# Tài liệu thực hành buổi 05

## Môn : Thị giác máy tính

## I. Mục đích:

- Nén ảnh dùng phương pháp SVD.
- Nhận dạng ảnh dùng phương pháp k-NN.
- Nhận dạng ảnh sử dụng YOLOv9.

## II. Nội dung:

### 1. Cài đặt và import các thư viện cần thiết:

```
import numpy as np
import scipy
import scipy.linalg
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
from skimage import io
import numpy as np
from skimage import io
from skimage import img_as_float
```

### 2. Nén ảnh:

Nén ảnh để giảm kích thước bộ nhớ dùng để lưu trữ ảnh. Một trong các phương pháp nén ảnh tránh mất thông tin là sử dụng giá trị kỳ dị (SVD) lên ảnh và giữ lại n giá trị kỳ dị đầu tiên (lớn nhất).

Ta sẽ sử dụng SVD để nén ảnh trên.

**Các bước thực hiện:**

- Đọc ảnh đầu vào.

```
image = io.imread('pitbull.jpg', as_gray=True)
plt.imshow(image)
plt.axis('off')
plt.show()
```

- Tính SVD của ảnh và giữ lại n kỳ dị lớn nhất cùng với vectors kỳ dị tương ứng.

```python
def compress_image(image, num_values):
    """Compress an image using SVD and keeping the top `num_values` singular values.

    Args:
        image: numpy array of shape (H, W)
        num_values: number of singular values to keep

    Returns:
        compressed_image: numpy array of shape (H, W) containing the compressed image
        compressed_size: size of the compressed image
    """
    compressed_image = None
    compressed_size = 0

    # YOUR CODE HERE
    # Steps:
    #     1. Get SVD of the image
    #     2. Only keep the top `num_values` singular values, and compute `compressed_image`
    #     3. Compute the compressed size
    # END YOUR CODE

    assert compressed_image.shape == image.shape, \
           "Compressed image and original image don't have the same shape"

    assert compressed_size > 0, "Don't forget to compute compressed_size"

    return compressed_image, compressed_size
```

(Hoàn thành đoạn code trên)

- Dựng lại ảnh.

```python
# Compress the image using `n` singular values
n = 10
compressed_image, compressed_size = compress_image(image, n)

compression_ratio = compressed_size / image.size

print("Data size (original): %d" % (image.size))
print("Data size (compressed): %d" % compressed_size)
print("Compression ratio: %f" % (compression_ratio))



plt.imshow(compressed_image, cmap='gray')
title = "n = %s" % n
plt.title(title)
plt.axis('off')
plt.show()
```

Kết quả thu được với n = 10 và n = 50:



### 3. Nhận dạng ảnh mặt người sử dụng phương pháp k-NN:

#### a) Giới thiệu về tập dữ liệu:

Ta sẽ sử dụng tập dữ liệu mặt người (của những người nổi tiếng).

Cấu trúc cây thư mục như sau:

```
faces/
    train/
        angelina jolie/
        anne hathaway/
        ...
    test/
        angelina jolie/
        anne hathaway/
        ...
```

Tập dữ liệu bao gồm 16 lớp. Mỗi lớp có 50 ảnh dùng để huấn luyện và 10 ảnh để kiểm tra.

Tất cả các ảnh đều có kích thước 64x64.

