The University of Oklahoma

Price College of Business

MIT-5432 Machine Learning

A Deep-Learning Approach to Driver Drowsiness Detection

Mentored by

Dr. Sophie Zhai

Submitted by

Shohruz Junaidov Srija Jasthi Vishal Reddy Musku

Table of Contents

Abstract

Driver drowsiness is one of the reasons for the large number of road accidents these days. With the advancement in Computer Vision technologies, smart/intelligent cameras are developed to identify drowsiness in drivers, thereby alerting drivers which in turn reduces accidents when they are in fatigue. In this work, a new framework is proposed using deep learning to detect driver drowsiness based on Eye state while driving the vehicle. To detect the face and extract the eye region from the face images, Viola-Jones face detection algorithm is used in this work. Stacked deep convolution neural network is developed to extract features from dynamically identified key frames from camera sequences and used for learning phase. A SoftMax layer in CNN classifier is used to classify the driver as sleep or non-sleep. This system alerts drivers with an alarm when the driver is in sleepy mood. The proposed work is evaluated on a collected dataset and shows better accuracy with 96.42% when compared with traditional CNN. The limitation of traditional CNN such as pose accuracy in regression is overcome with the proposed Staked Deep CNN.

Domain Introduction

Computer vision is a field of computer science that focuses on enabling computers to identify and understand objects and people in images and videos. Like other types of AI, computer vision seeks to perform and automate tasks that replicate human capabilities. In this case, computer vision seeks to replicate both the way humans see, and the way humans make sense of what they see. The range of practical applications for computer vision technology makes it a central component of many modern innovations and solutions. We are using computer vision technology in our project to detect the drowsiness of the drivers using their facial features, especially their eyes, to see whether they are sleeping or not.

Task Introduction

Image classification and object detection are two important tasks in the computer vision landscape. For instance, given an image of a cat, an image classifier would output a label such as “cat.” While image classification focuses on assigning a single label to an entire image, object detection models go a step beyond by recognizing and pinpointing the positions of numerous objects within an image. In other words, object detection not only categorizes the objects present, but also draws bounding boxes around them to indicate their exact location.

Our project involves solving two tasks. Firstly, identifying the location of eyes from the human face in the picture and then classifying the identified object image or giving a label as ‘close’ or ‘open’ as we are classifying based on the eyes in this scenario.

Dataset Introduction

The dataset used for this project originated from two different data sources. One of the datasets used is collected and hosted by an individual user in an AWS S3 bucket for public use. We used the dataset as it was suitable for our project. The other data set was found on Kaggle. We did not get the original datasets used in the paper as they were not available. We combined images from both the data sets and used it. The dataset that we used contains a total of 2852 images for both the categories combined (open -1 and closed eye - 0). We used 60-40 train-test split to train and test our model meaning we used 60% of the dataset to train and held 40% of it. We further split the 40% of test data into 30-70. Held 70% of it for testing and 30% of for validation. Our training dataset contains 2852 images in both categories combined. The number of images representing the closed eye are 1426 and the number of images representing the open eye are 1426. Our testing dataset contains 899 images in both categories combined. The number of images representing closed eye are 400 and the number of images representing the open eye are 399. Each image is of the size (128,128,1) in our dataset. Our validation dataset contains 342 images in both categories combined. The number of images representing the closed eye are 175 and the number of images representing the open eye are 167. The data we used including both the training and testing datasets are also very balanced as shown in the figures below.

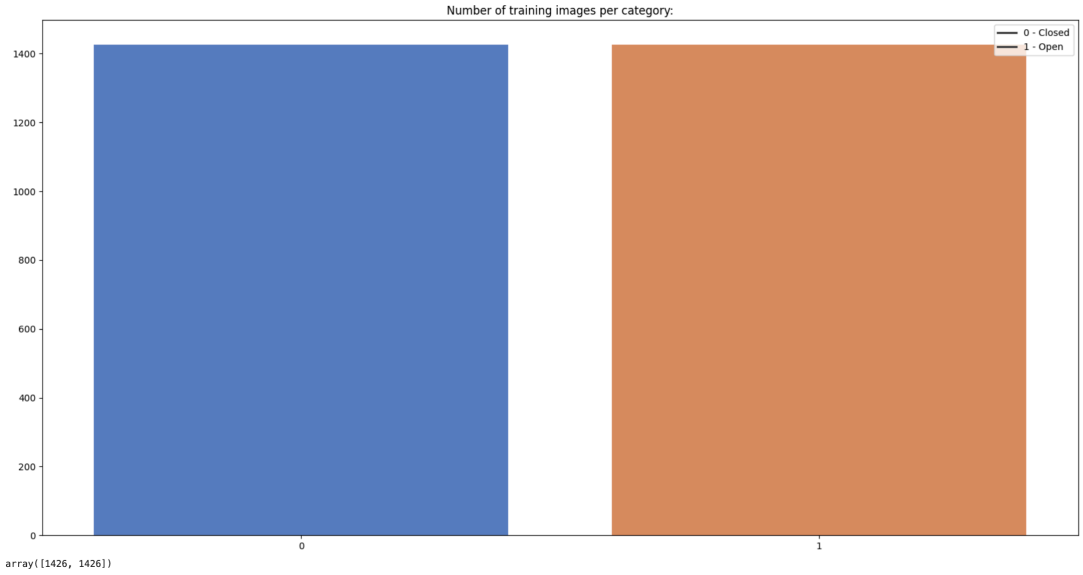


Figure 1: Total Number of Images in the Dataset

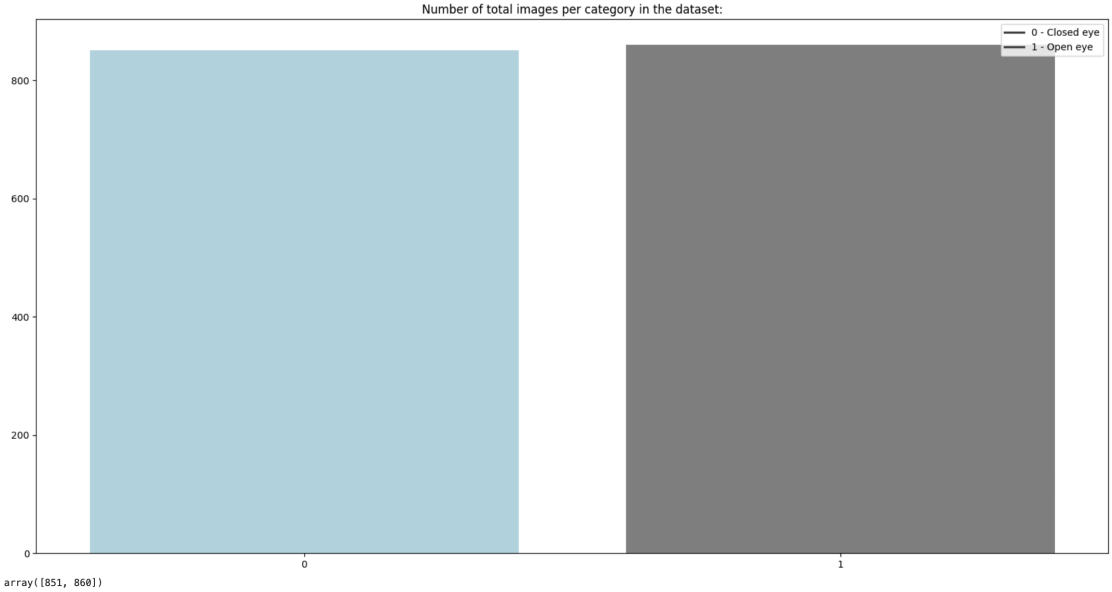


Figure 2: Number of images per category training dataset

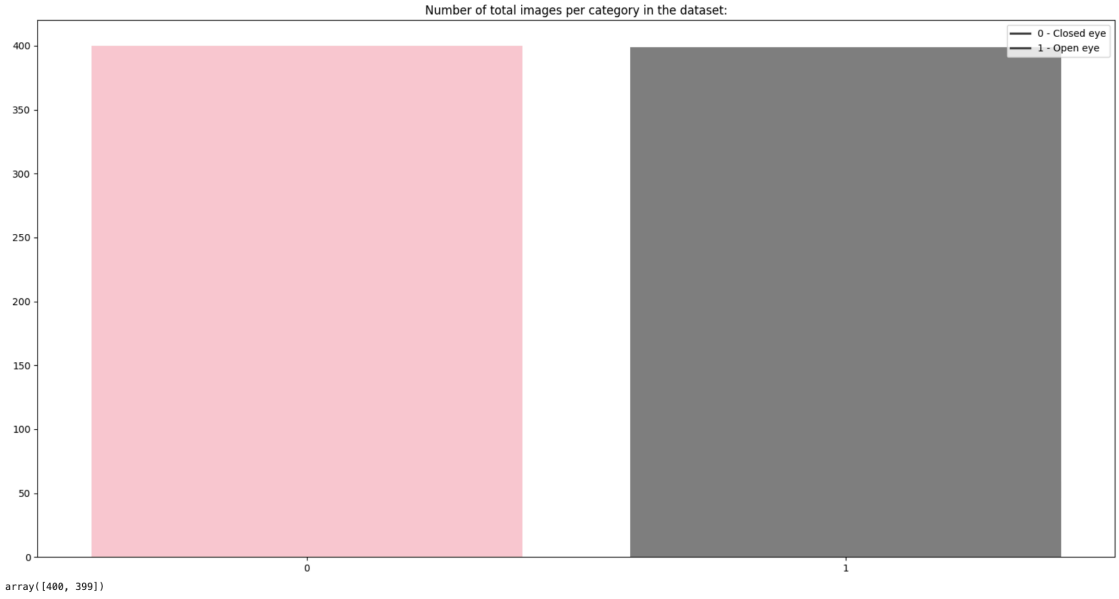


Figure 3: Number of images per category testing dataset

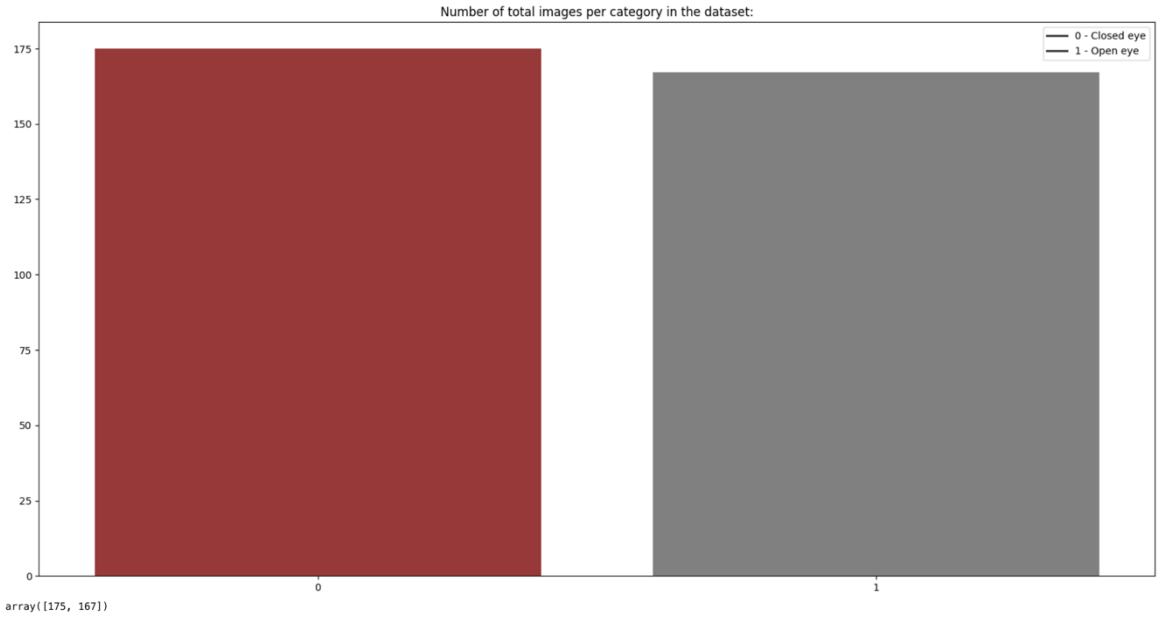


Figure 4: Number of images per category validation dataset

Problem Formulation

Driver drowsiness significantly contributes to road traffic accidents, being directly linked to around 15-20% of all serious incidents on European roads and accounting for approximately 2.2% of fatal crashes in the United States (Berghe, 2021)The development of an effective detection system is crucial for mitigating this risk, necessitating technologies that can reliably identify early signs of fatigue and initiate preventative measures. This project aims to harness computer vision technologies to analyze key physiological indicators of drowsiness, such as eye closure thereby providing real-time alerts to potentially fatigued drivers. This project leverages a given dataset of driver images to develop a computer vision system capable of detecting signs of drowsiness. This proactive detection mechanism is aimed at reducing the likelihood of accidents caused by drowsy driving, enhancing road safety in scenarios where continuous video monitoring is not feasible (Yang & Yi, 2024).

