



# Find my Photos!

Presented To: DR/SHADY ZAHRAN  
ENG/MOHAMED MOHEB

**Team Members:**  
**Ahmed Hazem**  
**Mohamed Omar**



## **Executive Summary:**

This project is centered around the development and implementation of an advanced person detection system integrated with an image recognition algorithm. The primary objective is to accurately identify individuals within a given set of images and provide each person's photos in which he appears. The main outcome of it is to rearrange photos depending on the person, it will achieve speeding up searching for person photo which is a part of the big list of photos, and the results will be saving time. In conclusion, the person detection and image recognition system presented in this project significantly advances the capabilities of identifying individuals within images. Its high accuracy and efficient photo tracking functionality make it a versatile tool with wide-ranging applications.

## Contents

Introduction .....	3
Literature Review .....	4
Methodology .....	5
<b>CNN information:</b> .....	5
2. <b>ReLU (Rectified Linear Unit): An activation functions commonly used in neural network.</b> ...	6
MTCNN Multi-task Cascaded Convolutional Networks: .....	7
Classification: .....	7
HOG (Histogram of Oriented Gradients): .....	13
Design Process: .....	13
Advantages.....	15
Coding Practices: .....	15
<i>Data Acquisition:</i> .....	15
<i>Datasets:</i> .....	16
Project Methodology: .....	17
Technical Details: .....	19
Result .....	33
Discussion: .....	59
Conclusion .....	59
Reference .....	61

## Introduction

In the contemporary landscape of digital imagery, our project aims to tackle the challenge of efficiently navigating extensive photo collections. Situated at the convergence of image processing, face recognition, and user interface design, our application responds to the growing need for a seamless solution in identifying and retrieving individual portraits within curated photo repositories. As individuals and organizations grapple with the task of swiftly locating specific photos within a large list of photos, our project addresses this demand by streamlining the process. We outline clear objectives: implementing robust image processing techniques, incorporating advanced face detection and recognition algorithms, and creating an efficient search algorithm. The project scope encompasses the development of a user-friendly standalone application, processing photos in various formats, and adhering to ethical considerations regarding privacy. The subsequent sections of this report will delve into the methodology, technical intricacies, and considerations related to privacy, testing, and deployment, offering a comprehensive understanding of the application's development and functionality.

## Literature Review

Facial detection and alignment play crucial roles in computer vision applications, including face recognition, human-computer interaction, and surveillance. This literature review explores two prominent approaches in the field: "Joint Face Detection and Alignment using Multitask Cascaded Convolutional Networks" by Zhang et al. and "Histograms of Oriented Gradients for Human Detection" by Dalal and Triggs. The objective is to understand the strengths and limitations of each method and discuss potential synergies for a combined approach.

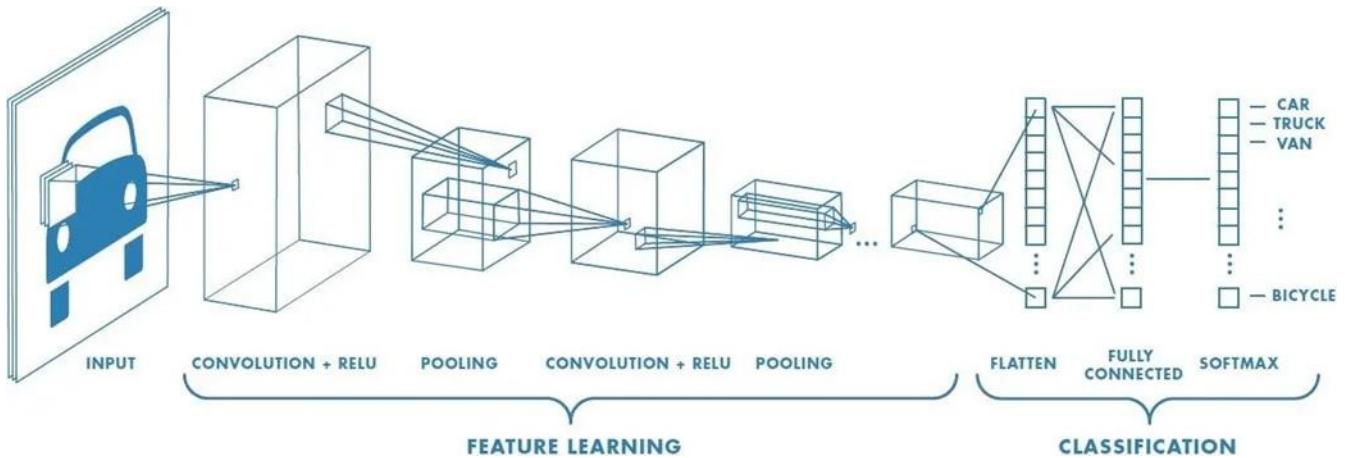
Multitask Cascaded Convolutional Networks for Face Detection and Alignment

Zhang et al. proposed a novel approach that jointly addresses face detection and alignment using Multitask Cascaded Convolutional Networks (MTCNN). MTCNN consists of three cascaded stages: face proposal network (P-Net), refinement network (R-Net), and calibration network (O-Net). This architecture effectively handles faces at different scales, achieving state-of-the-art results in terms of accuracy and speed. The integration of tasks optimizes computation, making it suitable for real-time applications. MTCNN has very good points of strengths like High accuracy it demonstrates superior face detection and alignment accuracy, Multiscale processing which the cascaded architecture efficiently handles faces of varying sizes, Real-time applicability which is designed to allows for efficient processing, making it suitable for real-time applications. But it has one limitation. It depends on labeled data: MTCNN relies on large amounts of labeled data for training, which may be challenging to obtain for specific domains.

The Histograms of Oriented Gradients (HOG) approach by Dalal and Triggs focuses on human detection but can be adapted for facial detection. HOG extracts gradient information from image regions and utilizes histograms to represent local object appearance and shape. It has strengths like Robust feature representation which it captures gradient information effectively, providing a robust representation of object appearance, Invariance to illumination changes: HOG is less affected by variations in lighting conditions, enhancing its generalization capabilities. But it has limitation like with color images which it operates on grayscale images, potentially limiting its performance in scenarios where color information is crucial, Sensitivity to scale changes which may face challenges in handling objects at different scales. Combining MTCNN and HOG leverages the strengths of both methods. MTCNN's multiscale processing and real-time applicability complement HOG's robust feature representation. Integrating these techniques could enhance the ability to get people photos per each person, especially when faced with challenges such as varying lighting conditions and scale changes.

# Methodology

## CNN information:



**CNN stages consist of 3 stages, each stage consist of 3 (convolution, relu,polling(subsampling)):**

### 1. Convolutional:

- A layer that consists of a set of “filters”. Linear/matrix multiplications operations are performed by using a subset of the weights of a dense layer. Nearby inputs are connected to nearby outputs. The weights for the convolutions at each location are shared, but they go through an activation function at the output, which is usually a non-linear operation. (sigmoid).
- **Stride:** apply filter by each n pixel in left or down.
- **Types:**
  - ❖ **valid:** It means no padding,  $p = 0$  so output will be  $n+2p - f+1 = n - f + 1$  [smaller]
    - EX: if you have a  $n$  by  $n$  image and convolve it with an  $f$ -by- $f$  filter, then the dimension of the output will be  $(n-f+1)$  by  $(n-f+1)$ .
    - And so, for the 6 by 6 image and the 3 by 3 filter on the left, the output image size would be a  $(6-3+1)$  by  $(6-3+1)$  which simplifies to a 4 by 4 output.
  - ❖ There are two problems when you convolve a filter to an image:
    - The shrinking output
    - Throwing away information from the edges of the image So Same come.
  - ❖ **same:** It means with padding so output will be  $n+2p - f+1$  [**Same or bigger**]
    - ❖ EX: if you have an  $n$  by  $n$  image and convolve it with an  $f$ -by- $f$  filter, then the dimension of the output will be  $(n+2p - f+1)$  by  $(n+2p - f+1)$ . And so, for the 6 by 6 image and the 3 by 3 filter on the left, the output image size would be a  $(6+2*1-3+1)$  by  $(6+2*1-3+1)$  which simplifies to a 6 by 6 output.
  - ❖ **full:** mix between them.
  - ❖ **Stride with padding:** output will be floor of  $[(n+2p-f)/S + 1]$ .

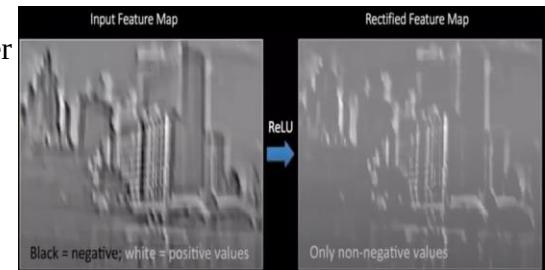
2. **ReLU (Rectified Linear Unit):** An activation functions commonly used in neural network.

**Definition:**  $f(x) = \max(0, x)$ , where  $x$  is the input to the function.

- if value 0 or smaller than 0 set it into 0

- if bigger than 0 let it

then purpose is to extract positive values only  
(features which we need, depending on conv filter  
which was applied)



3. **PRelu (Parametric Rectified Linear Unit):** is an extension of ReLU

It introduces a learnable parameter to the activation function.

- **Definition:**  $f(x) = \max(0, x) + a \cdot \min(0, x)$ , where  $a$  is a learnable **parameter**.

# **Pooling:** Replace each block in the input with a single output

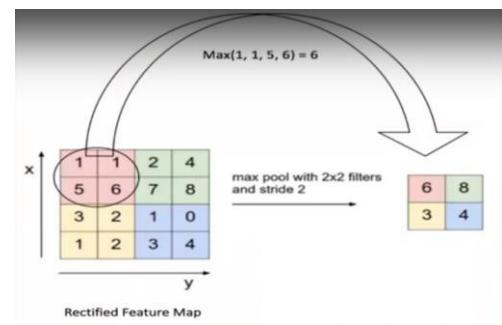
can be **max**: get the max value from batch.

can be **sum SIGMA** ( $\text{pixel} * \text{kernel element}) / n$  of pixels.

**Purpose:**

1- Force model to learn from important values only and disallow noise and garbage from moving.

2- Decreasing complexity



# **Flatten:** convert last layer 3D output into a 1D vector

# **Dense** (fully connected layer): A linear operation, describes how the neurons are connected to the next layer of neurons, generally followed by a non-linear activation function (sigmoid)  
subsampling: values will go to half using 2 X 2 filter and take the biggest of the 4 pixels.

# **Dropout:** telling to use n% from neurons to prevent overfitting

# **Softmax** : activation function is applied to scores to convert them into probabilities [0 to 1].

# **Output layer:** Produces the final predictions.

## MTCNN Multi-task Cascaded Convolutional Networks:

Classification:



### Design Process:

- **Algorithm Design:**

Before starting with algorithm, we should resize input image into two different scales to build an image pyramids. There is a question which should be answered why pyramids? The answer is so easy to be able to make different copies of the same image in different sizes to search for different sized faces within the image. This makes the downside which we must recalculate all indexes related to the stride therefore we have to multiply the index by 2 to get the correct coordinate.

### The proposed CNNs consist of 3 stages:

Most convolution layer uses stride of 2 why? Having a stride of 2 helps reduce computation operations to quarter therefore complexity without significantly sacrificing accuracy.

### First stage is called Proposal Network (P-Net):

The weights and biases of P-Net have been trained so that it outputs a relatively accurate bounding box.

- 1- All coordinates are being normalized from (0,0) top left corner to (1,1) bottom right corner. The weights and biases of P-Net have been trained so that it outputs a relatively accurate bounding box.
- 2 - It calculates confidence [0 to 1] to be face by the help of convolution layer filters for each generated boxes [many are not accurate]
- 3- It sorts them from highest confidence to the lowest confidence.
- 4- It picks out boxes with higher confidence
- 5- It standardizes the coordinate system which means converting all the coordinate systems to that of the actual “un-scaled” image.

**We face 2 issues:**

- 1- a lot of bounding boxes left.
- 2- 2- a lot of them overlap.

So, we found a solution. It is easy to use **Non-Maximum Suppression (NMS)**.

1- We check overlapping using something is called (IoU) we will talk about it later.

2- check 2 overlapping boxes confidence

3- Remove box which lowers confidence.

Therefore, it gets rid of redundant bounding boxes, allowing us to narrow our search down to one accurate box per face.

Kernel Coordinates	Bounding box: x1	Bounding box: y1	Bounding box: x2	Bounding box: y2	Confidence
(0,5)	0.50	0.25	0.80	0.58	0.98
(0,6)	0.45	0.17	0.75	0.49	0.96
(0,7)	0.52	0.08	0.73	0.41	0.94
(1,4)	0.42	0.34	0.67	0.66	0.92
(1,8)	0.40	0.01	0.66	0.32	0.96
(3,5)	0.32	0.22	0.49	0.59	0.88
(3,6)	0.25	0.20	0.52	0.53	0.91
(6,6)	0.01	0.18	0.25	0.50	0.89
(6,8)	0.02	0.00	0.22	0.33	0.90

**There is good question why we don't we just choose the box with the highest confidence and delete everything else?** There might be more than one face in other images. If so, we would end up deleting all the bounding boxes for the other faces.

Now lets take box coordinates from normalized scaled and converting it to coordinates in actual scale , lets take example :

In image,  $24 \times 24$  kernel is represented by the red box,  
We can calculate the width and height of the kernel:

$$500 - 200 = 300, 800 - 500 = 300$$

The width and height we get here are the width and height of  
the kernel when scaled back to its original size.

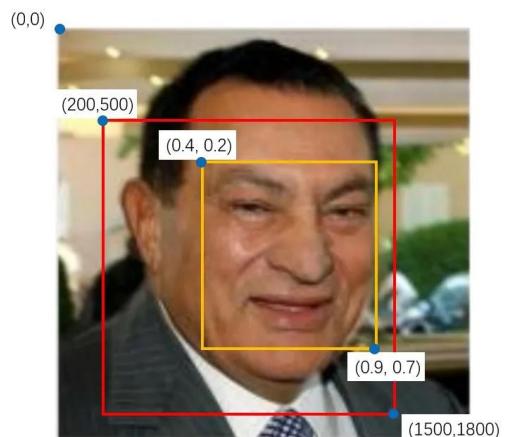
$$300 * 0.4 = 120, 300 * 0.2 = 60$$

$$300 * 0.9 = 270, 300 * 0.7 = 210$$

Finally, actual cord:

$$-(200+120, 500+60) \Rightarrow (320, 560)$$

$$-(200+270, 500+210) \Rightarrow (470, 710).$$

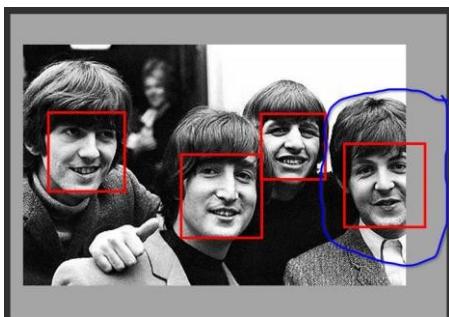


To achieve square bounding boxes, we adjust their shapes by  
extending the shorter sides. If the width is less than the height, we expand horizontally;  
conversely, if the height is less than the width, we expand vertically. This reshaping ensures  
uniformity in the bounding box dimensions.

Finally, bounding boxes were saved and passed on to second stage.

### Second stage is called Refine Network (R-Net):

Sometimes, an image may contain only a part of a face peeking in from the side of the frame.  
In that case, the network may return a bounding box.



For every bounding box:

- 1 - We create an array of the same size
- 2 - Copy the pixel values (the image in the bounding box) to the new array.
- 3 - Fill in everything else with a 0. [process of filling arrays with 0s is called **padding**.]
- 4 - We resize them to  $24 \times 24$  pixels
- 5 - Normalize them to values between -1 and 1:
  - Currently, the pixel values are between 0 to 255 (RGB values).
  - By subtracting each pixel value by half of 255 (127.5) and dividing it by 127.5,
    - we can keep their values between -1 and 1.

We can feed them into R-Net and gather its output.

R-Net's output is like that of P-Net: It includes:

- The coordinates of the new, more accurate bounding boxes.
- Well as the confidence level of each of these bounding boxes.
- we eliminate the boxes with lower confidence
- apply NMS on every box to further eliminate redundant boxes.

Since the coordinates of these new bounding boxes are based on the P-Net bounding boxes, we need to convert them to the standard coordinates.

After standardizing the coordinates, we reshape the bounding boxes to a square to be passed on to O-Net.

### **Third stage is called Output Network (O-Net):**

We have to first pad any boxes that are out-of-bounds. After we resize the boxes to 48 x 48 pixels, we can pass in the bounding boxes into O-Net

**Once again**, we get rid of the boxes with lower confidence levels and standardize both the bounding box coordinates and the facial landmark coordinates. Finally, we ran them through the last NMS. At this point, there should only be one bounding box for every face in the image.

### **O-Net provides 3 outputs:**

- the coordinates of the bounding box (out [0])
- the coordinates of the 5 facial landmarks (out [1])
- the confidence level of each box (out [2]).

### **• Coding Practices:**

- **python: 3.11**

- **libraries:**

- **cv2:** is a computer vision and machine learning software library.

- **NumPy:** is a powerful numerical computing library for Python.

- **pkg\_resources :** It provides runtime facilities for finding, introspecting, and managing Python packages and their resources.

- **TensorFlow:** TensorFlow is an open-source machine learning framework developed by the Google

Brain team.

- **keras :** is an open-source high-level neural networks API written in Python

- **tensorflow.keras.layers :** contains various pre-built layers for building neural network models, such as dense layers, convolutional layers, and recurrent layers.

- **tensorflow.keras.models :** provides functionalities for defining and training neural network models.

### **Data Acquisition:**

In the learning process we need different types of images (face, non-face and incompletely aligned face) for training each CNN such as :

1- FDDB (“Face Detection Dataset and Benchmark”) : contains a set of 2,845 images contains annotations for 5171 faces.

