Eye state drowsiness Detection

We can use this code for predecting if the eye if closed or open and also it can be used for mainly Driver Drowsiness Detection.

Import Libraries

In [3]:
```
#pip install opencv-python
```

```
Collecting opencv-python
  Downloading opencv_python-4.5.1.48-cp38-cp38-win_amd64.whl (34.9 MB)
Requirement already satisfied: numpy>=1.17.3 in c:\users\jack1\anaconda3\lib\si
te-packages (from opencv-python) (1.19.5)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.5.1.48
Note: you may need to restart the kernel to use updated packages.
```

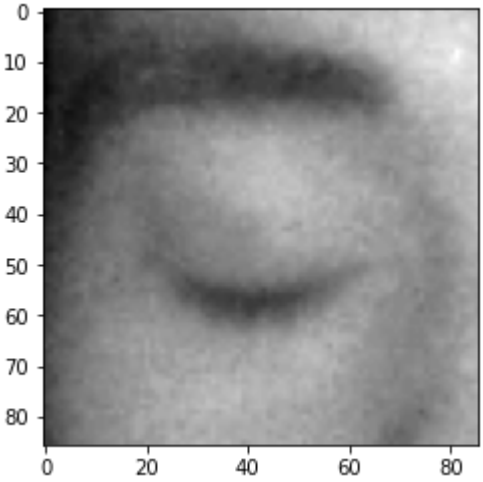
In [11]:
```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Read the images

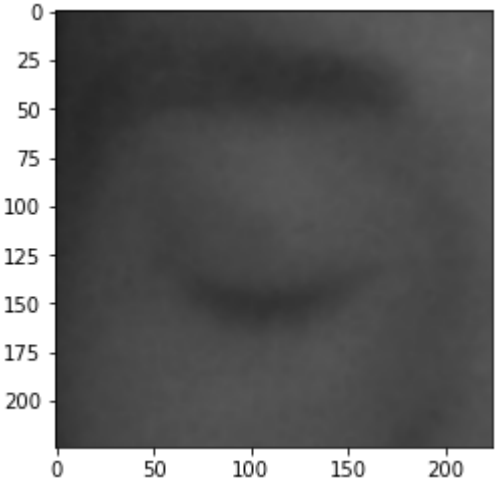
In [12]:
```
img_array = cv2.imread('C:/Users/jack1/Dropbox/My PC (LAPTOP-C56257JD)/Desktop/tr
```

In [13]:
```
import os
```

In [14]:
```
Datadirectory = 'C:/Users/jack1/Dropbox/My PC (LAPTOP-C56257JD)/Desktop/train'
Classes = ['Closed_Eyes', 'Open_Eyes']
# Datadirectory = 'dataset_new/train/'
# Classes = ['Closed', 'Open']
for category in Classes:
  path = os.path.join(Datadirectory, category)
  for img in os.listdir(path):
    img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
    backtorgb = cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
    plt.imshow(img_array, cmap="gray")
    plt.show()
    break
  break
```



In [15]:
```
img_size = 224
new_array = cv2.resize(backtorgb, (img_size,img_size))
plt.imshow(new_array, cmap="gray")
plt.show()
```



In [27]:
```
training_data = []

def create_training_data():
  for category in Classes:
      path = os.path.join(Datadirectory, category)
      class_num = Classes.index(category)
      for img in os.listdir(path):
        try :
           img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
           backtorgb = cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
           new_array = cv2.resize(backtorgb, (img_size,img_size))
           training_data.append([new_array, class_num])
        except Exception as e:
          pass
```

In [28]:
```
create_training_data()
```

In [29]:
```
print(len(training_data))
```

```
4000
```

In [30]:
```
import random
random.shuffle(training_data)
```

In [31]:
```
#here we reshape the image.
X = []
y = []
for features, label in training_data:
  X.append(features)
  y.append(label)

