Configuration Manual

MSc Research Project
Data Analytics

Samuel Biwei Tanga
Student ID: x20187784

School of Computing
National College of Ireland

Supervisor: Dr. Martin Alain

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Samuel Biwei Tanga
Student ID:	20167784
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Martin Alain
Submission Due Date:	16/12/2021
Project Title:	A Deep Learning Approach to Vehicle Make and Model Recognition with Specification Matching
Word Count:	1712
Page Count	19

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:	Samuel Biwei Tanga
Date:	16/12/2021

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

Samuel Biwei Tanga
X20167784

1 Introduction

This document provides comprehensive information on how to effectively replicate the implementation aspect of the research "A Deep Learning Approach to Vehicle Make and Model Recognition with Specification Matching" It provides in - depth information on how to configure the development environment and also information on software and hardware requirements needed for implementing, executing, and testing the models used in the research. The sections that follow this provide these processes.

2 System Configuration

The recommended software and hardware requirements are given in this section. Also given, is the configuration used by the author.

2.1 Hardware Configuration

Table 1: Hardware configuration

Hardware	Recommended	Used
Operating System	• Ubuntu 16.04 or later • macOS 10.12.6 or later • Windows 7 or later	Windows 10
RAM	At least 8GB	16GB
CPU	At least Core i5	Core i7
Hard Disk	At least HDD 500GB	SSD 250GB

2.2 Software Requirements

Table 2: Software Requirements

Software	Version Used
Python	3.9.5
pip	Pip 19.0
Google Chrome	96.0
Jupyter Notebook	6.4.0
Google Colab	

The Google Colab is what was used for data processing, training, and testing the models and also presentation of results. The PyCharm IDE was used to run the python scripts for the development of the GUI application.

2.3 Google Colab

Google colab[1] is an online IDE offered by google to run Jupyter notebooks. It is used mostly for deep learning and neural networks projects. Most packages are already installed on the google colab environment, users only need to import these packages in order to use them. The google colab is associated with the user's google account i.e. once a user has a google account, they can have access to google colab. Files used in google colab are preferably stored in a google drive. The figure below shows a google colab environment.

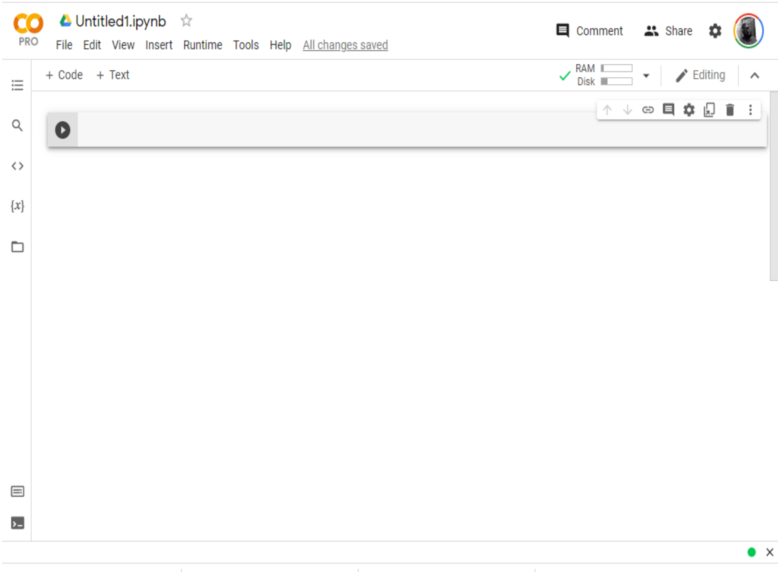

Figure 1: Google Colab Environment

3 Implementation

A complete step by step set of instructions with illustrated figures on how to replicate the project from data acquisition to generating results is shown in this section.

3.1 Data Acquisition

The dataset used for this project can be downloaded from GitHub[2] as shown in the Figure 2 below.

[1] https://research.google.com/colaboratory/
[2] https://github.com/faezetta/VMMRdb

