Driver Drowsiness Detection using OpenCV and YOLOv5 (Advanced Machine Learning Project)

The University of Texas at Austin
(Team Members: Aishwarya Rajeev, Akshaya Mudar, Prathyusha Vedulla
Juhi Patel, Snehal Naravane)

Blog Link: Medium article
Code Link: Github Link

Introduction:

Human activity recognition (HAR) can play a significant role in our daily interactions. This interaction can be amongst other humans to learn more about interpersonal relationships or individual daily habits that can predict the future. Being able to study another human's activity can detail a lot about what humans are feeling physically and emotionally. In this case, we have chosen to study human drowsiness. The use of HAR is gaining popularity in the ML and AI community.

We decided to work on human drowsiness as it is one of the leading causes in road crashes. In the US, at least 100,000 crashes annually occur due to drowsy drivers, of which 71,000 result injuries, and 1,550 fatalities. In a CDC survey, an estimated 1 in 25 adult drivers (aged 19 years or older) reported having fallen asleep while driving in the previous 30 days. Considering these incidents, car manufacturers are aggressively researching options to detect drowsiness based on driving patterns and sensor input to warn the driver of impending drowsiness. While plenty of academic research is undergoing in this area, we are yet to see large-scale implementation.

HAR is the process of interpreting human activities and motions using computer and machine learning technology. The motions in our activity can be interpreted and amazed as gestures and behaviors. However, tracking human activity is still a challenging field. Researchers have tapped into deep learning to overcome some of the challenges faced otherwise. HAR can be grouped into two categories; sensor-based and vision based. In this project, we have focused on the vision-based approach since all data is pulled from video.

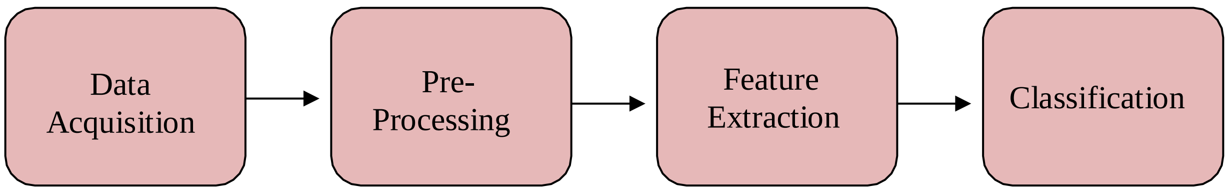


Figure 1: Process used for the vision-based approach HAR

Description of the Problem Statement:

The goal of the research is to identify driver tiredness and inform them at the appropriate moment to avoid any accidents. The problem at hand is to capture the driver's facial images using an image sensor and to detect if they are in a drowsy state or not. The deep learning model is supposed to predict the probability of being drowsy or awake.
We use a YOLOv5 model to predict whether a person feels drowsy or not based on whether the eyes are closed or open and if a yawn is detected or not. The project directly benefits the automotive sector, improves driving safety, and lowers the number of fatalities brought on by drowsy driving.

Dataset Collection:

We have 4 classes. Namely: open eye, close eye, yawn, unrecognizable. Initially we captured sample pictures through a webcam and manually labelled each image as open eye, close eye and with or without yawn. The major disadvantage with this training sample was that it had only 200 images and it was not exhaustive.
To address this, we selected a pre-classified image dataset found on Roboflow. This had around 2400 images in the training data balanced equally across the four classes. We also validated our model on the training set.
For testing, we initially tested on the test dataset from Roboflow where we knew the ground truth and then used videos from the UTA-RLDD datasets where participants had 3 states of alertness: alert, low vigilant, drowsy.