Related work

The landscape of driver drowsiness detection has been extensively explored using a variety of methods ranging from physiological measures to behavioral analytics employing advanced machine learning techniques. Traditional approaches have relied on physiological signals such as Electrocardiography (ECG), Electroencephalography (EEG), and Electrooculography (EOG) to assess a driver's alertness level. Although accurate, these methods are often impractical for real-time applications due to the intrusive nature of the required devices (Seyed Mohammad Reza

Noori, 2023). In recent years, the focus has shifted towards non-intrusive methods using computer vision and machine learning algorithms. Techniques such as the Viola-Jones face detection algorithm have been commonly used to identify facial features, which are then analyzed for signs of drowsiness (Danisman, Bilasco, Djeraba, & Ihaddadene, 2010).This method, along with the analysis of eye blink rates and yawning frequency, has proven effective but requires high-resolution images to accurately detect subtle changes in facial expressions (Abtahi, Hariri, & Shirmohammadi, 2011).

Advanced machine learning models, particularly Convolutional Neural Networks (CNNs), have been applied to improve detection accuracy by analyzing video frames for dynamic signs of fatigue, such as the duration of eye closures and head movements. These models leverage temporal and spatial features extracted from video sequences to predict drowsiness states more reliably (Dwivedi, Biswaranjan, & Sethi, 2014).

The integration of deep learning techniques has further enhanced the capability to detect nuanced patterns associated with drowsiness. For instance, systems employing deep networks like AlexNet, VGG-FaceNet, and Flow ImageNet have been tested on comprehensive driver datasets, achieving significant accuracy in real-world conditions (Sanghyuk Park, 2016).Moreover, emerging methods combining both visual and brain activity data have shown promise in creating robust detection systems that can operate effectively in diverse driving environments (Picot, Charbonnier, & Caplier, 2011).

Model details

The proposed model has three phases:

1. Pre-processing stage
2. Feature extraction
3. Deep CNN Classifier

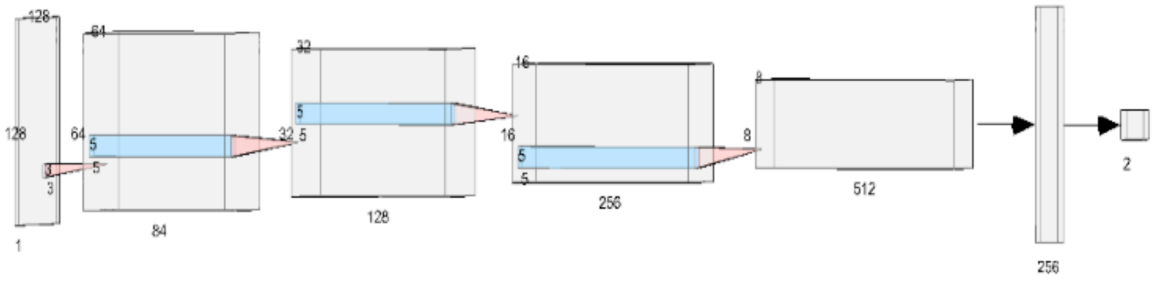


Figure 5: Proposed deep CNN (4 layer) model

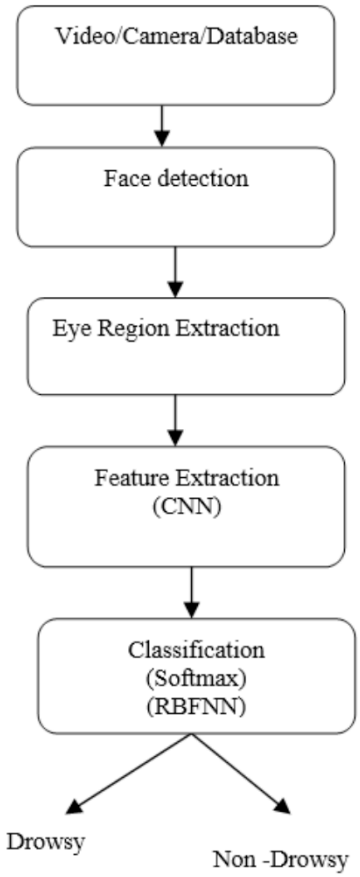


Figure 6: Proposed system architecture

We used two model architectures for this project. The primary architecture that we used, comes from the research paper that we chose. The secondary architecture was built just to experiment with a model to see how the performance of the model changes with additional convolutional layers and different number of filters. For our primary architecture, 4 convolutional layers and one fully connected layer are used. Extracted key images with size of 128 X 128 are passed as input to the convolution layer-1 (Conv2d_1). In Conv2d_1 input image is convolved with 84 filters of size 3x3. After convolution, batch Normalization, non-linear transformation ReLU, Max pooling over 2 × 2 cells are included in the architecture, which is followed by dropout with 0.25%. Conv2d_1 required 840 parameters. Batch_normalization_1 is done with 336 parameters. The output of convolution layer-1 is fed into the convolution layer- 2(Conv2d_2). In Conv2d_2, input is convolved with 128 filters with size 5x5 each. After convolution, batch Normalization, non-linear transformation ReLU, MaxPooling over 2 × 2 cells with stride 2 followed by dropout with 0.25% applied. Conv2d_2 required 268928 parameters. Batch_normalization_2 required 512 parameters. The output of convolution layer-2 is fed into the convolution layer- 3(Conv2d_3). In Conv2d_3, input is convolved with 256 filters with size 5x5 each. After convolution, Batch Normalization, non-linear transformation ReLU, MaxPooling over 2 × 2 cells with stride 2 followed by dropout with 0.25% applied, Conv2d_3 required 819456 parameters. Batch_normalization_3 required 1024 parameters. The output of convolution layer-3 is fed in to the convolution layer-

4(Conv2d_4). In Conv2d_4 input is convolved with 512 filters with size 5x5 each. After convolution, Batch Normalization, non-linear transformation ReLU, Max Pooling over 2 × 2 cells with stride 2 followed by dropout with 0.25% applied. Conv2d_4 required 3277312 parameters. Batch_normalization_4 required 2048 parameters. Fully connected layer that is dense_1 required 8388864 parameters.

Proposed CNN model required 12,757,874 trainable parameters. The output of classifier is two state, so output layer having only two outputs. Adam method is used for Optimization. Here softmax classifier is used for classification. In our proposed CNN framework, the 256 outputs of fully connected layer are the deep features retrieved from input eye images. The final 2 outputs can be the linear combinations of the deep features (Reddy & Behera, 2022).

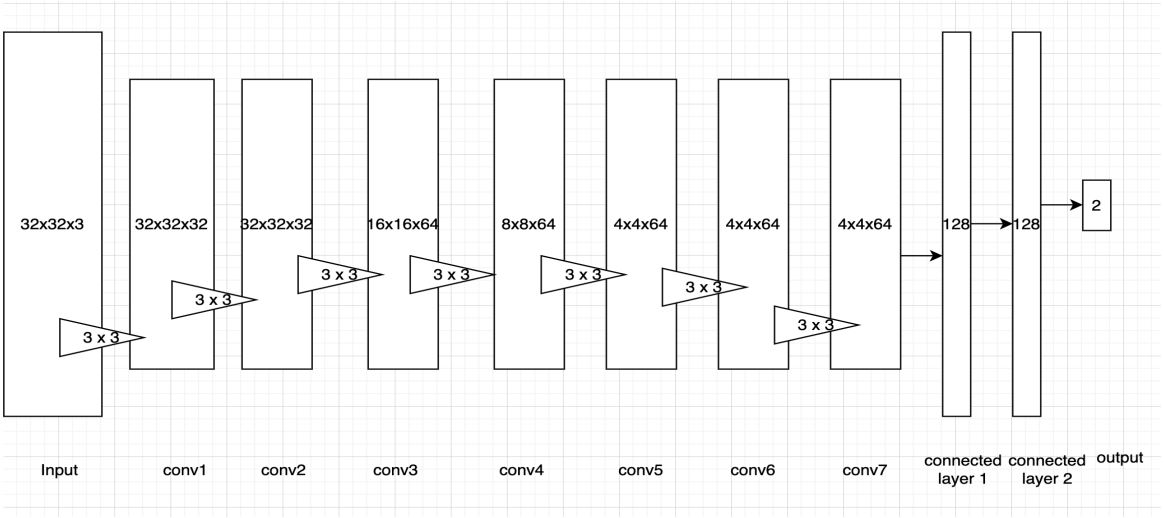


Figure 7: Deep CNN (7 layer) model used in our model

In the secondary model, 7 convolutional layers and 3 fully connected layers are used. Extracted key images with size of 128 X 128 resized to 32 X 32 and are passed as input to the convolution layer-1 (conv1). In conv1input image is convolved with 32 filters of size 3 X 3. After convolution with non-linear transformation ReLU, batch Normalization, second convolutional layer is included (conv2) with ReLU activation as well with 32 filters of size 3X3 just as in layer-1, followed by batch Normalization, dropout of 0.2% and a max pool layer of 2X2 size with stride as 2X2. Conv1 required 896 parameters. Batch normalization is done with 128 parameters. Conv2 required 9248 parameters. Batch_normalization_1 is done with 128 parameters. After that there were 5 convolutional layers conv3, conv4, conv5, conv6, and conv7. Each of these layers had 64 filters of size 3 X 3 and with ReLU activations. And each of these layers are followed by a batch normalization layer. The conv3 has a maxpool layer after batch normalisation layer. Then there is conv4 followed by batch normalisation, dropout layer with 0.3% and maxpool layers. Then there is conv followed by batch normalisation. Next, there is conv5 followed by batch normalisation, dropout layer with 0.4% and maxpool layers. Convolutional layers from 3 to 7 required

18496,36928,36928, 36928,36928 number of parameters respectively. Batch_normalization_2 , Batch_normalization_3, Batch_normalization_4, Batch_normalization_5, Batch_normalization_6 had 256 each parameters. These are input to 3 fully connected dense layers, 2 with ReLU activation and last one with softmax activation. Each of these had 32895, 16512, and 258 parameters. Proposed CNN model required 226,786 trainable parameters.

The output of classifier is two state, so output layer having only two outputs. Adam method is used for Optimization. Here softmax classifier is used for classification. In our proposed CNN framework, the 256 outputs to the fully connected layers are the deep features retrieved from input eye images. The final 2 outputs from fully connected layer 3 with softmax can be the linear combinations of the deep features.

