

# BÁO CÁO THỰC HÀNH

CÔNG NGHỆ INTERNET OF THINGS HIỆN ĐẠI

# Lab 5

## Xây dựng ứng dụng AI cơ bản trong mô hình IoT

GVHD: Phan Trung Phát

Lớp: NT532.021.MMCL.2

Họ và tên	MSSV
Tổng Võ Anh Thuận	21522652
Trịnh Vinh Đại	21521915
Lê Huỳnh Quang Vũ	21522797

### ĐÁNH GIÁ KHÁC (\*):

Nội dung	Kết quả
Tổng thời gian thực hiện bài thực hành trung bình (1)	11 ngày
Link Video thực hiện (2) (nếu có)	<a href="#">Video n Code</a>
Ý kiến (3) (nếu có) + Khó khăn + Đề xuất ...	
Điểm tự đánh giá (4)	10/10

(\*): phần (1) và (4) bắt buộc thực hiện.

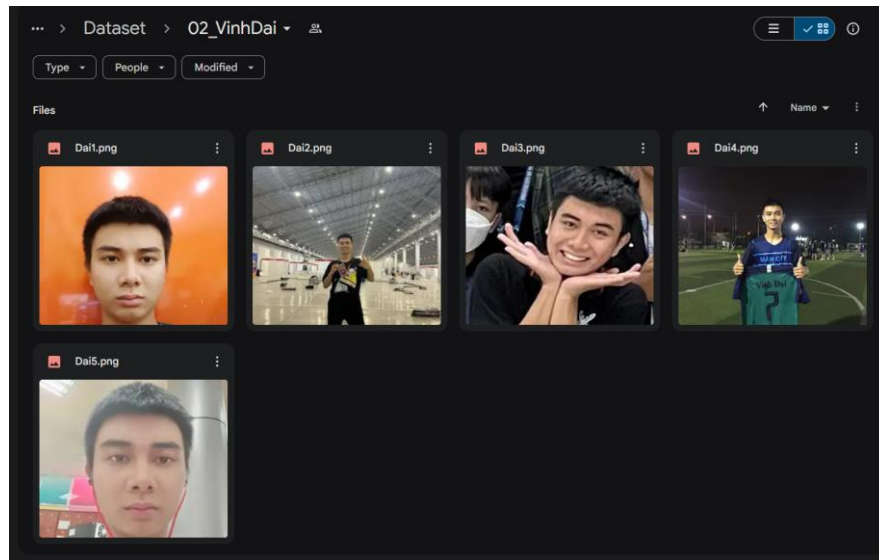
Phần bên dưới là báo cáo chi tiết  
của nhóm/cá nhân thực hiện.



**LƯU HÀNH NỘI BỘ**

**Câu hỏi 1. Thực hiện phần C, đảm bảo độ chính xác của các gương mặt đối với các thành viên trong nhóm trên 60%. Hãy nêu cách để bạn xử lý được vấn đề này. Từ đó, tìm hiểu những cách nào để tăng cường dữ liệu,... để tăng độ chính xác đối với mô hình?**

First of all, we create our own dataset consisting of images of our faces. Each folder will be named after our group members: AnhThuan, QuangVu, and VinhDai.



