UNIVERSITY of GREENWICH

Alliance with FPT Education

ĐẠI HỌC GREENWICH (VIỆT NAM)

Liên kết với Tổ chức Giáo dục FPT

Applying MTCNN and Facenet Deep Neural Networks for face recognition to check student attendance

COMP1682 Final Year Project

Programme Title, e.g. BSc Computer Science or BSc Computing

Student Name : Vi Long

Student ID : GCS18538

Instructors : Mr. Ho Hai Van, Mr. Le Ngoc Thanh

Submit Date: 28/11/2021

Project Product Link : https://github.com/ViLong/Attendance-System-Using-MTCNN-and-Facenet

Code Explaination

In my project this time, there are a total of 9 .py files containing code (the main.py file only contains the basic code, so I would like to skip it). I will explain the important parts of the code in turn below so that the lecturer can clearly understand what I have implemented in this project. I will explain in turn the important code in each file below:

- **managestudent.py :**

```
# ================ Variables ==================
self.var_fac = StringVar()
self.var_cou = StringVar()
self.var_year = StringVar()
self.var_sem = StringVar()
self.var_std_id = StringVar()
self.var_std_name = StringVar()
self.var_class = StringVar()
self.var_gender = StringVar()
self.var_dob = StringVar()
self.var_email = StringVar()
self.var_phone = StringVar()
self.var_address = StringVar()

# =========== Table Frame =============
table_frame = Frame(right_frame, bd=2, bg='white', relief='ridge')
table_frame.place(x=5, y=80, width=615, height=335)

scroll_x = ttk.Scrollbar(table_frame, orient=HORIZONTAL)
scroll_y = ttk.Scrollbar(table_frame, orient=VERTICAL)

self.student_table = ttk.Treeview(table_frame, column=(
'fac', 'cou', 'year', 'sem', 'id', 'name', 'class', 'gender', 'dob', 'email',
'phone', 'address','photo'),
                                  xscrollcommand=scroll_x.set,
yscrollcommand=scroll_y.set)

scroll_x.pack(side=BOTTOM, fill=X)
scroll_y.pack(side=RIGHT, fill=Y)
scroll_x.config(command=self.student_table.xview)
scroll_y.config(command=self.student_table.yview)

self.student_table['show'] = 'headings'

self.student_table.heading('fac', text='Faculty')
self.student_table.heading('cou', text='Course')
self.student_table.heading('year', text='Year')
self.student_table.heading('sem', text='Semester')
self.student_table.heading('id', text='Student ID')
self.student_table.heading('name', text='Student Name')
self.student_table.heading('class', text='Class')
self.student_table.heading('gender', text='Gender')
self.student_table.heading('dob', text='Birthday')
self.student_table.heading('email', text='Email')
self.student_table.heading('phone', text='Phone')
self.student_table.heading('address', text='Address')
self.student_table.heading('photo', text='Photo Sample Status')


self.student_table.column('fac', width=150)
self.student_table.column('cou', width=100)
self.student_table.column('year', width=100)
self.student_table.column('sem', width=100)
self.student_table.column('id', width=100)
self.student_table.column('name', width=100)
self.student_table.column('class', width=100)
self.student_table.column('gender', width=100)
self.student_table.column('dob', width=100)
self.student_table.column('email', width=100)
self.student_table.column('phone', width=100)
self.student_table.column('address', width=100)
self.student_table.column('photo', width=150)

self.student_table.pack(fill=BOTH, expand=1)
self.student_table.bind('<ButtonRelease>',self.get_cur)
self.fetch_data()
```

This part I declare to create a data frame in the application. This dataframe will display data pulled from a MySQL database. The data will display column by column just like in the MySQL database.

```
# ======================= Fetch Data ==========================
def fetch_data(self):
    conn = mysql.connector.connect(host='localhost', user="root",
password='Vilong242', db='face_recognition')
    cur = conn.cursor()
    cur.execute('select * from student')
    data = cur.fetchall()

    if len(data)!=0:
        self.student_table.delete(*self.student_table.get_children())
        for i in data:
            self.student_table.insert('',END,values=i)
        conn.commit()
    conn.close()
```

This is a short code to connect to the MySQL database, and it will get the data from MySQL to display on the system.

```
def add_data(self):
    if self.var_fac.get()=='Select Faculty' or self.var_std_name.get()=='' or
self.var_std_id.get()=='':
        messagebox.showerror('Error','All field are
required',parent=self.root)
    else:
        try:
            conn = 
mysql.connector.connect(host='localhost',user="root",password='Vilong242',db=
'face_recognition')
            cur=conn.cursor()
            cur.execute('insert into student 
values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)',(
                self.var_fac.get(),
                self.var_cou.get(),
                self.var_year.get(),
                self.var_sem.get(),
                self.var_std_id.get(),
                self.var_std_name.get(),
                self.var_class.get(),
                self.var_gender.get(),
                self.var_dob.get(),
                self.var_email.get(),
                self.var_phone.get(),
                self.var_address.get(),
                self.var_radio1.get()
                ))
            conn.commit()
            self.fetch_data()
            conn.close()
            messagebox.showinfo('Success', 'Student Information has been 
added successfully',parent=self.root)
        except Exception as es:
            messagebox.showerror('Error',f'Due to: 
{str(es)}',parent=self.root)
```

This is a code that helps add a new student information to the database, set up the required fields in the new student registration form and will display a message that the process of adding the student is successful, or meet error.

```
# ======================= Update Data ==========================
def update_data(self):
    if self.var_fac.get()=='Select Faculty' or self.var_std_name.get()=='' or 
self.var_std_id.get()=='':
        messagebox.showerror('Error','All field are 
required',parent=self.root)
    else:
        try:
            update = messagebox.askyesno('Update','Do you want to update this 
student ?',parent = self.root)
            if update>0:
                conn = mysql.connector.connect(host='localhost', user="root", 
password='Vilong242', db='face_recognition')
                cur = conn.cursor()
                cur.execute('update student set 
Faculty=%s,Course=%s,Year=%s,Semester=%s,Name=%s,Class=%s,Gender=%s,DOB=%s,Em
ail=%s,Phone=%s,Address=%s,PhotoSample=%s where Student_ID=%s',(
                    self.var_fac.get(),
                    self.var_cou.get(),
                    self.var_year.get(),
                    self.var_sem.get(),
                    self.var_std_name.get(),
                    self.var_class.get(),
                    self.var_gender.get(),
                    self.var_dob.get(),
                    self.var_email.get(),
                    self.var_phone.get(),
                    self.var_address.get(),
                    self.var_radio1.get(),
                    self.var_std_id.get()
                ))
            else :
                if not update:
                    return
            messagebox.showinfo('Success','Student Information has been
updated successfully',parent=self.root)
            conn.commit()
            self.fetch_data()
            conn.close()
        except Exception as es:
            messagebox.showerror('Error', f'Due to: {str(es)}',
parent=self.root)

# ===================== Delete Function ======================
def delete_data(self):
    if self.var_std_id.get()=='':
        messagebox.showerror('Error','Student ID must be
required',parent=self.root)
    else:
        try:
            delete = messagebox.askyesno('Delete','Do you want delete this
student ?',parent=self.root)
            if delete > 0:
                conn = mysql.connector.connect(host='localhost', user="root",
password='Vilong242',db='face_recognition')
                cur = conn.cursor()
                sql = 'delete from student where Student_ID=%s'
                value = (self.var_std_id.get(),)
                cur.execute(sql,value)
            else:
                if not delete:
                    return
            conn.commit()
            self.fetch_data()
            conn.close()
            messagebox.showinfo('Delete','Student has been deleted
successfully',parent=self.root)
        except Exception as es:
            messagebox.showerror('Error', f'Due to: {str(es)}',
parent=self.root)
```

Two pieces of code I declare two functions to update and delete student data in the MySQL database. Set up the required fields in the new student registration form and will display a message when the process succeeds or encounters an error.

```
#================== Search ==================
def -= (self):
    conn = mysql.connector.connect(host='localhost', user="root",
password='Vilong242', db='face_recognition')
    cur = conn.cursor()
    cur.execute('select * from student where '+str(self.search_by.get())+"
LIKE '%"+str(self.search_data.get())+"%'")
    result = cur.fetchall()

    if len(result)!=0:
        self.student_table.delete(*self.student_table.get_children())
        for i in result:
            self.student_table.insert("",END,values=i)
        conn.commit()
    conn.close()
```