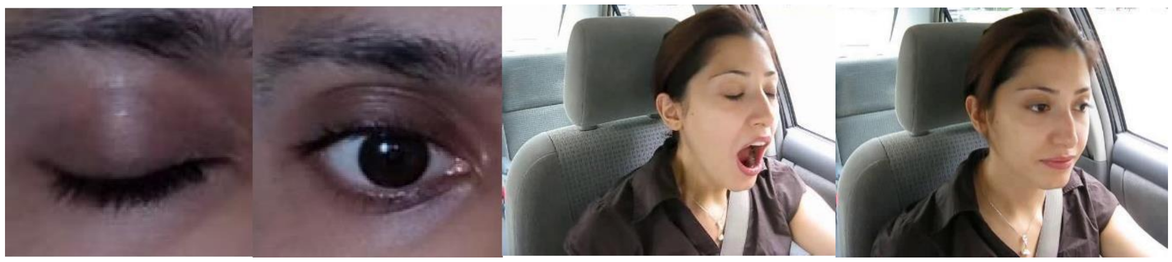


Figure 2: A snapshot of the training dataset. Labelled from left to right – closed, open, yawn, unrecognizable/no yawn

YOLO:

An object detection algorithm that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. YOLO is one of the most famous object detection algorithms due to its speed and accuracy.

Why did we use YOLO?

YOLO (You Only Look Once) is a real-time object detection algorithm.
The network looks at the complete image during training and test time and so it implicitly encodes contextual information about classes as well as their appearance and understands generalized object representation. The algorithm “only looks once” at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. The algorithm applies a single neural network (FCNN) to the full image, and then divides the image into regions where each cell is responsible for predicting a bounding box and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

- Easy to install
- Fast training
- Inference ports that work: You can infer with YOLOv5 on individual images, batch images, video feeds or webcam ports
- Intuitive data file system structure

Components of YOLO:

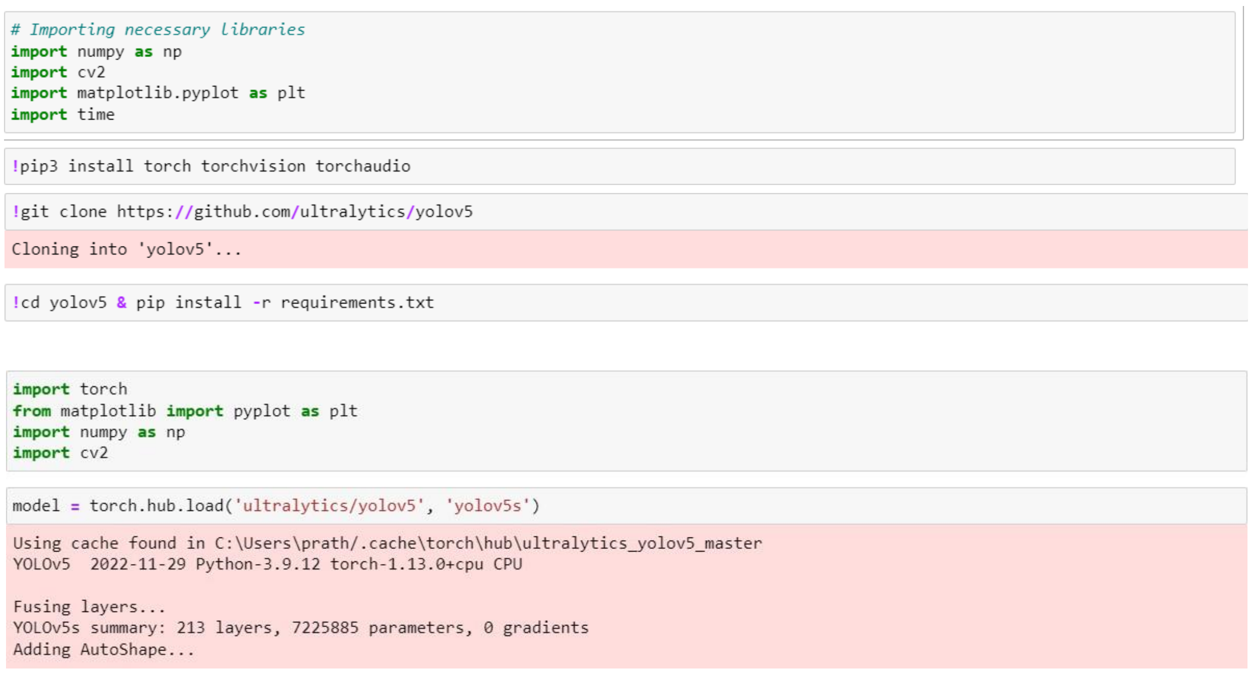
Backbone: A CNN that aggregates and forms image features at different granularities
Neck: A series of layers mix and combine image features to pass them forward for prediction
Head: Consume features from the neck and take the prediction steps