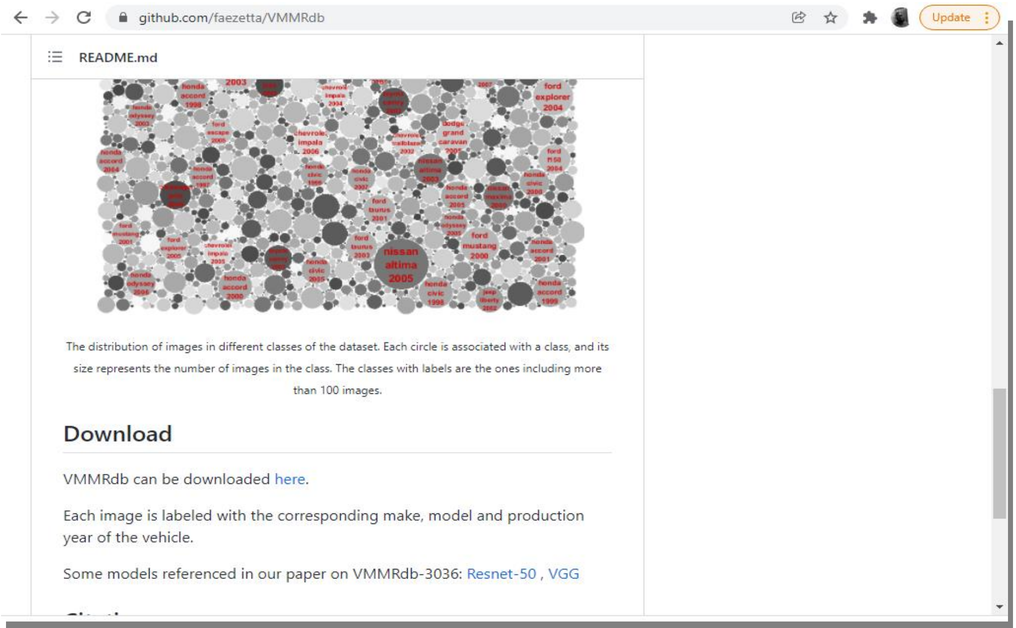

Figure 2: Github Download page

3.2 Data Storage

The acquired data is stored on the researcher's google drive in order to be used with google colab. A new folder is created in the google drive (this can be any name) which will store all the files associated with the project as soon as processing starts. In this case the folder has been named VehicleMarks.

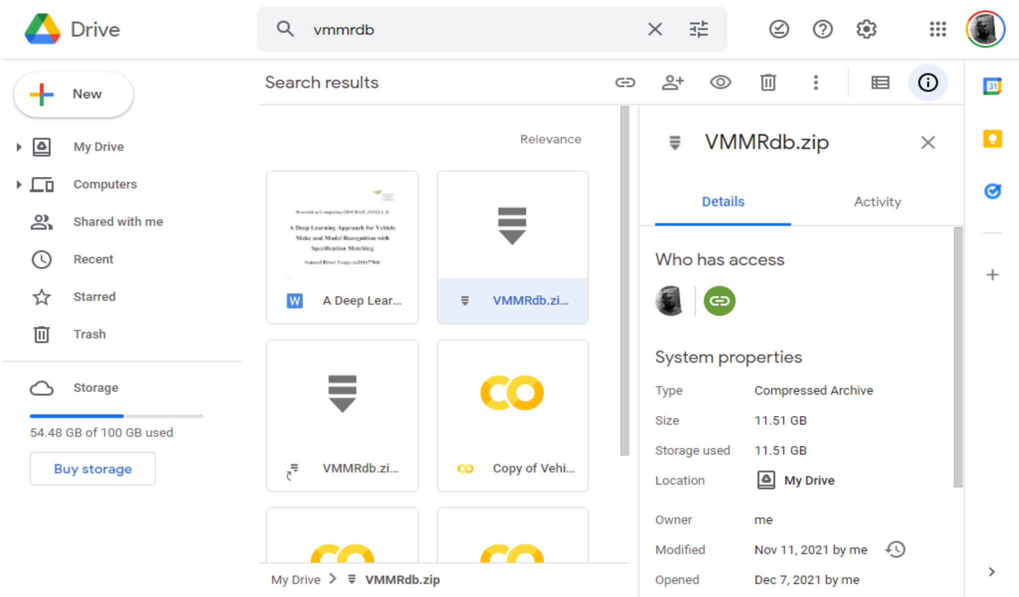

Figure 3: Google drive containing downloaded dataset

3.3 Data Preparation

The next step required is to connect the google drive to the google colab environment and then change the working directory to the folder that was created earlier to store all project files.

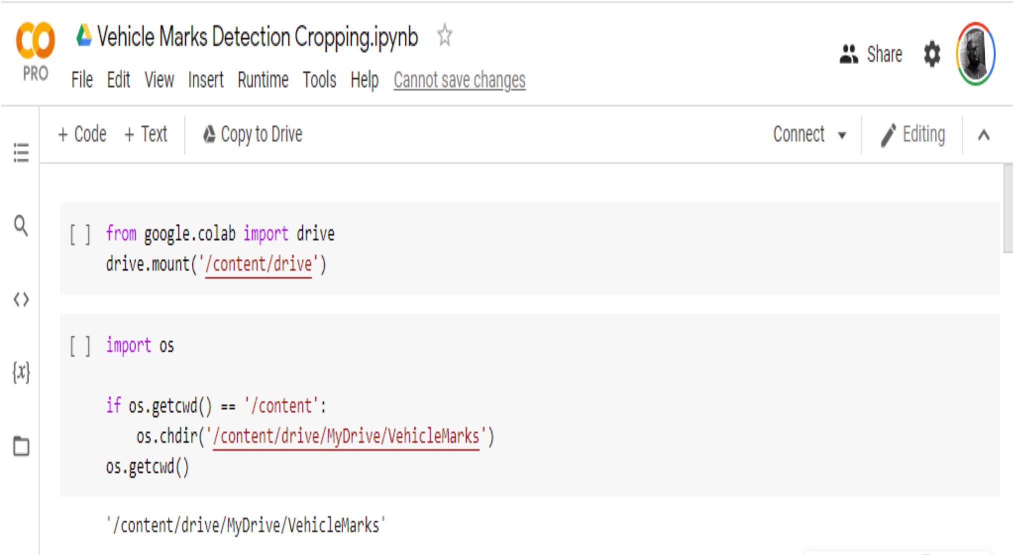

Figure 4: Connecting google drive to colab environment

After this the libraries needed for the initial process of the project are imported.

```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset, Subset
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.models.segmentation import deeplabv3_resnet101
from torchvision.datasets import ImageFolder

import io
from tqdm import tqdm
from random import shuffle
from typing import Tuple, List, Dict, Any, Optional, Callable
from zipfile import ZipFile
from collections import Counter
from PIL import Image
```

Figure 5: Importing Libraries

The next step is to define functions to iterate over the zipped dataset and also for selecting files in order to create a subset before passing in the arguments for number of classes, maximum and minimum number of images per class to be chosen. The figures 6 and 7 below illustrate this.