Now, since each folder only has about 5 images, the dataset is too small for model training. Thereby, after running the provided face alignment processing Python script, we use image augmentation techniques to increase the size of our dataset. Below is a snippet for our augmentation code (in augment.py):

```
data_dir = "Dataset/processed"
output_dir = "Dataset/augmented"

datagen = ImageDataGenerator(
    rotation_range=20, # Rotate images by 20 degrees randomly
    width_shift_range=0.2, # Shift images horizontally by 20%
    height_shift_range=0.2, # Shift images vertically by 20%
    shear_range=0.2, # Shear images by 20%
    zoom_range=0.2, # Zoom images by 20%
    horizontal_flip=True # Flip images horizontally randomly
)

# Loop through each subfolder class
for folder in listdir(data_dir):
    class_dir = f"{data_dir}/{folder}"

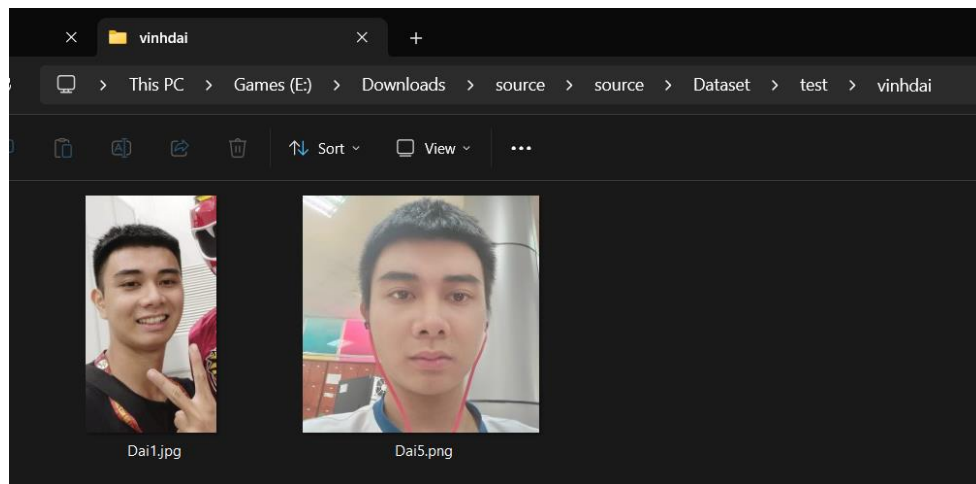
    # Check if it's a directory and not a file
    if os.path.isdir(class_dir):
        # Define output directory for this class
        output_class_dir = f"{output_dir}/{folder}"
        os.makedirs(output_class_dir, exist_ok=True) # Create directory if it doesn't exist
```

For each image in a folder, we performed rotation, width shift, height shift, and so on. By doing this, with each image, we have 20 new images as data. The resulting images look like below:



As a result, our initial dataset of only 15 images now has 300 images to be used as training data.

Moving on, we create a testing set in order to evaluate our model after the training phase. This testing set will not include any image from the training set:



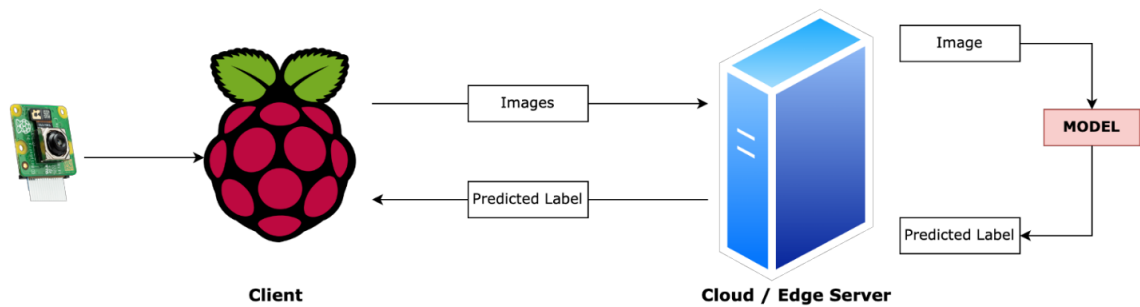
With that, we begin to train our SVC model with the provided classifier.py file: "python src/classifier.py TRAIN Dataset/augmented Models/20180402-114759.pb Models/facemodel.pkl --batch\_size 1000"