Advantages:

- It achieves high accuracy while also being able to run in real-time.
- It is open source
- YOLO model can also handle detecting smaller objects and far away objects

Methodology:

To train our model, we followed Nicholas Renotte's video on Youtube where he implemented a drowsiness detection code using YOLO and Pytorch.



```
# Importing necessary libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
import time
```

```
!pip3 install torch torchvision torchaudio
```

```
!git clone https://github.com/ultralytics/yolov5
```

```
Cloning into 'yolov5'...
```

```
!cd yolov5 & pip install -r requirements.txt
```

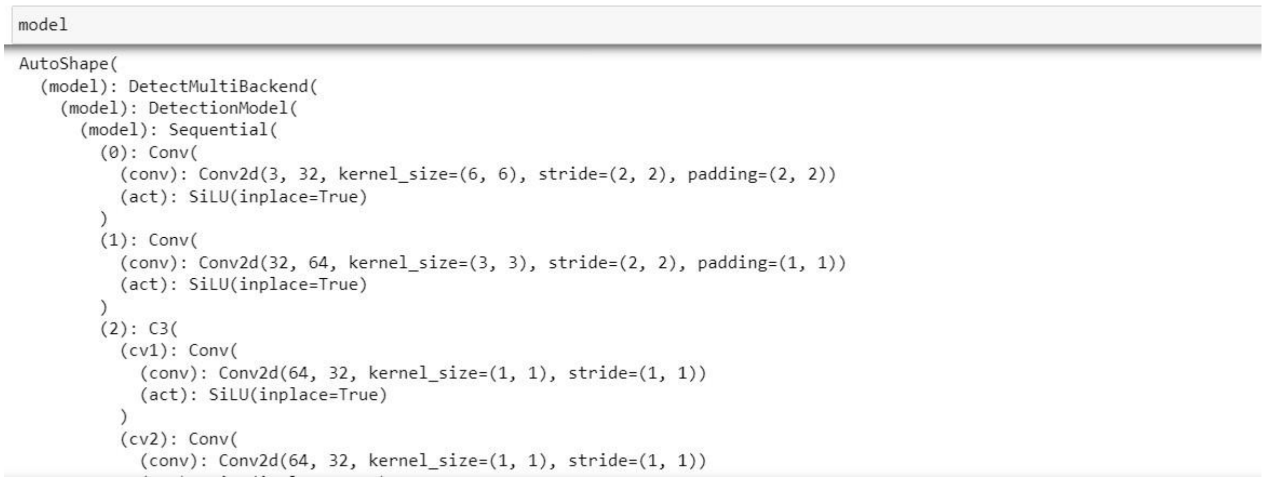
```
import torch
from matplotlib import pyplot as plt
import numpy as np
import cv2
```

```
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
```

```
Using cache found in C:\Users\prath/.cache\torch\hub\ultralytics_yolov5_master
YOLOv5  2022-11-29 Python-3.9.12 torch-1.13.0+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
Adding AutoShape...
```