In the above code, I declare a search function to help teachers search for student information in the database.

```
# ========== Generate Dataset or Take Photo button
============================
def generate_dataset(self):
    if self.var_fac.get()=='Select Faculty' or self.var_std_name.get()=='' or
self.var_std_id.get()=='':
        messagebox.showerror('Error','All field are
required',parent=self.root)
    else:
        try:
            conn = mysql.connector.connect(host='localhost', user="root",
password='Vilong242', db='face_recognition')
            cur = conn.cursor()
            cur.execute('select * from student')
            result = cur.fetchall()
            id = 0
            for x in result:
                id += 1
            cur.execute(
            'update student set
Faculty=%s,Course=%s,Year=%s,Semester=%s,Name=%s,Class=%s,Gender=%s,DOB=%s,Em
ail=%s,Phone=%s,Address=%s,PhotoSample=%s where Student_ID=%s',
                (
                    self.var_fac.get(),
                    self.var_cou.get(),
                    self.var_year.get(),
                    self.var_sem.get(),
                    self.var_std_name.get(),
                    self.var_class.get(),
                    self.var_gender.get(),
                    self.var_dob.get(),
                    self.var_email.get(),
                    self.var_phone.get(),
                    self.var_address.get(),
                    self.var_radio1.get(),
                    self.var_std_id.get() == id+1
                ))

            conn.commit()
            name = str(self.var_std_name.get())
            os.makedirs('dataSet/' +name+ '-' +str(id)+'')
            self.fetch_data()
            self.reset_data()
            conn.close()
```

I implemented the code to declare the generate_dataset function, the purpose of this function is to collect the face images of the students in the database. The above code helps to connect to the MySQL database, to save the collected face images by student ID. Collected images will be stored in dataSet folder.

```
# ============== Load Predifined data ==============
detector = MTCNN()
camera = cv2.VideoCapture(0)
sampleNum = 0

while True:
    ret, my_frame = camera.read()

    my_frame = cv2.flip(my_frame, 1)
    faces = detector.detect_faces(my_frame)
```

```
                if faces != [] :
                    for person in faces:
                        bounding_box = person['box']
                        keypoints = person['keypoints']

                        cv2.rectangle(my_frame,
                                      (bounding_box[0], bounding_box[1]),
                                      (bounding_box[0] + bounding_box[2],
bounding_box[1] + bounding_box[3]),
                                      (0, 155, 255),
                                      2)

                        cv2.circle(my_frame, (keypoints['left_eye']), 2, (0,
155, 255), 2)
                        cv2.circle(my_frame, (keypoints['right_eye']), 2, (0,
155, 255), 2)
                        cv2.circle(my_frame, (keypoints['nose']), 2, (0, 155,
255), 2)
                        cv2.circle(my_frame, (keypoints['mouth_left']), 2,
(0, 155, 255), 2)
                        cv2.circle(my_frame, (keypoints['mouth_right']), 2,
(0, 155, 255), 2)

                        sampleNum += 1

                        file_path = "dataSet/" +str(name)+ '-' +str(id)+''
"/" +str(name)+ "-" +str(id)+ "." + str(sampleNum) + ".jpg"
                        cv2.imwrite(file_path,my_frame)

                    cv2.putText(my_frame, str(sampleNum), (50, 50),
cv2.FONT_HERSHEY_COMPLEX, 2, (100, 200, 200), 2)
                    cv2.imshow('Take Photo', my_frame)

                if cv2.waitKey(100) & 0xFF == ord('q'):
                    break
                elif sampleNum > 100:
                    break
```

```
            camera.release()
            cv2.destroyAllWindows()
            messagebox.showinfo('Result', 'Taking photo completed !!!')

        except Exception as es:
            messagebox.showerror('Error', f'Due to: {str(es)}',
parent=self.root)
```

The above code declares to perform facial photography of students. I declare a variable to execute the MTCNN algorithm that I implemented in another .py file. Assign the camera variable to let the system use the webcam to collect images. Use OpenCV to represent and mark landmarks of faces detected in the image. The captured images will be saved in a folder with the student's name and id. Setup when the system captures one hundred images will stop and display a success message.

- **attendance.py :**

```
def fetch_data(self):
    with open('attendance.csv') as file:
        df = pd.read_csv('attendance.csv')

    self.attendance_table['column'] = list(df.columns)
    self.attendance_table["show"] = 'headings'
    for column in self.attendance_table['columns']:
        self.attendance_table.heading(column,text=column)

    self.attendance_table.delete(*self.attendance_table.get_children())
    df_rows= df.to_numpy().tolist()
    for row in df_rows:
        self.attendance_table.insert("",END,values = row)
```

This is a short code to connect to the MySQL database, and it will get the data from MySQL to display on the system.

```
def search(self):
    with open('attendance.csv','r') as file:
        filereader = csv.reader(file)
        filereader_list = list(filereader)

    self.attendance_table.delete(*self.attendance_table.get_children())
    word = self.name_data.get().title()
    if self.name_data.get():
        for i in filereader_list:
            if word in i:
                self.attendance_table.insert("", END, values=i)
```

In the above code, I declare a search function to help teachers search for student information in the database.

```
def sendemail(self):
    curr = datetime.now()
    dt = curr.strftime("%d/%m/%Y")

    subject = "Data Attendance"
    body = "Here are the data attendance"
    sender_email = "universitygreenwichattendance@gmail.com"
    receiver_email = "vilong242@gmail.com"
    password = 'mkbwaznnpkxulxiy'

    # Create a multipart message and set headers
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = subject
    message["Bcc"] = receiver_email  # Recommended for mass emails

    # Add body to email
    message.attach(MIMEText(body, "plain"))

    filename = "attendance.csv"  # In same directory as script

    # Open PDF file in binary mode
    with open(filename, "rb") as attachment:
        # Add file as application/octet-stream
        # Email client can usually download this automatically as attachment
        part = MIMEBase("application", "octet-stream")
        part.set_payload(attachment.read())

    # Encode file in ASCII characters to send by email
    encoders.encode_base64(part)

    # Add header as key/value pair to attachment part
    part.add_header(
        "Content-Disposition",
        f"attachment; filename= {filename}",
    )

    # Add attachment to message and convert message to string
    message.attach(part)
    text = message.as_string()

    # Log in to server using secure context and send email
    context = ssl.create_default_context()
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, text)
        messagebox.showinfo('Successfull', f'Attendance data on %s has been
sent'%dt)
```

In the above code, I declare a sendemail() function so that teachers can send a list of registered students to their personal email. I use python's smtplib library to setup SMTP Client to send email to another Email address on Internet via SMTP protocol. And when the user bbaasm button, the email will be automatically sent to the specified email. Besides, the system also displays a message sent successfully.

- **detect_face.py :**

The use of this .py file is to use OpenCV to create processing functions and create classes used for the face recognition algorithm MTCNN.

```
class Network(object):

    def __init__(self, inputs, trainable=True):
        # The input nodes for this network
        self.inputs = inputs
        # The current list of terminal nodes
        self.terminals = []
        # Mapping from layer names to layers
        self.layers = dict(inputs)
        # If true, the resulting variables are set as trainable
        self.trainable = trainable

        self.setup()

    def setup(self):
        """Construct the network. """
        raise NotImplementedError('Must be implemented by the subclass.')

    def load(self, data_path, session, ignore_missing=False):
        """Load network weights.
        data_path: The path to the numpy-serialized network weights
        session: The current TensorFlow session
        ignore_missing: If true, serialized weights for missing layers are 
ignored.
        """
        data_dict = np.load(data_path, encoding='latin1', 
allow_pickle=True).item() #pylint: disable=no-member

        for op_name in data_dict:
            with tf.variable_scope(op_name, reuse=True):
                for param_name, data in iteritems(data_dict[op_name]):
                    try:
                        var = tf.get_variable(param_name)
                        session.run(var.assign(data))
                    except ValueError:
                        if not ignore_missing:
                            raise
    def feed(self, *args):
        """Set the input(s) for the next operation by replacing the terminal
nodes.
        The arguments can be either layer names or the actual layers.
        """
        assert len(args) != 0
        self.terminals = []
        for fed_layer in args:
            if isinstance(fed_layer, string_types):
                try:
                    fed_layer = self.layers[fed_layer]
                except KeyError:
                    raise KeyError('Unknown layer name fed: %s' % fed_layer)
            self.terminals.append(fed_layer)
        return self

    def get_output(self):
        """Returns the current network output."""
        return self.terminals[-1]

    def get_unique_name(self, prefix):
        """Returns an index-suffixed unique name for the given prefix.
        This is used for auto-generating layer names based on the type-
prefix.
        """
        ident = sum(t.startswith(prefix) for t, _ in self.layers.items()) + 1
        return '%s_%d' % (prefix, ident)

    def make_var(self, name, shape):
        """Creates a new TensorFlow variable."""
        return tf.get_variable(name, shape, trainable=self.trainable)

    def validate_padding(self, padding):
        """Verifies that the padding is one of the supported ones."""
        assert padding in ('SAME', 'VALID')
```