After training is done, we run the model on the testing set and the result is as follows (ignore the VanHau and VanToan part, we're too lazy to remove them):

```
use tr.write.write.  
Calculating features for images  
2024-05-22 21:35:14.805091: I tensorflow/compiler/mlir/mlir_graph_optimiza  
Testing classifier  
Loaded classifier model from file "Models/facemodel.pkl"  
0 anhtuan: 0.991  
1 anhtuan: 0.952  
2 quangvu: 0.978  
3 vanhau: 0.993  
4 vantoan: 0.995  
5 vinh dai: 0.973  
6 vinh dai: 0.966  
Accuracy: 1.000  
  
(venv) E:\Downloads\source\source>
```



**Câu hỏi 2. Tích hợp mô hình đã được huấn luyện ở câu 1 vào mô hình IoT cơ bản được thể hiện tại Hình 1. Mô hình gồm có 1 Raspberry Pi và 1 PC / laptop cá nhân. Raspberry Pi đóng vai trò như Client, được sử dụng để thu thập hình ảnh từ camera về người dùng cần nhận diện. PC / laptop cá nhân đóng vai trò như một Edge / Cloud Server dùng để xử lý hình ảnh thu được từ Raspberry Pi và nhận diện hình ảnh đó. Từ đó, trả kết quả nhận diện về cho Client và hiển thị kết quả nhận diện lên màn hình. Quá trình gửi nhận dữ liệu giữa Raspberry Pi và Edge / Cloud Server sử dụng Internet (Có thể sử dụng một trong các giao thức như: HTTP, MQTT, RPC ...).**



*Hình 1. Tổng quan mô hình Lab 5*

## 1. Setup:

Components in use:

- 1 IMX 219-160 CSI camera
- 1 Jetson Nano B1
- 1 monitor
- 1 laptop

Wire connection:

- CSI camera directly connected to Jetson Nano.
- Jetson Nano serial connected to monitor.





## 2. Code explanation:

```
# Create a directory to save the images
url = 'http://172.31.11.66:8000/recog'
output_dir = "captured_frames"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
# Initialize the camera
cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
```

Define edge server address (URL) to send image for processing, output directory (output\_dir) to store input image and cv2 video capture to capture images.

```
frame_count = 0
start_time = time.time()
duration = 5 # capture duration in seconds

while True: # Capture 120 frames
    ret_val, frame = cap.read()
    if not ret_val:
        break

    # Save the frame as an image file
    frame_filename = os.path.join(output_dir, f"frame_{frame_count:04d}.jpg")
    cv2.imwrite(frame_filename, frame)
    frame_count += 1

    # Display the frame (optional)
```

```
cv2.imshow('CSI Camera', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Check if the duration has passed
if time.time() - start_time > duration:
    break
```

In loop, use cv2 to capture images in 5 seconds (duration) and save images as jpg file inside output directory.

```
frame_count -= 1
filename = f"frame_{frame_count:04d}.jpg"
img_path = f'{output_dir}/{filename}'
encoded_image_data = encode_image_to_base64(frame_filename)
data = {
    'image': encoded_image_data,
    'w': 160, # width
    'h': 160 # height
}

# Sending the POST request
sent_time = time.time()
response = requests.post(url, data=data)
received_time = time.time()
```

After finishing capturing, encode the last image to base 64, format to 160x160 resolution and save it as data to send to server. Get response from server and calculate processing time.

```
# Printing the response
print("=====")
print(f"Response from server:{response.text}")
print(f"Execution time:{received_time - sent_time}")
print("=====")
```

Print result to console.

**Câu hỏi 3. Viết thêm hàm để đo tổng thời gian nhận diện 1 hình ảnh. Thời gian được tính từ lúc Raspberry Pi gửi hình ảnh cho Edge / Cloud Server cho đến khi nhận được kết quả trả về.**



```
# Sending the POST request
sent_time = time.time()
response = requests.post(url, data=data)
received_time = time.time()

# Printing the response
print("=====")
print(f"Response from server:{response.text}")
print(f"Execution time:{received_time - sent_time}")
print("=====")
```

In this code, we save the time of sending and receiving requests, send a POST request to the URL with the data, and finally calculate and print out the total execution time.

