Face Detection

Arun Ross

Professor

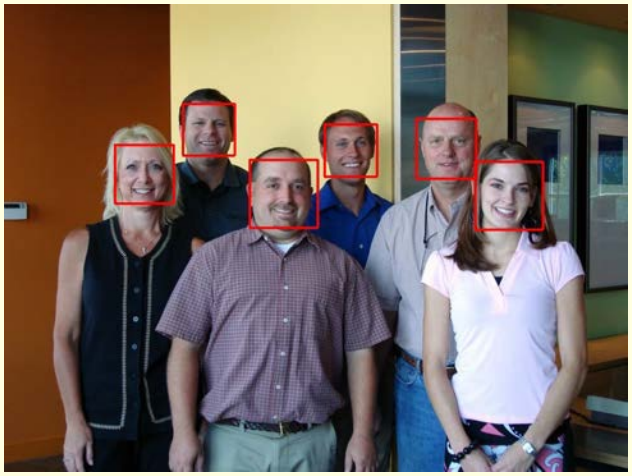
Michigan State University

rossarun@cse.msu.edu

http://www.cse.msu.edu/~rossarun

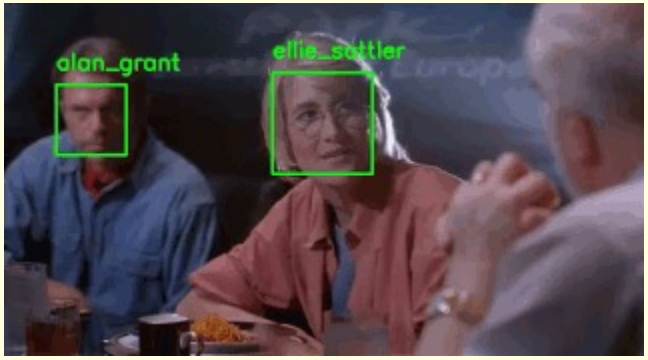
Face Detection

- Face detection has several applications:
  - Image retrieval
  - Law enforcement
  - Biometrics
  - Surveillance



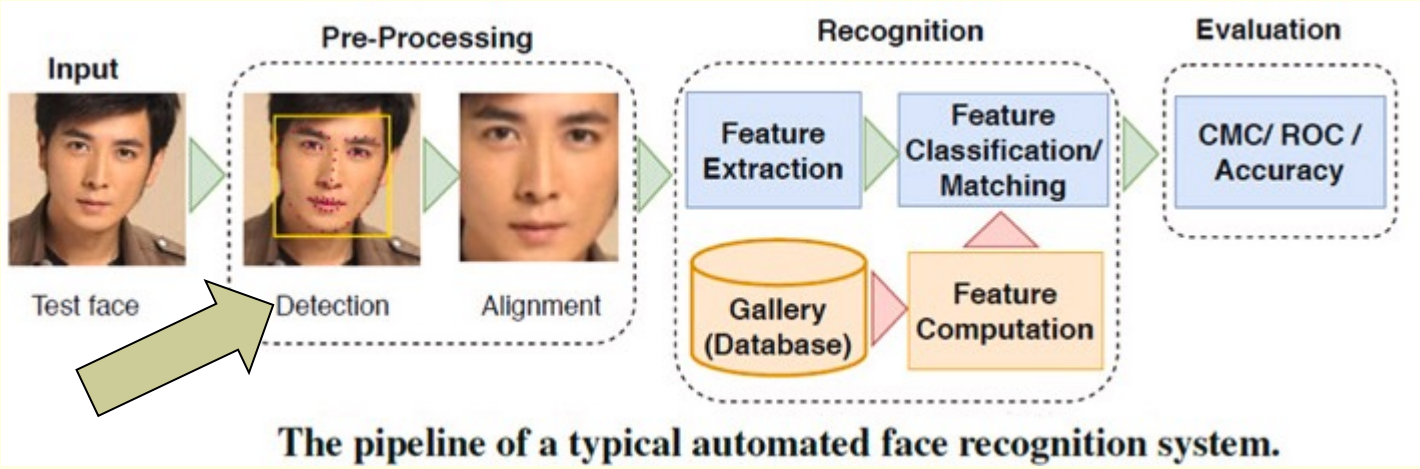
- Given an input image determine:
  - if faces are present in the image
  - the number of faces that are present
  - the location and extent of each face
  - the pose of each face
  - the identity of each face

Face Detection: Before Recognition



© Adrian Rosebrock

Image Courtesy: https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/



The pipeline of a typical automated face recognition system.

Guo and Zhang, "A Survey on Deep Learning Based Face Recognition," Computer Vision and Image Understanding (CVIU), 2019

How Many Faces?

How Many Faces?

How Many Faces?

Segmentation: Face Detection

Face: Optical Illusions

Face: Optical Illusions

In the Literature

- Appearance-based methods: use classifiers that operate directly on the normalized pixel intensity of the image without attempting to extract any facial features [Sung, 1998]
- Rule-based methods: employ knowledge of the components of the face viz., the eyes, nose and mouth, and their relationships [Yang, 1994]
- Feature-based methods: use grouping of edges, skin color, shape from shading, template matching methods, etc. to detect faces [Gao, 2002]
- Texture-based methods: use textural features to represent and detect facial patterns [Dai,1996]
- Deep-learning based methods: use deep convolutional neural networks to detect faces in multiple scales and rotations [

Face Detection: Example

https://www.youtube.com/watch?v=Zedlhu_QCjE

Viola Jones Technique Overview

- An image representation based on integral image that allows for very fast feature extraction

- A simple and efficient classifier based on Adaboost that selects a small number of important features from a huge library of potential features

- A combination of simple classifiers into a cascade structure that dramatically increases the speed of detector

