Drowsiness Detection

Project Working:

This project will work in two phases in first phase we will train a model on eyes if open/closed using different models. After achieving good accuracy and low loss we will give it to another code which is phase two in which we will use face detector libraries to detect a face so we can see where the eyes are and then employ our model to predict if our outputs are good or not.

For first phase training I tried different model architectures, but they were too big as I wanted my project to run on a micro-controller. I designed a model of my own to reduce the model file size but when I tried to further decrease it, I got many issues like validation loss was enormous. So, the model I used is the max size reduced and efficient model which I tried to develop.

There are many face-detection libraries, but I went with this was because it was more accurate and gave better results as compared to other libraries such as harsh cascade even thought it might be more computationally complex or consuming.

Step 1: Downloading and Uploading dataset

First, we will download the Drowsiness Detection Dataset from Kaggle the link is given below:

Dataset Link: [Human eyes open\close | Kaggle]

After downloading this dataset from the website, we will replace the word in labels as follows:

Open Eyes: 1

Close Eyes: 0

After this we will upload the data on our drive that we will mount on the Google Colab.

Step 2: Changing runtime and loading the data in Colab

- Create a new notebook in Colab
- Go on the Runtime tab and change the Runtime type to GPU and save it.
- Mount the Drive in which you uploaded the Dataset you want to train the model.
- After mounting the data read the dataset using the code given below.
- This code will unzip the file into the Colab hardisk.

```
!unzip /content/drive/MyDrive/Drowsiness2.zip -d /content/Untitled
```

Step 3: Importing Libraries and setting our Train and Test data Paths.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import tensorflow as tf
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model
from keras.optimizers import RMSprop
from keras.layers import Activation, Dropout, Flatten, Dense, GlobalMaxPoo
ling2D, Conv2D, MaxPooling2D
from keras.callbacks import CSVLogger
from sklearn.model_selection import train_test_split

path = '/content/Untitled/data'
train_dir = os.path.join(path, 'train')
test_dir = os.path.join(path, 'test')
print(train_dir)
print(test_dir)
print(os.listdir(train_dir))
```

Step 4: Preprocessing and Generation of Data

- Now we input image size of what we want.
- Batch size and Epochs.
- Preprocess our data (Rescale, zoom, flip, shear)
- Use an image generator to generate our data in class mode.

```
# Hyperparams
IMAGE_SIZE = 128
IMAGE_WIDTH, IMAGE_HEIGHT = IMAGE_SIZE, IMAGE_SIZE
EPOCHS = 10
BATCH_SIZE = 16

input_shape = (IMAGE_WIDTH, IMAGE_HEIGHT, 3)

# data generators
training_data_generator = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

validation_data_generator = ImageDataGenerator(rescale=1./255)
# Data preparation

training_generator = training_data_generator.flow_from_directory(
    train_dir,
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode="binary")
validation_generator = validation_data_generator.flow_from_directory(
    test_dir,
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode="binary")


sample, label = next(validation_generator)
print(sample[0])
print(label[0])
```

<u>Step 5</u>: My Model

- My Neural Network Uses 13 layers.
- Four Convolution layers
- Four Pool layers
- One Flatten layer
- One Dropout
- Three Dense Layers
- Stride of the Model is 1
- And filter size is 3x3

```
# model
model = Sequential()

model.add(Conv2D(32, 3, input_shape=input_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dropout(0.7))
model.add(Dense(128, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation = 'sigmoid'))

# model.add(Activation('sigmoid'))
model.summary()
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 12, 12, 256)       295168    
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 6, 6, 256)        0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 9216)              0         
                                                                 
 dropout (Dropout)           (None, 9216)              0         
                                                                 
 dense (Dense)               (None, 128)               1179776   
                                                                 
 dense_1 (Dense)             (None, 256)               33024     
                                                                 
 dense_2 (Dense)             (None, 1)                 257       
                                                                 
=================================================================
Total params: 1,601,473
Trainable params: 1,601,473
Non-trainable params: 0
```

Step 6: Compiling our model

- Now we will select our loss function according to our classes.
- And at the end we choose our Optimizers and Metrics.
- I used Adam optimizer and Binary Cossentropy.

```
# compile model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
print(len(training_generator.filenames))
```

Step 7: Fit/Train the data

- Now we will fit our data set.
- And save our model to a local machine.
- We trained for 10 epochs
- With Batch Size of 16.

