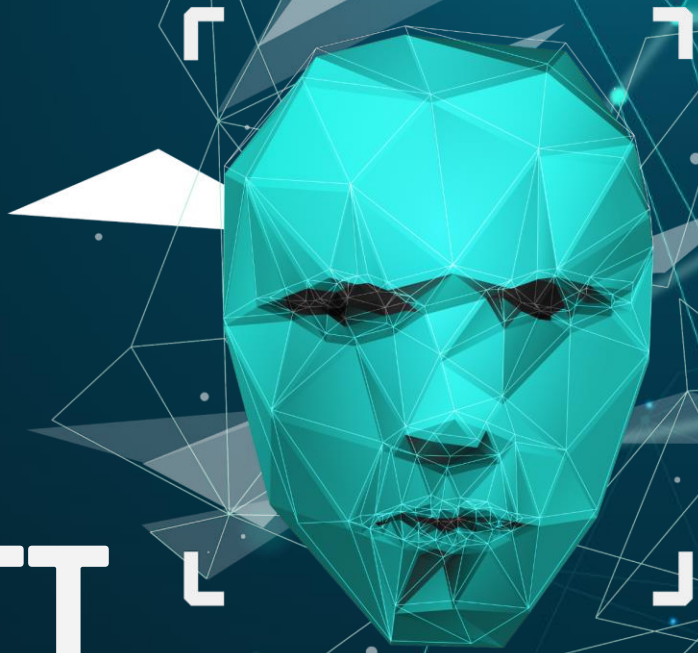
Báo cáo tiến độ lần 3
Nhóm 13-TT



ĐỀ TÀI : ĐIỂM DANH – CHẤM CÔNG BẰNG NHẬN DIỆN KHUÔNG MẶT

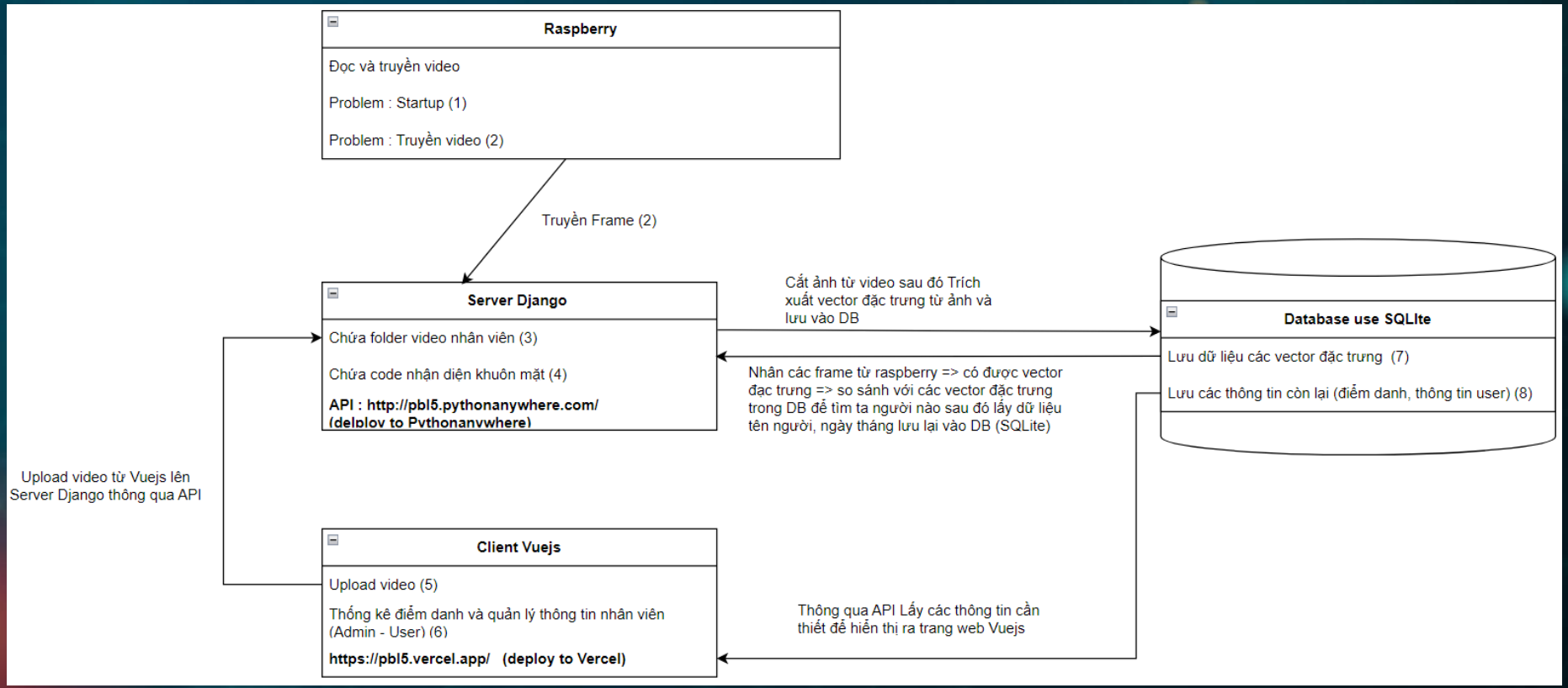
Nội dung chính gồm các phần sau :

Phần I . Deploy API Server Django, Client Vuejs , Run file py trong raspberry (Nguyễn Văn Mạnh – Captain)

Phần II. Cài đặt phần cứng Raspberry, Triển khai các API cho front-end (Nguyễn Công Cường)

Phần III. Xây dựng Model nhận diện khuôn mặt (Nguyễn Văn Hoàng Phúc – Trần Thanh Nguyên)

Nhắt lại Sơ đồ hệ thống



Deploy API Server Django, Client Vuejs Run file py trong raspberry

01

Nguyễn Văn Mạnh – Captain

Client Vuejs

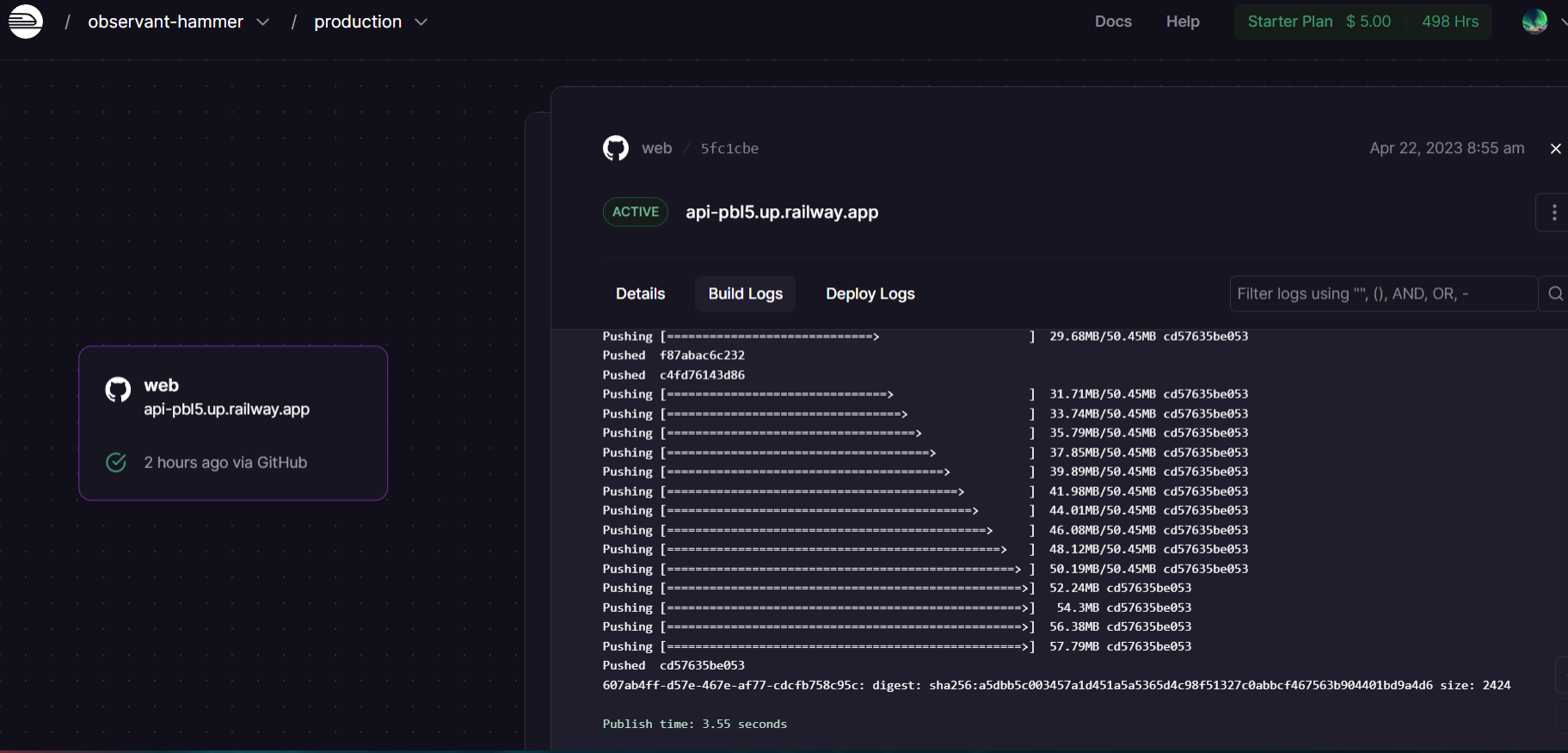
Demo User : https://pbl-5.vercel.app/main/login

Email : user123@gmail.com
Password : user123

Demo Admin : https://pbl-5.vercel.app/admin/login

Email : admin1@gmail.com
Password : admin1

Server Django



Server Django

- BlogAppDjango
- PBL5_API_Django
  - GET Get All
  - GET Get an employe
  - POST Add Employes
  - PUT Edit
  - DEL Delete
  - GET New Request
  - GET Get Image of User
  - POST Login
  - POST Register
  - PATCH Update Infor
  - GET Get all user
  - GET Get Admins
  - DEL Delete
  - GET Change PW

PATCH http://localhost:8000/users/12/update/ Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	Bulk Edit
fullname	Phhuc		
url_video	275256075_1345597292551162_3748684194680711...		