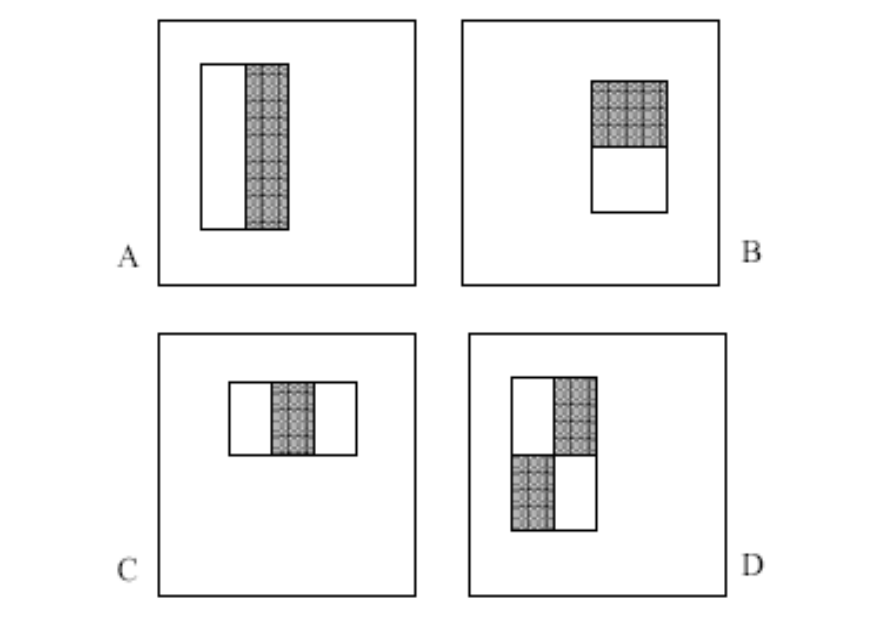
Features

- Three kinds of features are used:

a) Two-rectangle feature

b) Three-rectangle feature

c) Four-rectangle feature



$$f(x) = \sum_{i,j} W_{i,j} - \sum_{i,j} G_{i,j}$$

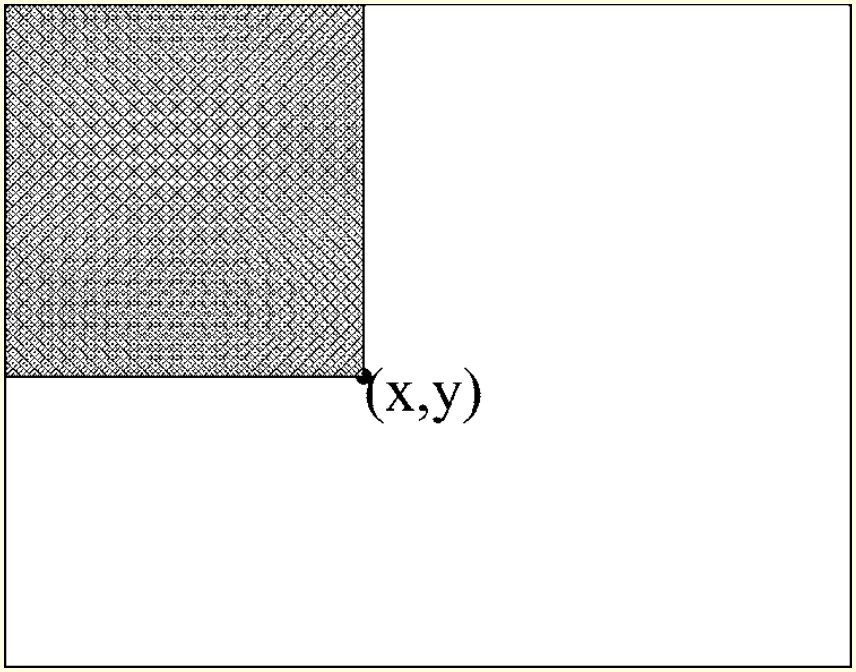
W : white regions

G : gray regions

Based on 24x24 image resolution:
a set of160,000 features is generated

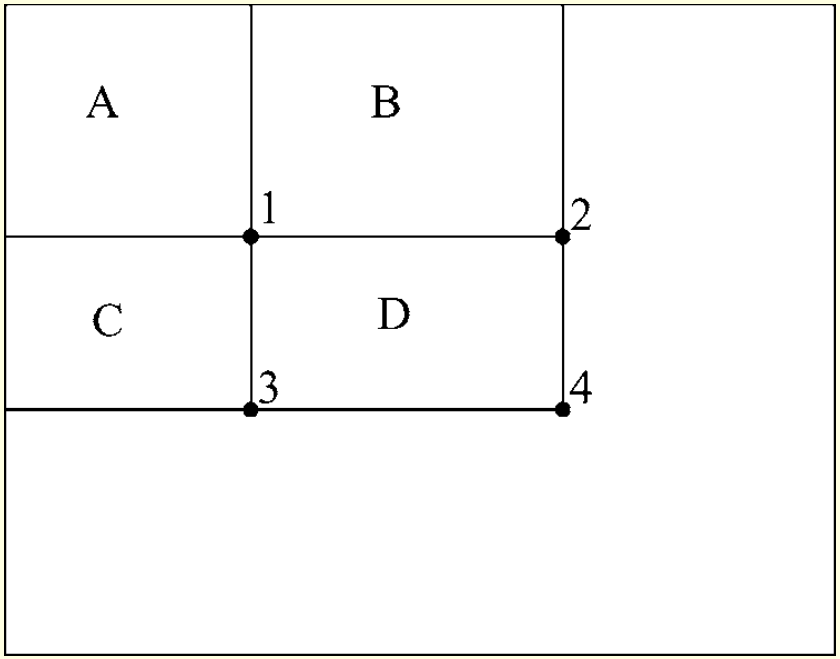
Integral Image

- Rectangle features can be computed very fast using an intermediate representation called integral image



The value of the integral image at point (*x, y) is the sum* of all the pixels above and to the left.

Integral Image



- The sum of the pixels within rectangle D can be computed with four array references
- The value of the integral image at location 1 is the sum of the pixels in rectangle A
- The value at location 2 is A+B, at location 3 is A+C, and at location 4 is A+B+C+D
- The sum within D can be computed as I(4) + I(1) - I(2) - I(3)

Feature Selection

- There are 160,000 rectangle filters (features) associated with each image sub-window of size 24x24

- Even though each feature can be computed very efficiently, computing the complete set is prohibitively expensive

- The hypothesis is that a very small number of these features can be combined to form an effective classifier

Adaboost

- A variant of Adaboost is used both to select the features and to train the classifier (Freund and Schapire, 1995).