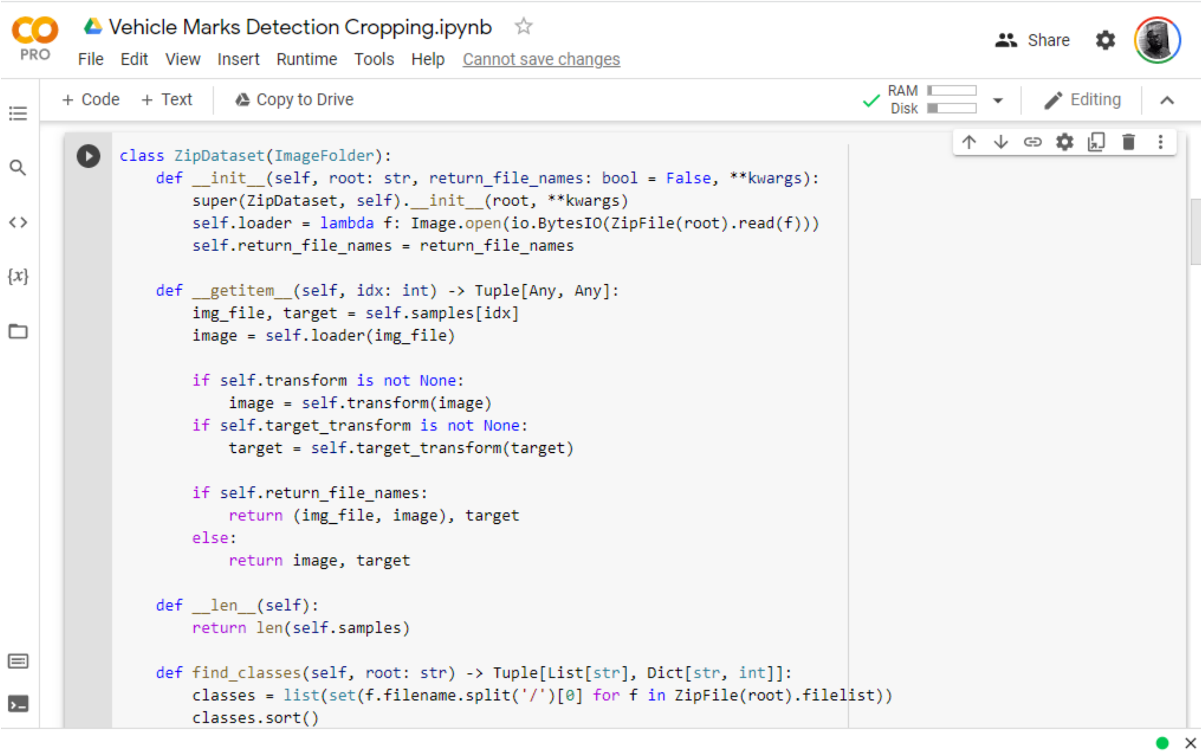

Figure 6: Function to iterate over images in zipped file

Figure 7: Function to select images in zipped file

The arguments for selecting the balanced subset dataset are passed. The below figure 8 shows the name for the new subset Cropped_v3, the image size, the minimum and maximum number of cars per class and also the number of classes.

```python
args = {
    'DATASET_ZIP_FILE': 'VMMRdb.zip',
    'CROPPED_ZIP_FILE': 'Cropped_v3.zip',
    'IMG_SIZE': 224,
    'BATCH_SIZE': 4,
    'MIN_CLASS_ELEMENTS': 26, # minimum number of cars per class to keep
    'MAX_CLASS_ELEMENTS': 27, # maximum number of cars per class to keep
    'MAX_NUM_CLASSES': 203 # maximum number of classes to keep
}
```

```python
transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Resize((args['IMG_SIZE'], args['IMG_SIZE'])),
    # torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

dataset = SelectiveDataset(
    args['DATASET_ZIP_FILE'],
    args['MIN_CLASS_ELEMENTS'],
    args['MAX_CLASS_ELEMENTS'],
    args['MAX_NUM_CLASSES'],
    return_file_names=True,
    transform=transform
)
```

Figure 8: Parsing the arguments for the subset creation

After the arguments are passed the next step is define the function for creating the subset, one important aspect is to ensure the images are saved in the 'jpeg' format. After this is done then the create dataset function is called to create the subset. Creation of the subset runs for about 5hrs 26mins.

```python
def create_cropped_dataset(dataset, model, out_size, cropped_zip_file, offset=0):
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    model = model.to(device).eval()

    dataset = Subset(dataset, range(offset, len(dataset)))
    loader = DataLoader(dataset, batch_size=args['BATCH_SIZE'])

    for (names, imgs), _ in tqdm(loader):
        imgs = imgs.to(device)
        # outs = model(imgs)
        outs = imgs

        for i, out in enumerate(outs):
            # boxes = out['boxes'].detach().cpu().numpy().astype(np.int_)

            # boxes_area = (boxes[:,2] - boxes[:,0]) * (boxes[:,3] - boxes[:,1])
            # x_min, y_min, x_max, y_max = boxes[boxes_area == np.max(boxes_area)][0]

            # image = imgs[i, ..., y_min:y_max, x_min:x_max].cpu().numpy() * 255
            image = out.cpu().numpy() * 255
            image = image.transpose(1, 2, 0).astype(np.uint8)
            image = Image.fromarray(image)
            image = image.resize((out_size, out_size))

            with ZipFile(cropped_zip_file, 'a') as f:
                byte_image = io.BytesIO()
                image.save(byte_image, format='JPEG')
                f.writestr(names[i], byte_image.getvalue())
```

```python
if True:
    create_cropped_dataset(dataset, model, args['IMG_SIZE'], args['CROPPED_ZIP_FILE'])
else:
    offset = len(ZipDataset(args['CROPPED_ZIP_FILE']))
    create_cropped_dataset(dataset, model, args['IMG_SIZE'], args['CROPPED_ZIP_FILE'], offset)
```

```
  0%|          | 0/1250 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/torch/functional.py:445: UserWarning: torch.meshgrid:
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
100%|██████████| 1250/1250 [5:26:32<00:00, 15.67s/it]
```