**Câu hỏi 4. Tìm hiểu các file trong mã nguồn được cung cấp. Trình bày các quy trình, bước xử lý, mô tả về những tìm hiểu của mình về mã nguồn trên.**

**align\_dataset\_mtcnn.py code explanation:**

Because this script is rather long, we will only explain the most crucial part that is the face detection and alignment code.

Within the main function, a TensorFlow graph is created, and the MTCNN model is initialized. This model consists of three neural networks: P-Net (Proposal Network), R-Net (Refinement Network), and O-Net (Output Network).

```
with tf.Graph().as_default():
    sess = tf.compat.v1.Session()
    with sess.as_default():
        pnet, rnet, onet = align.detect_face.create_mtcnn(sess, None)
```

- TensorFlow Graph: A new computational graph is created to run the MTCNN model.
- Session: A TensorFlow session is started, which will be used to execute operations defined in the graph.
- MTCNN Initialization: The create\_mtcnn function initializes the three networks (P-Net, R-Net, O-Net) that make up the MTCNN model.

MTCNN (Multi-task Cascaded Convolutional Networks) is a widely used deep learning model for face detection and alignment. It is designed to detect faces in images and provide facial landmarks with high accuracy and efficiency. The model consists of three stages, each using a different neural network to progressively refine the face detection results:

- P-Net (Proposal Network): This network scans the image in a sliding window fashion to propose candidate facial regions. It generates bounding boxes and corresponding scores for potential faces at various scales.



- R-Net (Refinement Network): The regions proposed by the P-Net are refined by the R-Net. This network further filters out false positives and refines the bounding box coordinates.
- O-Net (Output Network): The final stage, the O-Net, performs additional refinement on the remaining candidate regions. It also outputs five facial landmarks (such as the positions of the eyes, nose, and mouth) for each detected face.

Parameters for the face detection process are set like below:

```
minsize = 20 # minimum size of face
threshold = [ 0.6, 0.7, 0.7 ] # three steps's threshold
factor = 0.709 # scale factor
```

- minsize: The minimum size of faces to detect.
- threshold: Threshold values for the three stages of the MTCNN model.
- factor: The scaling factor used to create the image pyramid, which helps in detecting faces at different scales.

The script then iterates over the dataset, and for each image, it attempts to detect faces:

```
try:
    import imageio
    img = imageio.imread(image_path)
except (IOError, ValueError, IndexError) as e:
    errorMessage = '{}: {}'.format(image_path, e)
    print(errorMessage)
else:
    if img.ndim<2:
        print('Unable to align "%s"' % image_path)
        text_file.write('%s\n' % (output_filename))
        continue
    if img.ndim == 2:
        img = facenet.to_rgb(img)
    img = img[:, :, 0:3]
```

- Image Reading: The script reads the image from the file. If there is an error, it logs the error and continues to the next image.
- Image Conversion: If the image is grayscale, it is converted to RGB.

The MTCNN model is used to detect faces in the image:

```
bounding_boxes, _ = align.detect_face.detect_face(img, minsize, pnet, rnet, onet, threshold, factor)
nrof_faces = bounding_boxes.shape[0]
```

- detect\_face: This function uses the MTCNN model to detect faces in the image.
- bounding\_boxes: The detected bounding boxes for faces are returned. Each bounding box is represented by four coordinates: [x1, y1, x2, y2].
- nrof\_faces: The number of detected faces.