Implementation code snippets

Importing project dependencies

```
import numpy as np
import cv2
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Dropout, Flatten, Dense, MaxPool2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.optimizers import Adam, SGD
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from tensorflow.keras.regularizers import l2
import tensorflow as tf
import pandas as pd
from imutils.face_utils import FaceAligner
from imutils.face_utils import rect_to_bb
import imutils
import dlib
```

Dataset loading

```
!wget https://cainvas-static.s3.amazonaws.com/media/user_data/Yuvnish17/driver_drowsiness_detection_modified.zip
!unzip -qo driver_drowsiness_detection_modified.zip
```

```
--2024-04-27 06:05:19--  https://cainvas-static.s3.amazonaws.com/media/user_data/Yuvnish17/driver_drowsiness_detection_modified.zip
Resolving cainvas-static.s3.amazonaws.com (cainvas-static.s3.amazonaws.com)... 16.12.40.79, 16.12.40.71, 16.12.40.23, ...
Connecting to cainvas-static.s3.amazonaws.com (cainvas-static.s3.amazonaws.com)|16.12.40.79|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 127436131 (122M) [application/x-zip-compressed]
Saving to: 'driver_drowsiness_detection_modified.zip'

driver_drowsiness_d 100%[===================>] 121.53M  12.5MB/s    in 11s

2024-04-27 06:05:32 (10.7 MB/s) - 'driver_drowsiness_detection_modified.zip' saved [127436131/127436131]
```

Image resizing

```
[ ] data = np.load('driver_drowsiness_detection/dataset_compressed.npz', allow_pickle=True)
    X = data['arr_0']
    Y = data['arr_1']

    X = list(X)
    Y = list(Y)
    print(len(X))
    print(len(Y))
```

```
1452
1452
```

```
[ ] def rgb_to_grayscale(image):
        # Convert image to grayscale
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Use BGR2GRAY since OpenCV loads images in BGR format
        # Add a new axis to make it 3-dimensional
        gray_image = np.expand_dims(gray_image, axis=-1)
        return gray_image
```

```
[ ]
    for i in range(len(X)):
        img = X[i]
        img = rgb_to_grayscale(img)
        X[i]=img
```

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive')

    import os
    os.chdir("/content/gdrive/MyDrive/train2")

    os.getcwd()
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
'/content/gdrive/MyDrive/train2'
```

```
def read_images_from_folder(folder_path, target_size=(128, 128)):
    images = []
    for filename in os.listdir(folder_path):
        # Check if the file is an image
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Read the image using OpenCV in grayscale mode
            image = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_GRAYSCALE)
            # Resize the image to the target size
            image_resized = cv2.resize(image, target_size)
            # Add a channel dimension to make it 3D
            image_reshaped = np.expand_dims(image_resized, axis=-1)
            # Append the image to the list
            images.append(image_reshaped)
    # Convert the list of images to a numpy array
    images_array = np.array(images)
    return images_array

# Specify the folder containing the images
folder_path = "/content/gdrive/MyDrive/train2/closed"

# Read images from the folder
images_array = read_images_from_folder(folder_path)
X_2=images_array
# Check the shape of the resulting array
```

+ Code + Text

```
[ ]
    def read_images_from_folder(folder_path, target_size=(128, 128)):
        images = []
        for filename in os.listdir(folder_path):
            # Check if the file is an image
            if filename.endswith(".jpg") or filename.endswith(".png"):
                # Read the image using OpenCV in grayscale mode
                image = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_GRAYSCALE)
                # Resize the image to the target size
                image_resized = cv2.resize(image, target_size)
                # Add a channel dimension to make it 3D
                image_reshaped = np.expand_dims(image_resized, axis=-1)
                # Append the image to the list
                images.append(image_reshaped)
        # Convert the list of images to a numpy array
        images_array = np.array(images)
        return images_array

    # Specify the folder containing the images
    folder_path = "/content/gdrive/MyDrive/train2/open"

    # Read images from the folder
    images_array = read_images_from_folder(folder_path)
    X_3=images_array
    # Check the shape of the resulting array
```

```
[ ] X_3.shape
```

```
(700, 128, 128, 1)
```

```
# Create two sample arrays
array1 = X_2
array2 = X_3

# Create a new array with the desired shape
new_shape = (array1.shape[0] + array2.shape[0], array1.shape[1], array1.shape[2], array1.shape[3])
X_new = np.zeros(new_shape)

# Copy contents of array1 and array2 into result_array
X_new[:array1.shape[0], :, :, :] = array1
X_new[array1.shape[0]:, :, :, :] = array2

print("Resulting array shape:", X_new.shape)
```

```
Resulting array shape: (1400, 128, 128, 1)
```

```
[ ] label_encoder = LabelEncoder()
    Y = label_encoder.fit_transform(Y)
    print(Y.shape)

    print(set(Y))
```

```
(1452,)
{0, 1}
```

```
[ ] label_encoder.classes_
```

```
array(['Closed', 'Open'], dtype='<U6')
```

```
[ ] X = np.array(X)
    Y = np.array(Y)
    print(X.shape)
    print(Y.shape)
```

```
(1452, 128, 128, 1)
(1452,)
```

```
[ ] import numpy as np

    # Create two sample arrays
    array1 = X
    array2 = X_new

    # Create a new array with the desired shape
    new_shape = (array1.shape[0] + array2.shape[0], array1.shape[1], array1.shape[2], array1.shape[3])
    X_final = np.zeros(new_shape)

    # Copy contents of array1 and array2 into result_array
    X_final[:array1.shape[0], :, :, :] = array1
    X_final[array1.shape[0]:, :, :, :] = array2

    print("Resulting array shape:", X_final.shape)
```

```
Resulting array shape: (2852, 128, 128, 1)
```

```
import numpy as np

# Create an array of shape (700,) filled with zeros
zeros_array = np.zeros((700,), dtype=np.int64)

# Create an array of shape (700,) filled with ones
ones_array = np.ones((700,), dtype=np.int64)

# Concatenate zeros and ones arrays along the first axis
Y_5 = np.concatenate((zeros_array, ones_array))

print("Shape of the result array:", Y_5.shape)
```

```
Shape of the result array: (1400,)
```

```
[ ] Y_final = np.concatenate((Y, Y_5))
    Y_final.shape
```

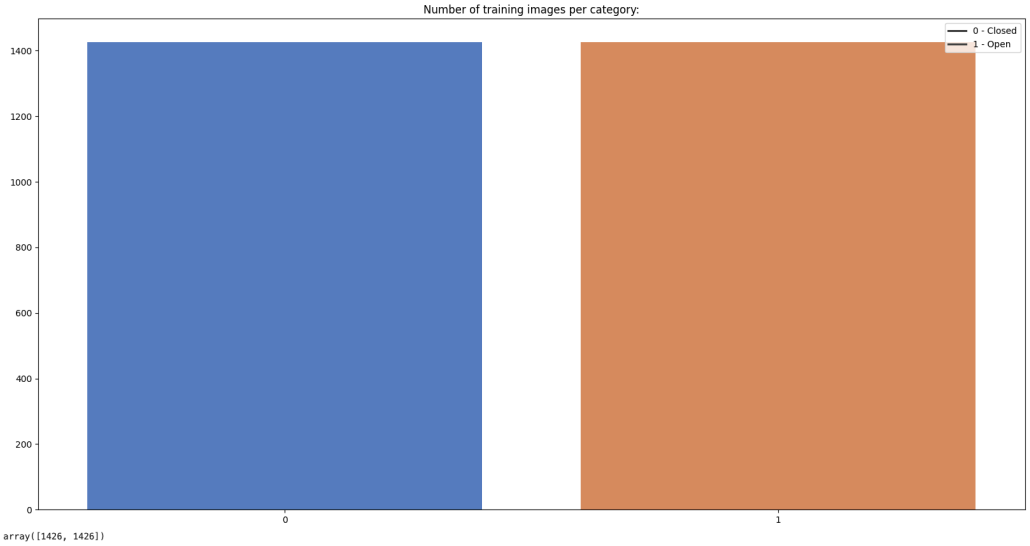
```
(2852,)
```

Data set split

```
[ ] from sklearn.model_selection import train_test_split


    X_train, X_temp, Y_train, Y_temp = train_test_split(X_final, Y_final, test_size=0.40)


    X_validation, X_test, Y_validation, Y_test = train_test_split(X_temp, Y_temp, test_size=0.7)
```

```
# Create a bar plot to visualize the distribution
unique_train, count = np.unique(Y_final, return_counts=True)
plt.figure(figsize=(20, 10))
sns.barplot(x=unique_train, y=count,palette="muted").set_title("Number of training images per category:")
legend_labels = ['0 - Closed', '1 - Open']
plt.legend(legend_labels, loc='upper right')
plt.show()
count
```

```
<ipython-input-19-85539d9fe845>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=unique_train, y=count,palette="muted").set_title("Number of training images per category:")
```

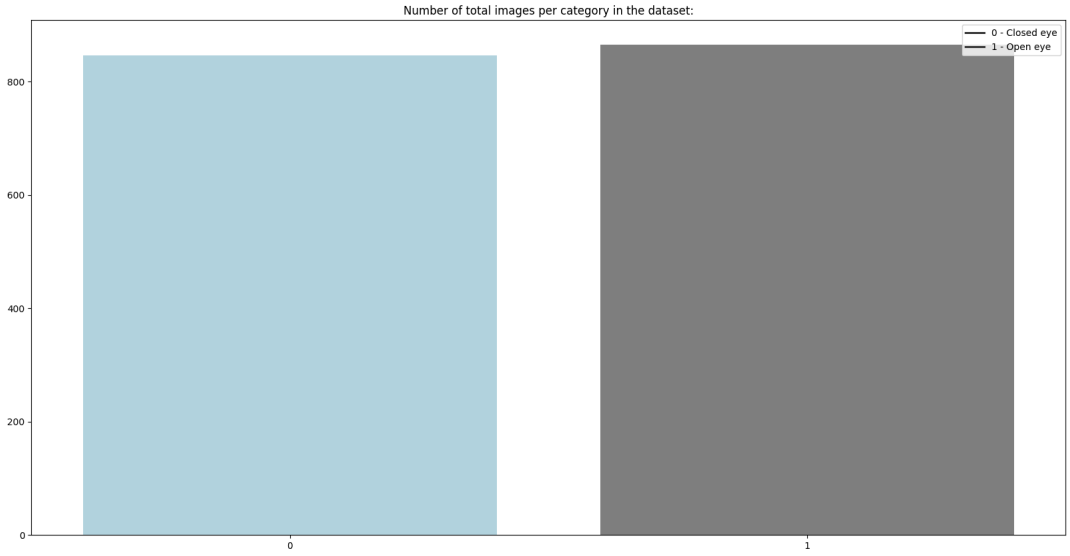