#### b) Tiền xử lý dữ liệu:

1) Load dữ liệu:

```python
def load_dataset(data_dir, train=True, as_grey=False, shuffle=True):
    """ Load faces dataset
    Args:
        data_dir - Directory containing the face datset.
        train - If True, load training data. Load test data otherwise.
        as_grey - If True, open images as grayscale.
        shuffle - shuffle dataset

    Returns:
        X - array of N images (N, 64, 64, 3)
        y - array of class labels (N,)
        class_names - list of class names (string)
    """
    y = []
    X = []
    class_names = []

    if train:
        data_dir = os.path.join(data_dir, 'train')
    else:
        data_dir = os.path.join(data_dir, 'test')

    for i, cls in enumerate(sorted(os.listdir(data_dir))):
        for img_file in os.listdir(os.path.join(data_dir, cls)):
            img_path = os.path.join(data_dir, cls, img_file)
            img = img_as_float(io.imread(img_path, as_grey=as_grey))
            X.append(img)
            y.append(i)
        class_names.append(cls)

    # Convert list of imgs and labels into array
    X = np.array(X)
    y = np.array(y)

    if shuffle:
        idxs = np.arange(len(y))
        np.random.shuffle(idxs)
        X = X[idxs]
        y = y[idxs]

    return np.array(X), np.array(y), class_names
```

Sau khi sử dụng hàm load_dataset dữ liệu sẽ được phân ra thành các tập train test và lưu lại class của các lớp tương ứng (Có thể tham khảo cách dùng hàm ở dưới).

```python
X_train, y_train, classes_train = load_dataset('faces', train=True, as_grey=True)
X_test, y_test, classes_test = load_dataset('faces', train=False, as_grey=True)

assert classes_train == classes_test
classes = classes_train
```

Hiển thị một số ảnh trong tập dữ liệu để xem thử:

```
num_classes = len(classes)
samples_per_class = 10
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx])
        plt.axis('off')
        if i == 0:
            plt.title(y)
plt.show()
```

Cụ thể ta sẽ chọn ngẫu nhiên 10 ảnh cho mỗi lớp để hiển thị đảm bảo hàm load_dataset hoạt động.

Ta sẽ nhận được kết quả như hình dưới:


2) Trích đặc trưng:

Tiến hành trích đặc trưng bằng cách flatten thành một vector.

Mỗi hình ảnh sẽ được biểu diễn bằng một vector đặt trưng 64 x 64 = 4096 chiều.

Sử dụng np.reshape để flatten cho 2 tập X_train và X_test.

```
# Trích đặc trưng cho ảnh bằng cách trải dài ảnh (flatten) thành một vector.
# Như thế mỗi ảnh sẽ được biểu diễn bằng một vector đặc trưng 64 x 64 = 4096 chiều.
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

### c) K-Nearest Neighbor (k-NN):

Ta thử sử dụng phương pháp k-NN trên các đặc trưng thô của ảnh (giá trị mức xám của tất cả các điểm ảnh) để phân lớp các ảnh kiểm tra trong thư mục test.

Các bước thực hiện:

1) Tính khoảng cách L2 (khoảng cách Euclide) từ mỗi ảnh kiểm tra nằm trong tập X_test và mỗi ảnh huấn luyện nằm trong X_train bằng hàm compute_distances.

```
def compute_distances(X1, X2):
    """Compute the L2 distance between each point in X1 and each point in X2.
    It's possible to vectorize the computation entirely (i.e. not use any loop).

    Args:
        X1: numpy array of shape (M, D) normalized along axis=1
        X2: numpy array of shape (N, D) normalized along axis=1

    Returns:
        dists: numpy array of shape (M, N) containing the L2 distances.
    """
    M = X1.shape[0]
    N = X2.shape[0]
    assert X1.shape[1] == X2.shape[1]

    dists = np.zeros((M, N))

    X1_repeat = np.repeat(X1,N,0)
    # Lặp mỗi ảnh trong tập test N lần để bằng số lượng khoảng cách cần tính từ 1 ảnh test đến N ảnh train
    X2_repeat = np.tile(X2,(M,1))
    # Lặp tập train M lần cho để tính khoảng cách từ M ảnh đến tập train
    diff = np.subtract(X1_repeat,X2_repeat)
    del X1_repeat
    del X2_repeat
    diff = diff**2
    result = np.sum(diff,1)**(0.5) #Khoảng cách euclid
    del diff
    dists = np.reshape(result,(M,N))
    del result
    # del = xóa biến vì các biến X1_repeat, X2_repeat,... sử dụng rất nhiều bộ nhớ