The script then processes the detected faces:





```
if nrof_faces>0:
    det = bounding_boxes[:,0:4]
    det_arr = []
    img_size = np.asarray(img.shape)[0:2]
    if nrof_faces>1:
        if args.detect_multiple_faces:
            for i in range(nrof_faces):
                det_arr.append(np.squeeze(det[i]))
        else:
            bounding_box_size = (det[:,2]-det[:,0])*(det[:,3]-det[:,1])
            img_center = img_size / 2
            offsets = np.vstack([ (det[:,0]+det[:,2])/2-img_center[1], (det[:,1]+det[:,3])/2-img_center[0] ])
            offset_dist_squared = np.sum(np.power(offsets,2.0),0)
            index = np.argmax(bounding_box_size-offset_dist_squared*2.0) # some extra weight on the centering
            det_arr.append(det[index,:])
    else:
        det_arr.append(np.squeeze(det))
```

- **Multiple Faces:** If multiple faces are detected and detect\_multiple\_faces is true, all faces are processed. Otherwise, the face closest to the center is chosen.
- **Bounding Box Selection:** For a single face, the bounding box is directly added to det\_arr. For multiple faces, the bounding box with the highest score (based on size and distance from the center) is selected if detect\_multiple\_faces is false.

The detected faces are cropped and saved as aligned thumbnails:

```
for i, det in enumerate(det_arr):
    det = np.squeeze(det)
    bb = np.zeros(4, dtype=np.int32)
    bb[0] = np.maximum(det[0]-args.margin/2, 0)
    bb[1] = np.maximum(det[1]-args.margin/2, 0)
    bb[2] = np.minimum(det[2]+args.margin/2, img_size[1])
    bb[3] = np.minimum(det[3]+args.margin/2, img_size[0])
    cropped = img[bb[1]:bb[3],bb[0]:bb[2],:]
    from PIL import Image
    cropped = Image.fromarray(cropped)
    scaled = cropped.resize((args.image_size, args.image_size), Image.BILINEAR)
    nrof_successfully_aligned += 1
    filename_base, file_extension = os.path.splitext(output_filename)
    if args.detect_multiple_faces:
        output_filename_n = "{}_{}".format(filename_base, i, file_extension)
    else:
        output_filename_n = "{}".format(filename_base, file_extension)
    imageio.imwrite(output_filename_n, scaled)
    text_file.write('%s %d %d %d %d\n' % (output_filename_n, bb[0], bb[1], bb[2], bb[3]))
else:
    print('Unable to align "%s"' % image_path)
    text_file.write('%s\n' % (output_filename))
```

- **Bounding Box Adjustment:** The bounding box coordinates are adjusted with a specified margin from command arguments.
- **Cropping and Resizing:** The face region is cropped and resized to the specified dimensions.
- **Saving:** The aligned face thumbnail is saved to the output directory, and the bounding box coordinates are recorded.

### face\_rec\_falsk.py code explanation:

Flask is a lightweight web framework written in Python. It's popular for its simplicity and flexibility, making it a good choice for building small to medium-sized web applications. In this file, we use Flask as a web server which will run the machine learning model upon receiving the request and return the model's output to the client.

```

with tf.Graph().as_default():

    # Cài đặt GPU nếu có
    gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.6)
    sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(gpu_options=gpu_options, log_device_placement=False))

    with sess.as_default():
        # Load the model
        print('Loading feature extraction model')
        facenet.load_model(FACENET_MODEL_PATH)

        # Get input and output tensors
        images_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("input:0")
        embeddings = tf.compat.v1.get_default_graph().get_tensor_by_name("embeddings:0")
        phase_train_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("phase_train:0")
        embedding_size = embeddings.get_shape()[1]
        # pnet, rnet, onet = align.detect_face.create_mtcnn(sess, "src/align")
        images_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("input:0")
        embeddings = tf.compat.v1.get_default_graph().get_tensor_by_name("embeddings:0")
        phase_train_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("phase_train:0")
        embedding_size = embeddings.get_shape()[1]

        pnet, rnet, onet = align.detect_face.create_mtcnn(sess, "src/align")

