



Driver Fatigue Detection using Convolutional Neural Networks

Introduction

- Road accidents mainly caused by the state of driver drowsiness. Detecting driver drowsiness (DDD) or weariness is a crucial and difficult task in preventing road-side collisions.
- We will build a Drowsiness Detection System to improve safety and prevent these incidents. When the driver's drowsiness is detected, the system would alert (alarm) the driver.
- We'll use a webcam to capture photographs as input. We will employ the cv2 technique offered by OpenCV.
- To find the face in the image, the image must be converted it to grayscale, as the OpenCV object detection algorithm only accepts grayscale images as input.

-
- It produces an array of detections with x,y coordinates as well as height, which is the width of the object's border box. We can now iterate across the faces, drawing boundary boxes for each one.
 - The image can then be running through the model to obtain a prediction. We display "Open" on the screen if the prediction is closer to 0. Otherwise, we display "Closed" (i.e., it's closer to 1).

Algorithm Used.

- The model we used is built using Convolutional Neural Networks (CNN).
- A convolutional neural network is a sort of deep neural network that works exceptionally well when it comes to picture classification.
- A CNN is made up of three layers: an input layer, an output layer, and a hidden layer with numerous layers.

Dataset Description

We are getting facial data from

UMass Amherst open eye face data: More than 13,000 photos of faces were gathered from the internet for the data collection. The name of the individual pictured has been written on each face.

Nanjing University closed eye face: This dataset includes 2423 subjects, including 1192 subjects with both eyes closed which were obtained directly from the Internet and 1231 subjects with both eyes open which were chosen from the Labeled Face in the Wild (LFW [2]) database.

Data Preparation

Data Pre-processing:

The following dependencies are required to pre-process the datasets.

1. CMAKE: required for face recognition
2. DLIB: required for face recognition
3. FACE_RECOGNITION: required for recognizing eye coordinates in a face.

```
#command line arguments to install follwing dependencies  
!brew install cmake  
!pip install dlib  
!pip install face_recognition  
!pip install playsound
```

Calling the respective functions: As we are using two different datasets, we require two different functions to pre-process the data.

Although the internal code has same logic, the only difference is how we iterate through the files in their respective folders.

```
#Preprocessing Data

crop_open_eye('open_eye_')
closed_eye_cropper('closed_eye')
```

Each image is retrieved from their folders and the coordinates are found using this snippet.

```
# Open the file using Image library/.
im=Image.open(open_eye_dataset + '/' + person+'/' +img_file)
# Apply facial recognition to the image file.
open_eye_face = face_recognition.load_image_file(open_eye_dataset + '/' + person + '/' + img_file)
# Facial coordinates are stored in the list.
face_coordinates = face_recognition.face_landmarks(open_eye_face)
```

After recognizing the coordinates, let us create a list to store their values for each eye. If the library fails to recognize the face, we skip the pre-processing step.

```
EYES = []  
try:  
    # append left eye and right eye to the above list.  
    EYES.append(face_coordinates[0]['left_eye'])  
    EYES.append(face_coordinates[0]['right_eye'])  
except:  
    continue
```

For each eye let us find the boundaries such that we can crop the eye from the face. To make sure that full eye is captured, we are adding a cushion to the range.

Adding some buffer space around the coordinates.

```
for eye in EYES:
    max_x = max([coordinate[0] for coordinate in eye])
    min_x = min([coordinate[0] for coordinate in eye])
    max_y = max([coordinate[1] for coordinate in eye])
    min_y = min([coordinate[1] for coordinate in eye])
# Finally x and y ranges are found.
xrange = -(min_x-max_x)
yrange = -(min_y-max_y)

if xrange > yrange:
    #Four boundaries are set to crop out the eye image from the face.
    right_boundary = round(.5*xrange-5+5) + max_x
    left_boudnary = min_x - round(.5*xrange-5+5)
    bottom_boundary = round(((right_boundary-left_boudnary) - yrange-5+5))/2 + max_y
    top_boundary = min_y - round(((right_boundary-left_boudnary) - yrange+8-8))/2
else:
    #Four boundaries are set to crop out the eye image from the face.
    bottom_boundary = round(.5*y_range) + max_y
    top_boundary = min_y - round(.5*y_range)
    right_boundary = round(((bottom_boundary-top_boundary) - xrange))/2 + max_x
    left_boudnary = min_x - round(((bottom_boundary-top_boundary) - yrange))/2
```

Now, we use the crop functionality of the PIL image library to crop out the eye from the face.

```
#Using above the ranges ,lets crop the eye from the face
im = Image.open(open_eye_dataset + '/' + person + '/' + img_file)
im = im.crop((left_boudnary, top_boundary, right_boundary, bottom_boundary))
```

Let us resize the image now such that all the images sent into the neural network will be having same uniform sizes.

```
# Resize image to a uniform size to sent into a model.
im = im.resize((80,80))
```

We, now create a folder and store it in a folder for sending it to training and then save it.

```
# Create a name for the folder to which the cropped image to be stored
name = '/content/preprocessed_open_eye/' + str(count) + '.jpg'
# save the image in a jpeg format.
im.save(name, 'JPEG')
```

Above all steps are same for the closed_eye_data.

