Face Detection in the Wild using Viola-Jones algorithm

Neel Dungarani
neelvija@buffalo.edu
50338201

Ashish Avinash Naware
ashishav@buffalo.edu
50337024

Introduction:

In this report we will discuss a machine learning method for the identification of faces as an object in given Image using Viola-Jones. That is a trivial job for a human, but a machine requires specific instructions and constraints. Voila-Jones needs full straight faces to make the detection task more achievable. The whole face will then point at the camera and cannot be turned to either side. Though such restrictions can be avoided with the data that we use to train our model. In the whole model training algorithm there is three key factors that differentiate the process. The first is the implementation of a new representation of images, called “Integral Image” which makes quite fast computation of features used by our detector. The second is a learning phase which is AdaBoost based Algorithm, which selects a limited number of essential visual features from around 1,60,000+ features. And the last one is a method for combining increasingly complex classifiers in a “cascade” that allows non-face window of image to be discarded early in the process while investing further computation on more promising windows like faces.

Implementation:

We have distributed the whole implementation of Viola-Jones algorithm in mainly three sub-processes i.e., Data Collection, Training Model, and Testing Model. And these sub-processes are further divided into few steps. For data collection we will use FDDB [2] as the dataset for this project. FDDB contains more than 2800 images and associated bounding box annotation, for more than 5700 faces. In training the model we have implemented it in four stages: Haar Feature Selection, Creating an Integral Image, AdaBoost Training, Cascading Classifiers. All human faces share some similar properties. These regularities may be matched using Haar Features. Computing the integral image helps in computing features in constant time and adds a significant speed advantage. AdaBoost refers to a particular method of training a boosted classifier which eventually generates strong classifier by combining several weak classifiers. A cascade of gradually more complex classifiers achieves even better detection rates. The testing model is achieved using sliding window approach over a testing image. Which takes every window on a given image with all different size variations possible for a window.

Data Collection:

In data collection we have a face detection dataset composed of thousands of images, the goal is to train a face detector using the images in the dataset. First, we took 10,372 images from FDDB [2] as the face dataset to train model with positive data and for the non-face data we took images from Stanford [4] and further sliced it into 16 sub-images of background images around 4330 images to train model with negative input so that our model can easily determine what is not a face easily. After we have got our data in directory we labelled each image in python with 1 for face image and 0 for non-face or background image while reading to discriminate images while training. We have also created validation set of 100 positive face images to validate weak learners at each cascade level.

Training Model:

The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:

- Robust – very high detection rate (true-positive rate) & very low false-positive rate always.
- Real time – For practical applications at least 2 frames per second must be processed.
- Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

The algorithm has four stages:

1. Creating an Integral Image
2. Haar Feature Selection
3. AdaBoost Training
4. Cascading Classifiers

Creating an Integral Image [1]:

An image representation called the integral image evaluates rectangular features in constant time, which gives them a considerable speed advantage over more sophisticated alternative features. Because each feature's rectangular area is always adjacent to at least one other rectangle, it follows that any two-rectangle feature can be computed in six array references, any three-rectangle feature in eight, and any four-rectangle feature in nine. For every Image in our training data set we will compute integral image in numpy array and pass it further in process to extract features for image.

We have defined “get_integral_images(input_images)” method to compute integral images which takes list of images from training dataset as argument and returns list of numpy arrays of integral images.

Haar Feature Selection [5]:

Haar-like features are image features used in object recognition to create bag of features from image. And we can train model with features from multiple images using classifier.

For Viola-Jones there are 5 specific feature types which we can be extract from image which helps best determine that image is face or not. In this project we are computing all combination for 5 features which are: 1x2, 2x1, 1x3, 3x1, 2x2. By computing all these features, in total, we are computing in total 162,336 features for each image in training dataset.

As we have around 15000 images in our training dataset including both face data and non-face data so when we compute all possible combination of features for every 5 types of features then it takes too much time to compute so to overcome this problem we have implemented multi-threading when we compute features which reduces running time for feature extraction at least five times. We have used “concurrent.futures” in order to achieve multi-threading.

We have defined below different methods for different feature type (In total 5):

1. calculate_feature_1(integral_images): for two-vertical-rectangle features
2. calculate_feature_2(integral_images): for two-horizontal-rectangle features
3. calculate_feature_3(integral_images): for three-horizontal-rectangle features
4. calculate_feature_4(integral_images): for three-vertical-rectangle features
5. calculate_feature_5(integral_images): for four-rectangle features

These all above methods take list of integral images as argument and returns the matrix of Images X features for each type. After we have all five types of features for all images we combine all features in to single matrix and flatten all features for each image into single entry.