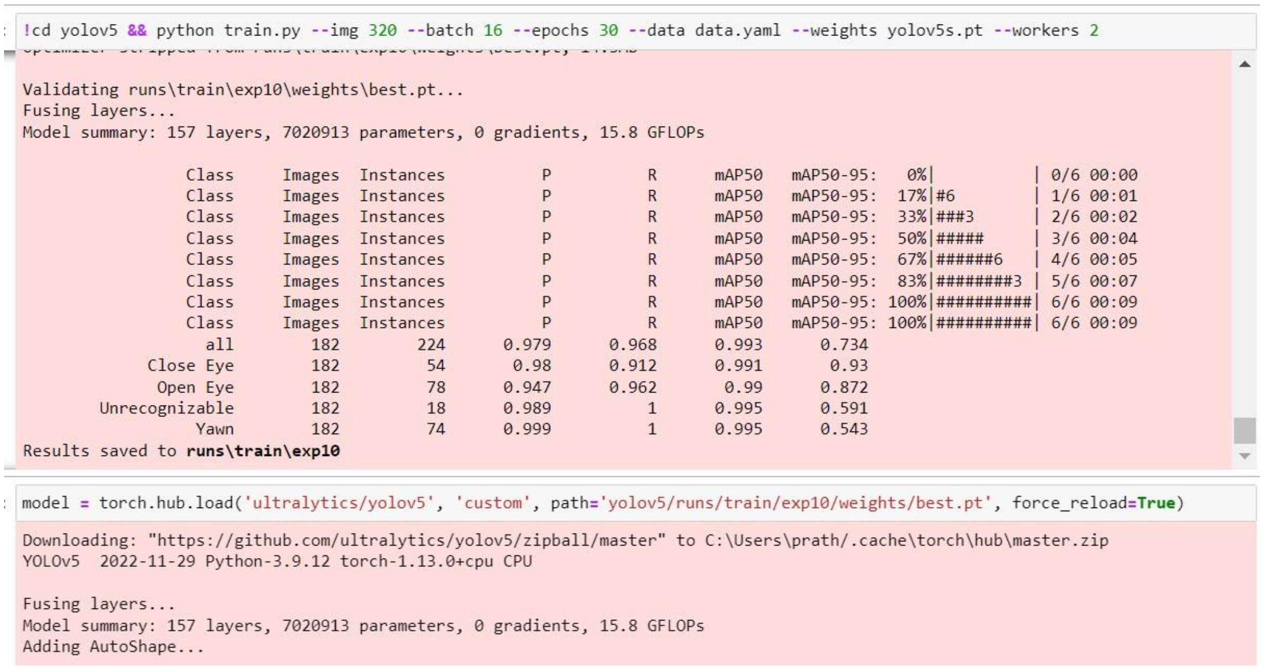
Figure 3: Importing all dependencies and libraries



```
model
```

```
AutoShape(
  (model): DetectMultiBackend(
    (model): DetectionModel(
      (model): Sequential(
        (0): Conv(
          (conv): Conv2d(3, 32, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
          (act): SiLU(inplace=True)
        )
        (1): Conv(
          (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
          (act): SiLU(inplace=True)
        )
        (2): C3(
          (cv1): Conv(
            (conv): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
            (act): SiLU(inplace=True)
          )
          (cv2): Conv(
            (conv): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
```

Figure 4: The YOLO model is a series of convolutional neural nets as displayed



```
!cd yolov5 && python train.py --img 320 --batch 16 --epochs 30 --data data.yaml --weights yolov5s.pt --workers 2
```

```
Validating runs\train\exp10\weights\best.pt...
Fusing layers...
Model summary: 157 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs

                 Class     Images  Instances          P          R      mAP50   mAP50-95:   0%|          | 0/6 00:00
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  17%|#6        | 1/6 00:01
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  33%|###3      | 2/6 00:02
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  50%|#####     | 3/6 00:04
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  67%|######6   | 4/6 00:05
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  83%|########3 | 5/6 00:07
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|##########| 6/6 00:09
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|##########| 6/6 00:09
                   all        182        224      0.979      0.968      0.993      0.734
             Close Eye        182         54       0.98      0.912      0.991       0.93
              Open Eye        182         78      0.947      0.962       0.99      0.872
        Unrecognizable        182         18      0.989          1      0.995      0.591
                  Yawn        182         74      0.999          1      0.995      0.543
Results saved to runs\train\exp10
```

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path='yolov5/runs/train/exp10/weights/best.pt', force_reload=True)
```

