Configuration Manual

MSc Research Project
MSc Data Analytics

Pushkar Dashpute
x18180124

School of Computing
National College of Ireland

Supervisor: Dr. Manaz Kaleel

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pushkar Anil Dashpute
Student ID:	x18180124
Programme:	MSc Data Analytics **Year:** 2019-2020
Module:	MSc Research Project
Lecturer:	Dr. Manaz Kaleel
Submission Due Date:	28th September, 2020
Project Title:	Identifying Driver Distraction Using Deep Neural Networks
Word Count:	919 **Page Count:** 13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Pushkar Anil Dashpute

Date: 27th September, 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pushkar Dashpute
x18180124

1 Introduction

The following configuration manual illustrates the requirements for implementing the system which was designed for detecting the distraction of drivers by using the Deep Learning models. Further, the manual will thoroughly explain the software and hardware requirements that were used for the successful implementation of the project.

2 System Configuration

Following are the hardware and software configuration which were used for the implementation of this Project.

The hardware configurations used for implementation are as follows:

2.1 Hardware Requirements:

Table 1 Hardware Requirements

Hardware	Configurations
System	Lenovo Legion Y740
Operating System	Windows 10 (64bit)
RAM	16 GB
Hard Disk	1 TB (Solid State Drive)
Graphics Card	NVIDIA RTX 2060 (6 GB)
Processor	Intel Core I7-9750

Figure 1 Operating System Configurations

The operating system used for this project was Windows 10 which was 64bit based.

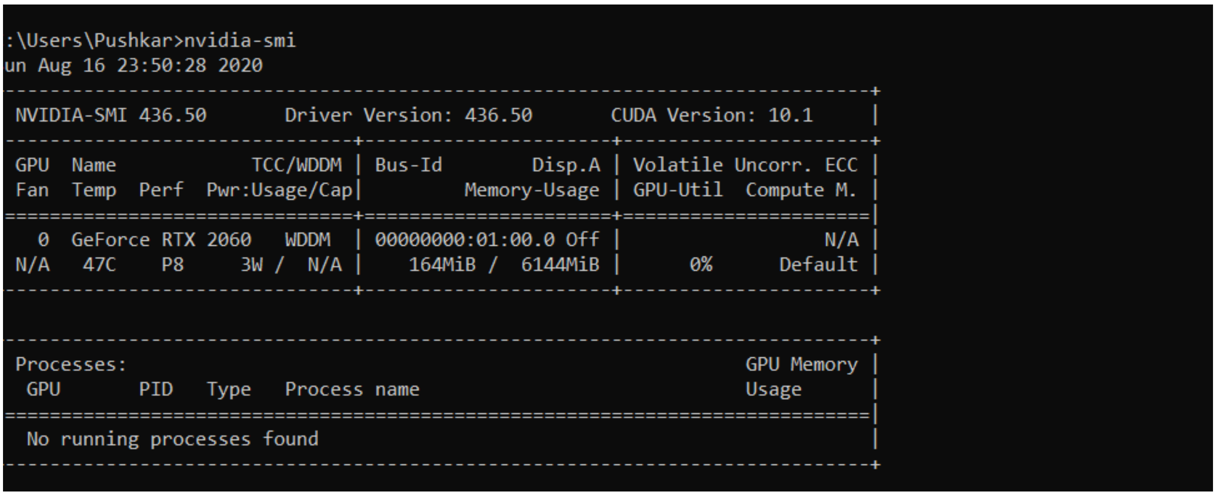

Figure 2 CUDA Version

The CUDA version used was 10.1.

