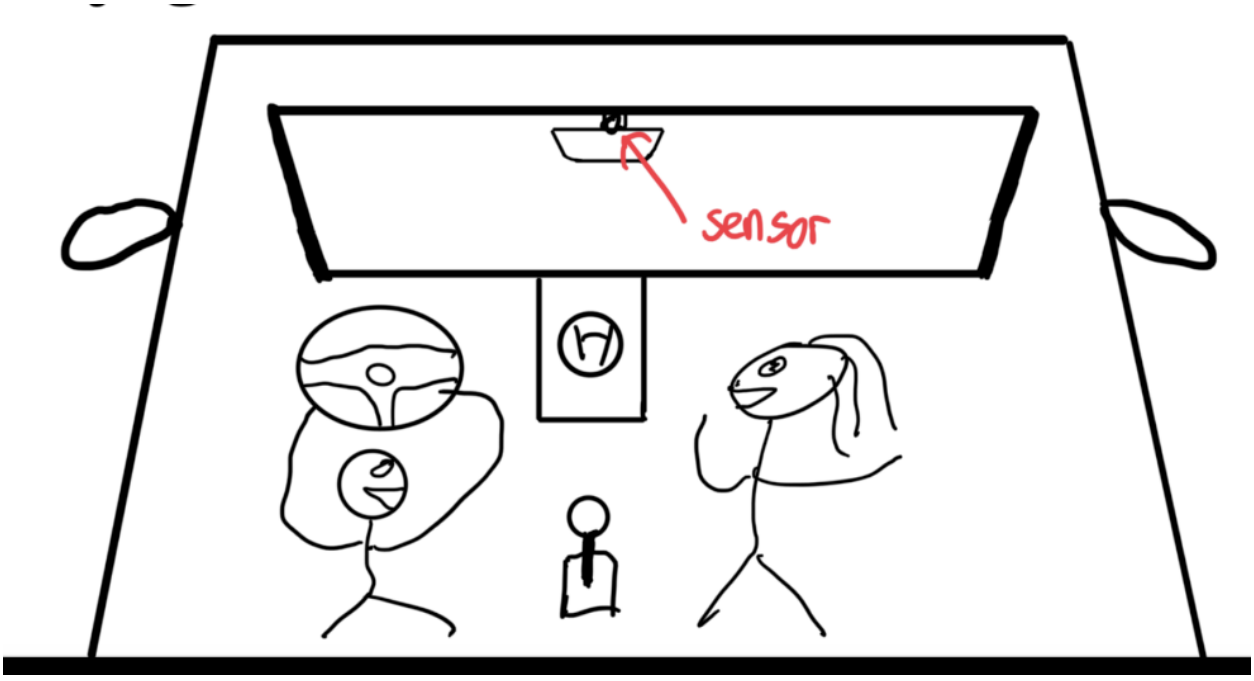
Project D - DRIVER DROWSINESS DETECTION

1. Project Introduction

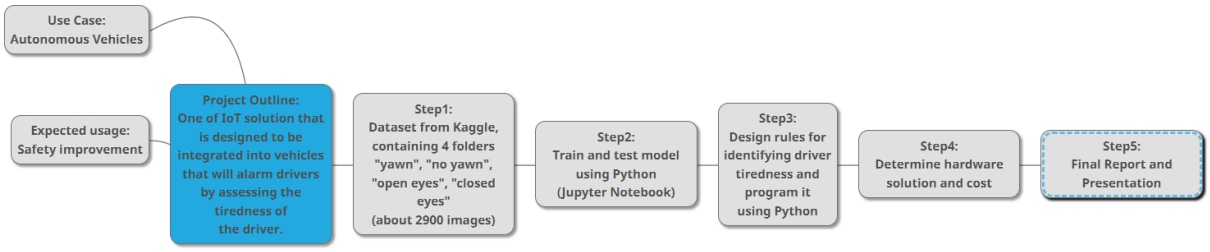
-Motivation: Most people are aware of the danger in drinking, texting, using drugs while driving. However, drowsy driving was underestimated by the general public; in fact, “A study by the AAA Foundation for Traffic Safety estimated that around 330,000 drowsy driving crashes occur annually. About 6,400 people were killed in such incidents.” (nsc.org, n.d.)

-Justification: Even though some modern vehicles are equipped with safety features such as lane assist and automatic emergency braking system, a lot of times, it is too late for such features to kick in; moreover, it is difficult for the investigators to determine whether a driver was drowsy at the time of a crash.

