Configuration Manual

MSc Research Project
Master of Science in Data Analytics

Saravana Ganesh Venkataramani
Student ID: X22120891

School of Computing
National College of Ireland

Supervisor: Dr.Syed Muslim Jameel

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Saravana Ganesh Venkataramani
Student ID:	X22120891
Programme:	MSc in Data Analytics **Year:** 2022-23
Module:	MSc Research project
Lecturer:	Dr.Syed Muslim Jameel
Submission Due Date:	18th September 2023
Project Title:	A deep learning approach for automatic traffic surveillance under extreme climatic conditions
Word Count:	**772** **Page Count:9**

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: Saravana Ganesh Venkataramani

Date: 18th September 2023

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

Saravana Ganesh Venkataramani
Student ID: X22120891

1. Introduction

The proposed research uses the machine learning (ML) model to predict and classify the traffic accidents and weather conditions based on images as inputs. In this manual, the configurations of the system (hardware and software), libraries, packages, techniques, plot methods, specifications, and other approaches implemented and utilized in the research for the development of VGG16-Net models are provided. The figures are derived from the screenshots of the python code developed.

2. System specification

The configuration of the system includes both software and hardware specifications. They are given below:

2.1 Hardware requirements

The table 1 and 2 shows the requirements of the system configuration as hardware and software requirements, specifically.

Table 1: Hardware configuration

System	LENOVO Ideapad
RAM	**16 GB RAM**
Operating System (OS)	Windows 10
Processor	Intel(R) Core(TM) i5 CPU
GPU	NVIDIA GeForce with 2GB space MTX250
Hard disk space/memory	2TB
System type	X64 processer with 64-bit OS

2.2 Software requirements

Python version of 3.7.3 is used as the software to develop the model. The current research uses two deep learning based VGG16 Net machine learning models; namely *traffic* and *weather* models. The models are built and implemented using the Jupyter notebook of version 6.4.12. The Anaconda Navigator is used for assessing the Jupyter notebook.

Table 2: Software configuration

Python	3.7.3 (64-bit)
Microsoft Excel	Windows 10 - 2019 edition
VS Code	1.63.1
Google Colab GPU	Tesla-K80
Anaconda	1.10.0

2.2.1 Packages and Libraries utilized

The following are the libraries and the packages adopted and utilized for the current research.

- ***Microsoft Package - Excel***

Figure 1 shows the Microsoft package – Excel sheet configuration/installation.

Figure 1: Microsoft package - Excel configuration *2019 edition*

- ***Python configuration***

Figure 2 portrays the packages and libraries used for the “weather model” and the figure 3 portrays the packages and libraries used for the “traffic” model i.e. the VGG16-Net ML models used, respectively. There are different packages and libraries in python where for this research, the following libraries and the packages were used:

- Os
- Numpy
- PyTorch Glob
- Matplotlib
- Torchvision

The Google Colab is configured to support the GPU utilized in this research (refer to figure 2).