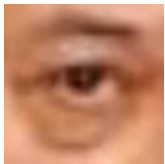
Input:
Open Eye Dataset



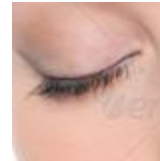
Closed Eye Dataset



Output:
Open Eye Dataset



Closed Eye Dataset



Project

<https://colab.research.google.com/drive/16-RQjI5NBKAUOEKTdg4a76xv4eZy8Pvg#scrollTo=ramCvsg5nrGM>

Code in its current state:

1. Images are stored in zipped files, so we need to unzip them.

**Phase 3 **

1. Unzip the open eyes and closed eyes folder.

```
[3]  #!/zip -r '/content/preprocessed_closed_eye.zip' 'preprocessed_closed_eye'  
     #!/zip -r '/content/preprocessed_open_eye.zip' 'preprocessed_open_eye'  
  
     !unzip '/content/preprocessed_closed_eye.zip' -d 'preprocessed_closed_eye'  
     !unzip '/content/preprocessed_open_eye.zip' -d 'preprocessed_open_eye'
```

2.Convert each image to an equivalent pixel array

PHASE 3

2. For each image in the folder convert them into equivalent pixel level data which has a size of 80*80 .

```
[ ] !pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)  
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.5)
```

```
import os  
from PIL import Image, ImageDraw  
import face_recognition  
import os  
import cv2 as c_v  
  
def loading_images(folder, eyes = 0):  
    images = []  
    for filename in os.listdir(folder):  
        try:  
            img = c_v.imread(os.path.join(folder,filename))  
            #image resizing  
            img = c_v.resize(img, (80,80))  
            images.append([img, eyes])  
        except:  
            continue  
  
    return images  
  
#Retreiving images from eyes open dataset. and converting it into a list  
open_eyes = loading_images("/content/preprocessed_closed_eye/content/preprocessed_closed_eye", 0)  
#Retreiving images from eyes closed dataset. and converting it into a list  
closed_eyes = loading_images("/content/preprocessed_open_eye/content/preprocessed_open_eye", 1)  
eyes = closed_eyes + open_eyes
```

3.Split the data into train and test split.

3.Splitting the whole data into X_train,X_test,y_train,y_test using train_test_split with a ration of 70 /30

```
[ ] import numpy as np
    from sklearn.model_selection import train_test_split

    X = []
    y = []
    for features, label in eyes:
        X.append(features)
        y.append(label)
    X = (np.array(X).reshape(-1, 80, 80, 3))/255.0
    y = np.array(y)

    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size=0.3)
```

Phase 4

Current Model architecture

This layer, rather than creating the entire image, produces sections of pixels, allowing for speedier models.

This may be more or less dense than the original photos, depending on the number of filters you use, but it will allow the model to learn more complex associations with fewer resources. We used 32 filters in total. Two 3x3s pooled together followed by three 3x3s pooled together was the best setup for me.

```

# Adding first three convolutional layers
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu', # activation function
    input_shape = (80,80,3) # shape of input (image)
))
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))

# Adding pooling after convolutional layers
model.add(MaxPooling2D(pool_size = (2,2))) # Dimensions of the region that you are pooling

# Adding second set of convolutional layers
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))

# Adding second set of convolutional layers
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))