    assert dists.shape == (M, N), "dists should have shape (M, N), got %s" % dists.shape

    return dists
```

2) Sử dụng hàm split_folds chia tập dữ liệu X_train thành 5 tập con (5 folds) để tìm k tốt nhất bằng phương pháp kiểm tra chéo (cross-validation).

- Tìm k bằng phương pháp kiểm tra chéo (Cross-Validation):

Ta không biết giá trị tốt nhất cho k là bao nhiêu. Vì thế, ta phải tìm k bằng phương pháp kiểm tra chéo.

Không được sử dụng tập dữ liệu test để chọn giá trị cho k, vì nếu làm thế ta đã ăn gian kết quả ( biết trước đáp án trước khi làm).

Chỉ được sử dụng tập huấn luyện để tìm giá trị k tốt nhất.

Phương pháp Cross-validation sẽ được thực hiện như sau: Ta sẽ chia tập huấn luyện thành một số tập con.

Với mỗi lần kiểm chéo, ta có:

- 80% dữ liệu để làm tập huấn luyện.
- 20% dữ liệu để kiểm tra chéo.

Ta sẽ tính độ chính xác nhận dạng cho mỗi tập dữ liệu và tính độ chính xác trung bình sau 5 lần kiểm tra chéo để chọn ra giá trị k cho độ chính xác trung bình cao nhất.

Mục tiêu của các hàm là trả về các tập huấn luyện (features và labels) cùng với các tập xác thực tương ứng. Trong mỗi fold, tập xác thực sẽ đại diện cho (1/num_folds) của dữ liệu, trong khi tập huấn luyện sẽ đại diện cho (num_folds-1)/num_folds. Nếu num_folds=5, điều này tương ứng với việc chia 80% / 20%.

```
def split_folds(X_train, y_train, num_folds):
    """Split up the training data into `num_folds` folds.
    Args:
        X_train: numpy array of shape (N, D) containing N examples with D features each
        y_train: numpy array of shape (N,) containing the label of each example
        num_folds: number of folds to split the data into

    Returns:
        X_trains: numpy array of shape (num_folds, train_size * (num_folds-1) / num_folds, D)
        y_trains: numpy array of shape (num_folds, train_size * (num_folds-1) / num_folds)
        X_vals: numpy array of shape (num_folds, train_size / num_folds, D)
        y_vals: numpy array of shape (num_folds, train_size / num_folds)

    """
    assert X_train.shape[0] == y_train.shape[0]

    validation_size = X_train.shape[0] // num_folds
    training_size = X_train.shape[0] - validation_size

    X_trains = np.zeros((num_folds, training_size, X_train.shape[1]))
    y_trains = np.zeros((num_folds, training_size), dtype=int)
    X_vals = np.zeros((num_folds, validation_size, X_train.shape[1]))
    y_vals = np.zeros((num_folds, validation_size), dtype=int)

    # YOUR CODE HERE
    # Hint: You can use the numpy array_split function.
    X = np.split(X_train,num_folds)
    Y = np.split(y_train,num_folds)
    for i in range(num_folds):
        X_vals[i] = X[i]
        y_vals[i] = Y[i]
        X_trains[i] = np.concatenate(np.delete(X,i,0),0)
        y_trains[i] = np.concatenate(np.delete(Y,i,0),0)
    # END YOUR CODE

    return X_trains, y_trains, X_vals, y_vals
```

3) Với tập con, và mỗi giá trị k, ta dự báo nhãn/lớp (classes/labels) của dữ liệu kiểm tra và tính độ chính xác nhận dạng (accuracy).