```
Downloading: "https://github.com/ultralytics/yolov5/zipball/master" to C:\Users\prath/.cache\torch\hub\master.zip
YOLOv5  2022-11-29 Python-3.9.12 torch-1.13.0+cpu CPU

Fusing layers...
Model summary: 157 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
Adding AutoShape...
```

Figure 4: Training the YOLO model on custom dataset from Roboflow

Testing the model:

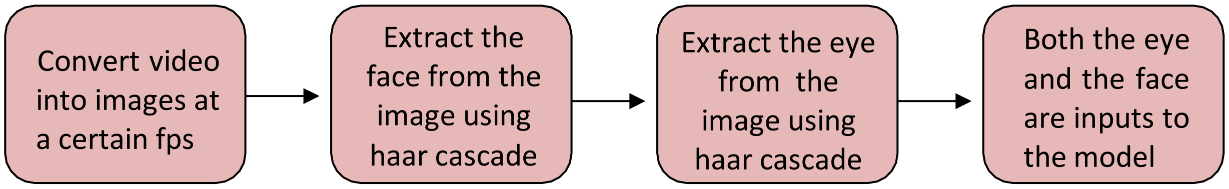
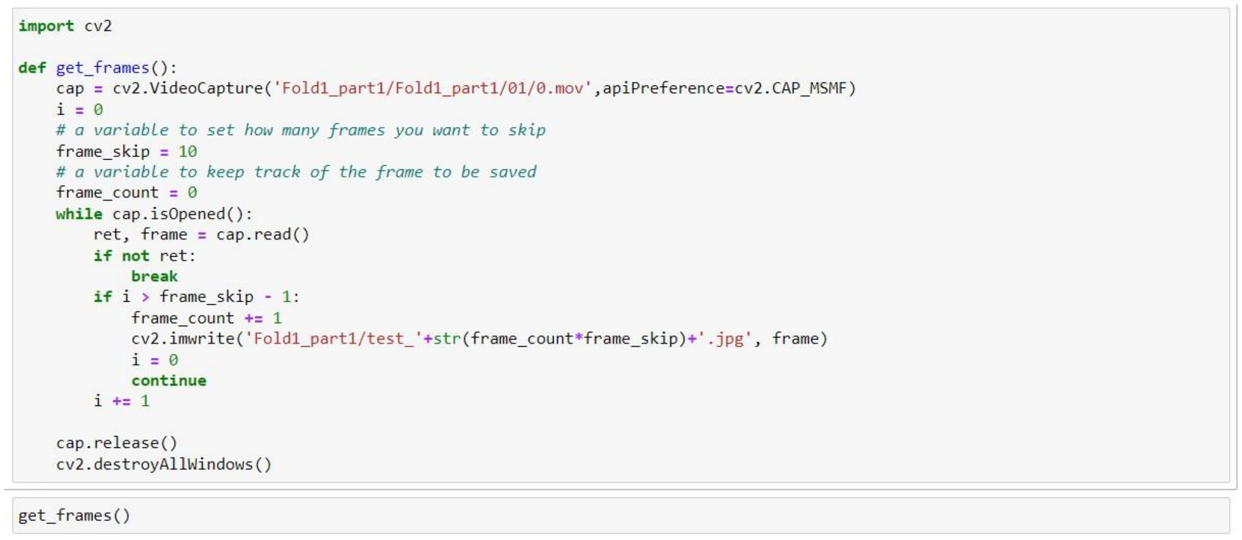

Convert video into images at a certain fps
Extract the face from the image using haar cascade
Extract the eye from the image using haar cascade
Both the eye and the face are inputs to the model

Figure 5: Pre-processing steps to use the UTA-RLDD dataset



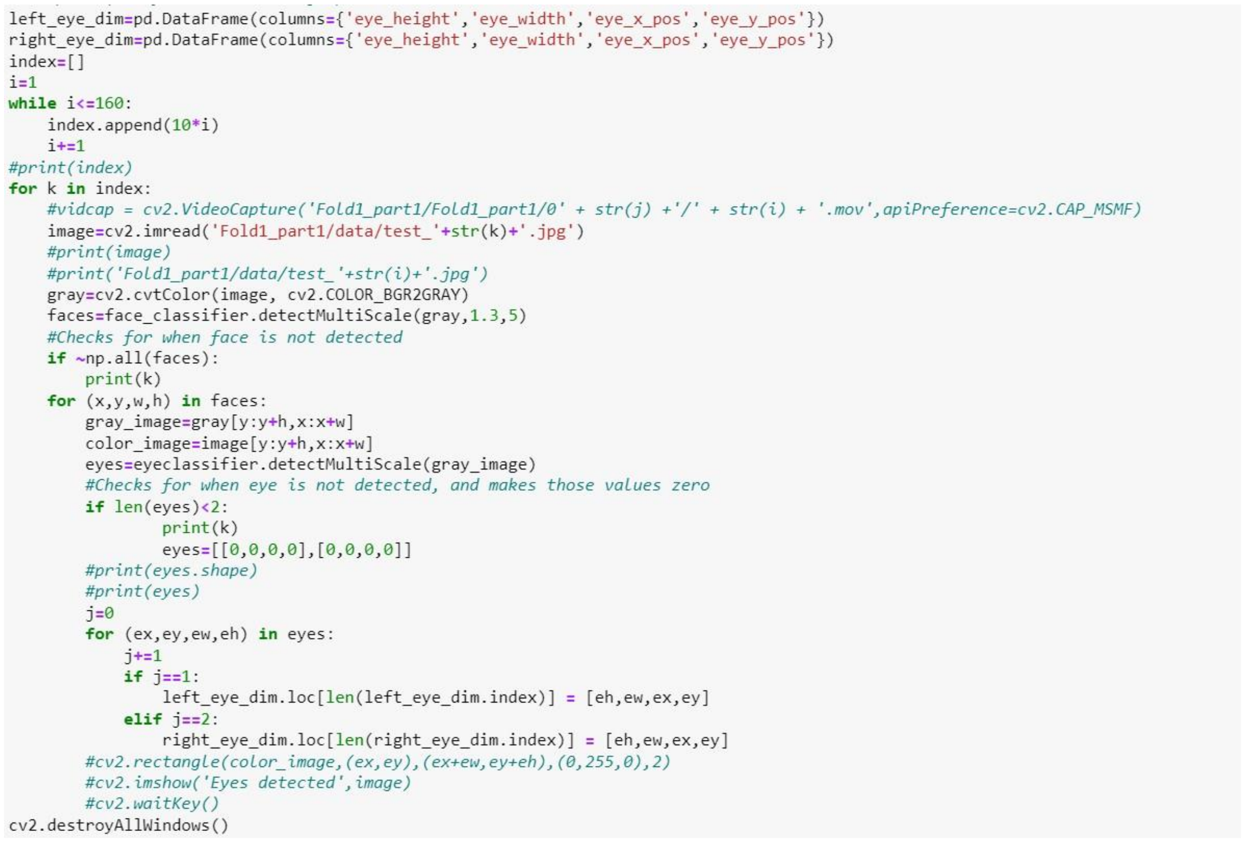
```
import cv2