- ***Configuration of Google Colab***

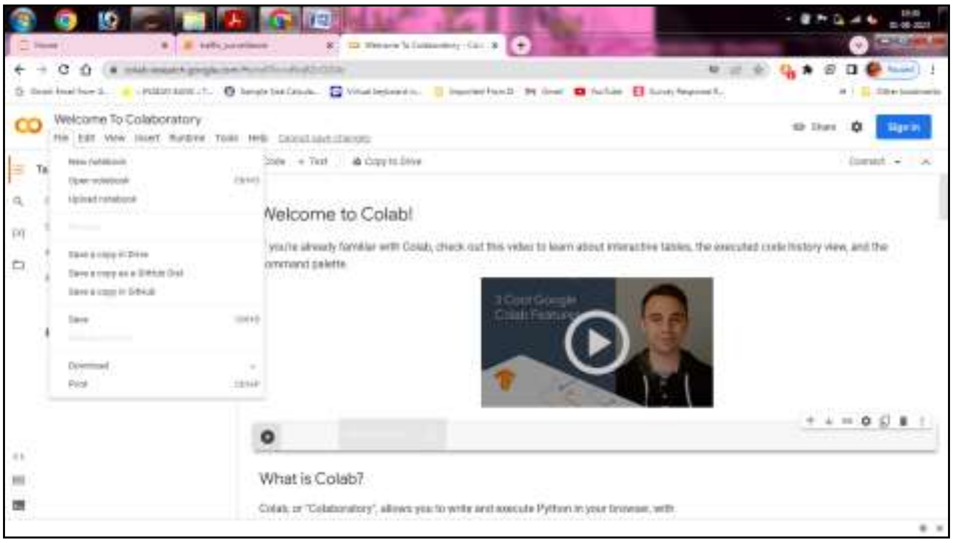

Figure 2: Google Collab

- *Configuration of Anaconda*

Figure 3: Anaconda Navigator - *2016 edition*

Figure 3 shows the usage of Anaconda Navigator of version 1.10.0 in 2016 edition along with Google Colab, GPU configuration.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

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

Figure 4: Libraries and packages

The libraries and the packages used for this research and model development are portrayed in the figure 4.

3. Implementation

3.1 Data acquisition

The datasets are the images obtained from videos and created through customization for the current research purposes. The datasets (refer to figure 5).

Figure 5: Dataset acquisition (random 500 images into traffic and weather folders, respectively)

```
!cp /content/gdrive/MyDrive/work/dataset.zip .
```

```
# keep random 500 files from all the sub folders and remove others
import os, cv2
import numpy as np
def remove_random_files(path, num_files_to_keep=200):
    for root, dirs, files in os.walk(path):
        if len(files) > num_files_to_keep:
            for file in random.sample(files, len(files) - num_files_to_keep):
                os.remove(os.path.join(root, file))

# import os
import random
remove_random_files("./traffic")
remove_random_files("./weather")
```

3.2 Data pre-processing

The datasets are pre-processed by image resizing; transforming (flipping and cropping randomly) and re-coloring (refer to figure 5) in the training, validation and testing phases of accident model. Similarly, the weather model's processing of datasets through training, validation and testing phases are represented in figure 6.

Figure 5: Image transforms in accident model

```
from torchvision import transforms
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(299),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.2),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'validation': transforms.Compose([
        transforms.Resize(299),
        transforms.CenterCrop(299),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(299),
        transforms.CenterCrop(299),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
}
```

Figure 6: Image transforms in weather model

```
import torch.utils.data
from torch.utils.data.sampler import SubsetRandomSampler
dataloaders = {
    "train": torch.utils.data.DataLoader(
        traffic_dataset, sampler=SubsetRandomSampler(train_indices), batch_size=16
    ),
    "validation": torch.utils.data.DataLoader(
        traffic_dataset_val, sampler=SubsetRandomSampler(validation_indices), batch_siz
e=16,
    ),
    "test": torch.utils.data.DataLoader(
        traffic_dataset_val, sampler=SubsetRandomSampler(test_indices), batch_size=32,
    ),
}
```

3.3 Models – Loading phase

The model developed is loaded where the VGG16-Net classification model accident and weather with the optimizer codes are shown is figures 7, 8 and 9 respectively.

Figure 7: Importing VGG16 model – Accident

```
from torchvision import models

vgg = models.vgg16(pretrained=True)
# resnet34 = nn.Sequential(*(list(resnet34.children())[:-1]))

head = nn.Sequential(
          nn.Linear(25088, 4096),
          nn.ReLU(inplace=True),
          nn.Linear(4096, 512),
          nn.ReLU(inplace=True),
          nn.Linear(512, 1),
          # nn.LogSoftmax(dim=1)
        )
vgg.classifier = head
for name, param in vgg.named_parameters():
    if "classifier" in name:
        continue
    else:
      param.requires_grad = False
vgg
# vgg(torch.rand(1, 3, 299, 299)).shape
```