Body Cookies Headers (10) Test Results Status: 200 OK Time: 6.71 s Size: 8.07 KB Save as Example

Pretty Raw Preview Visualize JSON


```
{
    "url": [
        [
            -0.09050779789686203,
            0.06793289631605148,
            0.011828586459159851,
            0.0468987412750721,
            -0.05647725984454155,
            -0.030707167461514473,
            0.012521924450993538,
            -0.12940669059753418,
            0.1100224182009697,
            -0.03621833026409149,
            0.19221241772174835,
            0.012922246009111404,
            -0.17708495259284973,
            -0.09255093336105347,
            -0.007459529616534 71,
```

Xây dựng REST API by Django

02

Nguyễn Công Cường

Cài đặt phần cứng Raspberry

Phần 1

Cài đặt hệ điều hành cho Raspberry:

Các thiết bị phần cứng cần thiết:

- Màn hình máy tính
- Bàn phím
- Raspberry pi 3 model B
- Thẻ nhớ
- Đầu đọc thẻ nhớ
- Các dây nối

Cài đặt hệ điều hành cho Raspberry:

Chương trình cài đặt hệ điều hành Raspberry

Install Raspberry Pi OS using Raspberry Pi Imager

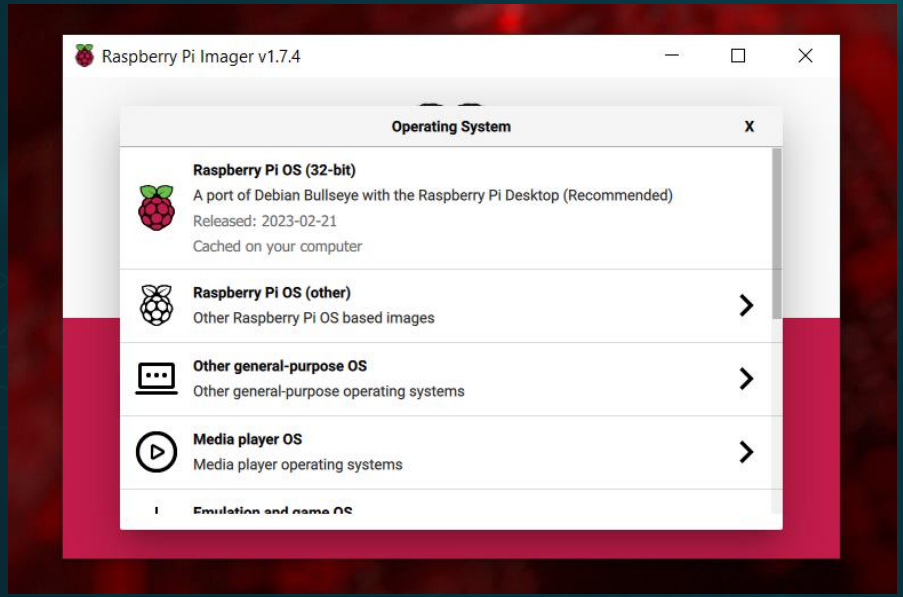
Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. Watch our 45-second video to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

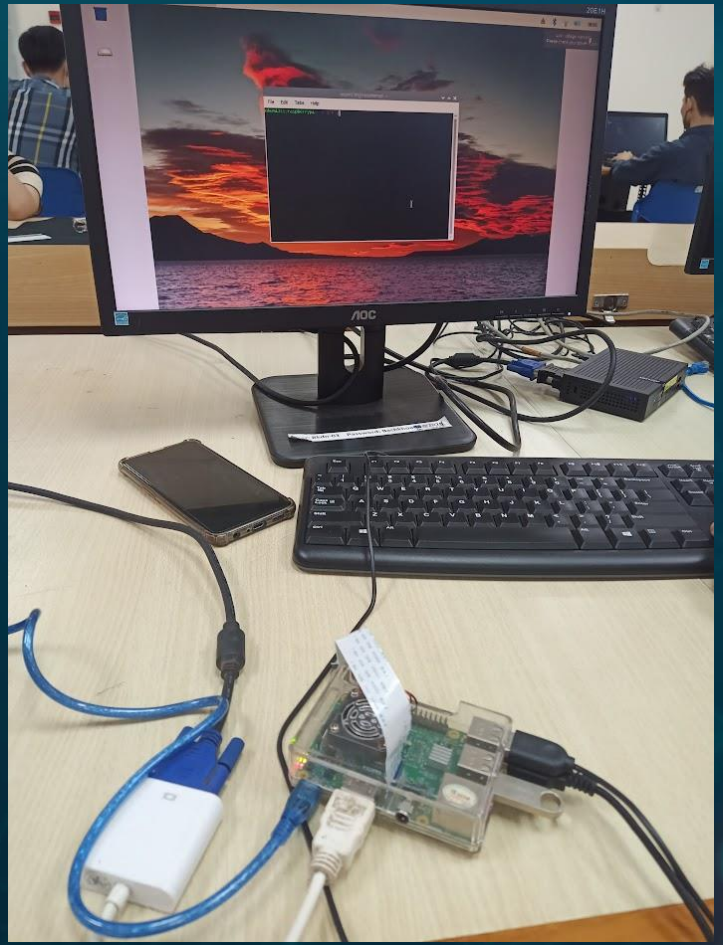
Download for Windows

Cài đặt hệ điều hành cho Raspberry:

Import hệ điều hành vào thẻ nhớ thông qua đầu đọc



Kết quả quá của quá trình cài đặt



Triển khai các API cho front-end

Phần 2

Bổ sung thêm một vài trường vào bảng Nhân viên

```python
class User(models.Model):

    id = models.AutoField(primary_key=True)
    email = models.EmailField(unique=True)
    role = models.CharField(max_length=5, default="user")
    password = models.CharField(max_length=255)
    fullname = models.CharField(max_length=255, null=True, blank=True)
    url_video = models.FileField(upload_to='static/videos/', null=True, blank=True)
    phone = models.CharField(max_length=20, null=True, blank=True)
    create_at = models.DateTimeField(auto_now_add=True, null=True, blank=True)
    update_at = models.DateTimeField(auto_now=True, null=True, blank=True)


    def save(self, *args, **kwargs):
        if not self.pk:
            self.password = hash_password(self.password)
        else:
            existing_user = User.objects.get(pk=self.pk)
            if existing_user.password != self.password:
                self.password = hash_password_if_changed(self.password)
        super(User, self).save(*args, **kwargs)


    def __str__(self):
        return self.id
```

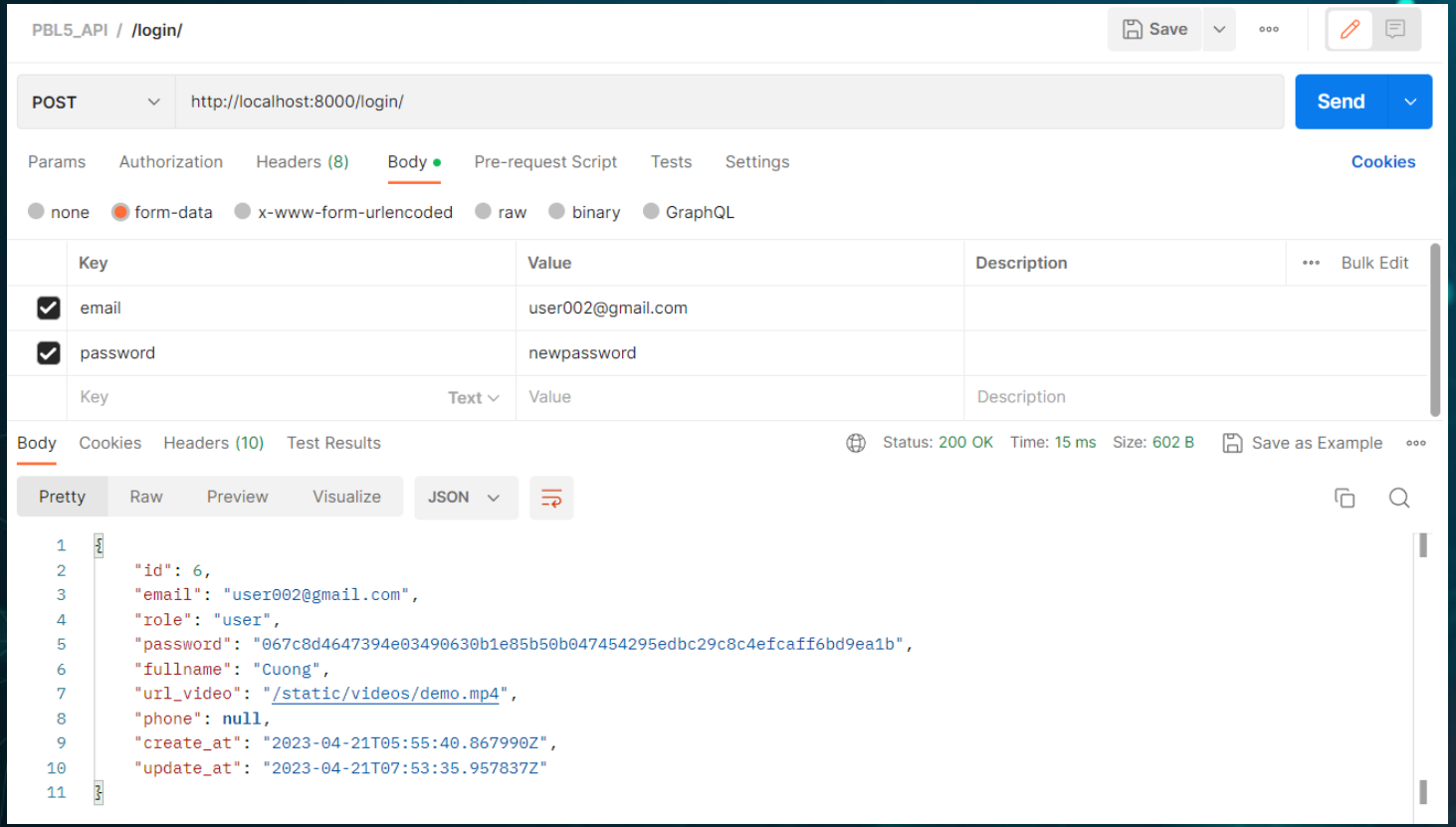
Xây dựng thêm bảng chứa các Encode của mỗi nhân viên

```
48
49  class Encode(models.Model):
50      id = models.AutoField(primary_key=True)
51      id_user = models.ForeignKey(User, on_delete=models.CASCADE)
52      encode = models.TextField()
53
54      def get_encode_list(self):
55          # Chuyển đổi chuỗi thành danh sách các giá trị số nguyên
56          return [int(x) for x in self.encode.split(',')]
57
58      def __str__(self):
59          return str(self.id)
60
```

Xây dựng API đăng nhập tài khoản cho nhân viên

```python
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from django.contrib.auth import authenticate

class LoginView(APIView):
    def post(self, request):
        email = request.data.get('email')
        password = request.data.get('password')

        try:
            user = User.objects.get(email=email)
        except User.DoesNotExist:
            return Response({'error': 'User does not exist'}, status=status.HTTP_401_UNAUTHORIZED)

        if hash_password(password) == user.password:
            serializer = UserSerializer(user)
            return Response(serializer.data)
        else:
            return Response({'error': 'Wrong password'}, status=status.HTTP_401_UNAUTHORIZED)
```