```

Above is the code for FaceNet model initialization which sets up a TensorFlow graph and session, configures the program to use a GPU with 60% memory fraction (GPUs are generally faster for processing machine learning tasks compared to CPUs). The code uses `facenet.load_model` to load a pre-trained facial recognition model called FaceNet.

```

@app.route('/recog', methods=['POST'])
@cross_origin()
def upload_img_file():
    if request.method == 'POST':
        # base 64
        name="Unknown"
        f = request.form.get('image')
        print(type(f))
        w = int(request.form.get('w'))
        h = int(request.form.get('h'))

        decoded_string = base64.b64decode(f)
        frame = np.fromstring(decoded_string, dtype=np.uint8)
        # frame = frame.reshape(w,h,3)
        frame = cv2.imdecode(frame, cv2.IMREAD_ANYCOLOR) # cv2.IMREAD_COLOR in OpenCV 3.1

        bounding_boxes, _ = align.detect_face.detect_face(frame, MIN_SIZE, pnet, rnet, onet, THRESHOLD, FACTOR)

```

The `@app.route('/recog', methods=['POST'])` decorator defines a route for the web application. This means that when a POST request is sent to the URL `/recog`, the function below will handle the request. The function then decodes the base64-encoded image data using `base64.b64decode`. Then, calls a function named `detect_face` which is responsible for detecting faces in the provided image, the result is assigned to the `bounding_boxes` variable.

```

if faces_found > 0:
    det = bounding_boxes[:, 0:4]
    bb = np.zeros((faces_found, 4), dtype=np.int32)
    for i in range(faces_found):
        bb[i][0] = np.maximum(det[i][0]-32/2, 0)
        bb[i][1] = np.maximum(det[i][1]-32/2, 0)
        bb[i][2] = np.maximum(det[i][2]+32/2, 160)
        bb[i][3] = np.maximum(det[i][3]+32/2, 160)
        #cropped = frame
        cropped = frame[bb[i][1]:bb[i][3], bb[i][0]:bb[i][2], :]
        scaled = cv2.resize(cropped, (160, 160),
                           interpolation=Image.BILINEAR)
        scaled = facenet.prewhiten(scaled)
        scaled_reshape = scaled.reshape(-1, 160, 160, 3)
        feed_dict = {images_placeholder: scaled_reshape, phase_train_placeholder: False}
        emb_array = sess.run(embeddings, feed_dict=feed_dict)
        predictions = model.predict_proba(emb_array)
        best_class_indices = np.argmax(predictions, axis=1)
        best_class_probabilities = predictions[
            np.arange(len(best_class_indices), best_class_indices)]
        best_name = class_names[best_class_indices[0]]
        print("Name: {}, Probability: {}".format(best_name, best_class_probabilities))

    if best_class_probabilities > 0.6:
        name = best_name
        break
    else:
        name = "Unknown"

```

The logic of the above code snippet is almost identical to the face alignment script (basically, if a face is detected, it crops the image to that face with a size of 160x160). The reason of this is because the input image must undergo the same pre-processing stage as when training the model, this is to ensure that the model will be able to predict with high accuracy. As required by the exercise, the server will only return the name of the person in the image if the model's confidence is above 0.6 (60%), else it will return "Unknown".

### facenet.py code explanation:

FaceNet is a facial recognition system developed by Google researchers in 2015. It can be used for tasks like facial verification (confirming someone's identity) and facial identification (recognizing someone from a database of faces). In the scope of this lab, FaceNet is used as a Python module for face recognition.

Overall, there are about 30 different performed functions but this project only uses plenty of these functions. We will go through the explanation of functions that are currently in use.

```

def get_image_paths_and_labels(dataset):
    image_paths_flat = []
    labels_flat = []
    for i in range(len(dataset)):
        image_paths_flat += dataset[i].image_paths
        labels_flat += [i] * len(dataset[i].image_paths)
    return image_paths_flat, labels_flat