- In the language of boosting the simple learning algorithm is called a weak learner

- The learner is called weak because we do not expect even the best classification function to classify the training data well

Boosting (Schapire 1989)

- Randomly select $n_1 < n$ samples from D without replacement to obtain D_1
  - Train weak classifier C_1

- Select $n_2 < n$ samples from D with half of the samples misclassified by C_1 to obtain D_2
  - Train weak classifier C_2

- Select all samples from D that C_1 and C_2 disagree on
  - Train weak classifier C_3

- Final classifier is vote of weak classifiers

Adaboost

- The weak learning algorithm is designed to select the single rectangle filter (feature) which **best separates** the positive and negative examples
- A weak classifier h(x, f, p, θ) consists of a filter (f), a window (x), a threshold (θ) and a polarity (p) indicating the direction of the inequality:

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases}$$

In practice no single filter can perform the classification task with low error. Features which are selected early in the process yield error rates between 0.1 and 0.3. Features selected in later rounds, as the task becomes more difficult, yield error rates between 0.4 and 0.5.

Initialization

- Step 1

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.

Insight: Every training sample has a weight associated with it

Updating

- Step 2: Do this "T" times

1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
2. Select the best weak classifier with respect to the weighted error

$$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \left| h(x_i, f, p, \theta) - y_i \right| .$$

Insight: Select the classifier that results in the least error, i.e., classifier that most correctly classifies face and non-face samples

Updating

- Step 2

3. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t, and θ_t are the minimizers of ϵ_t.
4. Update the weights:

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

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

Insight: Update weights associated with each training sample.
If sample was correctly classified: update its weight
If sample was incorrectly classified: retain old weight

Decision

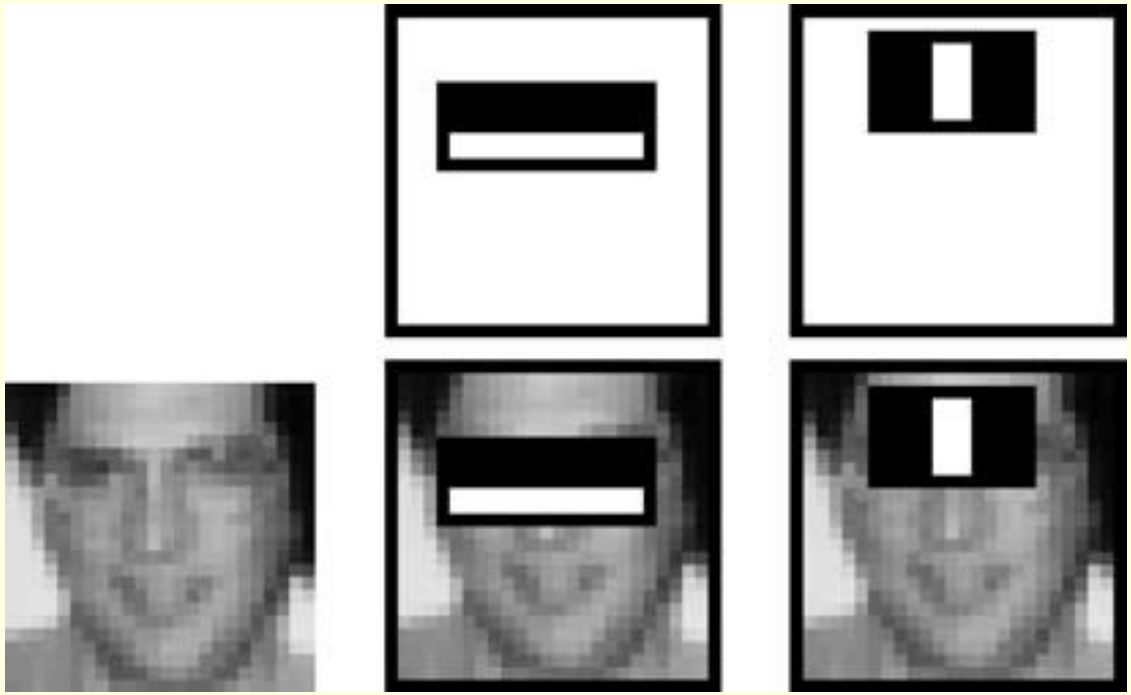
- Step 3: Final Strong Classifier

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

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

Insight: Final classifier is weighted sum of weak classifiers

Feature Selection Example



• The first and second filters (features) selected by AdaBoost. The two filters are shown in the top row and then overlaid on a typical training face in the bottom row

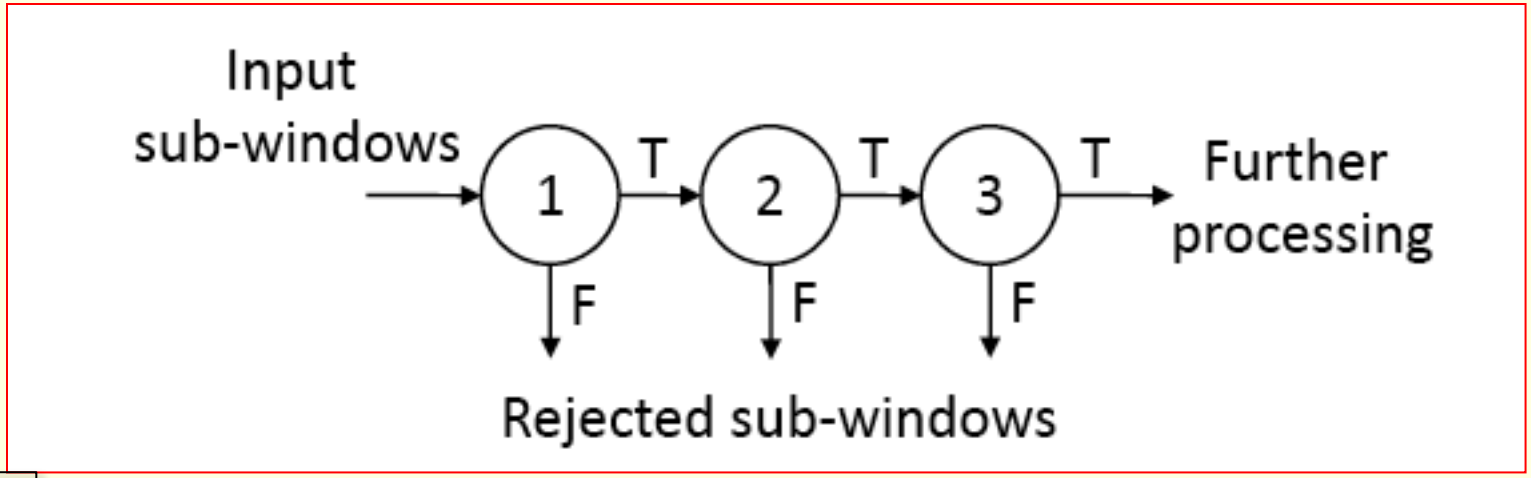
Cascade

- The key insight is that smaller, and therefore more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances

- Simple classifiers are used to reject the majority of sub-windows before more complex classifiers are called upon to achieve low false positive rates

Cascade

- A positive result from the first classifier triggers the evaluation of a second classifier which has also been adjusted to achieve very high detection rates
- A positive result from the second classifier triggers a third classifier, and so on
- A negative outcome at any point leads to the immediate rejection of the sub-window.