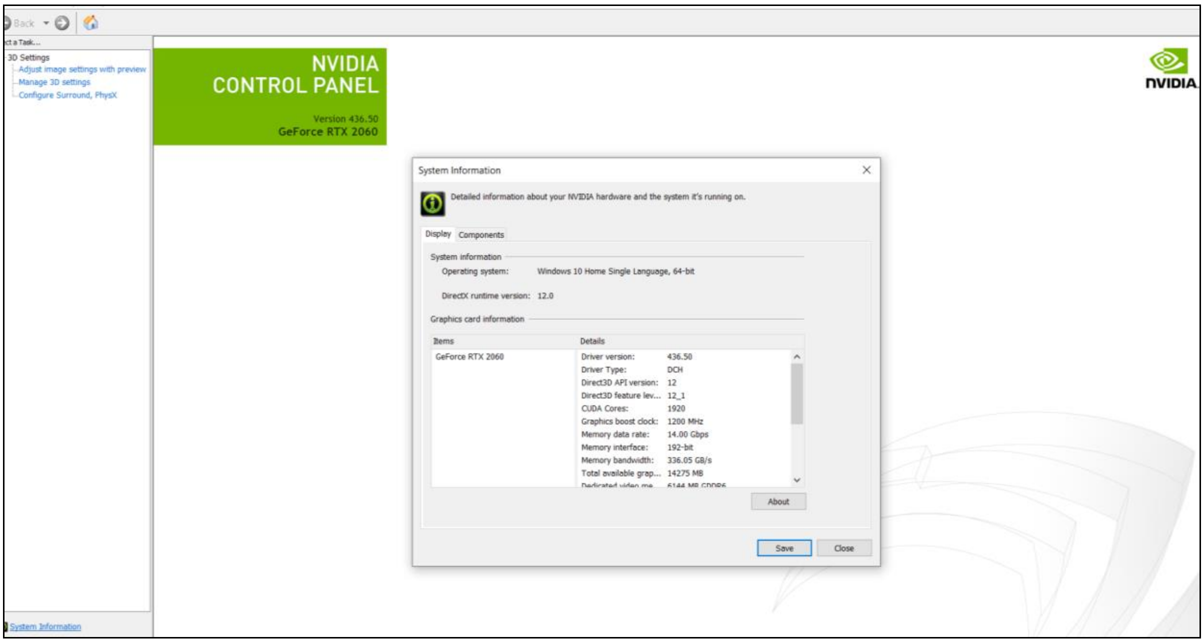

Figure 3 NVIDIA Driver Information

The GPU used for implementing the high-end deep learning models was Nvidia RTX 2060 with a size of 6GB.

2.2 Software Requirements

The software's used were as follows:

Table 2 Software Requirements

Software	Version
Python	3.7 (64 bit)
PyCharm Community	2020.2 (64 bit)
Microsoft Excel	2020 Edition

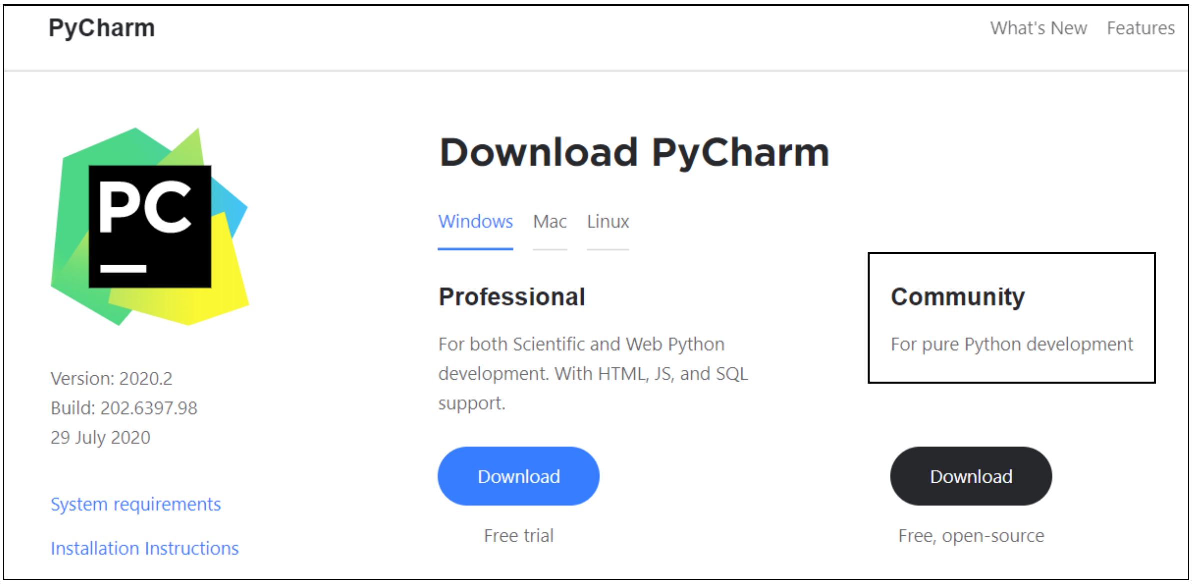

Figure 4 Downloading PyCharm

PyCharm can be downloaded from https://www.jetbrains.com/. There are two versions available namely Professional and Community. We will be using the Community. The IDE used for implementing the whole project was PyCharm. The Community version was used which is pure python-based development. The latest version 2020.2 was used. The steps involved in installing the software will be discussed further.

