Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

PROGRAMMING INTERGRATION PROJECT

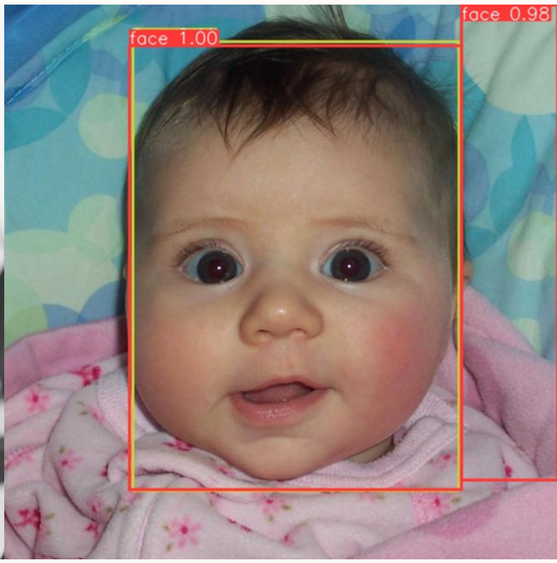
FACE DETECTION

Lecturer: Ph.D Tran Tuan Anh
Students: Le Hoang Duy – 2152040
Vo Thien Nam – 2111817
Le Duc Hoang Nam – 2111795

OVERVIEW

ABSTRACT

Human face detection is a critical component of facial recognition systems, simplifying the overall recognition problem by classifying input objects as either human faces or not.



CASCADE CLASSIFIER

HAAR-LIKE - ADABOOST

It is a method invented by Paul Viola and Michael J.Jones , that is about identifying human faces based on the feature-based approach, means that the facial recognition system will learn facial patterns from a set of sample images

COMPONENTS

Haar-like features: features are placed into image areas

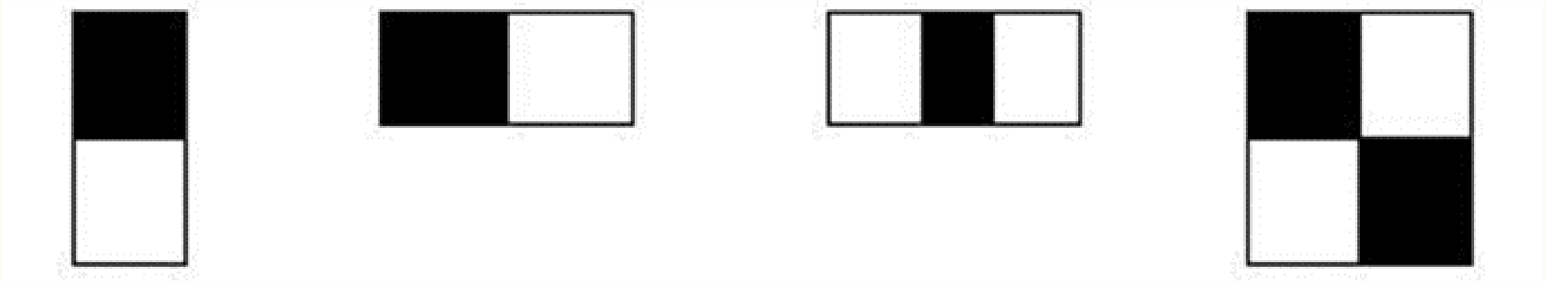
Integral Image: actually this is a tool that helps calculate Haar-like feature values faster

Adaboost: classifier (filter) operates based on the principle of combining weak classifiers to create a strong classifier

Cascade of Classifiers: cascade classifier with each level being an Adaboost classifier, which increases classification speed

HAAR-LIKE FEATURES

On an image, the face area is a collection of pixels that have different relationships compared to other image areas, these relationships create unique characteristics of the face. All human faces share the same characteristics after being converted to grayscale images



HAAR-LIKE FEATURES

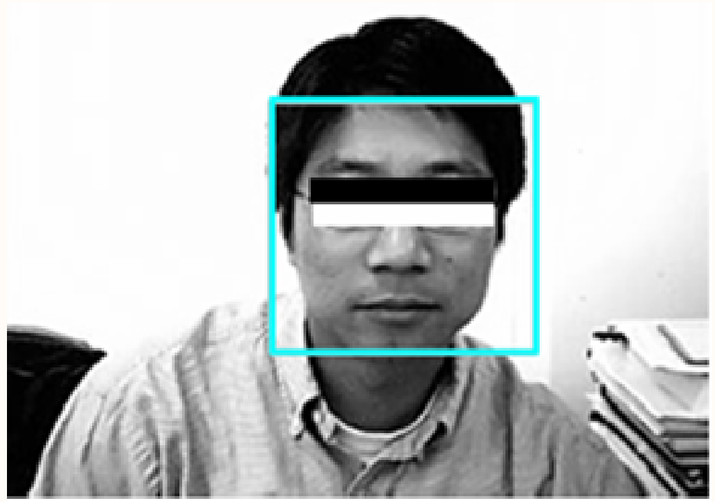
The value of a Haar-like feature is the difference between the sum of the gray values of the pixels in the “black” region and the sum of the gray values of the pixels in the “white” region

$$f(x) = Sum_{black\ rectangle}(pixel\ gray\ level) - Sum_{white\ rectangle}(pixel\ gray\ level) \quad (1)$$

So when placed on an image area, the Haar-like feature will calculate and give the characteristic value f(x) of that image area.

