press "Average Face". Save the resulting image to "1a.png". Open the image "IntegralTest2.png" and press "Integral Image. " Save the resulting image to "1b.png". Open the dataset "TrainingDataSmall.txt" and press "Compute Features. " Save the resulting image to "1c.png".

Step 2: Next, we are going to compute your classifier using AdaBoost. The AdaBoost classifier is implemented in the function `AdaBoost`. You need to implement two helper functions used by `AdaBoost`. Both of these functions are described in detail in the course notes and in the paper "Robust Real-time Object Detection" described above.

1. Implement `FindBestClassifier`. Given a specific feature, this helper function finds the best threshold, polarity and weight for a weak classifier, as well as its error. The feature's values for each training example are stored in "features" and the weights for every training example are stored in "dataWeights". You'll need to update the member variables `m_Polarity`, `m_Threshold` and `m_Weight` of "bestClassifier". You should return the error associated with the weak classifier. To help in computing these features, the array "featureSortIdx" stores the indices of the sorted features. When assigning the polarity, use either 1 or 0, with 1 = face above threshold, and 0 = face below threshold. Also note that for some machine this sorting might run slowly on Windows + VS2010, using Qt creator will be much faster.

2. Implement `UpdateDataWeights`. Given the addition of a new weak classifier, `UpdateDataWeights` updates the weighting of each training example. The new weak classifier and the feature value and label of each training example are passed into the function. After updating the weights, make sure they sum to 1.

Hint: The function `AdaBoost` outputs a text file called "AdaBoost.txt" (For Mac users it will be inside the executable package. You can use `show content` to find it.). Compare your numbers to those found in "AdaBoostTA.txt." The numbers won't be exactly the same since the features are chosen randomly, but you should notice a similar trend, i.e. you should see the error slowly decrease.

Required: Open "TrainingDataMedium.txt". Press "Compute Features". Press "AdaBoost". Save the resulting image as "2.png" (this took 30 seconds to compute on my machine.) The first two features should look fairly similar to the features shown in "Robust Real-time Object Detection" in Figure 5. Save the resulting classifier as "classifier.txt". Also turn in your file "AdaBoost.txt".

Step 3: In this step you will implement the functions necessary to detect faces in a query image.

The function `FindFaces` has already been implemented for you. It searches over all scales (between `minScale` and `maxScale`) and positions in the image for a face. At each location it calls the helper function `ClassifyBox` to compute the classification score for a face existing in the box specified by (`c0`, `r0`) and size. (`c0`, `r0`) is the upper left-hand corner of the box, and size is the width and height of the box.

1. Implement `ClassifyBox`. This should return the classification score for the box computed using your AdaBoost classifier. Hint: You should be able to use `ComputeFeatures` as a helper function.

2. Implement NMS (non-maximal suppression.) This step removes overlapping face detections. If two neighboring face detections exist within `xyThreshold` of each other in position AND `scaleThreshold` in scale, remove the less confident face detection.

Required: Open the classifier "classifier.txt" from the previous step. Open the image "barca2.jpg". Press "Find Faces" with default settings. Save the resulting image as "3a.png". Press "NMS" with default settings. Save the resulting image as "3b.png". Adjust the parameters to "Find Faces" and "NMS" to find the best results over the four images in the "query" directory, and save the resulting images as "3c.png", "3d.png", "3e.png" and "3f.png".

Bell and Whistles (extra credit)

(Whistle = 0.01 point, Bell = 0.02 points)



The function `InitializeFeatures` uses 3 of the 4 features described in "Robust Real-time Object Detection". Add the fourth feature (looks like 2x2 boxes) to the randomly generated features by updating `InitializeFeatures`. Show before and after face detections. Does this increase the accuracy of the face detections?






Try training using different datasets sizes and numbers of candidate weak classifiers. To do this make a new training data set file "TrainingData*.txt". The format is:

```
# of training examples (20000 max)
64 (patch size, should always be 64)
# of candidate weak classifiers (pick as many as you have memory for)
# of weak classifiers selected by AdaBoost (should be less than previous line)
faces (don't change, this is the directory containing faces)
```

background (don't change, this is the directory containing background patches)

Show face detection results before and after changing the parameters. Do any changes boost the performance?

  Implement the cascaded classifier described in "Robust Real-time Object Detection". You'll have to implement a cascade of AdaBoost classifiers. This should significantly reduce your time needed to detect faces.

  Gather another dataset of 64x64 patches of another type of object and train your classifier. Can you recognize anything else other than faces?