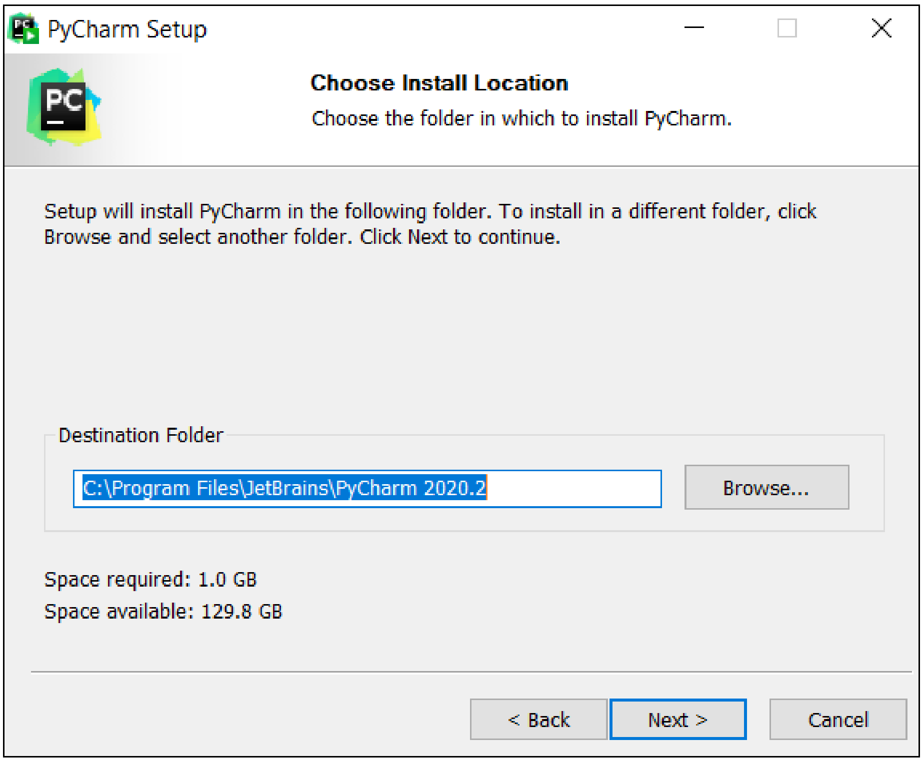

Figure 5 Installation folder for PyCharm

Choose the destination folder where you want to install the software and make sure you have enough space i.e. 1.0 GB required by the software.

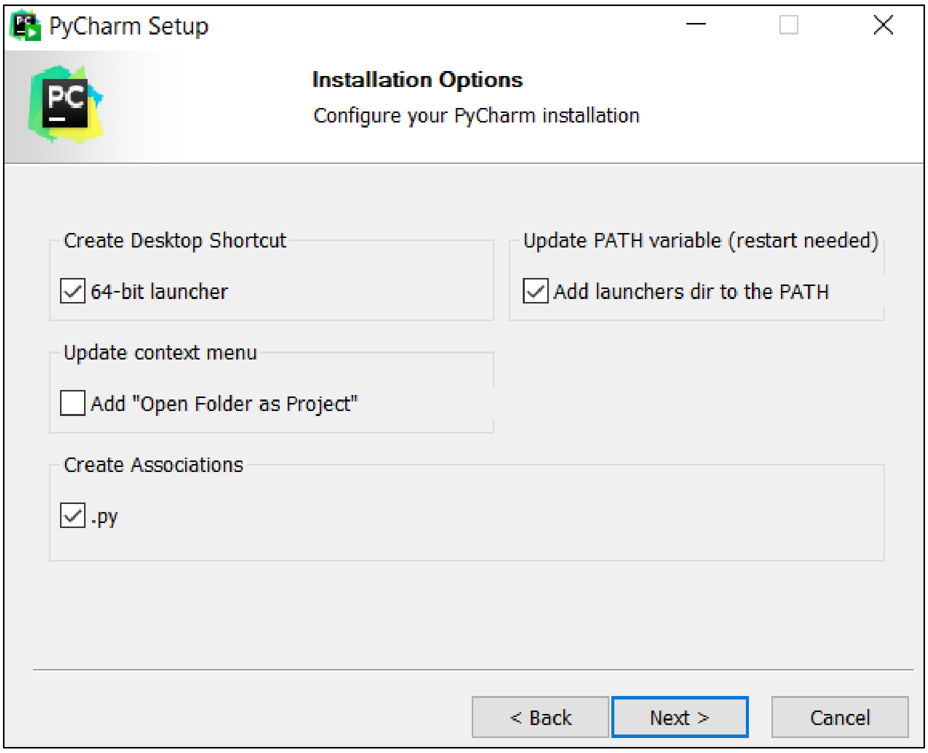

Figure 6 Installing Options

Choose the 64-bit launcher and also create associations with .py files. In addition to this, tick the Update PATH variable which will further add the launcher directory to the PATH.

