**Data Pre-processing**

Data pre-processing is a step of cleaning, transforming and aggregating data before it can work with algorithms. In this project we need to pre-process live video frames before they can be fed to a model using OpenCV we can detect and extract face(s) from frames. OpenCV provides three different algorithms for detecting faces in a stream of images viz., Cascade Classifier and MTCNN (Multitask Convolution Neural Network) using Haar Basis functions with AdaBoost as its core component first created by Viola-Jones and HOG (Histogram of Oriented Gradients) descriptor and object detector by Navneet Dalal and Bill Triggs. Cascade Classifier can process 25 images/second with precision of 95.24% and recall of 82.60%, MTCNN can process 3 images/second with precision of 98.02% and recall of 89.85% this reading is estimated using CPU it can be increased using better CPU or GPU. MTCNN can deal with scale and orientation of the face where Cascade Classifier cannot. For our purpose we will be using MTCNN for training face recognition algorithms and Cascade Classifier for real time face detection (MTCNN can also be used for real time face detection if GPU is in use).

**Cascade Classifier**

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar like features (based on Haar Wavelet used for compression of wave further notion will like from Haar like features) shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



**Fig x.1**

Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image (cumulative sum of pixels located at left top of corresponding pixel). However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.



**Fig x.2**

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found. There is a common trade-off between detection rate and precision for this classifier).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. The authors have a good solution for that.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot (further cascade classifiers don’t need to process that part again rather concentrating on more complex features), and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

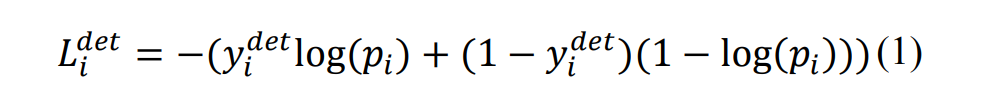
For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is that plan!

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window.

**MTCNN**

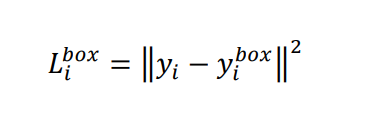
MTCNN is a multitask neural network model for face detection. In order to take into account the performance and accuracy, and avoid the huge performance consumption caused by traditional ideas such as sliding window and classifier, it first uses small model to generate target region candidate box with certain possibility, and then uses more complex model for fine classification and higher precision region box regression, and makes this step recursive to form a three-layer network, namely p-net , RNet, o-net, to achieve fast and efficient face detection. In the input layer, image pyramid is used to transform the scale of the initial image, and p-net is used to generate a large number of candidate target area frames. After that, R-Net is used for the first selection and border regression of these target area frames, and most of the negative examples are excluded. Then, the more complex and higher precision network o-net is used to discriminate and regress the remaining target area frames.

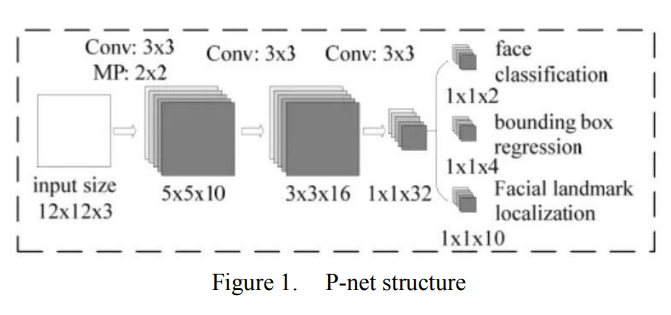
For face classification, MTCNN sets the learning objective as a binary classification problem, and uses cross entropy loss function for each sample xi



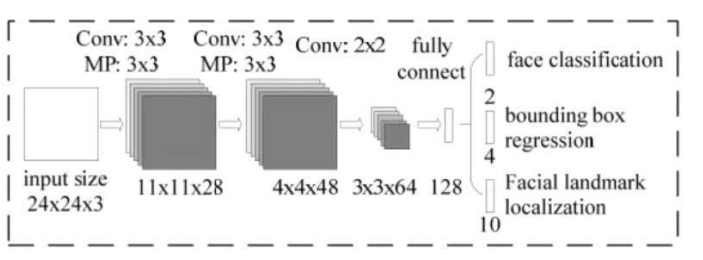
Pi is the sample xi predicted by neural network Probability of belonging to face the label of {1, 0}. 0 is the real image indicates that there is no face in the image, and 1 indicates that there is a face in the image.

For the candidate frame regression task, the bounding box regression algorithm is used to make the target frame predicted by the network model close to or coincide with the real target frame. MTCNN algorithm uses a 4dimensions vector (x, y, w, h) to output the predicted window, which represents the upper left coordinate of the prediction window and the width and height of the window respectively. For each human face candidate frame, the difference square loss function is used to predict the deviation between the candidate frame of network output and the nearest ground truth:

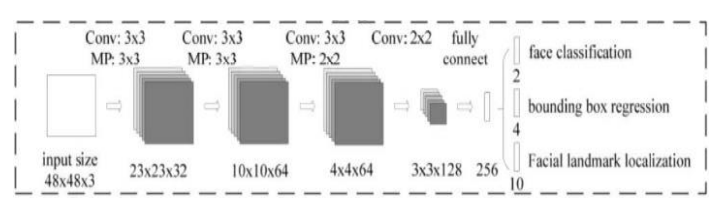




**P-net structure**



**R-net Structure**



**O-net Structure**