Figure 9: Creating the new susbset

3.4 Data Processing

The research created a new notebook for data processing and modelling. Each model is created on a new notebook (This was the researcher's choice; all the notebooks can be merged as one). For this project, PyTorch Lighting is used. PyTorch lighting is a PyTorch wrapper that gives full control and flexibility over codes. The trainer automates every other thing. The figure 10 below shows the installation of the module.

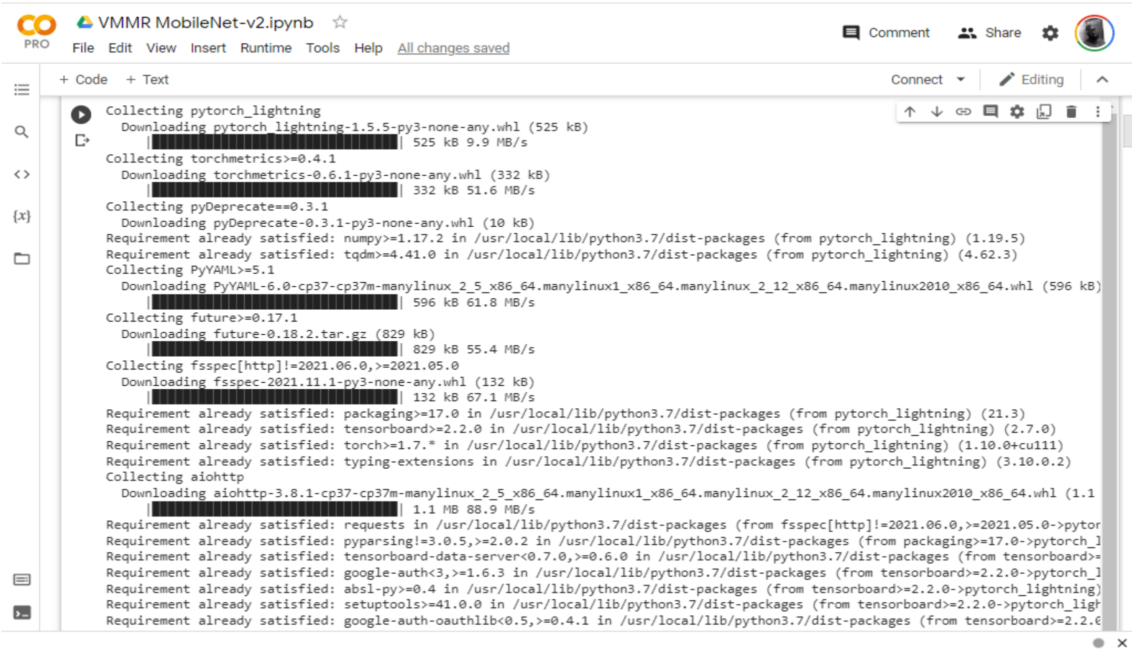

Figure 10: Installing the Pytorch Ligthing

The required libraries for building the models are imported after installing PyTorch_Ligthing

