Viola-Jones Face Detection

Roberto Fernandez

1 Introduction

In this project I attempted to implement the face detection algorithm described in the 2001 Viola and Jones paper. For some further insights and help on as to how to improve the algorithm I supplemented that paper with their 2004 paper which has more detail on how to implement the algorithm, as well as the posted Piazza hints.

2 Implementation

2.1 Correctness

Th provided algorithm uses both of the 2−rectangle features mentioned in the Viola-Jones paper as well as the 3−rectangle feature which attempts to recognize the difference in pixel intensity between the nose and cheeks. It traverses these features (˜30000 when only considering 2−rectangle features and ˜45000 when considering all 3 possibilities) to find the one feature, polarity, and threshold combination that minimizes the weighted error of the training set and then uses these such features in a strong classifier, appending such features until our error on the non-faces we care about being under 40%. Once this threshold is reached we name our strong classifier one of the cascade classifiers and start the process anew with a smaller training set consisting of all faces and only the non-faces that our current cascade classifiers still mis classify (i.e. false positives).

The 40% threshold for a new classifier was chosen compared to the suggested value of 30% due to initially having too many false positives when testing, and thus hoping that having more cascade classifiers would provide a suitable trade-off between false positives and false negatives (which it did!). We thus get around 5 cascade classifiers, each with an increasing number of features which depend on the size of our training data set.

I also used the suggested Θ trick in order to ensure no false negatives during training and then replaced this part of the hypothesis in training by instead comparing $\sum \alpha_t h_t > \gamma \sum \alpha_t$ for some $\gamma \in (0, 1)$. This thus provided more flexibility when it comes to trading between more false positives or negatives, in order to obtain an acceptable face detection rate. In the given image it seems like a reasonable γ is $\frac{1}{3}$ since this leads to the best ratio of correctly detected faces to false positives but it remains to be seen if this is over-fitting to the given data, and if we wanted to minimize false positives we could always set $\gamma > \frac{1}{3}$ but this would also lead to less faces being detected.

Another thing I considered was the drawing of rectangles on the final testing image. Since the sliding window is passed over the image with a stride length of 2 we would usually have quite a few overlapping rectangles. In order to avoid this I create a list of all detected faces (the rectangles corresponding to them and the values, e.g. likeliness of it being a face) and then simply traverse it from 'most likely to be a face' to 'least likely' while ignoring any that overlap with rectangles we have already processed. This presents the issue, however, of detection in images such as the one given where some faces are so close that rectangles corresponding to different faces will sometimes overlap. This is not taken care of here due to time constraints but in the future a more advanced heuristic can be implemented to take care of this case with some threshold for overlapping, since

currently there are quite a few faces that are actually detected by our classifier but due to this overlap rule are not actually shown. ☹

2.2 Efficiency

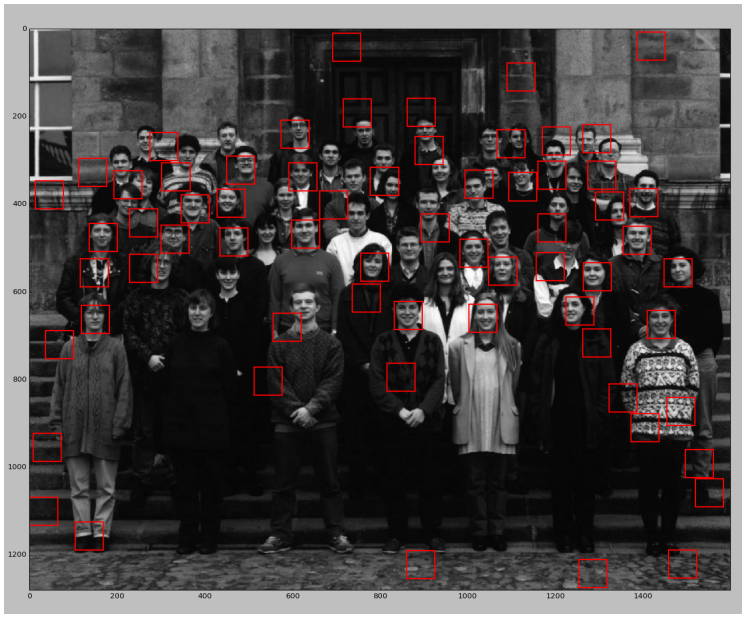
When it comes to efficiency some small changes had to be made in order to be able to train our algorithm efficiently (especially when considering all 3 types of features which increases our computation time greatly). In order to make training more efficient we do things such as pre-computing a $\Sigma = \bigcup_{f \in \mathscr{F}} \sigma$ array which basically contains the indexes that sort our feature values for every given feature over all images in the training set we care about. Σ only changes at the beginning of each cascade classifier when we remove some images and thus pre-computing it aids greatly, especially when the number of weak classifiers per cascade classifier is high. This can probably be further improved by simply removing the values we don't care about at each stage but that could introduce some bugs and the current implementation seems to train sufficiently quickly.

It would also probably be advantageous to simply not consider nonsensical features such as 2×1 rectangles since those are very unlikely to be good indicators of a face or non-face, but given that our current implementation trains on the entire data set in a couple hours this seemed unnecessary. Another interesting thing to note is that the 4−rectangle features actually seemed to not be very good indicators of a face or non-face since only about 1 of these features was ultimately selected to be in our classifiers, with most of the selected features being of the form of two horizontal rectangles with a few vertical rectangles and 3 rectangle features.

3 Results

If we run the learning algorithm on all 4000 images, where half are faces and half are not, using only 2 rectangle classifiers, we end up with a cascade classifier that has 5 different cascade functions, each with a varying number of weak classifiers that range from 7 to 23 in size, depending on how far into the cascade we are. We then run this cascade over the class.jpg image with a stride length of 2, and then use the overlap reduction method described above (which does not work well for faces near each other and thus could be improved given more time) and get the following result:

if we add the 3 rectangle classifier, however, the faces on the bottom row seem to be much better detected but we have a few more false positives throughout the edges of the image:



and then when we added the 4 rectangle feature we surprisingly had quite a bad classifier where many false positives and false negatives occurred:

in all these examples we can clearly see the shortcomings described above, such as the rectangles where we have many faces nearby being not showed due to our simple heuristic, and a few false positives due to setting our $\gamma = \frac{1}{3}, \frac{2}{3}, \frac{1}{5}$ (respectively in each case) in hopes of maximizing the number of faces we actually detect. Ultimately, the classifier that we have took around 1 hour to train and the training errors of the classifiers ranged from 0.065 for the best one to 0.3 for the worst, and if we run this cascade classifier on the training data again we get no false positives but around 20 false negatives since we remove the Θ that we had been using to artificially make the number of false negatives 0, thankfully this should be mitigated by the fact that the sliding window goes over the same face several times when we are testing. Ultimately, the optimal choice seems to lie between the 2 and 3 rectangle features given the thresholds that we set in our algorithm (clearly by changing these we should be able to get the case with 4 rectangles to actually improve our result) and given more time it would be possible to thus get better results.

4 Future Improvements

Looking at our results we can clearly see a few possible routes of improvement. Playing around with our values for γ and changing the Θ trick we are currently doing during training could also lead on improvements depending on how we decide to make sure that false positives are rare while carrying that property over to the testing classifier. It might also be possible to alter the thresholds that we set for a cascade classifier in hopes of being able to decrease the number of false positives, even if training time goes up. Furthermore, due to the early implementation of functions such as `compute_feature()` and `compute_strong()` it would definitely be possible to re-factor our code and consolidate our testing and training function suites.