```

```

model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))
model.add(Conv2D(
    filters = 32, # number of filters
    kernel_size = (3,3), # height/width of filter
    activation = 'relu' # activation function
))

# Add last pooling layer.
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

# Adding first dense layer with 256 nodes
model.add(Dense(256, activation='relu'))

# Adding a dropout layer to avoid overfitting
model.add(Dropout(0.3))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))

# adding output layer
model.add(Dense(1, activation = 'sigmoid'))

# compiling the model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=[tf.keras.metrics.AUC(curve = 'PR')])

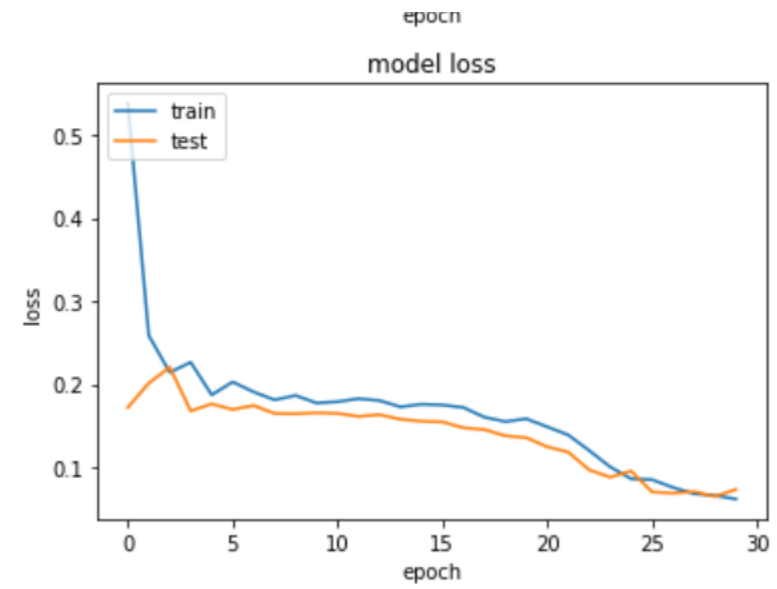
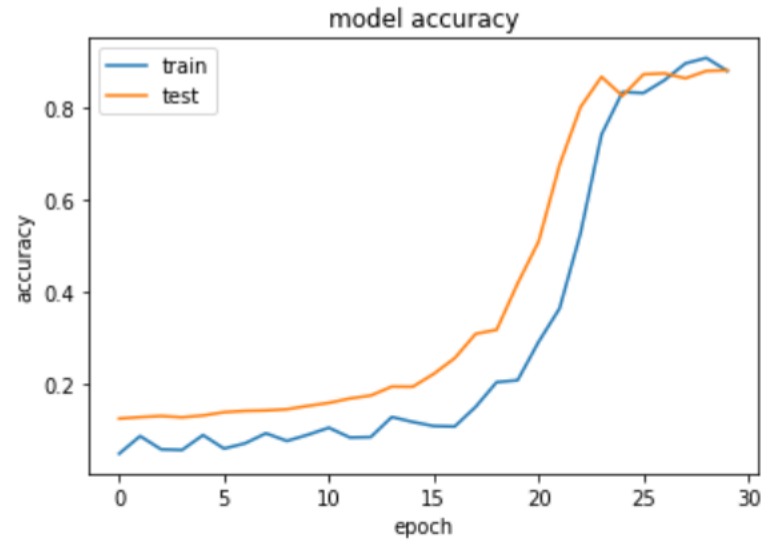
# fitting the model
model.fit(X_train,
        y_train,
        batch_size=500,
        validation_data=(X_test, y_test),
        epochs=30)

# evaluate the model
model.evaluate(X_test, y_test, verbose=1)