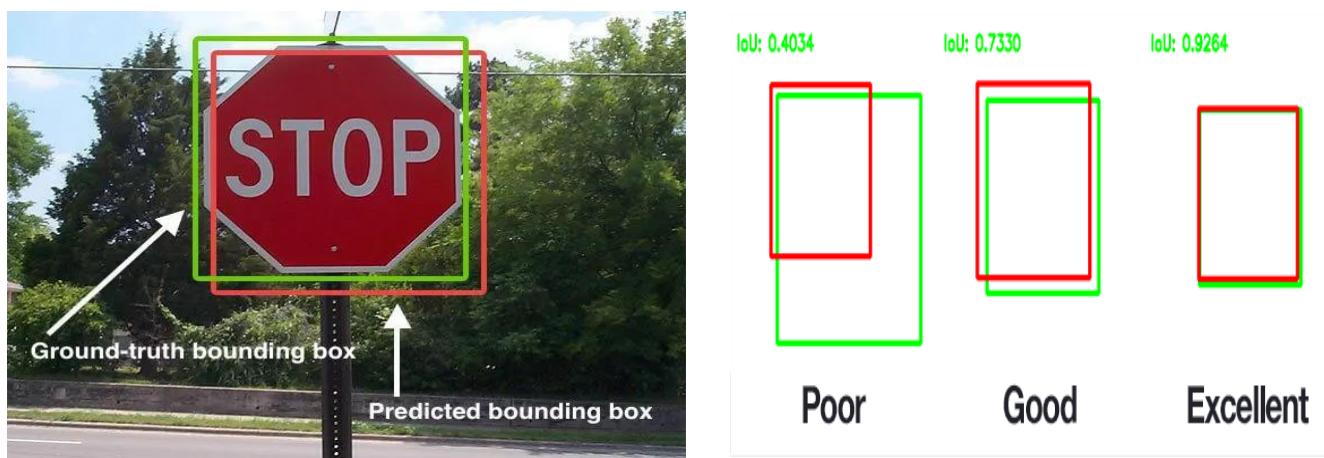
2- WIDER FACE has 32,203 images which has 32,203 images where

- A - 50% for testing according to the difficulty of images spitted into 3 groups
- B - 40% for training
- C - 10% for validation

3- AFLW contains the facial landmarks annotations for 24,386 faces

4- CelebA (“Celebrities Attributes”) is a large-scale face attributes dataset was created at the Chinese Academy of Sciences.

Four types of data annotation in our training process but before that we should speak about Intersection over Union (IoU).



It is an evaluation metric's performance of object detection by comparing the ground truth bounding box to the predicted bounding box and it is being computed as dividing the area of overlap between the bounding boxes by the area of union.

**Let's return to types:**

- A) Negatives: Regions that the Intersection-over-Union (IoU) ratio less than 0.3 to any ground-truth faces
- B) Positives: IoU above 0.65 to a ground truth face.
- C) Part faces: IoU between 0.4 and 0.65 to a ground truth face.
- D) Landmark faces: faces labeled 5 landmarks' positions.

### 1) P-Net:

- We randomly crop several patches from WIDER FACE to collect positives, negatives and part face.
- Then we crop faces from CelebA as landmark faces

### 2) R-Net:

- Detect face from wider face to collect positives , negatives and partface while landmark faces are detected from CelebA

### 3) O-Net:

Similar to R-Net to collect data but we use the first two stages of our framework to detect faces.

Before speaking about the effectiveness of online hard sample mining. Let's know some about it. It is a technique used in machine learning in the context of training deep neural networks, to improve the learning process by focusing on challenging. The term "online" indicates that this process occurs dynamically during the training procedure rather than being applied as a separate pre-processing step.

#### The effectiveness of online hard sample mining

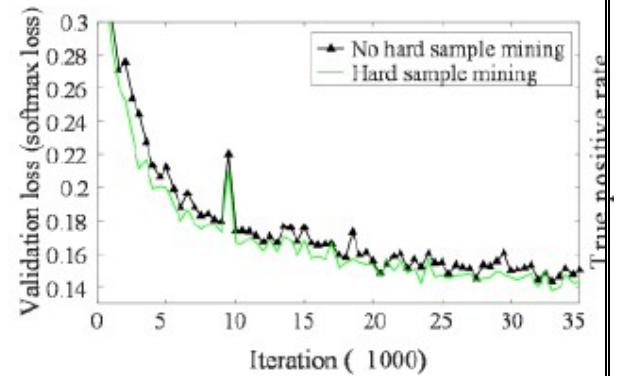
we train two O-Nets (with and without online hard sample mining) and compare their loss curves.

And We are using same training parameters including the network initialization

1- O-Nets was trained to classify the face.

2- Fix learning rate.

shows the loss curves from two different training ways. It is very clear that the hard sample mining is beneficial to performance improvement.

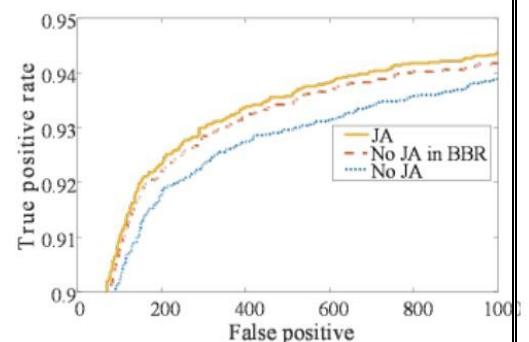


#### The effectiveness of joint detection and alignment

We can find that joint landmarks localization task learning is important for both face classification and bounding box regression tasks via

1- check the performances of two different O-Nets (joint facial landmarks regression task / do not join it) on FDDB (with the same First 2 layer).

2- compare the performance of bounding box regression in these two O-Nets.



#### Testing Methods:

The accuracy of model was 95.4%

**TABLE I**  
**COMPARISON OF SPEED AND VALIDATION ACCURACY OF OUR CNNs AND**  
**PREVIOUS CNNs [19]**

Group	CNN	300 Times Forward	Accuracy
Group1	12-Net [19]	0.038s	94.4%
Group1	P-Net	0.031s	94.6%
Group2	24-Net [19]	0.738s	95.1%
Group2	R-Net	0.458s	95.4%
Group3	48-Net [19]	3.577s	93.2%
Group3	O-Net	1.347s	95.4%

### **HOG (Histogram of Oriented Gradients):**

#### **Design Process:**

- **Algorithm Design:**

#### **Logic:**

HOG is a feature descriptor used for object detection in images. It works by capturing the distribution of gradient orientations in localized regions of an image. The basic idea is to represent an object based on the gradients of intensity in different directions, which helps capture the object's shape and edges.

#### **Theory:**

##### **1- Gradient Computation:**

Compute the gradient magnitudes and orientations of the image using first-order derivatives or Sobel masks. This step helps in detecting changes in light intensity, emphasizing edges and object boundaries.

## **2- Spatial/Orientation Binning:**

Divide the image into cells and further divide each cell into grids. For each grid, create histograms of gradient orientations based on the gradient magnitudes within that grid. This process helps capture local information about the image.

## **3- Block Normalization:**

Normalize the histograms within blocks of cells. This step is crucial for maintaining gradient values, ensuring that variations in illuminance do not affect the performance significantly. Common normalization methods include L2-Hys, L2-norm, L1-sqrt, and L1-norm.

## **4- Get the HOG Feature Vector:**

Construct a feature vector by concatenating the normalized histograms from all blocks. The feature vector represents the distribution of gradient orientations in the entire image.

## **Steps to Perform HOG:**

### **1- Gamma/Color Normalization (Optional):**

Adjust color intensities using gamma correction. This step is optional and might involve experimenting with different color spaces, but it has a minor impact on the overall HOG descriptor.

### **2- Gradient Computation:**

Compute gradients using first-order derivatives or Sobel masks. Choose the appropriate method for color or grayscale images.

### **3- Spatial/Orientation Binning (Dividing the image into cells):**

Divide the image into cells and further into grids. Calculate histograms of gradient orientations within each grid.

### **4- Block Normalization:**

Normalize histograms within blocks of cells to account for variations in illuminance.

## **5- Get the HOG Feature Vector:**

Concatenate normalized histograms to form a feature vector representing the entire image.

## **Advantages**

HOG is effective for object detection, especially in scenarios like pedestrian detection and medical image analysis.

It is computationally efficient for detecting small-scaled objects without requiring a powerful GPU.

## **Coding Practices:**

**Programming Language:** Python

## **Libraries/Tools:**

*the Histogram of Oriented Gradients (HOG) algorithm is commonly implemented using various image processing libraries. Some of the popular libraries and tools that provide functions for HOG feature extraction include:*

### ***OpenCV:***

*OpenCV (Open Source Computer Vision Library) is a widely-used computer vision library that includes functions for HOG feature extraction. The cv2.HOGDescriptor class in OpenCV allows you to compute HOG descriptors for images.*

### ***dlib:***

*dlib is a C++ toolkit with Python bindings that includes machine learning and computer vision algorithms. The dlib.get\_frontal\_face\_detector() function can be combined with HOG for face detection.*

## ***Data Acquisition:***

The Histogram of Oriented Gradients (HOG) algorithm itself doesn't have a specific dataset associated with it. Instead, the choice of dataset depends on the specific task you are trying to solve using HOG, such as object detection, human detection, or face recognition.

## **Datasets:**

### **MIT Pedestrian Database:**

A collection of images containing pedestrians in city scenes.

Comprises 509 training images and 200 test images, with front or back views and limited pose variations.

Commonly used for evaluating pedestrian detection algorithms.

### **INRIA Dataset:**

A more challenging dataset with 1805 images of humans cropped from personal photos.

Features individuals in various orientations against diverse backgrounds, including crowds.

Includes bystanders from image backgrounds, introducing pose and background variability.

## **Training**

### **Positive Training Examples:**

Selected 1239 positive training examples from the INRIA dataset and their left-right reflections (2478 images in total).

### **Negative Training Examples:**

Utilized a fixed set of 12,180 patches sampled randomly from person-free training photos as initial negative examples.

### **Detector Training:**

Trained a preliminary detector using positive and negative examples.

Conducted an exhaustive search for false positives (hard examples) in 1218 negative training photos.

Retrained the detector using the augmented set (initial negative set + hard examples) to produce the final detector.

### **Retraining Process:**

Significantly improved detector performance, especially at low false positive rates.

Additional rounds of retraining were considered but deemed not necessary.

## **Project Methodology:**

### **Design Process:**

- **Algorithm Design:**

- 1- Read path of images from folder
- 2- Read image and save it into object and make list of images
- 3- Use MTCNN to get faces boxes cord with threshold [0.8, 0.9, 0.9]  
min confidence for each layer
- 4- Convert mtcnn box format into face\_recognition box format.:  
# face\_recognition : top, right, bottom, left  
# MTCNN: left top RightP(width) bottomP(height)
- 5- Use face\_recognition to calculate encoding using hog with  
[num\_jitters=100] to rescale and zoom and etc. to extract more details
- 6- Make Person List and ConnectiontoList
- 7- Create Persons from First image
- 8- Create Connection between first image and each person
- 9- Check images from second images to last image:
  - if person exist in Person List [using encoding difference]: add connection then break
  - if person exist in Person List: Create Person, add connection then break
- 10- start check PersonList
  - check each person's connections.

- **Coding Practices:**

**python:** 3.11

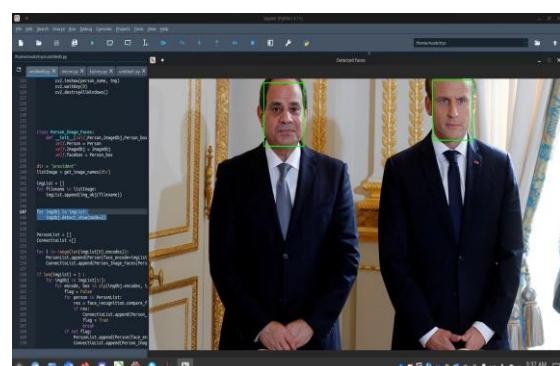
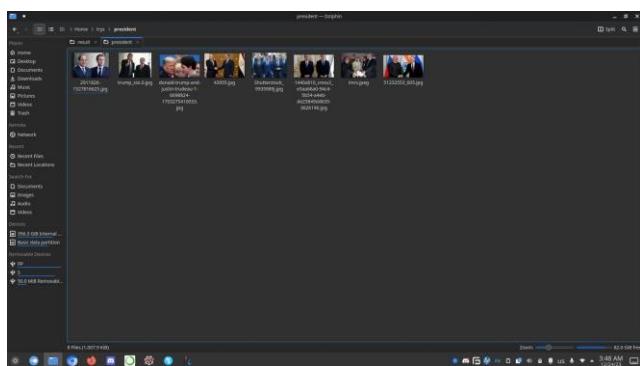
**libraries:**

- cv2: is a computer vision and machine learning software library.
- copy: create copy of image
- Faker: Use random names
- tkinter: standard GUI (Graphical User Interface) toolkit for the Tk GUI toolkit.
- os: Access OS stuff like files
- mtcnn: for face classification
- face\_recognition: for using hog algorithm to calculate encode for each person
- dlib : has face\_recognition dependences
- cuda drivers: For using GPU in calculation

**Data Acquisition:** We don't use any data to train, because mtcnn is already trained, just push pictures.

### **Testing Methods:**

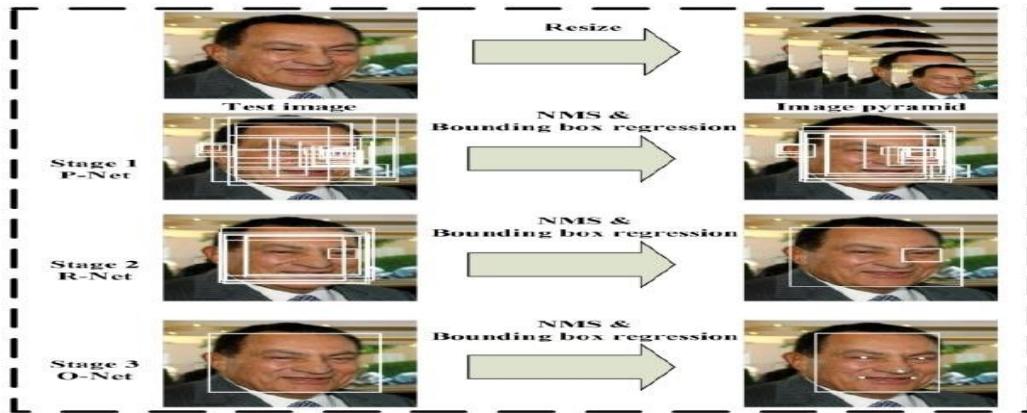
- Successfully Read input image from path **[Success]**
- Classify faces **[Success]**
- Recognize each person faces in images **[Success]**

A screenshot of a Python code editor showing the same face detection script. The image path has been changed to 'image\_00000000000000000000000000000001.jpg'. The rest of the code remains the same, including the face detection logic and the preview image.

## Technical Details:

### MTCNN

- System Architecture:



#### P-Net:

Input Layer: 12x12 kernels

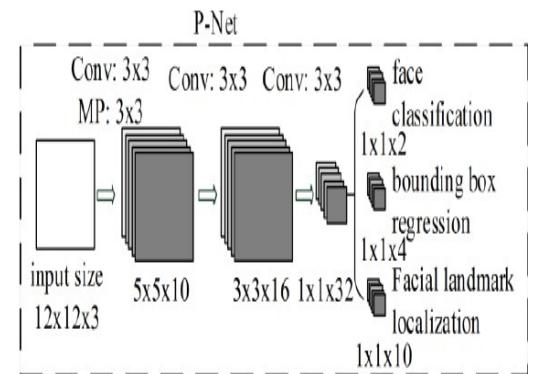
1- Convolution layer 10 kernel each of them is 3x3 using valid PreLU layer  
max pooling with using same

2- Convolution layer 16 kernel each of them is 3x3 using valid PreLU

3 Convolution layer 32 kernel each of them is 3x3 using valid. PreLU

The third layer activation is separate to convolution layers, and a softmax layer after one of those convolution layers (softmax assigns decimal probabilities to every result, and the probabilities add up to 1).

Probabilities: (face in the area or there **isn't** a face).



4- Convolution 4-1 outputs the probability of a face being in each bounding box, and convolution 4-2 outputs the coordinates of the bounding boxes.

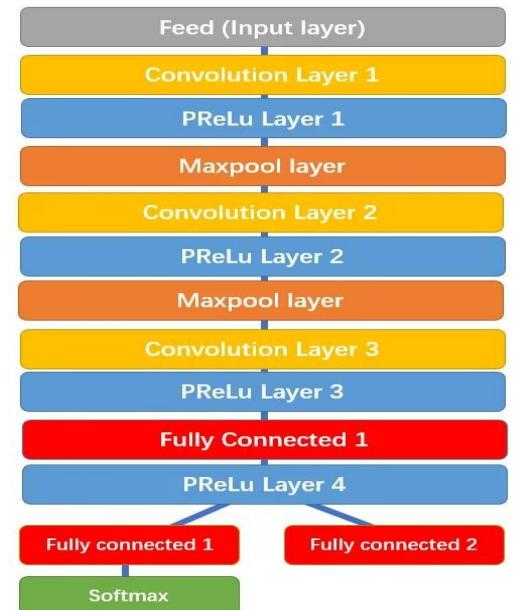
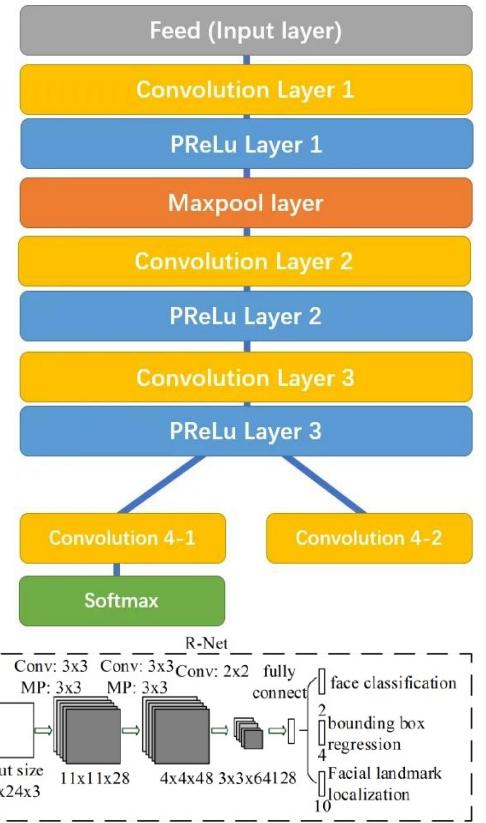
**There is no Dense Layer here [Fully connected Layer]**

### R-Net:

Input Layer : 24x24 kernels

- 1- Convolution layer 28 kernel each them is 3x3 using valid PreLU layer  
max pooling with using same
- 2- Convolution layer 48 kernel each them is 3x3 using valid PreLU  
max pooling with using valid
- 3- Convolution layer 64 kernel each them is 3x3 using valid PreLU  
Flatten  
fully connected layer with 128 dim  
PreLU
- 4- output 1 layer fully connected layer with 2 neurons and softmax for face classification  
output 2 layer fully connected layer with 4 neurons for bounding box regression

Similarly, R-Net splits into two layers in the end, giving out two outputs: the coordinates of the new bounding boxes and the machine's confidence in each bounding box



## O-Net:

input: reshape into (48, 48, 3)

1- Convolution layer 32 kernel each of them is 3x3 using valid  
PreLU layer  
max pooling with using same

2- Convolution layer 64 kernel each of them is 3x3 using valid  
PreLU  
max pooling with using valid.