-Our idea is to use embedded IoT device to identify drivers drowsiness according to drivers’ eye position and mouth position using simple machine learning techniques and real life experiments. When a driver is identified to be drowsy while driving, alarms will be triggered to warn the driver, The goal of this concept is to contribute to road safety improvement.

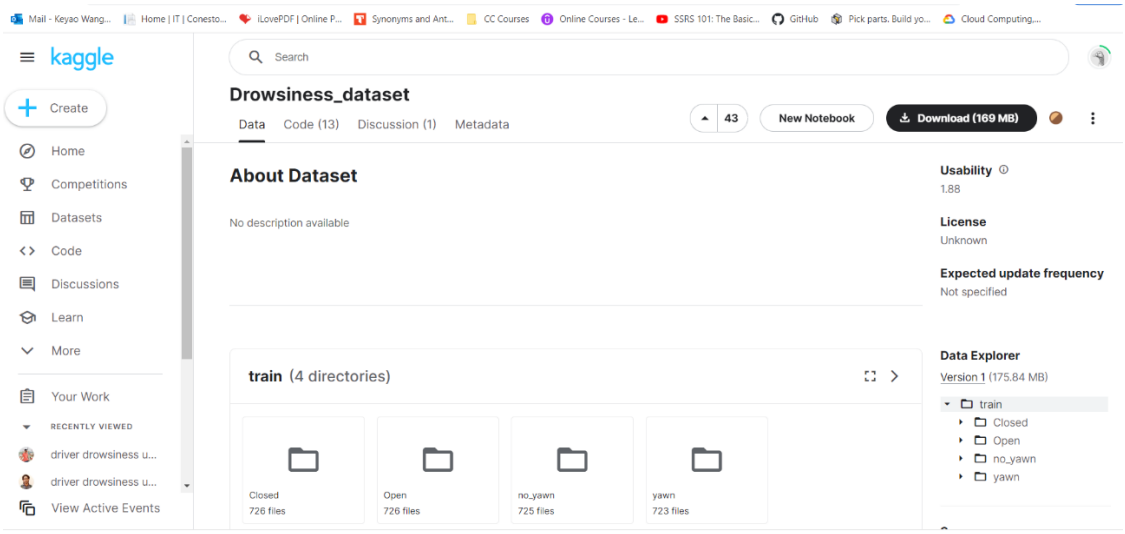


2. Solution Architecture



3. Data Access

-'Drowsiness' dataset from Kaggle which consists of 2900 images.
https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset



4. Train/Test Dataset "yawn & no yawn" :

In [1]:
```
from tensorflow.keras.preprocessing import image
from PIL import Image
import random
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import cv2
import matplotlib.pyplot as plt
```

Creating a data set

```
def DataSet():
    #Define paths for training sets
    train_path_yawn = 'W:/Case study dataset/train/yawn/'
    train_path_no_yawn = 'W:/Case study dataset/train/no_yawn/'
    # List the file name of the image in the path
    imglist_train_yawn = os. listdir(train_path_yawn)
    imglist_train_no_yawn = os.listdir(train_path_no_yawn)
    # Define numpy objects to convert images to numoy vector form
    X_train = np.empty((len(imglist_train_yawn) + len(imglist_train_no_yawn), 224, 224, 3))
    Y_train = np.empty((len(imglist_train_yawn) + len(imglist_train_no_yawn), 2))
    #Record the number of pictures
    count = 0
    for img_name in imglist_train_yawn:
        #Get the path of the image
        img_path = train_path_yawn + img_name
        # The image.load_img() function reads the corresponding image and converts it to the target size
        img = image.load_img(img_path, target_size=(224, 224))
        # Turn the image into a numpy array and divide by 255 to normalize, so the shape of img is (224,224,3)
        img = image.img_to_array(img) / 255.0
        # # Load the processed image into the X_train object
        X_train[count] = img
        ## Load the processed images into the defined Y_train object, which is 1,0 since they are all yawn images
        Y_train[count] = np.array((1,0))
        count+=1
```

```
    # Go through the pictures without yawning
    for img_name in imglist_train_no_yawn:
        img_path = train_path_no_yawn + img_name
        # The image.load_img() function reads the corresponding image and converts it to the target size
        img = image.load_img(img_path, target_size=(224, 224))
        # Turn the image into a numpy array and divide by 255 to normalize, so the shape of img is (224,224,3)
        img = image.img_to_array(img) / 255.0
        # # Load the processed image into the X_train object
        X_train[count] = img
        # Load the processed images into the defined Y_train object, which is 1,0 since they are all yawn images
        Y_train[count] = np.array((0,1))
        count+=1
    # Shuffle the data in the train set
    index = [i for i in range(len(X_train))]
    random.shuffle(index)
    X_train = X_train[index]
    Y_train = Y_train[index]
    #return two train dataset
    return X_train,Y_train
X_train,Y_train = DataSet()
#No test set is required, and each validation splits the data from the training set to the test set
np.save('W:/Case study dataset/test3/X_train', X_train, allow_pickle=True, fix_imports=True)
np.save('W:/Case study dataset/test3/Y_train', Y_train, allow_pickle=True, fix_imports=True)
print('X_train shape : ',X_train.shape)
print('Y_train shape : ',Y_train.shape)
```

```
X_train shape :  (1448, 224, 224, 3)
Y_train shape :  (1448, 2)
```