HAAR-LIKE FEATURES

To detect faces, the system will create a sub-window of fixed size that scans the entire input image. Thus, there will be many sub-images corresponding to each sub-window, Haar-like features will be placed on these sub-windows to calculate the value of the feature. These values are then used by the classifier to confirm whether the frame is a face or not.



HAAR-LIKE FEATURES

For each of the above features, a weak classifier $h_k(x)$ is defined as follows:

$$h_k(x) = \begin{cases} 1 \; if \; p_k f_k(x) < p_k \theta_k \\ 0 \; otherwise \end{cases} \qquad (2)$$

With:

x: current 24x24 pixel sub-window of image

θ_k: threshold

f_k: Haar-like feature value

p_k: parity indicating the direction of the inequality sign

We understand the above formula simply as follows: when the value of the Haar-like feature k: f_k at sub-window x exceeds a threshold θ_k, the classifier $h_k(x)$ will conclude that sub-window x is a face. ($h_k(x)=1$), and if f_k does not exceed that threshold, it is not a face.

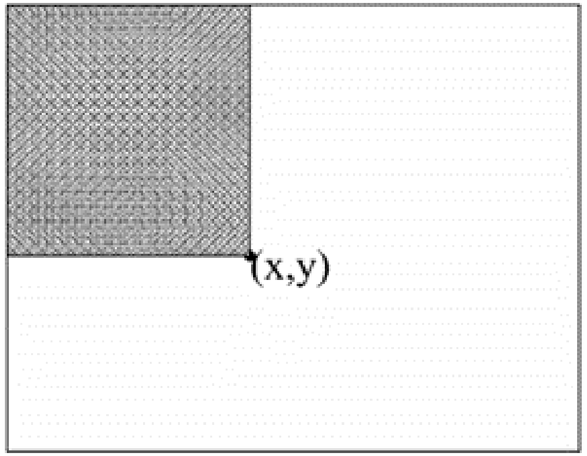
PROBLEM WITH HAAR-LIKE FEATURES

Determining whether a sub-window is a face or not requires the use of many Haar-like features. For each size, feature type and position in the sub-window, we get a feature corresponding to a weak classifier x. So the complete set of features in a sub-window is very large. This is not suitable for real-time response because the processing time is very long

INTEGRAL IMAGE

As stated above, the number of Haar-like features is very large and the amount of calculating the values of these features is very large. Therefore, integral images are introduced to quickly calculate features and reduce processing time.

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y')$$



INTEGRAL IMAGE

After converting the image into an integral image, calculating the value of Haar-like features will be very simple.

1	1	1
1	1	1
1	1	1

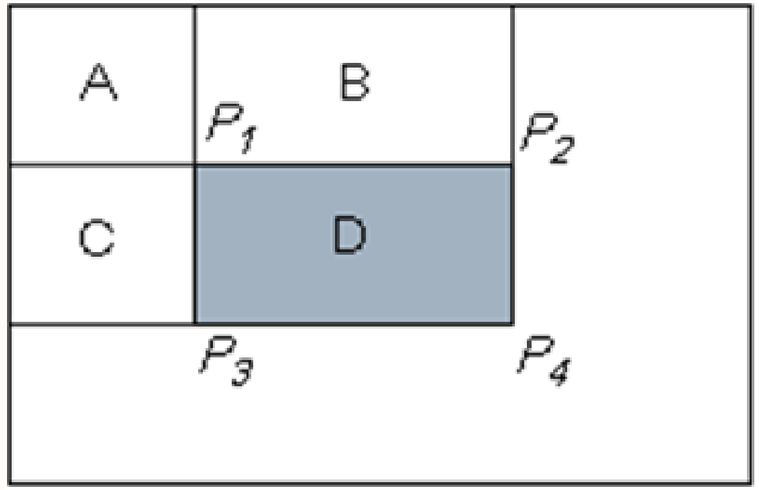
ảnh ban đầu

1	2	3
2	4	6
3	6	9

ảnh tích hợp

APPLY INTEGRAL IMAGE

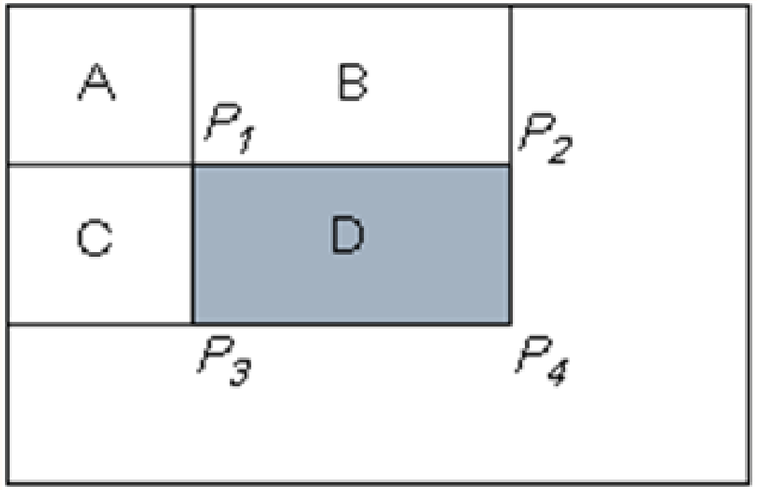
To calculate the Haar-like feature value, we must calculate the total pixel value in a rectangular area on the image. For example, area D in the figure:



APPLY INTEGRAL IMAGE

To calculate the Haar-like feature value, we must calculate the total pixel value in a rectangular area on the image. For example, area D in the figure:

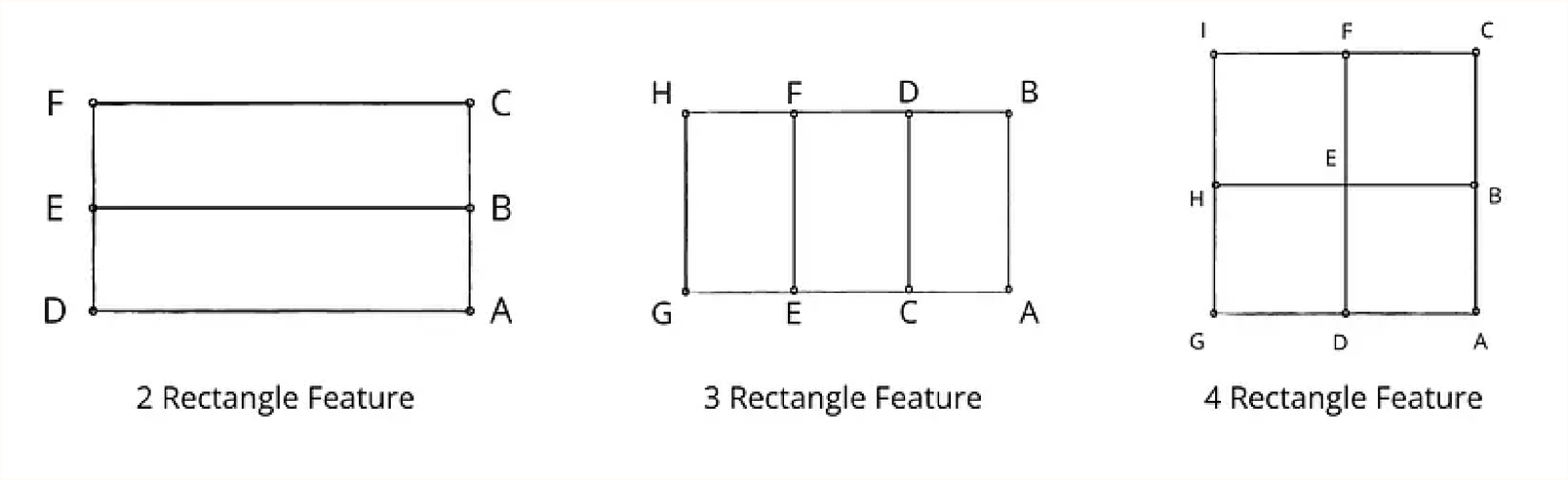
$$D = P1 + P4 - P2 - P3$$



APPLY INTEGRAL IMAGE

Using integral images we can achieve constant time evaluation of Haar features.

1. Edge Features requires only 6 memory lookups
2. Line Features requires only 8 memory lookups.
3. Diagonal Features requires only 9 memory lookups.



BOOSTING AND ADABOOST

Boosting refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor. AdaBoost also known as Adaptive Boosting is one of the most popular Boosting techniques used.

Given example images $(x_1, t_1), \ldots, (x_n, t_n)$ with $t_i \in \{0, 1\}$

1. Initialize weights: $w_n^{(1)} = \frac{1}{N}$ with n = 1, 2, …, N.
2. For m = 1, …, M:

(a) Building a weak classifier h_m:

+ With each feature j, train a classifier h_j which is restricted to using a single feature.

The error is evaluated:

$$E_j = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n) \qquad (1.0)$$

With $I(h_m(x_n) \neq t_n) = 1$ if $h_m(x_n) \neq t_n$ and = 0 otherwise.

+ Then choose the weak-classifier h_j with lowest error that is h_m.

(b) Update the weights:

+ Calculate:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \qquad (1.1)$$

and:

$$\alpha_m = ln \frac{1-\epsilon_m}{\epsilon_m} \qquad (1.2)$$

+ then update:

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(h_m(x_n) \neq t_n)\} \qquad (1.3)$$