Cascade

- The structure of the cascade reflects the fact that within any single image an overwhelming majority of sub-windows are negative

- As such, the cascade attempts to reject as many negatives as possible at the earliest stage possible

Training Dataset

- The face training set consisted of 4916 hand labeled faces scaled and aligned to a base resolution of 24 by 24 pixels.

Training Dataset

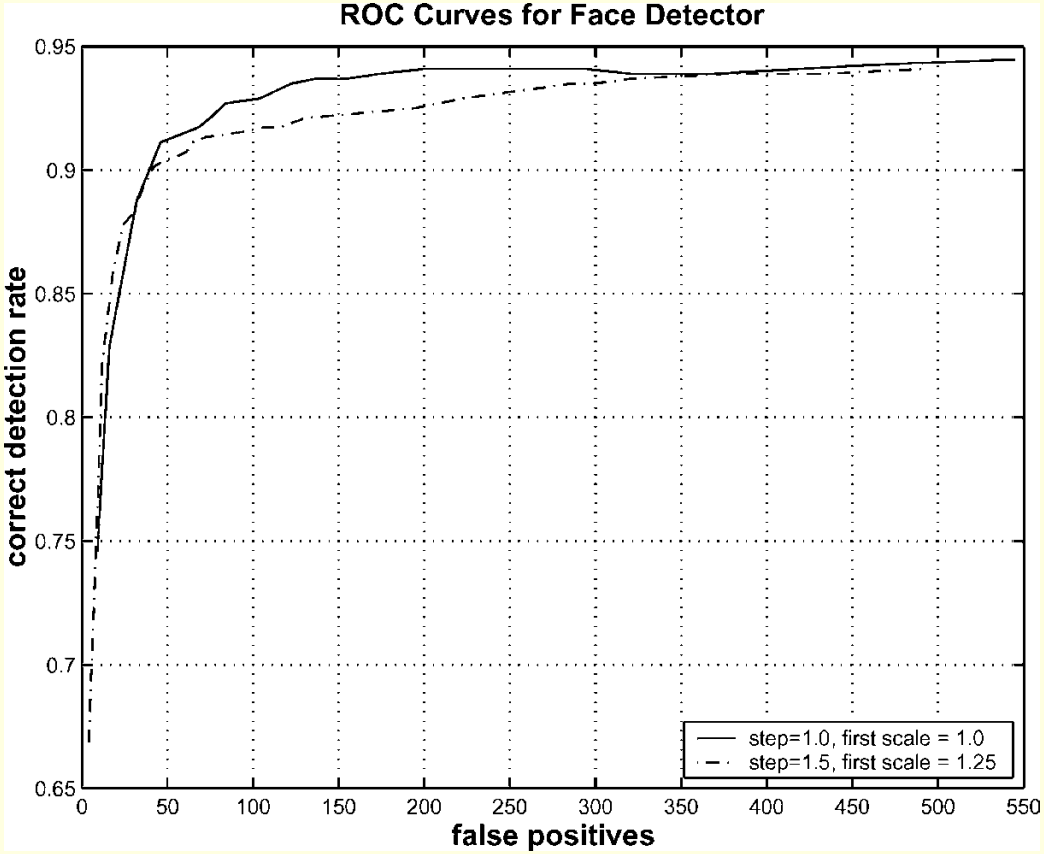


Negative Samples

The non-face sub-windows used to train the first level of the cascade were collected by selecting random sub-windows from a set of 9500 images which did not contain faces.

Experiments on a Real-World Test Set

- The system was tested on the MIT + CMU frontal face test set. This set consists of 130 images with 507 labeled frontal faces.



How many classifiers?

- The final detector is a 38 layer cascade of classifiers which included a total of 6060 features

© Rohan Gupta

Image Courtesy: https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999

Principle of Adaboost

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

Strong classifier

Features vector

Weight

Weak classifier

Credits to Antonio Torralba @MIT and Andras Ferencz @Princeton

OpenCV Demo

- Enter windows command line

- Usage: FaceDetector image_name.jpg window_size

  e.g., FaceDetector test3.jpg 40
  e.g., FaceDetector test3.jpg 20 (more false positives)

- The detection output would be automatically saved as detect_output.jpg in the local path.