In order to avoid overfitting, we use random sample for testing.

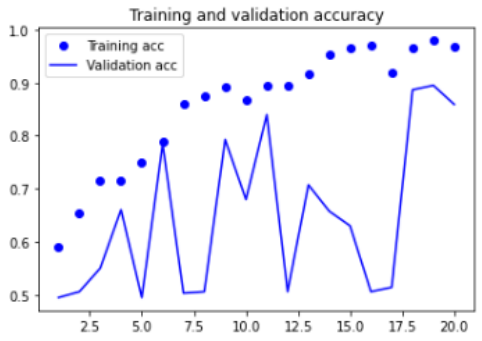
Training model

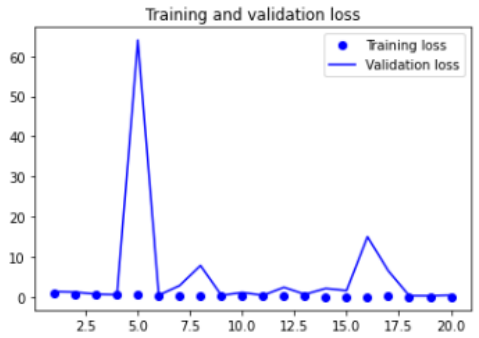
In [3]:
```
import scipy
from scipy import ndimage
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
```

In [4]:
```
X_train = np.load('W:/Case study dataset/test3/X_train.npy')
Y_train = np.load('W:/Case study dataset/test3/Y_train.npy')
model = ResNet50(weights=None,classes=2)
model.compile(optimizer=Adam(learning_rate =0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
#model save path
filepath='W:/Case study dataset/model/myresnet50model_2classes_times=1_e20bs10_etimes={epoch:02d}_valacc={val_accuracy:.2f}.h5'
#Save it after every round of training
checkpoint = ModelCheckpoint(filepath, verbose=1,save_best_only=False,save_weights_only=False)
callbacks_list = [checkpoint]
#training
history=model.fit(X_train, Y_train, epochs=20, batch_size=10,validation_split=0.25,callbacks=callbacks_list)
```

```
Epoch 1/20
109/109 [==============================] - ETA: 0s - loss: 0.9563 - accuracy: 0.5902
Epoch 1: saving model to W:/Case study dataset/model\myresnet50model_2classes_times=1_e20bs10_etimes=01_valacc=0.49.h5
109/109 [==============================] - 397s 4s/step - loss: 0.9563 - accuracy: 0.5902 - val_loss: 1.4504 - val_accuracy: 0.
4945
Epoch 2/20
109/109 [==============================] - ETA: 0s - loss: 0.6168 - accuracy: 0.6529
Epoch 2: saving model to W:/Case study dataset/model\myresnet50model_2classes_times=1_e20bs10_etimes=02_valacc=0.51.h5
109/109 [==============================] - 359s 3s/step - loss: 0.6168 - accuracy: 0.6529 - val_loss: 1.2927 - val_accuracy: 0.
5055
```

```
# draw
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





5. Train/Test the eye dataset(same procedure):

```
X_train shape :  (1452, 224, 224, 3)
Y_train shape :  (1452, 2)
```

Training model

```
import scipy
from scipy import ndimage
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
```