X = np.array(X).reshape(-1, img_size, img_size, 3)
```

In [32]:
```
X.shape
```

Out[32]:
```
(4000, 224, 224, 3)
```

In [33]:
```
X = X/255.0
```

In [34]:
```
Y = np.array(y)
```

In [35]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Model Building

Since i am using transfer learning to get appropriate result, i download the pre-trained mobilenet model.

In [36]:
```
model = tf.keras.applications.mobilenet.MobileNet()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5 (https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5)
17227776/17225924 [==============================] - 10s 1us/step
```

In [37]:
```
model.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
_________________________________________________________________
conv1_relu (ReLU)            (None, 112, 112, 32)      0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)      288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32)      128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, 112, 112, 32)      0
_________________________________________________________________
conv_pw_1 (Conv2D)           (None, 112, 112, 64)      2048
```

In [38]:
```
base_input = model.layers[0].input
```

In [39]:
```
base_output = model.layers[-4].output
```

In [40]:
```
Flat_layer = layers.Flatten()(base_output)
final_output = layers.Dense(1)(Flat_layer)
final_output = layers.Activation('sigmoid')(final_output)
```

In [41]:
```
new_model = keras.Model(inputs = base_input, outputs = final_output)
```

In [42]:
```
new_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
_________________________________________________________________
conv1_relu (ReLU)            (None, 112, 112, 32)      0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)      288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32)      128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, 112, 112, 32)      0
_________________________________________________________________
conv_pw_1 (Conv2D)           (None, 112, 112, 64)      2048
```

In [43]:
```
new_model.compile(loss="binary_crossentropy", optimizer = "adam", metrics = ["acc
```

In [44]:
```
new_model.fit(X,Y, epochs = 2, validation_split = 0.1)
#Note: Increase the number of epoch to get more appropriate result, accuracy.
```

```
Epoch 1/2
113/113 [==============================] - 589s 5s/step - loss: 0.0782 - accura
cy: 0.9599 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 2/2
113/113 [==============================] - 544s 5s/step - loss: 0.0153 - accura
cy: 0.9956 - val_loss: 5.6848e-04 - val_accuracy: 1.0000
```

Out[44]:
```
<tensorflow.python.keras.callbacks.History at 0x207061c7c40>
```

In [45]:
```
new_model.save('my_model.h5')
```

In [46]:
```
new_model = tf.keras.models.load_model('my_model.h5')
```

Test whether the eye is closed or open

As the value ranges from 0 to 1, we the eye is detected as closed, then the value will be nearer to zero. And if the eye is detected as open then it will be near to 1.

In [49]:
```
img_array = cv2.imread('C:/Users/jack1/Dropbox/My PC (LAPTOP-C56257JD)/Desktop/tr
#backtorgb = cv2.cvtColor(img_array, cv2.COLOR_GRAY2BGR)
new_array = cv2.resize(backtorgb, (img_size, img_size))
```

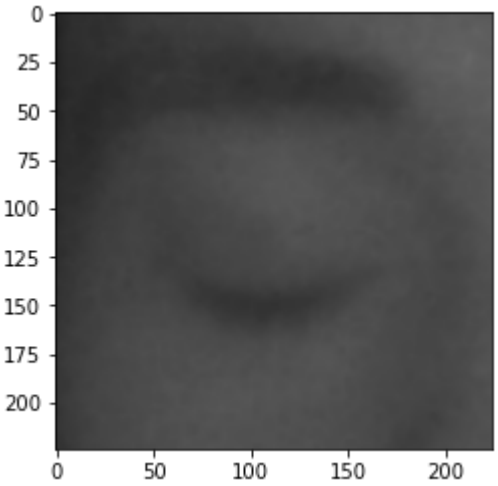
In [50]:
```
X_input = np.array(new_array).reshape(1, img_size, img_size, 3)
```

In [51]:
```
X_input.shape
```

Out[51]:
```
(1, 224, 224, 3)
```

In [52]:
```
plt.imshow(new_array)
```

Out[52]:
```
<matplotlib.image.AxesImage at 0x2070687da00>
```



In [53]:
```
X_input = X_input/255.0
```

In [54]:
```
prediction = new_model.predict(X_input)
```

In [55]:
```
prediction
```

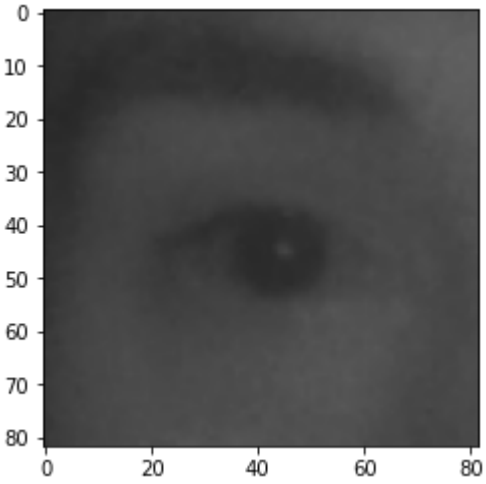
Out[55]:
```
array([[9.618662e-17]], dtype=float32)
```

Since the eye is closed, we got the prediction value nearer to 0, i.e. 0.0000000000009618.

In [130]:
```
img = cv2.imread('C:/Users/jack1/Dropbox/My PC (LAPTOP-C56257JD)/Desktop/train/Op
```

In [131]:
```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