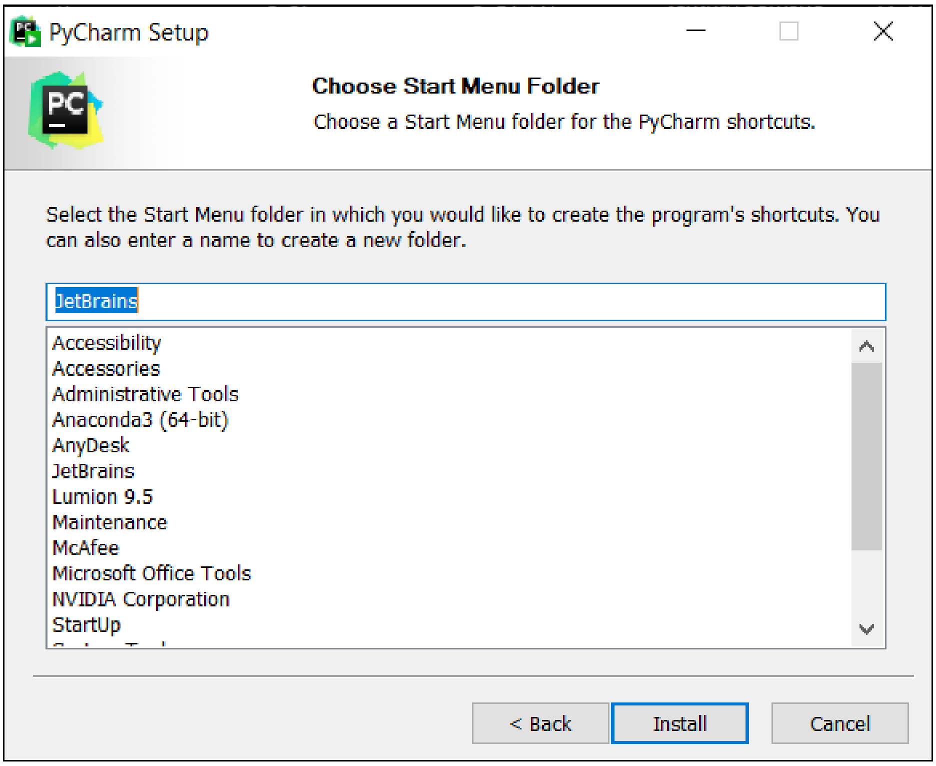

Figure 7 Choosing Start Menu Folder

Choose the folder for Start Menu

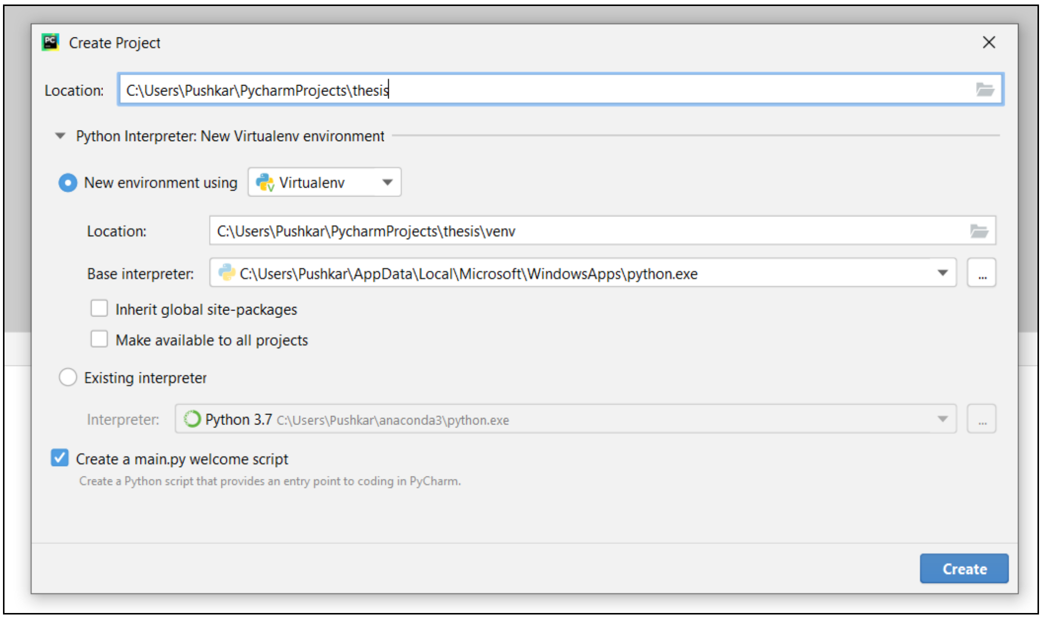

Figure 8 Project creation

The project will be created in this step where the environment will be created for the further implementation of the project. All the required libraries by the project will be stored in this environment.