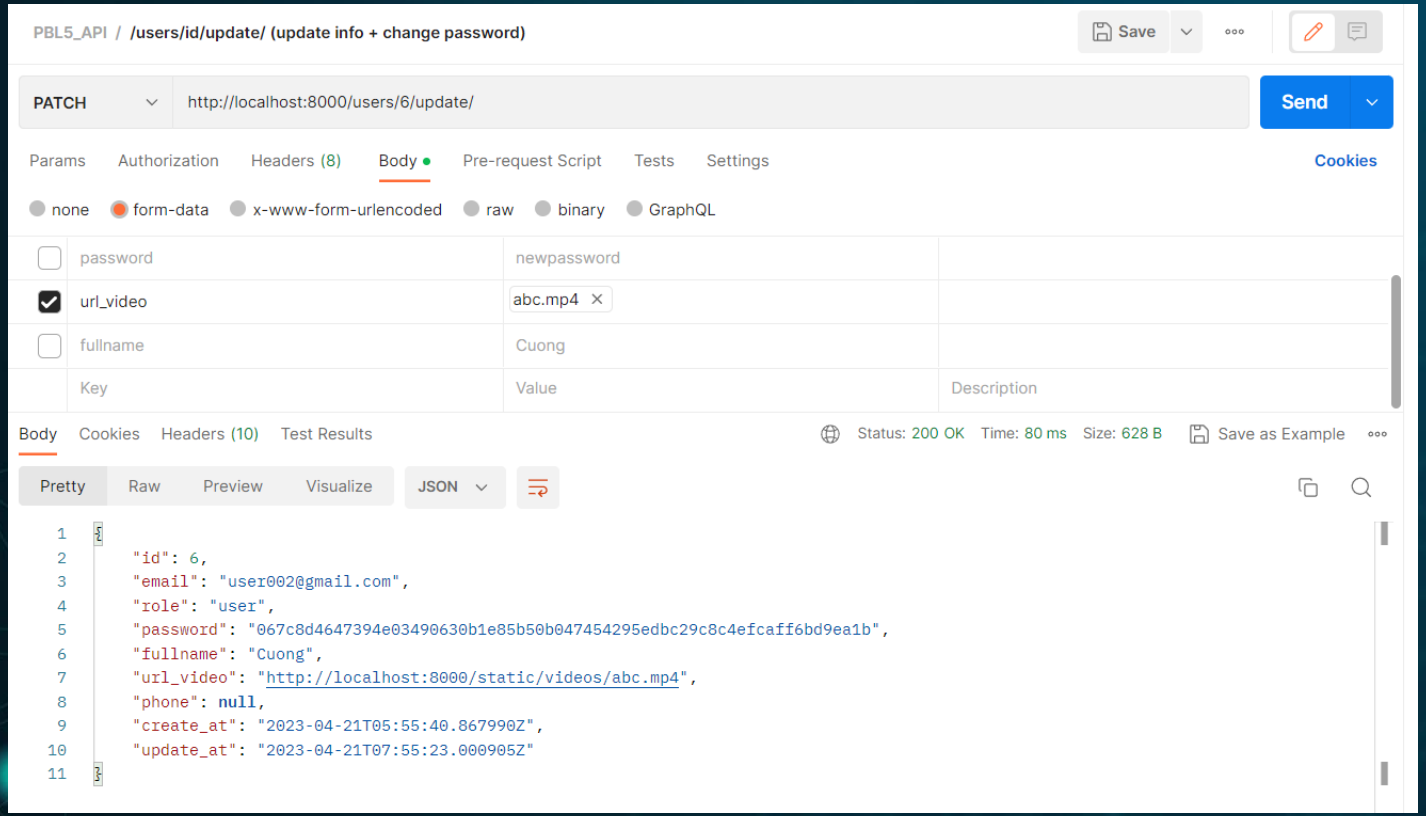
Khi client gửi tài khoản + mật khẩu chính xác, API sẽ trả về thông tin của nhân viên



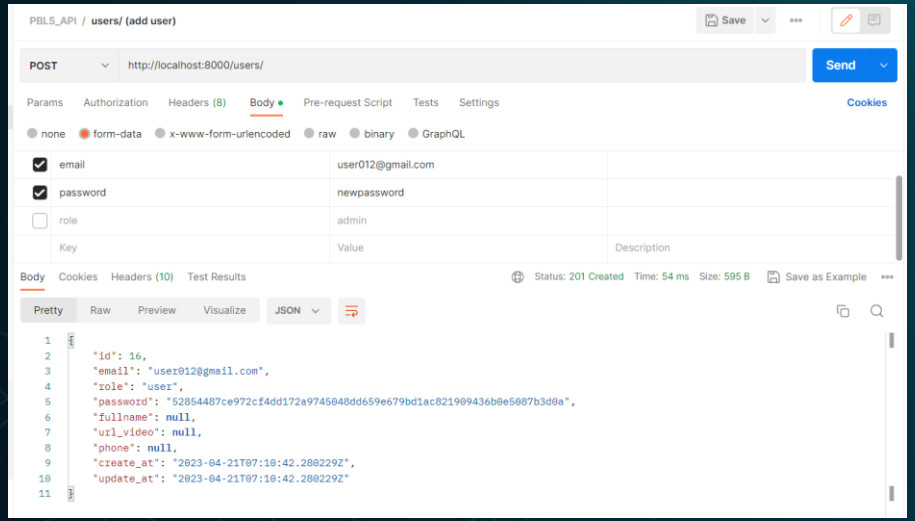
Xây dựng API cập nhập thông tin nhân viên

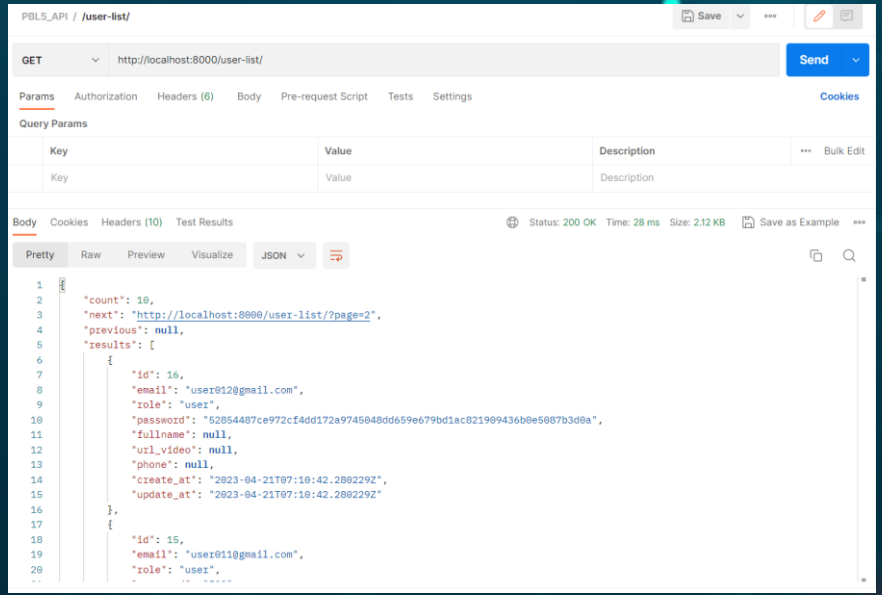
```python
import os
from django.conf import settings

class UserUpdateAPIView(generics.UpdateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    def patch(self, request, *args, **kwargs):
        instance = self.get_object()
        serializer = self.get_serializer(instance, data=request.data, partial=True)
        serializer.is_valid(raise_exception=True)

        if 'url_video' in request.data:
            # Xóa video cũ
            if instance.url_video:
                path = instance.url_video.path
                if os.path.isfile(os.path.join(settings.MEDIA_ROOT, path)):
                    os.remove(os.path.join(settings.MEDIA_ROOT, path))

            # Lưu video mới
            instance.url_video = request.data['url_video']


        serializer.save()
        return Response(serializer.data)
```

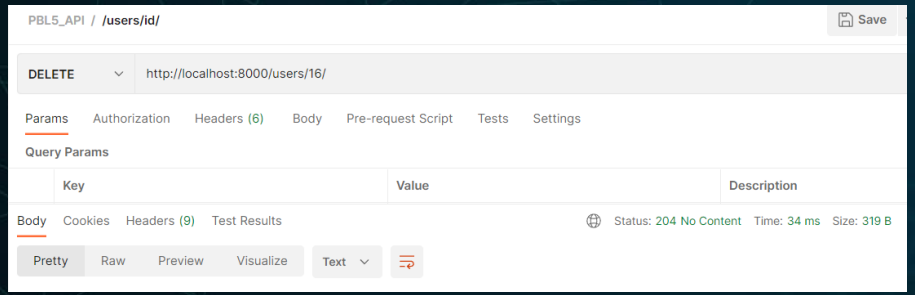
Thông tin nhân viên sẽ được cập nhập qua phương thức PATCH từ Client



Một số API khác







Add Staff, Get List Staff, Del Staff, …

Tài liệu tham khảo:

www.raspberrypi.com/software/

chat.openai.com

www.django-rest-framework.org

Thank for watching

Xây dựng Model nhận diện khuôn mặt

03

Nguyễn Văn Hoàng Phúc – Trần Thanh Nguyên

Phần thuật toán:

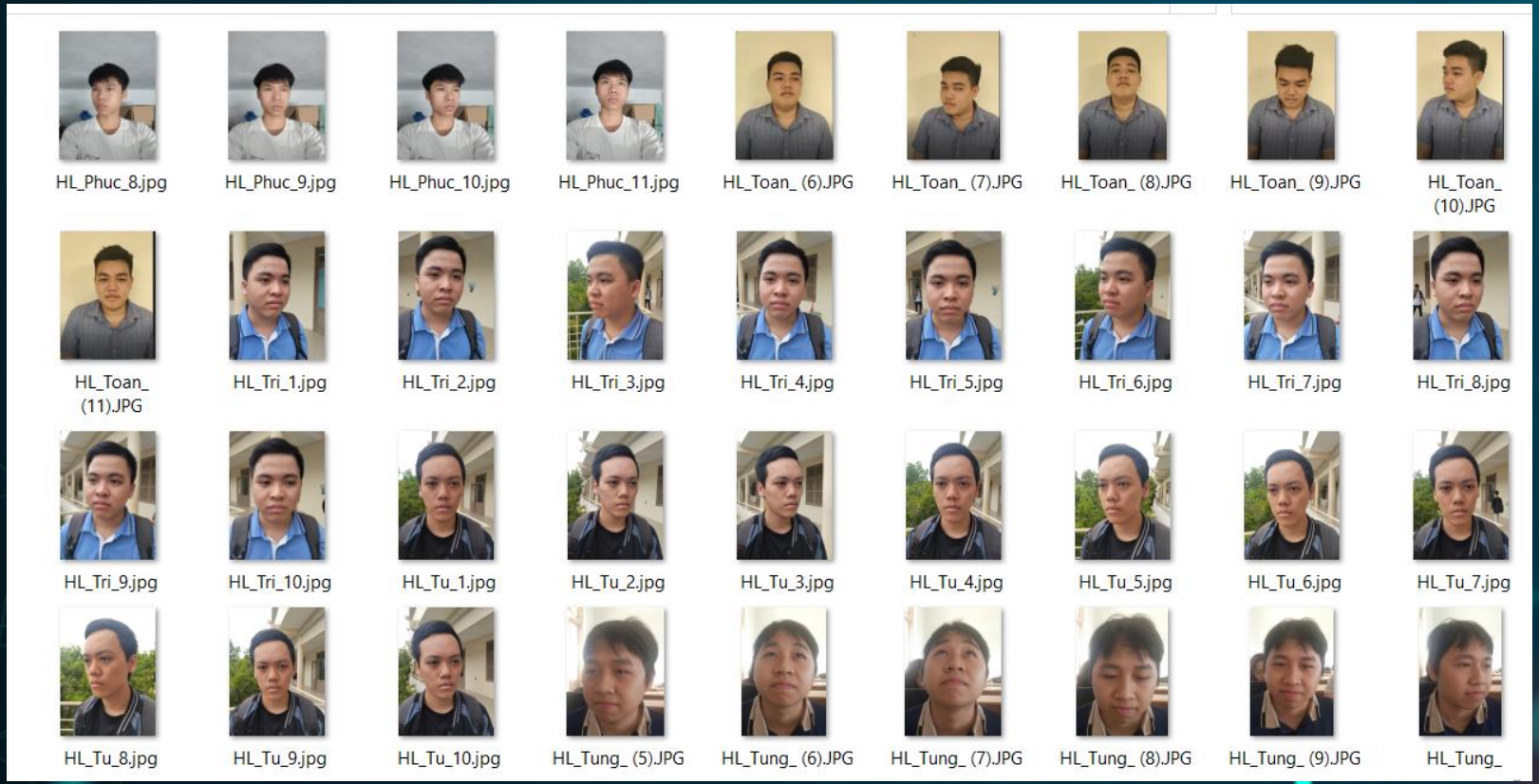
Đầu vào:

Tập dữ liệu huấn luyện (các khuôn mặt được chọn làm gốc để nhận diện)
Tập dữ liệu kiểm thử (các khuôn mặt lấy ra từ camera cần nhận diện)

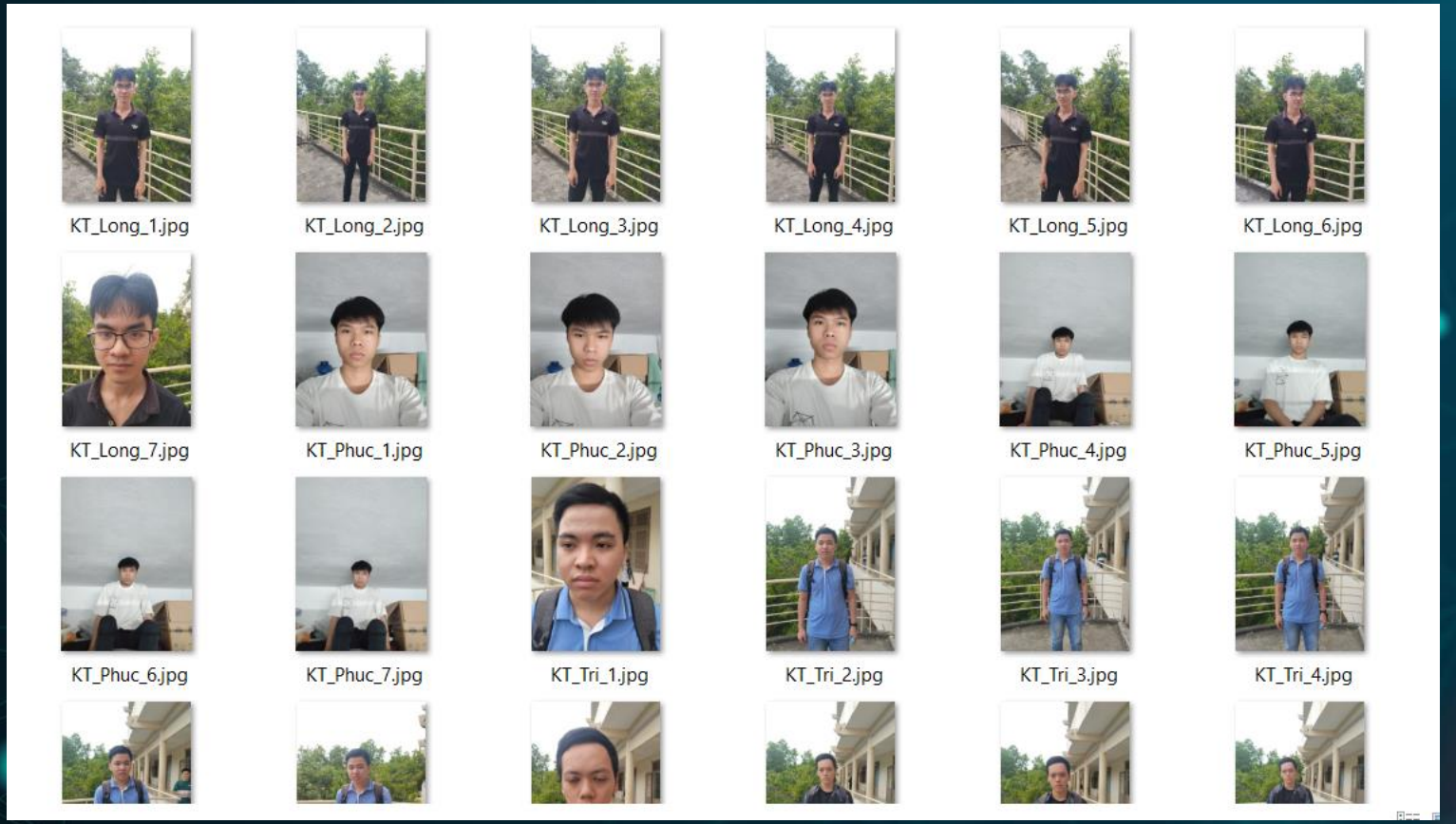
Đầu ra:

Kết quả nhận diện trên tập dữ liệu kiểm thử
Hiệu suất nhận diện bao nhiều phần trăm