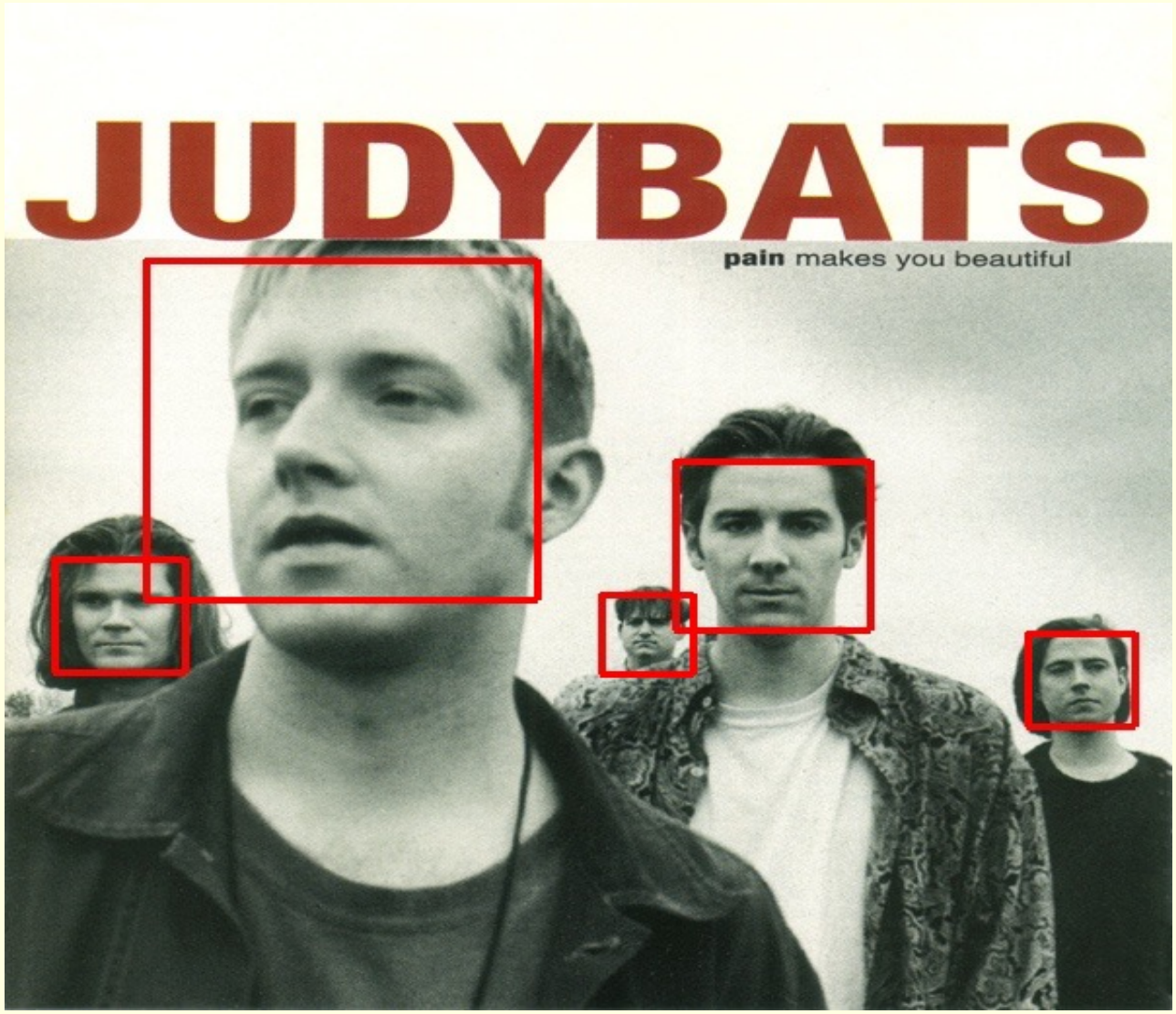
- The following results are produced when a minimum window size of 40 is used

Sample Results

Sample Results

Sample Results

Sample Results



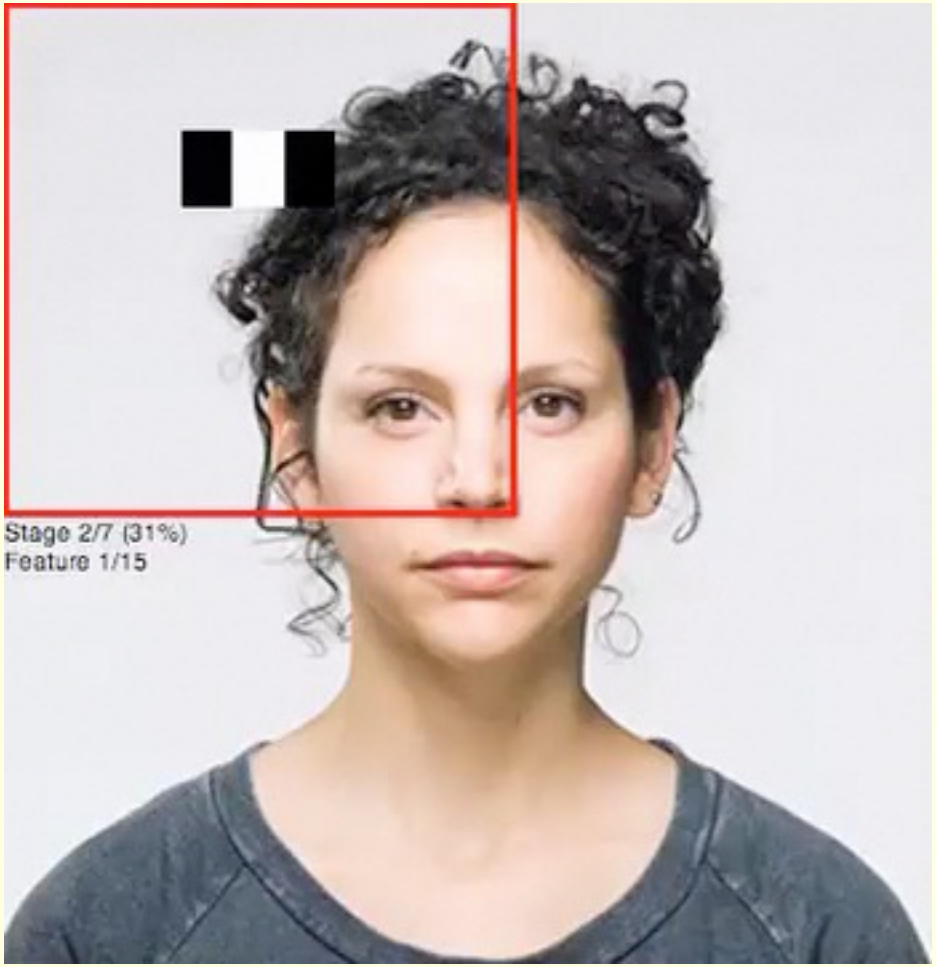
Conclusions by VJ

- The first contribution is a new technique for computing a rich set of image features using the integral image

- The second contribution of this paper is a simple and efficient classifier built from computationally efficient features using AdaBoost for feature selection