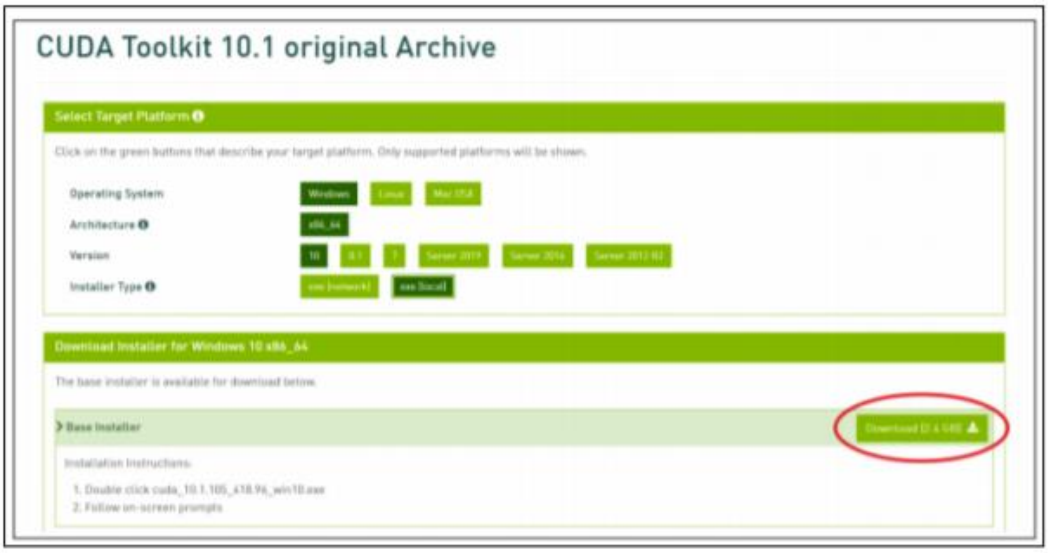

Figure 9 CUDA 10.1

The GPU needs to have a CUDA installed on the system, thus the CUDA with version 10.1 was installed from the official website of NVIDIA developers[1]. Further, the cuDNN version 7.6.5 was all installed which is compatible with the CUDA 10.1[2]. Later the CUDA, cuDNN was configured with the windows along with TensorFlow[3].

---

[1] https://developer.nvidia.com/cuda-downloads

[2] https://developer.nvidia.com/rdp/cudnn-archive#a-collapse765-101

[3] https://towardsdatascience.com/installing-tensorflow-with-cuda-cudnn-and-gpu-support-on-windows-10-60693e46e78

3 Project Implementation

3.1 Data Collection

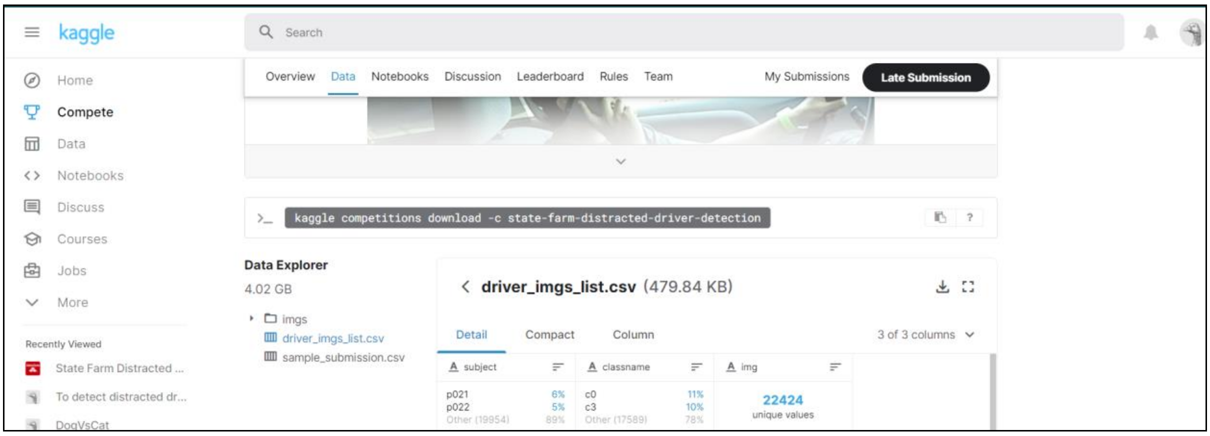

Figure 10 Website for data collection

The data which was used for the implementation was picked from Kaggle. The dataset was named as State Farm Distracted Driver Detection which was available under Kaggle competition.

3.2 Data Preparation

dataset.py

```python
from keras.applications.resnet50 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
# preprocessing_function is applied on each image but only after re-sizing & augmentation
# (resize => augment => pre-process)
data_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
# flow_From_directory generates batches of augmented data (where augmentation can be color conversion, etc)

#defining function for data generation train folder
def train_generator(train_dir, image_size, batch_size_training):
        train_generator = data_generator.flow_from_directory(train_dir, target_size=(image_size, image_size),
                                                             batch_size=batch_size_training, class_mode='categorical')
        return train_generator

#defining function for data generation validation folder
def validation_generator(val_dir, image_size, batch_size_validation):
        validation_generator = data_generator.flow_from_directory(val_dir, target_size=(image_size, image_size),
                                                                  batch_size=batch_size_validation, class_mode='categorical')
        return validation_generator

#defining function for data generation test folder
def test_generator(test_dir, image_size, batch_size_test):
        test_generator = data_generator.flow_from_directory(test_dir, target_size=(image_size, image_size),
                                                            batch_size=batch_size_test, class_mode='categorical')
        return test_generator
```

Figure 11 Data Preparation

The data preparation process where the data generators were created in the dataset.py file. The train, test, and validation generators for respective folders were called.