```

Epoch 1/30

Accuracy Plot



Frame

```
# set a counter
counter = 0

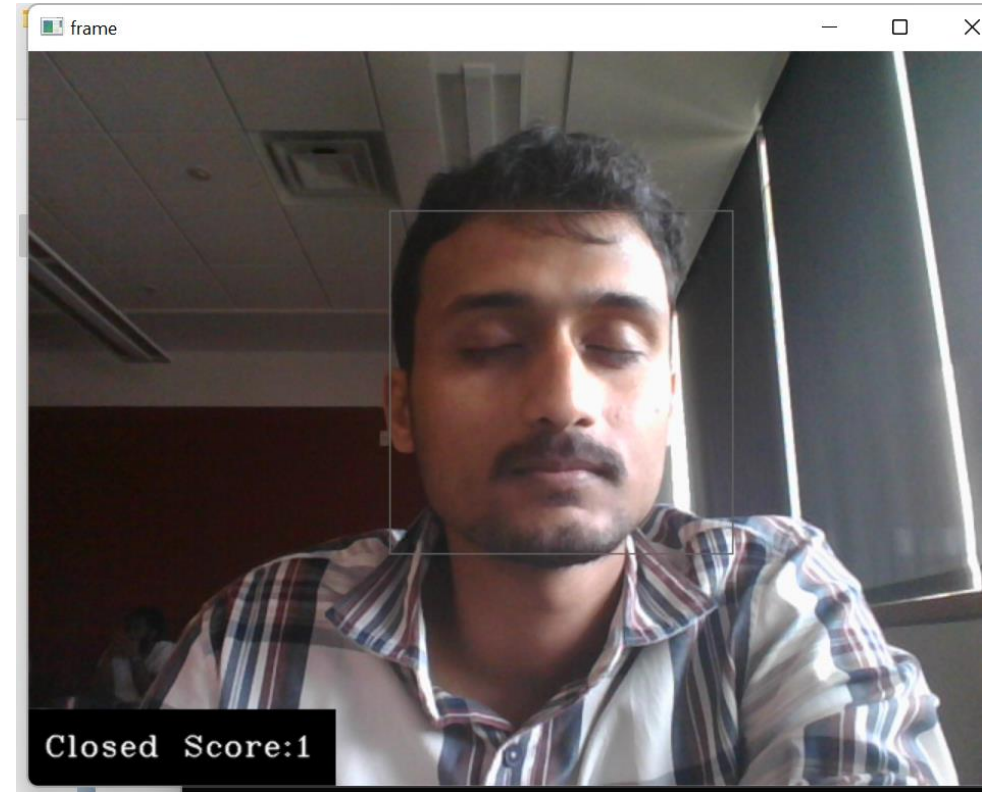
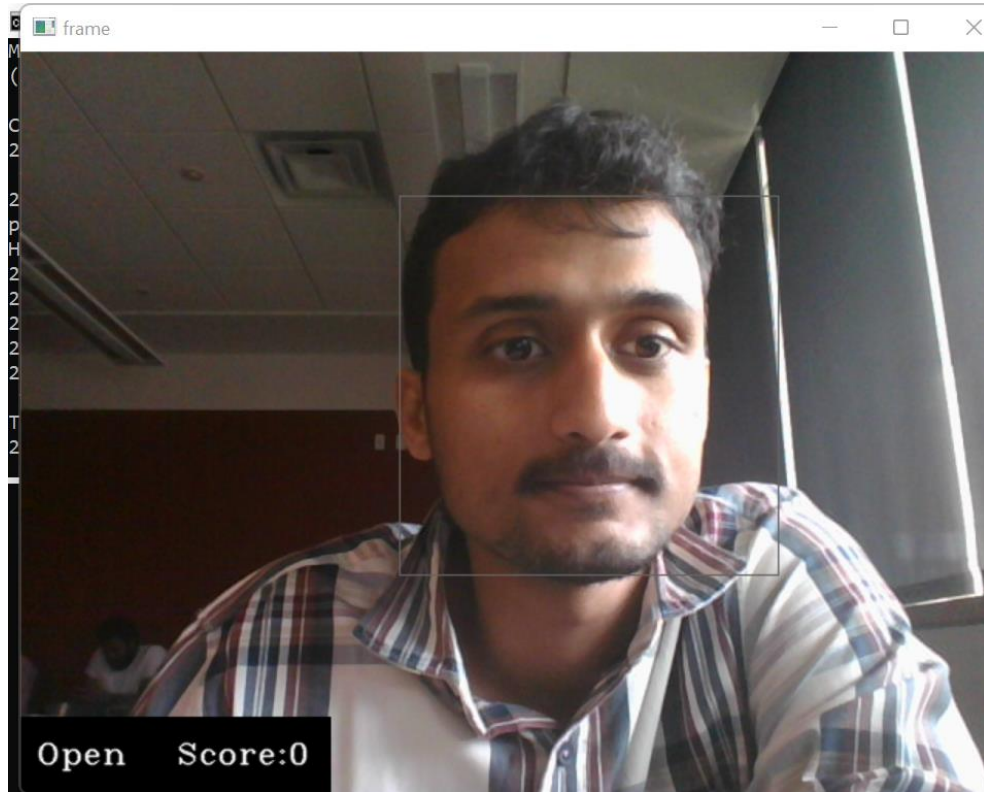
# create a while loop that runs while webcam is in use
while True:

    # capture frames being outputted by webcam
    ret, frame = cap.read()

    # use only every other frame to manage speed and memory usage
    count_of_frame = 0
    if count_of_frame == 0:
        count_of_frame += 1
        pass
    else:
        count = 0
        continue

    # function called on the frame
    predicted_image = eye_cropper(frame)
    try:
        predicted_image = predicted_image/255.0
    except:
        continue
```

Output



How are you planning to evaluate your results?

1.As of now, we will be showing the result with the accuracy which is almost equal to 80%.

Timeline

Title	Deadline
Data Pre-processing <ul style="list-style-type: none">• Data Collection• Data Augmentation	02/14/2022
Code Implementation <ul style="list-style-type: none">• Model building	03/14/2022
Accuracy Evaluation <ul style="list-style-type: none">• Testing	03/31/2022
Submission	04/15/2022

References

- Qaisar Abbas, “Hybrid Fatigue: A Real-time Driver Drowsiness Detection using Hybrid Features and Transfer Learning” International Journal of Advanced Computer Science and Applications (IJACSA), 11(1), 2020. <http://dx.doi.org/10.14569/IJACSA.2020.0110173>.
- Sahayadhas, A.; Sundaraj, K.; Murugappan, M. Detecting Driver Drowsiness Based on Sensors: A Review. *Sensors* 2012, 12, 16937-16953.

Thank You