Figure 8: Importing VGG16 model – Weather

```
from torchvision import models
from torch import nn
vgg = models.vgg16(pretrained=True)
# resnet34 = nn.Sequential(*(list(resnet34.children())[:-1]))

head = nn.Sequential(
          nn.Linear(25088, 4096),
          nn.ReLU(inplace=True),
          nn.Linear(4096, 512),
          nn.ReLU(inplace=True),
          nn.Linear(512, 3),
          # nn.LogSoftmax(dim=1)
        )
vgg.classifier = head
for name, param in vgg.named_parameters():
    if "classifier" in name:
        continue
    else:
      param.requires_grad = False
vgg
# vgg(torch.rand(1, 3, 299, 299)).shape
```

Figure 9: Loading of Optimizer algorithm (both weather and accident models)

```
import torch.optim as optim
from torch.optim import lr_scheduler

from torch.nn.modules.loss import BCEWithLogitsLoss
from torch.optim import lr_scheduler

#Loss
loss_fn = BCEWithLogitsLoss()
# criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(vgg.parameters(), lr=0.001, momentum=0.9)
# optimizer_ft = optim.Adam(vgg.parameters(),lr=0.0005)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

3.4 Model's architecture - Creation phase

The architecture of the VGG16-Net developed for traffic accident prediction is shown below in the figure 10 and the weather prediction is shown below in the figure 11, respectively:

Figure 10: VGG16-Net layers architecture for accident model

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
de=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
de=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=4096, out_features=512, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

Figure 11: VGG16-Net layers architecture for weather model

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
de=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
de=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=4096, out_features=512, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=512, out_features=3, bias=True)
  )
)
```

3.5 Modeling

The OS and Numpy are imported and the video frames are obtained. The video path is defined as “traffic/Accident”, “traffic/Normal”, “weather/Normal”, “weather/Snowy”, and “weather/Rainy”. The video files are loaded, images are obtained, processed, and passed through the model for training, validation and testing phases, of the VGG16 model developed. The current research has no annotations (refer to figure 12) and thus, the classified images are saved in their respective (pre-defined) folders, respectively.

Figure 12: Labels and Folders of classification

```
import os, cv2
import numpy as np

def read_anno_file(anno_file):
    assert os.path.exists(anno_file), "Annotation file does not exist!" + anno_file
    result = []
    with open(anno_file, 'r') as f:
        for line in f.readlines():
            items = {}
            items['vid'] = line.strip().split(',[')[0]
            labels = line.strip().split(',[')[1].split('],')[0]
            items['label'] = [int(val) for val in labels.split(',')]
            others = line.strip().split(',[')[1].split('],')[1].split(',')
            items['startframe'], items['vid_ytb'], items['lighting'], items['weather'],
items['ego_involve'] = others
            result.append(items)
    f.close()
    return result


def get_video_frames(video_file, topN=50):
    # get the video data
    cap = cv2.VideoCapture(video_file)
    ret, frame = cap.read()
    video_data = []
    while (ret):
        video_data.append(frame)
        ret, frame = cap.read()
    print("original # frames: %d"%(len(video_data)))
    assert len(video_data) >= topN
    video_data = video_data[:topN]
    return video_data


anno_file = "Crash-1500.txt"
anno_data = read_anno_file(anno_file)
import os
try:
    os.makedirs("traffic/Accident")
    os.makedirs("traffic/Normal")
    os.makedirs("weather/Normal")
    os.makedirs("weather/Snowy")
    os.makedirs("weather/Rainy")
except:
    pass
video_path = "Crash-1500"
for anno in anno_data:
    video_file = os.path.join(video_path, anno['vid'] + ".mp4")
    assert os.path.exists(video_file), "Video file does not exist!" + video_file
    # read frames
    frames = get_video_frames(video_file, topN=50)
    labels = anno['label']
    weather = anno['weather']
    assert weather in ['Normal', 'Snowy', 'Rainy'], "weather is not correct!"
    # print("file: %s, # frames: %d, # labels: %d."%(video_file, len(frames), len(label
s)))
    # print(len(labels))
    for idx, im in enumerate(frames):
        if idx < 35:
            continue
        im = cv2.resize(im, (299, 299))
```