```
array([1426, 1426])
```

Double-click (or enter) to edit

```
unique_test, count_test = np.unique(Y_train, return_counts=True)
plt.figure(figsize=(20, 10))
colors = ["lightblue", "grey"]
sns.barplot(x=unique_test, y=count_test,palette=colors).set_title("Number of total images per category in the dataset:")
legend_labels=['0 - Closed eye', '1 - Open eye']
plt.legend(legend_labels, loc='upper right')
plt.show()
count_test
```

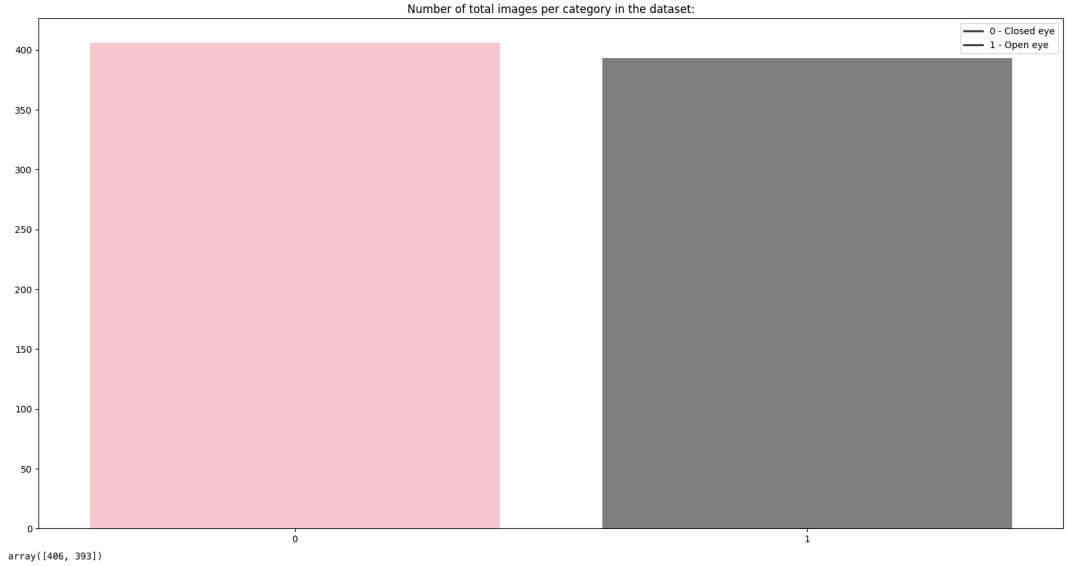
```
<ipython-input-20-fcacfe4fe3ae>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=unique_test, y=count_test,palette=colors).set_title("Number of total images per category in the dataset:")
```




```
array([846, 865])
```

```
unique_test, count_test = np.unique(Y_test, return_counts=True)
plt.figure(figsize=(20, 10))
colors = ["pink", "grey"]
sns.barplot(x=unique_test, y=count_test,palette=colors).set_title("Number of total images per category in the dataset:")
legend_labels=['0 - Closed eye', '1 - Open eye']
plt.legend(legend_labels, loc='upper right')
plt.show()
count_test
```

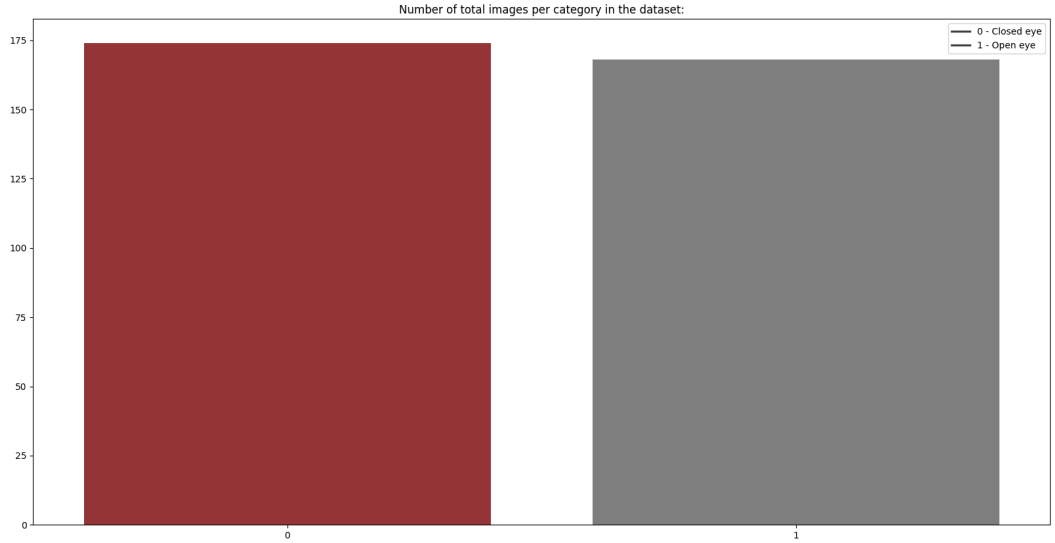
```
<ipython-input-21-3ae133774be8>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=unique_test, y=count_test,palette=colors).set_title("Number of total images per category in the dataset:")
```




```
array([406, 393])
```

```
unique_test, count_test = np.unique(Y_validation, return_counts=True)
plt.figure(figsize=(20, 10))
colors = ["brown", "grey"]
sns.barplot(x=unique_test, y=count_test,palette=colors).set_title("Number of total images per category in the dataset:")
legend_labels=['0 - Closed eye', '1 - Open eye']
plt.legend(legend_labels, loc='upper right')
plt.show()
count_test
```

```
<ipython-input-22-6ed109feb836>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=unique_test, y=count_test,palette=colors).set_title("Number of total images per category in the dataset:")
```




```
array([174, 168])
```

```
print("Train")
print(X_train.shape)
print(Y_train.shape)

print("Test")
print(X_test.shape)
print(Y_test.shape)

print("validate")
print(X_validation.shape)
print(Y_validation.shape)

Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)
Y_validation=to_categorical(Y_validation)

print("Train")
print(X_train.shape)
print(Y_train.shape)

print("Test")
print(X_test.shape)
print(Y_test.shape)

print("validate")
print(X_validation.shape)
print(Y_validation.shape)
```

```
Train
(1711, 128, 128, 1)
(1711,)
Test
(799, 128, 128, 1)
(799,)
validate
(342, 128, 128, 1)
(342,)
Train
(1711, 128, 128, 1)
(1711, 2)
Test
(799, 128, 128, 1)
(799, 2)
validate
(342, 128, 128, 1)
(342, 2)
```

Sample images in each category

```
figure1 = plt.figure(figsize=(5, 5))
idx_closed = np.where(Y_final==0)
img_closed = X[idx_closed[0][0]]
plt.imshow(img_closed)
plt.title('Image of Closed Eye representing Driver is sleeping')
plt.axis('off')
plt.show()

figure2 = plt.figure(figsize=(5, 5))
idx_open = np.where(Y_final==1)
img_open = X[idx_open[0][0]]
plt.imshow(img_open)
plt.title('Image of Open Eye representing Driver is not sleeping')
plt.axis('off')
plt.show()
```

Image of Closed Eye representing Driver is sleeping

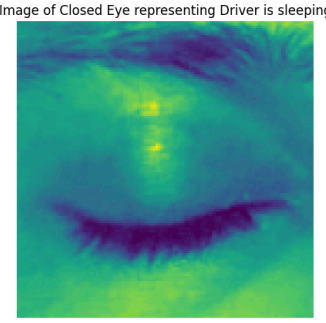
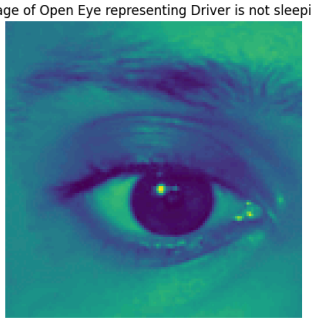


Image of Open Eye representing Driver is not sleeping



Model

```
# Define the CNN model
model = Sequential()

# Convolutional Layer 1
model.add(Conv2D(84, (3, 3), activation='relu', input_shape=(128, 128, 1)))  # Assuming input images are RGB
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2), strides=(2, 2)))
model.add(Dropout(0.25))

# Convolutional Layer 2
model.add(Conv2D(128, (5, 5), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2), strides=(2, 2)))
model.add(Dropout(0.25))

# Convolutional Layer 3
model.add(Conv2D(256, (5, 5), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2), strides=(2, 2)))
model.add(Dropout(0.25))

# Convolutional Layer 4
model.add(Conv2D(512, (5, 5), activation='relu'))
model.add(BatchNormalization())
# model.add(MaxPool2D((2, 2), strides=(2, 2)))
# model.add(Dropout(0.25))

# Flatten layer to transition from convolutional layers to fully connected layers
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(256, activation='relu'))

# Output Layer
model.add(Dense(2, activation='softmax'))  # Assuming two output classes (binary classification)



# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

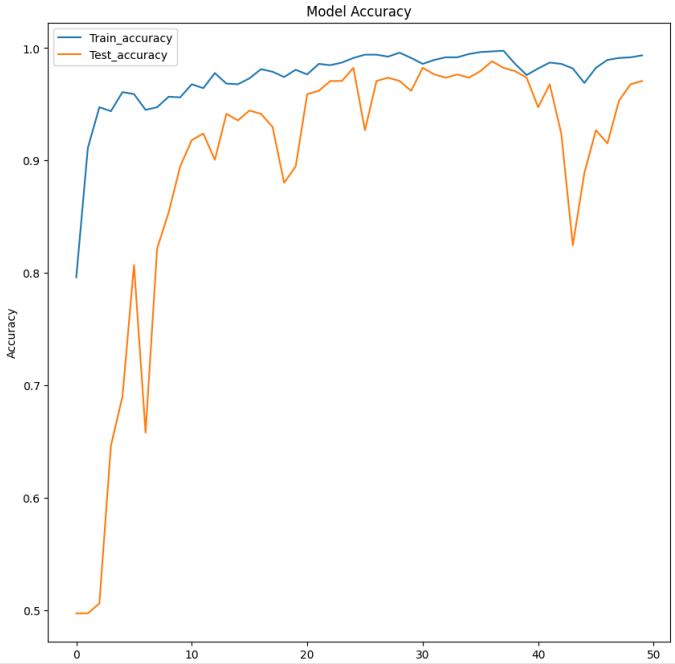
```
model.summary()
```

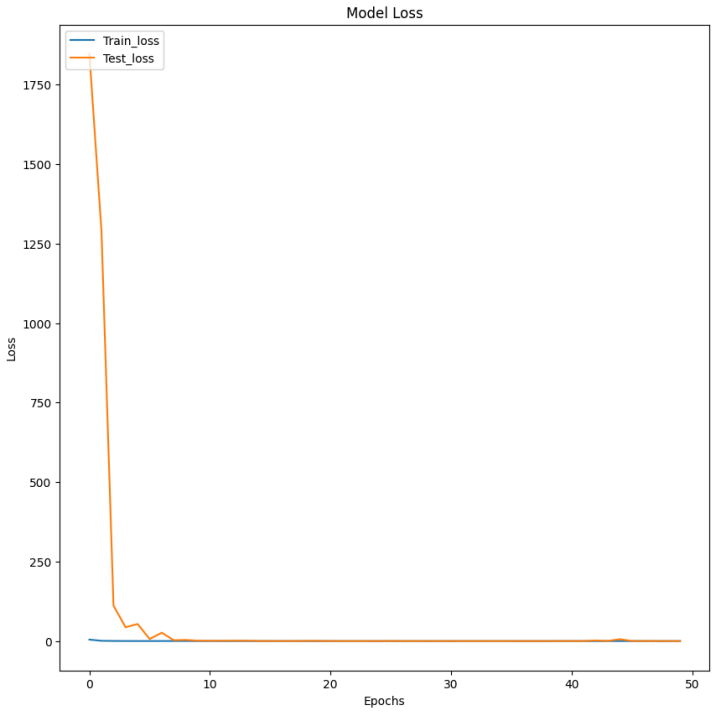
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 126, 126, 84)      840

 batch_normalization_8 (Bat  (None, 126, 126, 84)      336
 chNormalization)

 max_pooling2d_6 (MaxPoolin  (None, 63, 63, 84)        0
 g2D)

 dropout_6 (Dropout)         (None, 63, 63, 84)        0

 conv2d_9 (Conv2D)           (None, 59, 59, 128)       268928

 batch_normalization_9 (Bat  (None, 59, 59, 128)       512
 chNormalization)

 max_pooling2d_7 (MaxPoolin  (None, 29, 29, 128)       0
 g2D)

 dropout_7 (Dropout)         (None, 29, 29, 128)       0

 conv2d_10 (Conv2D)          (None, 25, 25, 256)       819456

 batch_normalization_10 (Ba  (None, 25, 25, 256)       1024
 tchNormalization)

 max_pooling2d_8 (MaxPoolin  (None, 12, 12, 256)       0
 g2D)

 dropout_8 (Dropout)         (None, 12, 12, 256)       0

 conv2d_11 (Conv2D)          (None, 8, 8, 512)         3277312

 batch_normalization_11 (Ba  (None, 8, 8, 512)         2048
 tchNormalization)

 flatten_2 (Flatten)         (None, 32768)             0

 dense_4 (Dense)             (None, 256)               8388864

 dense_5 (Dense)             (None, 2)                 514

=================================================================
Total params: 12759834 (48.67 MB)
Trainable params: 12757874 (48.67 MB)
Non-trainable params: 1960 (7.66 KB)
_________________________________________________________________
```