The above section will declare the overall classes that will be used in the MTCNN algorithm, want to know more about the classes in the algorithm. Please review the report file that I submitted. There I explained in detail the classes included in the MTCNN algorithm

```
    @layer
    def conv(self,
             inp,
             k_h,
             k_w,
             c_o,
             s_h,
             s_w,
             name,
             relu=True,
             padding='SAME',
             group=1,
             biased=True):
        # Verify that the padding is acceptable
        self.validate_padding(padding)
        # Get the number of channels in the input
        c_i = int(inp.get_shape()[-1])
        # Verify that the grouping parameter is valid
        assert c_i % group == 0
        assert c_o % group == 0
        # Convolution for a given input and kernel
        convolve = lambda i, k: tf.nn.conv2d(i, k, [1, s_h, s_w, 1], padding=padding)
        with tf.variable_scope(name) as scope:
            kernel = self.make_var('weights', shape=[k_h, k_w, c_i // group, c_o])
            # This is the common-case. Convolve the input without any further complications.
            output = convolve(inp, kernel)
            # Add the biases
            if biased:
                biases = self.make_var('biases', [c_o])
                output = tf.nn.bias_add(output, biases)
            if relu:
                # ReLU non-linearity
                output = tf.nn.relu(output, name=scope.name)
            return output
```

Declare the Convolution Layer in the MTCNN algorithm

```
    @layer
    def prelu(self, inp, name):
        with tf.variable_scope(name):
            i = int(inp.get_shape()[-1])
            alpha = self.make_var('alpha', shape=(i,))
            output = tf.nn.relu(inp) + tf.multiply(alpha, -tf.nn.relu(-inp))
    return output
```

Declare the PreLu Layer in the MTCNN algorithm

```
@layer
def max_pool(self, inp, k_h, k_w, s_h, s_w, name, padding='SAME'):
    self.validate_padding(padding)
    return tf.nn.max_pool(inp,
                          ksize=[1, k_h, k_w, 1],
                          strides=[1, s_h, s_w, 1],
                          padding=padding,
                          name=name)
```

Declare the Maxpool Layer in the MTCNN algorithm

```
@layer
def fc(self, inp, num_out, name, relu=True):
    with tf.variable_scope(name):
        input_shape = inp.get_shape()
        if input_shape.ndims == 4:
            # The input is spatial. Vectorize it first.
            dim = 1
            for d in input_shape[1:].as_list():
                dim *= int(d)
            feed_in = tf.reshape(inp, [-1, dim])
        else:
            feed_in, dim = (inp, input_shape[-1].value)
        weights = self.make_var('weights', shape=[dim, num_out])
        biases = self.make_var('biases', [num_out])
        op = tf.nn.relu_layer if relu else tf.nn.xw_plus_b
        fc = op(feed_in, weights, biases, name=name)
        return fc
```

Declare the Fully Connected Layer in the MTCNN algorithm

```
 @layer
def softmax(self, target, axis, name=None):
    max_axis = tf.reduce_max(target, axis, keepdims=True)
```

```
        target_exp = tf.exp(target-max_axis)
        normalize = tf.reduce_sum(target_exp, axis, keepdims=True)
        softmax = tf.div(target_exp, normalize, name)
        return softmax
```

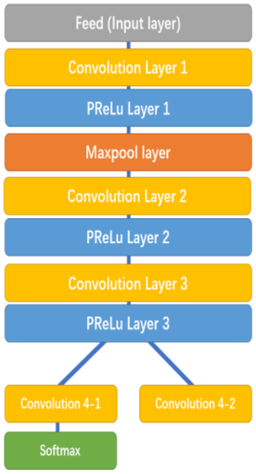
Declare the Softmax Layer in the MTCNN algorithm

Next, after declaring the classes used in the algorithm, I will in turn arrange the classes that have just formed into the 3 main networks of MTCNN that are: P -Net, R-Net and O-Net

```
class PNet(Network):
    def setup(self):
        (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
            .conv(3, 3, 10, 1, 1, padding='VALID', relu=False, name='conv1')
            .prelu(name='PReLU1')
            .max_pool(2, 2, 2, 2, name='pool1')
            .conv(3, 3, 16, 1, 1, padding='VALID', relu=False, name='conv2')
            .prelu(name='PReLU2')
            .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv3')
            .prelu(name='PReLU3')
            .conv(1, 1, 2, 1, 1, relu=False, name='conv4-1')
            .softmax(3,name='prob1'))

        (self.feed('PReLU3') #pylint: disable=no-value-for-parameter
            .conv(1, 1, 4, 1, 1, relu=False, name='conv4-2'))
```

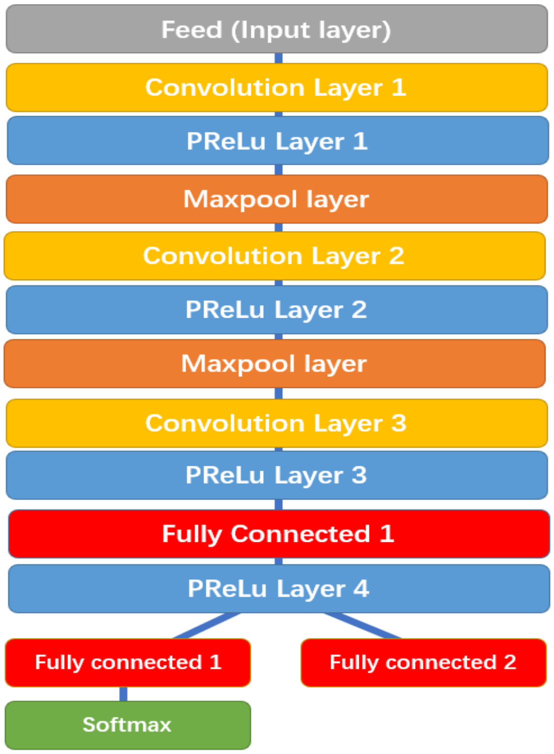
Above is the arrangement of the P-Net class in order:



```
class RNet(Network):
    def setup(self):
        (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
             .conv(3, 3, 28, 1, 1, padding='VALID', relu=False, name='conv1')
             .prelu(name='prelu1')
             .max_pool(3, 3, 2, 2, name='pool1')
             .conv(3, 3, 48, 1, 1, padding='VALID', relu=False, name='conv2')
             .prelu(name='prelu2')
             .max_pool(3, 3, 2, 2, padding='VALID', name='pool2')
             .conv(2, 2, 64, 1, 1, padding='VALID', relu=False, name='conv3')
             .prelu(name='prelu3')
             .fc(128, relu=False, name='conv4')
             .prelu(name='prelu4')
             .fc(2, relu=False, name='conv5-1')
             .softmax(1,name='prob1'))

        (self.feed('prelu4') #pylint: disable=no-value-for-parameter
             .fc(4, relu=False, name='conv5-2'))
```

Above is the arrangement of the R-Net class in order:

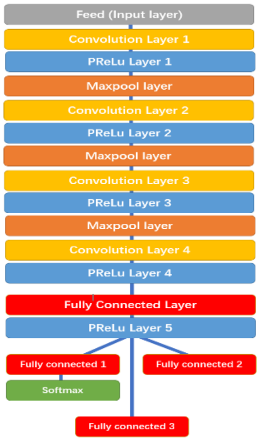


```
class ONet(Network):
    def setup(self):
        (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
             .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv1')
             .prelu(name='prelu1')
             .max_pool(3, 3, 2, 2, name='pool1')
             .conv(3, 3, 64, 1, 1, padding='VALID', relu=False, name='conv2')
             .prelu(name='prelu2')
             .max_pool(3, 3, 2, 2, padding='VALID', name='pool2')
             .conv(3, 3, 64, 1, 1, padding='VALID', relu=False, name='conv3')
             .prelu(name='prelu3')
             .max_pool(2, 2, 2, 2, name='pool3')
             .conv(2, 2, 128, 1, 1, padding='VALID', relu=False,
name='conv4')
             .prelu(name='prelu4')
             .fc(256, relu=False, name='conv5')
             .prelu(name='prelu5')
             .fc(2, relu=False, name='conv6-1')
             .softmax(1, name='prob1'))

        (self.feed('prelu5') #pylint: disable=no-value-for-parameter
             .fc(4, relu=False, name='conv6-2'))

        (self.feed('prelu5') #pylint: disable=no-value-for-parameter
             .fc(10, relu=False, name='conv6-3'))
```

