Face Detection

Mutian "Joshua" LIU

November 18, 2015

1 Highlights in Design of Implementation

- **Overarching Idea**
  Since I wanted to combine the object-oriented programming method and the matrix or vector manipulation method (which is commonly seen in R or MATLAB code) in this project, I wrote my code from scratch. During my coding, I made reference to the code by Nedelina Teneva provided with the homework assignment and the code by Simon Hohberg from GitHub[1]. Doing so makes my code more extensible (e.g. extend from default 2 types of features to all 5 types of features) and makes the analysis of results easier.

- **Harr-Like Features**
  I implemented all 5 types of features of dimensions (1, 2), (2, 1), (2, 2), (1, 3), (3, 1). For details, see `feature.py`. Due to the constraints of my computer, I did not compute all possible Harr-like features. Instead, I used stride of 4 for both scaling the feature (e.g. dimension increases from (1, 2) to (5, 10)) and moving the feature (e.g. upper-left coordinate moves from (0, 0) to (0, 4)), resulting in 38640 features in total. I trained all 4000 images, which took 1654.2108 seconds.

- **Cascading in Training**
  Given that in the final cascading, the images that are classified as non-face by the first strong classifier are not going to be processed by the second classifier, I trained my strong classifiers accordingly. When training the n-th strong classifier (i.e. n-th round of AdaBoost), I delete all the training images that are classified by the strong classifier given by $(n-1)$-th round of AdaBoost. In this way, the n-th strong classifier would focus exclusively on the images that are detected as faces by the $(n-1)$-th strong classifier, which will make it more specific to the purpose given the process of cascading.

- **False Positive and False Negative Thresholding**
  Because of the previous bullet point, I have to make sure that the false negative rate for each strong classifier is very low. I tried several times according to the tips given in the instruction to adjust Θ in order that there is no false negative at all. However,

[1] URL=`https://github.com/Simon-Hohberg/Viola-Jones`

this turned out to be problematic: if Θ is set to be the lowest score positive images got, then in the next round of training, there will be no training image classified as non-face, which will result in the selection of exactly the same set of weak classifiers, and thus make the result useless and AdaBoost run infinitely.

Instead, I used an alternative way to solve the problem. I set a threshold of false positive rate (FPR) and a threshold of false negative rate (FNR). At the end of each textititeration of AdaBoost, I run the strong classifier constructed so far on the whole training set. If the training FPR and FNR are acceptable, then return the strong classifier and start the next round of AdaBoost. In practice, I want the false negative to be near to zero, so the threshold of FNR I set was 0.01. I train on the first 500 positive images and first 500 negative images and use the rest of the training set as the validation set, and choose the threshold of FPR as 0.3.

As a result, doing so made the total number of cascading layers very small (in my case 3) and the number of weak classifiers in each layer large (in my case 19, 19, 26).

- **Caching of Results**
  Since the entire process from calculation of features to AdaBoost, and finally to running cascade on testing image is very long, in each step I used Python library `pickle` to serialize the result so far to the disk so that I can start from an arbitrary point in the process.

- **Face Detection on Testing Image**
  On the testing image, we slide the 64×64 window throughout the image to detect faces. From my experiments, I found that moving the window with step of 10px generates the best result.

2 How to Run the Code

- **Preparation**
  Make sure to put 2000 positive training images and 2000 negative training images into `faces/` and `background/` directories respectively. Put the testing image `class.jpg` into the same directory with the `.py` scripts. Be sure to have all 5 `.py` files before running. Check that there is at leat 5 GB space left on your disk.

- **How to Run**
  Change working directory to the directory in which you store all the scripts. Run `python boost.py` to get the strong classifiers. After the previous step is finished, run `python detect.py` to get the face detection on the testing image.

- **About Outputs**
  After running `boost.py`, there will be two new directories created. In the directory `saved_0to1999/`, the calculated features and integral images are stored. In the directory `boost/`, the result of AdaBoost is stored. After running `detect.py`, there will be no new files created. The final result will be shown directly.

3 Display of Result

Figure 1: Face Detection on Testing Image

4 Discussion on Result

- **General Comment**
  We see that nearly all (56 out of 57) faces are detected, which means that the FNR of the final classifier is low. However, we see a lot of non-face objects (such as hands, clothes, and the wall) are detected as faces, which means that the FPR is high as compared to FNR. This is expected, given my particular implementation aforementioned.

- **Training and Resulting Cascading Layers**

Layer	# Weak Classifier	Time (sec)	FPR	FNR	Detection Rate
1	19	232.7071	5.15%	4.80%	95.03%
2	19	241.5122	2.65%	12.00%	92.68%
3	26	246.7746	0.45%	21.00%	89.28%
Total	64	720.9939	1.55%	7.20%	95.63%

Figure 2: Training Errors of Single Layers and Final Cascading Classifier

- Note that FPR, FNR are calculated based on 2000 positive/negative images, and detection rate is calculated based on all 4000 pictures. The FPR, FNR, and detection rate items for "total" row are based on the result of cascading of strong classifiers.
- We see that the FPR goes down while FNR goes up as we go deeper in the cascading layers because of the design that a cascading layer does not need to take care of images deemed non-face by previous layer. Though we see that the cascading layers on their own are becoming worse in terms of detection rate, when they are used in a cascading way, they together perform better than any single one of them. From this observation, we see that cascading does work and improve detection rate.