```
X_train = np.load('W:/Case study dataset/test2/X_train.npy')
Y_train = np.load('W:/Case study dataset/test2/Y_train.npy')
model = ResNet50(weights=None,classes=2)
model.compile(optimizer=Adam(learning_rate =0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
#model save path
filepath='W:/Case study dataset/model-e/myresnet50model_2classes_times=1_e20bs10_etimes={epoch:02d}_valacc={val_accuracy:.2f}.h5
#Save it after every round of training
checkpoint = ModelCheckpoint(filepath, verbose=1,save_best_only=False,save_weights_only=False)
callbacks_list = [checkpoint]
#training
history=model.fit(X_train, Y_train, epochs=20, batch_size=10,validation_split=0.25,callbacks=callbacks_list)
```

```
Epoch 1/20
109/109 [==============================] - ETA: 0s - loss: 0.6539 - accuracy: 0.7493
Epoch 1: saving model to W:/Case study dataset/model-e\myresnet50model_2classes_times=1_e20bs10_etimes=01_valacc=0.49.h5
109/109 [==============================] - 379s 3s/step - loss: 0.6539 - accuracy: 0.7493 - val_loss: 1.5684 - val_accuracy: 0.
4876
```

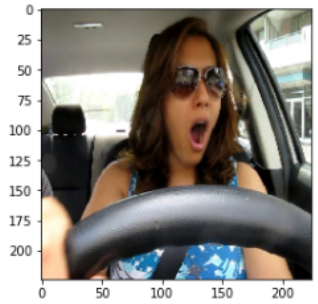
6. Connect model to real application

Invoking model

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

```
# Import the model
model = tf.keras.models.load_model('W:/Case study dataset/model/myresnet50model_2classes_times=1_e20bs10_etimes=19_valacc=0.90.h5
# test data path
img_path = 'W:/Case study dataset/test/yawn.jpg'
# The original image used in the test needs to be scaled to the 224×224 size image matched by the model
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)
img = image.img_to_array(img)/ 255.0
# Adds a fourth dimension to the batch
img = np.expand_dims(img, axis=0)
print(model.predict(img))
```

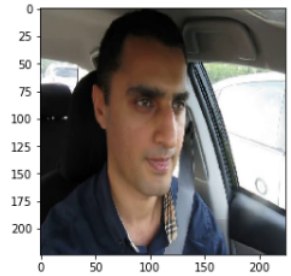
```
1/1 [==============================] - 1s 1s/step
[[9.999994e-01 5.846878e-07]]
```



It can be seen that the result given by the model is close to (1,0), and the picture is judged to be yawning

```
Import the model
del = tf.keras.models.load_model('W:/Case study dataset/model/myresnet50model_2classes_times=1_e20bs10_etimes=19_valacc=0.90.h5')
test data path
g_path = 'W:/Case study dataset/test/noyawn.jpg'
The original image used in the test needs to be scaled to the 224×224 size image matched by the model
g = image.load_img(img_path, target_size=(224, 224))
t.imshow(img)
g = image.img_to_array(img)/ 255.0
Adds a fourth dimension to the batch
g = np.expand_dims(img, axis=0)
int(model.predict(img))
```

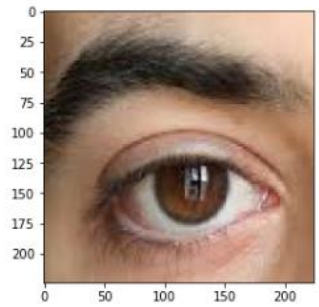
```
1/1 [==============================] - 1s 826ms/step
[[3.7041432e-04 9.9962962e-01]]
```



It can be seen that the result given by the model is close to (0,1), and the picture is judged to be noyawning

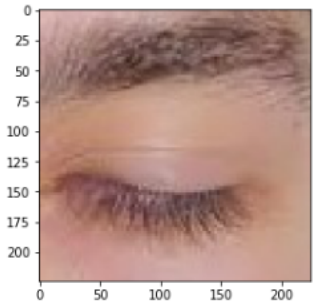
```
# Import the model
model = tf.keras.models.load_model('W:/Case study dataset/model-e/myresnet50model_2classes_times=1_e20bs10_etimes=17_valacc=0.99.
# test data path
img_path = 'W:/Case study dataset/test/Open.jpg'
# The original image used in the test needs to be scaled to the 224×224 size image matched by the model
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)
img = image.img_to_array(img)/ 255.0
# Adds a fourth dimension to the batch
img = np.expand_dims(img, axis=0)
print(model.predict(img))
```