3- Convolution layer 64 kernel each of them is 3x3 using valid  
PreLU  
max pooling with using same

4- Convolution layer 128 kernel each them is 3x3 using valid  
PreLU  
fully connected layer with 256 neurons  
PreLU

5- out1 : fully connected layer with 2 neurons and softmax for face classification

out2 : fully connected layer with 4 neurons for bounding box regression  
out3 : fully connected layer with 10 neurons for facial landmark localization 2 for 5 (locations of the eyes, nose, mouth)

### •Algorithm Explanation:

#### Training

We use three tasks to train CNN detectors: face/non-face classification, bounding box regression, and facial landmark localization.

1) Face classification: The **learning objective** is formulated as a two-class classification problem.

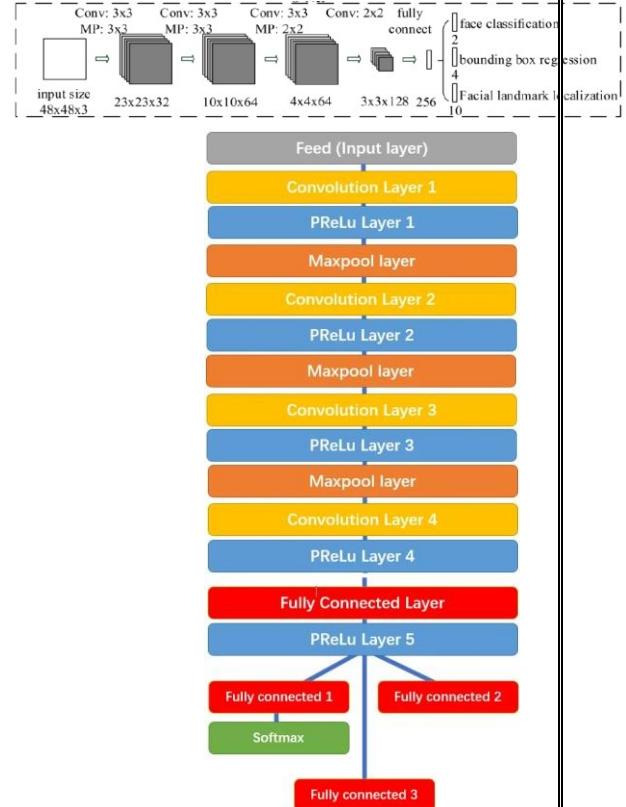
For each sample , we use the **cross-entropy loss**:

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i)))$$

where  $p_i$  is the probability produced by the network that indicates a sample being a face. The notation

$\hat{y}_i^{det} \in \{0,1\}$  denotes label of ground-truth.

2) Bounding box regression: For each window, we predict the offset between it and the nearest



ground truth .The learning objective is formulated as a regression problem, and we employ the Euclidean loss for each sample :

$$L_i^{box} = \|\hat{y}_i^{box} - y_i^{box}\|_2^2$$

where  $\hat{y}_i^{box}$  regression target obtained from the network and  $y_i^{box}$  is the ground-truth coordinate. There are four coordinates, including left top, height and width, and thus  $y_i^{box} \in \mathbb{R}^4$

3) Facial landmark localization: Similar to the bounding box regression task, facial landmark detection is formulated as a regression problem and we minimize the Euclidean loss:

$$L_i^{landmark} = \|\hat{y}_i^{landmark} - y_i^{landmark}\|_2^2$$

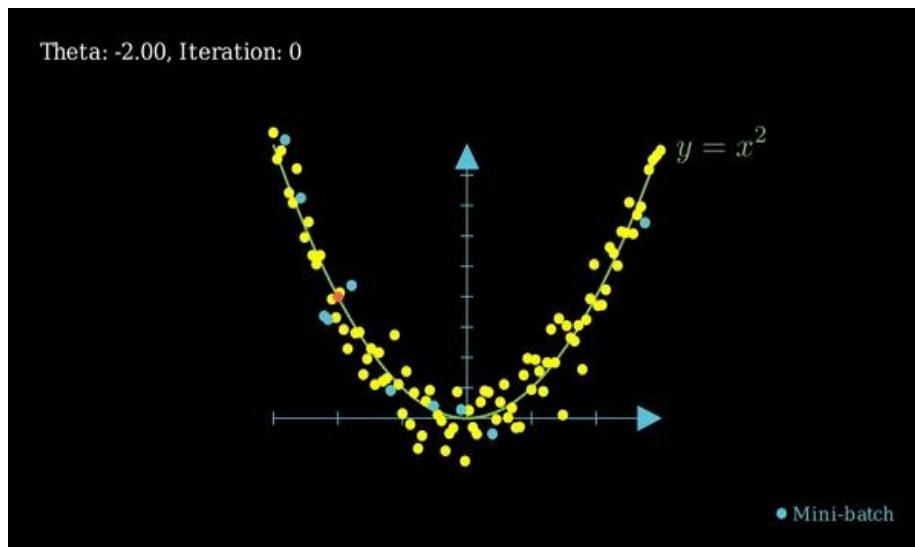
where  $\hat{y}_i^{landmark}$  is the facial landmark's coordinate obtained from the network and  $y_i^{landmark}$  is the ground-truth coordinate. There are five facial landmarks, including left eye, right eye,nose, left mouth corner, and right mouth corner, and thus  $y_i^{landmark} \in \mathbb{R}^{10}$

4) Multi-source training:

Since we employ different tasks in each CNNs, there are different types of training images in the learning process, such as face, non-face and partially alignedface. In this case, some of the loss functions ( Eq. (1)-(3) )) are not used. For example, for the sample of background region, we only compute  $L_i^{det}$ , and the other two losses are set as 0. This can be implemented directly with a sample type indicator. Then the overall learning target can be formulated as:

$$\min \sum_{i=1}^N \sum_{j \in \{det, box, landmark\}} \alpha_j \beta_i^j L_i^j$$

where is the number of training samples. denotes on the task importance. We use  $\alpha_{det}=1$ ,  $\alpha_{box}=0.5$ ,  $\alpha_{landmark}=0.5$  in P-Net and R-Net, ( $\alpha_{det}=1$ ,  $\alpha_{box}=0.5$  ,  $\alpha_{landmark}=1$ )in O-Net for more accurate facial landmarks localization .  $\beta_i^j$  is the sample type indicator. In this case, it is natural to employ stochastic gradient descent to train the CNNs.



- **Code Implementation:**

Welcome to Network Factory where mtcnn is being built.

**P-net:**

```
def build_pnet(self, input_shape=None):
    if input_shape is None:
        input_shape = (None, None, 3)

    p_inp = Input(input_shape)

    p_layer = Conv2D(10, kernel_size=(3, 3), strides=(1, 1), padding="valid")(p_inp)
    p_layer = PReLU(shared_axes=[1, 2])(p_layer)
    p_layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")(p_layer)

    p_layer = Conv2D(16, kernel_size=(3, 3), strides=(1, 1), padding="valid")(p_layer)
    p_layer = PReLU(shared_axes=[1, 2])(p_layer)

    p_layer = Conv2D(32, kernel_size=(3, 3), strides=(1, 1), padding="valid")(p_layer)
    p_layer = PReLU(shared_axes=[1, 2])(p_layer)

    p_layer_out1 = Conv2D(2, kernel_size=(1, 1), strides=(1, 1))(p_layer)
    p_layer_out1 = Softmax(axis=3)(p_layer_out1)

    p_layer_out2 = Conv2D(4, kernel_size=(1, 1), strides=(1, 1))(p_layer)

    p_net = Model(p_inp, [p_layer_out2, p_layer_out1])

    return p_net
```

**R-Net:**

```
def build_rnet(self, input_shape=None):
    if input_shape is None:
        input_shape = (24, 24, 3)

    r_inp = Input(input_shape)

    r_layer = Conv2D(28, kernel_size=(3, 3), strides=(1, 1), padding="valid")(r_inp)
    r_layer = PReLU(shared_axes=[1, 2])(r_layer)
    r_layer = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="same")(r_layer)

    r_layer = Conv2D(48, kernel_size=(3, 3), strides=(1, 1), padding="valid")(r_layer)
    r_layer = PReLU(shared_axes=[1, 2])(r_layer)
    r_layer = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="valid")(r_layer)

    r_layer = Conv2D(64, kernel_size=(2, 2), strides=(1, 1), padding="valid")(r_layer)
    r_layer = PReLU(shared_axes=[1, 2])(r_layer)
    r_layer = Flatten()(r_layer)
    r_layer = Dense(128)(r_layer)
    r_layer = PReLU()(r_layer)

    r_layer_out1 = Dense(2)(r_layer)
    r_layer_out1 = Softmax(axis=1)(r_layer_out1)

    r_layer_out2 = Dense(4)(r_layer)

    r_net = Model(r_inp, [r_layer_out2, r_layer_out1])

    return r_net
```

## O-Net:

```
def build_onet(self, input_shape=None):
    if input_shape is None:
        input_shape = (48, 48, 3)

    o_inp = Input(input_shape)
    o_layer = Conv2D(32, kernel_size=(3, 3), strides=(1, 1), padding="valid")(o_inp)
    o_layer = PReLU(shared_axes=[1, 2])(o_layer)
    o_layer = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="same")(o_layer)

    o_layer = Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding="valid")(o_layer)
    o_layer = PReLU(shared_axes=[1, 2])(o_layer)
    o_layer = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="valid")(o_layer)

    o_layer = Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding="valid")(o_layer)
    o_layer = PReLU(shared_axes=[1, 2])(o_layer)
    o_layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")(o_layer)

    o_layer = Conv2D(128, kernel_size=(2, 2), strides=(1, 1), padding="valid")(o_layer)
    o_layer = PReLU(shared_axes=[1, 2])(o_layer)

    o_layer = Flatten()(o_layer)
    o_layer = Dense(256)(o_layer)
    o_layer = PReLU()(o_layer)

    o_layer_out1 = Dense(2)(o_layer)
    o_layer_out1 = Softmax(axis=1)(o_layer_out1)
    o_layer_out2 = Dense(4)(o_layer)
    o_layer_out3 = Dense(10)(o_layer)

    o_net = Model(o_inp, [o_layer_out2, o_layer_out3, o_layer_out1])
    return o_net
```

## The Trigger to build it is in MTCNN CLASS:

```
class MTCNN(object):
    """
    Allows to perform MTCNN Detection ->
        a) Detection of faces (with the confidence probability)
        b) Detection of keypoints (left eye, right eye, nose, mouth_left, mouth_right)
    """

    def __init__(self, weights_file: str = None, min_face_size: int = 20, steps_threshold: list = None,
                 scale_factor: float = 0.709):
        """
        Initializes the MTCNN.
        :param weights_file: file uri with the weights of the P, R and O networks from MTCNN. By default it will load
        the ones bundled with the package.
        :param min_face_size: minimum size of the face to detect
        :param steps_threshold: step's thresholds values
        :param scale_factor: scale factor
        """
        if steps_threshold is None:
            steps_threshold = [0.6, 0.7, 0.7]

        if weights_file is None:
            weights_file = pkg_resources.resource_stream('mtcnn', 'data/mtcnn_weights.npy')

        self._min_face_size = min_face_size
        self._steps_threshold = steps_threshold
        self._scale_factor = scale_factor

        self._pnet, self._rnet, self._onet = NetworkFactory().build_P_R_O_nets_from_file(weights_file)
```

```
def build_P_R_O_nets_from_file(self, weights_file):
    weights = np.load(weights_file, allow_pickle=True).tolist()

    p_net = self.build_pnet()
    r_net = self.build_rnet()
    o_net = self.build_onet()

    p_net.set_weights(weights[ 'pnet' ])
    r_net.set_weights(weights[ 'rnet' ])
    o_net.set_weights(weights[ 'onet' ])

    return p_net, r_net, o_net
```

## **HOG(Histogram of Oriented Gradients):**

The feature descriptor defines an image by capturing color intensity and outlining edges using first-order directive kernels. It calculates gradient magnitude, determines orientation, and creates a histogram with bins storing gradient magnitudes based on orientation angles.

## **Steps to perform HOG:**

- Gamma/Color Normalization (Optional)
- Gradient Computation
- Spatial/Orientation Binning (Dividing the image into cells).
- Block Normalization.
- Get the HOG Feature Vector.

### **Step 1: Gamma/Color Normalization (Optional):**

Gamma normalization adjusts illuminance or color intensities using a power-law transformation. Experimentation with color spaces (e.g., RGB, LAB) shows minimal impact on HOG descriptors, especially with block normalization. This step is optional; you can skip it or test effects during free time. Gamma correction is demonstrated below.

```
S = C.R^(gamma)
To encode gamma,
gamma= 1/gamma
To decode gamma,
gamma= gamma

C = color intensity
R = (Image_input)/C
```

### **Step 2: Gradient Computation:**

Compute image gradients using a simple 1-D [-1,0,1] mask for optimal results in detecting changes in light intensity.

Forward Difference:  $f'(x) = \frac{f(x + h) - f(x)}{h}$

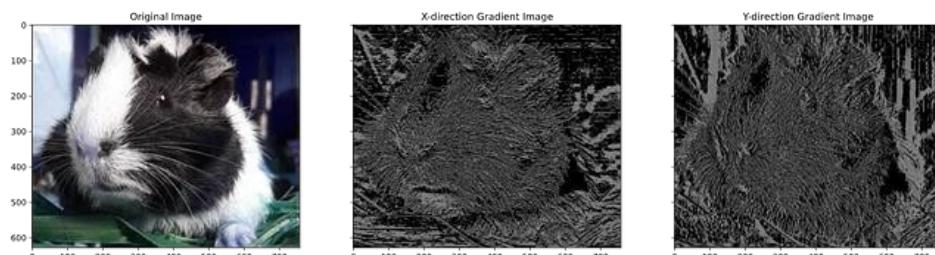
Backward Difference:  $f'(x) = \frac{f(x) - f(x - h)}{h}$

Central Difference:  $f'(x) = \frac{f(x + h) - f(x - h)}{2h}$

**Magnitude :**  $||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

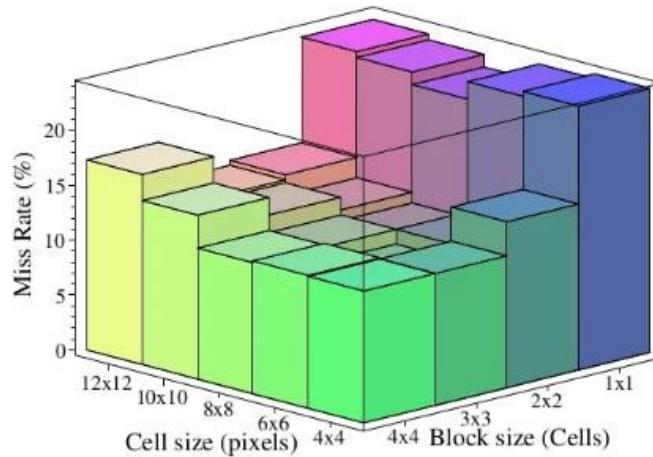
**Direction :**  $\theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$

The horizontal gradient detects horizontal edges, and the vertical gradient detects upright position edges. In a colored image, gradient values are sharper than in grayscale. Square root transformation is applied to both horizontal and vertical gradient magnitudes for histogram creation. Simultaneously, orientation magnitudes are obtained using the arctangent of Y and X values.



### Step 3: Spatial/Orientation Binning (Dividing the image into cells)

Scan the image in  $63 \times 63$  blocks with a  $2 \times 2$  grid of  $8 \times 8$  pixels in each, resulting in  $8 \times 8$  gradients and orientations. Each cell contains 128 values. Customize grid cells, blocks, and orientation bins (default is  $8 \times 8$  cells with 4 grids per block). Increasing orientation bins up to 9 improves performance. Choose 'signed' (0–180) or 'unsigned' (0–360) degree values for histograms. For human detection, use  $3 \times 3$  cell blocks with  $6 \times 6$ -pixel cells for optimal results.



#### Step 4: Block Normalization:

To address variations in gradient values due to illuminance differences, normalize cells using block normalization. Apply this to each 2 x 2 grid, choosing from four schemes. L2-Hys, L2-norm, and L1-sqrt perform equally well, while L1-norm reduces performance by 5%.