```
aug = ImageDataGenerator(rotation_range=20, zoom_range=0.2, horizontal_flip=True)
hist = model.fit(aug.flow(X_train, Y_train, batch_size=128), batch_size=128, epochs=50, validation_data=(X_validation, Y_validation))
```

```
Epoch 22/50
14/14 [==============================] - 5s 370ms/step - loss: 0.0356 - accuracy: 0.9860 - val_loss: 0.0629 - val_accuracy: 0.9620
Epoch 23/50
14/14 [==============================] - 5s 373ms/step - loss: 0.0356 - accuracy: 0.9848 - val_loss: 0.0641 - val_accuracy: 0.9708
Epoch 24/50
14/14 [==============================] - 5s 380ms/step - loss: 0.0357 - accuracy: 0.9871 - val_loss: 0.0646 - val_accuracy: 0.9708
Epoch 25/50
14/14 [==============================] - 6s 388ms/step - loss: 0.0235 - accuracy: 0.9912 - val_loss: 0.0370 - val_accuracy: 0.9825
Epoch 26/50
14/14 [==============================] - 5s 368ms/step - loss: 0.0256 - accuracy: 0.9942 - val_loss: 0.3450 - val_accuracy: 0.9269
Epoch 27/50
14/14 [==============================] - 5s 381ms/step - loss: 0.0203 - accuracy: 0.9942 - val_loss: 0.0685 - val_accuracy: 0.9708
Epoch 28/50
14/14 [==============================] - 5s 366ms/step - loss: 0.0239 - accuracy: 0.9924 - val_loss: 0.0570 - val_accuracy: 0.9737
Epoch 29/50
14/14 [==============================] - 5s 374ms/step - loss: 0.0278 - accuracy: 0.9959 - val_loss: 0.0913 - val_accuracy: 0.9708
Epoch 30/50
14/14 [==============================] - 5s 366ms/step - loss: 0.0323 - accuracy: 0.9912 - val_loss: 0.0628 - val_accuracy: 0.9620
Epoch 31/50
14/14 [==============================] - 5s 380ms/step - loss: 0.0378 - accuracy: 0.9860 - val_loss: 0.0404 - val_accuracy: 0.9825
Epoch 32/50
14/14 [==============================] - 5s 368ms/step - loss: 0.0284 - accuracy: 0.9895 - val_loss: 0.0719 - val_accuracy: 0.9766
Epoch 33/50
14/14 [==============================] - 5s 366ms/step - loss: 0.0178 - accuracy: 0.9918 - val_loss: 0.0753 - val_accuracy: 0.9737
Epoch 34/50
14/14 [==============================] - 5s 372ms/step - loss: 0.0171 - accuracy: 0.9918 - val_loss: 0.0533 - val_accuracy: 0.9766
Epoch 35/50
14/14 [==============================] - 5s 367ms/step - loss: 0.0192 - accuracy: 0.9947 - val_loss: 0.0698 - val_accuracy: 0.9737
Epoch 36/50
14/14 [==============================] - 6s 385ms/step - loss: 0.0116 - accuracy: 0.9965 - val_loss: 0.0550 - val_accuracy: 0.9795
Epoch 37/50
14/14 [==============================] - 5s 368ms/step - loss: 0.0137 - accuracy: 0.9971 - val_loss: 0.0504 - val_accuracy: 0.9883
Epoch 38/50
14/14 [==============================] - 5s 371ms/step - loss: 0.0082 - accuracy: 0.9977 - val_loss: 0.0485 - val_accuracy: 0.9825
Epoch 39/50
14/14 [==============================] - 6s 387ms/step - loss: 0.0500 - accuracy: 0.9860 - val_loss: 0.0439 - val_accuracy: 0.9795
Epoch 40/50
14/14 [==============================] - 5s 366ms/step - loss: 0.1068 - accuracy: 0.9760 - val_loss: 0.1023 - val_accuracy: 0.9737
Epoch 41/50
14/14 [==============================] - 6s 384ms/step - loss: 0.0920 - accuracy: 0.9819 - val_loss: 0.1420 - val_accuracy: 0.9474
Epoch 42/50
14/14 [==============================] - 5s 366ms/step - loss: 0.0399 - accuracy: 0.9871 - val_loss: 0.0967 - val_accuracy: 0.9678
Epoch 43/50
14/14 [==============================] - 5s 368ms/step - loss: 0.0492 - accuracy: 0.9860 - val_loss: 1.9291 - val_accuracy: 0.9240
Epoch 44/50
14/14 [==============================] - 6s 392ms/step - loss: 0.1468 - accuracy: 0.9819 - val_loss: 0.4096 - val_accuracy: 0.8246
Epoch 45/50
14/14 [==============================] - 5s 366ms/step - loss: 0.1136 - accuracy: 0.9690 - val_loss: 5.9746 - val_accuracy: 0.8889
Epoch 46/50
14/14 [==============================] - 5s 381ms/step - loss: 0.0594 - accuracy: 0.9825 - val_loss: 0.3993 - val_accuracy: 0.9269
Epoch 47/50
14/14 [==============================] - 5s 396ms/step - loss: 0.0350 - accuracy: 0.9895 - val_loss: 0.2949 - val_accuracy: 0.9152
Epoch 48/50
14/14 [==============================] - 5s 372ms/step - loss: 0.0293 - accuracy: 0.9912 - val_loss: 0.1400 - val_accuracy: 0.9532
Epoch 49/50
14/14 [==============================] - 5s 365ms/step - loss: 0.0197 - accuracy: 0.9918 - val_loss: 0.0796 - val_accuracy: 0.9678
Epoch 50/50
14/14 [==============================] - 5s 368ms/step - loss: 0.0162 - accuracy: 0.9936 - val_loss: 0.0790 - val_accuracy: 0.9708
```

```
figure = plt.figure(figsize=(10, 10))
plt.plot(hist.history['accuracy'], label='Train_accuracy')
plt.plot(hist.history['val_accuracy'], label='Test_accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc="upper left")
plt.show()

figure2 = plt.figure(figsize=(10, 10))
plt.plot(hist.history['loss'], label='Train_loss')
plt.plot(hist.history['val_loss'], label='Test_loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc="upper left")
plt.show()
```





Performance evaluation

```
[ ] pred = model.evaluate(X_test, Y_test)
    print(f'Test Set Accuracy: {pred[1]}')
    print(f'Test Set Loss: {pred[0]}')
```

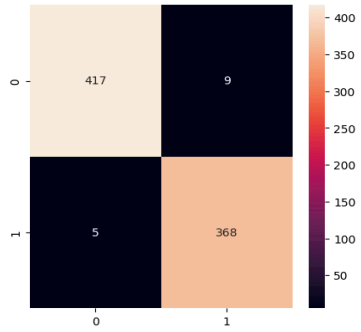
```
25/25 [==============================] - 1s 19ms/step - loss: 0.2615 - accuracy: 0.9825
Test Set Accuracy: 0.9824780821800232
Test Set Loss: 0.2615313231945038
```

```
[ ] ypred = model.predict(X_test)
    ypred = np.argmax(ypred, axis=1)
    Y_test_pred = np.argmax(Y_test, axis=1)
    print(classification_report(Y_test_pred, ypred))
```

```
25/25 [==============================] - 1s 18ms/step
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       426
           1       0.98      0.99      0.98       373

    accuracy                           0.98       799
   macro avg       0.98      0.98      0.98       799
weighted avg       0.98      0.98      0.98       799
```

```
matrix = confusion_matrix(Y_test_pred, ypred)
df_cm = pd.DataFrame(matrix, index=[0, 1], columns=[0, 1])
figure = plt.figure(figsize=(5, 5))
sns.heatmap(df_cm, annot=True, fmt='d')
```

```
<Axes: >
```

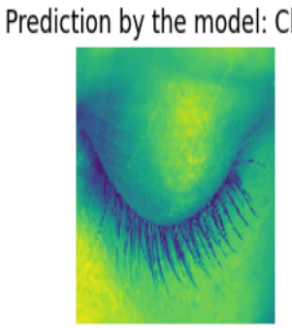



```
[ ] labels = ['Closed', 'Open']
    img_closed2 = cv2.imread('/content/driver_drowsiness_detection/closed_eye2.jpg')
    img_open1 = cv2.imread('/content/driver_drowsiness_detection/open_eye.jpg')
    img_open2 = cv2.imread('/content/driver_drowsiness_detection/open_eye2.jpg')


    img_closed2 = cv2.resize(img_closed2, (128, 128))
    img_open1 = cv2.resize(img_open1, (128, 128))
    img_open2 = cv2.resize(img_open2, (128, 128))



    img_closed2 = rgb_to_grayscale(img_closed2)
    img_open1   = rgb_to_grayscale(img_open1)
    img_open2   = rgb_to_grayscale(img_open2)


    img_closed2 = np.array(img_closed2)
    img_open1 = np.array(img_open1)
    img_open2 = np.array(img_open2)


    img_closed2 = np.expand_dims(img_closed2, axis=0)
    img_open1 = np.expand_dims(img_open1, axis=0)
    img_open2 = np.expand_dims(img_open2, axis=0)
```

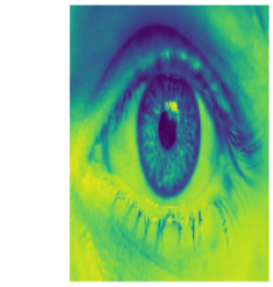
```
[ ]
    ypred_closed2= model.predict(img_closed2)
    ypred_open1=model.predict(img_open1)
    ypred_open2=model.predict(img_open2)
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 31ms/step
```

```
[ ] figure = plt.figure(figsize=(2, 2))
    img_closed1 = np.squeeze(img_closed1, axis=0)
    plt.imshow(img_closed2)
    plt.axis('off')
    plt.title(f'Prediction by the model: {labels[np.argmax(ypred_closed1[0], axis=0)]}')
    plt.show()
```




```
figure = plt.figure(figsize=(2, 2))
img_open1 = np.squeeze(img_open1, axis=0)
plt.imshow(img_open1)
plt.axis('off')
plt.title(f'Prediction by the model: {labels[np.argmax(ypred_open1[0], axis=0)]}')
plt.show()
```



Testing results

Experiment (1-5) were done using our secondary architecture model to test how epochs, batch sizes, validation datasets affect the accuracy of the models.

Experiment 1 (7-layer CNN):

Results when tested with epoch size - 10 and Batch size - 128.

Performance evaluation

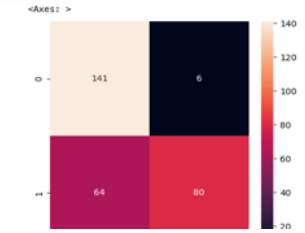
```
pred = model.evaluate(X_test, Y_test)
print(f'Test Set Accuracy: {pred[1]}')
print(f'Test Set Loss: {pred[0]}')
```