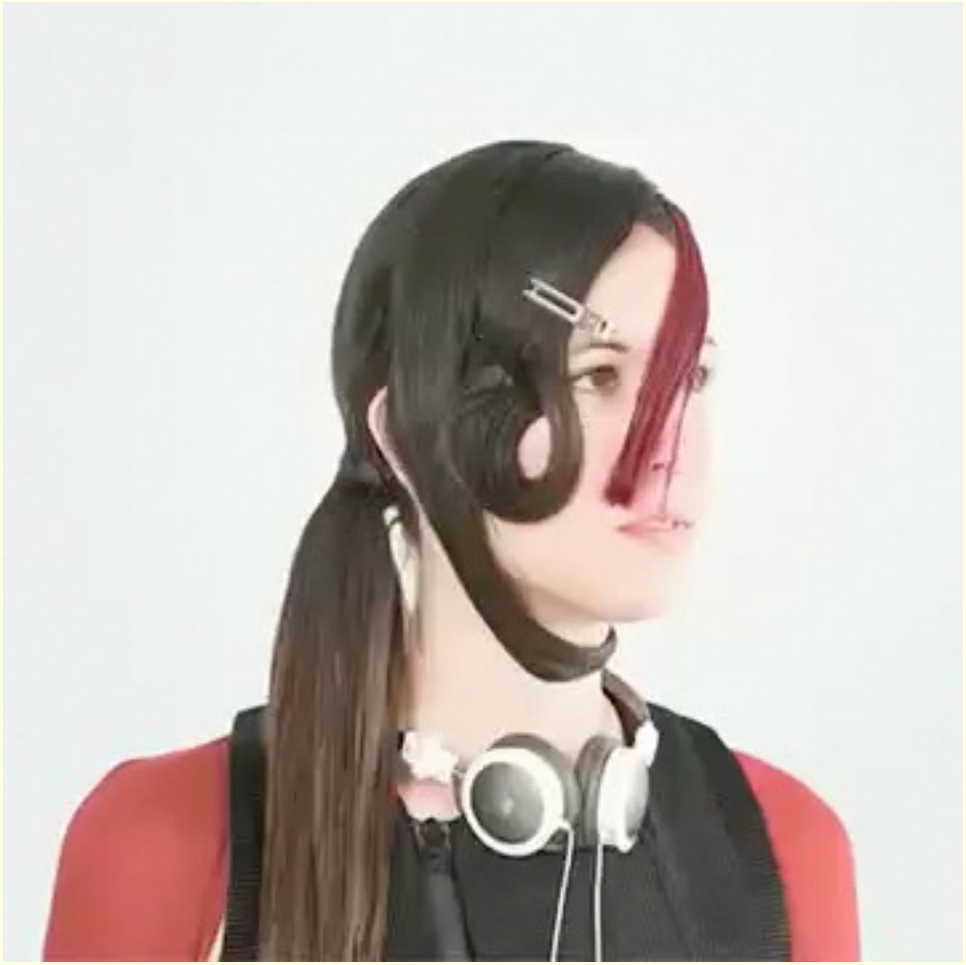
- The third contribution of this paper is a technique for constructing a cascade of classifiers which radically reduces computation time while improving detection accuracy

VJ Face Detector



VJ Face Detector: Fail Cases!

VJ Face Detector: Fail Cases!



VJ Face Detector: Fail Cases!

VJ Face Detector: Fail Cases!

VJ Face Detector: Fail Cases!

VJ Face Detector: Fail Cases!

Face Detection With Deep Learning

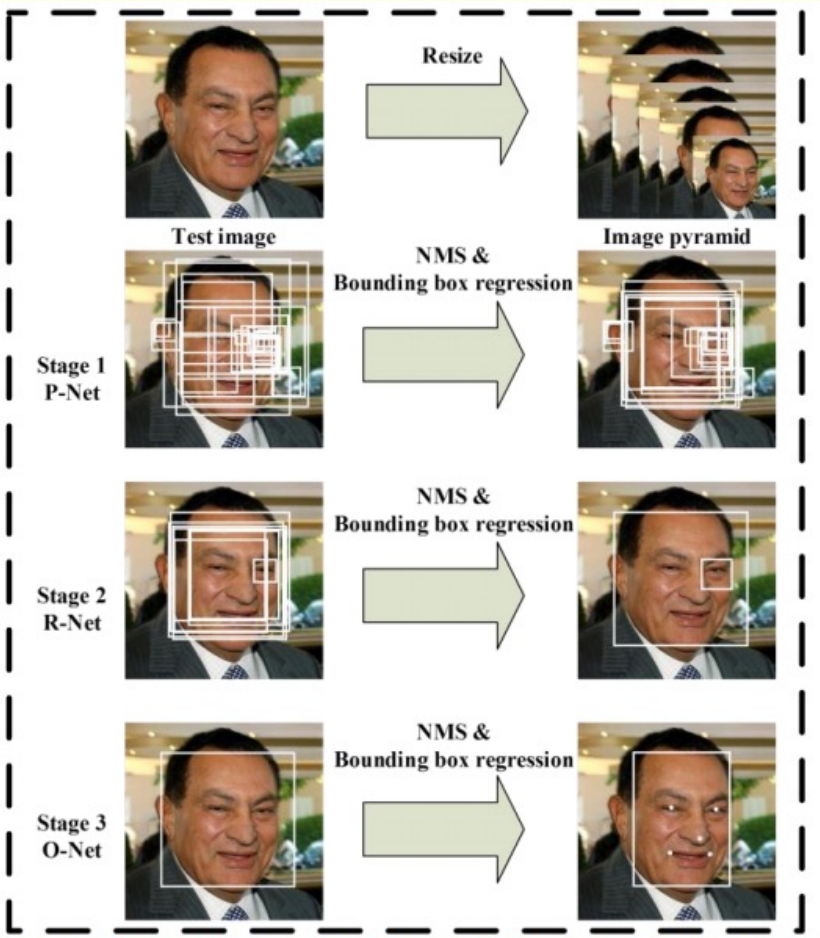
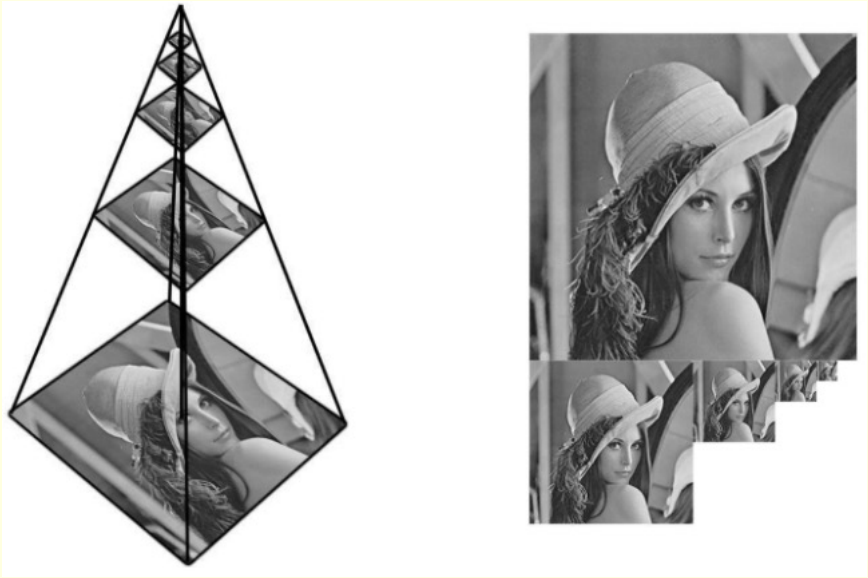