AdaBoost Training [1][6]:

Problems in machine learning often suffer from the curse of dimensionality — In our case each training sample consist of 162,336 Haar-features, and evaluating every feature can reduce not only the speed of classifier training and execution, but in fact reduce predictive power. Unlike neural networks and SVMs, the AdaBoost training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features need not be computed.

AdaBoost is an iterative boosting ensemble algorithm which produces multiple weak learners in iterations and in the end ensembles all of weak learners in to strong classifier. In AdaBoost in each iteration it selects the best feature as a best weak learner for that iteration out of all available features by considering the error rate of selected weak learner being lowest. And considering the error rate of that selected weak learner AdaBoost changes the weights for each sample for the next iteration so that each successive iteration produces better weak learner than its predecessor. In each iteration of AdaBoost the weight of wrongly placed samples (i.e., false positives and false negatives) are increased and weights of rightly places samples (i.e., true positives and true negatives) are decreased so it gives more weights to errors so that errors can be rectified in next iteration while classifying.

In our implementation we have defined “ada_boost(itrs, feature_data, length_positive, length_negative, training_labels, weight_matrix)” which takes these arguments: itrs (number of weak-learners expected in return), feature_data(feature values of each sample), length_positive(number of face samples), length_negative(number of non-face samples), weight_matrix(initial weight for each sample). And returns batch of weak learners and its alpha values with the last updated weight matrix for the batch. how much weak-learners will be there in batch is defined by ‘itrs’. we initialize weights in cascade blocks.

Our implementation:

For number of iterations obtained from cascade we run the for loop.First, step is to normalize weights for each samples. Then we build a numpy array matrix of dimensions n*m where n = number of input images and m = total number of features Multiply above matrix by weight matrix to obtain 'feature_vs_img_matrix'. This feature matrix is formed in such a way that first p rows will belong to positive image domain and rest of the rows will belong to negative domain where p is number of positive examples. After that we compute threshold by using ROC curve with help of sklearn.ROC library where we input true labels and corresponding weights in order to obtain optimum threshold Then apply the threshold to the 'feature_vs_img_matrix' to obtain error matrix such that if the feature value is >= threshold then it is 1 else 0. We add weights of all misclassified features in and store it in 'total_error_count' Later we select the best feature who has the least error count among total_error_count list. Once the best feature is selected, we update the weight matrix with below formula mentioned in viola-jones algorithm.

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$

We also calculate alpha

$$\alpha_t = \log \frac{1}{\beta_t}$$

return indices of best features and alpha to the calling function

Cascading Classifiers :

When we test our model on an image and to detect faces in image we have to compute prediction for each sub-window possible in test image of each size. And on average only 0.01% of all sub-windows are positive(faces). And model spends equal computation time on all sub-window but with cascading we can create such a model that most of the non-face sub-windows are discarded in initial stages of prediction computation and it will invest more computation on more promising sub-windows.

Cascading creates a sub-group of weak learners being selected by the AdaBoost algorithm in each iteration. And such group of weak learners are called as cascades and each cascade has its own TruePrediction rate which determines that only sub-windows with that much or higher TruePrediction rate can pass through that cascade. And what should be the TruePrediction rate for each cascade is determined on the basis of the position in the queue that cascade is computed on test image. Initial cascades have lower true positive rate but as sub-window passes through each cascade the filter criteria increases and in the last cascade it checks for almost 0.9 true prediction rate and only windows with face in it will get through this cascade and if any window that passes through all the cascades is determined as face. And windows that fails to pass criteria for any cascade in the queue is discarded.

Cascading creates a sub-group of weak learners being selected by the AdaBoost algorithm in each iteration. And such group of weak learners are called as cascades and each cascade has its own TruePrediction rate which

determines that only sub-windows with that much or higher TruePrediction rate can pass through that cascade. And what should be the TruePrediction rate for each cascade is determined on the basis of the position in the queue that cascade is computed on test image. Initial cascades have lower true positive rate but as sub-window passes through each cascade the filter criteria increases and in the last cascade it checks for almost 0.9 true prediction rate and only windows with face in it will get through this cascade and if any window that passes through all the cascades is determined as face. And windows that fails to pass criteria for any cascade in the queue is discarded.

