SMAI Project Report

Topic - Rapid Object Detection using a Boosted Cascade of Simple Features

Team Members : -

- Abhijnan Vegi (2020101004)
- Kushagra Kharbanda (2020101002)
- Shavak Kansal (2020101023)
- Pallav Koppisetti (2020102070)

Introduction

This paper describes a machine learning approach for visual object detection that can process images quickly and achieve high detection rates. Other face detection systems , auxiliary information (image difference or pixel color) is used while this system only uses information extracted from a single grayscale image. This technique was popularly used before CNNs became mainstream. A CNN is a single classifier which looks at a full image and applies matrix operations to arrive at a classification, Viola-Jones takes an ensemble approach. It uses cascades of weak classifiers where each individual classifier is weaker than the successive classifier cascade as the combined results produce a strong classifier.

Three significant contributions distinguish this work from other works.

1. The first is the introduction of a new picture representation known as the **"Integral Image,"** which allows our detector's features to be computed very quickly. A few operations per pixel can be used to compute the integral image from an image. Once computed, any of these **Harr-like features** can be computed in constant time at any location.
2. The second contribution of this research is a strategy for building a classifier using **AdaBoost** by selecting a minimal number of significant features. The total number of Harr-like features within any image sub-window is extraordinarily

huge, substantially greater than the number of pixels. To enable rapid categorization, the learning process must ignore the vast majority of available data and concentrate on a small collection of key features.

3. The paper's third key addition is a way for merging successively more complicated classifiers in a **cascade** structure, which dramatically boosts the detector's speed by focusing attention on promising portions of the image. The idea behind focus of attention techniques is that it is typically easy to quickly determine where an object would appear in a picture. Only these potential places will be subjected to more advanced processing, thus saving a lot of time and computational power.

All these three ideas combined help in reducing the overall time complexity by a drastic amount. This enables the algorithm to be used in real-time applications where the detector comfortably runs at 15 frames per second.

This algorithm, with a very long training time, is restricted to binary classification tasks such as object detection due to it’s nature .

Algorithm

1. Haar-like Features and Integral Image

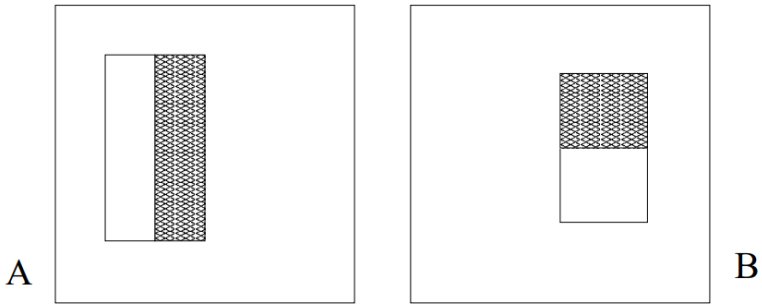
We use Haar-like filters on the image to compute the features. Now, the motivation behind using these features rather than using pixels directly is that features can act to encode ad-hoc domain knowledge that is difficult to learn using a finite quantity of training data. Also, using these features helps us boost the speed of our algorithm as we don’t have to work at the pixel level anymore.

The features are shaded black and white. These black and white regions help us in assigning a positive or a negative sign to a particular feature in the scalar product. So, we are essentially taking a scalar product between our image sub-window and a Haar-like template and deciding the sign by checking the region they lie in. So, we can say that the feature associated with a particular Haar-like filter P and an image sub-window I can be given as follows