3. The final strong classifier is:

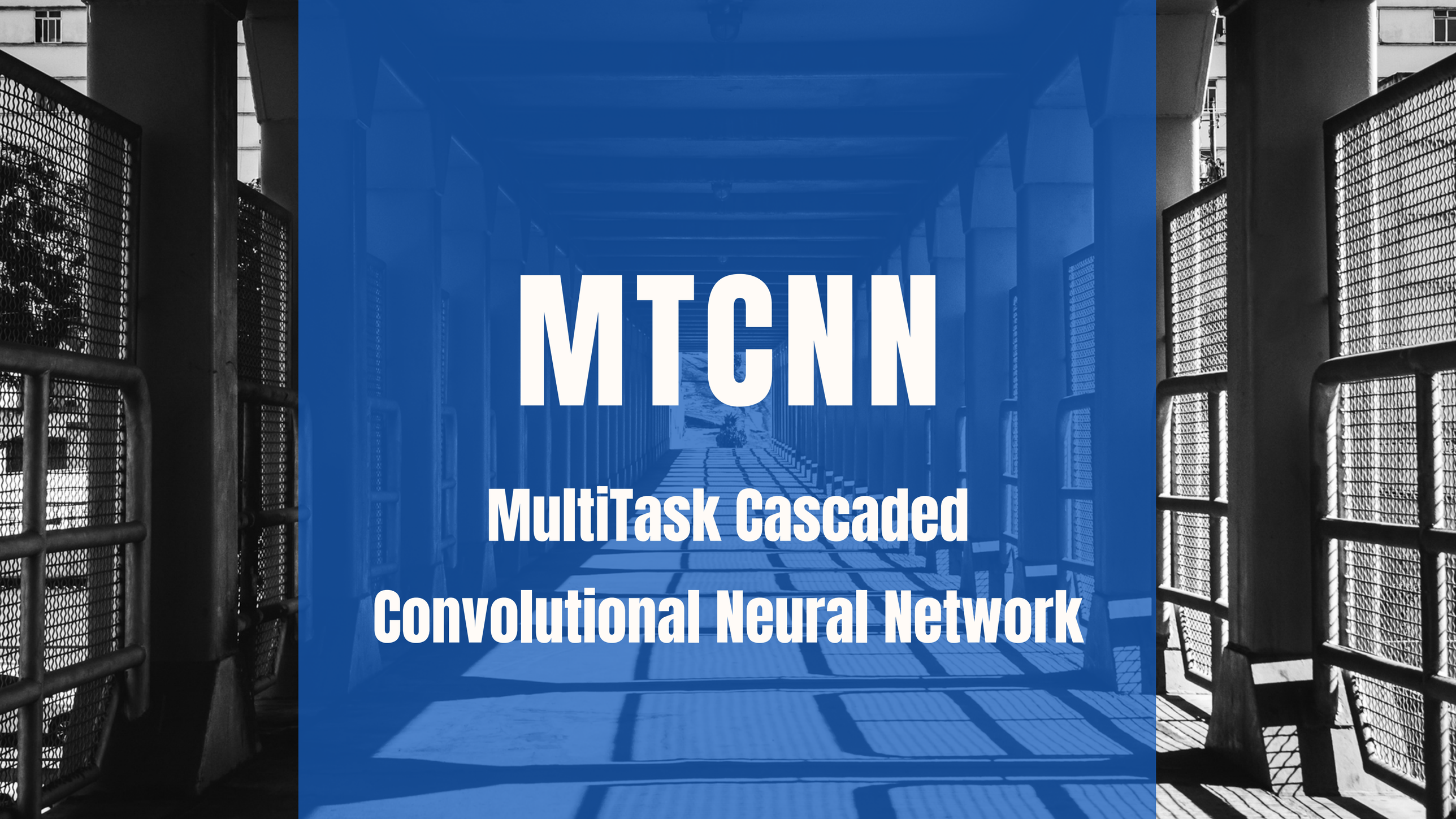
$$H_M(x) = sign[\sum_{m=1}^{M} \alpha_m h_m(x)] \qquad (1.4)$$

CASCADE OF CLASSIFIER

- Strong features are formed into a binary classifier. : Positive matches are sent along to the next feature. Negative matches are rejected and exit computation.
- Reduces the amount of computation time spent on false windows.
- Threshold values might be adjusted to tune accuracy. Lower threshold yield higher detection rated and more false positives.

CASCADE OF CLASSIFIER

In simple terms, each feature acts as a binary classifier in a cascade filter. If an extracted feature from the image is passed through the classifier and it predicts that the image consists of that feature then it is passed on to the next classifier for next feature existence check otherwise it is discarded and next image is checked