- **Features Selected in Cascading Layers**
  Visualization of the top 5 features selected in every layer are shown in figure 3.

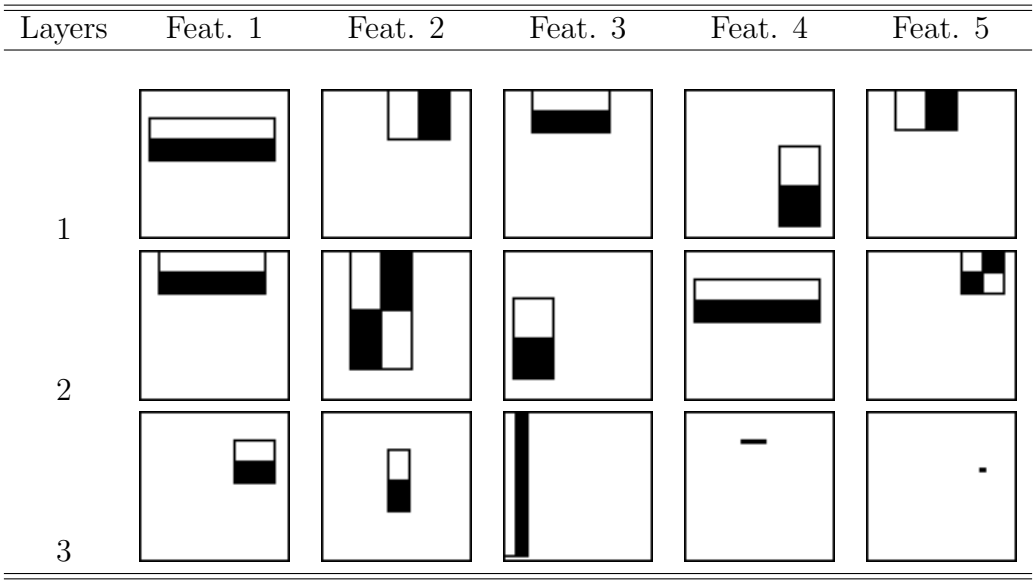

Figure 3: Top 5 Features in Each Layer

For most of the features selected, there is no very obvious connection to features of human faces. However, we can still take some rational guesses about the connection between them and faces.

- Feature 1 in layer 1 seems to cover the eye area.
- Feature 1 in layer 2 seems to separate human forehead from the background.
- Feature 2 in layer 2 seems to cover the whole face area.
- Feature 2 in later 2 seems to describe the distinction of nose and mouth areas.

But still, among the selected features, some of them do not provide much information (e.g. feature 4 and 5 of layer 3). This may be regarded as a piece of evidence that in machine learning, some black-box algorithm can generate very good result.

5 Further Discussion on Visualization of Result

In section 3, the result is displayed in a way such that each positive detection is shown by a rectangle with red border. However, we see that there are many overlapping rectangles, making the result very messy.

Considering this point, I tried two ways to improve the visualization of the final result. Both of them are generated by small steps (4 pixels) when scanning the testing image with 64×64 window (and thus the script works slower). By doing that, we assume that the faces are circled out by multiple windows. Let "intensity" of a pixel be the number of positive windows it is in.

Note that the result might be different from that in section 3 due to a different step size (4 instead of 10) in this section.

- **Intensity as Transparency**
  Color the pixels with yellow for which the transparency (alpha value) is determined by the intensity of pixels divided by the maximum intensity throughout the testing image. The result is cited in figure 4.

Figure 4: Illustration of Intensity as Transparency Method

We see that though we have a very good grasp about the relative intensity of every

pixel, it is not quite clear if some faces are detected or not because they are only contained by a small number of windows. In general, this is not quite an ideal way to visualize the result since the detection results are not discernible enough.

- **Thresholding on Intensity**
  In this method, we still count intensity of every pixel. However, we color the pixel as long as the intensity is not less than a threshold (in this case 6). The result is shown in figure 5.

Figure 5: Illustration of Thresholding on Intensity Method

This time, we see that the detected faces are shown more clearly than in the previous method and in section 3. By thresholding, we also eliminated some false positives in our final result (notice that the blocks on the ground detected as faces in section 3 are not shown as faces this time).

As far as I am concerned, I think that this is the best among the three methods I have tried to visualize the result. You can try this by un-commenting the function `testfill()` in the `main()` function of `detect.py` and run it.

6 Limits and Possible Improvements

This is a very short discussion about how we can make our result better. We see that except for some false positives on the ground and on the wall, most other false positives occur on people's clothes and body parts (especially hands, look at the two men's hands detected as faces in figure 1).

After having a glimpse on the training set for background, I found out that there are not many images of body parts other than face and of clothes. I think that just by adding some of those images will significantly lower the false positive rate on this particular testing image.

Also, if the computing power can support, more features (i.e. smaller strides or even covering all the possible features) may improve the detection rate in general. The setting of FNR and FPR in the AdaBoost step can also be set by mode rigorous cross validation.