```
10/10 [==============================] - 0s 36ms/step - loss: 0.5894 - accuracy: 0.7595
Test Set Accuracy: 0.7594501972198486
Test Set Loss: 0.5894344449043274
```

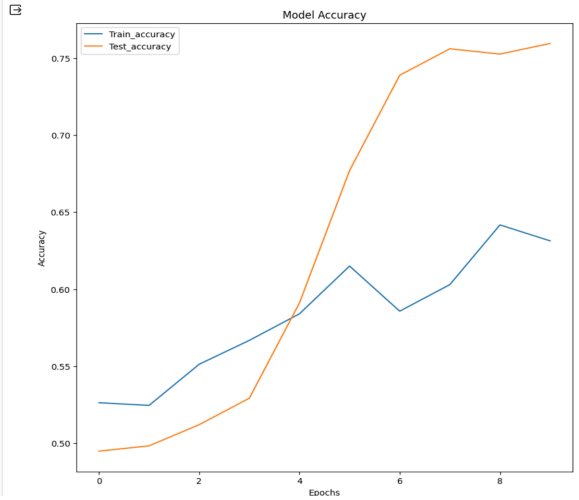
```
ypred = model.predict(X_test)
ypred = np.argmax(ypred, axis=1)
Y_test_pred = np.argmax(Y_test, axis=1)
print(classification_report(Y_test_pred, ypred))
```

```
10/10 [==============================] - 1s 32ms/step
              precision    recall  f1-score   support

           0       0.69      0.96      0.80       147
           1       0.93      0.56      0.70       144

    accuracy                           0.76       291
   macro avg       0.81      0.76      0.75       291
weighted avg       0.81      0.76      0.75       291
```

```
matrix = confusion_matrix(Y_test_pred, ypred)
df_cm = pd.DataFrame(matrix, index=[0, 1], columns=[0, 1])
figure = plt.figure(figsize=(5, 5))
sns.heatmap(df_cm, annot=True, fmt='d')
```

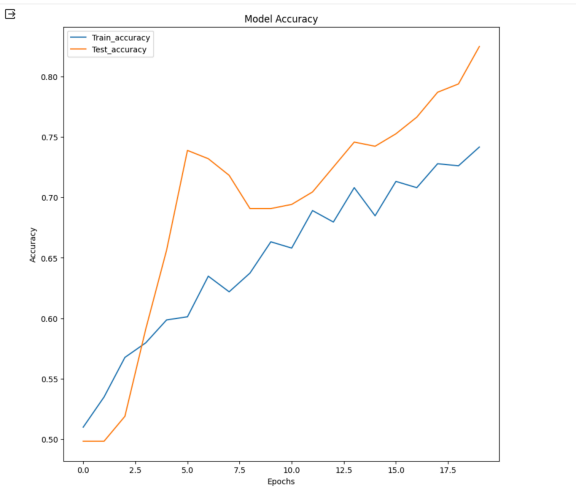


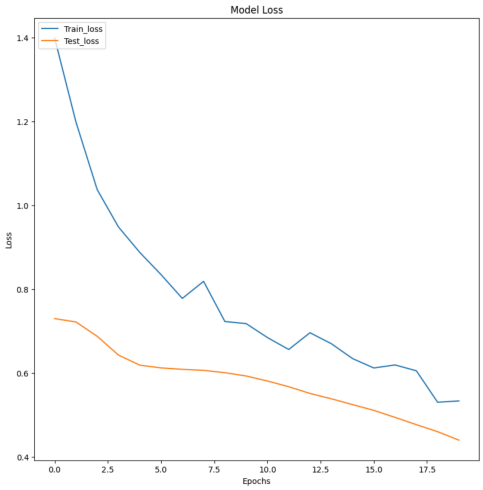

```
figure2 = plt.figure(figsize=(10, 10))
plt.plot(hist.history['loss'], label='Train_loss')
plt.plot(hist.history['val_loss'], label='Test_loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc="upper left")
plt.show()
```



Experiment 2 (7-layer CNN):

Results when tested with epoch size – 20 and batch size - 128:

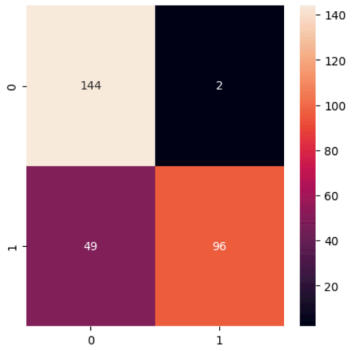





```
10/10 [==============================] - 0s 36ms/step
              precision    recall  f1-score   support

           0       0.75      0.99      0.85       146
           1       0.98      0.66      0.79       145

    accuracy                           0.82       291
   macro avg       0.86      0.82      0.82       291
weighted avg       0.86      0.82      0.82       291
```

```
matrix = confusion_matrix(Y_test_pred, ypred)
df_cm = pd.DataFrame(matrix, index=[0, 1], columns=[0, 1])
figure = plt.figure(figsize=(5, 5))
sns.heatmap(df_cm, annot=True, fmt='d')
```

<Axes: >

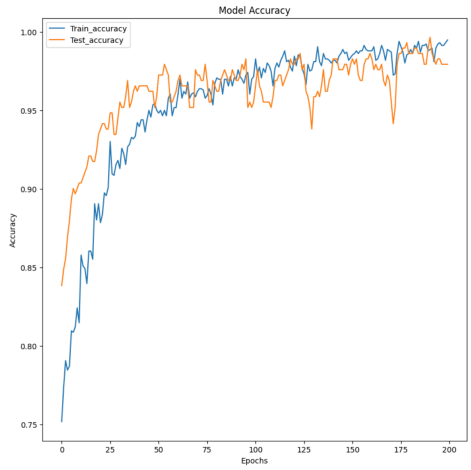


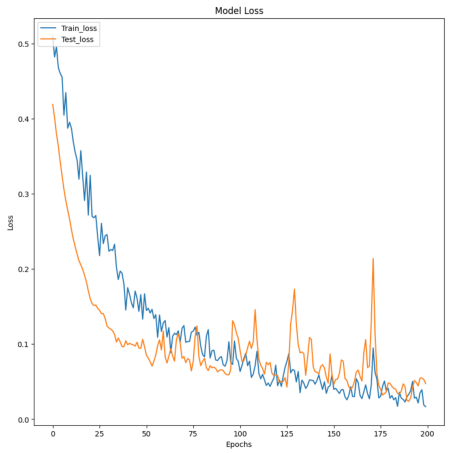
Experiment 3 (7-layer CNN):

Results when tested with epoch size - 200 and batch size - 128:

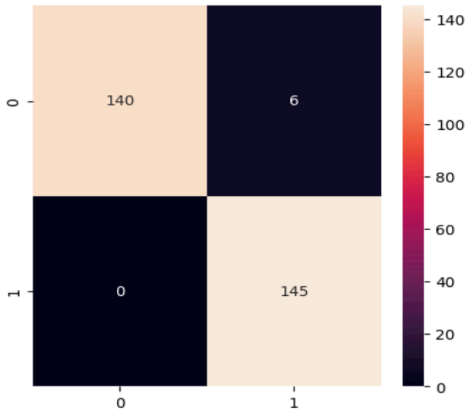
Test Set Accuracy: 0.9793814420700073

Test Set Loss: 0.04740602895617485



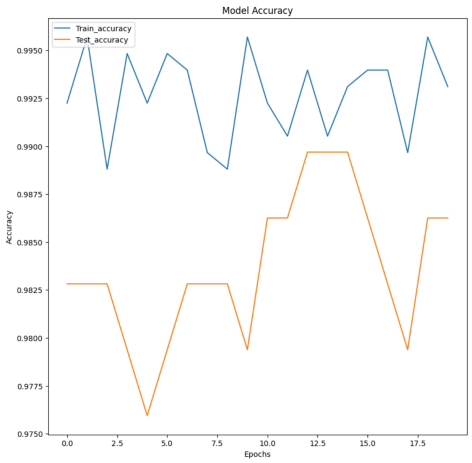


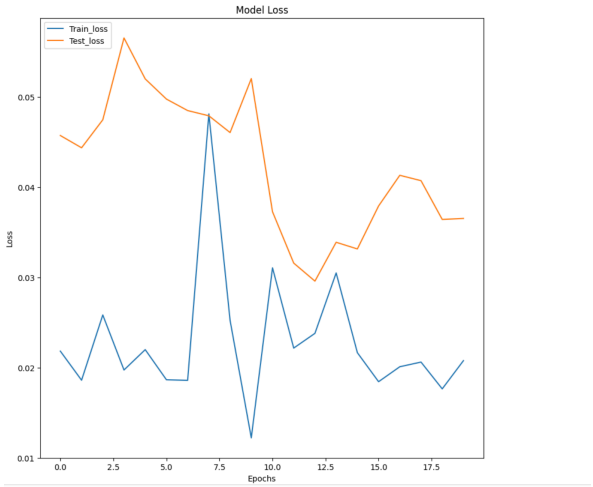

```
10/10 [==============================] - 0s 39ms/step
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       146
           1       0.96      1.00      0.98       145

    accuracy                           0.98       291
   macro avg       0.98      0.98      0.98       291
weighted avg       0.98      0.98      0.98       291
```



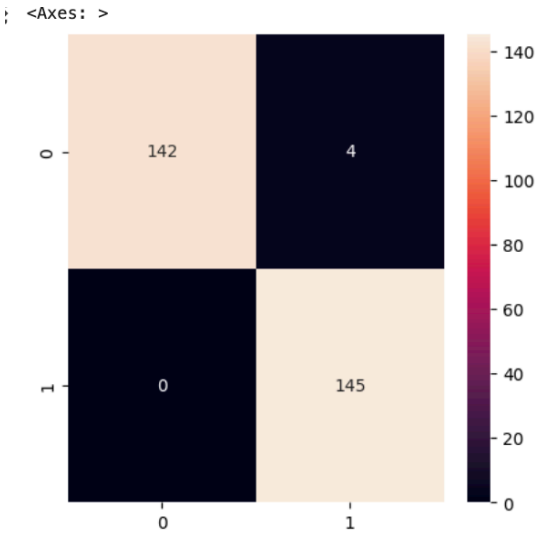
Experiment 4 (7-layer CNN):

Results when tested with epoch size - 200 and batch size - 256:



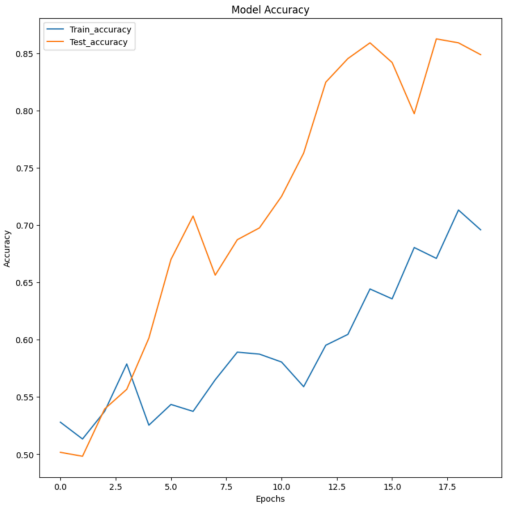


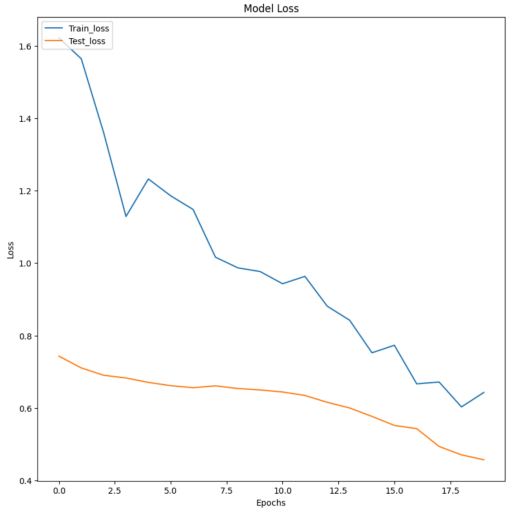

```
10/10 [==============================] - 0s 37ms/step
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       146
           1       0.97      1.00      0.99       145

    accuracy                           0.99       291
   macro avg       0.99      0.99      0.99       291
weighted avg       0.99      0.99      0.99       291
```



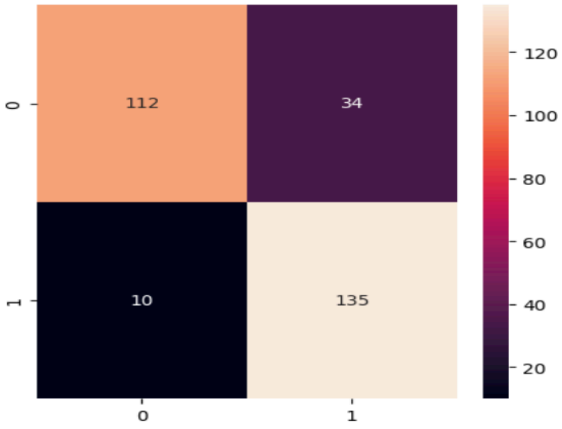
Experiment 5 (7-layer CNN):

Results when tested with filter size: 10 x 10 (conv1 and conv2), epoch 20, batch size: 256.