MTCNN

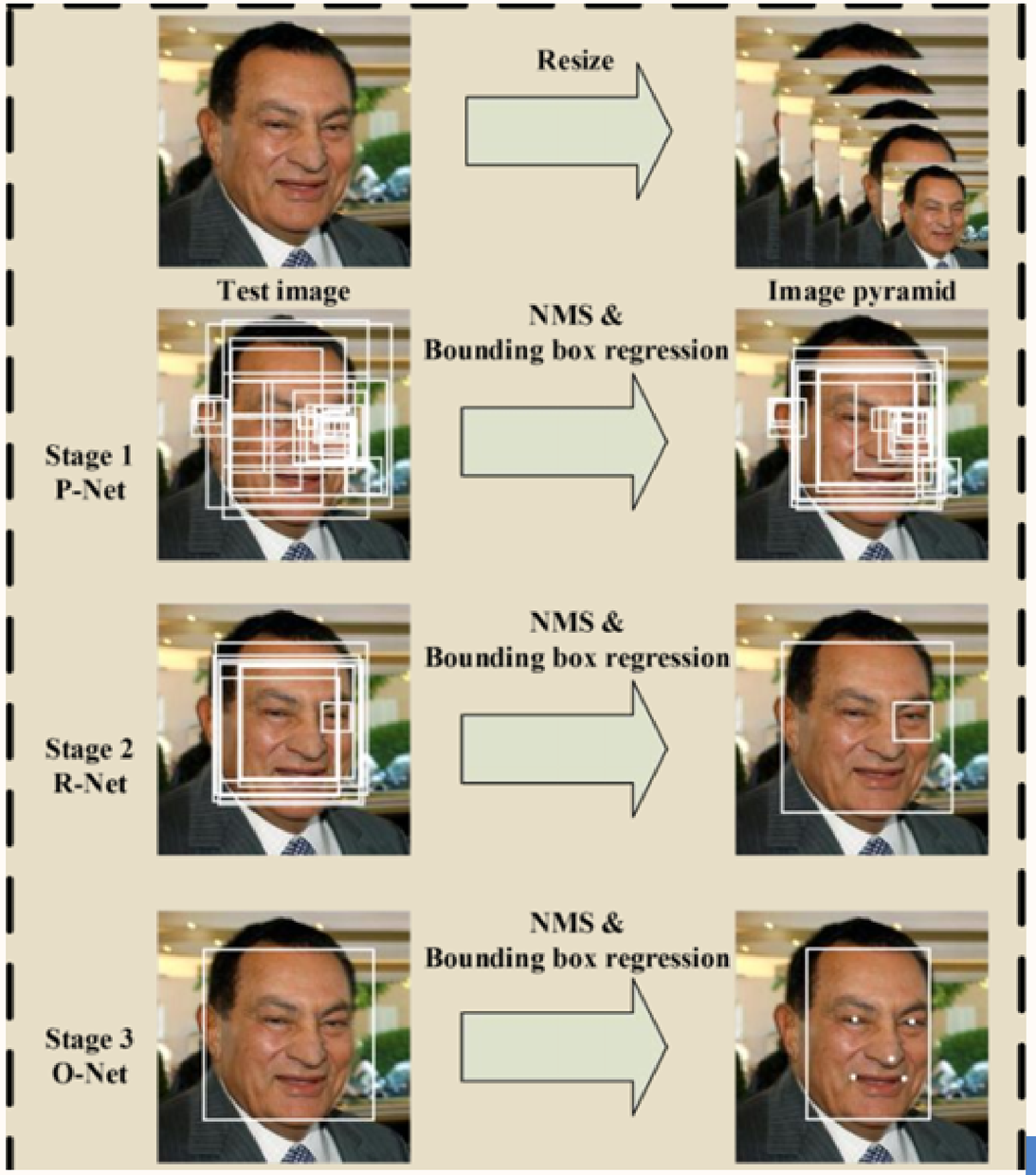
MultiTask Cascaded Convolutional Neural Network

PROBLEM WITH VIOLA-JONES DETECTOR

If the faces are of different sizes, the image is resized and the classifier runs on every image. The accuracy in such cases is not very great. The main disadvantage is the lack of robustness. The Viola-Jones classifier assumes the faces are looking straight at the camera (or almost straight), lighting is uniform and face is well visible