```
# train model

history=model.fit(
    training_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
)


model.save('/content/drive/MyDrive/models/Drowsiness_model.h5')
 #   validation_steps=len(validation_generator.filenames) // BATCH_SI
ZE

model = tf.keras.models.load_model('/content/drive/MyDrive/models/Dro
wsiness_model.h5')
```

Output:

```
81675
Epoch 1/10
5105/5105 [==============================] - 371s 71ms/step - loss: 0.1713 - accuracy: 0.9335 - val_loss: 0.1599 - val_accuracy: 0.9448
Epoch 2/10
5105/5105 [==============================] - 357s 70ms/step - loss: 0.0853 - accuracy: 0.9700 - val_loss: 0.0840 - val_accuracy: 0.9705
Epoch 3/10
5105/5105 [==============================] - 357s 70ms/step - loss: 0.0693 - accuracy: 0.9757 - val_loss: 0.1274 - val_accuracy: 0.9504
Epoch 4/10
5105/5105 [==============================] - 352s 69ms/step - loss: 0.0632 - accuracy: 0.9780 - val_loss: 0.1325 - val_accuracy: 0.9485
Epoch 5/10
5105/5105 [==============================] - 361s 71ms/step - loss: 0.0575 - accuracy: 0.9799 - val_loss: 0.2493 - val_accuracy: 0.9172
Epoch 6/10
5105/5105 [==============================] - 353s 69ms/step - loss: 0.0553 - accuracy: 0.9809 - val_loss: 0.3370 - val_accuracy: 0.9038
Epoch 7/10
5105/5105 [==============================] - 356s 70ms/step - loss: 0.0531 - accuracy: 0.9815 - val_loss: 0.0867 - val_accuracy: 0.9609
Epoch 8/10
5105/5105 [==============================] - 365s 71ms/step - loss: 0.0520 - accuracy: 0.9827 - val_loss: 0.1195 - val_accuracy: 0.9531
Epoch 9/10
5105/5105 [==============================] - 351s 69ms/step - loss: 0.0495 - accuracy: 0.9829 - val_loss: 0.0677 - val_accuracy: 0.9780
Epoch 10/10
5105/5105 [==============================] - 349s 68ms/step - loss: 0.0472 - accuracy: 0.9840 - val_loss: 0.1049 - val_accuracy: 0.9500
```

<u>Step 8</u>: Predicting/Validating

- Now we will predict to check if our model was trained right or not.
- And see some output images with prediction.

```
sample1, label1 = next(validation_generator)
predictions = model.predict(sample1)
print(predictions)


def check_results():
   class_names = [ 'Open_Eyes', 'Closed_Eyes']
   sample1, label1 = next(validation_generator)
   predictions = model.predict(sample1)
   for num in range(len(predictions)):
       if predictions[num] > 0.5:
           print('prediction: '+'Closed_Eyes'+' ' + str(int(predictions[nu
m]*100))+ '%')
       else:
           print('prediction: '+'Open_Eyes'+' ' + str(100- int(predictions
[num]*100))+ '%')

       print('actual: '+ class_names[int(label1[num])])
       plt.imshow(sample1[num])
       plt.show()

check_results()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Output:

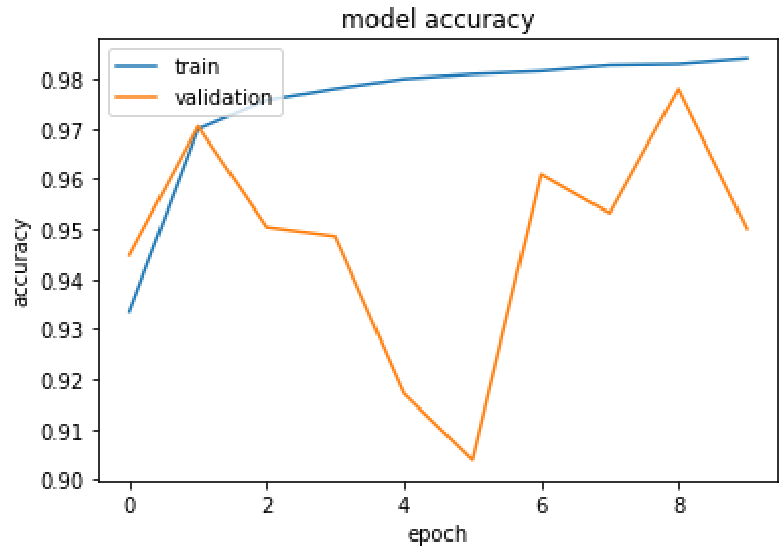


Figure 1:Model Accuracy and Validation

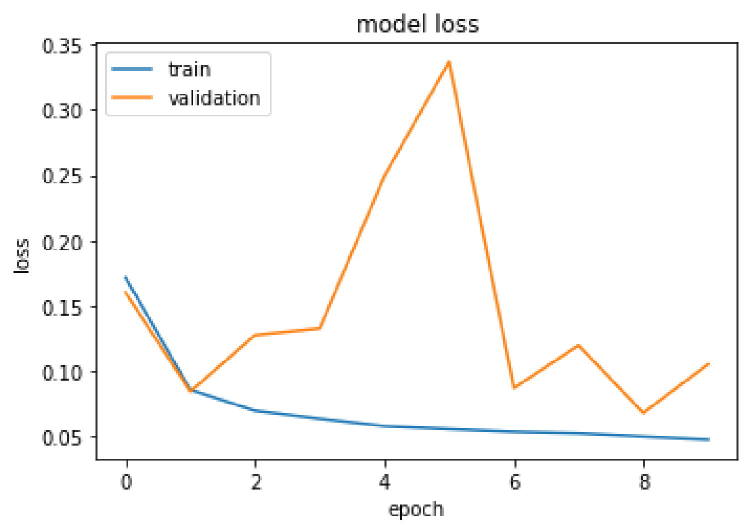


Figure 2: Validation and Training Loss

<u>Step 9</u>: Final (testing our model)

- Use face detection model to detect eyes.
- Apply our eyes closed and open model to detect the output.
- Then apply conditions accordingly to detect our output.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 10 19:12:01 2022

@author: sanau
"""

import cv2
from google.colab.patches import cv2_imshow
import numpy as np
from playsound import playsound
from PIL import Image, ImageDraw
import face_recognition
from tensorflow import keras
eye_model = keras.models.load_model('/content/drive/MyDrive/models/Drowsin
ess_model.h5')

# webcam frame is inputted into function
def eye_cropper(frame):

    # create a variable for the facial feature coordinates
    facial_features_list = face_recognition.face_landmarks(frame)

    # create a placeholder list for the eye coordinates
    # and append coordinates for eyes to list unless eyes
    # weren't found by facial recognition
    try:
        eye = facial_features_list[0]['left_eye']
    except:
        try:
            eye = facial_features_list[0]['right_eye']
        except:
            return

    # establish the max x and y coordinates of the eye
    x_max = max([coordinate[0] for coordinate in eye])
    x_min = min([coordinate[0] for coordinate in eye])
    y_max = max([coordinate[1] for coordinate in eye])
    y_min = min([coordinate[1] for coordinate in eye])
```

```
    # establish the range of x and y coordinates
    x_range = x_max - x_min
    y_range = y_max - y_min
    # in order to make sure the full eye is captured,

    # 50% cushion added to the axis with a larger range and
    # then match the smaller range to the cushioned larger range
    if x_range > y_range:
        right = round(.5*x_range) + x_max
        left = x_min - round(.5*x_range)
        bottom = round((((right-left) - y_range))/2) + y_max
        top = y_min - round((((right-left) - y_range))/2)
    else:
        bottom = round(.5*y_range) + y_max
        top = y_min - round(.5*y_range)
        right = round((((bottom-top) - x_range))/2) + x_max
        left = x_min - round((((bottom-top) - x_range))/2)

    # crop the image according to the coordinates determined above
    cropped = frame[top:(bottom + 1), left:(right + 1)]

    # resize the image
    cropped = cv2.resize(cropped, (128,128))
    image_for_prediction = cropped.reshape(-1, 128, 128, 3)

    return image_for_prediction

# initiate webcam
cap = cv2.VideoCapture(0)
w = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
h = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
if not cap.isOpened():
    raise IOError('Cannot open webcam')

# 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
    frame_count = 0
```

```
    if frame_count == 0:
        frame_count += 1
        pass
    else:
        count = 0
        continue

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

    # get prediction from model
    prediction = eye_model.predict(image_for_prediction)

    # Based on prediction, display either "Open Eyes" or "Closed Eyes"
    if prediction > 0.5:
        counter = 0
        status = 'Open'

        cv2.rectangle(frame, (round(w/2) - 110,20), (round(w/2) + 110, 80)
, (38,38,38), -1)

        cv2.putText(frame, status, (round(w/2)-
80,70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2, cv2.LINE_4)
        x1, y1,w1,h1 = 0,0,175,75
        ## Draw black backgroun rectangle
        cv2.rectangle(frame, (x1,x1), (x1+w1-20, y1+h1-20), (0,0,0), -1)
        ## Add text
        cv2.putText(frame, 'Active', (x1 +int(w1/10), y1+int(h1/2)), cv2.F
ONT_HERSHEY_SIMPLEX, 0.7, (0, 255,0),2)
    else:
        counter = counter + 1
        status = 'Closed'

        cv2.rectangle(frame, (round(w/2) - 110,20), (round(w/2) + 110, 80)
, (38,38,38), -1)

        cv2.putText(frame, status, (round(w/2)-
104,70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 2, cv2.LINE_4)
        x1, y1,w1,h1 = 0,0,175,75
        ## Draw black backgroun rectangle
```