```
10/10 [==============================] - 2s 160ms/step
              precision    recall  f1-score   support

           0       0.92      0.77      0.84       146
           1       0.80      0.93      0.86       145

    accuracy                           0.85       291
   macro avg       0.86      0.85      0.85       291
weighted avg       0.86      0.85      0.85       291
```



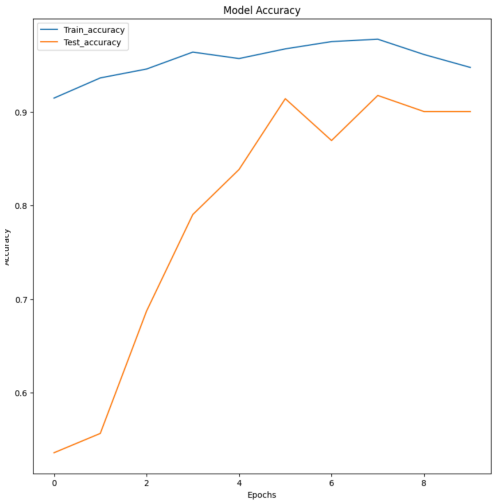
Experiment (6-9) were performed on our primary model architecture (from paper). We tried adjusting epoch, batch size to test the model for 6-8 including the test set as validation set for training (Therefore high accuracy). For the 9th experiment we separated the validation and test set so that the model doesn't see the test set until the testing process.

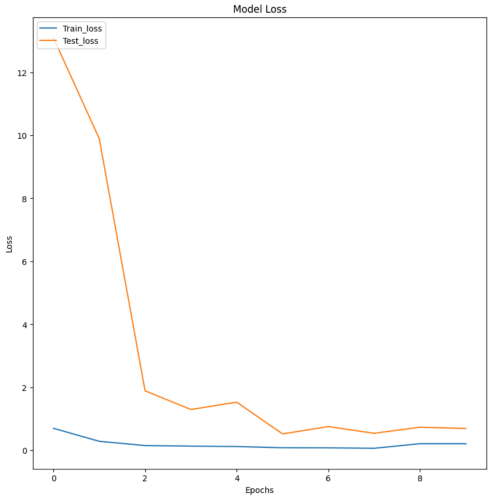
Experiment 6 (Original CNN (4 layer)):

Results when tested with original results from paper with 10 epochs:

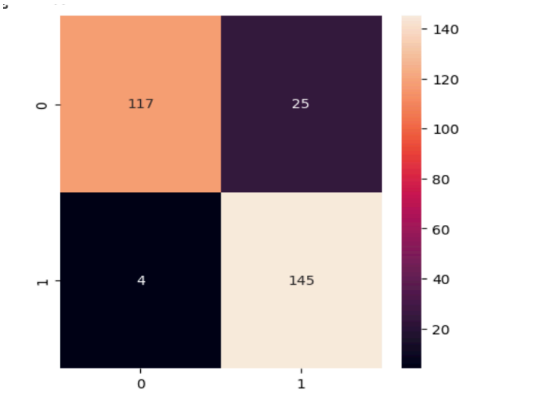
Test Set Accuracy: 0.900343656539917

Test Set Loss: 0.6964724063873291






```
10/10 [==============================] - 37s 4s/step
              precision    recall  f1-score   support

           0       0.97      0.82      0.89       142
           1       0.85      0.97      0.91       149

    accuracy                           0.90       291
   macro avg       0.91      0.90      0.90       291
weighted avg       0.91      0.90      0.90       291
```

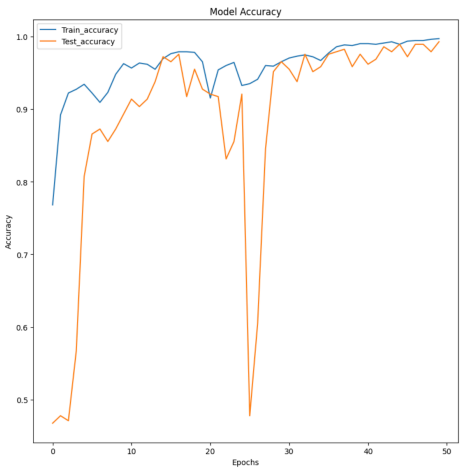


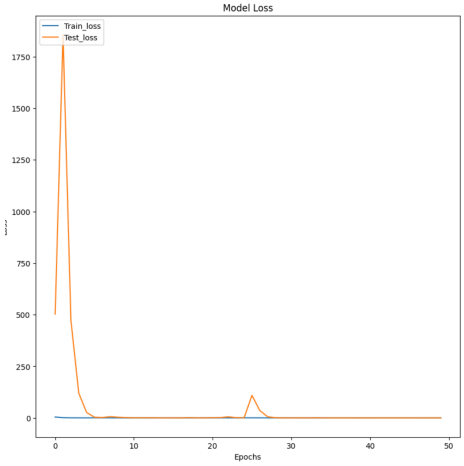
Experiment 7 (Original CNN (4 layer)):

Architecture from paper with 50 epochs:

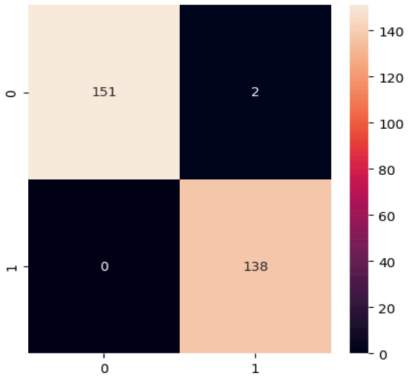
Test Set Accuracy: 0.993127167224884

Test Set Loss: 0.022899432107806206






```
10/10 [==============================] - 0s 14ms/step
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       153
           1       0.99      1.00      0.99       138

    accuracy                           0.99       291
   macro avg       0.99      0.99      0.99       291
weighted avg       0.99      0.99      0.99       291
```



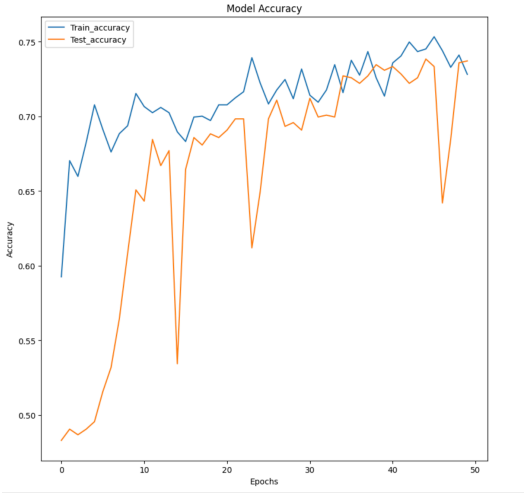
Experiment 8 (Original CNN (4 layer)):

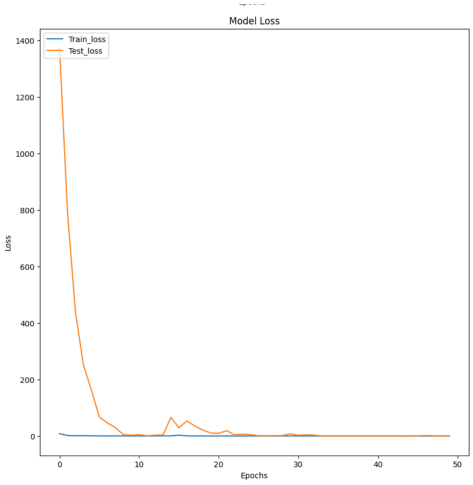
Results when tested with additional data from Kaggle dataset: Dataset Split (80-20)

Test Set Accuracy: 0.737171471118927

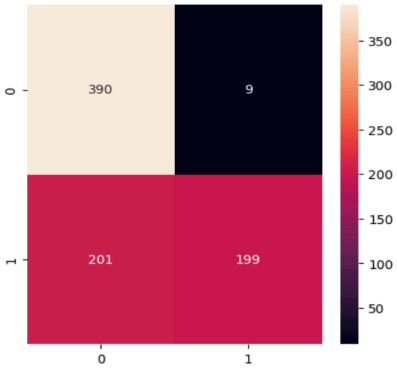
Test Set Loss: 0.4173743724822998

Issue - Overfitting






```
25/25 [==============================] - 1s 17ms/step
              precision    recall  f1-score   support

           0       0.66      0.98      0.79       399
           1       0.96      0.50      0.65       400

    accuracy                           0.74       799
   macro avg       0.81      0.74      0.72       799
weighted avg       0.81      0.74      0.72       799
```