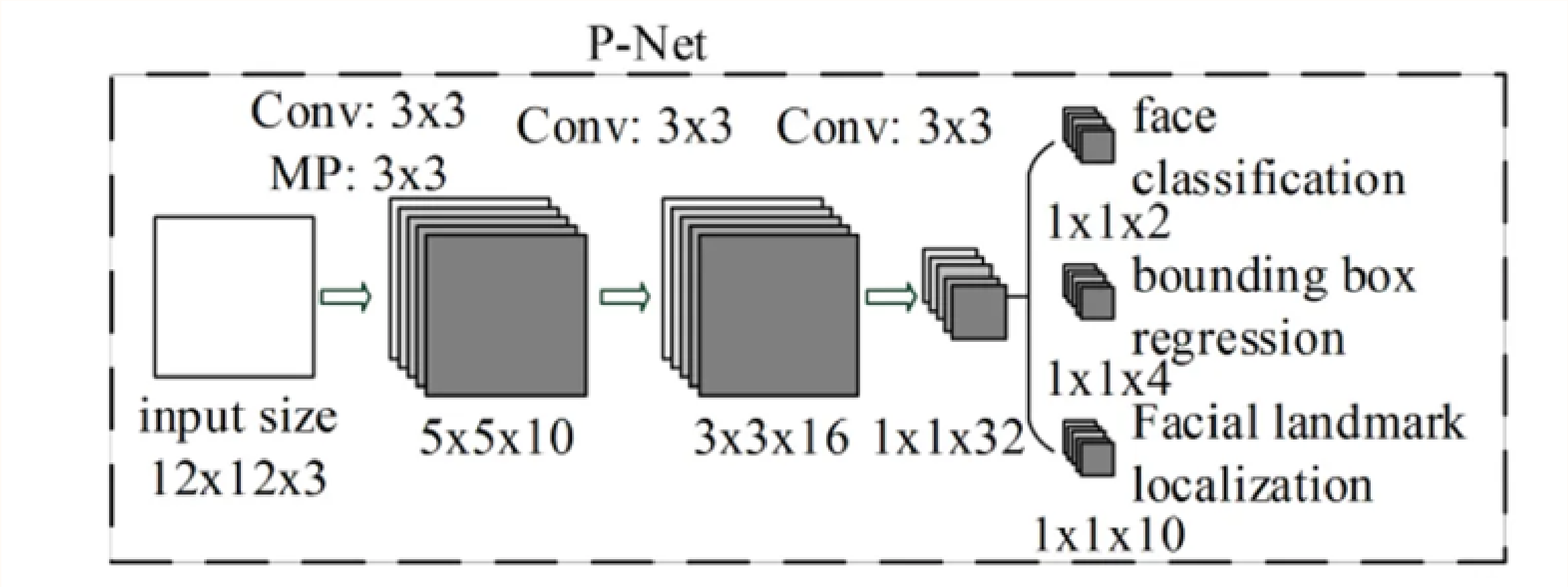
MTCNN

MTCNN is a modern tool for face detection, leveraging a 3-stage neural network detector: the P-Net, the R-Net, and the O-Net.



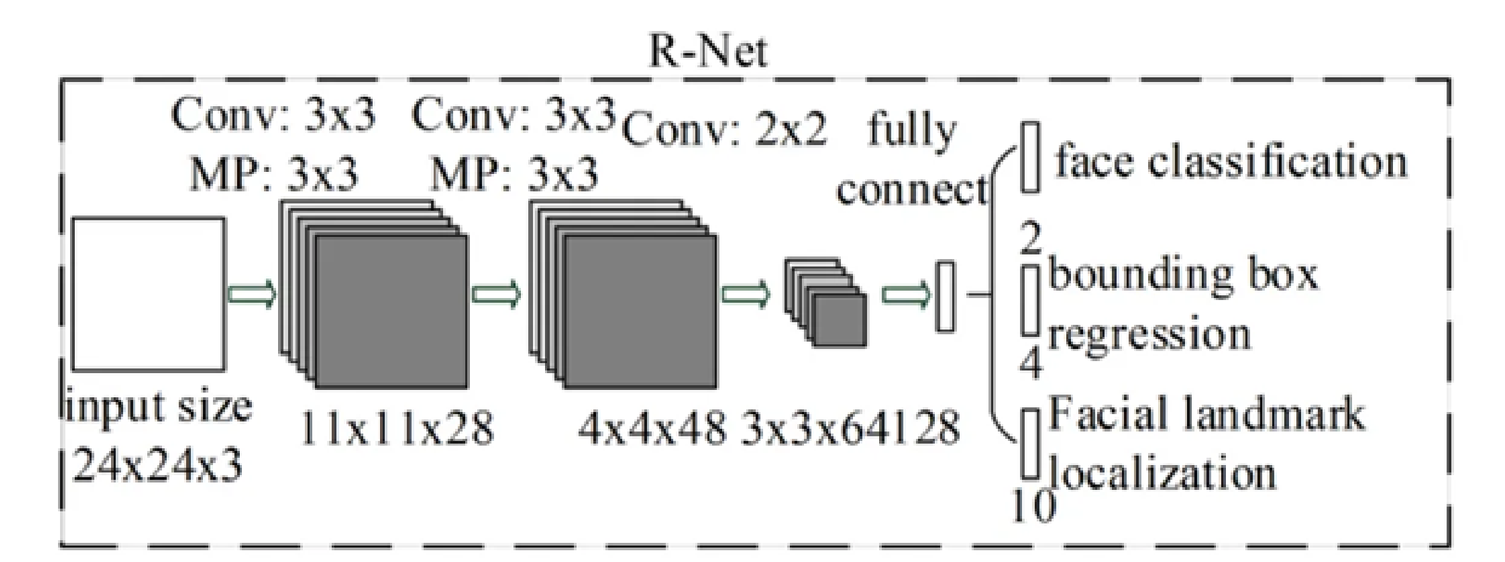
STAGE 1: THE PROPOSAL NETWORK

We exploit a fully convolutional network, to obtain the candidate windows and their bounding box regression vectors. Then we use the estimated bounding box regression vectors to calibrate the candidates. After that, we employ non-maximum suppression (NMS) to merge highly overlapped candidates.