- Normalization

$$\frac{2}{\sqrt{(2)^2 + (4)^2}} = \frac{2}{\sqrt{20}} = \frac{2}{4.47} = 0.447$$

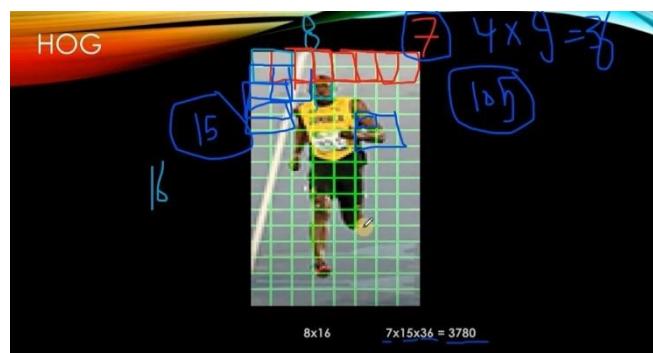
$$\frac{2}{\sqrt{20}} \cdot \frac{4}{\sqrt{20}} = \frac{2}{4.47} \cdot \frac{4}{4.47} = 0.447 \cdot 0.894$$

$$\frac{4}{\sqrt{(2)^2 + (4)^2}} = \frac{4}{\sqrt{20}} = \frac{4}{4.47} = 0.894$$

$$\frac{4}{\sqrt{20}} \cdot \frac{8}{\sqrt{20}} = \frac{4}{4.47} \cdot \frac{8}{4.47} = 0.447 \cdot 0.894$$

#### Step 5: Get the HOG Feature Vector:

After block normalization, obtain the feature vector. With 9 orientation bins, each 8x8 grid generates 9 feature vectors. In a 4-grid block, you get 36 feature vectors, resulting in a 36 x 1 vector for a 16 x 16 block. For a 512 x 512 image with 63 x 63 blocks, the final feature vector is 142,884-dimensional. This is smaller than the 786,432 features in a colored image without HOG descriptors, making the HOG feature vector construction and visualization a multistep process.



## Pre

```
import cv2
import copy
from faker import Faker
import tkinter as tk
import os
from mtcnn import MTCNN
import face_recognition
```

```
def on_close(event):
    window.destroy()
```

```
def handle_input(event):
    # Check if the input is '0' and close the window if true
    if event.char == '0':
        window.destroy()
```

```
class person:
```

```
class Person:  
    def __init__(self,face_encode,name=None):  
        # result = value_if_true if condition else value_if_false  
        self.name = Faker().name if name is None else name  
        self.mainface = face_encode  
  
    def showMeInIMGs(self,connections):  
        print("Person : "+str(self.name)+"\n")  
        for connection in connections:  
            if connection.Person == self:  
                connection.ImageObj.selectface(connection.facebox,str(self.name))
```

```
class image:
```

```
def detect_faces(self):  
    detector = MTCNN(steps_threshold=[0.8, 0.9, 0.9])  
    return detector.detect_faces(self.image)
```

```
def detect_show(self, mode):
    # mode 1 = MTCNN
    # mode 2 = face_recognition
    tmp = self.getimage_copy()
    tmp = cv2.cvtColor(tmp, cv2.COLOR_BGR2RGB)
    if mode == 1:
        for left, top, rightP, bottomP in self.boxes_mtcnn:
            cv2.rectangle(tmp, (left, top), (left+rightP, top+bottomP), (0, 255, 0), 2)
    elif mode == 2:
        for top, right, bottom, left in self.boxes:
            cv2.rectangle(tmp, (left, top), (right, bottom), (0, 255, 0), 2)
    cv2.imshow('Detected Faces', tmp)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
def getimage_copy(self):
    return copy.deepcopy(self.image)
```

```
def convertMTCNNT2FRT(self, boxes):
    myboxs = []
    for left, top, rightP, bottomP in boxes:
        # face_recognition : top, right, bottom, left
        # MTCNN : left top RightP(width) bottomP(height)
        right = left+rightP
        bottom = top+bottomP
        myboxs.append((top, right, bottom, left))
    return myboxs
```

```
def getfaceinpic(self, box, index=0):
    tmp = self.getimage_copy()
    tmp = cv2.cvtColor(tmp, cv2.COLOR_BGR2RGB)
    top, right, bottom, left = box
    cv2.rectangle(tmp, (left, top), (right, bottom), (0, 255, 0), 2)
    cv2.imshow('Detected Face '+str(index), tmp)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```

self.image_path = image_path
self.image = face_recognition.load_image_file(dir+'/'+image_path)
# FOR PRESENTATION
#self.image=cv2.resize(self.image, (800,600))
self.faces = self.detect_faces()
self.boxes_mtcnn = [tuple(d['box']) for d in self.faces]
self.boxes = self.convertMTCNN2FRT(self.boxes_mtcnn)
print("I: I am going to calculate Encoding , it will take time")
self.encoding = face_recognition.face_encodings(self.image, self.boxes, num_jitters=100)
if self.encoding:
    print("I: Done PRO")

```

```

def selectface(self,face_boxes,person_name):
    tmp = self.getimage_copy()
    tmp = cv2.cvtColor(tmp, cv2.COLOR_BGR2RGB)
    top, right, bottom, left = face_boxes
    cv2.rectangle(tmp, (left, top), (right, bottom), (0, 255, 0), 2)
    cv2.imshow(person_name, tmp)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

```

## class connection:

```

class Person_Image_Faces:
    def __init__(self,Person,ImageObj,Person_box):
        self.Person = Person
        self.ImageObj = ImageObj
        self.facebox = Person_box

```

## Main:

```
dir = "president"
listImage = get_image_names(dir)

imgList = []
for filename in listImage:
    imgList.append(img_obj(filename))

for imgObj in imgList:
    imgObj.detect_show(mode=2)

PersonList = []
ConnectioList = []

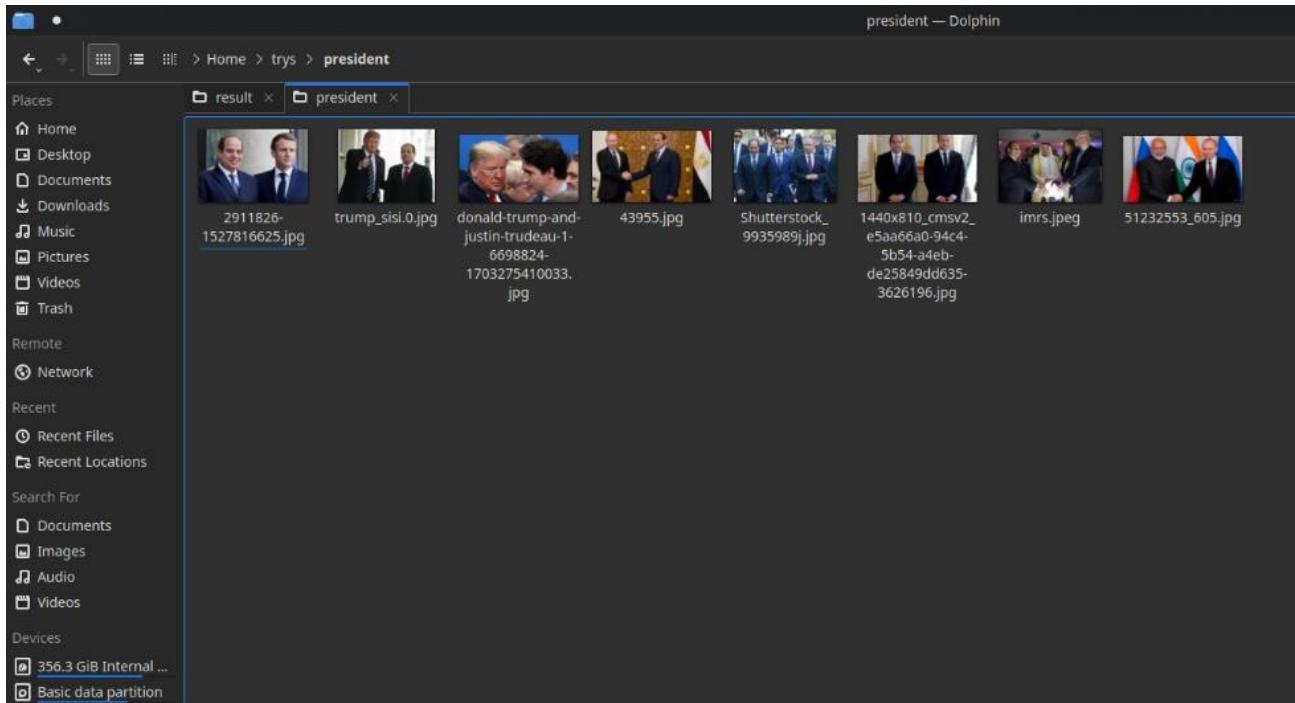
for i in range(len(imgList[0].encodes)):
    PersonList.append(Person(face_encode=imgList[0].encodes[i]))
    ConnectioList.append(Person_Image_Faces(PersonList[-1],imgList[0],Person_box=imgList[0].boxs[i]))

if len(imgList) > 1 :
    for imgObj in imgList[1:]:
        for encode, box in zip(imgObj.encodes, imgObj.boxs):
            flag = False
            for person in PersonList:
                res = face_recognition.compare_faces([person.mainface], encode,tolerance=0.5)[0]
                if res:
                    ConnectioList.append(Person_Image_Faces(person,imgObj,Person_box=box))
                    flag = True
                    break
            if not flag:
                PersonList.append(Person(face_encode=encode))
                ConnectioList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))

counter = 0
for person in PersonList:
    counter = counter +1
    window = tk.Tk()
    window.title("Person Images")
    message = """ Peson """ + str(counter)
    label = tk.Label(window, text=message, font=("Arial", 12))
    label.pack(padx=10, pady=10)
    window.protocol("WM_DELETE_WINDOW", on_close)
    window.bind('<Key>', handle_input)
    window.mainloop()
    person.showMeInIMGs(ConnectioList)
```

# Result

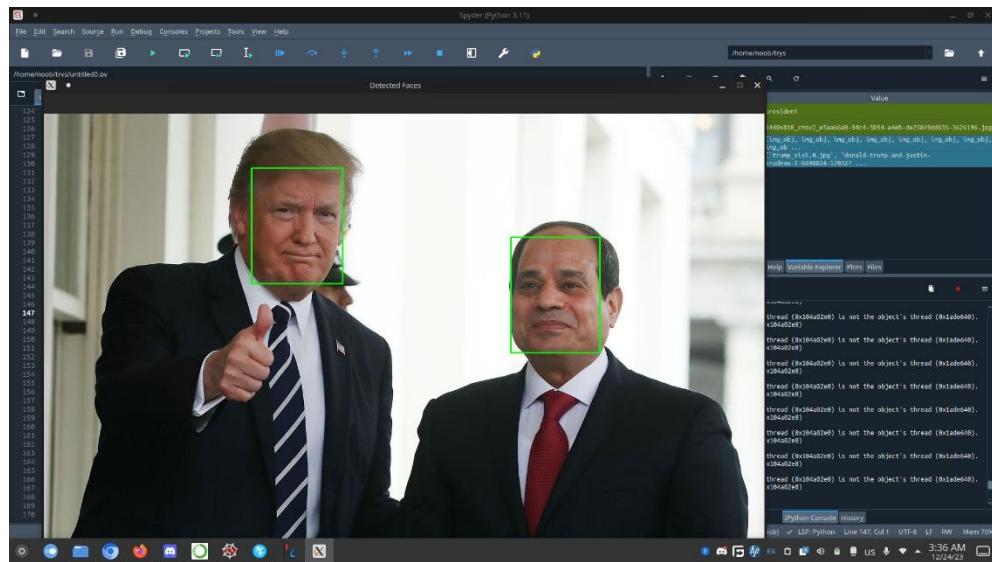
## Input images:



- Images have many people for each image.

## Classification-Result:

- Check the image and predict image has a person or not if predicted person draws a square for his face using a box coordinate generated by O-NET based on confidence.



Spyder (Python 3.11)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/noob/trys
untitled0.py mtcnn.py factory.py untitled1.py mm.py untitled2.py untitled4.py*
124     cv2.imshow(person_name, tmp)
125     cv2.waitKey(0)
126     cv2.destroyAllWindows()
127
128
129
130
131
132
133 class Person_Image_Faces:
134     def __init__(self,Person,ImageObj,Person_box):
135         self.Person = Person
136         self.ImageObj = ImageObj
137         self.Person_box = Person_box
138
139     dir = "president"
140     listImage = []
141     for filename in listImage:
142         imgList.append(imgObj)(filename)
143
144     for imgObj in imgList:
145         imgObj.detect_show(mode=2)
146
147     PersonList = []
148     ConnecttoList = []
149
150     for t in range(len(imgList[0].encodes)):
151         PersonList.append(Person(face_encode=imgList[0].encodes[t]))
152         ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0],Person_box=ImgList[0].Person_box))
153
154     if len(imgList) > 1 :
155         for imgObj in imgList[1:]:
156             for encode, box in zip(imgObj.encodes, imgObj.boxes):
157                 flag = False
158                 for person in PersonList:
159                     res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)
160                     if res:
161                         ConnecttoList.append(Person_Image_Faces(person,imgObj),Person_box=box)
162                         flag = True
163                         break
164                 if not flag:
165                     PersonList.append(Person(face_encode=encode))
166                     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgObj),Person_box=box)
167
168
169
170
171
172
173
174
175
176

```

Detected Faces

Value

Name	Type	Size	Value
dir	str	9	president
filename	str	63	1440x810_csv2_e1aa6a0-94c4-1b5d-a4eb-de2158499d635-3626196.jpg
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
listImage	list	8	[trump-stl1.0.jpg, 'donald-trump-and-justin-trudeau-1-6598824-179327...', ...]

RW Mem 70%

3:36 AM 12/24/23

Spyder (Python 3.11)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/noob/trys
untitled0.py mtcnn.py factory.py untitled1.py mm.py untitled2.py untitled4.py*
124     cv2.imshow(person_name, tmp)
125     cv2.waitKey(0)
126     cv2.destroyAllWindows()
127
128
129
130
131
132
133 class Person_Image_Faces:
134     def __init__(self,Person,ImageObj,Person_box):
135         self.Person = Person
136         self.ImageObj = ImageObj
137         self.Person_box = Person_box
138
139     dir = "president"
140     listImage = get_image_names(dir)
141
142     for filename in listImage:
143         imgList.append(imgObj)(filename)
144
145
146     for imgObj in imgList:
147         imgObj.detect_show(mode=2)
148
149     PersonList = []
150     ConnecttoList = []
151
152     for t in range(len(imgList[0].encodes)):
153         PersonList.append(Person(face_encode=imgList[0].encodes[t]))
154         ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0],Person_box=ImgList[0].Person_box))
155
156     if len(imgList) > 1 :
157         for imgObj in imgList[1:]:
158             for encode, box in zip(imgObj.encodes, imgObj.boxes):
159                 flag = False
160                 for person in PersonList:
161                     res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)
162                     if res:
163                         ConnecttoList.append(Person_Image_Faces(person,imgObj),Person_box=box)
164                         flag = True
165                         break
166                 if not flag:
167                     PersonList.append(Person(face_encode=encode))
168                     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgObj),Person_box=box)
169
170
171
172
173
174
175
176

```

Detected Faces

Value

Name	Type	Size	Value
dir	str	9	president
filename	str	63	1440x810_csv2_e1aa6a0-94c4-1b5d-a4eb-de2158499d635-3626196.jpg
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
listImage	list	8	[trump-stl1.0.jpg, 'donald-trump-and-justin-trudeau-1-6598824-179327...', ...]

Help Variable Explorer Plots Files

: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
thread (0x104a02e0)  
Object::newToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).  
Cannot move to target thread (0x1ade640)

IPython Console History

conda: noob (Python 3.11.5) Completions: conda(noob) LSP: Python Line 147, Col 1 UTF-8 LF RW US 3:36 AM 12/24/23

Spyder (Python 3.11)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/noob/trys
untitled0.py mtcnn.py factory.py untitled1.py mm.py untitled2.py untitled4.py*
124 cv2.imshow(person_name, tmp)
125 cv2.waitKey(0)
126 cv2.destroyAllWindows()
127
128
129
130
131
132
133 class Person_Image_Faces:
134     def __init__(self,Person,ImageObj,Person_box):
135         self.Person = Person
136         self.ImageObj = ImageObj
137         self.facebox = Person_box
138
139 dir = "president"
140 listImage = []
141 for filename in listImage:
142     imgList.append(imgObj)(filename)
143
144 for imgObj in imgList:
145     imgObj.detect_show(mode=2)
146
147 PersonList = []
ConnectoList = []
148
149 for l in range(len(imgList[0].encodes)):
150     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
151     ConnectoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_b
152
153 if len(imgList) > 1 :
154     for imgObj in imgList[1:]:
155         for encode, box in zip(imgObj.encodes, imgObj.boxes):
156             flag = False
157             for person in PersonList:
158                 res = face_recognition.compare_faces([person.mainface], encode
159                 if res:
160                     ConnectoList.append(Person_Image_Faces(person,imgObj),Pers
161                     flag = True
162                     break
163             if not flag:
164                 PersonList.append(Person(face_encode=encode))
165                 ConnectoList.append(Person_Image_Faces(PersonList[-1],imgObj),
166
167
168
169
170

```

Detected Faces

3:37 AM 12/24/23

Spyder (Python 3.11)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/noob/trys
untitled0.py mtcnn.py factory.py untitled1.py mm.py untitled2.py untitled4.py*
124 cv2.imshow(person_name, tmp)
125 cv2.waitKey(0)
126 cv2.destroyAllWindows()
127
128
129
130
131
132
133 class Person_Image_Faces:
134     def __init__(self,Person,ImageObj,Person_box):
135         self.Person = Person
136         self.ImageObj = ImageObj
137         self.facebox = Person_box
138
139 dir = "president"
140 listImage = []
141 for filename in listImage:
142     imgList.append(imgObj)(filename)
143
144 for imgObj in imgList:
145     imgObj.detect_show(mode=2)
146
147 PersonList = []
ConnectoList = []
148
149 for l in range(len(imgList[0].encodes)):
150     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
151     ConnectoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_b
152
153 if len(imgList) > 1 :
154     for imgObj in imgList[1:]:
155         for encode, box in zip(imgObj.encodes, imgObj.boxes):
156             flag = False
157             for person in PersonList:
158                 res = face_recognition.compare_faces([person.mainface], encode
159                 if res:
160                     ConnectoList.append(Person_Image_Faces(person,imgObj),Pers
161                     flag = True
162                     break
163             if not flag:
164                 PersonList.append(Person(face_encode=encode))
165                 ConnectoList.append(Person_Image_Faces(PersonList[-1],imgObj),
166
167
168
169
170

```

Detected Faces

Help Variable Explorer Plots Files

Console 3/6/A

```

Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)

```

Python Console History

conda: noob (Python 3.11.5) ✓ Completions: conda(noob) ✓ LSP: Python Line 147, Col 1 UTF-8 LF RW Mem 70%

3:37 AM 12/24/23

Spyder (Python 3.11)

/home/noob/trys

untitled0.py mtcnn.py factory.py untitled1.py mm.py untitled2.py untitled4.py\*

```

124     cv2.imshow(person_name, tmp)
125     cv2.waitKey(0)
126     cv2.destroyAllWindows()
127
128
129
130
131
132
133 class Person_Image_Faces:
134     def __init__(self,Person,ImageObj,Person_box):
135         self.Person = Person
136         self.ImageObj = ImageObj
137         self.facebox = Person_box
138
139     dir = "president"
140     listImage = []
141
142     for filename in listImage:
143         imgList.append(img_obj)(filename)
144
145
146     for imgObj in imgList:
147         imgObj.detect_show(mode=2)
148
149
150
151     PersonList = []
152     ConnectoList = []
153
154     for l in range(len(imgList[0].encodes)):
155         PersonList.append(Person(face_encode=tngList[0].encodes[l]))
156         ConnectoList.append(Person_Image_Faces(PersonList[-1],imgObj))
157
158     if len(imgList) > 1 :
159         for imgObj in imgList[1:]:
160             for encode, box in zip(imgObj.encodes, imgObj.boxes):
161                 flag = False
162                 for person in PersonList:
163                     res = face_recognition.compare_faces([person.mainface], encode,tolerance=0.5)[0]
164                     if res:
165                         ConnectoList.append(Person_Image_Faces(person,imgObj,Person_box=box))
166                         flag = True
167                         break
168                 if not flag:
169                     PersonList.append(Person(face_encode=encode))
170                     ConnectoList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))

```

Detected Faces

Name Type Size Value

dir	str	9	president
filename	str	63	1440x810_msv2_eaa6a0-94c4-1b5d-adeb-de21949d9635-3626196.jpg
list	list	8	[img_obj, img_obj, img_obj, img_obj, img_obj, img_obj, img_obj, img_obj]
image	list	8	['trump.sist.6.jpg', 'donald-trump-and-justin-trudeau-1-6598824-178327...', ...]