def get_frames():
    cap = cv2.VideoCapture('Fold1_part1/Fold1_part1/01/0.mov',apiPreference=cv2.CAP_MSMF)
    i = 0
    # a variable to set how many frames you want to skip
    frame_skip = 10
    # a variable to keep track of the frame to be saved
    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        if i > frame_skip - 1:
            frame_count += 1
            cv2.imwrite('Fold1_part1/test_'+str(frame_count*frame_skip)+'.jpg', frame)
            i = 0
            continue
        i += 1

    cap.release()
    cv2.destroyAllWindows()
```

```
get_frames()
```

Figure 6: Converting the video into image at 1 fps



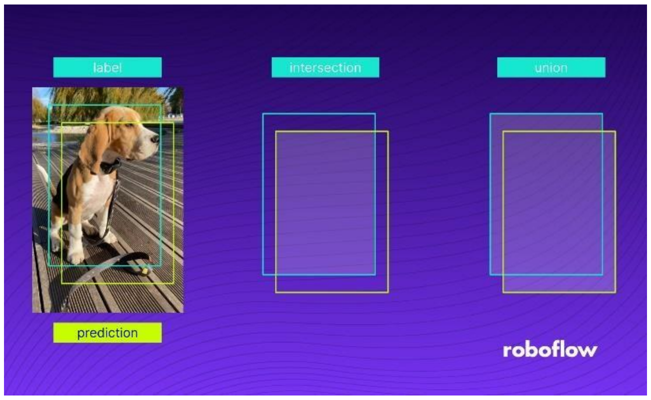
```
left_eye_dim=pd.DataFrame(columns={'eye_height','eye_width','eye_x_pos','eye_y_pos'})
right_eye_dim=pd.DataFrame(columns={'eye_height','eye_width','eye_x_pos','eye_y_pos'})
index=[]
i=1
while i<=160:
    index.append(10*i)
    i+=1
#print(index)
for k in index:
    #vidcap = cv2.VideoCapture('Fold1_part1/Fold1_part1/0' + str(j) +'/' + str(i) + '.mov',apiPreference=cv2.CAP_MSMF)
    image=cv2.imread('Fold1_part1/data/test_'+str(k)+'.jpg')
    #print(image)
    #print('Fold1_part1/data/test_'+str(i)+'.jpg')
    gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces=face_classifier.detectMultiScale(gray,1.3,5)
    #Checks for when face is not detected
    if ~np.all(faces):
        print(k)
    for (x,y,w,h) in faces:
        gray_image=gray[y:y+h,x:x+w]
        color_image=image[y:y+h,x:x+w]
        eyes=eyeclassifier.detectMultiScale(gray_image)
        #Checks for when eye is not detected, and makes those values zero
        if len(eyes)<2:
                print(k)
                eyes=[[0,0,0,0],[0,0,0,0]]
        #print(eyes.shape)
        #print(eyes)
        j=0
        for (ex,ey,ew,eh) in eyes:
            j+=1
            if j==1:
                left_eye_dim.loc[len(left_eye_dim.index)] = [eh,ew,ex,ey]
            elif j==2:
                right_eye_dim.loc[len(right_eye_dim.index)] = [eh,ew,ex,ey]
        #cv2.rectangle(color_image,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
        #cv2.imshow('Eyes detected',image)
        #cv2.waitKey()
cv2.destroyAllWindows()
```

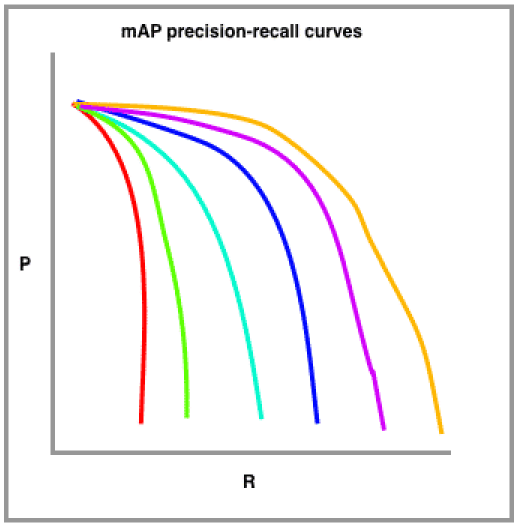
Figure 7: Extracting the face and the eye form the image using haar cascade files

Results:

The mean average precision (mAP) is used to evaluate object detection models such as R-CNN and YOLO. The mAP computes a score by comparing the ground-truth bounding box to the detected box. The higher the score, the more precise the model's detections.

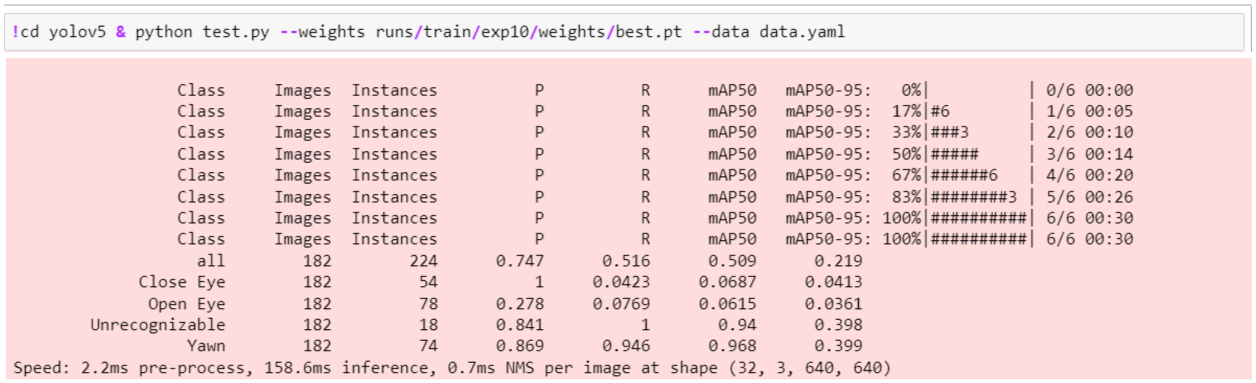
To train a model we use an image along with the ground-truth. This truth is the bounding box for each object in the image. The model will not predict the exact ground-truth but a good prediction is one where the bounding box predicted can still allow correct object-detection. A good explanation on the meaning and use of mAP score can be found here :–
https://blog.roboflow.com/mean-average-precision/





Source: https://blog.roboflow.com/mean-average-precision/

While we were able to achieve a very high mAP score during training, our test mAP score was 50.9%. There were a high number of misclassifications, and this could be due to a higher threshold of IoU set than necessary.