```

To be able to train a model, we need to label each image of the dataset. This function helps us do that task. At first, it will scan the whole dataset. Then, it will assign all image paths to *image\_paths\_flat* and all labels with the format name of each label as *i\*len\_of\_image\_path* to *labels\_flat*. Finally, return *image\_paths\_flat* and *labels\_flat*.

```
def get_dataset(path, has_class_directories=True):
    dataset = []
    path_exp = os.path.expanduser(path)
    classes = [path for path in os.listdir(path_exp) \
                if os.path.isdir(os.path.join(path_exp, path))]
    classes.sort()
    nrof_classes = len(classes)
    for i in range(nrof_classes):
        class_name = classes[i]
        facedir = os.path.join(path_exp, class_name)
        image_paths = get_image_paths(facedir)
        dataset.append(ImageClass(class_name, image_paths))

    return dataset
```

This function will get the dataset by parsing the path. First, it gets all classes in the dataset and sorts them by Alphabet order. For each class, the function will create an ImageClass object and append it to the dataset. Finally, return the dataset.

```
def load_model(model, input_map=None):
    # Check if the model is a model directory (containing a metagraph and a checkpoint file)
    # or if it is a protobuf file with a frozen graph
    model_exp = os.path.expanduser(model)
    if (os.path.isfile(model_exp)):
        print('Model filename: %s' % model_exp)
        with gfile.FastGFile(model_exp, 'rb') as f:
            graph_def = tf.compat.v1.GraphDef()
            graph_def.ParseFromString(f.read())
            tf.import_graph_def(graph_def, input_map=input_map, name='')
    else:
        print('Model directory: %s' % model_exp)
        meta_file, ckpt_file = get_model_filenames(model_exp)

        print('Metagraph file: %s' % meta_file)
        print('Checkpoint file: %s' % ckpt_file)

        saver = tf.train.import_meta_graph(os.path.join(model_exp, meta_file), input_map=input_map)
        saver.restore(tf.get_default_session(), os.path.join(model_exp, ckpt_file))
```

As its name suggests, this function will check whether the given model exists. Then import that model for use.

```
def load_data(image_paths, do_random_crop, do_random_flip, image_size, do_prewhiten=True):
    nrof_samples = len(image_paths)
    images = np.zeros((nrof_samples, image_size, image_size, 3))
```

```

for i in range(nrof_samples):
    import imageio
    img = imageio.imread(image_paths[i])
    if img.ndim == 2:
        img = to_rgb(img)
    if do_prewhiten:
        img = prewhiten(img)
    img = crop(img, do_random_crop, image_size)
    img = flip(img, do_random_flip)
    images[i,:,:,:] = img
return images

```

This function gets 4 arguments: string path to images, boolean if the image should be cropped, boolean if the image should be flipped, and each image's size value. For each image, choose if it should be used in RGB or black-and-white color mode, crop or flip the image is optional and save that image to images list. Finally, return the images list.

We found a pipeline that use facenet module in the code, which is *classifier.py* file.

```

if args.use_split_dataset:
    dataset_tmp = facenet.get_dataset(args.data_dir)
    train_set, test_set = split_dataset(dataset_tmp, args.min_nrof_images_per_class,
args.nrof_train_images_per_class)
    if (args.mode == 'TRAIN'):
        dataset = train_set
    elif (args.mode == 'CLASSIFY'):
        dataset = test_set
    else:
        dataset = facenet.get_dataset(args.data_dir)

```

First, it uses the `get_dataset` function to get the dataset.

```

# Check that there are at least one training image per class
for cls in dataset:
    assert(len(cls.image_paths)>0, 'There must be at least one image for each class in
the dataset')

paths, labels = facenet.get_image_paths_and_labels(dataset)

print('Number of classes: %d' % len(dataset))
print('Number of images: %d' % len(paths))

```

Then, it uses the `get_image_paths_and_labels` function.

```

# Load the model

```



```
print('Loading feature extraction model')
facenet.load_model(args.model)
```

After that, use load\_model function to load the model.

```
for i in range(nrof_batches_per_epoch):
    ...
    images = facenet.load_data(paths_batch, False, False, args.image_size)
    ...
```

