CMSC 25400 Machine Learning
Homework 3

Deqing Fu

October 25, 2017

1 Inputs and Setup

1.1 Inputs

We are given 4000 training images with each having the dimension of 64×64. Among these 4000 training images, there are 2000 of them being faces and another 2000 being backgrounds. Due to computation bottleneck and time constraint, I only select 2,000 images as the training data, i.e. 1000 faces and 1000 background images.

1.2 Setup

Due to computation bottleneck, I'm using MATLAB as the programming language because of its power in matrix computation and built-in multiprocessing/multithreading optimization. The MATLAB version I'm using is MATLAB R2016A, and unfortunately, it does not allow users to combine all functions to one file, hence I wrote each function in its own independent *.m* file.

1.3 Pre-processing

The first thing we have to pre-process is the integral images data, i.e. the data we are using to train the model. The pre-processing process is written in the MATLAB file of *preprocessing.m*. It takes all 4000 images, and shuffles them with one face image and one background image. The data is stored in an independent file called *iimages.txt*.

Another preprocessing process is to generate the feature table. What I did is store the feature table coordinates, i.e. $x1, y1, x1, y1, x2, y2, x2, y2$ for the black and white part of each haar-like feature. And then I stored them into an independent file call *featuretbl.txt*. I choose the extension of the size of the haar-like features to be 3 pixels and the step of moving the haar-like features to be 3 pixels, and start generating these haar-like feature coordinates from the window size of 2×2. Hence, I got 50820 features. This will be used for the training.

2 Variables Functions

2.1 Variables

The main training script is written in the file *training.m*, in which N is the total number of data, and in this case I set it to 2000 (with 1000 images being faces and another 1000 being backgrounds). T is maximum number of boosting of every layer, and I set it to 20. The purpose of having this limit of T is that, in some cases, the False Positive Rate might vibrate around 0.4 for a long time before beginning to decrease to below 0.3. And for a single boosting, 0.3 and 0.4 is not a large difference. The variable *cascades* is the number determining how many cascades I'm using for training, and her I set it to 5.

2.2 Training Function

All MATLAB functions take $featuretbl$ and *images* as input variables because it's said that it's very slow in MATLAB to get access to global variables.

computeFeature(): the function $computeFeature()$ takes in the image index i, and the feature index f and two constant $featuretbl$ and *iimages* to compute the feature of certain image over certain feature.

bestLearner(): The function *bestLearner*() takes in a given weight w, and a given feature index f and three constant variables N, $featuretbl$, *iimages* to compute the threshold θ and polarity p for a single feature.

adaBoost(): The function *adaBoost*() takes in T being the maximum number of boosting, N being the total number of data in a certain layer, $featuretbl$ being the global variable (even though passed through the function, the value of it remains constant), and *iimages*, *labels* being the integral images and labels for the data of this layer of boosting. The return values of this function are the α's and the $weakLearners$ which contain the threshold θ's, the polarity p's and the indices indicating which feature is pivotal for this weak learner. We compute the **False Positive Rate** every round and determine if the value is under 0.3. If it's under 0.3, then we terminate the boosting and return the α's and $weakLearners$. If it's over 0.3, then we continue to boost another round. If the boosting process boost t rounds, then it returns t α's and t $weakLearners$.

strongLearner_helper(): This is just a helper function computing the Θ, and how the Θ is computing is written in *adaBoost*(). The $strongLearner\_helper()$ returns the summation of $\alpha_t h_t(x)$ of each input image. Hence in the *adaBoost*() function, it uses $strongLearner\_helper()$ to compute the $\alpha_t h_t(x)$ of all the faces, and then find the minimum of the value, and sets that value to Θ. Hence this will decrease the **False Negative Rate**.

strongLearner(): This function takes in an image index i and the *theta* computed by *adaBoost*() using $strongLearner\_helper()$ to compute the predicting value of a given image.

cascade(): The *cascade*() is the main function we are using and it takes in variables $featuretbl$ being the feature table, *iimages* being the data of all training images, *labels* being the label of the training images, *cas* being the number of cascades, N being the total number of data, and T being the maximum number of boosting of each layer. The return value of this function is a set of strong learners. And this is the final result we are using for detecting

2.3 Testing Functions and Scripts

predictStrongLearner() and **prediction**(): These two functions are testing counterparts of $strongLearner()$.

class_photo_preprocess(): The preprocess of class photo is just calculating the iimage (integral image value) of 64 sub-windows of the *class.jpg* image and store their coordinates information in the *coordinates.txt* file.