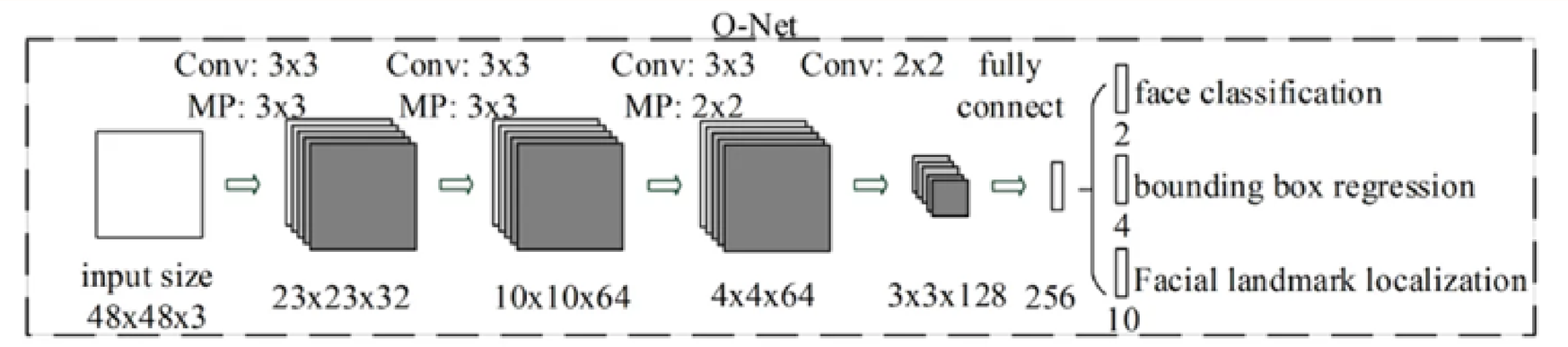
STAGE 2: THE REFINE NETWORK

It takes the P-Net bounding boxes as its inputs, and refines its coordinates. The R-Net further reduces the number of candidates, performs calibration with bounding box regression and employs non-maximum suppression (NMS) to merge overlapping candidates



STAGE 3: THE OUTPUT NETWORK

This stage is similar to the R-Net, but this Output Network aims to describe the face in more detail and output the five facial landmarks' positions for eyes, nose and mouth.




YOLO V8

YOLO V8

YOLOv8 or You Only Live Once v8 is a state-of-art algorithm that was released since May 2023. It’s known as the latest version of YOLO family of algorithms that famous for their speed and accuracy.

THE DIFFERENCE BETWEEN YOLO V8 AND R-CNN

R-CNN

This method first generate potential bounding boxes in an image and run a classifier on this proposed boxes.

After classification, post-processing is used to refine the bounding boxes, which involves of hard and slow pipeline.

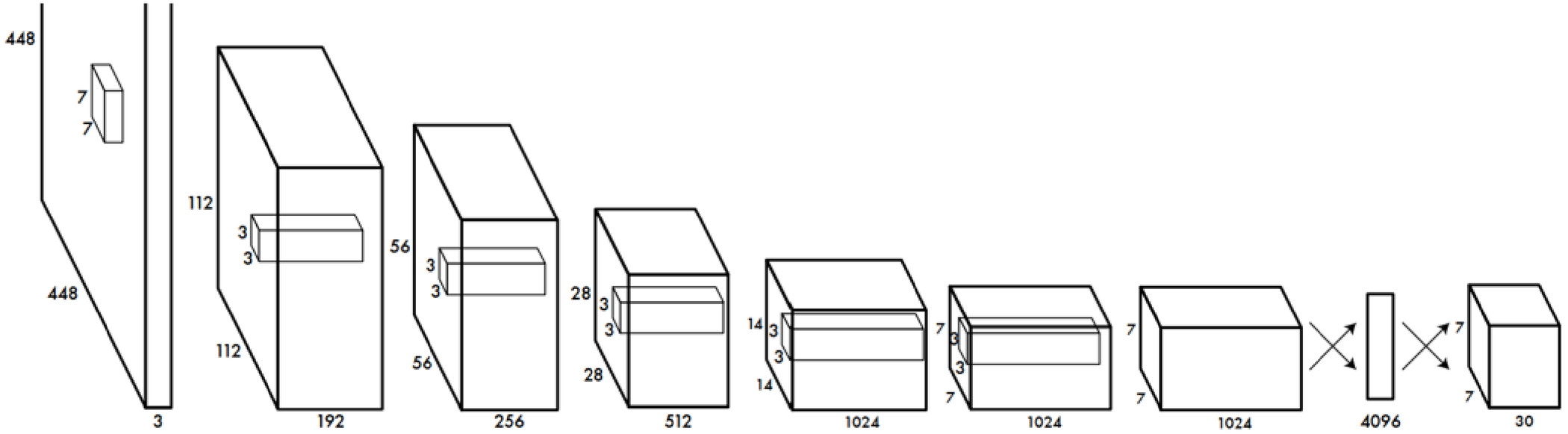
YOLO V8

YOLO present these processes with a simple convolutional neural network to sketch the bounding boxes and **simultaneously** calculate the class probabilities for those boxes as well.

ARCHITECTURE AND WORKS OF YOLO V8

- The system divides the input images into an S x S grid.
- Each grid cell predicts B bounding boxes and confidence scores for these boxes.
- Each bounding box consists of 5 predictions: x,y,w,h and the confidence. The (x,y) coordinates represents the center of the box relative to the bounds of grid cell.

ARCHITECTURE AND WORKS OF YOLO V8



ARCHITECTURE AND WORKS OF YOLO V8

A loss function also in accounted through layers in the system:

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$


HOG

HOG

HOG (Histogram of Oriented Gradients) combined with SVM (Support Vector Machine) is widely used to object detection. The main idea of HOG is to use histograms of gradients directions as a feature descriptor.