And finally the O-Net class:



```
def create_mtcnn(sess, model_path):
    if not model_path:
        model_path,_ = os.path.split(os.path.realpath(__file__))

    with tf.variable_scope('pnet'):
        data = tf.placeholder(tf.float32, (None,None,None,3), 'input')
        pnet = PNet({'data':data})
        pnet.load(os.path.join(model_path, 'det1.npy'), sess)
    with tf.variable_scope('rnet'):
        data = tf.placeholder(tf.float32, (None,24,24,3), 'input')
        rnet = RNet({'data':data})
        rnet.load(os.path.join(model_path, 'det2.npy'), sess)
    with tf.variable_scope('onet'):
        data = tf.placeholder(tf.float32, (None,48,48,3), 'input')
        onet = ONet({'data':data})
        onet.load(os.path.join(model_path, 'det3.npy'), sess)

    pnet_fun = lambda img : sess.run(('pnet/conv4-2/BiasAdd:0',
'pnet/prob1:0'), feed_dict={'pnet/input:0':img})
    rnet_fun = lambda img : sess.run(('rnet/conv5-2/conv5-2:0',
'rnet/prob1:0'), feed_dict={'rnet/input:0':img})
    onet_fun = lambda img : sess.run(('onet/conv6-2/conv6-2:0', 'onet/conv6-
3/conv6-3:0', 'onet/prob1:0'), feed_dict={'onet/input:0':img})
    return pnet_fun, rnet_fun, onet_fun
```

Finally, set up a complete MTCNN algorithm

Next, after creating a complete MTCNN algorithm. The next thing, I need to declare a detect_face function to apply the MTCNN algorithm I just code in to be able to recognize faces.

```
def detect_face(img, minsize, pnet, rnet, onet, threshold, factor):
    """Detects faces in an image, and returns bounding boxes and points for
them.
    img: input image
    minsize: minimum faces' size
    pnet, rnet, onet: caffemodel
    threshold: threshold=[th1, th2, th3], th1-3 are three steps's threshold
    factor: the factor used to create a scaling pyramid of face sizes to
detect in the image.
    """
    factor_count=0
    total_boxes=np.empty((0,9))
    points=np.empty(0)
    h=img.shape[0]
    w=img.shape[1]
    minl=np.amin([h, w])
    m=12.0/minsize
    minl=minl*m
    # create scale pyramid
    scales=[]
    while minl>=12:
        scales += [m*np.power(factor, factor_count)]
        minl = minl*factor
        factor_count += 1
```

It will first create a pyramid to scale the entire input image

```
    # first stage
    for scale in scales:
        hs=int(np.ceil(h*scale))
        ws=int(np.ceil(w*scale))
        im_data = imresample(img, (hs, ws))
        im_data = (im_data-127.5)*0.0078125
        img_x = np.expand_dims(im_data, 0)
        img_y = np.transpose(img_x, (0,2,1,3))
        out = pnet(img_y)
        out0 = np.transpose(out[0], (0,2,1,3))
        out1 = np.transpose(out[1], (0,2,1,3))

        boxes, _ = generateBoundingBox(out1[0,:,:,1].copy(),
out0[0,:,:,:].copy(), scale, threshold[0])

        # inter-scale nms
        pick = nms(boxes.copy(), 0.5, 'Union')
        if boxes.size>0 and pick.size>0:
            boxes = boxes[pick,:]
            total_boxes = np.append(total_boxes, boxes, axis=0)

    numbox = total_boxes.shape[0]
    if numbox>0:
        pick = nms(total_boxes.copy(), 0.7, 'Union')
        total_boxes = total_boxes[pick,:]
        regw = total_boxes[:,2]-total_boxes[:,0]
        regh = total_boxes[:,3]-total_boxes[:,1]
        qq1 = total_boxes[:,0]+total_boxes[:,5]*regw
        qq2 = total_boxes[:,1]+total_boxes[:,6]*regh
        qq3 = total_boxes[:,2]+total_boxes[:,7]*regw
        qq4 = total_boxes[:,3]+total_boxes[:,8]*regh
        total_boxes = np.transpose(np.vstack([qq1, qq2, qq3, qq4,
total_boxes[:,4]]))
        total_boxes = rerec(total_boxes.copy())
        total_boxes[:,0:4] = np.fix(total_boxes[:,0:4]).astype(np.int32)
        dy, edy, dx, edx, y, ey, x, ex, tmpw, tmph = pad(total_boxes.copy(),
w, h)

    numbox = total_boxes.shape[0]
    if numbox>0:
        # second stage
        tempimg = np.zeros((24,24,3,numbox))
        for k in range(0,numbox):
            tmp = np.zeros((int(tmph[k]),int(tmpw[k]),3))
            tmp[dy[k]-1:edy[k],dx[k]-1:edx[k],:] = img[y[k]-1:ey[k],x[k]-
1:ex[k],:]
            if tmp.shape[0]>0 and tmp.shape[1]>0 or tmp.shape[0]==0 and
tmp.shape[1]==0:
                tempimg[:,:,:,k] = imresample(tmp, (24, 24))
            else:
                return np.empty()
        tempimg = (tempimg-127.5)*0.0078125
        tempimg1 = np.transpose(tempimg, (3,1,0,2))
        out = rnet(tempimg1)
        out0 = np.transpose(out[0])
        out1 = np.transpose(out[1])
        score = out1[1,:]
        ipass = np.where(score>threshold[1])
        total_boxes = np.hstack([total_boxes[ipass[0],0:4].copy(), np.expand_dims(score[ipass].copy(),1)])
        mv = out0[:,ipass[0]]
        if total_boxes.shape[0]>0:
            pick = nms(total_boxes, 0.7, 'Union')
            total_boxes = total_boxes[pick,:]
            total_boxes = bbreg(total_boxes.copy(), np.transpose(mv[:,pick]))
            total_boxes = rerec(total_boxes.copy())

    numbox = total_boxes.shape[0]
    if numbox>0:
        # third stage
        total_boxes = np.fix(total_boxes).astype(np.int32)
        dy, edy, dx, edx, y, ey, x, ex, tmpw, tmph = pad(total_boxes.copy(), w, h)
        tempimg = np.zeros((48,48,3,numbox))
        for k in range(0,numbox):
            tmp = np.zeros((int(tmph[k]),int(tmpw[k]),3))
            tmp[dy[k]-1:edy[k],dx[k]-1:edx[k],:] = img[y[k]-1:ey[k],x[k]-1:ex[k],:]
            if tmp.shape[0]>0 and tmp.shape[1]>0 or tmp.shape[0]==0 and tmp.shape[1]==0:
                tempimg[:,:,:,k] = imresample(tmp, (48, 48))
            else:
                return np.empty()
        tempimg = (tempimg-127.5)*0.0078125
        tempimg1 = np.transpose(tempimg, (3,1,0,2))
        out = onet(tempimg1)
        out0 = np.transpose(out[0])
        out1 = np.transpose(out[1])
        out2 = np.transpose(out[2])
        score = out2[1,:]
        points = out1
        ipass = np.where(score>threshold[2])
        points = points[:,ipass[0]]
        total_boxes = np.hstack([total_boxes[ipass[0],0:4].copy(),
np.expand_dims(score[ipass].copy(),1)])
        mv = out0[:,ipass[0]]

        w = total_boxes[:,2]-total_boxes[:,0]+1
        h = total_boxes[:,3]-total_boxes[:,1]+1
        points[0:5,:] = np.tile(w,(5, 1))*points[0:5,:] +
np.tile(total_boxes[:,0],(5, 1))-1
        points[5:10,:] = np.tile(h,(5, 1))*points[5:10,:] +
np.tile(total_boxes[:,1],(5, 1))-1
        if total_boxes.shape[0]>0:
            total_boxes = bbreg(total_boxes.copy(), np.transpose(mv))
            pick = nms(total_boxes.copy(), 0.7, 'Min')
            total_boxes = total_boxes[pick,:]
            points = points[:,pick]

    return total_boxes, points
```