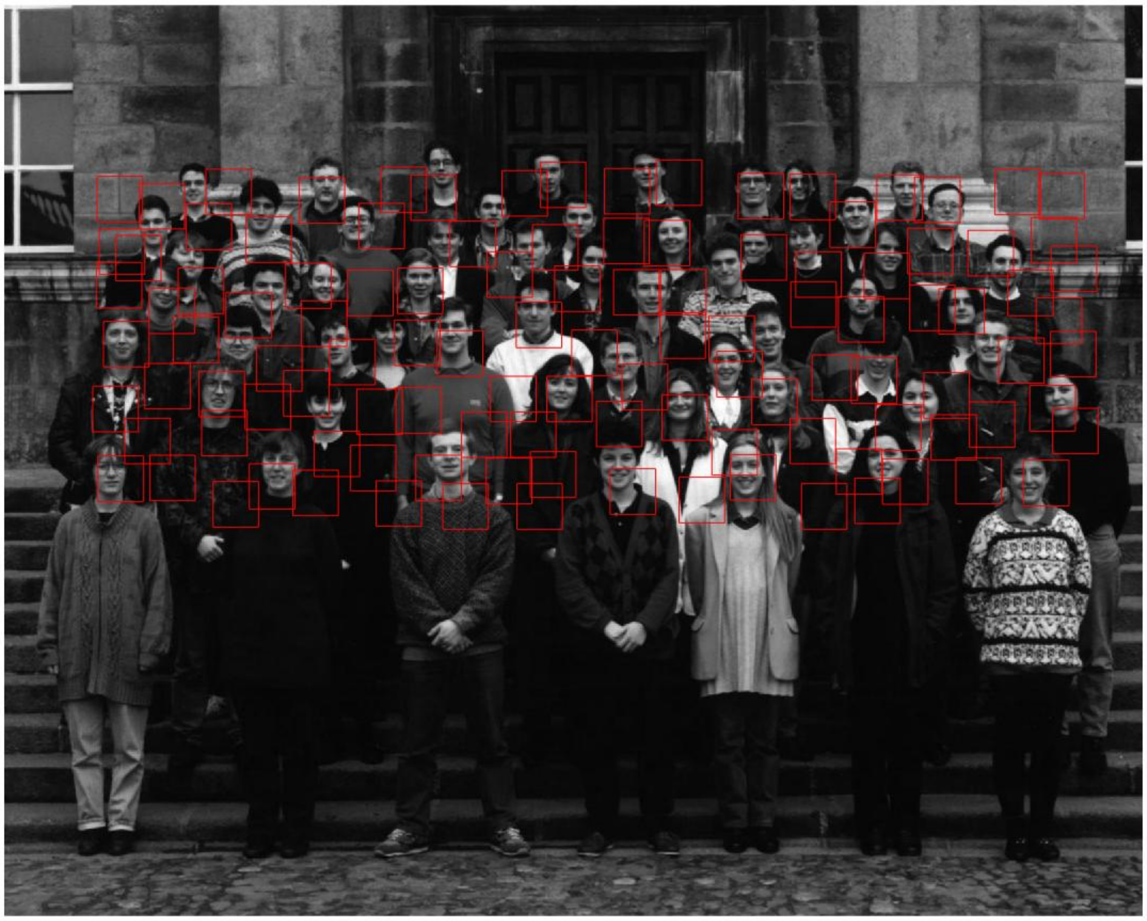
class_photo_testing(): this script just uses results from *cascade*() function to predict if every sub-window is face or background. After computing this, it eliminates the overlapping of detection by combine those detected faces' coordinates which are in the same circle of radius $32\sqrt{2}$. Then using MATLAB built-in functions to plot the result.

3 Results

# of layer	# of negative numbers (over 1000)	# of boosting	FPR	#total time to run
1	1000	10	0.281	4639.92 seconds
2	281	8	0.117	2173.77 seconds
3	33	20	0.818	4279.43 seconds
4	27	20	0.703	5217.99 seconds
5	19	20	0.632	4961.19 seconds

The training result is not so satisfying because after 2 layers of cascading, it has a hard time of reducing false positive rate, leading to that every layer fully used 20 boosting rounds, which is the manually set limit. The problem is, maybe I can choose a larger limit of boosting rounds of each layer and hopefully it can reduce the False Positive Rate.

Figure 1: Final Result



4 Comments

4.1 About the Result

My result is very unsatisfying. I made a lot of trivial errors at the beginning. The code did work correctly for a week. I only had the last two days of adjusting the values I choose to make it work fine. And unfortunately, it still not works well. The main error I made was I used Viola-Jones' paper's adaBoost algorithm, in which they set 0 to represent non-faces but I used Professor Kondor's labelling that -1 represents non-face backgrounds. Hence it took me tens of hours to debug and thanks to Angela Wu of helping me find out this huge bug on Tuesday, the day before the Wednesday deadline.

4.2 What need to change

4.2.1 Optimization of Code

Yes. As the codes consume so much time of running, it's pivotal to optimize the code. For the first few days after the homework was released, I chose Python to implement it. And unfortunately, I know near nothing about multi-processing Python and Python itself is so slow. And more importantly, I didn't vectorize so much during my Python implementation. Then I decided to switch to MATLAB, whose *parfor* keyword is simple to use for parallelization locally, i.e. multiprocessing/multithreading. But as I copied codes from Python to MATLAB, I still didn't do well on vectorizing. And more importantly, I ignored the fact that MATLAB access columns faster than accessing rows — the cache structure of MATLAB may be different from C family languages, and more likely to be similar to FORTRAN. Hence, these several things combined to make the code running not so fast, even though faster than the original Python code.

4.2.2 More Data

As we can see, the detector detected a lot of faces on people's sweaters, unsurprisingly. Because if you scrutinize the 2000 background data, nearly non of them are about people's clothes. Maybe more data on that will make the detector more accurate.

4.3 For the TA if he/she wants to run the codes

Please run the following scripts in order:
preprocessing.m
feature_tbl.m
training.m
class_photo_preprocess.m
class_photo_testing.m
Thanks!