Console 36/A

```

::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
move to target thread (0x1ade640).

::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
move to target thread (0x1ade640).

::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
move to target thread (0x1ade640).

::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
move to target thread (0x1ade640).

::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x1ade640).

Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x1ade640).

Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x1ade640).

Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x1ade640).

```

conda: noob (Python 3.11.5) ✓ Completions: condainnoob ✓ LSP: Python Line 147, Col 1 UTF-8 LF RW Mem 69%

3:37 AM 12/24/23

Spyder (Python 3.11)

/home/noob/trys

untitled0.py mtcnn.py factory.py untitled1.py

Detected Faces

File Edit Search Source Run Debug Consoles Projects Tools View Help

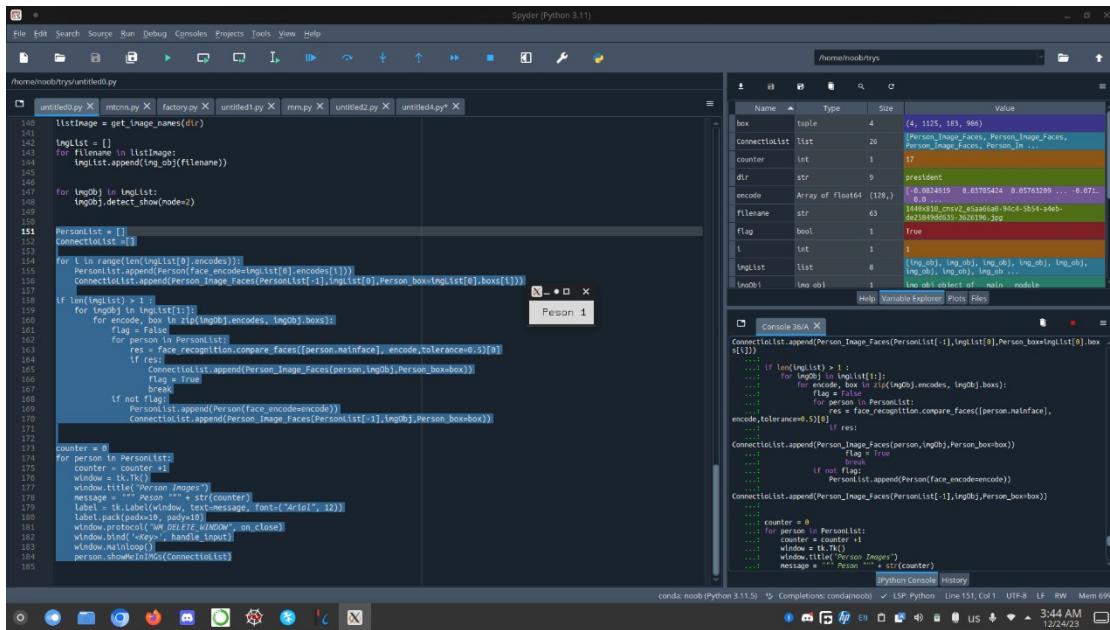
File Edit Search Source Run Debug Consoles Projects Tools View Help

3:37 AM 12/24/23

## Recognition:

- Get person's picture where her appeared in it:

### 1. Person one:



The screenshot shows the Spyder Python 3.11 IDE interface. The code editor displays a script named `untitled0.py` containing Python code for face recognition. The variable explorer on the right shows various variables and their values, including `box`, `ConnectoList`, `counter`, `dlt`, `encode`, `filename`, `Flag`, `l`, `ImgList`, and `imgObj`. The console window at the bottom shows the execution of the code.

```
listImage = get_image_names(dlt)
imgList = []
for filename in listImage:
    imgList.append(imgObj(filename))

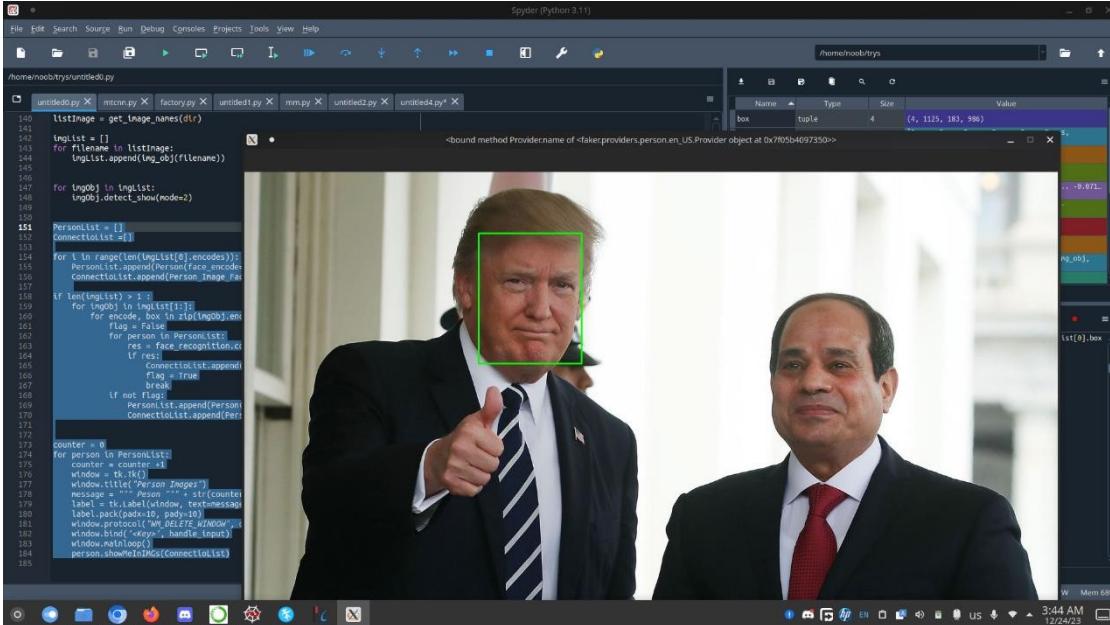
for imgObj in imgList:
    imgObj.detect_show(mode=2)

PersonList = []
ConnectoList = []

for i in range(len(imgList[0].encodes)):
    PersonList.append(Person_Image_Faces(imgList[0].encodes[i]))
    ConnectoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box,imgList[0].boxs[i]))

if len(ImgList) > 1:
    for imgObj in ImgList[1:]:
        for encode, box in zip(imgObj.encodes, imgObj.boxs):
            flag = False
            for person in PersonList:
                res = face_recognition.compare_faces([person.natface], encode, tolerance=0.5)
                if res:
                    ConnectoList.append(Person_Image_Faces(person, imgObj, Person_box))
                    flag = True
                    break
                if not flag:
                    PersonList.append(Person(face.encode=encode))
                    ConnectoList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))

counter = 0
for person in PersonList:
    counter += 1
    window = tk.Tk()
    window.title("Person Images")
    message = str(counter)
    label = tk.Label(window, text=message, font="Arial", 12)
    label.pack(pady=10)
    window.protocol("WM_DELETE_WINDOW", on_close)
    window.bind("<Key>", handle_input)
    window.mainloop()
    person.showInIMGS(ConnectoList)
```