Out[131]: <matplotlib.image.AxesImage at 0x20720a2d400>



In [132]:
```
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalf
eyeCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
```

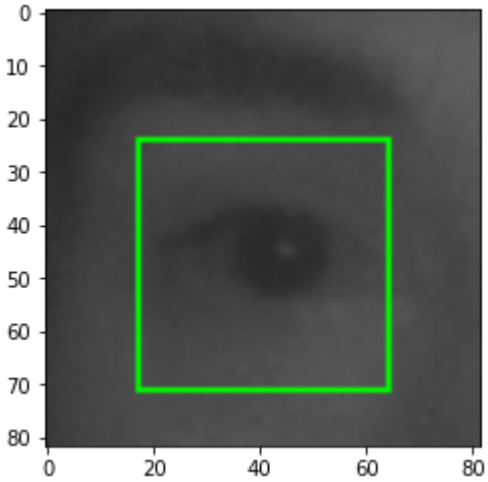
In [133]:
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

In [134]:
```
eyes = eyeCascade.detectMultiScale(gray, 1.1, 4)
```

In [135]:
```
for (x, y, w, h) in eyes:
  cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 1)
```

In [136]:
```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

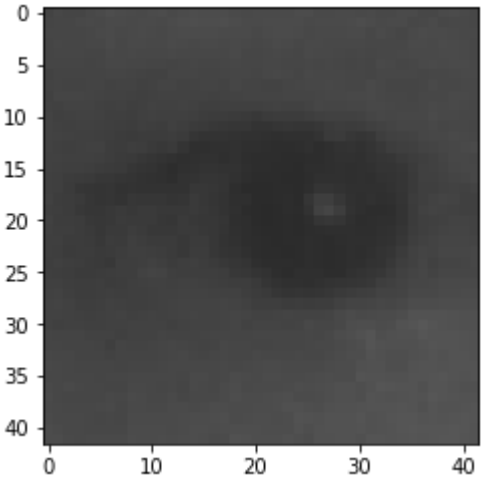
Out[136]: <matplotlib.image.AxesImage at 0x2070bab4a00>



In [137]:
```
eyeCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
eyes = eyeCascade.detectMultiScale(gray, 1.1, 4)
for x, y,w, h in eyes:
  roi_gray = gray[y:y+h, x:x+w]
  roi_color = img[y:y+h, x:x+w]
  eyess = eyeCascade.detectMultiScale(roi_gray)
  if len(eyess) == 0:
    print("eyes not detected")
  else:
    for ex, ey, ew, eh in eyess :
      eyes_roi = roi_color[ey:ey+eh, ex:ex+ew]
```

In [138]:
```
plt.imshow(cv2.cvtColor(eyes_roi, cv2.COLOR_BGR2RGB))
```

Out[138]:
```
<matplotlib.image.AxesImage at 0x20720ae5f10>
```



In [139]:
```
eyes_roi.shape
```

Out[139]:
```
(42, 42, 3)
```

In [140]:
```
final_img = cv2.resize(eyes_roi, (224,224))
final_img = np.expand_dims(final_img, axis=0)
final_img = final_img/255.0
```

In [141]:
```
new_model.predict(final_img)
```

Out[141]:
```
array([[1.5477661e-07]], dtype=float32)
```

Since the eye is open, we got the prediction value nearer to 1.

In [ ]:
```

```