Then, set up the system to scan the input image, apply the MTCNN algorithm just declared above and return the results as the location coordinates and landmarks of the face.

```
# function [boundingbox] = bbreg(boundingbox,reg)
def bbreg(boundingbox,reg):
    """Calibrate bounding boxes"""
    if reg.shape[1]==1:
        reg = np.reshape(reg, (reg.shape[2], reg.shape[3]))

    w = boundingbox[:,2]-boundingbox[:,0]+1
    h = boundingbox[:,3]-boundingbox[:,1]+1
    b1 = boundingbox[:,0]+reg[:,0]*w
    b2 = boundingbox[:,1]+reg[:,1]*h
    b3 = boundingbox[:,2]+reg[:,2]*w
    b4 = boundingbox[:,3]+reg[:,3]*h
    boundingbox[:,0:4] = np.transpose(np.vstack([b1, b2, b3, b4 ]))
    return boundingbox
```

```
def generateBoundingBox(imap, reg, scale, t):
    """Use heatmap to generate bounding boxes"""
    stride=2
    cellsize=12

    imap = np.transpose(imap)
    dx1 = np.transpose(reg[:,:,0])
    dy1 = np.transpose(reg[:,:,1])
    dx2 = np.transpose(reg[:,:,2])
    dy2 = np.transpose(reg[:,:,3])
    y, x = np.where(imap >= t)
    if y.shape[0]==1:
        dx1 = np.flipud(dx1)
        dy1 = np.flipud(dy1)
        dx2 = np.flipud(dx2)
        dy2 = np.flipud(dy2)
    score = imap[(y,x)]
    reg = np.transpose(np.vstack([ dx1[(y,x)], dy1[(y,x)], dx2[(y,x)],
dy2[(y,x)] ]))
    if reg.size==0:
        reg = np.empty((0,3))
    bb = np.transpose(np.vstack([y,x]))
    q1 = np.fix((stride*bb+1)/scale)
    q2 = np.fix((stride*bb+cellsize-1+1)/scale)
    boundingbox = np.hstack([q1, q2, np.expand_dims(score,1), reg])
    return boundingbox, reg
```

Finally, we will declare 2 more functions to help draw boxes on the screen to mark faces on the input image

- **facenet.py :**

Next, will come to the file facenet.py. The goal of this file is to create a Facenet algorithm to apply to face recognition techniques

```
def triplet_loss(anchor, positive, negative, alpha):
    """Calculate the triplet loss according to the FaceNet paper

    Args:
      anchor: the embeddings for the anchor images.
      positive: the embeddings for the positive images.
      negative: the embeddings for the negative images.

    Returns:
      the triplet loss according to the FaceNet paper as a float tensor.
    """
    with tf.variable_scope('triplet_loss'):
        pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor, positive)),
1)
        neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor, negative)),
1)

        basic_loss = tf.add(tf.subtract(pos_dist, neg_dist), alpha)
        loss = tf.reduce_mean(tf.maximum(basic_loss, 0.0), 0)

    return loss


def center_loss(features, label, alfa, nrof_classes):

    nrof_features = features.get_shape()[1]
    centers = tf.get_variable('centers', [nrof_classes, nrof_features],
dtype=tf.float32,
                              initializer=tf.constant_initializer(0),
trainable=False)
    label = tf.reshape(label, [-1])
    centers_batch = tf.gather(centers, label)
    diff = (1 - alfa) * (centers_batch - features)
    centers = tf.scatter_sub(centers, label, diff)
    with tf.control_dependencies([centers]):
        loss = tf.reduce_mean(tf.square(features - centers_batch))
    return loss, centers
```

First, I declare the function triplet loss, center loss of the Facenet algorithm. It will return a loss value and a centers value. These are the two decisive values to calculate the distance between the two vectors after being extracted

```
def create_input_pipeline(input_queue, image_size,
nrof_preprocess_threads, batch_size_placeholder):
    images_and_labels_list = []
    for _ in range(nrof_preprocess_threads):
        filenames, label, control = input_queue.dequeue()
        images = []
        for filename in tf.unstack(filenames):
            file_contents = tf.read_file(filename)
            image = tf.image.decode_image(file_contents, 3)
            image = tf.cond(get_control_flag(control[0], RANDOM_ROTATE),
                            lambda: tf.py_func(random_rotate_image,
[image], tf.uint8),
                            lambda: tf.identity(image))
            image = tf.cond(get_control_flag(control[0], RANDOM_CROP),
                            lambda: tf.random_crop(image, image_size +
(3,)),
                            lambda:
tf.image.resize_image_with_crop_or_pad(image, image_size[0],
image_size[1]))
            image = tf.cond(get_control_flag(control[0], RANDOM_FLIP),
                            lambda:
tf.image.random_flip_left_right(image),
                            lambda: tf.identity(image))
            image = tf.cond(get_control_flag(control[0],
FIXED_STANDARDIZATION),
                            lambda: (tf.cast(image, tf.float32) - 127.5) /
128.0,
                            lambda:
tf.image.per_image_standardization(image))
            image = tf.cond(get_control_flag(control[0], FLIP),
                            lambda: tf.image.flip_left_right(image),
                            lambda: tf.identity(image))
            # pylint: disable=no-member
            image.set_shape(image_size + (3,))
            images.append(image)
        images_and_labels_list.append([images, label])

    image_batch, label_batch = tf.train.batch_join(
        images_and_labels_list, batch_size=batch_size_placeholder,
        shapes=[image_size + (3,), ()], enqueue_many=True,
        capacity=4 * nrof_preprocess_threads * 100,
        allow_smaller_final_batch=True)

    return image_batch, label_batch
```

Next, I declare and set the path for the input images (specifically here are the face images that have been detected at the MTCNN algorithm). After the system calculates and processes complex expressions with the help of tensorflow. The result will return the vector and label values of the input images

```
def get_control_flag(control, field):
    return tf.equal(tf.mod(tf.floor_div(control, field), 2), 1)


def _add_loss_summaries(total_loss):
    """Add summaries for losses.

    Generates moving average for all losses and associated summaries for
    visualizing the performance of the network.

    Args:
      total_loss: Total loss from loss().
    Returns:
      loss_averages_op: op for generating moving averages of losses.
    """
    # Compute the moving average of all individual losses and the total
loss.
    loss_averages = tf.train.ExponentialMovingAverage(0.9, name='avg')
    losses = tf.get_collection('losses')
    loss_averages_op = loss_averages.apply(losses + [total_loss])

    # Attach a scalar summmary to all individual losses and the total
loss; do the
    # same for the averaged version of the losses.
    for l in losses + [total_loss]:
        # Name each loss as '(raw)' and name the moving average version of
the loss
        # as the original loss name.
        tf.summary.scalar(l.op.name + ' (raw)', l)
        tf.summary.scalar(l.op.name, loss_averages.average(l))

    return loss_averages_op
```

In the above code, I proceed to calculate the average value of the loss values declared above. This value is required to train data

```
def train(total_loss, global_step, optimizer, learning_rate,
moving_average_decay, update_gradient_vars,
          log_histograms=True):
    # Generate moving averages of all losses and associated summaries.
    loss_averages_op = _add_loss_summaries(total_loss)

    # Compute gradients.
    with tf.control_dependencies([loss_averages_op]):
        if optimizer == 'ADAGRAD':
            opt = tf.train.AdagradOptimizer(learning_rate)
        elif optimizer == 'ADADELTA':
            opt = tf.train.AdadeltaOptimizer(learning_rate, rho=0.9,
epsilon=1e-6)
        elif optimizer == 'ADAM':
            opt = tf.train.AdamOptimizer(learning_rate, beta1=0.9,
beta2=0.999, epsilon=0.1)
        elif optimizer == 'RMSPROP':
            opt = tf.train.RMSPropOptimizer(learning_rate, decay=0.9,
momentum=0.9, epsilon=1.0)
        elif optimizer == 'MOM':
            opt = tf.train.MomentumOptimizer(learning_rate, 0.9,
use_nesterov=True)
        else:
            raise ValueError('Invalid optimization algorithm')

        grads = opt.compute_gradients(total_loss, update_gradient_vars)
    # Apply gradients.
    apply_gradient_op = opt.apply_gradients(grads,
global_step=global_step)

    # Add histograms for trainable variables.
    if log_histograms:
        for var in tf.trainable_variables():
            tf.summary.histogram(var.op.name, var)

    # Add histograms for gradients.
    if log_histograms:
        for grad, var in grads:
            if grad is not None:
                tf.summary.histogram(var.op.name + '/gradients', grad)

    # Track the moving averages of all trainable variables.
    variable_averages = tf.train.ExponentialMovingAverage(
        moving_average_decay, global_step)
    variables_averages_op =
variable_averages.apply(tf.trainable_variables())

    with tf.control_dependencies([apply_gradient_op,
variables_averages_op]):
        train_op = tf.no_op(name='train')

    return train_op
```