3.3 Data Pre-Processing

folder_automation.py

```
import os
import glob, shutil
#creating directories for binary classification
safe_path = "C:/Users/Pushkar/Desktop/imgs/train/safe-driving"
distracted_path = "C:/Users/Pushkar/Desktop/imgs/train/distracted-driving"
os.mkdir(safe_path)
print("Safe Driving Directory Created Successfully")
os.mkdir(distracted_path)
print("Distracted Driving Directory Created Successfully")
preprocessing_path = "C:/Users/Pushkar/Desktop/imgs/after-preprocessing"
os.mkdir(preprocessing_path)
logfile_path = "C:/Users/Pushkar/Desktop/imgs/logs"
os.mkdir(logfile_path)
aug_path = "C:/Users/Pushkar/Desktop/imgs/aug_safe_driving"
os.mkdir(aug_path)
print("aug_safe_driving Folder Created Successfully")
aug_path = "C:/Users/Pushkar/Desktop/imgs/saved_weights"
os.mkdir(aug_path)
print("saved_weights Folder Created Successfully")
print("##############################################################################################################")
#moving the images from the different class folder c0-c9 to safe-driving and distracted-driving folder.
for file in glob.glob('C:/Users/Pushkar/Desktop/imgs/train/c0/img*'):
    shutil.move(file, safe_path)
print("c0 Files moved successfully")
for file in glob.glob('C:/Users/Pushkar/Desktop/imgs/train/c1/img*'):
    shutil.move(file, distracted_path)
print("c1 Files moved successfully")
for file in glob.glob('C:/Users/Pushkar/Desktop/imgs/train/c2/img*'):
    shutil.move(file, distracted_path)
print("c2 Files moved successfully")
```

Figure 12 Folder Structuring

The folders structuring was done in this file named folder_automation.py where the 10 data folders ranging from c0 to c9 were transformed into two folders namely safe-driving and distracted-driving.

folder_automation.py img_aug.py

```
import cv2 as cv
import glob
import imageio
import os
from imgaug import augmenters as iaa
# get images
images = []
files = glob.glob("C:/Users/Pushkar/Desktop/imgs/train/safe-driving/*.jpg")

# stacking images
for myFile in files:
    image = cv.imread(myFile)
    image_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)  # change from BRG to RGB
    images.append(image_rgb)

sometimes = lambda aug: iaa.Sometimes(0.5, aug)

# defining sequence of augmentation steps that will be applied to every image.
seq = iaa.Sequential(
    [
        #
        # Apply the following augmenters to images.
        iaa.Fliplr(0.4),  # horizontally flip 40% of all images
        iaa.Flipud(0.1),  # vertically flip 10% of all images
        # Apply affine transformations to some of the images
        sometimes(iaa.Affine(
            rotate=(-10, 10),
        )),
```

Figure 13 Image Augmentation (a)

In this step, the img_aug.py file was created where the image augmentation was done particularly on a safe driving folder in order to balance the data.

```
    # Blur each image with varying strength using
            # gaussian blur (sigma between 0 and 3.0),
            # average/uniform blur (kernel size between 2x2 and 7x7)
            # median blur (kernel size between 3x3 and 11x11).
            iaa.OneOf([
                iaa.GaussianBlur((0, 3.0)),
                iaa.AverageBlur(k=(2, 7)),
                iaa.MedianBlur(k=(3, 11)),
            ])],
        random_order=True
)
# specify directory to save your images
write_to_dir = "C:/Users/Pushkar/Desktop/imgs/aug_safe_driving"
# number of augmented images per image
n_augs_per_image = 10
for i, image in enumerate(images):
    image_augs = seq.augment_images([image] * n_augs_per_image)
    for j, image_aug in enumerate(image_augs):
        imageio.imwrite(os.path.join(write_to_dir, "%03d_%02d.jpg" % (i, j)), image_aug)
```

Figure 14 Image Augmentation (b)

The various techniques are used in the code above namely horizontal flip, vertical flip, colour augmentation, and affine transformation.

pre_processing.py

```
import glob
import os
from random import choice
import shutil
#Moving images from aug_safe_driving to safe-driving
safe_path = "C:/Users/Pushkar/Desktop/imgs/train/safe-driving"
for file in glob.glob('C:/Users/Pushkar/Desktop/imgs/aug_safe_driving/*'):
    shutil.move(file, safe_path)

# setting up directory names
trainPath = 'C:/Users/Pushkar/Desktop/imgs/after-preprocessing/train'
valPath = 'C:/Users/Pushkar/Desktop/imgs/after-preprocessing/val'
testPath = 'C:/Users/Pushkar/Desktop/imgs/after-preprocessing/test'
#directory where the images are stored
crsPath = 'C:/Users/Pushkar/Desktop/imgs/train'

# splitup ratio into 70:15:15
train_ratio = 0.7
val_ratio = 0.15
test_ratio = 0.15

# To get number of classes
num_classes = len(list(os.walk(crsPath))[0][1])

files_count = {}
# To keep track of classes, count and dir
for i in range(num_classes):
    fc = 0
    files = list(os.walk(crsPath))[0][0] + "/" + list(os.walk(crsPath))[0][1][i]
    dataset_name = os.path.split(files)[-1]
```