4. Evaluation metrics – Performance evaluation

The performance of the VGG16-Net model for traffic accident (refer to figure 13) and weather (refer to figure 14) predictions are evaluated through metric evaluation by plotting the epoch runs in the graphs, they are represented below:

Figure 13: Loss and Accuracy plotting of accident model

```
def plot_accuraies(accuracies):
    import matplotlib.pyplot as plt
    import numpy as np
    def convert_to_cpu(acc):
        return [a*100 if type(a)==float else a.cpu()*100 for a in acc]
    # plot the data
    plt.plot(convert_to_cpu(accuracies['train']['accu']), 'r-', label = "Train Accurac
y")
    plt.plot(convert_to_cpu(accuracies['validation']['accu']), 'b-', label = "validatio
n Accuracy")

    # set the limits
    # plt.xlim([0, 100])
    # plt.ylim([0, 24])

    plt.title('Accuracy PLot (Accident Detection Model)')
    plt.legend(loc='best')
    # display the plot
    plt.show()
    # import matplotlib.pyplot as plt
    # plot the data
    plt.plot(convert_to_cpu(accuracies['train']['loss']),'r-', label = "Train Loss")
    plt.plot(convert_to_cpu(accuracies['validation']['loss']), 'b-', label = "validatio
n Loss")

    # set the limits
    # plt.xlim([0, 1])
    # plt.ylim([0, 1])

    plt.title('Loss PLot (Accident Detection Model)')
    plt.legend(loc='best')
    # display the plot
    plt.show()
```

Figure 14: Loss and Accuracy plotting of accident model

```
def plot_accuraies(accuracies):
    import matplotlib.pyplot as plt
    import numpy as np
    def convert_to_cpu(acc):
        return [a*100 if type(a)==float else a.cpu()*100 for a in acc]
    # plot the data
    plt.plot(convert_to_cpu(accuracies['train']['accu']), 'r-', label = "Train Accurac
y")
    plt.plot(convert_to_cpu(accuracies['validation']['accu']), 'b-', label = "validatio
n Accuracy")

    # set the limits
    # plt.xlim([0, 100])
    # plt.ylim([0, 24])

    plt.title('Accuracy PLot (Weather Model)')
    plt.legend(loc='best')
    # display the plot
    plt.show()
    # import matplotlib.pyplot as plt
    # plot the data
    plt.plot(convert_to_cpu(accuracies['train']['loss']),'r-', label = "Train Loss")
    plt.plot(convert_to_cpu(accuracies['validation']['loss']), 'b-', label = "validatio
n Loss")

    # set the limits
    # plt.xlim([0, 1])
    # plt.ylim([0, 1])

    plt.title('Loss PLot (Weather Model)')
    plt.legend(loc='best')
    # display the plot
    plt.show()
```

References

[1] Xiao. H, Zhang. F, Shen. Z, Wu. K and Zhang. J, (2021), "Classification of weather phenomenon from images by using deep convolutional neural network", *Earth and Space Science,* 8(e2020EA001604): 1-9.

[2] Totare, R, Bhalsing, V, Lende, M, Maramwar, T, Naikwadi, C. (2023), "Car Damage Detection and Price Prediction Using Deep Learning", *International Journal of Creative Research Thoughts (IJCRT),* 11(6), pp. 759-766.

[3] Naufal. M.F and Kusuma. S.F, (2021), "Weather image classification using convolutional neural network with transfer learning", In: *AIP Conference Proceedings,* 2470 (050004): 1-13.