Then, I proceed to declare a data train function so that the system can learn the existing face data in the system with the help of tensorflow.

```
def prewhiten(x):
    mean = np.mean(x)
    std = np.std(x)
    std_adj = np.maximum(std, 1.0 / np.sqrt(x.size))
    y = np.multiply(np.subtract(x, mean), 1 / std_adj)
    return y
```

```
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):
        img = misc.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


def get_label_batch(label_data, batch_size, batch_index):
    nrof_examples = np.size(label_data, 0)
    j = batch_index * batch_size % nrof_examples
    if j + batch_size <= nrof_examples:
        batch = label_data[j:j + batch_size]
    else:
        x1 = label_data[j:nrof_examples]
        x2 = label_data[0:nrof_examples - j]
        batch = np.vstack([x1, x2])
    batch_int = batch.astype(np.int64)
    return batch_int


def get_batch(image_data, batch_size, batch_index):
    nrof_examples = np.size(image_data, 0)
    j = batch_index * batch_size % nrof_examples
    if j + batch_size <= nrof_examples:
        batch = image_data[j:j + batch_size, :, :, :]
    else:
        x1 = image_data[j:nrof_examples, :, :, :]
        x2 = image_data[0:nrof_examples - j, :, :, :]
        batch = np.vstack([x1, x2])
    batch_float = batch.astype(np.float32)
    return batch_float


def get_triplet_batch(triplets, batch_index, batch_size):
    ax, px, nx = triplets
    a = get_batch(ax, int(batch_size / 3), batch_index)
    p = get_batch(px, int(batch_size / 3), batch_index)
    n = get_batch(nx, int(batch_size / 3), batch_index)
    batch = np.vstack([a, p, n])
    return batch


def get_learning_rate_from_file(filename, epoch):
    with open(filename, 'r') as f:
        for line in f.readlines():
            line = line.split('#', 1)[0]
            if line:
                par = line.strip().split(':')
                e = int(par[0])
                if par[1] == '-':
                    lr = -1
                else:
                    lr = float(par[1])
                if e <= epoch:
                    learning_rate = lr
                else:
                    return learning_rate
```

Declare the above functions so that the system can calculate the learning rate. This value is a required value to be able to train data

```
class ImageClass():
    "Stores the paths to images for a given class"

    def __init__(self, name, image_paths):
        self.name = name
        self.image_paths = image_paths

    def __str__(self):
        return self.name + ', ' + str(len(self.image_paths)) + ' images'

    def __len__(self):
        return len(self.image_paths)

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.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))
```

```
def get_model_filenames(model_dir):
    files = os.listdir(model_dir)
    meta_files = [s for s in files if s.endswith('.meta')]
    if len(meta_files) == 0:
        raise ValueError('No meta file found in the model directory (%s)'
% model_dir)
    elif len(meta_files) > 1:
        raise ValueError('There should not be more than one meta file in
the model directory (%s)' % model_dir)
    meta_file = meta_files[0]
    ckpt = tf.train.get_checkpoint_state(model_dir)
    if ckpt and ckpt.model_checkpoint_path:
        ckpt_file = os.path.basename(ckpt.model_checkpoint_path)
        return meta_file, ckpt_file

    meta_files = [s for s in files if '.ckpt' in s]
    max_step = -1
    for f in files:
        step_str = re.match(r'(^model-[\w\- ]+.ckpt-(\d+))', f)
        if step_str is not None and len(step_str.groups()) >= 2:
            step = int(step_str.groups()[1])
            if step > max_step:
                max_step = step
                ckpt_file = step_str.groups()[0]
    return meta_file, ckpt_file
```

After the system has finished training the data, the newly trained memorized information will be stored in a model. The above function helps to save the model to the storage directory.

```
def distance(embeddings1, embeddings2, distance_metric=0):
    if distance_metric == 0:
        # Euclidian distance
        diff = np.subtract(embeddings1, embeddings2)
        dist = np.sum(np.square(diff), 1)
    elif distance_metric == 1:
        # Distance based on cosine similarity
        dot = np.sum(np.multiply(embeddings1, embeddings2), axis=1)
        norm = np.linalg.norm(embeddings1, axis=1) *
np.linalg.norm(embeddings2, axis=1)
        similarity = dot / norm
        dist = np.arccos(similarity) / math.pi
    else:
        raise 'Undefined distance metric %d' % distance_metric

    return dist
```

Next, I will declare a function that calculates and returns the value of the distance between the vector and the trained vectors. The smaller the distance between the vectors, the greater the similarity.

```
def calculate_roc(thresholds, embeddings1, embeddings2, actual_issame,
nrof_folds=10, distance_metric=0,
                  subtract_mean=False):
    assert (embeddings1.shape[0] == embeddings2.shape[0])
    assert (embeddings1.shape[1] == embeddings2.shape[1])
    nrof_pairs = min(len(actual_issame), embeddings1.shape[0])
    nrof_thresholds = len(thresholds)
    k_fold = KFold(n_splits=nrof_folds, shuffle=False)

    tprs = np.zeros((nrof_folds, nrof_thresholds))
    fprs = np.zeros((nrof_folds, nrof_thresholds))
    accuracy = np.zeros((nrof_folds))

    indices = np.arange(nrof_pairs)

    for fold_idx, (train_set, test_set) in
enumerate(k_fold.split(indices)):
        if subtract_mean:
            mean = np.mean(np.concatenate([embeddings1[train_set],
embeddings2[train_set]]), axis=0)
        else:
            mean = 0.0
        dist = distance(embeddings1 - mean, embeddings2 - mean,
distance_metric)

        # Find the best threshold for the fold
        acc_train = np.zeros((nrof_thresholds))
        for threshold_idx, threshold in enumerate(thresholds):
            _, _, acc_train[threshold_idx] = calculate_accuracy(threshold,
dist[train_set], actual_issame[train_set])
        best_threshold_index = np.argmax(acc_train)
        for threshold_idx, threshold in enumerate(thresholds):
            tprs[fold_idx, threshold_idx], fprs[fold_idx, threshold_idx],
_ = calculate_accuracy(threshold,

dist[test_set],

actual_issame[

test_set])
        _, _, accuracy[fold_idx] =
calculate_accuracy(thresholds[best_threshold_index], dist[test_set],

actual_issame[test_set])

        tpr = np.mean(tprs, 0)
        fpr = np.mean(fprs, 0)
    return tpr, fpr, accuracy


def calculate_accuracy(threshold, dist, actual_issame):
    predict_issame = np.less(dist, threshold)
    tp = np.sum(np.logical_and(predict_issame, actual_issame))
    fp = np.sum(np.logical_and(predict_issame,
np.logical_not(actual_issame)))
    tn = np.sum(np.logical_and(np.logical_not(predict_issame),
np.logical_not(actual_issame)))
    fn = np.sum(np.logical_and(np.logical_not(predict_issame),
actual_issame))
```

```
    tpr = 0 if (tp + fn == 0) else float(tp) / float(tp + fn)
    fpr = 0 if (fp + tn == 0) else float(fp) / float(fp + tn)
    acc = float(tp + tn) / dist.size
    return tpr, fpr, acc


def calculate_val(thresholds, embeddings1, embeddings2, actual_issame,
far_target, nrof_folds=10, distance_metric=0,
                  subtract_mean=False):
    assert (embeddings1.shape[0] == embeddings2.shape[0])
    assert (embeddings1.shape[1] == embeddings2.shape[1])
    nrof_pairs = min(len(actual_issame), embeddings1.shape[0])
    nrof_thresholds = len(thresholds)
    k_fold = KFold(n_splits=nrof_folds, shuffle=False)

    val = np.zeros(nrof_folds)
    far = np.zeros(nrof_folds)

    indices = np.arange(nrof_pairs)

    for fold_idx, (train_set, test_set) in
enumerate(k_fold.split(indices)):
        if subtract_mean:
            mean = np.mean(np.concatenate([embeddings1[train_set],
embeddings2[train_set]]), axis=0)
        else:
            mean = 0.0
        dist = distance(embeddings1 - mean, embeddings2 - mean,
distance_metric)

        # Find the threshold that gives FAR = far_target
        far_train = np.zeros(nrof_thresholds)
        for threshold_idx, threshold in enumerate(thresholds):
            _, far_train[threshold_idx] = calculate_val_far(threshold,
dist[train_set], actual_issame[train_set])
        if np.max(far_train) >= far_target:
            f = interpolate.interp1d(far_train, thresholds,
kind='slinear')
```

```
            threshold = f(far_target)
        else:
            threshold = 0.0

        val[fold_idx], far[fold_idx] = calculate_val_far(threshold,
dist[test_set], actual_issame[test_set])

    val_mean = np.mean(val)
    far_mean = np.mean(far)
    val_std = np.std(val)
    return val_mean, val_std, far_mean

def calculate_val_far(threshold, dist, actual_issame):
    predict_issame = np.less(dist, threshold)
    true_accept = np.sum(np.logical_and(predict_issame, actual_issame))
    false_accept = np.sum(np.logical_and(predict_issame,
np.logical_not(actual_issame)))
    n_same = np.sum(actual_issame)
    n_diff = np.sum(np.logical_not(actual_issame))
    val = float(true_accept) / float(n_same)
    far = float(false_accept) / float(n_diff)
    return val, far



def list_variables(filename):
    reader = training.NewCheckpointReader(filename)
    variable_map = reader.get_variable_to_shape_map()
    names = sorted(variable_map.keys())
    return names

def write_arguments_to_file(args, filename):
    with open(filename, 'w') as f:
        for key, value in iteritems(vars(args)):
            f.write('%s: %s\n' % (key, str(value)))
```