Figure 15 Pre-processing (a)

```python
# cycle for Test directory
for k, v in files_count.items():
    imgs = []
    for img in glob.glob(v[1] + "/" + "*.jpg"):
        img = os.path.split(img)[-1]
        imgs.append(img)

    for x in range(countForTest):
        fileJpg = choice(imgs)  # get name of random image from origin directory
        z = os.path.join(crsPath, k, fileJpg)
        if not os.path.exists(os.path.join(testPath, k)):
            os.makedirs(os.path.join(testPath, k))
        try:
            # move both files into test directory
            shutil.move(os.path.join(crsPath, k, fileJpg), os.path.join(testPath, k, fileJpg))

        except Exception as e:
            print(e)
        # remove files from arrays
        imgs.remove(fileJpg)

for dir in glob.glob("C:/Users/Pushkar/Desktop/imgs/aug_safe_driving"):
    os.rmdir(dir)

for dir in glob.glob("C:/Users/Pushkar/Desktop/imgs/train"):
    shutil.rmtree(dir)
```

Figure 16 Pre-processing (b)

The data was split into the ratio of 70:15:15 for the created for the train, validation, and test folder. The data balancing was done in this file named pre_processing.py.

3.4 Model Building

The snippets in this section will be all about the different models that were used for implementing the driver distraction detection system

```python
import argparse

def parse_arguments(*args):
    parser = argparse.ArgumentParser()

    parser.add_argument('-model_type','--model_type', default='mycnn', type=str, choices=['vgg19', 'mycnn', 'thinmobilenet', 'xception']
                    help='vgg19|mycnn|thinmobilenet|xception')

    ######################## Dataset options ##############################

    parser.add_argument('-tr_dir', '--train_dir', default='C:/Users/Pushkar/Desktop/imgs/after-preprocessing/train', type=str,
                        help='Path to the train directory')

    parser.add_argument('-vl_dir', '--val_dir', default='C:/Users/Pushkar/Desktop/imgs/after-preprocessing/val', type=str,
                            help='Path to the val directory')

    parser.add_argument('-tst_dir', '--test_dir', default='C:/Users/Pushkar/Desktop/imgs/after-preprocessing/test', type=str,
                        help='Path to the test directory')

    parser.add_argument('-cls', '--num_classes', default=2, type=int,
                        help='Number of dataset classes')

    parser.add_argument('-res', '--image_resize', default=224, type=int,
                        help='Image Resize value')
    ############## Model options #####################################

    parser.add_argument('-lr', '--learning_rate', default=1e-5, type=float,
                        help='Learning rate for the generator')

    parser.add_argument('-of', '--objective_function', default='binary_crossentropy', type=str,
```

Figure 17 Calling Hyperparameters

The options.py file obtains the values for different hyper-parameters which are essential for the model building phase.

```
model.py
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, MaxPooling2D
import warnings
warnings.filterwarnings("ignore")
from keras.layers.core import Dense, Dropout, Flatten, Activation
from keras.layers.normalization import BatchNormalization
from keras.layers import Conv2D, MaxPooling2D

#defining MyModel class for all the pretraining models which include base model.
class MyModel():
    def __init__(self, baseModel, classes, D):
        self.baseModel = baseModel
        self.classes = classes
        self.D = D

    def build(self):
        """
            It takes basemodel , number of classes, and number of hidden units as a input.
            And return a new CNN architecure.

        """
        headModel = self.baseModel.output
        headModel = Flatten(name="Flatten")(headModel)
        headModel = Dense(self.D, activation='relu')(headModel)
        headModel = Dropout(0.5)(headModel)
        headModel = Dense(self.classes, activation='softmax')(headModel)
        return headModel
```

Figure 18 Model Building MyModel

```
#defining MyCNN class for the CNN model.
class MyCNN():
    def __init__(self, classes):
        self.classes = classes


    def build(self):
        model = Sequential()
        model.add(Convolution2D(32, (3, 3), input_shape=(224, 224, 3)))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.5))
        model.add(Convolution2D(64, (3, 3)))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.5))
        model.add(Convolution2D(128, (3, 3)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(2))
        model.add(Activation('softmax'))
        return model
```