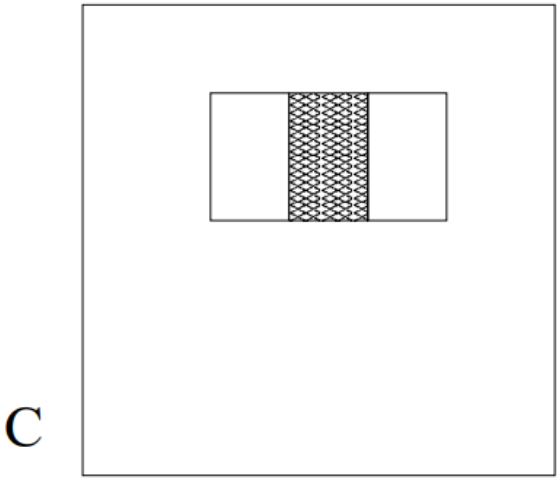
$$\sum_{1\leq i\leq N}\sum_{1\leq j\leq N} I(i,j)1_{P(i,j)\ is\ white} - \sum_{1\leq i\leq N}\sum_{1\leq j\leq N} I(i,j)1_{P(i,j)\ is\ black}$$

We use three different kinds of features in our model.

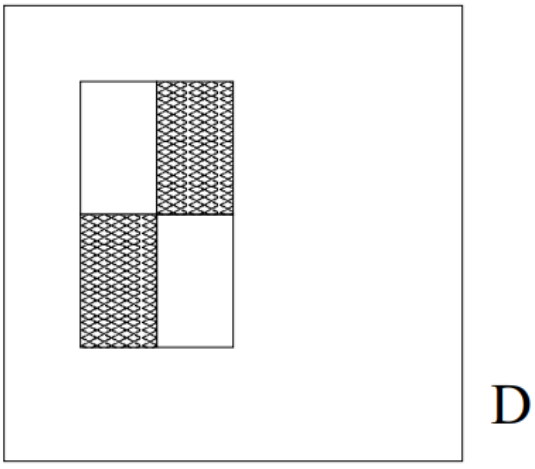
1. Two-rectangle features



2. Three rectangle features



3. Four rectangle features

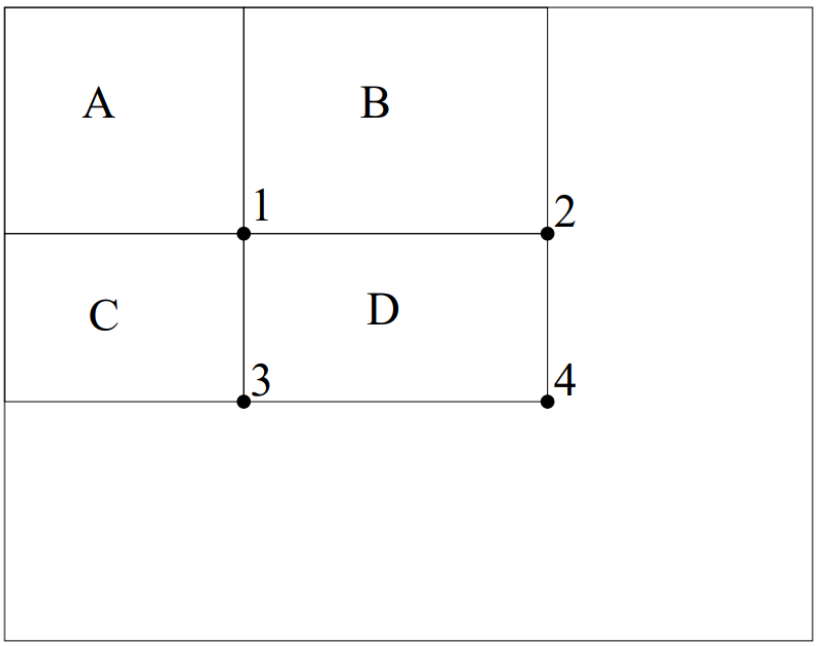


In the detection phase of the VJ-object detection framework, a window of the target size is moved over the input image. For each subsection of the image, the Haar-like feature is calculated. **T**his difference is then compared to a learned threshold that separates non-objects from objects. Because such a Haar-like feature is only a weak learner or classifier (its detection quality is slightly better than random guessing), many Haar-like features are necessary to describe an object with sufficient accuracy.