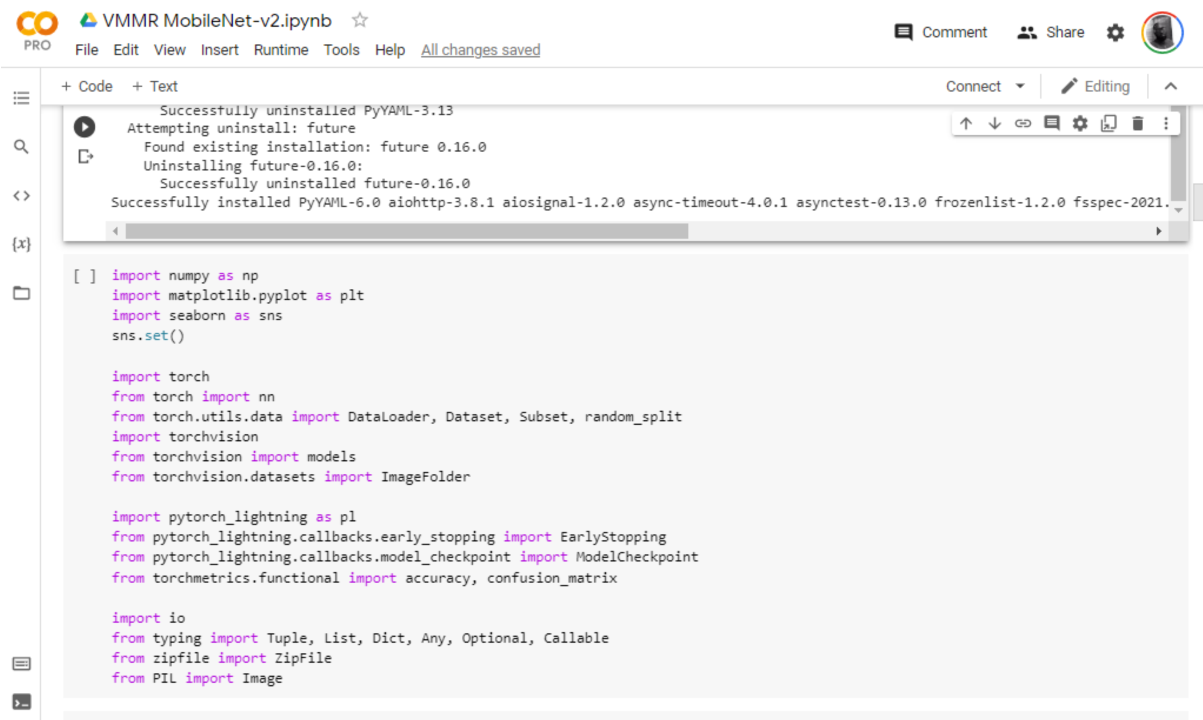

Figure 11: Importing other required libraries

Again, the function for iterating over the zipped data set is defined like was done for the original dataset.

```
[ ] class ZipDataset(ImageFolder):
        def __init__(self, root: str, return_file_names=False, return_targets=True, **kwargs):
            super(ZipDataset, self).__init__(root, **kwargs)
            self.loader = lambda f: Image.open(io.BytesIO(ZipFile(root).read(f)))
            self.return_file_names = return_file_names
            self.return_targets = return_targets

        def __getitem__(self, idx: int) -> Tuple[Any, Any]:
            img_file, target = self.samples[idx]
            image = self.loader(img_file)

            if self.transform is not None:
                image = self.transform(image)
            if self.target_transform is not None:
                target = self.target_transform(target)

            to_return = None
            if self.return_file_names:
                to_return = (img_file, image)
            else:
                to_return = image

            if self.return_targets:
                to_return = (to_return, target)

            return to_return

        def __len__(self):
            return len(self.samples)
```

Figure 12: Function to iterate over images in the subset zipped file

Next the Lighting module is created. All the training loop details are embedded here. The lighting module handles running the training, validation and the test dataloaders, as well as putting batches and computations on the right devices.

VMMR MobileNet-v2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text Connect Editing

```
class PLZipDataset(pl.LightningDataModule):
    def __init__(
        self,
        train_root: str,
        test_root: Optional[str] = None,
        return_file_names: bool = False,
        batch_size: int = 1,
        **kwargs
    ):
        super(PLZipDataset, self).__init__()

        self.batch_size = batch_size
        self.train_dataset = ZipDataset(train_root, return_file_names, **kwargs)
        self.test_dataset = None

        if test_root is not None:
            self.test_dataset = ZipDataset(test_root, return_file_names, **kwargs)


    ''' def setup(self, stage: Optional[str] = None):
        if stage in [None, 'fit']:
            train_count = int(self.train_ratio * len(self.dataset))
            val_count = len(self.dataset) - train_count
            self.train_data = self.dataset
            # self.train_data, self.val_data = random_split(self.dataset, [train_count, val_count]) '''

    def train_dataloader(self):
        return DataLoader(self.train_dataset, batch_size=self.batch_size, shuffle=True)

    def val_dataloader(self):
```

Figure 13: Function to iterate over images in zipped file

Next the parameters for training are parsed as arguments. Parameters such as learning rate ‘LR’, number of epochs, Image size, Patience Value and batch size.

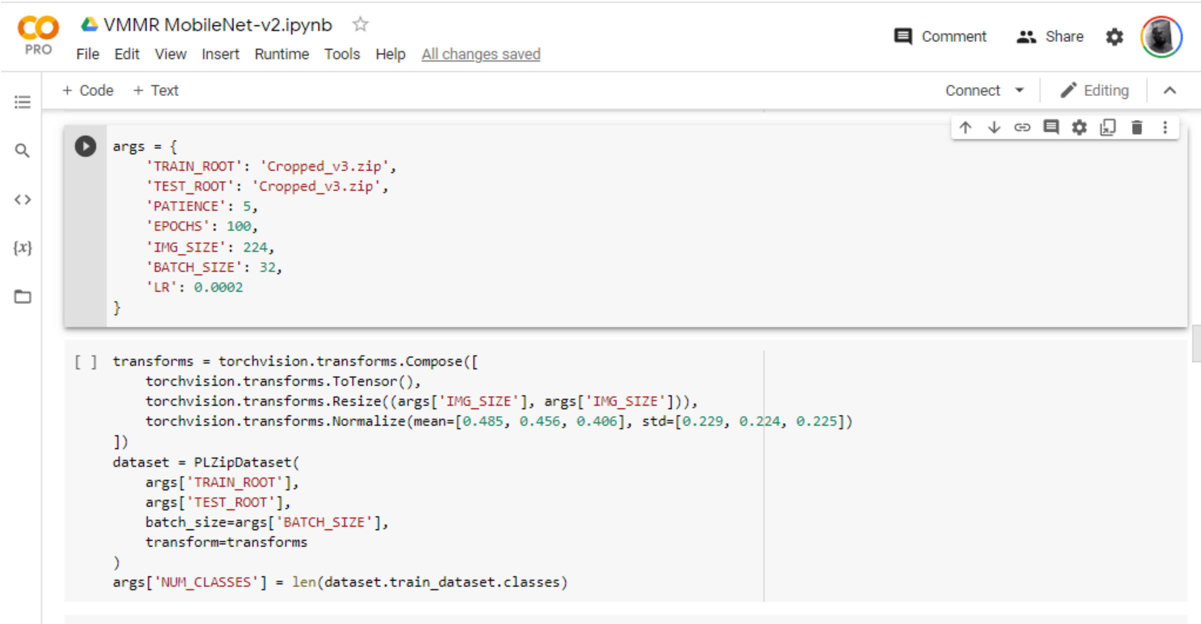

Figure 14: Passing in the training parameters

The next step is to create the model class, in this case the MobileNet-V2 class.