```
# Tạo 1 danh sách các giá trị cho k
k_choices = list(range(5, 101, 5))

# Dictionnary mapping k values to accuracies
# Với mỗi giá trị k, ta cần phải tính độ chính xác nhận dạng `num_folds` (5) lần.
# Ví dụ k_to_accuracies[1] sẽ có các giá trị [0.22, 0.23, 0.19, 0.25, 0.20] với 5 lần.
k_to_accuracies = {}

for k in k_choices:
    print("Running for k=%d" % k)
    accuracies = []
    for i in range(num_folds):
        # Dự báo/nhận dạng
        fold_dists = compute_distances(X_vals[i], X_trains[i])
        y_pred = predict_labels(fold_dists, y_trains[i], k)

        # Tính và in tỉ lệ nhận dạng đúng
        num_correct = np.sum(y_pred == y_vals[i])
        accuracy = float(num_correct) / len(y_vals[i])
        accuracies.append(accuracy)

    k_to_accuracies[k] = accuracies
    print(k_to_accuracies[k])
```

Để dự đoán nhãn của dữ liệu ta sẽ khởi tạo hàm predict_labels:

```
def predict_labels(dists, y_train, k=1):
    """Given a matrix of distances `dists` between test points and training points,
    predict a label for each test point based on the `k` nearest neighbors.

    Args:
        dists: A numpy array of shape (num_test, num_train) where dists[i, j] gives
               the distance betwen the ith test point and the jth training point.

    Returns:
        y_pred: A numpy array of shape (num_test,) containing predicted labels for the
                test data, where y[i] is the predicted label for the test point X[i].
    """
    num_test, num_train = dists.shape
    y_pred = np.zeros(num_test, dtype=int)

    for i in range(num_test):
        closest_y = []
        d = np.argsort(dists[i])[0:k]
        # d =  lấy ra k vị trí có distance nhỏ nhất của hình i
        closest_y = y_train[d]
        # lấy nhãn của k vị trí đó cho vào closest y
        label = np.zeros((1,16)) # Tập dl có 16 nhãn
        for j in closest_y:
            label[0,j] += 1 # Đếm nhãn trong tập closest
        # Lấy vị trí có số lần xuất hiện nhiều nhất
        y_pred[i] = np.argmax(label)

    return y_pred
```

4) Sử dụng giá trị k tốt nhất tìm được bằng phương pháp cross-validation, ta phân lớp lại tập kiểm tra X_test để tính lại độ chính xác nhận dạng.

```
best_k = None
# Chọn k có trung bình accuracy cao nhất
best_k = (np.argsort(accuracies_mean)[-1]+1)*5

y_test_pred = predict_labels(dists, y_train, k=best_k)