```
14/14 [==============================] - 6s 384ms/step - loss: 0.3726 - accuracy: 0.7487 - val_loss: 0.9059 - val_accuracy: 0.7334
Epoch 238/250
14/14 [==============================] - 6s 401ms/step - loss: 0.4554 - accuracy: 0.7586 - val_loss: 0.8554 - val_accuracy: 0.7171
Epoch 239/250
14/14 [==============================] - 6s 421ms/step - loss: 0.4682 - accuracy: 0.7499 - val_loss: 0.7946 - val_accuracy: 0.7196
Epoch 240/250
14/14 [==============================] - 5s 381ms/step - loss: 0.3660 - accuracy: 0.7522 - val_loss: 0.6632 - val_accuracy: 0.7247
Epoch 241/250
14/14 [==============================] - 5s 378ms/step - loss: 0.3720 - accuracy: 0.7534 - val_loss: 0.3842 - val_accuracy: 0.7334
Epoch 242/250
14/14 [==============================] - 6s 407ms/step - loss: 0.3669 - accuracy: 0.7516 - val_loss: 0.4052 - val_accuracy: 0.7297
Epoch 243/250
14/14 [==============================] - 6s 378ms/step - loss: 0.3949 - accuracy: 0.7458 - val_loss: 0.8875 - val_accuracy: 0.6145
Epoch 244/250
14/14 [==============================] - 6s 381ms/step - loss: 0.3789 - accuracy: 0.7382 - val_loss: 0.6381 - val_accuracy: 0.6583
Epoch 245/250
14/14 [==============================] - 5s 378ms/step - loss: 0.4179 - accuracy: 0.7428 - val_loss: 0.6872 - val_accuracy: 0.6195
Epoch 246/250
14/14 [==============================] - 6s 412ms/step - loss: 0.5175 - accuracy: 0.7370 - val_loss: 0.5486 - val_accuracy: 0.7284
Epoch 247/250
14/14 [==============================] - 5s 380ms/step - loss: 0.3878 - accuracy: 0.7434 - val_loss: 0.5384 - val_accuracy: 0.7309
Epoch 248/250
14/14 [==============================] - 5s 379ms/step - loss: 0.3666 - accuracy: 0.7499 - val_loss: 0.5284 - val_accuracy: 0.7397
Epoch 249/250
14/14 [==============================] - 6s 402ms/step - loss: 0.3962 - accuracy: 0.7528 - val_loss: 0.4770 - val_accuracy: 0.7359
Epoch 250/250
14/14 [==============================] - 6s 408ms/step - loss: 0.3566 - accuracy: 0.7639 - val_loss: 0.3980 - val_accuracy: 0.7522
```

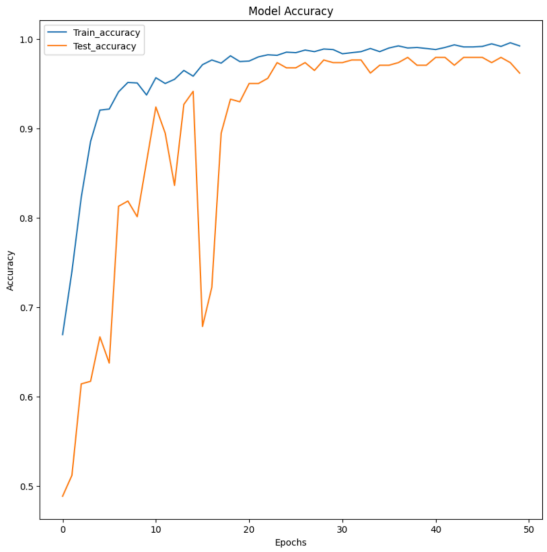
Experiment 9 (Original CNN (4 layer)):

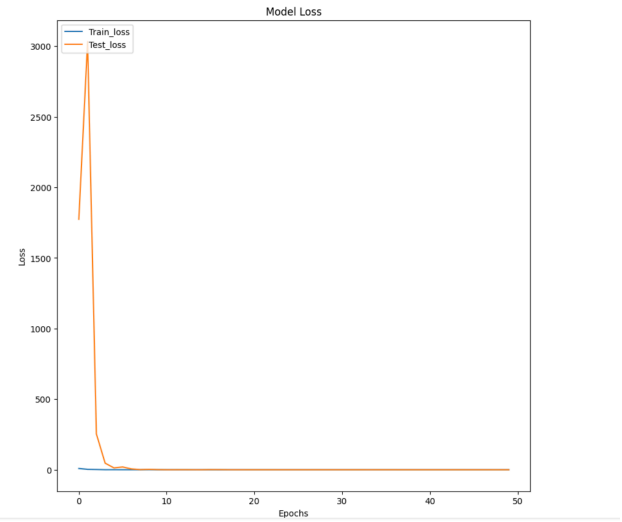
Results when tested with dataset split of (train-test, test-validation) 60:40, 70:30:

Test Set Accuracy: 0.987171471118927

Test Set Loss: 0.0273743724822998

Issue: Overfitting



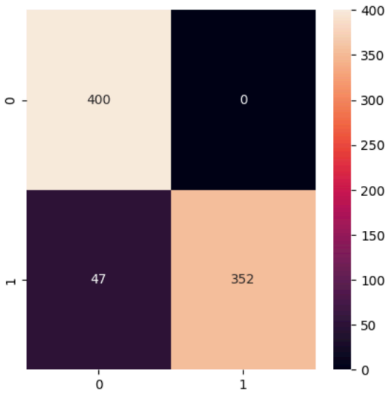



```
ypred = model.predict(X_test)
ypred = np.argmax(ypred, axis=1)
Y_test_pred = np.argmax(Y_test, axis=1)
print(classification_report(Y_test_pred, ypred))
```

```
25/25 [==============================] - 1s 16ms/step
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       400
           1       1.00      0.88      0.94       399

    accuracy                           0.94       799
   macro avg       0.95      0.94      0.94       799
weighted avg       0.95      0.94      0.94       799
```

```
matrix = confusion_matrix(Y_test_pred, ypred)
df_cm = pd.DataFrame(matrix, index=[0, 1], columns=[0, 1])
figure = plt.figure(figsize=(5, 5))
sns.heatmap(df_cm, annot=True, fmt='d')
```

```
<Axes: >
```



For experiment 10, we fixed all the issues we experienced before, primarily the overfitting issue from experiment 8 and 9. Experiment 10 is the best model for the dataset that we have currently.

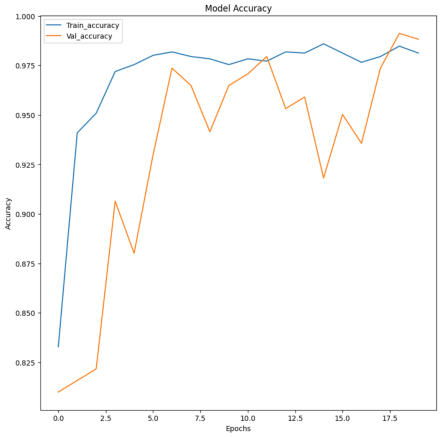
Experiment 10 (Modified 3-layer CNN model):

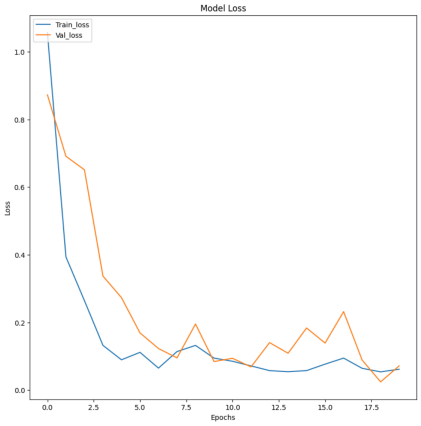
Results when tested with modified architecture, increased batch size, manually adjusted learning rate (0.0001), and increased dropout rates to fix overfitting.

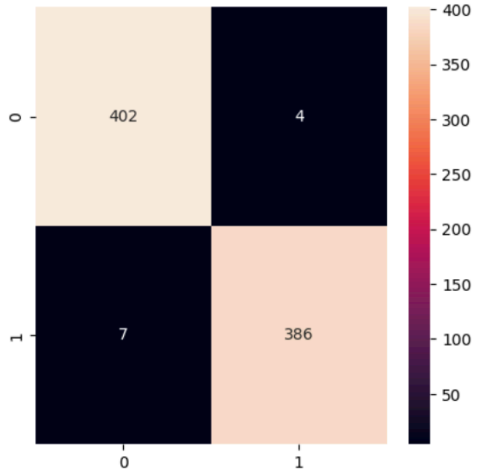
Test Set Accuracy: 0.9862328171730042

Test Set Loss: 0.05531303212046623

Result: Better accuracy with significantly decreased overfitting on the model while maintaining 98% accuracy.








```
25/25 [==============================] - 0s 6ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       406
           1       0.99      0.98      0.99       393

    accuracy                           0.99       799
   macro avg       0.99      0.99      0.99       799
weighted avg       0.99      0.99      0.99       799
```

Test observations:

Increasing accuracy increases the accuracy until a point but beyond it is a bad especially the dataset is small as it may introduce overfitting.
Including a separate validation set improves the model performance on the predicting unseen data well.
If model is overfit, modified architecture to remove few layers, reducing training parameters, increasing data size, adjusting learning rate manually would help in fixing and improving the model performance on unseen data.

Comparison

In this comparison, we highlighted the superior performance of our model over the originally implemented model detailed in the source paper. Our deep CNN achieved a commendable accuracy of 98%, demonstrating a notable improvement. Several factors contribute to this enhancement:

Dataset Quality: Our dataset's superior quality, with high-resolution images and clear distinctions between drowsy and non-drowsy states, allowed for more effective training.

Balanced Data: Unlike the dataset used in the source paper, ours was meticulously balanced between the categories, which helped in reducing bias during the learning process.

Split Configuration: We implemented a 60-40 training-test split with a further division of the test segment into 70-30 for testing and validation, which ensured rigorous testing and more reliable validation of our model's performance.

These factors, combined with the innovative adjustments to the CNN architecture, contributed significantly to the outcomes observed.

Discussion

The project's outcomes illuminate several critical aspects of enhancing machine learning model performance:

Architectural Innovation: Utilizing a deep CNN allowed us to track and analyze multiple parameters simultaneously. This capability is pivotal in complex image recognition tasks such as drowsiness detection, where subtle features like eye closure need to be detected reliably.

Pre-trained Models: The use of open-source, pre-trained models expedited our development process and provided a robust foundation for our customizations. This approach not only saves time but also leverages the community's collective advancements.

Overfitting Issue resolution: Experienced overfitting issue while evaluating our model. So, we tried multiple approaches to improve it while not compromising on the model scores like reducing parameters by simplifying model architecture, manually adjusted learning rate, changing dataset split size, etc.

Team Collaboration: Working as a team, we combined diverse skills and perspectives, which enriched the project's execution and outcome. The collaborative effort in tasks such as data preparation, coding, and iterative testing was instrumental in achieving the results.

This project reinforced the value of a structured and collaborative approach to tackling complex problems with machine learning.

Conclusion

Our experiments with varying epoch sizes indicated a clear trend: larger epoch sizes generally lead to improved accuracy. However, the optimal size and configuration of epochs depend significantly on the specific characteristics of the dataset and the model architecture. Primarily, our experiments also highlighted the issue overfitting and what are some of the things to improve it. Future studies should explore:

Deep vs. Simple CNN: A comparative study between deep and simple CNN architectures will provide insights into the efficiency and efficacy of deeper network layers in drowsiness detection.

Feature Map Analysis: Investigating how changes in the feature map configurations affect the model's accuracy can uncover potential optimizations, enhancing the model's ability to generalize across different datasets and conditions.

In conclusion, our project not only advanced our understanding of deep learning applications in real-world scenarios but also set a foundation for future explorations into more nuanced machine learning strategies in the field of driver safety.

References

(Reddy & Behera, 2022), (Driver Drowsiness Detection using CNN), (Zahra Mardi, 2011)

Bibliography

Abtahi, S., Hariri, B., & Shirmohammadi, S. (2011). Driver drowsiness monitoring based on yawning detection. *IEEE*.

Berghe, W. V. (2021). *European Road Safety Observatory*. Retrieved from SWOV Institute for Road Safety Research (NL): https://road-safety.transport.ec.europa.eu/system/files/2021-07/road_safety_thematic_report_speeding.pdf

Danisman, T., Bilasco, I. M., Djeraba, C., & Ihaddadene, N. (2010). Drowsy driver detection system using eye blink patterns. *IEEE*.

Driver Drowsiness Detection using CNN. (n.d.). *AITS Journal* .

Dwivedi, K., Biswaranjan, K., & Sethi, A. (2014). Drowsy driver detection using representation learning. *IEEE*.

Picot, A., Charbonnier, S., & Caplier, A. (2011). On-Line Detection of Drowsiness Using Brain and Visual Information. *IEEE*.

Reddy, T. K., & Behera, L. (2022). Driver Drowsiness Detection: An Approach Based on Intelligent Brain–Computer Interfaces. *IEEE*.

Sanghyuk Park, F. P. (2016). Driver Drowsiness Detection System Based on Feature Representation Learning Using Various Deep Networks. *Semantic Scholar*.

Seyed Mohammad Reza Noori, M. M. (2023). Driving Drowsiness Detection Using Fusion of Electroencephalography, Electrooculography, and Driving Quality Signals. *semanticscholar*.

Yang, E., & Yi, O. (2024). Enhancing Road Safety: Deep Learning-Based Intelligent Driver Drowsiness Detection for Advanced Driver-Assistance Systems. *MDPI Open Access Journals*.

Zahra Mardi, S. N. (2011). EEG-Based Drowsiness Detection for Safe Driving Using Chaotic Features and Statistical Tests. *Original Article*.