Project #[Viola Jones]

CMSC 25400, Prof. Kondor
Ken Chen

Data Input and Pre-processing

1 Image data

For the training process, I took all the 2000 faces and 2000 backgrounds into as my training set, which are all in RGB format and comprise 64×64×3 pixels. To transform them into greyscale, I used the scipy.misc.imread function, with the flatten parameter set as $True$, so that the images were all coerced to greyscale representations by its default equation. Then, as suggested in the lecture, I used the fucntion **compute-integral-image()** to transform all these images into integral image representations, which eased the computation of feature values.

2 Feature Input

As suggested from the paper by Viola and Jones(2001), I've selected three shapes of rectangles as my filters: the "two-rectangle" filters as B in Figure 1, the "three-rectangle" filters as C, and the "four-rectangle" filters, where I calculated the pixel intensity difference as advised. All the features can be generated once with the function **compute-feature()**. There are in total 6400 "two-rectangle", 1336 "three-rectangle" and 1530 "four-rectangle" features.

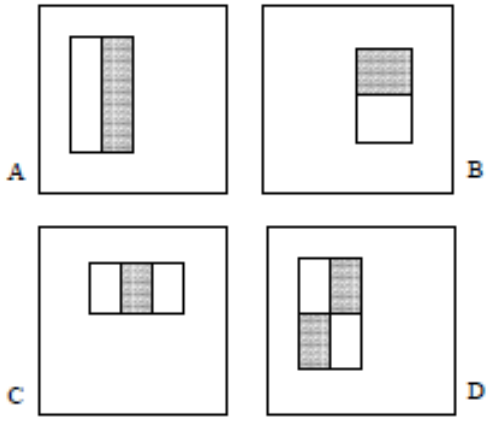

Figure 1: Example features from the paper

Training Process

1. Compute the features

Though advised against, I decided to compute the features for all images at the first step, stored them in a matrix, instead of computing them on the fly. The reason is that I don't have an formidably large number of features as my base learners, and

doing so can shorten the time consumption for later. Meanwhile, I sorted the images by their feature values for all features, so that I don't need to duplicate these $O(nlog(n))$ sortings at each round of boosting. All these information is stored in the matrix **feat-mat**, with another matrix **labels-mat** storing the labels associated with the sorted images for each feature. This step can be implemented by **compute-feat-mat()** and **compute-labels-mat()**.

2. Adaboost: build a strong learner

Essentially, I mirrored the Adaboosting process by Viola and Jones(2001), except for the weight updating process. The weight updating diverges in that:

- Viola, Jones:

$$D_{t+1,i} \leftarrow D_{t,i} exp(-y_i h_t(x_i))$$
$$D_{t+1,i} \leftarrow \frac{D_{t+1,i}}{\sum D_{t+1,j}}$$

- While for me, I combined the advice from the lecture and the paper:

$$D_{t+1,i} \leftarrow D_{t,i} exp(-\alpha_t y_i h_t(x_i)/2)$$
$$D_{t+1,i} \leftarrow \frac{D_{t+1,i}}{\sum_j D_{t+1,i}}$$

  The reason I took these strategies is that: (1) I divide the inner part in $exp()$ by half because it will otherwise overly separate the correctly and falsely predicted data, as our training set is not very large so that we can have a good error rate and a large α at the very first. And the model will be stuck a one particular feature. (2) I normalized the weights exactly by their sum, instead of using the Z_t, because sometimes the Z_t normalizer will give rise to errors rate above one.

I set the an upper bound of 30 for the each boosting iteration (to avoid overfitting at this classifier), so the iteration will be terminated at once when the 'artificial' false positive rate hit 0.3 or the iteration has been processed 30 times. A classifier(strong learner) can then be generated right away.

3. Cascades: chaining the classifiers

As each classifier has forced an artificially zero false negative rate:

$$h(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x) - \Theta)$$

I chained a sequence of classifiers, where each level of the cascade (a classifier) receives the samples classified as face from the previous one as its training set. Again to eschew overfitting on the training set, I set the maximal length of the cascade to

be 5. Each classifier can be trained with the function **adaboost()**, and then a new training set will be selected, together with new corresponding **feat-mat** and **labels-mat**. Then the training will proceed to the next level of the cascade. Successfully, I killed all the false positives at the fourth classifier.

4. Training Results

of layers	of negative samples	of boosting	False positive rate	Running time (seconds)
1	2000	7	0.278	1019.82
2	556	19	0.261	323.09
3	145	20	0.248	298.76
4	36	15	0.0	125.04

Table 1: Training results

Generally I am happy with these testing results. The convergence was bit slow at layer 2 and 3, where the false positive rates took 19 and 20 rounds of boosting to hit 0.3. Finally, we have no misclassifications at the end of this cascade.

Testing

1. Testing preparation

I inherited many of the productions from the training process: (1) the feature list. (2) The feature indexes, polarities(p), thresholds(θ), α, and the Θ for each layer of the cascade from the training process. These will be used for implementing prediction. Besides, I created the function **generate-coor()** that can form a 64×64 square can walk through the test-image, with steps of 32 pixels at a time, zigzaggedly. Then a list of coordinates can be produced. These will be thrown into my big cascade.

2. Testing process

At each of these coordinates, I sliced from the test-image a 64×64 image for the cascade. What the classifying rule differs from the training process is that here I inflated the Θ by 2.5 so that I can protect faces from being recognized as backgrounds. As the Θ's are chosen to fit the training set, and I have a good fitting of the training set, they might be too strict for the testing set. Moving across the list of coordinates, I tested the 64×64 images on the fly. Once I found a face at a coordinate, I will make a leap horizontally, so that I can avoid overlapping at a row. But I decided to not make leaps vertically, since I may catch coordinates that contain a face better.

3. Testing process

Figure 2: Testing result

The testing results are very disappointing. There are many missing faces and misclassified backgrounds. My cascade classifier is obviously very sensitive to areas where whites and blacks contrast sharply. I think the classifier can be optimized by including more complex shapes of features, so that the classifier may be able to identify facial characteristics that are distinguishable from other random objects, rather than just compare blacks and whites.