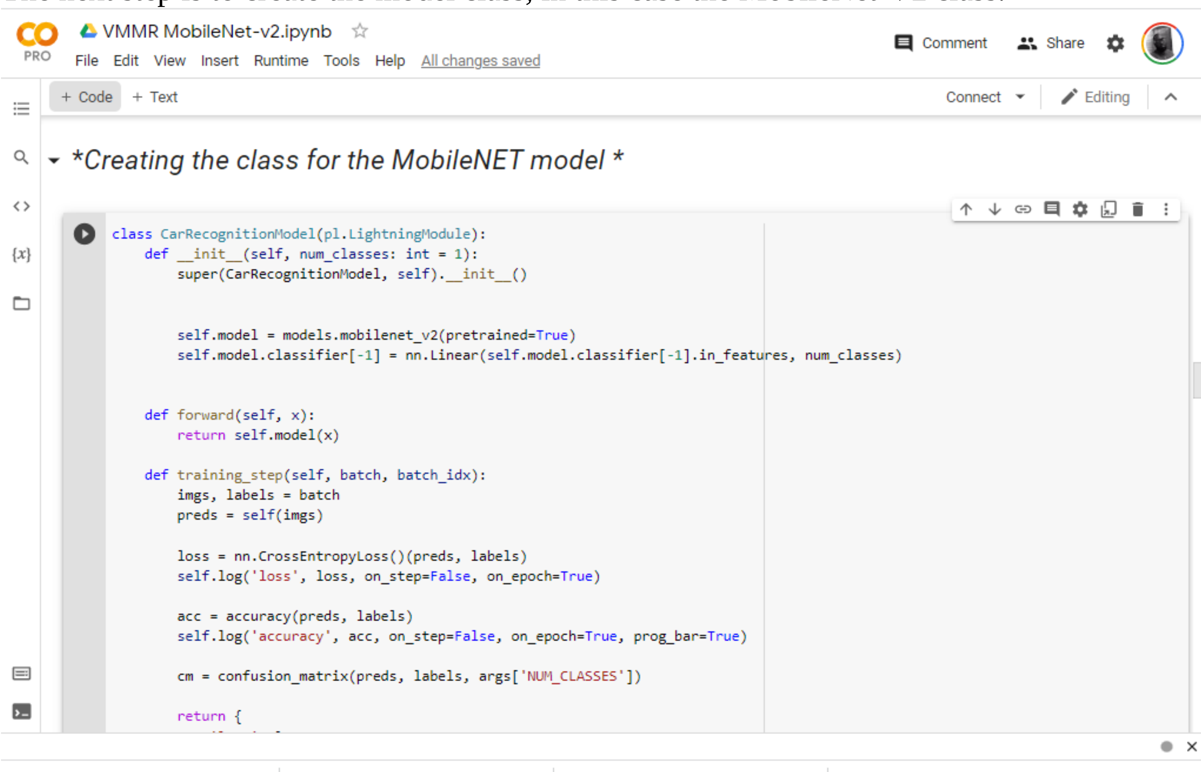

Figure 15: Creating the class for the MobileNet vehicle recognition model

The next step is to define the training and validation step function, this functions are still wrapped within the Vehicle recognition module. Also within this class is the optimizer function.