In our implementation as we have so many features to compute, To improve the performance we have made few changes in an actual implementation of cascading algorithm. In actual cascading algorithm to create a cascade, in each iteration it calculates true prediction rate by weak learner with validation set. But to improve performance instead of validating true prediction rate for each weak learner before adding it to cascade, we are computing a batch of weak learners and then validate true prediction rate for that batch and after adding that batch to cascade if true prediction rate is met with the allowed true prediction rate then we will go ahead and create another cascade otherwise we will generate another batch of weak learners. This will reduce the running time of validation of around length of batch.

In this project we have defined “build_cascades(test_data, list_of_test_data_features, list_of_all_features, length_positive, length_negative, training_labels)” method which takes these arguments : test_data (validation set), list_of_test_data_features(feature matrix of validation set), list_of_all_features(feature matrix of training set), length_positive(number of face samples), length_negative(number of non-face samples), training_labels(classification labels for training dataset) and returns cascade groups.

Our implementation:

We have defined 4 cascade blocks each having different levels of accuracy.First, we Initialize weight matrix such that for all positive images weight will be 1/(2*no.of positive examples) and negative images will have weight of 1/(2*no. of negative examples).Then we build first cascade block by calling adaBoost till we obtain desired accuracy (in our case we have build 4 cascade blocks with accuracies 50%,60%,70%,80%). store all best features returned by adaboost in respective cascade block. store corresponding alpha values in corresponding alpha list. from each image from test image data(which are face images), extract all features that are chosen by adaboost and multiply it with alpha value. If it is greater than sum of all alpha * 0.5 then it is identified as face. identify all faces from the test input and then calculate accuracy. Repeat the process until criteria for the accuracy is fulfilled. finally, form a chain of cascades by adding all cascade blocks in list_of_cascades also store corresponding alpha value list in list_of_alpha.

After the implementation of cascades when we have trained our algorithm and we have out cascades we are writing this cascade group in to model.json file to reuse this model every time we want to test any image to detect faces.

Testing Model :

To detect faces in any image we need to test that particular image on the model that we have trained. To test image we have implemented sliding window approach. In which we have taken 24x24 as a base window size and slide it over the test image and we compute prediction for each window.

Sliding window[7]:

In Sliding window we have started with the 24x24 as a base window size and slide this window over the image and compute the prediction model and after we slid through with one window size we have increased window size by multiplying the previous window size to 1.5 and again slide through and we increase window size till we have maximum window size possible for window.

For each window, first we resize the window to 24x24 and then compute the feature matrix and then compare the only features that are there listed in the cascade groups. If any window passes through all the cascade groups then we define that widow as face and save its dimensions to result.json and do on to the next window. If at any point in computing cascade predictions if window fails to meet the criteria for any cascade then we discard that window and go on to next window which computing further cascades.

Conclusion:

We have implemented all the implementation of viola jones algorithm from integral images to creating cascades in model. We have calculated integral images and from those integral images of all samples we are computing 1,62K+ features and then with help of cascade and AdaBoost we are creating 4 cascade groups with increasing accuracy requirements. To improve performance time we have made few changes in the minor steps as, we implemented multi-threading in computing features as we will compute 1,62,336 features for around 15000 training samples so it gives better performance with multithreading and then we merge the all features in to single matrix at the end, we also made small change in implementation of cascades algorithm, as while creating cascades it requires you to compute accuracy for each weak learner with validation set and then add it to the cascade, what we are doing is instead of computing accuracy for each weak learner we are creating small batches of weak classifier and then computing its accuracy and add it to cascade group accordingly, which helps decreasing the running time a lot and we can validate each weak learner with more validation data. Due to computing resource limitation we were unable to test the implementation with higher number of training samples which ideally should be around 15000. However, we simulated end-to-end flow with having 200 training samples consisting of 150 face data and 50 non-face data and 50 samples as a validation set. We have added output of this simulation as sample-model.json in model_files directory.

References:

[1] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR’01), 2001.

[2] V. Jain and E. L. Miller, “Fddb: a benchmark for face detection in unconstrained settings,” 2010.

[3] https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework

[4] http://dags.stanford.edu/projects/scenedataset.html

[5] https://en.wikipedia.org/wiki/Haar-like_feature

[6] https://en.wikipedia.org/wiki/AdaBoost

[7] https://en.wikipedia.org/wiki/AdaBoost

[8]https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

[Note: Both teammates have equal contribution in the implementation as well as report.]