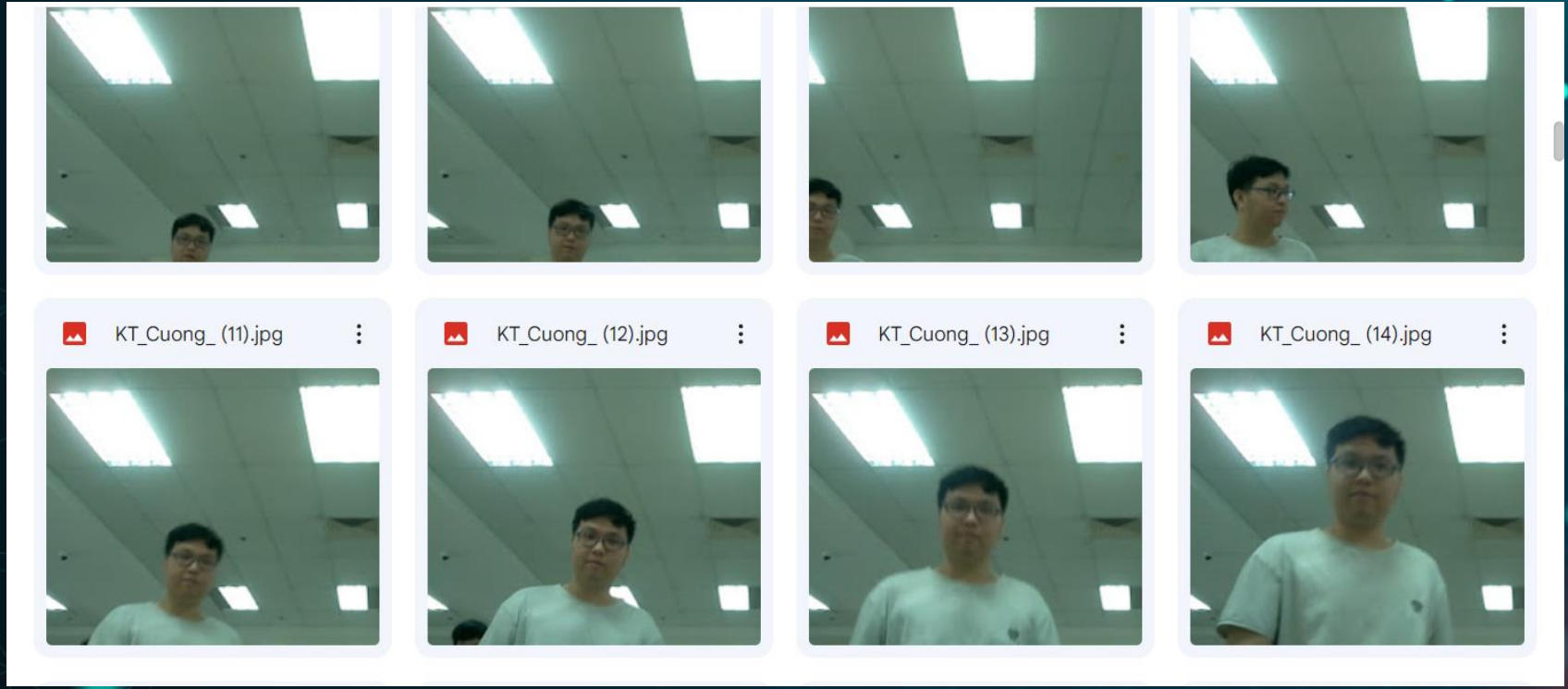
Tập dữ liệu train



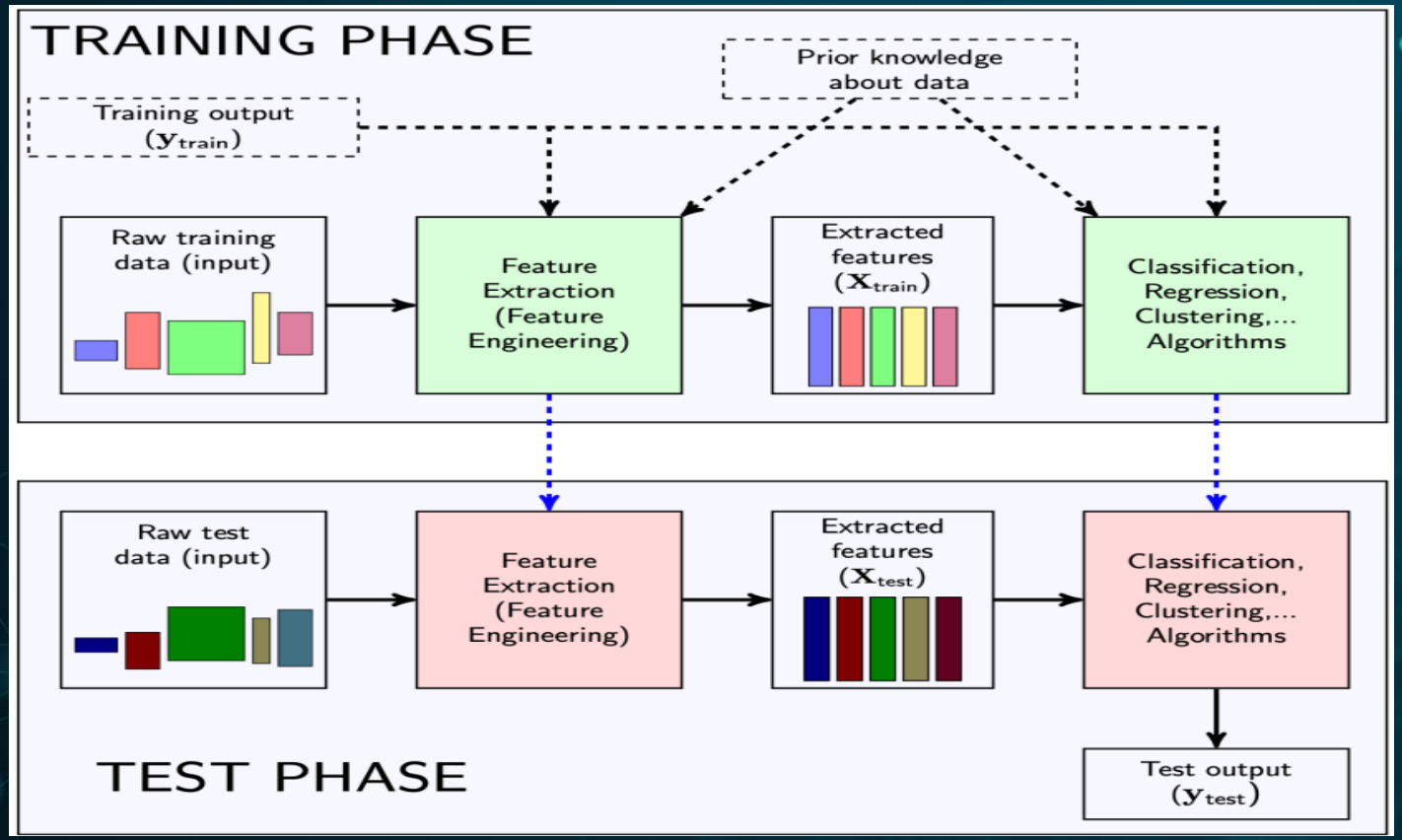
Tập dữ liệu test



Tập dữ liệu test



Mô hình chung cho các bài toán học máy



Phần thuật toán:

Trích xuất đặc trưng: Hai mô hình đơn giản trong xử lí ảnh và nhận dạng đối tượng được sử dụng là LBP (Local Binary Patterns) và HoG (Histogram of Oriented Gradients).

- LBP là một phương pháp đặc trưng đơn giản, dựa trên việc so sánh các giá trị pixel trong vùng lân cận của mỗi điểm ảnh với giá trị trung tâm của vùng đó. Kết quả của quá trình so sánh này được biểu diễn dưới dạng một chuỗi nhị phân, gọi là mẫu LBP. Mẫu LBP được sử dụng để mô tả tính năng cục bộ của ảnh và có thể được sử dụng để nhận dạng đối tượng.
- HoG là một phương pháp trích xuất đặc trưng dựa trên việc tính toán bộ mô tả hướng của các cạnh trong ảnh. HoG được sử dụng để phát hiện đối tượng trong ảnh bằng cách so sánh bộ mô tả của ảnh với bộ mô tả của các đối tượng đã biết trước và tìm kiếm sự khớp nhau.

Phần thuật toán:

Đối với mỗi mô hình ta thực hiện các phương pháp tiền xử lí dữ liệu khác nhau:

- Chuyển đổi định dạng dữ liệu (gray scale): Vì màu sắc không quyết định đến đặc trưng được trích xuất từ các mô hình LBP, HoG, nên chiều màu sắc sẽ bị loại bỏ trở thành ảnh trắng đen.
- Loại bỏ nhiễu (xử lí ngoại lệ): Sử dụng các phương pháp loại bỏ nhiễu như Gaussian Blur, Median Blur, Bilateral Filtering.
- Chuẩn hóa dữ liệu: Hiện tại màu sắc của các bức ảnh sẽ là trắng đen tuy nhiên mức độ đậm nhạt chưa thống nhất nên sẽ chuẩn hóa mức độ đậm nhạt về mức min max (min: trắng , max: đen)

Tiền sử lí dữ liệu

Mỗi hình ảnh trong tập dữ liệu đầu vào sẽ được:

- Đọc ra với hệ màu trắng đen:

Hiện tại đọc ra hệ màu cv2.IMREAD_GRAYSCALE

- Xác định tọa độ khuôn mặt và cắt hình ảnh khuôn mặt

Detect face bằng hàm face_locations trong thư viện.
Cắt ảnh tương tự như cắt mảng 2 chiều.

- Chuẩn hóa khuôn mặt về kích thước thích hợp

Hiện tại chọn được kích thước là 64*64 cho mỗi khuôn mặt

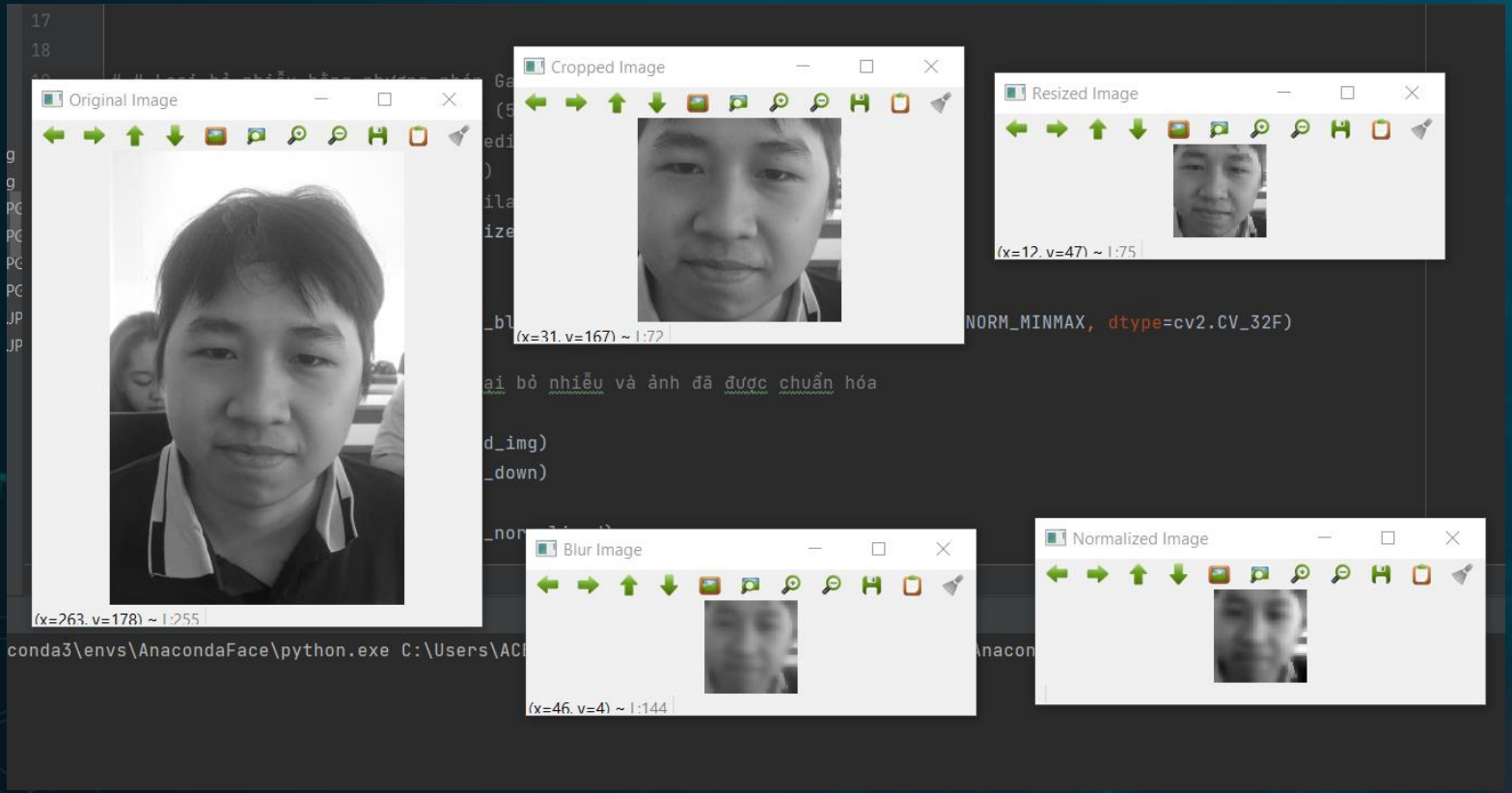
- Xử lí ngoại lệ, làm mờ ảnh

GaussianBlur, medianBlur, bilateralFilter.

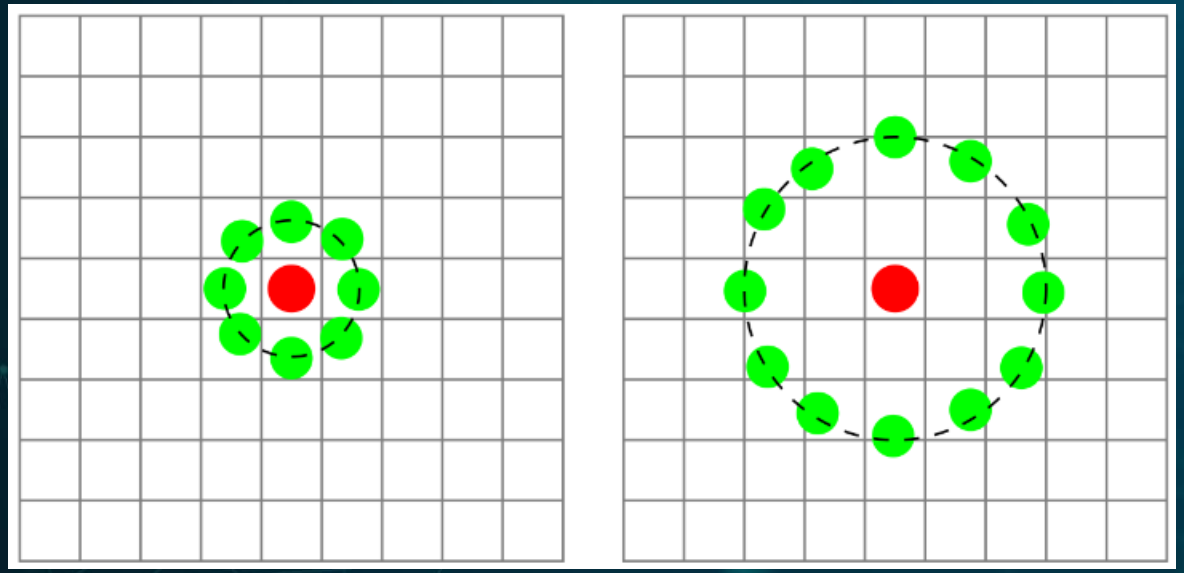
- Chuẩn hóa dữ liệu

image = cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)

Giai đoạn xử lí tiền dữ liệu



Sử dụng LBP để trích xuất đặc trưng



D = 3, góc = 45 độ

D = 5, góc = 30 độ

Mỗi pixel (điểm ảnh) sẽ có một giá trị duy nhất thể hiện độ xám của điểm đó (0-255). Chúng ta đi so sánh giá trị của các điểm ảnh lân cận với điểm trung tâm để thu được giá trị của điểm ảnh LBP.