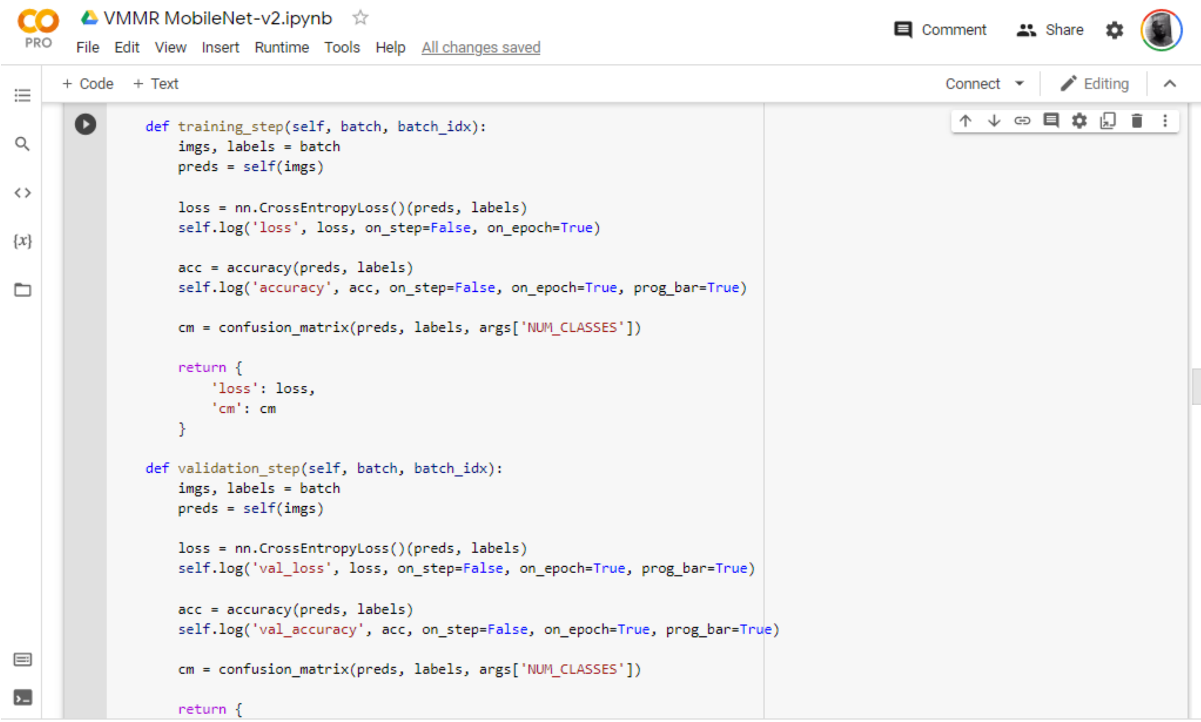

Figure 16: Training and Validation step function

The next step is to run the training code, embedded in this code is the callback function for early stopping which prevents the model from being overfitted.

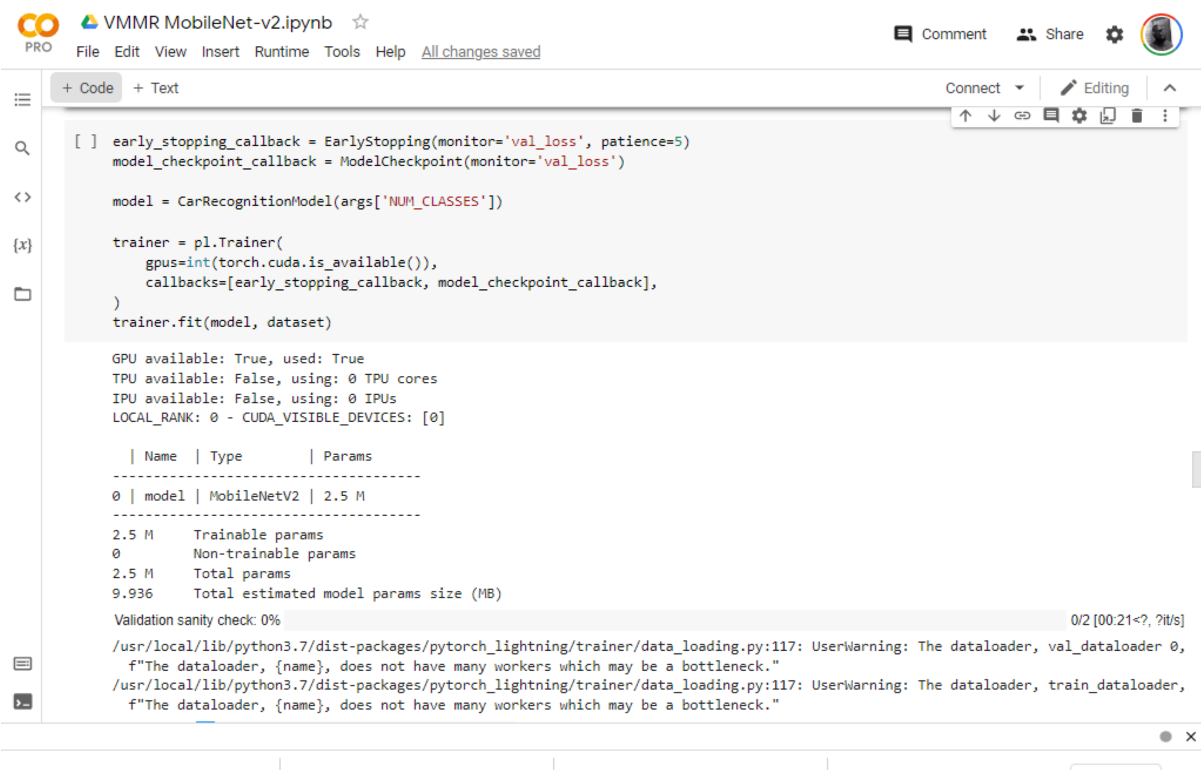

Figure 17: Training initiation

The figure 18 below shows the training process of the MobileNet-v2 model.

Figure 18: Training the MobileNet-v2 model

The next step is to load the logged trainer metrics and also write the codes for testing the trained model with random images from the test set (in this case, 12 images were chosen) as shown in figure 19 and 20 below. In order to chose which training checkpoint the testing will be carried on there is a need to go into the folder created earlier in the drive where all project files are saved. In the lighting logs folder the final epoch checkpoint file is stored there. The path is copied in pasted in load from checkpoint argument as shown below.