Fig. 1. Pipeline of our cascaded framework that includes three-stage multi-task deep convolutional networks. Firstly, candidate windows are produced through a fast Proposal Network (P-Net). After that, we refine these candidates in the next stage through a Refinement Network (R-Net). In the third stage, The Output Network (O-Net) produces final bounding box and facial landmarks position.

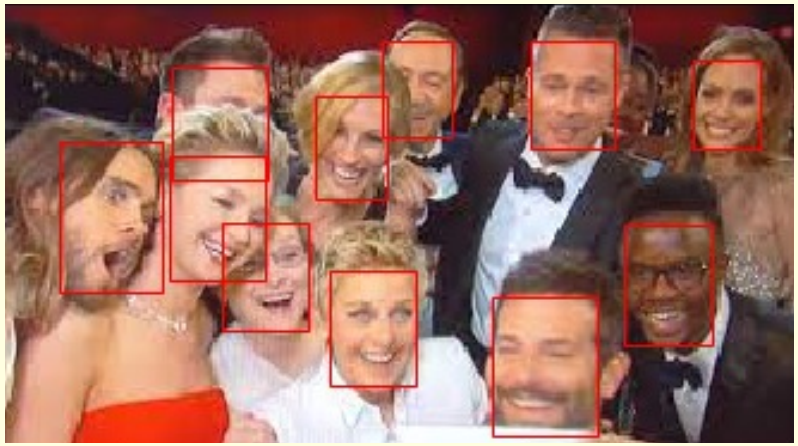


Zhang et al., "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks", IEEE Signal Processing Letters (SPL), 2016

MTCNN Examples

Zhang et al., "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks", IEEE Signal Processing Letters (SPL), 2016

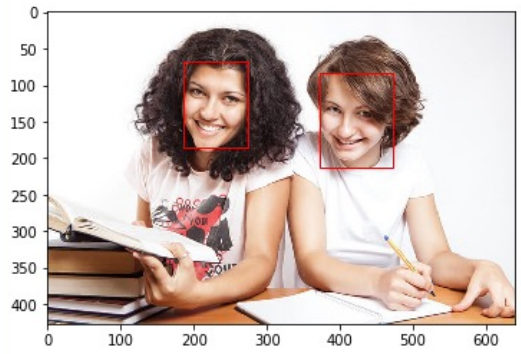
MTCNN Examples



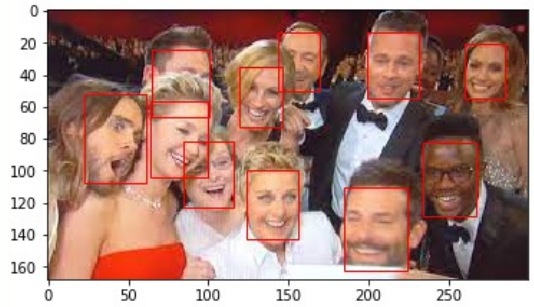
Zhang et al., "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks", IEEE Signal Processing Letters (SPL), 2016

MTCNN Code

```python
# -*- coding: utf-8 -*-
"""
Created on Sun May 16 10:28:38 2021

@author: banerj24
"""
# face detection with mtcnn on a photograph
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mtcnn.mtcnn import MTCNN

# draw an image with detected objects
def draw_image_with_boxes(filename, result_list):
    # load the image
    data = pyplot.imread(filename)
    # plot the image
    pyplot.imshow(data)
    # get the context for drawing boxes
    ax = pyplot.gca()
    # plot each box
    for result in result_list:
        # get coordinates
        x, y, width, height = result['box']
        # create the shape
        rect = Rectangle((x, y), width, height, fill=False, color='red')
        # draw the box
        ax.add_patch(rect)
    # show the plot
    pyplot.savefig('Manyfaces.png')
    pyplot.show()

filename = 'test2.jpg'
# load image from file
pixels = pyplot.imread(filename)
#pyplot.imshow(pixels)
# create the detector, using default weights
detector = MTCNN()
# detect faces in the image
faces = detector.detect_faces(pixels)
# display faces on the original image
draw_image_with_boxes(filename, faces)
```




```
In [2]: runfile('S:/SROP Biometrics Course/FaceDetect.py', wdir='S:/SROP Biometrics Course')
```




```
In [3]:
```

https://machinelearningmastery.com/how-to-perform-face-detection-with-classical-and-deep-learning-methods-in-python-with-keras/

More Examples

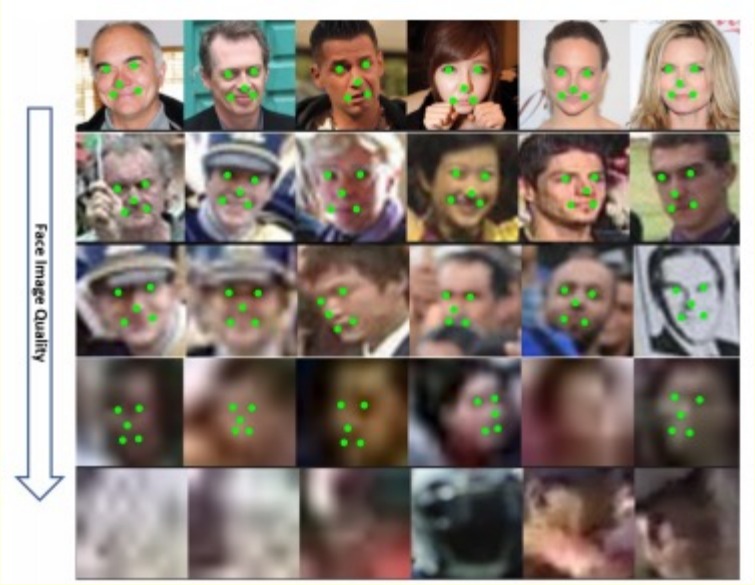
Image Courtesy: https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c