It is easy to see that even for small images, there are a lot of features (over 160,000 for a 24 x 24 image) to be computed. Since the algorithm requires iterating over all features, they must be computed very efficiently. In order to do this, Viola and Jones introduced the integral image. The integral image is defined by the following relationship

$$\mathrm{II}(i,j) = \begin{cases} \sum_{1 \le s \le i} \sum_{1 \le t \le j} I(s,t), & 1 \le i \le N \; and \; 1 \le j \le N \\ 0, & otherwise \end{cases}$$

What this relationship says is that the integral image at a point (x, y) is the sum of all pixels above and left of the current pixel. Now, if we know the integral image at each point, we can calculate the features in constant time. For example, if we have the following rectangular regions



Then, the sum of the pixels in region D is ii(4) + ii(1) — ii(2) — ii(3), which is just four array references. This saves a lot of time in feature extraction.

2. Best feature selection using Adaboost

Even though we have found a way to calculate each feature efficiently, we still have to take care of the fact that we have a large number of features to deal with. What we want to do is that we want to find out a small number of features that can be combined to form an effective classifier. Now, the main challenge that we face is to find these particular features.

This method is based on a modified version of the Adaboost algorithm on Haar like features to find a cascade of classifiers (where a strong classifier is constructed out of many weak ones). So, we design a weak learning algorithm which selects the best weak classifier. We determine the best weak classifier by checking which one of the classifiers best separates the positive and negative examples.

So, for each feature, an optimum threshold is determined such that the minimum number of samples are misclassified. A weak classifier $h_j(x)$ thus consists of a feature f_j, a threshold θ_j and a parity p_j indicating direction of the inequality sign

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & otherwise \end{cases}$$

Here, x is a sub window of an image

Adaboost Algorithm

- Example images :- $(X_1, y), (X_2, y), ...(X_n, y)$ where X represents the image and $y_i = 0, 1$ for no-face (negative) and face (positive) images respectively.
- First, we initialize the weights $w_i = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, ..., T$; where T is the number of classifiers
- Normalize the weights, making it a probability distribution

»

- Now for each feature j, train a single feature classifier h_j , whose error is evaluated w.r.t $w_t, \epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
- Choose the weak classifier with lowest error ϵ . Update the weights

»

$$where\ \mathrm{e}_i = \begin{cases} 0, & \text{if correctly classified} \\ 1, & otherwise \end{cases}$$

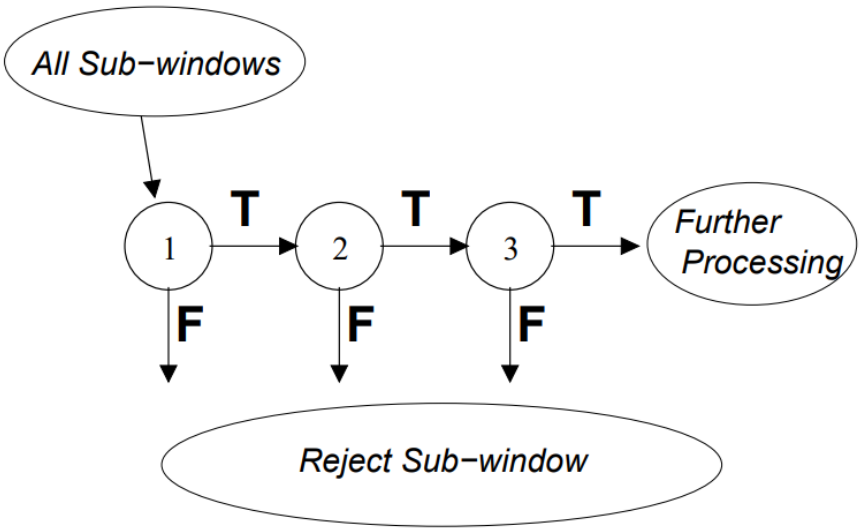
- The final strong classifier formed is :

»

3. The Attentional Cascade

In order to detect multiple faces in any image, a face-detection system typically slides a fixed size sub-window over each location in the image. For a large image, and given that the image will have to be scaled and rescaled to account for different sized faces, this amounts to a lot of Viola-Jones. If the system must detect faces in real time, the computation must be quick. Viola and Jones developed the "Attentional Cascade" for this purpose.