Based on the distance value that the system has calculated, I have implemented functions to help calculate the percentage similarity of the input image compared to the face images that have been trained through the segments above code.

align_mtcnn.py :

```
def align_mtcnn(input_dir,
                  output_dir,
                  image_size=182,
                  margin=44,
                  random_order=None,
                  gpu_memory_fraction=1.0,
                  detect_multiple_faces=False):

    sleep(random.random())
    output_dir = os.path.expanduser(output_dir)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    # Store some git revision info in a text file in the log directory
    src_path, _ = os.path.split(os.path.realpath(__file__))
    store_revision_info(src_path, output_dir, ' '.join(sys.argv))
    dataset = get_dataset(input_dir)

    print('Creating networks and loading parameters')

    with tf.Graph().as_default():
        gpu_options =
tf.GPUOptions(per_process_gpu_memory_fraction=gpu_memory_fraction)
        sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options,
log_device_placement=False))
        with sess.as_default():
            pnet, rnet, onet = create_mtcnn(sess, None)

    minsize = 20  # minimum size of face
    threshold = [0.6, 0.8, 0.8]  # three steps's threshold
    factor = 0.709  # scale factor

    # Add a random key to the filename to allow alignment using multiple
processes
    random_key = np.random.randint(0, high=99999)
    bounding_boxes_filename = os.path.join(output_dir,
'bounding_boxes_%05d.txt' % random_key)

    with open(bounding_boxes_filename, "w") as text_file:
        nrof_images_total = 0
        nrof_successfully_aligned = 0
        if random_order:
            random.shuffle(dataset)
        for cls in dataset:
            output_class_dir = os.path.join(output_dir, cls.name)
            if not os.path.exists(output_class_dir):
                os.makedirs(output_class_dir)
                if random_order:
                    random.shuffle(cls.image_paths)
            for image_path in cls.image_paths:
                nrof_images_total += 1
                filename =
os.path.splitext(os.path.split(image_path)[1])[0]
                output_filename = os.path.join(output_class_dir, filename
+ '.png')
                print(image_path)
                if not os.path.exists(output_filename):
                    try:
                        img = misc.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 = to_rgb(img)
                        img = img[:, :, 0:3]

                        bounding_boxes, _ = detect_face(img, minsize,
pnet, rnet, onet, threshold, factor)
                        nrof_faces = bounding_boxes.shape[0]
                        if nrof_faces > 0:
                            det = bounding_boxes[:, 0:4]
                            det_arr = []
                            img_size = np.asarray(img.shape)[0:2]
                            if nrof_faces > 1:
                                if 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))

                            for i, det in enumerate(det_arr):
                                det = np.squeeze(det)
                                bb = np.zeros(4, dtype=np.int32)
                                bb[0] = np.maximum(det[0] - margin / 2, 0)
                                bb[1] = np.maximum(det[1] - margin / 2, 0)
                                bb[2] = np.minimum(det[2] + margin / 2,
img_size[1])
                                bb[3] = np.minimum(det[3] + margin / 2,
img_size[0])
                                cropped = img[bb[1]:bb[3], bb[0]:bb[2], :]
                                scaled = misc.imresize(cropped,
(image_size, image_size), interp='bilinear')
                                nrof_successfully_aligned += 1
                                filename_base, file_extension =
os.path.splitext(output_filename)
                                if detect_multiple_faces:
                                    output_filename_n =
"{}_{}{}".format(filename_base, i, file_extension)
                                else:
                                    output_filename_n =
"{}{}".format(filename_base, file_extension)
                                misc.imsave(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))
```

This is the function that executes the MTCNN algorithm that has been initialized in the detect_face.py file. When executing this function, the system will proceed to use the MTCNN algorithm and the detect_face function I declared above to detect the face coordinates in the input image. I have set thresholds for face detection to be 0.6, 0.8, 0.8 respectively. With such a threshold ratio, the system will certainly not be able to miss any faces in the image.
Next, the system will proceed to cut the faces in the input and output images for easy data training, the coordinates of the detected faces will be saved in a .txt file in the face_align.. folder.

- **face_contrib.py :**

```
class Face:
    def __init__(self):
        self.name = None
        self.bounding_box = None
        self.image = None
        self.container_image = None
        self.embedding = None
```

First, I create a class Face to hold variables that store face information

```
class Recognition:
    def __init__(self, facenet_model_checkpoint, classifier_model):
        self.detect = Detection()
        self.encoder = Encoder(facenet_model_checkpoint)
        self.identifier = Identifier(classifier_model)
    def add_identity(self, image, person_name):
        faces = self.detect.find_faces(image)

        if len(faces) == 1:
            face = faces[0]
            face.name = person_name
            face.embedding = self.encoder.generate_embedding(face)
            return faces

    def identify(self, image):
        faces = self.detect.find_faces(image)

        for i, face in enumerate(faces):
            if debug:
                cv2.imshow("Face: " + str(i), face.image)
            face.embedding = self.encoder.generate_embedding(face)
            face.name, face.prob = self.identifier.identify(face)

        return faces

class Identifier:
    def __init__(self, classifier_model):
        with open(classifier_model, 'rb') as infile:
            self.model, self.class_names = pickle.load(infile)

    def identify(self, face):
        if face.embedding is not None:
            predictions = self.model.predict_proba([face.embedding])
            best_class_indices = np.argmax(predictions, axis=1)
            return self.class_names[best_class_indices[0]],
predictions[0][best_class_indices[0]]
```

Next, I declare the Recognition and Identifier classes to assign identifiers to the recognized faces, if that face is the face of an existing face in the model, that face's identifier is the corresponding label. response.

```
class Encoder:
    def __init__(self, facenet_model_checkpoint):
        self.sess = tf.Session()
        with self.sess.as_default():
            load_model(facenet_model_checkpoint)

    def generate_embedding(self, face):
        # Get input and output tensors
        images_placeholder =
tf.get_default_graph().get_tensor_by_name("input:0")
        embeddings =
tf.get_default_graph().get_tensor_by_name("embeddings:0")
        phase_train_placeholder =
tf.get_default_graph().get_tensor_by_name("phase_train:0")

        prewhiten_face = prewhiten(face.image)

        # Run forward pass to calculate embeddings
        feed_dict = {images_placeholder: [prewhiten_face],
phase_train_placeholder: False}
        return self.sess.run(embeddings, feed_dict=feed_dict)[0]
```

Above is the code to convert the images directly from the camera into embedding vectors so that we can calculate the distance between these vectors and the vectors already in the model.

```
class Detection:
    # face detection parameters
    minsize = 20  # minimum size of face
    threshold = [0.65, 0.7, 0.7]  # three steps's threshold
    factor = 0.709  # scale factor

    def __init__(self, face_crop_size=160, face_crop_margin=32):
        self.pnet, self.rnet, self.onet = self._setup_mtcnn()
        self.face_crop_size = face_crop_size
        self.face_crop_margin = face_crop_margin
    def _setup_mtcnn(self):
        with tf.Graph().as_default():
            gpu_options =
tf.GPUOptions(per_process_gpu_memory_fraction=gpu_memory_fraction)
            sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options,
log_device_placement=False))
            with sess.as_default():
                return create_mtcnn(sess, None)

    def find_faces(self, image):
        faces = []

        bounding_boxes, _ = detect_face(image, self.minsize,
                                                          self.pnet,
self.rnet, self.onet,
                                                          self.threshold,
self.factor)
        for bb in bounding_boxes:
            face = Face()
            face.container_image = image
            face.bounding_box = np.zeros(4, dtype=np.int32)

            img_size = np.asarray(image.shape)[0:2]
            face.bounding_box[0] = np.maximum(bb[0] - self.face_crop_margin /
2, 0)
            face.bounding_box[1] = np.maximum(bb[1] - self.face_crop_margin /
2, 0)
            face.bounding_box[2] = np.minimum(bb[2] + self.face_crop_margin /
2, img_size[1])
            face.bounding_box[3] = np.minimum(bb[3] + self.face_crop_margin /
2, img_size[0])
            cropped = image[face.bounding_box[1]:face.bounding_box[3],
face.bounding_box[0]:face.bounding_box[2], :]
            face.image = misc.imresize(cropped, (self.face_crop_size,
self.face_crop_size), interp='bilinear')

            faces.append(face)
        return faces
```

Finally, here is the code to help execute the calculation function applying the triplet loss function declared in the facenet.py file. After the calculation is complete, the distance between the two vectors will be returned and compared with the set thresholds of 0.6,0.7, and 0.7. If this threshold is exceeded, the label value assigned to that vector will be returned.