In this loop, get images by using load\_data function.

```
def split_dataset(dataset, min_nrof_images_per_class, nrof_train_images_per_class):
    train_set = []
    test_set = []
    for cls in dataset:
        paths = cls.image_paths
        # Remove classes with less than min_nrof_images_per_class
        if len(paths) >= min_nrof_images_per_class:
            np.random.shuffle(paths)
            train_set.append(facenet.ImageClass(cls.name, paths[:nrof_train_images_per_class]))
            test_set.append(facenet.ImageClass(cls.name, paths[nrof_train_images_per_class:]))
    return train_set, test_set
```

In this function, create ImageClass objects to store images in train\_set and test\_set.


### SVC (Support Vector Classifier) Model:

For the face classification part (in classifier.py above), we use an SVC model with linear kernel for both training and inference.

SVC is a specific implementation of the Support Vector Machine algorithm that is designed specifically for classification tasks. In other words, SVC is an SVM used for classification. It seeks to find the hyperplane that best separates the data points into different classes.

The mathematical concepts involved in SVC:

- **Hyperplane Equation:** In a binary classification problem, the goal is to find a hyperplane that separates the data points of two classes. Mathematically, a hyperplane is defined by the equation:  $w \cdot x + b = 0$ . Here,  $w$  is the weight vector perpendicular to the hyperplane,  $x$  represents a data point, and  $b$  is the bias term.
- **Margins:** The margin is the distance between the hyperplane and the nearest data points from each class. The larger the margin, the more confident we are in the classification. The margin can be calculated as the distance between two parallel hyperplanes, one for each class. Mathematically, the margin is inversely proportional to the norm of the weight vector  $w$ .
- **Support Vectors:** Support vectors are the data points that are closest to the hyperplane. These are the points that play a crucial role in determining the



position of the hyperplane. The support vectors are the ones that contribute to the margin calculation.

- **Soft Margin:** In real-world datasets, it's often not possible to have a perfect separation between classes. The concept of a soft margin SVM allows for some misclassification by allowing data points to be on the wrong side of the margin or even the wrong side of the hyperplane. This is controlled by introducing slack variables.
- **Objective Function:** The focal function of the SVC envelope consists of two main parts: minimizing the model complexity (the magnitude of the number vector) and minimizing the minimum type of error during training training. There are two main meanings: Maximize margin distance and Minimize classification error.
- **Kernel Trick:** For non-linearly separable data, SVM uses the kernel trick to map the data into a higher-dimensional space where a linear hyperplane can separate the data. Common kernel functions include polynomial kernels and radial basis function (RBF) kernels.

Solving the optimization problem yields the optimal values of  $w$  and  $b$  that define the separating hyperplane. Optimization can be achieved using various optimization algorithms such as the Sequential Minimal Optimization (SMO) algorithm.



## YÊU CẦU CHUNG

### 1) Đánh giá

- Chuẩn bị tốt các yêu cầu đặt ra trong bài thực hành.
- Sinh viên hiểu và tự thực hiện được bài thực hành, trả lời đầy đủ các yêu cầu đặt ra.
- Nộp báo cáo kết quả chi tiết những đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả *(nếu có)*; giải thích cho quan sát *(nếu có)*.
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

### 2) Báo cáo

- File **.PDF** hoặc **.docx**. Tập trung vào nội dung, giải thích.
- Nội dung trình bày bằng Font chữ **Cambria hoặc Times New Roman** (*tuy nhiên, phải chuyển đổi hết báo cáo này sang 1 font chữ thống nhất*) – **cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.**
- Đặt tên theo định dạng: Mã lớp-LabX\_MSSV1\_MSSV2. (trong đó X là Thứ tự buổi Thực hành).  
Ví dụ: NT532.021.1-Lab01\_25520001\_25520002
- Nếu báo cáo có nhiều file, nén tất cả file vào file **.ZIP** với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm bài thực hành.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại [courses.uit.edu.vn](https://courses.uit.edu.vn).

**Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.**

## HẾT