Figure 19 Model Building MyCNN

The model.py file consists of the MyModel and the CNN model. Both of these models were defined in this file.

```python
def main(args):

    NUM_CLASSES = args.num_classes
    IMAGE_RESIZE = args.image_resize
    OBJECTIVE_FUNCTION = args.objective_function

    train_dir = args.train_dir
    val_dir = args.val_dir
    test_dir = args.test_dir

    # Common accuracy metric for all outputs, but can use different metrics for different output
    LOSS_METRICS = ['accuracy']
    NUM_EPOCHS = args.num_epochs

    # These steps value should be proper FACTOR of no.-of-images in train & valid folders respectively
    # Training images processed in each step would be no.-of-train-images / STEPS_PER_EPOCH_TRAINING
    STEPS_PER_EPOCH_TRAINING = args.steps_per_epoch_train
    STEPS_PER_EPOCH_VALIDATION = args.steps_per_epoch_val

    # These steps value should be proper FACTOR of no.-of-images in train & valid folders respectively
    # NOTE that these BATCH* are for Keras ImageDataGenerator batching to fill epoch step input
    BATCH_SIZE_TRAINING = args.train_batch
    BATCH_SIZE_VALIDATION = args.val_batch
    BATCH_SIZE_TESTING = args.test_batch
```

Figure 20 Main function ()

The first part of train.py file includes the configuration of the model for the training phase.

```python
    #VGG19 Model
    if args.model_type == 'vgg19':
        # load pretrained model:
        baseModel = VGG19(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
        # initialize head of network
        my_model = MyModel(baseModel, classes=NUM_CLASSES, D=256)
        headModel = my_model.build()
        # replace the FC with headModel:
        my_model = Model(inputs=baseModel.input, outputs=headModel)
        # # unfreezing some of conv layers:
        for layer in baseModel.layers[15:]:
            layer.trainable = True

    # Xception Model
    elif args.model_type == 'xception':
        # load pretrained model:
        baseModel = Xception(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
        # initialize head of network
        my_model = MyModel(baseModel, classes=NUM_CLASSES, D=256)
        headModel = my_model.build()
        # replace the FC with headModel:
        my_model = Model(inputs=baseModel.input, outputs=headModel)
        # unfreezing some of conv layers:
        for layer in baseModel.layers[15:]:
            layer.trainable = True
```

Figure 21 VGG19 and Xception

```
# Thin MobileNet Model
elif args.model_type == 'thinmobilenet':
        # load pretrained model:
        baseModel = MobileNetV2(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
        # initialize head of network
        my_model = MyModel(baseModel, classes=NUM_CLASSES, D=256)
        headModel = my_model.build()
        # replace the FC with headModel:
        my_model = Model(inputs=baseModel.input, outputs=headModel)
        # unfreezing some of conv layers:
        for layer in baseModel.layers[15:]:
                layer.trainable = True

#CNN Model
elif args.model_type == 'mycnn':
        # initialize head of network
        my_model = MyCNN(classes=NUM_CLASSES)
        my_model = my_model.build()

print(my_model.summary())

#SGD Optimizer used for all the models
sgd = optimizers.SGD(lr=args.learning_rate)
my_model.compile(optimizer = sgd, loss=OBJECTIVE_FUNCTION, metrics=LOSS_METRICS)
image_size = IMAGE_RESIZE
```

Figure 22 Thin MobileNet and CNN

Figure 20 and 21 show the code for different models used in the implementation namely, VGG19, Xception, Thin MobileNet, and CNN.

train.py plot.py

```
import matplotlib.pyplot as plt


def accuracy_plot(fit_history):
    plt.figure(1, figsize=(15, 8))
    plt.subplot(221)
    plt.plot(fit_history.history['acc'])
    plt.plot(fit_history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'])


def loss_plot(fit_history):
    plt.figure(1, figsize=(15, 8))
    plt.subplot(222)
    plt.plot(fit_history.history['loss'])
    plt.plot(fit_history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'])
```

Figure 23 Plotting the graphs

The plot.py file plots the graphs to showcase the different model's performance in terms of accuracy and loss against the training done with the total number of epochs.

```
train.py    test.py
import cv2
import numpy as np
import glob
from keras.preprocessing import image
from tensorflow.keras.models import load_model
import os
import matplotlib.pyplot as plt
import time
import logging
import os
import warnings
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
warnings.filterwarnings("ignore", category=FutureWarning)

model = load_model("C:/Users/Pushkar/Desktop/imgs/saved_weights/thinmobilenet_best_weights.h5")

# print(model.summary())

TEST_DIR = 'C:/Users/Pushkar/Desktop/imgs/test/'
# f, ax = plt.subplots(5, 5, figsize = (15, 15))

images = []
titles = []

count = 1

for img in glob.glob(TEST_DIR + "*.jpg"):
```

Figure 24 Testing

The test.py file performs the predictions for each of the model whether a driver is driving the car safe or if he is getting distracted while driving.

References

A. K. Vani, R. N. Raajan, D. Haretha Winmalar. and R. Sudharsan. (2020) 'Using the Keras Model for Accurate and Rapid Gender Identification through Detection of Facial Features' in 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2020, pp. 572-574, doi: 10.1109/ICCMC 48092.2020.ICCMC000106.

H. Shin, K. Lee, and C. Lee. (2020) 'Data Augmentation Method of Object Detection for Deep Learning in Maritime Image' in 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, Korea (South), 2020, pp. 463-466, doi: 10.1109/BigComp48618.2020.00-25.