- **train.py :**

```
def train(data_dir,
        model,
        classifier_filename,
        use_split_dataset=None,
        batch_size=1000,
        image_size=160,
        seed=123,
        min_nrof_images_per_class=20,
        nrof_train_images_per_class=10):

    with tf.Graph().as_default():
        with tf.Session() as sess:
            np.random.seed(seed=seed)
            if use_split_dataset:
                dataset_tmp = get_dataset(data_dir)
                train_set, test_set = split_dataset(dataset_tmp,
min_nrof_images_per_class,

nrof_train_images_per_class)
                dataset = train_set
            else:
                dataset = get_dataset(data_dir)

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

            paths, labels = get_image_paths_and_labels(dataset)


            messagebox.showinfo("Trainning Process",f'Number of people: %d'%
len(dataset))
            messagebox.showinfo("Trainning Process", f'Number of images: %d'
% len(paths))

            # Load the model
            load_model(model)

            # Get input and output tensors
            images_placeholder =
tf.get_default_graph().get_tensor_by_name("input:0")
            embeddings =
tf.get_default_graph().get_tensor_by_name("embeddings:0")
            phase_train_placeholder =
tf.get_default_graph().get_tensor_by_name("phase_train:0")
            embedding_size = embeddings.get_shape()[1]

            # Run forward pass to calculate embeddings
            messagebox.showinfo('Training Process','Calculating features for
images')
            nrof_images = len(paths)
            nrof_batches_per_epoch = int(math.ceil(1.0 * nrof_images /
batch_size))
            emb_array = np.zeros((nrof_images, embedding_size))
            for i in range(nrof_batches_per_epoch):
                    start_index = i * batch_size
                    end_index = min((i + 1) * batch_size, nrof_images)
                    paths_batch = paths[start_index:end_index]
                    images = load_data(paths_batch, False, False, image_size)
                    feed_dict = {images_placeholder: images,
phase_train_placeholder: False}
                    emb_array[start_index:end_index, :] =
sess.run(embeddings, feed_dict=feed_dict)

            classifier_filename_exp = os.path.expanduser(classifier_filename)

            # =================== Train classifier =====================
            messagebox.showinfo('Trainning Conduct','Training classifier')
            model = SVC(kernel='linear', probability=True)
            model.fit(emb_array, labels)

            # Create a list of class names
            class_names = [cls.name.replace('_', ' ') for cls in dataset]

            # Saving classifier model
            with open(classifier_filename_exp, 'wb') as outfile:
                pickle.dump((model, class_names), outfile)
            messagebox.showinfo('Trainning Finish','Data has been trained
successfully !!!')

if __name__ == '__main__':
    align_mtcnn('dataSet', 'face_align')
    train('face_align/', 'models/20180402-114759.pb',
'models/your_model.pkl')
```

The file train.py is the file that executes the data train functions that have been initialized in the system. When conducting data training, the system will proceed to take the face images detected by MTCNN, process them and then pass them to the Facenet algorithm for processing.

Set condition is that there must be more than two faces for the system to start training. When conducting the training process, the system will display a message about how many people and how many pictures. Then the system loads the model to use for storage. After the

training is complete, the system will display a success message and save the data containing the vectors that have been labeled with the face into the model.

- **detection.py :**

```
def attendance(i, n, c, f):
    with open('attendance.csv', 'r+', newline='\n') as file:
        DataList = file.readlines()
        namelist = []
        for data in DataList:
            ent = data.split(',')
            namelist.append(ent[0])
        if ((i not in namelist) and (n not in namelist) and (c not in
namelist) and (f not in namelist)):
            curr = datetime.now()
            dt = curr.strftime("%d/%m/%Y")
            h = curr.strftime('%H:%M:%S')
            file.writelines(f'\n{i},{n},{c},{f},{dt},{h}')
```

In the file detection.py , I initialized a attendance function to remember the information of the recognized face. The recognized faces will be automatically stored information and recognized date and time in the file attendance.csv.

```
def add_overlays(frame, faces, frame_rate, colors, confidence=0.8):
    if faces is not None:
        for idx, face in enumerate(faces):
            face_bb = face.bounding_box.astype(int)
            cv2.rectangle(frame, (face_bb[0], face_bb[1]), (face_bb[2],
face_bb[3]), colors[idx], 2)

            if face.name and face.prob:
                if face.prob > confidence:
                    class_name = face.name
                    s = str(class_name)
                    r = re.findall(r'\d', s)
                    id =''.join(r)

                    conn = mysql.connector.connect(host='localhost',
user="root", password='Vilong242',
                                                   db='face_recognition')
                    cur = conn.cursor()

                    cur.execute("select Student_ID from student where
Student_ID=" + str(id))
                    i = cur.fetchone()
                    i = '+'.join(i)

                    cur.execute("select Name from student where Student_ID="
+ str(id))
                    n = cur.fetchone()
                    n = '+'.join(n)

                    cur.execute("select Class from student where Student_ID="
+ str(id))
                    c = cur.fetchone()
                    c = '+'.join(c)

                    cur.execute("select Faculty from student where
Student_ID=" + str(id))
                    f = cur.fetchone()
                    f = '+'.join(f)



                    cv2.putText(frame, f'Student ID:{i}', (face_bb[0],
face_bb[3] + 20), cv2.FONT_HERSHEY_COMPLEX, 0.6, colors[idx], thickness=2,
lineType=1)
                    cv2.putText(frame, f'Name:{n}', (face_bb[0], face_bb[3] +
45), cv2.FONT_HERSHEY_COMPLEX, 0.6, colors[idx], thickness=2, lineType=1)
                    cv2.putText(frame, f'Class:{c}', (face_bb[0], face_bb[3]
+ 70), cv2.FONT_HERSHEY_COMPLEX, 0.6, colors[idx], thickness=2, lineType=1)
                    cv2.putText(frame, f'Faculty:{f}', (face_bb[0],
face_bb[3] + 95), cv2.FONT_HERSHEY_COMPLEX, 0.6, colors[idx], thickness=2,
lineType=1)
                    cv2.putText(frame, '{:.02f}'.format(face.prob * 100),
(face_bb[0], face_bb[3] + 120), cv2.FONT_HERSHEY_SIMPLEX, 0.6, colors[idx],
thickness=1, lineType=2)
```

```
                    attendance(i,n,c,f)
                else :
                    class_name = 'Unknown People'
                    cv2.putText(frame, class_name, (face_bb[0], face_bb[3] +
20), cv2.FONT_HERSHEY_SIMPLEX, 0.6,colors[idx], thickness=2, lineType=2)
        cv2.putText(frame, str(frame_rate) + " fps", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), thickness=2, lineType=2)
```

The above code executes the display of the basic information of the face stored in the database when the system recognizes it. The information stored in the database will be retrieved and displayed on the identification screen. If the face is not recognized, the system will return the result Unknown People.

```
def run(model_checkpoint, classifier, output_file=None):
    frame_interval = 3  # Number of frames after which to run face detection
    fps_display_interval = 5  # seconds
    frame_rate = 0
    frame_count = 0

    camera = cv2.VideoCapture(0)
    ret, frame = camera.read()
    width = frame.shape[1]
    height = frame.shape[0]
    if output_file is not None:
        video_format = cv2.VideoWriter_fourcc(*'XVID')
        out = cv2.VideoWriter(output_file, video_format, 20, (width, height))

    face_recognition = Recognition(model_checkpoint, classifier)
    start_time = time.time()
    colors = np.random.uniform(0, 255, size=(1, 3))
    while True:
        # Capture frame-by-frame
        ret, frame = camera.read()

        if (frame_count % frame_interval) == 0:
            faces = face_recognition.identify(frame)
            for i in range(len(colors), len(faces)):
                colors = np.append(colors, np.random.uniform(150, 255,
size=(1, 3)), axis=0)
            # Check our current fps
            end_time = time.time()
            if (end_time - start_time) > fps_display_interval:
                frame_rate = int(frame_count / (end_time - start_time))
                start_time = time.time()
                frame_count = 0

        add_overlays(frame, faces, frame_rate, colors)

        frame_count += 1
        cv2.imshow('Welcome to school', frame)
        if output_file is not None:
            out.write(frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # When everything is done, release the capture
    if output_file is not None:
        out.release()
    camera.release()
    cv2.destroyAllWindows()


if __name__ == '__main__':
    run('models', 'models/your_model.pkl')
```

The above code executes the loading of data in the model to launch the declared calculation functions. The end purpose is to return the result as the value of that face.