RetinaFace

- RetinaFace - A single stage face detector

- Performs pixel-wise face localization on different scales

- Predicts 3D shape face information

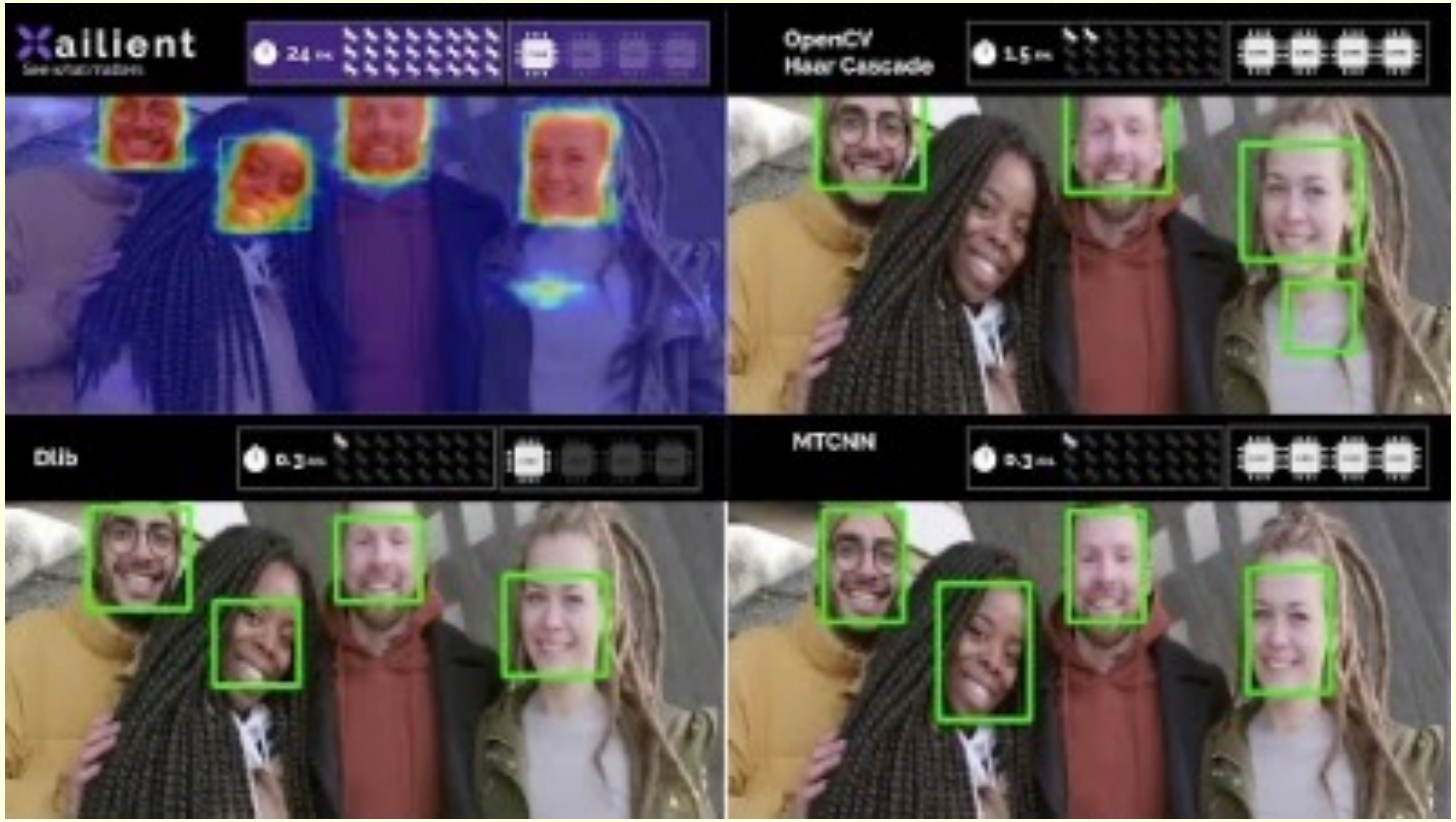


Deng et al., "RetinaFace: Single Stage Dense Face-Localisation in the Wild," arXiv, 2019

RetinaFace Examples

Image Courtesy: https://arxiv.org/pdf/1905.00641.pdf

Commercial Face Detector



Video Courtesy: https://www.edge-ai-vision.com/2021/01/the-battlefield-of-cutting-edge-face-detectors/
https://www.youtube.com/watch?v=TWWJ5eyOCVI

Open-Source Face Detectors

List of Face Detection Software Solutions	Deployment	Device Supported	Unique Feature
OpenBR	Open-API, On-Premise	Windows, Mac, Linux	A complete NIST compliant software that evaluates facial recognition, detection, and land-marking.
Flandmark	Open-API	Windows, Linux	A structured output classifier that relies on Deformable Part Models (DPM).
OpenFaceTracker	Open-API	Windows	It is developed as a modular library. Hence, it can either enable or disable some part of the software.
OpenEBTS	Open-API	Windows, Web-Based	It includes two open-source APIs: OpenEBTS API and OpenM1 API.
Bioenable Tech - iFace	Open-API, On-Premise	Web-Based	It is a multi-biometric identification program integrated with a 630MHz high-speed multi bio processor.
Bioenable Tech - vFace	Open-API, On-Premise	Web-Based	vFace system is made available in elegant ergonomic design and requires 3inch TFT touch screen, nine digits user ID, and T9 input.
FacePlusPlus	Cloud-Hosted	Windows, Mac, Web-Based	Integrated C+E solutions, excellent accuracy, robust anti-spoofing technique, and frequent model updating.
DeepFace	Cloud-Hosted	Windows, Mac, Web-Based	It facilitates creating a database from faces and search based-on a given image or name.

Image Courtesy: https://www.goodfirms.co/blog/best-free-open-source-face-detection-software-solutions