num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('For k = %d, got %d / %d correct => accuracy: %f' % (best_k, num_correct, num_test, accuracy))
```

## III. Bài tập:

Object detection với **YOLOv9**:

(Phần thực hành này khuyến khích sử dụng google colab hoặc máy tính cá nhân có GPU)

### 1. Tổng quan về YOLO và các thuật toán phát hiện đối tượng:

YOLO (You Only Look Once) là một thuật toán nổi bật trong lĩnh vực phát hiện đối tượng (object detection), được biết đến với hiệu quả và tốc độ xử lý cao.

YOLO là một thuật toán phát hiện đối tượng thời gian thực, được Joseph Redmon giới thiệu vào năm 2016. Thay vì phân chia quá trình phát hiện đối tượng thành nhiều bước như các phương pháp truyền thống (ví dụ: R-CNN), YOLO xử lý toàn bộ hình ảnh trong một bước duy nhất và dự đoán các bounding box và class label đồng thời.

**Cách thức hoạt động:**

- YOLO chia hình ảnh đầu vào thành lưới (grid) và mỗi ô trong lưới sẽ chịu trách nhiệm phát hiện các đối tượng có tâm nằm trong ô đó
- Dự đoán bounding box: Mỗi ô sẽ dự đoán B bounding box, bao gồm tọa độ (x, y, chiều rộng, chiều cao) và độ tin cậy (confidence score).
- Dự đoán nhãn: Đối với mỗi ô, YOLO dự đoán khả năng đối tượng thuộc vào một trong số các class.

Bạn có thể xem thêm thông tin về YOLOv9 tại đây:

https://arxiv.org/pdf/2402.13616

### 1. Chuẩn bị dữ liệu:

Thu thập dữ liệu: Dữ liệu train có thể thu thập trên internet hoặc từ bất cứ nguồn nào có sẵn.

2. **Xử lý dữ liệu:**

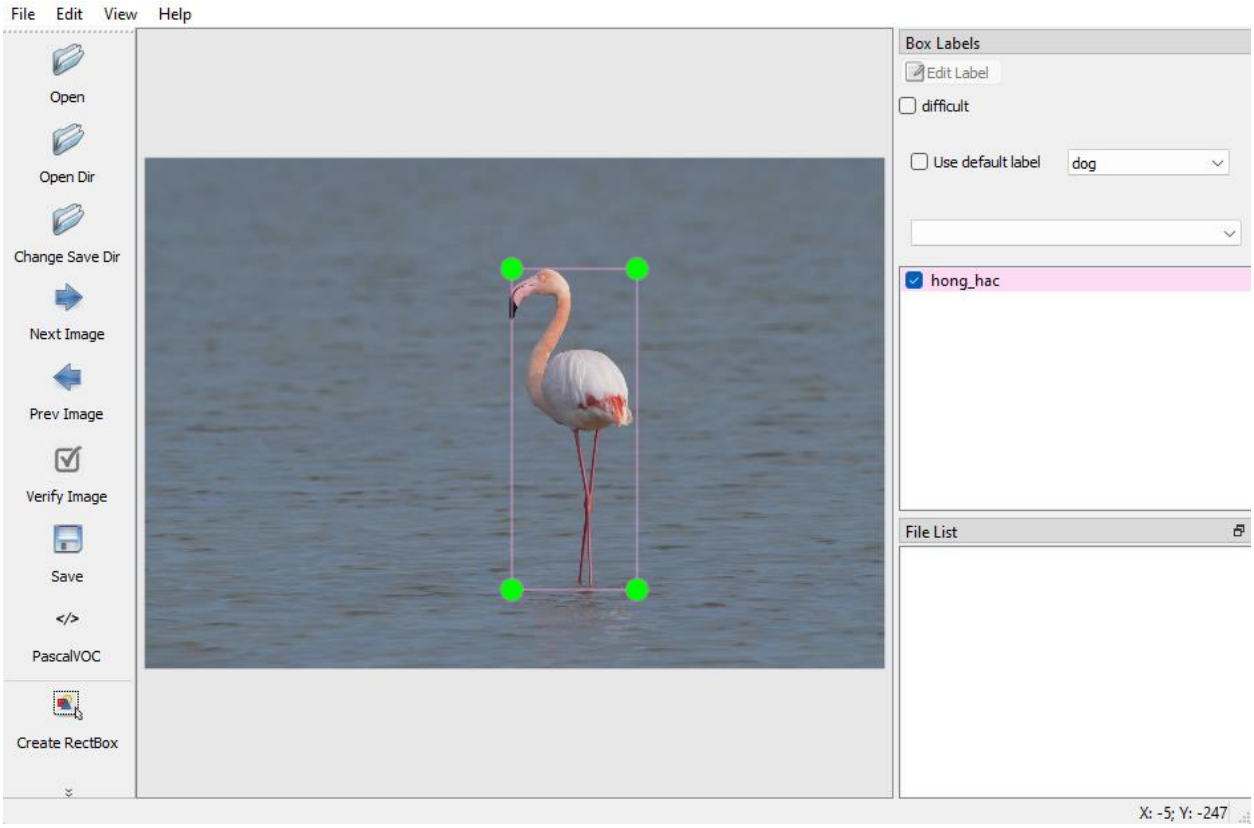
- Lọc dữ liệu gây nhiễu: Loại bỏ các hình ảnh gây nhiễu loạn thông tin (ảnh gồm nhiều nhãn khác nhau).