```
1/1 [==============================] - 1s 857ms/step
[[9.9999833e-01 1.6645153e-06]]
```




```
# Import the model
model = tf.keras.models.load_model('W:/Case study dataset/model-e/myresnet50model_2classes_times=1_e20bs10_etimes=17_valacc=0.99.
# test data path
img_path = 'W:/Case study dataset/test/Closed.jpg'
# The original image used in the test needs to be scaled to the 224×224 size image matched by the model
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)
img = image.img_to_array(img)/ 255.0
# Adds a fourth dimension to the batch
img = np.expand_dims(img, axis=0)
print(model.predict(img))
```

```
1/1 [==============================] - 1s 1s/step
[[2.9157038e-04 9.9970835e-01]]
```



Define rules for alerting drivers:

=> If driver yawns, first warning will be triggered.

=> If the drivers eye is closed, a warning will be triggered every 0.5 seconds until driver opens eyes.

=> This is an infinite loop which runs every 0.5 seconds, the physical sensor will take a picture of the driver every 0.5sec and load into the data set for the system to check for the above rule.

Practical Application

```
import time
```

```
while True:
    # Test the face first
    model = tf.keras.models.load_model('W:/Case study dataset/model/myresnet50model_2classes_times=1_e20bs10_etimes=19_valacc=0.9
    # test data path ,update as the camera shoots
    img_path = 'W:/Case study dataset/ptest/face.jpg'
    # The original image used in the test needs to be scaled to the 224×224 size image matched by the model
    img = image.load_img(img_path, target_size=(224, 224))
    img = image.img_to_array(img)/ 255.0
    # Adds a fourth dimension to the batch
    img = np.expand_dims(img, axis=0)
    resultf = model.predict(img)
    # If the driver yawns
    if resultf[0][0]>0.9 :
        while True:
            print("warning")
            # check the eye
            model2 = tf.keras.models.load_model('W:/Case study dataset/model-e/myresnet50model_2classes_times=1_e20bs10_etimes=1
            # test data path, update as the camera shoots
            img_path2 = 'W:/Case study dataset/ptest/eye.jpg'
            # The original image used in the test needs to be scaled to the 224×224 size image matched by the model
            img2 = image.load_img(img_path2, target_size=(224, 224))
            img2 = image.img_to_array(img2)/ 255.0
            # Adds a fourth dimension to the batch
            img2 = np.expand_dims(img2, axis=0)
            resulte = model.predict(img2)
            # Check if driver's eye is open
            if resulte[0][0]>0.9:
                # stop warning
                break
            else:
                # Keep warning until driver's eyes open
                time.sleep(0.5)
                continue
    else:
        time.sleep(0.5)
        continue
```

Check that it can successfully print warning messages:

```
WARNING:tensorflow:6 out of the last 7 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001F
2A9840670> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) cr
eating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of t
ensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option
that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retraci
ng and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 1s 818ms/step
warning
1/1 [==============================] - 0s 83ms/step
warning
1/1 [==============================] - 0s 142ms/step
warning
1/1 [==============================] - 0s 90ms/step
warning
1/1 [==============================] - 0s 115ms/step
warning
1/1 [==============================] - 0s 88ms/step
warning
1/1 [==============================] - 0s 74ms/step
```

6. Summary

-Techniques used: Machine Learning Classification

-Algorithm used: CNN (Convolutional neural network)

-model: pre-trained model “Resnet50” from the Tensorflow package.

7. Opportunity

-From a safety point of view, this concept will likely to improve the overall safety among all drivers across the globe, reduce accident rates caused by impaired driving and potentially save lives. Moreover, it will also ease the investigating procedures when deadly accidents happen, since the actions of the drivers will be recorded.

8. Challenges

-When it comes to IoT applications, privacy is always a concern by the general public. Most people wont feel comfortable when being monitored by a camera.

9. Potential solution

1. All pictures being captured are stored locally only.
2. Optional switch, the driver has the option to turn the device off.
3. Erase all data when each driving session is over. (Start fresh each time)

References:

-nsc.org. Drivers are Falling Asleep Behind the Wheel. National Safety Council. (n.d.). Retrieved November 12, 2022, from https://www.nsc.org/road/safety-topics/fatigued-driver.