```
!cd yolov5 & python test.py --weights runs/train/exp10/weights/best.pt --data data.yaml
```

```
                 Class     Images  Instances          P          R      mAP50   mAP50-95:   0%|          | 0/6 00:00
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  17%|#6        | 1/6 00:05
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  33%|###3      | 2/6 00:10
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  50%|#####     | 3/6 00:14
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  67%|######6   | 4/6 00:20
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  83%|########3 | 5/6 00:26
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|##########| 6/6 00:30
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|##########| 6/6 00:30
                   all        182        224      0.747      0.516      0.509      0.219
             Close Eye        182         54          1     0.0423     0.0687     0.0413
              Open Eye        182         78      0.278     0.0769     0.0615     0.0361
        Unrecognizable        182         18      0.841          1       0.94      0.398
                  Yawn        182         74      0.869      0.946      0.968      0.399
Speed: 2.2ms pre-process, 158.6ms inference, 0.7ms NMS per image at shape (32, 3, 640, 640)
```

Figure 8: Results on the UTA-RLDD dataset

Future Improvements and Way Forward:

There are a few things we can do going forward to enhance our findings and hone the models. We plan to improve this score by training for a higher number of epochs.
We plan to test our model on the UTA RLDD data. The RLDD dataset consists of around 30 hours of RGB videos of 60 healthy participants. The videos are labeled at three stages of drowsiness/alertness.
We also plan on incorporating an alarm system that goes off whenever the state of an image is predicted for more than 3 continuous frames. After all, our main project is based on alerting sleepy drivers.

Acknowledgment:

We also want to thank Dr. Joydeep Ghosh for giving us this learning opportunity and for all the invaluable advice he was able to offer throughout this process.

References:

1. https://sites.google.com/view/utarldd/home
2. https://www.youtube.com/watch?v=tFNJGim3FXw
3. https://roboflow.com/
4. https://blog.roboflow.com/mean-average-precision/
5. https://www.cdc.gov/sleep/features/drowsy-driving.html
6. https://wires.onlinelibrary.wiley.com/doi/am-pdf/10.1002/widm.1254