KING FAHD UNIVERSITY OF
PETROLEUM & MINERALS


# Mask detection program

## CISE-483
## AI & ML for Robotics

Name: Ali Hassan Alnemer

ID: 201627320

# Contents

# 1-Introduction

Since the beginning of the COVID-19 pandemic, monitoring the wearing of masks in crowded places has become very important, as it is directly associated with limiting the spreading of the virus. Monitoring the wearing of masks in the spread of the pandemic is very difficult because it requires a large number of employees. Also, direct interaction of the employees with people could be dangerous to the health of the employees. In this project, I designed a gate opens automatically for people who wear a mask only using a program that can automatically classify faces into two classes, either wearing a mask or not wearing a mask using machine learning techniques.

# 2-Project description

In this project, I designed a python program that uses a convolutional neural network to identify whether a person is wearing a mask or not using a camera. Then I will connect the program with the Arduino, which will represent the control unit in a gate, and if the person wears a mask, the gate will open, and if he does not wear a mask, it will be closed. The steps I took to complete the program are as follows.

1- Data set collecting
2- Data filtering
3- Image processing and labelling
4- Model training
5- Life cam and prediction
6- Arduino and hardware connection

# 3- Designing processes

## a. Data set collecting

For the data, I collected 60,000 images for each class, 120,000 in total. I get 60,000 images of faces with a mask from Maskedface-net project, it is a project similar to mine, but it focuses more on wearing the mask correctly or incorrectly (Cabani, 2020). Also, I get 60,000 images of faces without a mask from Celeba project which is an open-source dataset of faces images (Celeba, 2016).

## b. Data filtering

The data I collected is not ideal for training because the faces in the pictures are located at different distances and different angles, which makes the training process inaccurate. To filter the data, I used the (haarcascade_frontalface_default classifier) tool to recognize faces from the photos and crop the faces only from photos as figure 1 shows. I modified and used Dhavalsays conde as appendix A shows, which will take the data and choose the clear images only and cut the faces from them and produce a new file containing the new data (Dhavalsays, 2020). This process reduced the data from 120000 images to 60000 images.

*Figure 1: the cropped and original images*

## c. Image processing and labelling

For image processing, I modified and resized the filtered images to be suitable for training. First, I changed the colors of the images to grey color using OpenCV to reduce the diminution of the images from 3 to 1 as Figure 3 shows. Then, I changed the size of the images to 100X100 as Figure 4 shows. For data labelling, I extracted all images into an array using a loop and I corresponded each image to its class using (append) function as appendix B shows (Kinsley, 2018). Then, after I shuffled the data, I separated the futures and labels each in a separate array as appendix B shows. then I saved the data using Pickle tool.

*Figure 2: the original image*

```
DATADIR = "data/train/with_mask/00003_Mask.jpg"
img = cv2.imread(DATADIR)
img_array = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)
plt.imshow(img_array , cmap='gray')
```

```
]: <matplotlib.image.AxesImage at 0x22dedb879d0>
```

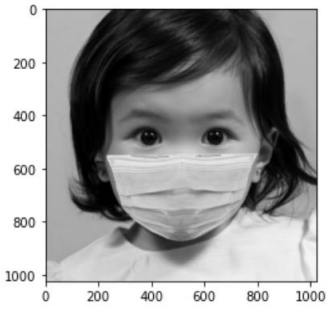


*Figure 3: the grey image*

```
resize_img = cv2.resize(img_array, (100, 100))
plt.imshow(resize_img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x22def4a4c70>
```

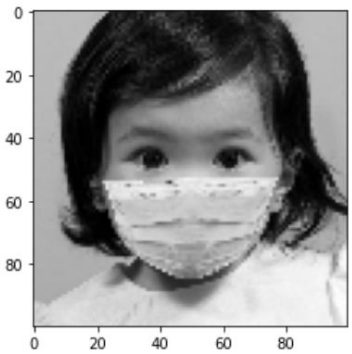


*Figure 4: the resized image*

## d. Model training

After preparing the data for training, I used (train_test_split) function to split training data and testing data, 80% for the training data and 20% for the testing data. I chose Keras sequential model to train using my data. I used 2 hidden layers with 124 nodes and a (tanh) activation function for each, and one output node layer with a (sigmoid) activation function. For optimization, I used 0.0001 learning rate and 100 epoch as appendix C shows. After training the model I got 99.995% accuracy from testing data persecution.

## e. Life cam and prediction

For the face recognition I used (haarcascade_frontalface_default) classifier model from the OpenCV library to detect faces from the cam, this classifier is widely used in applications that require face detection such as my project (OpenCV, 2001). Then, I implemented my face mask model in this classifier to take each frame that contains a face and crop it and resize it and modify it as I did in the training data. then I passed the resized and modified image to the model as shown in appendix D. If the model gives me a prediction lower than 0.999 it will be marked as wearing a mask otherwise it is not wearing a mask as figure 5.