The screenshot shows the Spyder Python 3.11 IDE interface again, but this time it displays a comparison image between two men. A green bounding box highlights the face of Donald Trump, indicating a successful match or detection.

```
listImage = get_image_names(dlt)
imgList = []
for filename in listImage:
    imgList.append(imgObj(filename))

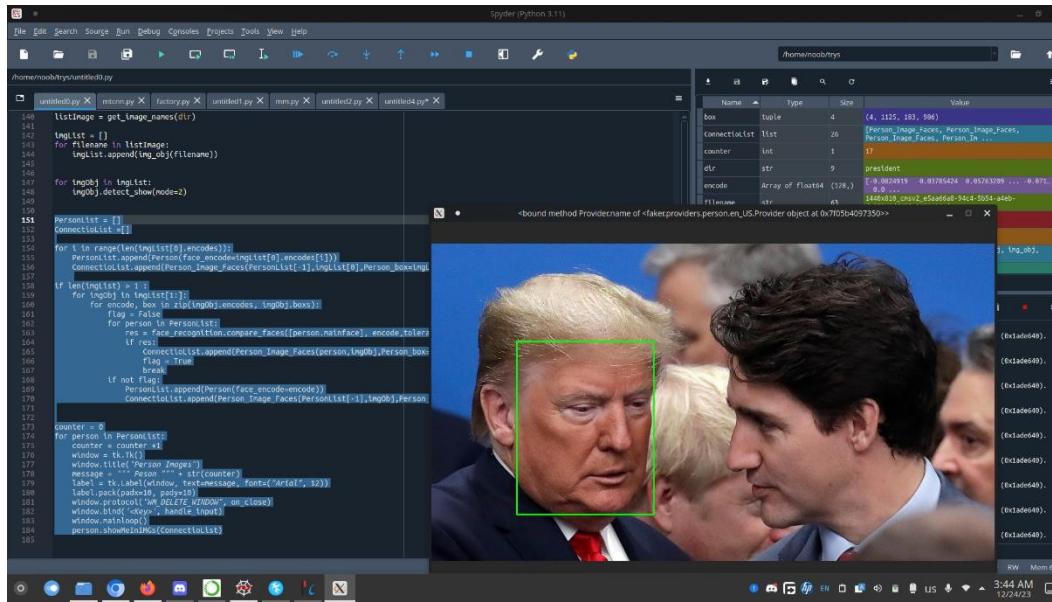
for imgObj in imgList:
    imgObj.detect_show(mode=2)

PersonList = []
ConnectoList = []

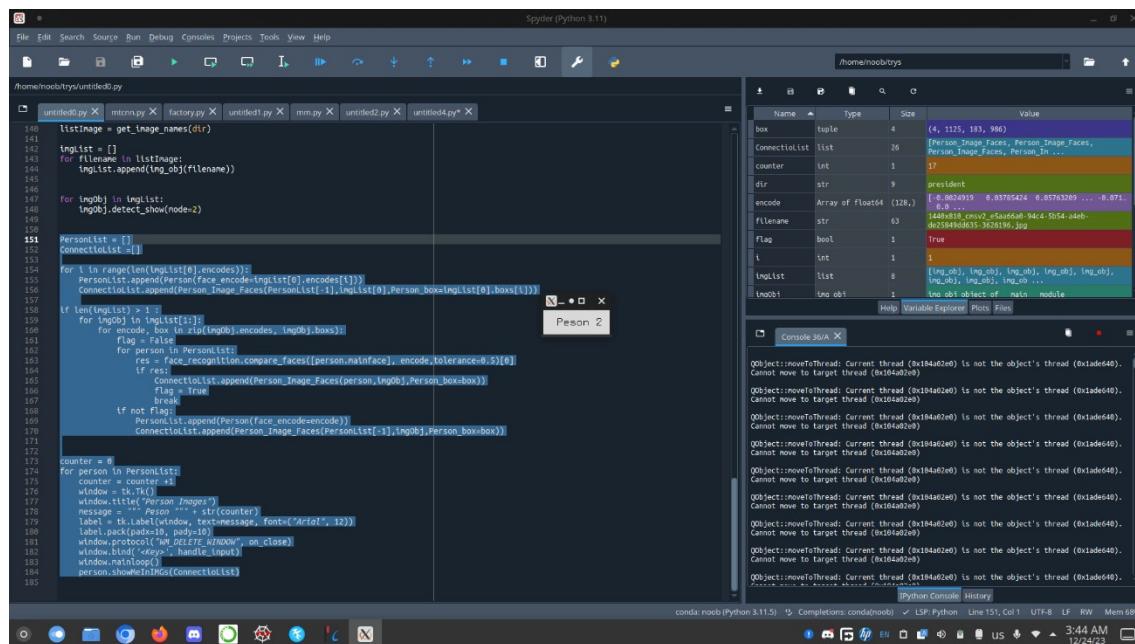
for i in range(len(imgList[0].encodes)):
    PersonList.append(Person_Image_Faces(imgList[0].encodes[i]))
    ConnectoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box,imgList[0].boxs[i]))

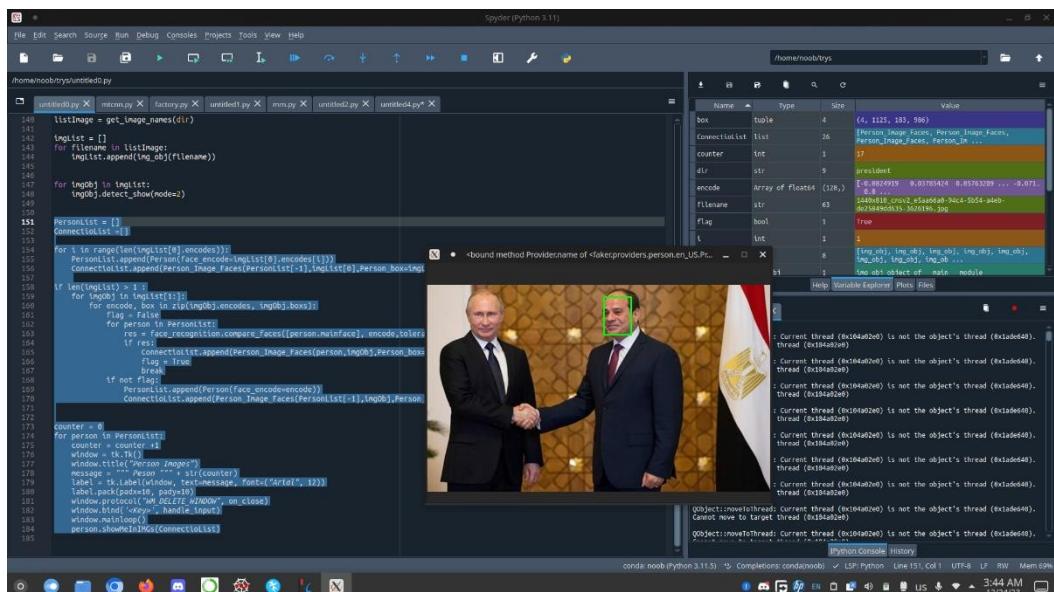
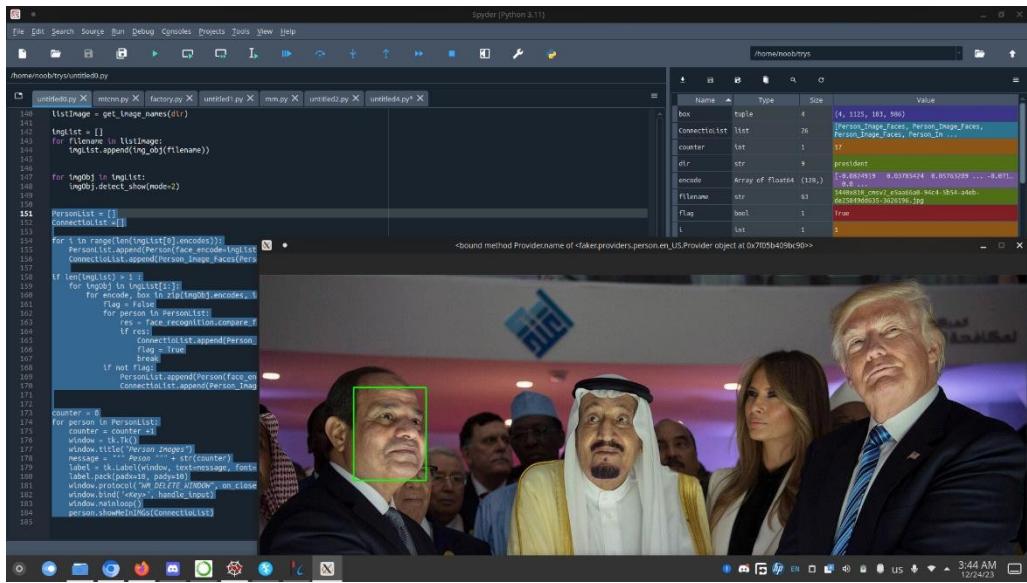
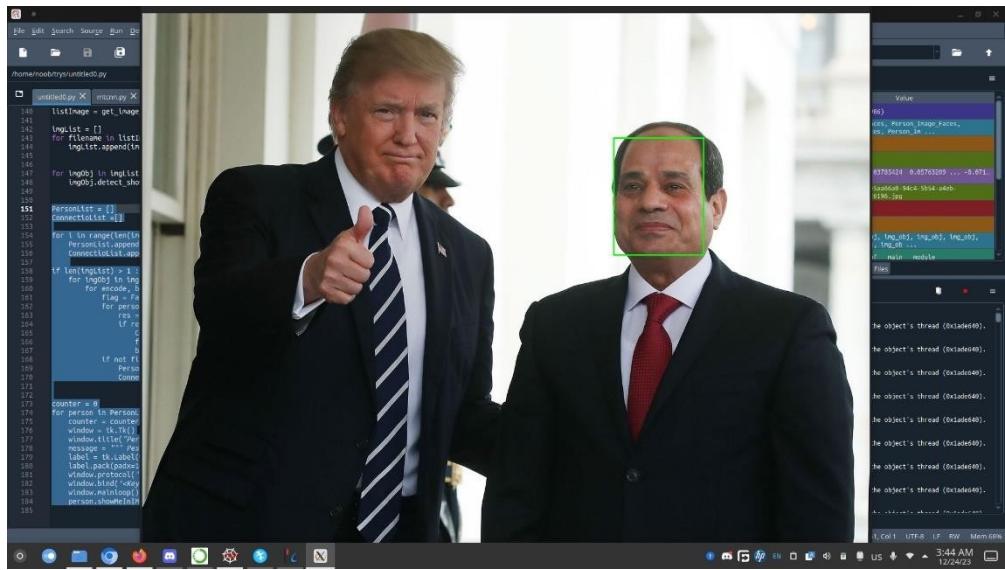
if len(ImgList) > 1:
    for imgObj in ImgList[1:]:
        for encode, box in zip(imgObj.encodes, imgObj.boxs):
            flag = False
            for person in PersonList:
                res = face_recognition.compare_faces([person.natface], encode, tolerance=0.5)
                if res:
                    ConnectoList.append(Person_Image_Faces(person, imgObj, Person_box))
                    flag = True
                    break
                if not flag:
                    PersonList.append(Person(face.encode=encode))
                    ConnectoList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))

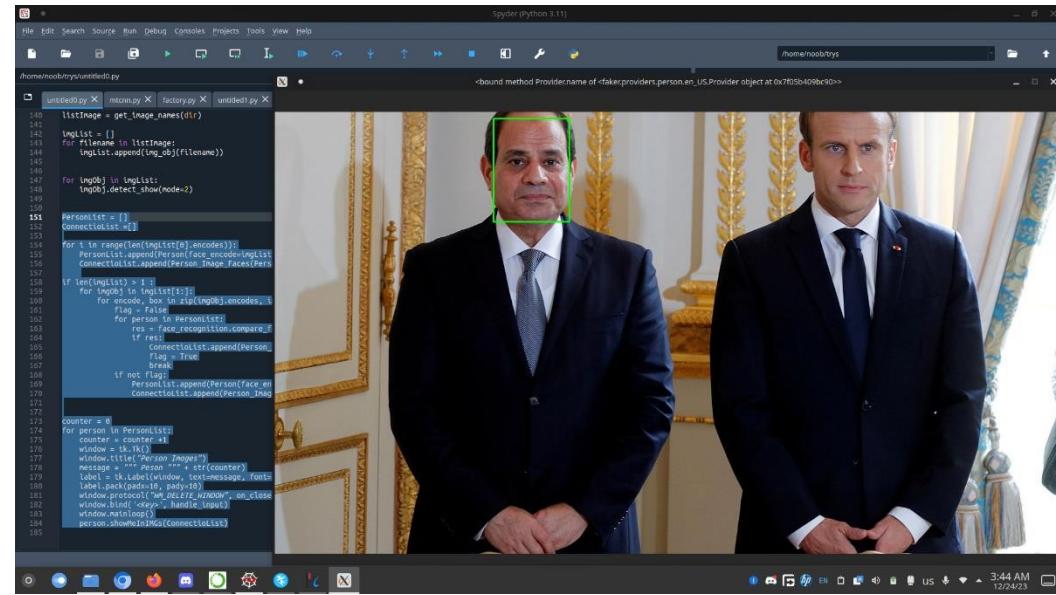
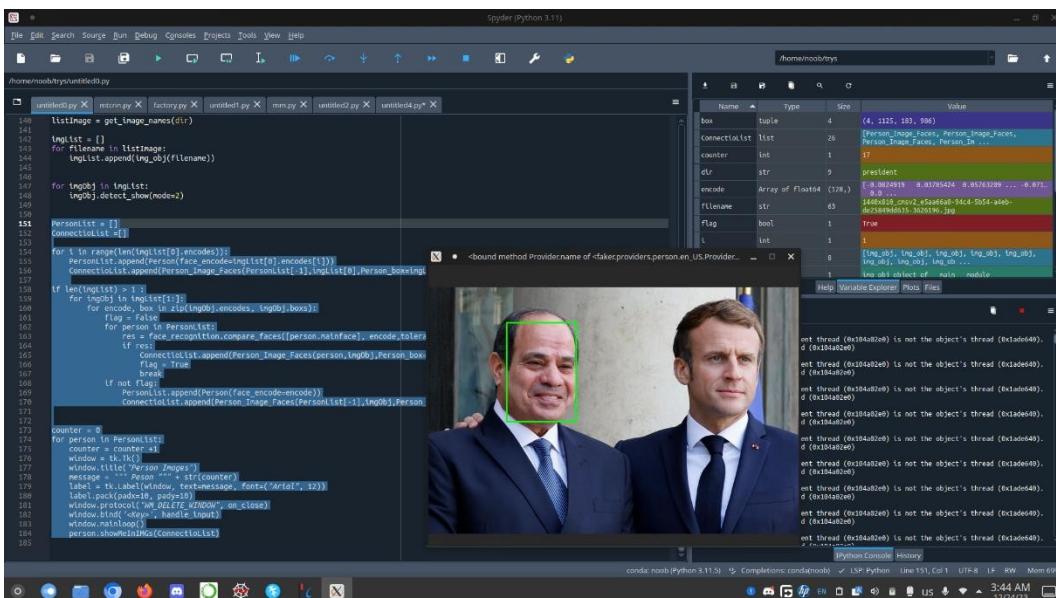
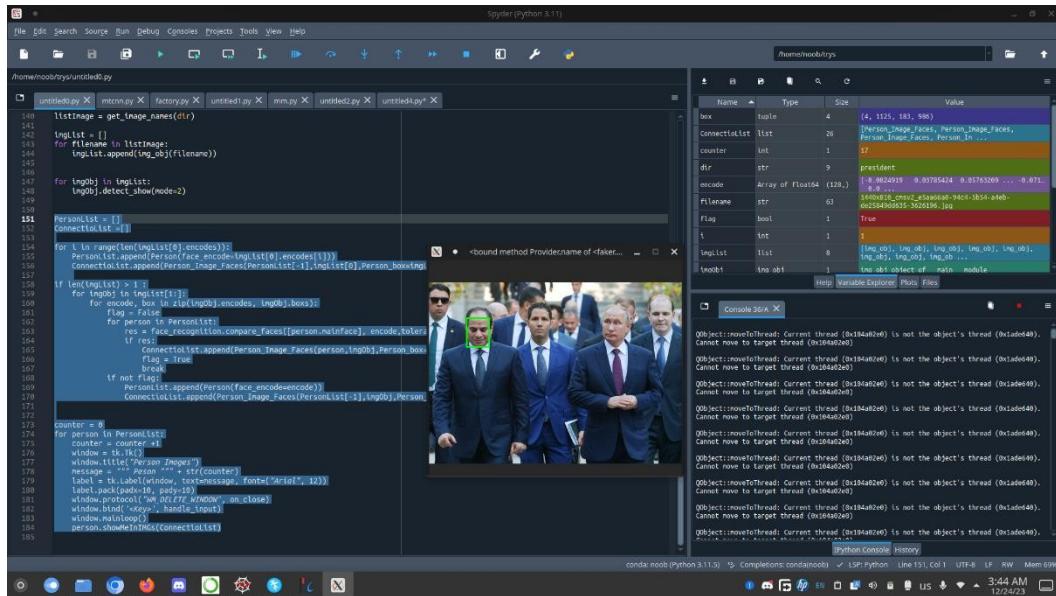
counter = 0
for person in PersonList:
    counter += 1
    window = tk.Tk()
    window.title("Person Images")
    message = str(counter)
    label = tk.Label(window, text=message, font="Arial", 12)
    label.pack(pady=10)
    window.protocol("WM_DELETE_WINDOW", on_close)
    window.bind("<Key>", handle_input)
    window.mainloop()
    person.showInIMGS(ConnectoList)
```



## 2. Person two:







### 3. Person 3:

The screenshots show the execution of a Python script designed for multi-person face detection and tracking. The script uses OpenCV's face detection and tracking modules. It reads a list of image file names, processes them, and displays the results in a Tkinter-based application.

**Code (Top Window):**

```
listImage = get_image_names(dlr)
imgList = []
for filename in listImage:
    imgList.append(imgObj[filename])
for imgObj in imgList:
    imgObj.detect_show(node=2)

personList = []
connectionList = []

for i in range(len(imgList)):
    personList.append(PersonFaceEncodingList[i].encodes[i])
    connectionList.append(PersonImageFaces(PersonList[i-1], imgList[i], PersonBoxingList[i]))
```

**Tkinter Application (Bottom Window):**

```
listImage = get_image_names(dlr)
imgList = []
for filename in listImage:
    imgList.append(imgObj[filename])
for imgObj in imgList:
    imgObj.detect_show(node=2)

personList = []
connectionList = []

for i in range(len(imgList)):
    personList.append(PersonFaceEncodingList[i].encodes[i])
    connectionList.append(PersonImageFaces(PersonList[i-1], imgList[i], PersonBoxingList[i]))
```

## 4. Person 4:

Spyder (Python 3.11)

```

/home/nobodrys/untitled0.py
100 listImage = get_image_names(dir)
101 imgList = []
102 for filename in listImage:
103     imgList.append(img_obj(filename))
104
105 for imgObj in imgList:
106     imgObj.detect_faces(mode=2)
107
108 personList = []
109 connectList = []
110
111 for i in range(len(imgList[0].encodes)):
112     personList.append(PersonFace(imgList[0].encodes[i]))
113     connectList.append(PersonImageFaces(personList[-1],imgList[0].Person_box,imgList[0].box))
114
115 if len(imgList) > 1:
116     for imgObj in imgList[1:]:
117         for encode in imgObj.encodes:
118             flag = False
119             for person in personList:
120                 if person.nameFace == encode:
121                     if res:
122                         connectList.append(PersonImageFaces(person,imgObj,Person_box=box))
123                         flag = True
124                     break
125             if not flag:
126                 personList.append(PersonFace(encode))
127                 connectList.append(PersonImageFaces(personList[-1],imgObj,Person_box=box))
128
129 counter = 0
130 for person in personList:
131     counter += 1
132     window = Tk()
133     window.title("Face Images")
134     message = "Person " + str(counter)
135     Label(window, text=message, font=("Arial", 12)).pack(side=TOP)
136     Label(window, text="").pack(side=BOTTOM)
137     window.protocol("WM_DELETE_WINDOW", on_close)
138     window.bind("x", handle_input)
139     window.mainloop()
140     person.showInImg(connectList)
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185

```

Variable Explorer

Name	Type	Size	Value
box	tuple	4	(4, 1125, 163, 986)
ConnectionList	list	26	[Person.Image_Faces, Person.Image_Faces, ...]
counter	int	1	17
dir	str	9	president
encode	Array of float64 [128, ]	1	[0.682419 0.83785424 0.85763289 ... -0.871...
filename	str	63	16494109_cmyv_e5aa5d94c4-5b54-adcb-9249-956533-362519fc.jpg
flag	bool	2	True
i	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, ...]
imgObj	img obj	1	img obj object of nato module

Console 36/A

```

Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)

```

Spyder (Python 3.11)

```

/home/nobodrys/untitled0.py
100 listImage = get_image_names(dir)
101 imgList = []
102 for filename in listImage:
103     imgList.append(img_obj(filename))
104
105 for imgObj in imgList:
106     imgObj.detect_faces(mode=2)
107
108 personList = []
109 connectList = []
110
111 for i in range(len(imgList[0].encodes)):
112     personList.append(PersonFace(imgList[0].encodes[i]))
113     connectList.append(PersonImageFaces(personList[-1],imgList[0].Person_box,imgList[0].box))
114
115 if len(imgList) > 1:
116     for imgObj in imgList[1:]:
117         for encode in imgObj.encodes:
118             flag = False
119             for person in personList:
120                 if person.nameFace == encode:
121                     if res:
122                         connectList.append(PersonImageFaces(person,imgObj,Person_box=box))
123                         flag = True
124                     break
125             if not flag:
126                 personList.append(PersonFace(encode))
127                 connectList.append(PersonImageFaces(personList[-1],imgObj,Person_box=box))
128
129 counter = 0
130 for person in personList:
131     counter += 1
132     window = Tk()
133     window.title("Face Images")
134     message = "Person " + str(counter)
135     Label(window, text=message, font=("Arial", 12)).pack(side=TOP)
136     Label(window, text="").pack(side=BOTTOM)
137     window.protocol("WM_DELETE_WINDOW", on_close)
138     window.bind("x", handle_input)
139     window.mainloop()
140     person.showInImg(connectList)
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185

```

Variable Explorer

Name	Type	Size	Value
0	tuple	4	(4, 1125, 163, 986)
26	list	26	[Person.Image_Faces, Person.Image_Faces, ...]
1	int	1	17
9	str	9	president
6	float64 [128, ]	1	[0.682419 0.83785424 0.85763289 ... -0.871...
63	str	63	16494109_cmyv_e5aa5d94c4-5b54-adcb-9249-956533-362519fc.jpg
1	bool	2	True
1	int	1	1
8	list	8	[imgObj, imgObj, imgObj, imgObj, ...]
1	img obj	1	img obj object of nato module

Console 36/A

```

Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)
: Current thread (0x104e02e0) is not the object's thread (0x1ade40).
Cannot move to target thread (0x104e02e0)

```

## 5. Person 5:

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named `untitled0.py`. The code implements face detection and recognition using OpenCV and Face Recognition libraries. It reads multiple image files from a directory, processes them to extract faces, and compares them against a main face. A Tkinter window is used to display the results. On the right, the Variable Explorer shows various variables and their values, including lists of images, bounding boxes, and confidence scores. The console output at the bottom indicates errors related to thread access.

```

140 ltsImage = get_Img_names(r"")
141
142 imgList = []
143 for filename in ltsImage:
144     imgList.append(imgObj)(filename)
145
146
147 for imgObj in imgList:
148     imgObj.detect_faces(mode=2)
149
150
151 personList = []
152 connectList = []
153
154 for i in range(len(imgList[0].encodes)):
155     personList.append(Person_Image_Faces(personList[i],imgList[0].Person_BoxingList[i].box[1]))
156     connectList.append(Person_Image_Faces(personList[i],imgList[0].Person_BoxingList[i].box[1]))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode in imgObj.encodes:
161             flag = False
162             for person in personList:
163                 if face_recognition.compare_faces([person.mainFace], encode, tolerance=0.5)[0]:
164                     if res:
165                         connectList.append(Person_Image_Faces(person, imgObj, person_box=box))
166                     flag = True
167                     break
168             if not flag:
169                 personList.append(Person_Image_Faces(encode))
170                 connectList.append(Person_Image_Faces(personList[-1],imgObj, person_box=box))
171
172
173 counter = 0
174 for person in personList:
175     person = person[0]
176     window = tk.Tk()
177     window.title("Person Images")
178     label = tk.Label(window, text="Message", font="Ariel", 12)
179     label.pack(padx=10, pady=10)
180     window.protocol("WM_DELETE_WINDOW", on_close)
181     window.bind('<Escape>', handle_tkquit)
182     window.mainloop()
183     person.showInIMCs(connectList)
184
185

```



## 6. Person 6:

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** The main window displays Python code for face recognition. The code uses the `get\_image\_names` function to list files, iterates through them, and for each file, it loads the image, detects faces, and encodes them. It then compares these encoded faces against a list of known persons' faces. If a match is found, it adds the person's name and bounding box to a list. Finally, it creates a Tkinter window titled "Person Images" to display the images with their respective bounding boxes.
- Variable Explorer:** A separate window titled "Variable explorer" shows the current state of variables. Key variables include:
  - box:** tuple (4, 1125, 181, 980)
  - Connectolist:** list [26] [Person\_Image\_Faces, Person\_Img\_Faces, ...]
  - counter:** int 1 17
  - dir:** str prendent
  - encode:** Array of Float64 (128,)
  - filename:** str 63 d:\258495d635-3626196.jpg
  - Flag:** bool 1 True
  - i:** int 1 1
  - imglist:** list [8] [imgobj, imgobj, imgobj, imgobj, ...]
  - imgobj:** imgobj 1 one ob1 is object of win module
- Console:** Shows several error messages indicating that certain objects are not the object's thread (0x10402e0).
- Python Console:** Displays the command `conda info` and its output.

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** The main window displays Python code for face recognition, identical to the one in the previous screenshot.
- Tkinter Image Viewer:** A separate window titled "Person Images" is displayed, showing a group of people. One person's face is highlighted with a green bounding box, indicating a successful detection and comparison result.
- Variable Explorer:** A separate window titled "Variable explorer" shows the current state of variables, similar to the one in the first screenshot.
- Console:** Shows several error messages indicating that certain objects are not the object's thread (0x10402e0).