To classify an image, the attentional cascade employs a series of Viola-Jones classifiers, each progressively more complex. The nth classifier only considers an image if the n-1th classifier classifies it as a positive example. If a classifier determines that the image is not a positive example, then the cascade is terminated at that layer.



A cascade, for example could include classifiers that consider 1 feature, 5 features, 10 features, 50 features etc in that order in subsequent steps/layers of the cascade. The advantage of introducing this structure early on is that negative examples are weeded out early without consuming a lot of computational time. And because only a

fraction of the features are computed for negative examples, this significantly reduces computation time for finding the negative examples in a large image.

Following classifiers are trained using examples that have passed through all of the previous stages, much like a decision tree. As a result, the second classifier is challenged more than the first. Examples that pass the first stage are "harder" than typical examples. Deeper classifiers' receiver operating characteristic (ROC) curve is pushed downward by more difficult examples. Deeper classifiers have a higher false positive rate at a given detection rate.

The only difference we have in training using this method is that subsequent classifiers are trained on only the positive examples(which includes true positives and false positives). For, example, the first classifier is trained on all training examples, but the second one is trained only on all positives samples and the false positive samples. Hence, each subsequent classifiers ends up focusing on progressively harder features. The final outcome of this is that the false positive rate is reduced dramatically.

In practice, a very simple framework is used to train and generate and effective classifier. Each stage of the cascade reduces the rate of false positives while increasing the rate of detection. A target is chosen with the goal of reducing false positives while increasing detection. Each stage of the cascade is trained by adding features until the target detection and false positive rates are met. Stages are added until the overall false positive rate and detection rate target is met.

Analysis

We trained a model and stored it in a pickle file named `model.pkl`

The purpose of this project was to implement the ViolaJones face detection algorithm and obtain reasonable performance. The results we tested meet our expectations and the experimental solution proposed effectively improves the performance of the final implementation. In terms of the
processing time of our final detector, surprisingly the time bottle neck was not at the AdaBoost learning phase but the Haar-like rectangular feature extraction phase.

Our model which had the layer structure - `[5,10,15,20,25]` took 3 hours to train. This fact indicates that firstly the attentional cascade applied in ViolaJones algorithm indeed effectively speeds up the training process. Secondly Haar-like feature space is actually pretty
huge and time consuming to compute. It is also likely that our code for feature extraction can be re-factored in a more time-efficient manner.

Besides, there are many other aspects that can be improved. First of all, data preparation seems to be a major source of performance limitation. Given more image data sets with more ideal image size (around 24 by 24 pixels), we are able to train a more solid classifier. With sub-window larger than 19 by 19 pixels a lot more feature values can be extracted for both Cascade and SVM classifier and thus more details of the face can be well captured.

The accuracy that we got on testing our model on the cmu dataset is given below

```
Testing Accuracy:  0.928709055876686
Testing Precision:  1.0
Testing Recall:  0.8574181117533719
Confusion matrix: tp : 445, tn : 519, fp : 0, fn : 74
```

The false negatives that we get are majorly from the first layer, since it has weak classifiers, it has more number of false negatives. But, the false positives are so low as they are detected by the deeper layers, which are stronger classifiers compared to their predecessors. Hence, the number of false negatives is so high.

Hyperparameter tuning

We tried training with different values of number of features for different layers. We observed that by keeping less number of features in the first layer, the testing accuracy was coming out to be pretty low. But as we increased the number of features in the first layer the training time shot up dramatically. So, we had to settle for a sweet spot and have the following layer structure - `[5,10,15,20,25]` . With this layer structure it took us 3 hours to train and we got an accuracy of about 93%.

Drawbacks of our Algorithm

1. High number of false negatives.
2. It is very sensitive to rotation, orientation, lighting and other such factors.
3. Training time is very high.

Work Distribution

The work was done equally by all members. As there were no clear division of the work, people were working together on the same piece of code. We collaborated using tools such as Liveshare. Many a times, we all used to sit together and debug

the code, so we can’t provide a clear demarcation of how much work was done by whom, but we can assure that everyone put in equal efforts