Sử dụng LBP để trích xuất đặc trưng

ảnh gốc

LBP Lib

My LBP

Sử dụng LBP để trích xuất đặc trưng

```
# đã có LBP_img
hist, _ = np.histogram(out_our, bins=256)
# chuan hoa
hist = np.float32(hist) / np.sum(hist)
```

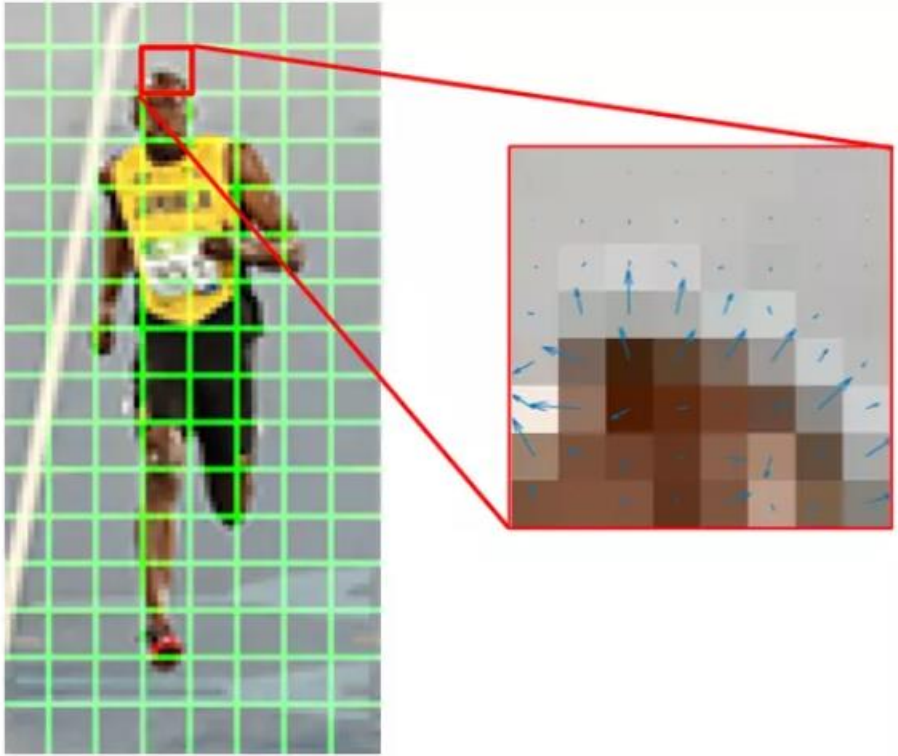
Như vậy ta có việc trích xuất đặc trưng bằng LBP thực chất là thống kê histogram của các điểm ảnh trên LBP image.

Ta có một siêu tham số đặc trưng cho LBP là bins – số khoảng để chia các điểm ảnh trong LBP image

Sử dụng HoG để trích xuất đặc trưng

```
    # Tính gradient
    gx, gy = gradient(image)

    # Tính độ lớn và hướng của gradient trung bình trong mỗi ô pixel
    magnitude, angle = magnitude_orientation(gx, gy)

    # Tính histogram của hướng gradient trong mỗi ô pixel
    histogram = orientation_histogram(magnitude, angle, orientations, pixels
_per_cell, cells_per_block)

    # Tính đặc trưng HOG
    hog_feature = histogram.ravel()
    # print(hog_feature)
    return hog_feature
```

Sử dụng HoG để trích xuất đặc trưng



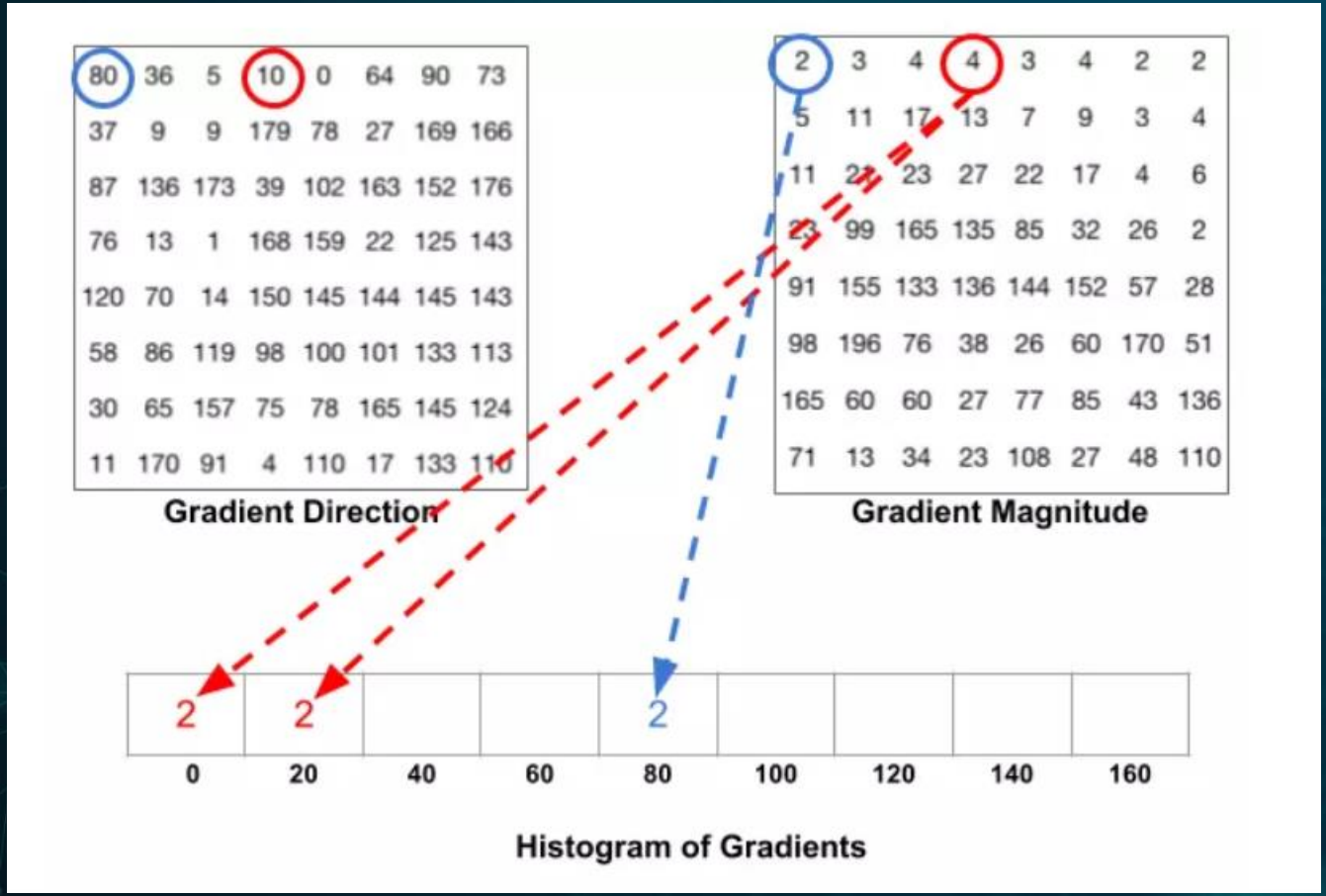
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Sử dụng HoG để trích xuất đặc trưng



Các mô hình phân cụm dữ liệu

01 **SVM**
Principal Component Analysis

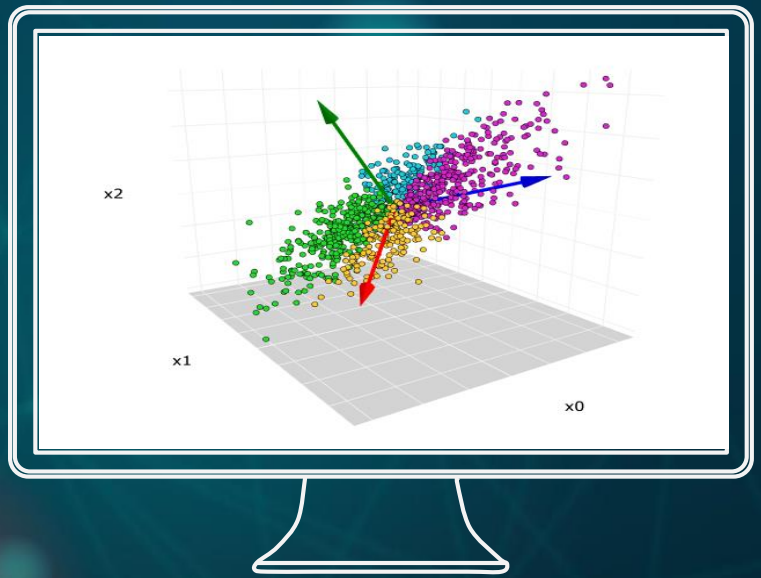
02 **LDA**
Linear Discriminant Analysis

03 **KNN**
k-Nearest Neighbor

04 **RandomForest**

05 **RandomForest**

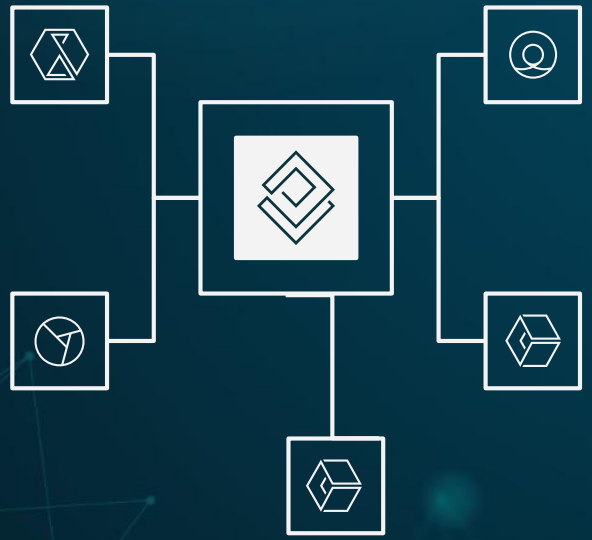
Giảm chiều dữ liệu - Phương pháp PCA



PCA(Principal Component Analysis) : là phương pháp chọn ra những đặt trưng quan trọng

Ma trận đặt trưng hiện tại của các ảnh là ma trận 1 x 128 (vẫn còn khá lớn)

Các bước của PCA



Tính toán ma trận hiệp phương sai:

Tính toán các thành phần chính

Lựa chọn số lượng thành phần chính

Tính toán ma trận trọng số

Biểu diễn dữ liệu mới

Hiện tại có nhiều thư viện hỗ trợ PCA : như numpy , scikit-learn pandas...