## 7. Person 7



The screenshot shows two open Spyder Python environments. The top environment displays the code for 'untitled0.py' which performs face recognition on multiple images and displays results in a Tkinter window. The bottom environment shows the same code running, with the Tkinter window displaying the image of the four leaders and highlighting King Salman's face with a green bounding box. The Python console in the bottom window shows numerous error messages related to thread operations.

```
listImage = get_image_names(dlr)
imgList = []
for filename in listImage:
    imgList.append(Image.open(filename))

for imgObj in imgList:
    imgObj.detect(mode=2)

PersonList = []
ConnectioList = []

for i in range(len(imgList[0].encodes)):
    PersonList.append(Person(face.encode(imgList[0].encodes[i])))
    ConnectioList.append(Person.Image_Faces(PersonList[-1], imgList[0].Person_box+imgList[0].boxs[i]))
```

This block of code initializes lists for people and connections. It iterates through the first image's encodes, creating a person object for each encode and adding it to the PersonList and its corresponding connection object to the ConnectioList.

```
if len(imgList) > 1:
    for imgObj in imgList[1:]:
        for encode, box in zip(imgObj.encodes, imgObj.boxs):
            flag = False
            for person in PersonList:
                res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
                if res:
                    ConnectioList.append(Person.Image_Faces(person, imgObj, Person_box=box))
                    flag = True
            if not flag:
                PersonList.append(Person(face_encode=encode))
                ConnectioList.append(Person.Image_Faces(PersonList[-1], imgObj, Person_box=box))
```

This block handles comparing faces between all images. For each subsequent image, it iterates through all people in the PersonList. If a match is found, a connection is added. If no match is found, a new person object is created and added to the PersonList along with a new connection object.

```
counter = 0
for person in PersonList:
    counter = counter + 1
    window = tk.Tk()
    window.title('Person Images')
    message = tk.Label(window, text='Image ' + str(counter))
    label = tk.Label(window, text=message, font=("Arial", 12))
    label.pack(padx=10, pady=10)
    window.protocol("WM_DELETE_WINDOW", on_close)
    window.mainloop()
    window.after(1000)
    person.showInImgS(connectioList)
```

This block creates a Tkinter window for each person found. It sets the window title, displays a message, and waits for a key press before closing the window. It also includes a short delay and calls the 'showInImgS' method on the person object.

```
listImage = get_image_names(lsr)
imgList = []
for filename in listImage:
    imgList.append(Image.open(filename))

for imgObj in imgList:
    imgObj.detect_s()

PersonList = []
ConnectioList = []

for i in range(len(imgList[0].encodes)):
    PersonList.append(Person(face.detect_s()))
    ConnectioList.append(Person.Image_Faces(PersonList[-1], imgList[0].Person_box+imgList[0].boxs[i]))
```

This block is identical to the one above, but uses the 'detect\_s' method instead of 'detect'.

```
if len(imgList) > 1:
    for imgObj in imgList[1:]:
        for encode, box in zip(imgObj.encodes, imgObj.boxs):
            flag = False
            for person in PersonList:
                res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
                if res:
                    ConnectioList.append(Person.Image_Faces(person, imgObj, Person_box=box))
                    flag = True
            if not flag:
                PersonList.append(Person(face.detect_s()))
                ConnectioList.append(Person.Image_Faces(PersonList[-1], imgObj, Person_box=box))
```

This block is identical to the one above, but uses the 'detect\_s' method instead of 'detect'.

```
counter = 0
for person in PersonList:
    counter = counter + 1
    window = tk.Tk()
    window.title('Person Images')
    message = tk.Label(window, text='Image ' + str(counter))
    label = tk.Label(window, text=message, font=("Arial", 12))
    label.pack(padx=10, pady=10)
    window.protocol("WM_DELETE_WINDOW", on_close)
    window.mainloop()
    window.after(1000)
    person.showInImgS(connectioList)
```

This block is identical to the one above, but uses the 'detect\_s' method instead of 'detect'.

## 8. Person 8:

Spyder (Python 3.11)

/home/noob/trys/untitled0.py

```
140 listImage = get_image_names(dtr)
141 
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145 
146 
147 for imgObj in imgList:
148     imgObj.detect_show(node=2)
149 
150 
151 personList = []
152 connectList = []
153 
154 for i in range(len(imgList[0].encodes)):
155     personList.append(PersonFace_encode=encList[i], Person_Img_Faces=connectList[i])
156     connectList.append(Person_Img_Faces(personList[-1], imgList[0].Person_box=encList[i].boxs[i]))
157 
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, imgObj in zip(imgObj.encodes, imgObj.boxs):
161             for person in personList:
162                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
163                 if res == True:
164                     connectList.append(Person_Img_Faces(person, imgObj, Person_box=box))
165                     flag = True
166                     break
167                 if not flag:
168                     personList.append(PersonFace_encode=encode)
169                     connectList.append(Person_Img_Faces(personList[-1], imgObj, Person_box=box))
170 
171 counter = 0
172 for person in personList:
173     counter = counter + 1
174     window = tk.Tk()
175     window.title("Person Images")
176     message = "Person"
177     label = tk.Label(window, text=message, font=("Arial", 12))
178     label.pack(padx=10, pady=10)
179     window.protocol("WM_DELETE_WINDOW", on_close)
180     window.bind('<key>', handle_input)
181     window.mainloop()
182 person.showInINGS(connectList)
```

Person 8

Variable Explorer

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnectList	list	26	[Person_Img_Faces, Person_Img_Faces, Person_Img_Faces, Person_In ...]
counter	int	1	17
dtr	str	9	president
encode	Array of float64	(128,)	[ -0.0824919  0.83785424  0.85763289 ... -0.871 ...]
filename	str	63	140x810_csv2_e5adda9-94c4-5d4-adeb-de59496853-3626196.jpg
flag	bool	1	True
i	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
imgObj	imgObj	1	imgObj object of main module

Console 36/A

```
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
```

Python Console History

```
conda: noob (Python 3.11.5) ✓ Completions: condanoob ✓ LSP: Python Line 151, Col 1 UTF-8 LF RW Mem 68%
```

3:45 AM 12/24/23

File Edit Search Source Run

/home/noob/trys/untitled0.py

```
140 listImage = get_image_names(dtr)
141 
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145 
146 
147 for imgObj in imgList:
148     imgObj.detect_show(node=2)
149 
150 
151 personList = []
152 connectList = []
153 
154 for i in range(len(imgList[0].encodes)):
155     personList.append(PersonFace_encode=encList[i], Person_Img_Faces=connectList[i])
156     connectList.append(Person_Img_Faces(personList[-1], imgList[0].Person_box=encList[i].boxs[i]))
157 
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, imgObj in zip(imgObj.encodes, imgObj.boxs):
161             for person in personList:
162                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
163                 if res == True:
164                     connectList.append(Person_Img_Faces(person, imgObj, Person_box=box))
165                     flag = True
166                     break
167                 if not flag:
168                     personList.append(PersonFace_encode=encode)
169                     connectList.append(Person_Img_Faces(personList[-1], imgObj, Person_box=box))
170 
171 counter = 0
172 for person in personList:
173     counter = counter + 1
174     window = tk.Tk()
175     window.title("Person Images")
176     message = "Person"
177     label = tk.Label(window, text=message, font=("Arial", 12))
178     label.pack(padx=10, pady=10)
179     window.protocol("WM_DELETE_WINDOW", on_close)
180     window.bind('<key>', handle_input)
181     window.mainloop()
182 person.showInINGS(connectList)
```

## 9. Person 9:

The screenshot shows two Python environments running side-by-side. Both environments are displaying the same image of four people (Sisi, King Salman, Melania, and Trump) gathered around a glowing globe.

**Top Environment (Spyder):**

- Code:** The code is identical to the one in the bottom environment, showing the detection of King Salman's face and its bounding box.
- Output:** A green bounding box highlights King Salman's face. The Spyder interface includes a variable explorer showing values for various variables like `imgList` and `imgObj`, and a console window with several error messages related to thread manipulation.

**Bottom Environment (Spyder):**

- Code:** The code is identical to the one in the top environment, showing the detection of King Salman's face and its bounding box.
- Output:** A green bounding box highlights King Salman's face. The Spyder interface includes a variable explorer showing values for various variables like `imgList` and `imgObj`, and a console window with several error messages related to thread manipulation.

## 10. Person 10:

The following screenshots show the Python code for face recognition and its execution results.

**Code (untitled0.py):**

```

140 listImage = get_image_names(dlr)
141
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145
146
147 for imgObj in imgList:
148     imgObj.detect_show(node=2)
149
150
151 PersonList = []
152 ConnectioList = []
153
154 for i in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[i]))
156     ConnectioList.append(Person_Image_Faces(Person_In ...
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, i):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_f
164                 if res:
165                     ConnectioList.append(person)
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_en
170                 ConnectioList.append(Person_Imag
171
172 counter = 0
173 for person in PersonList:
174     counter = counter + 1
175     window = tk.TK()
176     window.title("Person Images")
177     message = "Person " + str(counter)
178     label = tk.Label(window, text=message, font=
179     label.pack(padx=10, pady=10)
180     window.protocol("WM_DELETE_WINDOW", on_clos
181     window.bind("<key>", handle_input)
182     window.mainloop()
183     person.showInIMGS(ConnectioList)
184
185

```

**Execution Results:**

Two screenshots of the Spyder Python IDE showing the execution of the code and the resulting output.

**Screenshot 1:**

- Code execution window: Shows the Python code for face recognition.
- Variable Explorer window: Shows variables like `box` (tuple), `ConnectioList` (list), `counter` (int), `dir` (str), `encode` (Array of float64), `filename` (str), `flag` (bool), and `i` (int).
- Image view: Shows a group of people with a green bounding box highlighting a specific person's face.
- System tray: Shows icons for various applications like a browser, file manager, and terminal.

**Screenshot 2:**

- Code execution window: Shows the same Python code.
- Variable Explorer window: Shows variables like `box` (tuple), `ConnectioList` (list), `counter` (int), `dir` (str), `encode` (Array of float64), `filename` (str), `flag` (bool), and `i` (int).
- Console window: Shows numerous error messages related to thread management, such as "Cannot move to target thread".
- Image view: Shows a person labeled "Person 10" with a green bounding box.
- System tray: Shows icons for various applications.

## 11. Person 11:

Spyder (Python 3.11)

```

/home/noob/trys/untitled0.py

140 listImage = get_image_names(dtr)
141
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145
146
147 for imgObj in imgList:
148     imgObj.detect_show(mode=2)
149
150
151 PersonList = []
152 ConnecttoList = []
153
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
156     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box(imgList[0].boxs[l])))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxs):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
164                 if res:
165                     ConnecttoList.append(Person_Image_Faces(person, imgObj, Person_box(box)))
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_encode=encode))
170                 ConnecttoList.append(Person_Image_Faces(PersonList[-1], imgObj, Person_box(box)))
171
172
173 counter = 0
174 for person in PersonList:
175     counter = counter + 1
176     window = tk.Tk()
177     window.title("Person Images")
178     message = "Person " + str(counter)
179     label = tk.Label(window, text=message, font=("Arial", 12))
180     label.pack(padx=10, pady=10)
181     window.protocol("WM_DELETE_WINDOW", on_close)
182     window.bind('<Key-y>', handle_input)
183     window.mainloop()
184     person.showInIMGS(ConnecttoList)
185

```

Person 11

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnecttoList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dtr	str	9	president
encode	Array of float64	128,	[0.0824919 0.83785424 0.85763289 ... -0.871...
filename	str	63	140e810_cmv2_e5aa6e0-94c4-5054-adeb-de25849dd053-3262196.jpg
flag	bool	1	True
l	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
imgObj	Ima obj	1	ima obj object of main module

Spyder (Python 3.11)

```

/home/noob/trys/untitled0.py

140 listImage = get_image_names(dtr)
141
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145
146
147 for imgObj in imgList:
148     imgObj.detect_show(mode=2)
149
150
151 PersonList = []
152 ConnecttoList = []
153
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
156     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box(imgList[0].boxs[l])))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxs):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
164                 if res:
165                     ConnecttoList.append(Person_Image_Faces(person, imgObj, Person_box(box)))
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_encode=encode))
170                 ConnecttoList.append(Person_Image_Faces(PersonList[-1], imgObj, Person_box(box)))
171
172
173 counter = 0
174 for person in PersonList:
175     counter = counter + 1
176     window = tk.Tk()
177     window.title("Person Images")
178     message = "Person " + str(counter)
179     label = tk.Label(window, text=message, font=("Arial", 12))
180     label.pack(padx=10, pady=10)
181     window.protocol("WM_DELETE_WINDOW", on_close)
182     window.bind('<Key-y>', handle_input)
183     window.mainloop()
184     person.showInIMGS(ConnecttoList)
185

```

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnecttoList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dtr	str	9	president
encode	Array of float64	128,	[0.0824919 0.83785424 0.85763289 ... -0.871...
filename	str	63	140e810_cmv2_e5aa6e0-94c4-5054-adeb-de25849dd053-3262196.jpg
flag	bool	1	True
label	int	1	1
plist	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
obj	Ima obj	1	ima obj object of main module

Spyder (Python 3.11)

/home/noob/trys/untitled0.py

```
140 listImage = get_image_names(dlr)
141 imgList = []
142 for filename in listImage:
143     imgList.append(img_obj(filename))
144
145 for imgObj in imgList:
146     imgObj.detect_show(mode=2)
147
148
149
150
151 PersonList = []
152 ConnecttoList = []
153
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
156     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxes):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainFace], encode,
164                                                     tolerance=0.6)
165                 if res:
166                     ConnecttoList.append(Person_Image_Faces(person, imgObj, Perso
167
168                     flag = True
169                     break
170
171             if not flag:
172                 PersonList.append(Person(face_encode=encode))
173                 ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgObj))
174
175
176 Counter = 0
177 for person in PersonList:
178     counter = counter +1
179     window = tk.Tk()
180     window.title("Person Images")
181     message = tk.Label(window, text=message, font=("Arial", 12))
182     label = tk.Label(window, text="Person "+ str(counter))
183     label.pack(padx=10, pady=10)
184     window.protocol("WM_DELETE_WINDOW", on_close)
185     window.mainloop()
186     person.showInIMGs(ConnecttoList)
```

Variable explorer

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnecttoList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dlr	str	9	president
encode	Array of floats64 (128,)	1	[-0.0824919, 0.63785424, 0.05763289 ... -0.071...
filename	str	63	1404415_cavv2_55a6d649c4-2954-a4eb-de28495dd635-3626196.jpg
flag	bool	1	True
i	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
imgObj	img obj	1	two ob1 object of main module

Console 3/6/A

```
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
```

conda: noob (Python 3.11.5) ✓ Completions: condain/noob ✓ LSP: Python Line 151, Col 1 UTF-8 LF RW Mem 69%

3:45 AM 12/24/23

Spyder (Python 3.11)

/home/noob/trys/untitled0.py

```
140 listImage = get_image_names(dlr)
141 imgList = []
142 for filename in listImage:
143     imgList.append(img_obj(filename))
144
145 for imgObj in imgList:
146     imgObj.detect_show(mode=2)
147
148
149
150
151 PersonList = []
152 ConnecttoList = []
153
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
156     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxes):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainFace], encode,
164                                                     tolerance=0.6)
165                 if res:
166                     ConnecttoList.append(Person_Image_Faces(person, imgObj, Perso
167
168                     flag = True
169                     break
170
171             if not flag:
172                 PersonList.append(Person(face_encode=encode))
173                 ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgObj))
174
175
176 Counter = 0
177 for person in PersonList:
178     counter = counter +1
179     window = tk.Tk()
180     window.title("Person Images")
181     message = tk.Label(window, text=message, font=("Arial", 12))
182     label = tk.Label(window, text="Person "+ str(counter))
183     label.pack(padx=10, pady=10)
184     window.protocol("WM_DELETE_WINDOW", on_close)
185     window.mainloop()
186     person.showInIMGs(ConnecttoList)
```

Variable explorer

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnecttoList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dlr	str	9	president
encode	Array of floats64 (128,)	1	[-0.0824919, 0.63785424, 0.05763289 ... -0.071...
filename	str	63	1404415_cavv2_55a6d649c4-2954-a4eb-de28495dd635-3626196.jpg
flag	bool	1	True
i	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
imgObj	img obj	1	two ob1 object of main module

Console 3/6/A

```
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
Qobject::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0)
```

conda: noob (Python 3.11.5) ✓ Completions: condain/noob ✓ LSP: Python Line 151, Col 1 UTF-8 LF RW Mem 69%

3:46 AM 12/24/23

## 12. Person 12:

Spyder (Python 3.11)

/home/noob/trys/untitled0.py

```
140 listImage = get_image_names(dtr)
141 
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145 
146 for imgObj in imgList:
147     imgObj.detect_show(mode=2)
148 
149 
150 
151 PersonList = []
152 ConnectioList = []
153 
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=encodes[l]))
156     ConnectioList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box, imgList[0].boxs[l]))
157 
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxs):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
164                 if res:
165                     ConnectioList.append(Person_Image_Faces(person, imgObj, Person_box=box))
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_encode=encode))
170                 ConnectioList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))
171 
172 
173 counter = 0
174 for person in PersonList:
175     counter = counter + 1
176     window = tk.Tk()
177     window.title("Person Images")
178     message = "Person " + str(counter)
179     label = tk.Label(window, text=message, font=("Arial", 12))
180     label.pack(padx=10, pady=10)
181     window.protocol("WM_DELETE_WINDOW", on_close)
182     window.bind('<Key>', handle_input)
183     window.mainloop()
184     person.showInIMGS(ConnectioList)
185 
```

Person 12

Name Type Size Value

box	tuple	4	(4, 1125, 183, 986)
ConnectioList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dir	str	9	president
encode	Array of float64	128,	[ 0.082919 0.83785424 0.85763289 ... -0.871 ... 0.0 ...]
filename	str	63	1404810_csv2_e5aa65a9-94c4-5054-adeb-de58496863-3626196.jpg
flag	bool	1	True
l	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
imgObj	img obj	1	img obj object of main module