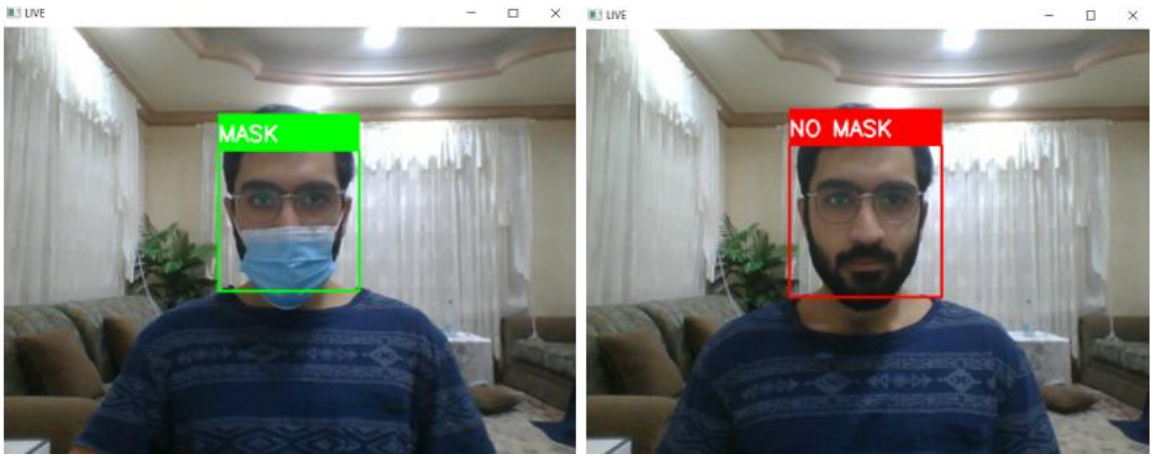


*Figure 5: the final result*

## f. Arduino and hardware designing

I used the Arduino UNO as a controller that controls a gait open and closes using a servo motor as shown in figure 6. In the Arduino, I used "Firmata" stander code which is a library on the Arduino that allows the Arduino to communicate with python software (Arduino, 2019). Then I wrote the function in appendix E which will open the gait if the prediction is "wearing a mask" and close if the prediction is "not wearing a mask".

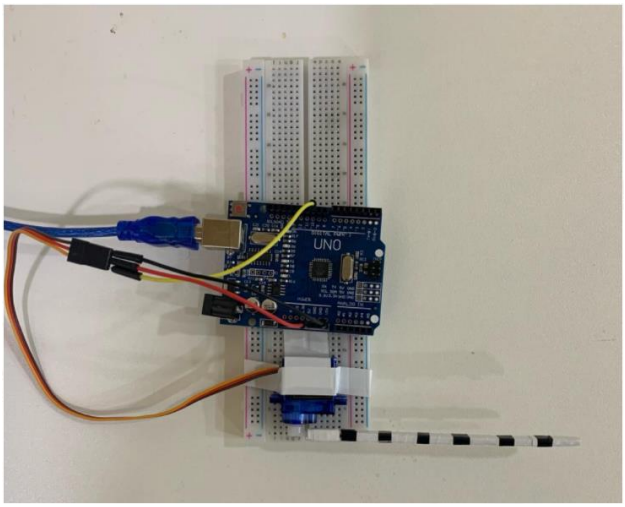


*Figure 6: the hardware setup*

# 4- Project result

After the program and the hardware connection are completed, I start testing the project on myself and other people. The program gives very good results and the hardware part works as expected. However, in some difficult angles, the program sometimes gives wrong predictions but in general, it gives correct results.

# 5-References


1- Cabani. (2020, April 28). *Cabani/Maskedface-Net: Maskedface-net is a dataset of human faces with a correctly and incorrectly worn mask based on the dataset Flickr-faces-HQ (FFHQ)*. GitHub. Retrieved October 28, 2021, from https://github.com/cabani/MaskedFace-Net

2- *Celeba. (2016) Large-scale celebfaces attributes (celeba) dataset*. CelebA Dataset. (2016). Retrieved October 28, 2021, from http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

3- Dhavalsays, C. (2020). *py/data_cleaning.ipynb at master · codebasics/py*. GitHub. https://github.com/codebasics/py/blob/master/DataScience/CelebrityFaceRecognition/model/data_cleaning.ipynb

4- Kinsley, H. (2018, August). *Python Programming Tutorials*. Pythonprogramming. https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/

5- OpenCV. (2001). *OpenCV: Cascade Classifier*. Open Source Computer Vision. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

6- Arduino. (2019, December). *Arduino - Firmata*. Firmata Library. https://www.arduino.cc/en/reference/firmata