1. FIND INTENSITY GRADIENT OF THE IMAGE

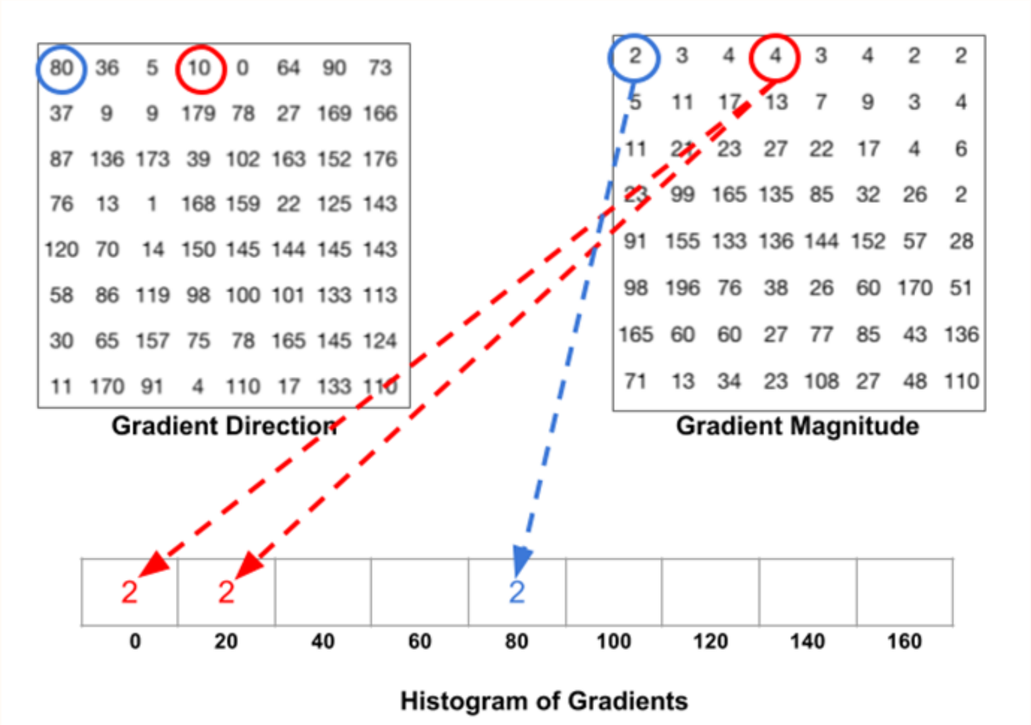
Filter smoothened image with a Sobel filter in both horizontal and vertical direction to get first derivatives Gx and Gy. Then count edge gradient magnitude G and direction θ for each pixel as follows:

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

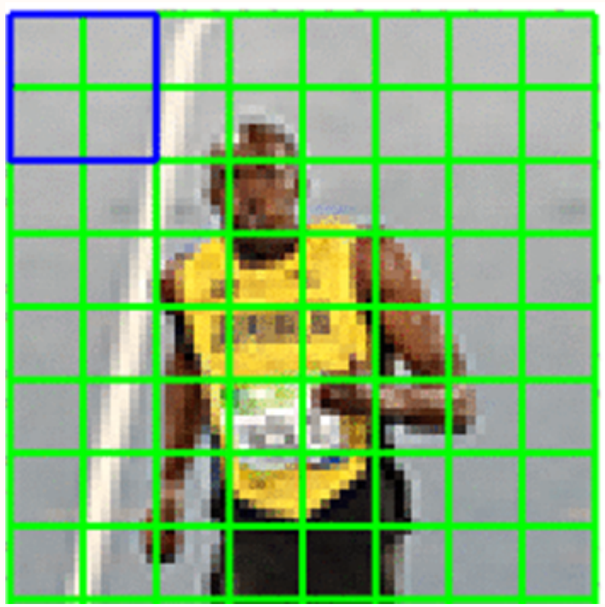
2. CALCULATE HISTOGRAMS OF GRADIENTS IN CHOSEN CELLS

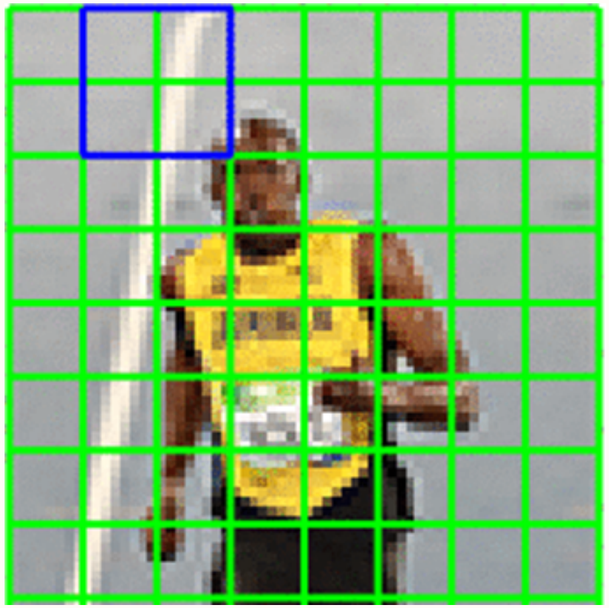
Divide image to equally size cells. In our case 64x64 px images where split to 64 8x8 cells.



3. BLOCK NORMALIZATION

Initialize block size as a multiplicity of cell size. Now for all histograms obtained in previous step, their values are normalized within block. Results are attached to final feature list.





IMPLEMENTATION AND DEMO


THANK
YOU