```
cv2.rectangle(frame, (x1,x1), (x1+w1-20, y1+h1-20), (0,0,0), -1)
        ## Add text
        cv2.putText(frame, 'Active', (x1 +int(w1/10), y1+int(h1/2)), cv2.F
ONT_HERSHEY_SIMPLEX, 0.7, (0, 255,0),2)

        # if the counter is greater than 3, play and show alert that user
is asleep
        if counter > 2:

            ## Draw black background rectangle
            cv2.rectangle(frame, (round(w/2) - 160, round(h) - 200), (roun
d(w/2) + 160, round(h) - 120), (0,0,255), -1)
            cv2.putText(frame, 'DRIVER SLEEPING', (round(w/2)-
136,round(h) - 146), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,0), 2, cv2.LINE_4)
            cv2_imshow(frame)
            k = cv2.waitKey(1)
            ## Sound
            playsound('/content/alarm.wav')
            counter = 1
            continue

    cv2_imshow(frame)
    k = cv2.waitKey(1)
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

To run the file for detection of drowsiness we will have to run:

- Install Face Detection library
- Sound Alarm Library
- Then give the location of the trained model file (h file)
- Then run the code.

<u>Output</u>:

Implementation on Controller:

We can run our code and implement the model on Raspberry Pie as the total storage memory of our model is 18 mb and that of face library is 100 mb and the total space memory required for our model is 150 mb.

The Ram required to run a three second video to detect the output is

CNN Layers	**Memory**	**Parameters**
Input Layer	128x128x3=49152	0
Conv2D	126x126x32=508032	896
Max Pooling	63x63x32=127008	0
Conv2D	61x61x64=238144	18496
Max Pooling	30x30x64=57600	0
Conv2D	28x28x128=100352	73856
Max Pooling	14x14x128=25088	0
Conv2D	12x12x256=36864	295168
Max Pooling	6x6x25=9216	0
Flatten	1x1x9216=9216	0
Dropout	1x1x9216=9216	0
Dense	1x1x128=128	1179776
Dense	1x1x256=256	33024
Dense	1x1x1=1	257

RAM in Bytes	1170273x4=4681092	
RAM in Mega Bytes	**4.681092 MB**	

As we can see from above calculations the total RAM required for running the model is 4MB for forward and one frame. Extra procedures will also require some memory usage which cannot be calculated.

Processing power used is given as:

At idle:

```
CPU Usage = 14.38
```

At first few frames the processing power becomes:

```
CPU Usage = 14.48
```

And at the next few frames it becomes:

```
CPU Usage = 14.5
```

The process for few frames increases with 0.2% every iteration so it makes it easily runnable on Raspberry Pie.

As the RAM required for forward propagation is only 4 MB, we cannot use any small controllers such as Arduino. But we can use the controllers given in image given below:

<u>Orange Pie</u>:

Figure 3:Orange Pie Zero

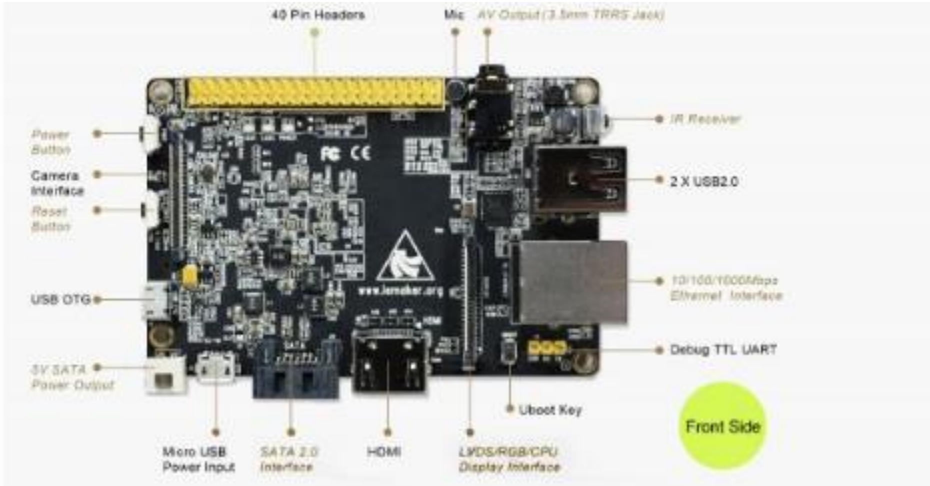
Hardware specification

CPU	H2 Quad-core Cortex-A7 H.265/HEVC 1080P.
GPU	·Mali400MP2 GPU @600MHz ·Supports OpenGL ES 2.0
Memory (SDRAM)	256MB/512MB DDR3 SDRAM(Share with GPU)(256MB version is Standard version)
Onboard Storage	TF card (Max. 32GB)/ Spi Flash
Onboard Network	10/100M Ethernet RJ45 POE is default off.
Onboard WIFI	XR819, IEEE 802.11 b/g/n
Audio Input	MIC
Video Outputs	Supports external board via 13pins
Power Source	USB OTG can supply power
USB 2.0 Ports	Only One USB 2.0 HOST, one USB 2.0 OTG
Buttons	Power Button
Low-level peripherals	26 Pins Header, compatible with Raspberry Pi B+ 13 Pins Header, with 2x USB, IR pin, AUDIO(MIC, AV)
LED	Power led & Status led
Supported OS	Android, Lubuntu, Debian, Raspbian

Orange Pi PC H3:

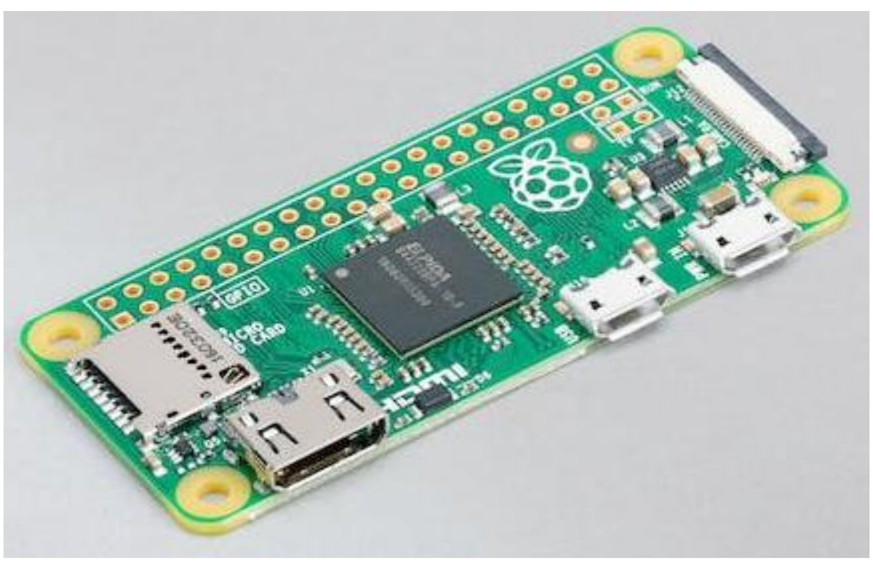
CPU	H3 Quad-core Cortex-A7 H.265/HEVC 4K
GPU	·Mali400MP2 GPU @600MHz ·Supports OpenGL ES 2.0
Memory (SDRAM)	1GB DDR3 (shared with GPU)

Banana pie M3:



1. SOC:Allwinner® A20(sun 7i)
2. CPU:ARM® Cortex™-A7 Dual-Core1GHz (ARM v7 instruction set)
3. GPU:Mali400MP2 Complies with OpenGL ES 2.0/1.1 (hardware acceleration support)
4. SDRAM:1GB DDR3 (shared with GPU)

Raspberry Pie Zero:



1GHz single-core CPU
512MB RAM
Mini HDMI port
Micro USB OTG port
Micro USB power
HAT-compatible 40-pin header
Composite video and reset headers
CSI camera connector (v1.3 only)

We can also reduce the RAM usage and processing usage by reducing the number of frames it operates to detect the drowsiness of the driver.

Conclusion:

We detected drowsiness of a driver using CNN and Python Face Recognition libraries and succeeded in reducing the memory and processing it uses so it is implementable on the Controller.

We used face recognition software so we can detect eyes on a face as the Neural

Network doesn’t directly detect images form a face, so we used it to detect face and see where the eyes are present then use the trained model to detect our output.