# 6-Appendix

## 1- Appendix A

### Data set cleaning

1) Cropping the faces from the data

2) Regenerate the data with faces only

In [ ]:
```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

In [ ]:
```
def get_cropped_image_if_2_eyes(image_path):
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        if len(eyes) >= 2:
            return roi_color
```

In [ ]:
```
path_to_data = "./data/"
path_to_cr_data = "./data/cropped/"
```

In [ ]:
```
import os
img_dirs = []
for entry in os.scandir(path_to_data):
    if entry.is_dir():
        img_dirs.append(entry.path)
```

In [ ]:
```
import shutil
if os.path.exists(path_to_cr_data):
     shutil.rmtree(path_to_cr_data)
os.mkdir(path_to_cr_data)
```

In [ ]:
```
cropped_image_dirs = []
file_names_dict = {}

for img_dir in img_dirs:
    count = 1
    name = img_dir.split('/')[-1]
    print(name)

    file_names_dict[name] = []

    for entry in os.scandir(img_dir):
        roi_color = get_cropped_image_if_2_eyes(entry.path)
        if roi_color is not None:
            cropped_folder = path_to_cr_data + name
            if not os.path.exists(cropped_folder):
                os.makedirs(cropped_folder)
                cropped_image_dirs.append(cropped_folder)
                print("Generating cropped images in folder: ",cropped_folder)

            cropped_file_name = name + str(count) + ".png"
            cropped_file_path = cropped_folder + "/" + cropped_file_name

            cv2.imwrite(cropped_file_path, roi_color)
            file_names_dict[name].append(cropped_file_path)
            count += 1
```

## 2- Appendix B

### Image processing and labelling

1) Load the data

2) Save the data

3) Image processing

4) Label the data

5) Split train and test data

In [76]:
```
IMG_SIZE=100
```

In [3]:
```


DATADIR = "data"

CATEGORIES = ["with_mask", "without_mask"]
data = []
for category in CATEGORIES:

        path = os.path.join(DATADIR,category)
        class_num = CATEGORIES.index(category)

        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                data.append([new_array, class_num])
            except Exception as e:
                pass


```

```
100%|██████████| 29322/29322 [56:47<00:00,  8.60it/s]
100%|██████████| 33814/33814 [06:42<00:00, 84.03it/s]
```

In [4]:
```
filename = 'data1'
outfile = open(filename,'wb')
pickle.dump(data,outfile)
outfile.close()

```

In [77]:
```
infile = open('data1','rb')
data = pickle.load(infile)
infile.close()

```

In [78]:
```
random.shuffle(data)
```

In [79]:
```
X = []
Y = []

for features,label in data:
    X.append(features)
    Y.append(label)


X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
Y = np.array(Y)
X = X / 255.0

print(X.shape)
print(Y.shape)

```

```
(63136, 100, 100, 1)
(63136,)
```

In [80]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size = 0.20)
```

# 3- Appendix C

## Model training and testing

1) Trane the convolutional neural network

2) Test the convolutional neural network

In [82]:
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
from keras import optimizers
```

In [92]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(IMG_SIZE, IMG_SIZE)),
    tf.keras.layers.Dense(124, activation='tanh'),
    tf.keras.layers.Dense(124, activation='tanh'),
    tf.keras.layers.Dense(1 ,activation='sigmoid')
])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_5 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_15 (Dense)             (None, 124)               1240124
_________________________________________________________________
dense_16 (Dense)             (None, 124)               15500
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 125
=================================================================
Total params: 1,255,749
Trainable params: 1,255,749
Non-trainable params: 0
_________________________________________________________________
```

In [93]:
```
model.compile(optimizer=optimizers.RMSprop(lr=0.00001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [94]:
```
model.fit(xtrain, ytrain, epochs=100)
```

In [95]:
```
test_loss, test_acc = model.evaluate(xtest,  ytest, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
395/395 - 2s - loss: 0.0234 - accuracy: 0.9946

Test accuracy: 0.9946151375770569
```

In [96]:
```
model.save("mask model 9", save_format="h5")
```

# 4- Appendix D

## Life cam and prediction

1) Recognising face using life camera

2) Predict if the face wearing a mask or not

3) Show the result

In [53]:
```
from keras.models import load_model
import cv2
import numpy as np
```

In [115]:
```
model = load_model('mask model 9')

```

In [120]:
```


face_clsfr = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

source = cv2.VideoCapture(0)

labels_dict={0:'MASK',1:'NO MASK'}
color_dict={0:(0,255,0),1:(0,0,255)}

while(True):

    ret,img=source.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=face_clsfr.detectMultiScale(gray,1.3,3)

    for (x,y,w,h) in faces:

        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(IMG_SIZE,IMG_SIZE))
        normalized=resized/255
        reshaped=np.reshape(normalized,(-1,IMG_SIZE,IMG_SIZE,1))
        ar = np.expand_dims(reshaped, axis = 0)
        result=model.predict(ar)




        if  result <=0.999:
            label=0

        else:
            label=1

        cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
        cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
        cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)


    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)

    if(key==27):
        break

cv2.destroyAllWindows()
source.release()
```

# 5- Appendix E

In [23]:

```
from pyfirmata import Arduino, SERVO,util
```

In [34]:

```
pin=10
board=Arduino('COM3')

board.digital[pin].mode=SERVO



def gatard(x):
    if x==0:
        board.digital[10].write(90)

    elif x==1:
        board.digital[10].write(0)
```