KNN (k-Nearest Neighbor)

Định nghĩa : là một thuật toán học có giám sát dùng để phân loại hoặc dự đoán dữ liệu mới dựa trên các điểm dữ liệu đã biết.

Sử dụng phương pháp : tính khoảng cách , dựa vào nhãn các k điểm gần nhất để đưa ra dự đoán nhãn của điểm này

Thư viện sciki-learn: có hàm kneighbersclassified hỗ trợ tính toán

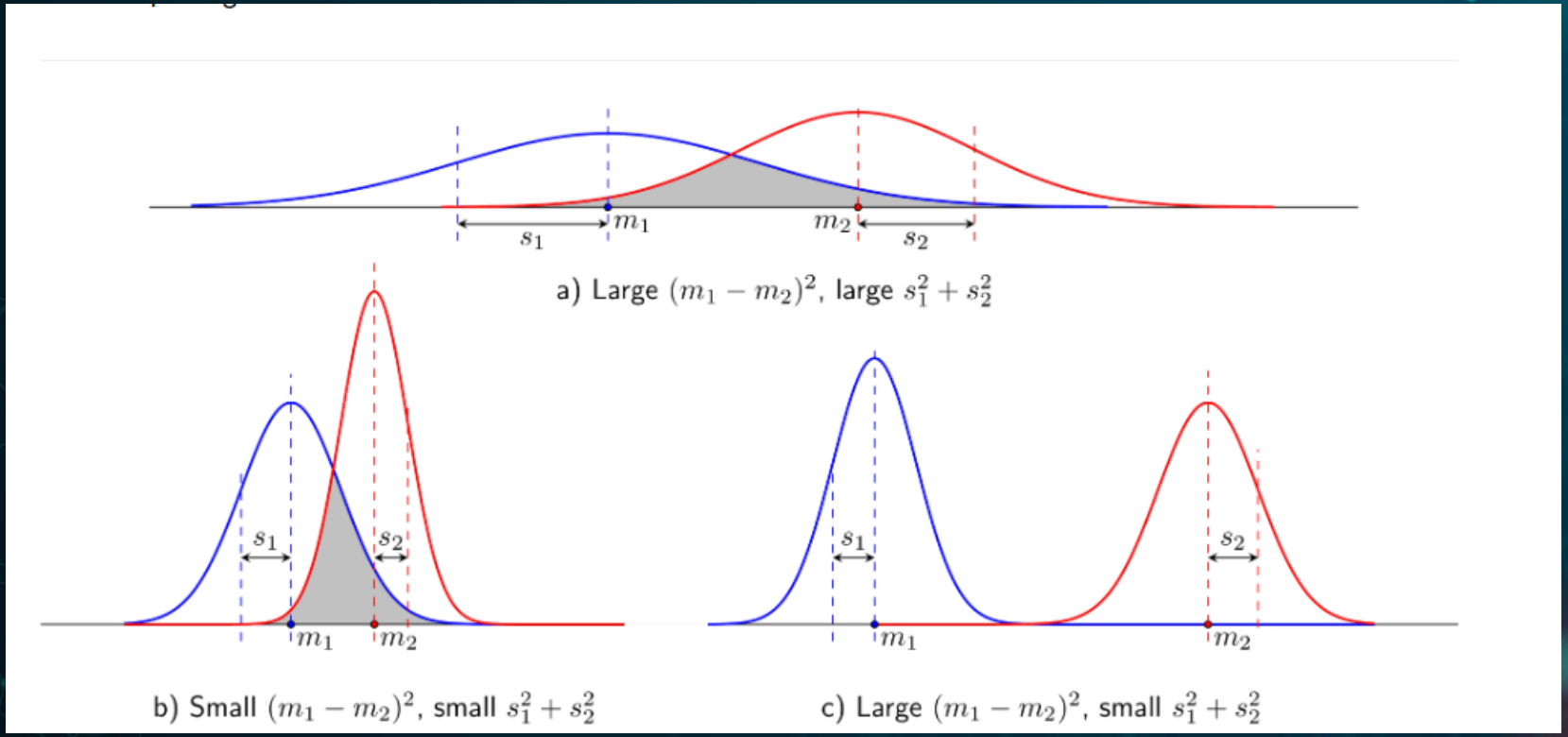
KNN (k-Nearest Neighbor)

Định nghĩa : là một thuật toán học có giám sát dùng để phân loại hoặc dự đoán dữ liệu mới dựa trên các điểm dữ liệu đã biết.

Sử dụng phương pháp : tính khoảng cách , dựa vào nhãn các k điểm gần nhất để đưa ra dự đoán nhãn của điểm này

Thư viện sciki-learn: có hàm kneighbersclassified hỗ trợ tính toán

LDA (Linear Discriminant Analysis)



LDA (Linear Discriminant Analysis)

Có thể bạn đang tự hỏi, độ lệch chuẩn và khoảng cách giữa hai kỳ vọng đại diện cho các tiêu chí gì:

- Như đã nói, độ lệch chuẩn nhỏ thể hiện việc dữ liệu ít phân tán. Điều này có nghĩa là dữ liệu trong mỗi class có xu hướng giống nhau. Hai phương sai s1 bình phương và s2 bình phương còn được gọi là các **within-class variances**.
- Khoảng cách giữa các kỳ vọng là lớn chứng tỏ rằng hai classes nằm xa nhau, tức dữ liệu giữa các classes là khác nhau nhiều. Bình phương khoảng cách giữa hai kỳ vọng (m1−m2) bình phương còn được gọi là **between-class variance**.

LDA là thuật toán đi tìm giá trị lớn nhất của hàm mục tiêu:

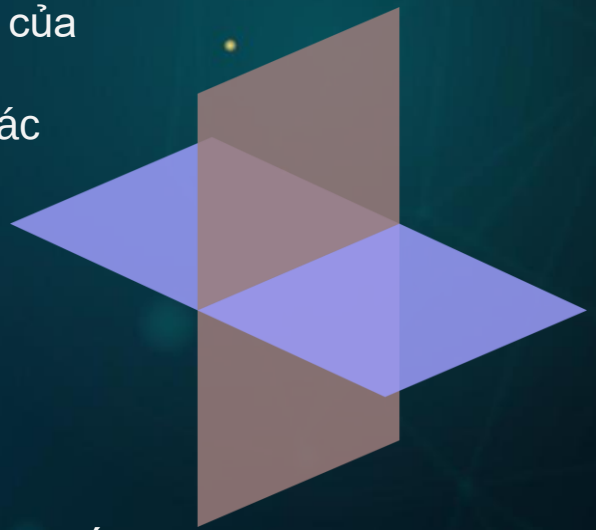
$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \qquad (4)$$

LDA (Linear Discriminant Analysis)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import numpy as np

# Các vector đặc trưng của các khuôn mặt đã biết
known_faces = np.array(encodeListKnow)

# Nhãn của các khuôn mặt đã biết
known_labels = np.array(classnames)

# Các vector đặc trưng của các khuôn mặt chưa biết
unknown_faces = np.array(encodeListUnknow)

# Khởi tạo mô hình LDA
lda = LinearDiscriminantAnalysis()

# Huấn luyện mô hình LDA với các vector đặc trưng đã biết và nhãn tương ứng
lda.fit(known_faces, known_labels)

# Dự đoán nhãn của các khuôn mặt chưa biết bằng cách sử dụng mô hình LDA đã huấn luyện
predicted_labels = lda.predict(unknown_faces)