Console 36/A

```
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640). Cannot move to target thread (0x104a02e0).
```

IPython Console History

conda: noob (Python 3.11.5) ✓ Completions: condanoob ✓ LSP: Python Line 151, Col 1 UTF-8 LF RW Mem 69%

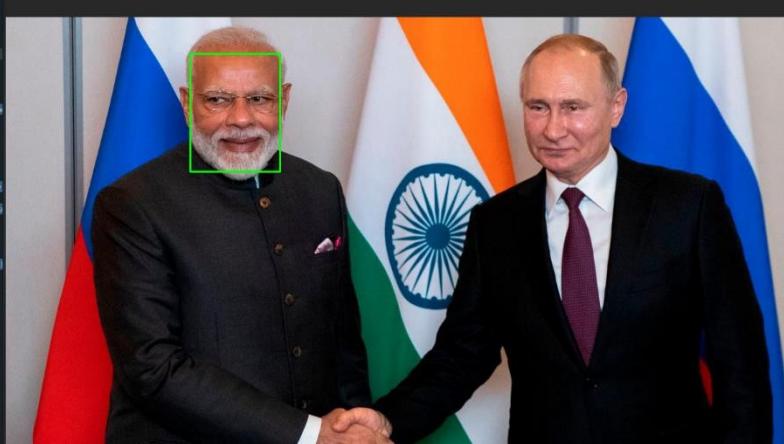
3:46 AM 12/24/23

Spyder (Python 3.11)

/home/noob/trys/untitled0.py

```
140 listImage = get_image_names(dtr)
141 
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145 
146 for imgObj in imgList:
147     imgObj.detect_show(mode=2)
148 
149 
150 
151 PersonList = []
152 ConnectioList = []
153 
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=encodes[l]))
156     ConnectioList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box, imgList[0].boxs[l]))
157 
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxs):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
164                 if res:
165                     ConnectioList.append(Person_Image_Faces(person, imgObj, Person_box=box))
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_encode=encode))
170                 ConnectioList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))
171 
172 
173 counter = 0
174 for person in PersonList:
175     counter = counter + 1
176     window = tk.Tk()
177     window.title("Person Images")
178     message = "Person " + str(counter)
179     label = tk.Label(window, text=message, font=("Arial", 12))
180     label.pack(padx=10, pady=10)
181     window.protocol("WM_DELETE_WINDOW", on_close)
182     window.bind('<Key>', handle_input)
183     window.mainloop()
184     person.showInIMGS(ConnectioList)
185 
```

<bound method Provider.name of <faker.providers.person.en\_US.Provider object at 0x7f05b4071350>



3:46 AM 12/24/23

### 13. Person 13:

Spyder (Python 3.11)

```
File Edit Search Source Run Debug Consoles Projects Tools View Help /home/noob/trys/untitled0.py
```

This screenshot shows the Spyder IDE interface running Python 3.11. The code in the editor is for a face recognition application. It reads images from a directory, extracts faces, encodes them, and compares them against a main face to identify people. A Tkinter window is displayed, showing three men in suits. A green bounding box highlights the face of the man on the right.

The code highlights the following variable values in the Variable Explorer:

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnectioList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dirl	str	9	president
encode	Array of float64	(128,)	[ 0.822919 0.83785424 0.85763289 ... -0.871 ...]
filename	str	63	1440x810_cmy2_e5aa6a9-94c4-5054-adef-de25b49dd635-3262196.jpg
flag	bool	1	True
i	int	1	1
imgList	list	8	[imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj, imgObj]
inobj	Ima ob	1	ima ob object of main module

Console 36/A (Python Console History):

```
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
```

Bottom status bar: conda: noob (Python 3.11.5) Completions: condanoob LSP: Python Line 151, Col 1 UTF-8 LF RW Mem 69%

3:46 AM 12/24/23

Spyder (Python 3.11)

```
File Edit Search Source Run Debug Consoles Projects Tools View Help /home/noob/trys/untitled0.py
```

This screenshot shows the Spyder IDE interface again, running Python 3.11. The code and variable values are identical to the first screenshot, but the Tkinter window now displays a single image of three men in suits. The image is larger and centered compared to the previous screenshot.

Console 36/A (Python Console History):

```
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
Object::moveToThread: Current thread (0x104a02e0) is not the object's thread (0x1ade640).
Cannot move to target thread (0x104a02e0)
```

Bottom status bar: conda: noob (Python 3.11.5) Completions: condanoob LSP: Python Line 151, Col 1 UTF-8 LF RW Mem 70%

3:46 AM 12/24/23

## 14. Person 14:

Spyder (Python 3.11)

```
/home/noob/noob/try/untitled0.py
```

```

140 listImage = get_image_names(dlr)
141
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145
146
147 for imgObj in imgList:
148     imgObj.detect_show(node=2)
149
150
151 PersonList = []
152 ConnecttoList = []
153
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
156     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxes):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)
164                 if res:
165                     ConnecttoList.append(Person_Image_Faces(person, imgObj, Person_box= imgList[0].boxes[l]))
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_encode=encode))
170                 ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))
171
172
173 counter = 0
174 for person in PersonList:
175     counter = counter +1
176     window = tk.Tk()
177     window.title("Person Images")
178     message = "Person " + str(counter)
179     label = tk.Label(window, text=message, font=("Arial", 12))
180     label.pack(padx=10, pady=10)
181     window.protocol("WM_DELETE_WINDOW", on_close)
182     window.bind('<Key>', handle_input)
183     window.mainloop()
184     person.showInIMGS(ConnecttoList)
185

```

Spyder (Python 3.11)

```
/home/noob/noob/try/untitled0.py
```

```

140 listImage = get_image_names(dlr)
141
142 imgList = []
143 for filename in listImage:
144     imgList.append(imgObj)(filename)
145
146
147 for imgObj in imgList:
148     imgObj.detect_show(node=2)
149
150
151 PersonList = []
152 ConnecttoList = []
153
154 for l in range(len(imgList[0].encodes)):
155     PersonList.append(Person(face_encode=imgList[0].encodes[l]))
156     ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgList[0].Person_box= imgList[0].boxes[l]))
157
158 if len(imgList) > 1:
159     for imgObj in imgList[1:]:
160         for encode, box in zip(imgObj.encodes, imgObj.boxes):
161             flag = False
162             for person in PersonList:
163                 res = face_recognition.compare_faces([person.mainface], encode, tolerance=0.5)[0]
164                 if res:
165                     ConnecttoList.append(Person_Image_Faces(person, imgObj, Person_box=box))
166                     flag = True
167                     break
168             if not flag:
169                 PersonList.append(Person(face_encode=encode))
170                 ConnecttoList.append(Person_Image_Faces(PersonList[-1],imgObj,Person_box=box))
171
172
173 counter = 0
174 for person in PersonList:
175     counter = counter +1
176     window = tk.Tk()
177     window.title("Person Images")
178     message = "Person " + str(counter)
179     label = tk.Label(window, text=message, font=("Arial", 12))
180     label.pack(padx=10, pady=10)
181     window.protocol("WM_DELETE_WINDOW", on_close)
182     window.bind('<Key>', handle_input)
183     window.mainloop()
184     person.showInIMGS(ConnecttoList)
185

```

## 15. Person 15:

Two screenshots of the Spyder Python IDE showing the execution of a script named `untitled0.py`. The script performs face recognition on multiple images and displays the results in a Tkinter window.

The code reads images from a directory (`listImage`) and encodes each person's face. It then compares these encodes against a main face (`mainFace`). If a match is found, it adds the person's name and bounding box to a list (`PersonList`).

After processing all images, a Tkinter window titled "Person Images" is displayed, showing a grid of images with green bounding boxes around detected faces. The Variable Explorer shows the state of variables, including the list of detected persons (`PersonList`), the list of encodes (`imgList`), and the main encode (`mainFace`).

Console output shows numerous errors indicating that objects cannot be moved to target threads, likely due to GUI updates from a different thread.



## 16. Person 16:

Two screenshots of the Spyder Python IDE showing the execution of a script named `untitled0.py`.

**Screenshot 1:**

The code in `untitled0.py` performs the following steps:

- Imports required modules: `mrcnn`, `factory`, `mm`, and `united4`.
- Initializes lists for images and detections.
- Loops through images in `listImage`:
  - For each image, it detects faces and appends them to `imgList`.
  - For each image in `imgList`, it performs face recognition to find matches against all other images in `imgList`. If a match is found, it appends the detection results to `ConnectioList`.
- Creates a Tkinter window titled "Person Images".
- Creates a label in the window and sets its text to "Person".
- Packs the label and creates a window protocol button labeled "DELETE WINDOW".
- Binds the "X" button to the "on\_close" event.
- Enters the Tkinter main loop.
- Shows the window.

The **Variable Explorer** shows the following variables:

Name	Type	Size	Value
box	tuple	4	(4, 1125, 183, 986)
ConnectioList	list	26	[Person_Image_Faces, Person_Image_Faces, Person_Image_Faces, Person_In ...]
counter	int	1	17
dir	str	9	president
encode	Array of float64	1128	[0.0824919 0.83785424 0.85763289 ... -0.871...
filename	str	63	1440x810_cmv2_e5aa65a9-94c4-5054-afab-de25b49d6053-3c26196.jpg
flag	bool	1	True
i	int	1	1
imgList	list	8	[img_obj, img_obj, img_obj, img_obj, img_obj, img_obj, img_obj, img_obj]
inobj	img obj	1	img obj object of main module

**Screenshot 2:**

The window title is now "Person Images" and contains the text "Person". Below the text, there is a **<bound method Provider.name of <faker...>** placeholder.

The **Variable Explorer** shows the same variable values as in Screenshot 1.

## 17. Person 17:

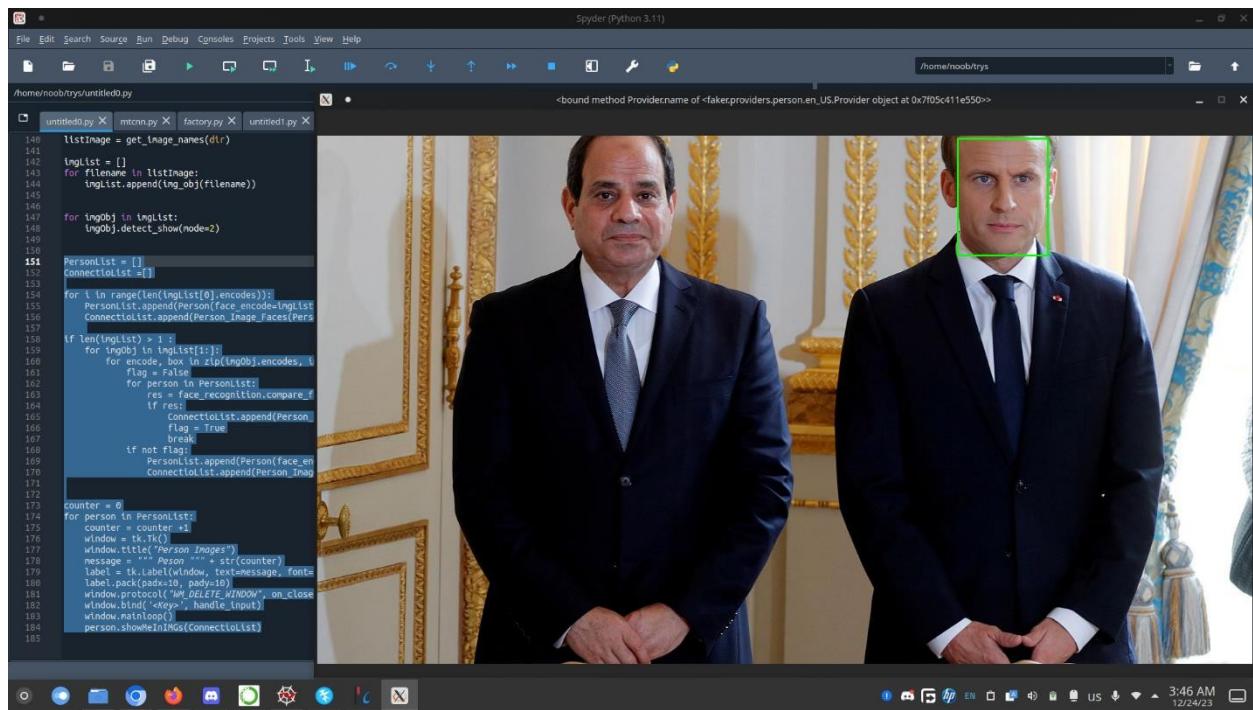
Two screenshots of the Spyder Python 3.11 IDE showing the execution of a script named `/home/noob/trys/untitled0.py`.

The script performs the following steps:

- Imports necessary modules: `os`, `cv2`, `numpy`, `face_recognition`, `tkinter`, and `tkinter.messagebox`.
- Creates lists for `listImage` and `imgList`.
- Loops through `listImage` to append each file name to `imgList`.
- For each image in `imgList`, it uses `imgObj.detect_show(mode=2)` to detect faces.
- Creates `PersonList` and `ConnectioList`.
- For each image in `imgList`, it extracts the main face and encodes it, then appends it to `PersonList` and `ConnectioList`.
- For each image in `imgList` (excluding the first), it compares the main face of the current image with all previously encoded faces. If a match is found, it appends the person's name and bounding box to `ConnectioList`. A flag is set to `True` if a match is found.
- After processing all images, it creates a Tkinter window titled "Person Images".
- In the window, it displays the images from `imgList` sequentially. For each image, it checks if a match was found. If so, it draws a green bounding box around the detected face.
- When a user clicks "Next", it moves to the next image. When "Delete" is clicked, it removes the current image from the list.
- The window has a "Close" button.
- Finally, it calls `person.showInIMGS(ConnectioList)`.

**Screenshot 1:** Shows the initial state of the script execution. The Variable Explorer shows local variables like `box`, `ConnectioList`, `counter`, `dir`, `encode`, `filename`, `flag`, `l`, `imgList`, and `imgObj`. The Console shows numerous error messages indicating that threads are being moved.

**Screenshot 2:** Shows the application window after processing two images. The second image features a green bounding box around the detected face of a man in a suit. The Variable Explorer and Console show updated data and error messages.



## Discussion:

The presented program exhibits a sophisticated approach to the task of consolidating pictures of individuals across multiple images, demonstrating a seamless integration of advanced algorithms and libraries. The use of the MTCNN (Multi-task Cascaded Convolutional Networks) algorithm for face detection introduces a reliable mechanism for identifying faces within images, while the subsequent conversion of face box formats ensures compatibility with the face\_recognition library. Leveraging face\_recognition with the HOG (Histogram of Oriented Gradients) algorithm and additional techniques like num\_jitters enable the extraction of intricate facial features, contributing to accurate face recognition. One notable aspect of the algorithm is its systematic organization, starting with the reading of image paths, processing of images, and the establishment of a Person List and Connection List. The initialization of persons from the first image and the creation of connections between them provide a solid foundation for subsequent iterations. The iterative process of analyzing each image and checking for existing persons based on encoding differences ensures a comprehensive and dynamic approach to handling varied datasets. The choice of coding practices, incorporating a suite of libraries such as cv2, copy, Faker, tkinter, os, mtcnn, face\_recognition, dlib, and CUDA drivers, reflects a commitment to efficiency and performance. The inclusion of CUDA drivers for GPU acceleration underscores the program's scalability, allowing for faster processing and handling of larger datasets. Despite the absence of training data, the program showcases its adaptability by relying on the pre-trained MTCNN model. This decision simplifies the user experience, as users can seamlessly push pictures into the application without the need for additional training steps. In the testing phase, the program has demonstrated success in critical functionalities, including reading input images, classifying faces, and recognizing individuals. The robustness of the algorithm, combined with effective coding practices and library utilization, positions this application as a valuable tool in scenarios where efficient face recognition and picture organization are paramount, such as smartphone photo studios or similar environments. Overall, the discussion highlights the program's strengths in algorithmic design, coding practices, and real-world applicability.

## Conclusion

In conclusion, the developed program is a robust solution for consolidating pictures of individuals across multiple images, serving as an asset in smartphone photo studios or settings requiring efficient face recognition and categorization. The algorithm systematically reads image paths, processes images using the MTCNN face detection algorithm with specified confidence thresholds and converts face box formats for compatibility with the face\_recognition library. Utilizing face\_recognition with HOG and num\_jitters, the program calculates face encodings to extract detailed features. It establishes a Person List and Connection List, initializing persons from the first image and creating connections between them. Subsequent images are then analyzed, checking for existing people in the Person List based on encoding differences, and creating new persons as needed. The program leverages libraries such as cv2, copy, Faker, tkinter, os, mtcnn, face\_recognition, dlib, and CUDA drivers for GPU acceleration. Notably, the application excels in testing, successfully reading input images, classifying faces, and recognizing individuals, showcasing its efficacy in seamlessly organizing and consolidating pictures through the adept integration of advanced algorithms and libraries.

## Reference

- [1] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection." Available: <https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>.
- [2] D. Adakane, "What are Haar Features used in Face Detection ?," Medium, Nov. 13, 2019. <https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b>
- [3] Chi-Feng Wang, "How Does A Face Detection Program Work? (Using Neural Networks)," Medium, Jul. 27, 2018. <https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff>
- [4] Kushagra Tiwary, "Histograms Of Oriented Gradients for Human Detection, N. Dalal & B. Triggs," Medium, Nov. 08, 2017. <https://medium.com/@ktiworthy2/scattered-thoughts-on-ml-68d30f44da19>
- [4] C.-F. Wang, "What Does A Face Detection Neural Network Look Like?" Medium, Jul. 27, 2018. <https://towardsdatascience.com/face-detection-neural-network-structure-257b8f6f85d1>
- [5] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499–1503, Oct. 2016, doi: <https://doi.org/10.1109/lsp.2016.2603342>.
- [6] M. L. & Statistics, "Stochastic Gradient Descent: Unveiling the Core of Neural Network Training," Medium, Nov. 29, 2023. <https://medium.com/@ML-STATS/stochastic-gradient-descent-unveiling-the-core-of-neural-network-training-14a9008f1cce> (accessed Dec. 28, 2023).