- Thay đổi tên ảnh: Đổi tên ảnh theo một format thống nhất để có thể dễ xử lí sau này.
- Chuẩn hóa kích thước ảnh: Chuẩn hóa kích thước của tất cả các ảnh theo đúng kích thước mong muốn.

3. **Gán nhãn dữ liệu:**

Trước khi huấn luyện mô hình, dữ liệu hình ảnh cần được gán nhãn chính xác. Ta có thể sử dụng các công cụ như LabelImg (Xem hướng dẫn cài đặt tại đây) hoặc bất cứ công cụ nào khác để tạo file nhãn dưới dạng txt cho các hình ảnh của mình.

Ta sẽ chia thành 2 thư mục là thư mục images chứa các ảnh và một thư mục labels chứa nhãn của ảnh là các file txt.

Mở ứng dụng labelImg, một giao diện sẽ hiển thị lên như sau:




Trong đó:

- Chọn Open Dir và trỏ vào thư mục images đã tạo ở trên để load các ảnh.
- Chọn Change Save Dir và trỏ vào thư mục labels ở trên, để lưu file gán nhãn.

Duyệt qua từng file ảnh, chọn Create Rectbox để vẽ hình chữ nhật quanh vật thể và gắn nhãn cho nó. Nhớ nhấn Ctrl +S để lưu lại thao tác gán nhãn với từng file ảnh.

Lưu ý: Việc gán nhãn cần chính xác để mô hình học được đúng các đối tượng trong ảnh.

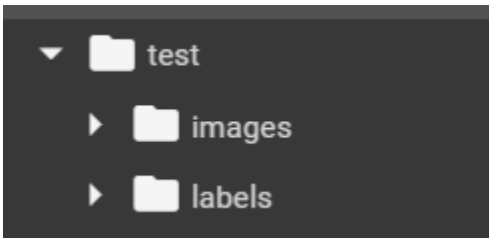
Ta đưa 2 thư mục này lên google drive.

4. **Phân chia dữ liệu:**

Phân chia dữ liệu hình ảnh thành tập huấn luyện và tập kiểm tra theo tỉ lệ 70% train, 20% test và 10% validation. Đảm bảo rằng nhãn đi kèm với từng ảnh được sao chép chính xác.

Tạo thư mục split_data để lưu các thư mục trên chứa ảnh và nhãn đã được phân chia.

Bạn có thể xem cấu trúc thư mục sau phân chia thông qua ví dụ sau:




5. **Tạo file data.yaml:**

Tạo file data.yaml để lưu các thông tin bên dưới:

```
train: ..split-data/train/images
val: ..split-data/valid/images
test: ..split-data/test/images

nc: ..

names: [..]
```

**Trong đó:**

- Train, val, test: đường dẫn đến thư mục chứa dữ liệu tương ứng
- nc: Số lượng class của tập dữ liệu
- names: tên của các class trong tập dữ liệu (ví dụ: names: ['Chair', 'Sofa', 'Table'])

6. **Tải và cài đặt mô hình YOLO:**

Tiến hành cài đặt thông qua các bước sau:

- Tải mã nguồn YOLOv9 ở link: https://github.com/WongKinYiu/yolov9
- Cài đặt các thư viện cần thiết thông qua file requirements.txt đi kèm với mã nguồn.

**7. Huấn luyện mô hình:**

Chúng ta tiến hành huấn luyện mô hình YOLO bằng cách gọi file train_dual.py và truyền vào các tham số phù hợp.

Đây là file mà mã nguồn đã cung cấp sẵn cho chúng ta sử dụng nên việc chúng ta cần làm là khám phá các tham số đầu vào của hàm này để tinh chỉnh cho phù hợp với tập dữ liệu của chúng ta. Các tham số từ hàm này có thể kể lần lượt như sau:

- hyp: Đây là đường dẫn tới file `.yaml` chứa các siêu tham số (hyperparameters) hoặc trực tiếp là một từ điển (dictionary) chứa các siêu tham số. Các siêu tham số bao gồm những giá trị như học suất (`learning rate`), trọng số của bộ tối ưu (`weight decay`), v.v. mà người dùng có thể cấu hình.
- save_dir: thư mục lưu kết quả huấn luyện.
- epochs: số lượng epoch cho quá trình huấn luyện.
- batch_size: kích thước của batch dữ liệu trong quá trình huấn luyện.
- weights: tệp chứa trọng số ban đầu của mô hình, có thể là mô hình được huấn luyện trước.
- single_cls: nếu đặt là `True`, nó sẽ coi tất cả các lớp là một lớp duy nhất.
- evolve: nếu là `True`, quá trình huấn luyện sẽ tìm kiếm các siêu tham số tối ưu qua quá trình tiến hóa.
- data: đường dẫn tới tệp `.yaml` chứa thông tin về tập dữ liệu.
- cfg: cấu hình kiến trúc của mô hình.
- resume: nếu đặt là `True`, nó sẽ tiếp tục huấn luyện từ điểm dừng trước đó.
- noval: nếu đặt là `True`, nó sẽ không thực hiện quá trình đánh giá (validation).

- nosave: nếu đặt là `True`, nó sẽ không lưu các trọng số mô hình.
- workers: số luồng xử lý để nạp dữ liệu.
- freeze: danh sách các lớp của mô hình cần đóng băng (không huấn luyện).
- device: Thiết bị được dùng để huấn luyện mô hình (ví dụ như `cpu` hoặc `cuda`).

```
#Train
# train yolov9 models
!python ../yolov9/train_dual.py \
--workers 8 --device 0 --batch 4 --data ../data.yaml \
--img 640 --cfg ../yolov9/models/detect/yolov9-e-custome.yaml \
--weights ../yolov9/yolov9-e.pt --name yolov9-c \
--hyp ../yolov9/data/hyps/hyp.scratch-high.yaml \
--min-items 0 --epochs 50 --close-mosaic 15
```

Có thể tham khảo các tham số như trên.

Trong quá trình train sẽ hiển thị bảng thông tin về bước train và các thông số như sau:

```
Epoch   GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/49     8.48G     0.7472      1.156      1.474          6        640: 100% 114/114 [01:20<00:00,  1.42it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 21/21 [00:09<00:00,  2.31it/s]
             all        161        161      0.847      0.804      0.938      0.714

Epoch   GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/49     8.48G     0.8734      1.478      1.588          8        640: 100% 114/114 [01:18<00:00,  1.45it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 21/21 [00:09<00:00,  2.21it/s]
             all        161        161      0.886      0.869      0.947      0.746

Epoch   GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/49     8.48G     0.9355      1.707      1.598          0        640: 100% 114/114 [01:17<00:00,  1.47it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 21/21 [00:09<00:00,  2.30it/s]
             all        161        161      0.912      0.541      0.843      0.612

Epoch   GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 4/49     8.48G     0.9928      1.643      1.645          3        640: 100% 114/114 [01:17<00:00,  1.47it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 21/21 [00:09<00:00,  2.28it/s]
             all        161        161      0.715      0.828      0.802      0.626

Epoch   GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 5/49     8.48G     0.9609      1.286      1.586          3        640: 100% 114/114 [01:17<00:00,  1.48it/s]
           Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 21/21 [00:08<00:00,  2.42it/s]
             all        161        161      0.795      0.585      0.817       0.58
```

Tìm hiểu về các thông số và cho biết train đến khi nào? Xem gì để biết model đã tốt? Nếu chưa tốt thì cần điều chỉnh những tham số nào?

### 8. Đánh giá mô hình:

Tiến hành đánh giá mô hình sau khi huấn luyện.

Tiến hành kiểm thử với 100 ảnh bất kỳ kết quả thu được có thể tương tự như ảnh dưới:


Lập bản excel đánh giá các chỉ số như accuracy, precision, recall và F1 của mô hình vừa huấn luyện được.