# In ra các nhãn được dự đoán
print(predicted_labels)
```

SVM (Support Vector Machine)

Định nghĩa : là một thuật toán học có giám sát được sử dụng cho các bài toán phân loại và hồi quy. Cơ chế hoạt động của SVM được thực hiện bằng cách tìm một siêu mặt phẳng (hyperplane) phân chia tốt nhất các điểm dữ liệu thuộc các lớp khác nhau

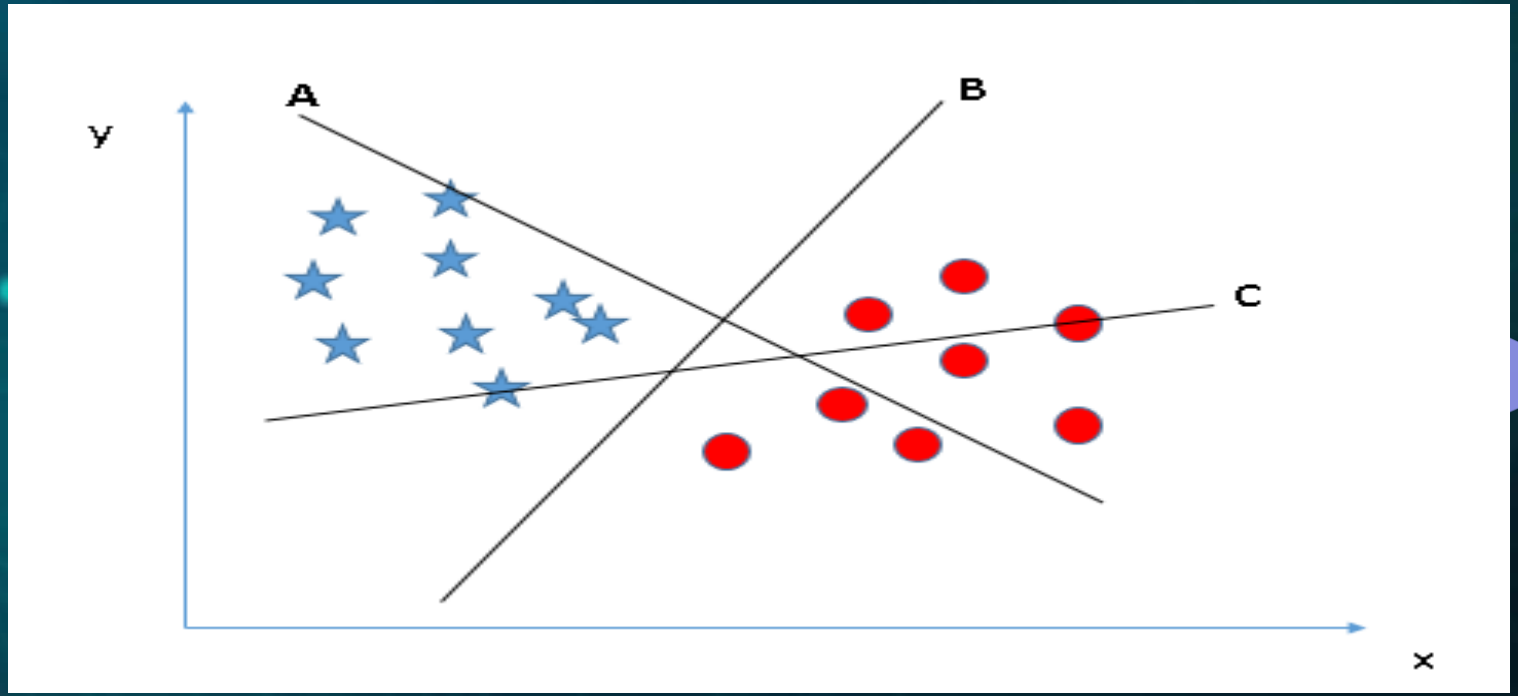


Các bước tiến hành :

- Lựa chọn kernel function và các siêu tham số (hyperparameters) cho SVM.
- Huấn luyện SVM trên dữ liệu huấn luyện.
- Đánh giá hiệu suất của SVM trên dữ liệu kiểm tra.
- Tinh chỉnh các siêu tham số của SVM để cải thiện hiệu suất.

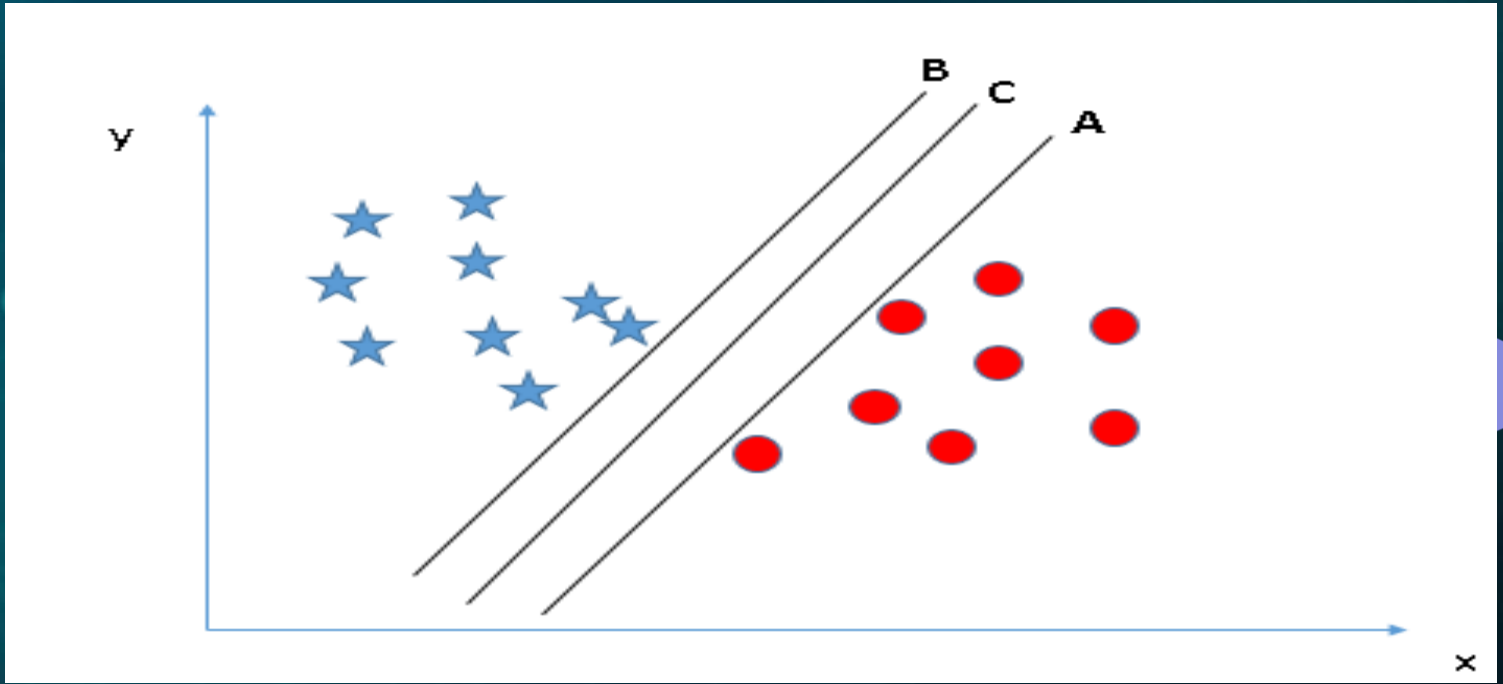
Thư viện sciki-learn: có hàm SVC hỗ trợ tính toán

SVM (Support Vector Machine)



Quy tắc 1 :Chọn một hyper-plane để phân chia hai lớp tốt nhất

SVM (Support Vector Machine)



Quy tắc thứ hai chính là xác định khoảng cách lớn nhất từ điểu gần nhất của một lớp nào đó đến đường hyper-plane

Thống kê hiệu xuất nhận diện

134 train 81 test	LBP		LBPLib	
	128	256	128	256
LDA	27.848101265822784% (Th=0.01)	26.582278481012654 % (th=0.01)	28.0517471284723%(Th=0.01)	27.14914823294% (th=0.01)
Distance	30.37974683544304% (Th =0.04)	43.04%	31.7491274182332%(Th=0.04)	42.72%
Distance_Mean	26.582278481012654% (Th=0.06)	35.44303797468354% (Th=0.06)	26.6481495132324 % (Th=0.06)	36.19%
KNN (k=3)	29.11%	35.44%	30.13%	36.19%
SVM (Threadhold = 16)	13.92%	13.92%	13.92%	13.92%
Gradient Booting	20.25%	30.38%	21.17%	29.20%
Random Forest	29.11%	29.11%	30.24%	30.24%
	HoG		HoG Lib	
	block = (3,3)	block = (1,1)	block = (3,3)	block = (1,1)
LDA	60.76%	46.84%	61.13%	47.67%
Distance	60.75949367088608% (Th=1.1)	53.16455696202531% (Th=2.1)	61.1328471923479% (Th=1.1)	54.189429413235% (Th= 2.1)
Distance_Mean	69.62025316455697%(Th=1.1)	65.82278481012658% (Th=2.1)	68.1749834194382% (Th=1.1)	68.187414824142% (Th= 2.1)
KNN (k=3)	53.16%	53.16%	54.85%	54.85%
SVM (Threadhold = 16)	65.82278481012658% (Th=16)	73.42%	65.164914924921% (Th=16)	72.37%
Gradient Booting	49.37%	48.10%	50.18%	49.32%
Random Forest	74.68%	62.03%	74.19%	62.92%

Thống kê hiệu xuất nhận diện

	LBP+256			
	không chuẩn hóa dữ liệu			
	không xử lí ngoại lệ	xử lí Gaussian Blur	xử lí Median Blur	Bilateral Filtering
LDA	26.582278481012654% (th=0.01)	35.44%	36.71%	31.65%
Distance	43.04%	53.16%	59.49%	54.43%
Distance_Mean	35.44303797468354% (Th=0.06)	53.16%	56.96%	58.23%
KNN (k=3)	35.44%	41.77%	56.96%	54.43%
SVM (Threadhold = 1(	13.92%	13.92%	13.92%	13.92%
Gradient Booting	30.38%	40.51%	31.65%	30.38%
Random Forest	29.11%	58.23%	58.23%	45.57%
	chuẩn hóa dữ liệu			
	không xử lí ngoại lệ	xử lí Gaussian Blur	xử lí Median Blur	Bilateral Filtering
LDA	22.78%	31.65%	37.97%	54.43%
Distance	43.04%	55.70%	58.23%	58.23%
Distance_Mean	32.91%	55.70%	56.96%	24.05%
KNN (k=3)	36.71%	46.84%	55.70%	54.43%
SVM (Threadhold = 1(	13.92%	13.92%	13.92%	13.92%
Gradient Booting	32.91%	36.71%	58.23%	24.05%
Random Forest	27.85%	63.29%	53.16%	43.04%

Thống kê hiệu xuất nhận diện

	HoG+(3,3)			
	không chuẩn hóa dữ liệu			
	không xử lí ngoại lệ	xử lí Gaussian Blur	xử lí Median Blur	Bilateral Filtering
LDA	60.76%	62.03%	64.56%	58.23%
Distance	60.75949367088608% (Th=1.1	55.69620253164557% (Th=1.3)	58.23%	63.29%
Distance_Mean	69.62025316455697%(Th=1.1)	64.55696202531645% (Th=1.4)	63.29%	67.09%
KNN (k=3)	53.16%	59.49%	58.23%	58.23%
SVM (Threadhold = 1(	65.82278481012658% (Th=16)	68.35443037974683% (Th=16)	70.89%	68.35%
Gradient Booting	49.37%	39.24%	48.10%	39.24%
Random Forest	74.68%	68.35443037974683% (Th=16)	68.35%	59.49%
	chuẩn hóa dữ liệu			
	không xử lí ngoại lệ	xử lí Gaussian Blur	xử lí Median Blur	Bilateral Filtering
LDA	59.49%	60.76%	64.56%	56.96%
Distance	56.96%	55.70%	58.23%	63.29%
Distance_Mean	65.82%	64.56%	63.29%	67.09%
KNN (k=3)	54.43%	58.23%	59.49%	58.23%
SVM (Threadhold = 1(	65.82%	68.35%	70.89%	68.35%
Gradient Booting	37.97%	37.97%	41.77%	39.24%
Random Forest	73.42%	64.56%	72.15%	63.29%

Kết luận

So sánh mô hình LBP vs mô hình HoG:

- Ta có thể thấy rõ sự chênh lệch hiệu xuất của hai mô hình LBP với mô hình HoG (đối với LBP thì hiệu xuất cải thiện tối đa là 63.29% trên tập kiểm thử còn mô hình HoG có hiệu xuất lên đến 74.68%)
=> Bản chất của 2 mô hình có sự khác biệt rõ rệt

- Việc sử dụng các phương pháp xử lí tiền dữ liệu cho hiệu quả cao ở mô hình LBP nhưng lại không mang lại hiệu quả đáng kể ở mô hình HoG
=> Do LBP dễ bị ảnh hưởng bởi nhiễu và các giá trị cần được chuẩn hóa hơn là HoG

Tài liệu tham khảo:

Tìm hiểu phương pháp LBP:

https://minhng.info/tutorials/local-binary-patterns-lbp-opencv.html#:~:text=Local%20Binary%20Patterns%20(hay%20c%C3%B2n,m%C3%A1y%20%C4%91%E1%BB%83%20h%E1%BB%8Dc%20%2F%20ph%C3%A2n%20lo%E1%BA%A1i.

Tìm hiểu phương pháp HoG:

https://viblo.asia/p/tim-hieu-ve-phuong-phap-mo-ta-dac-trung-hog-histogram-of-oriented-gradients-V3m5WAwxZO7

Các phương pháp tiền xử lí dữ liệu + classification:

Tài liệu học tập môn KHDL, Trí tuệ nhân tạo