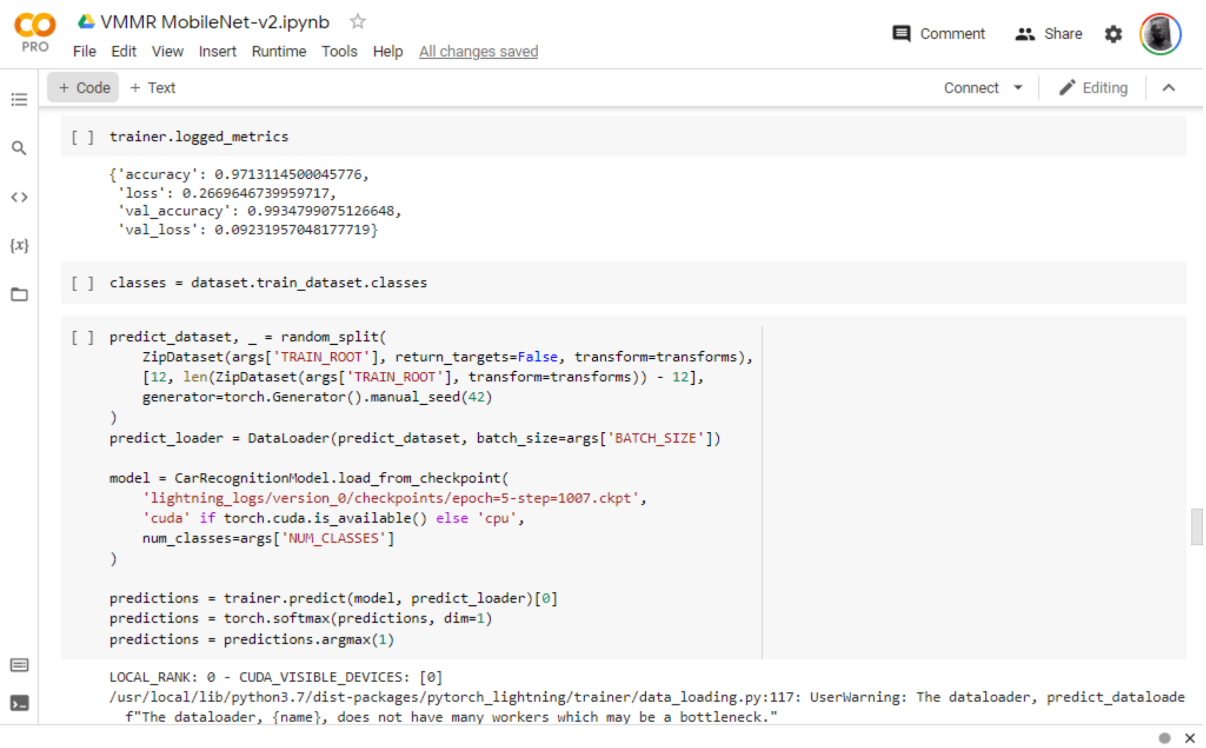

Figure 19: Showing logged metrics and feeding training checkpoint for testing

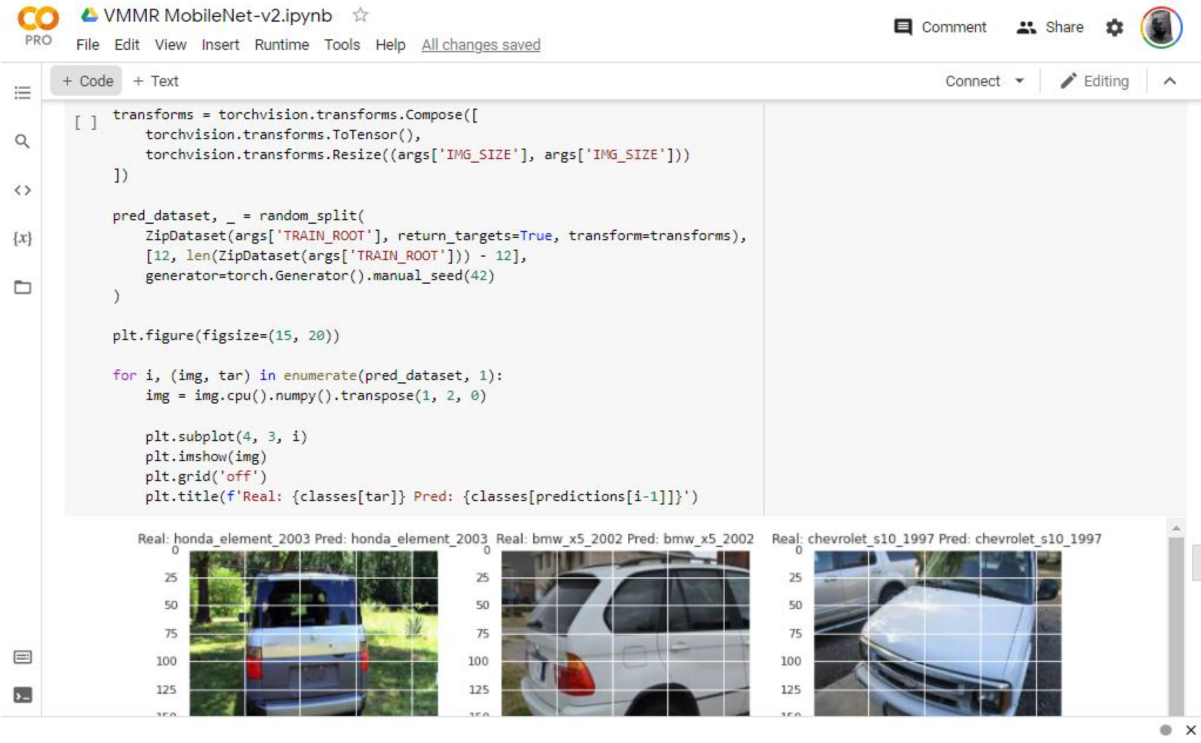

Figure 20: Testing module

The figure 21 below shows the results of the testing over the randomly selected images.

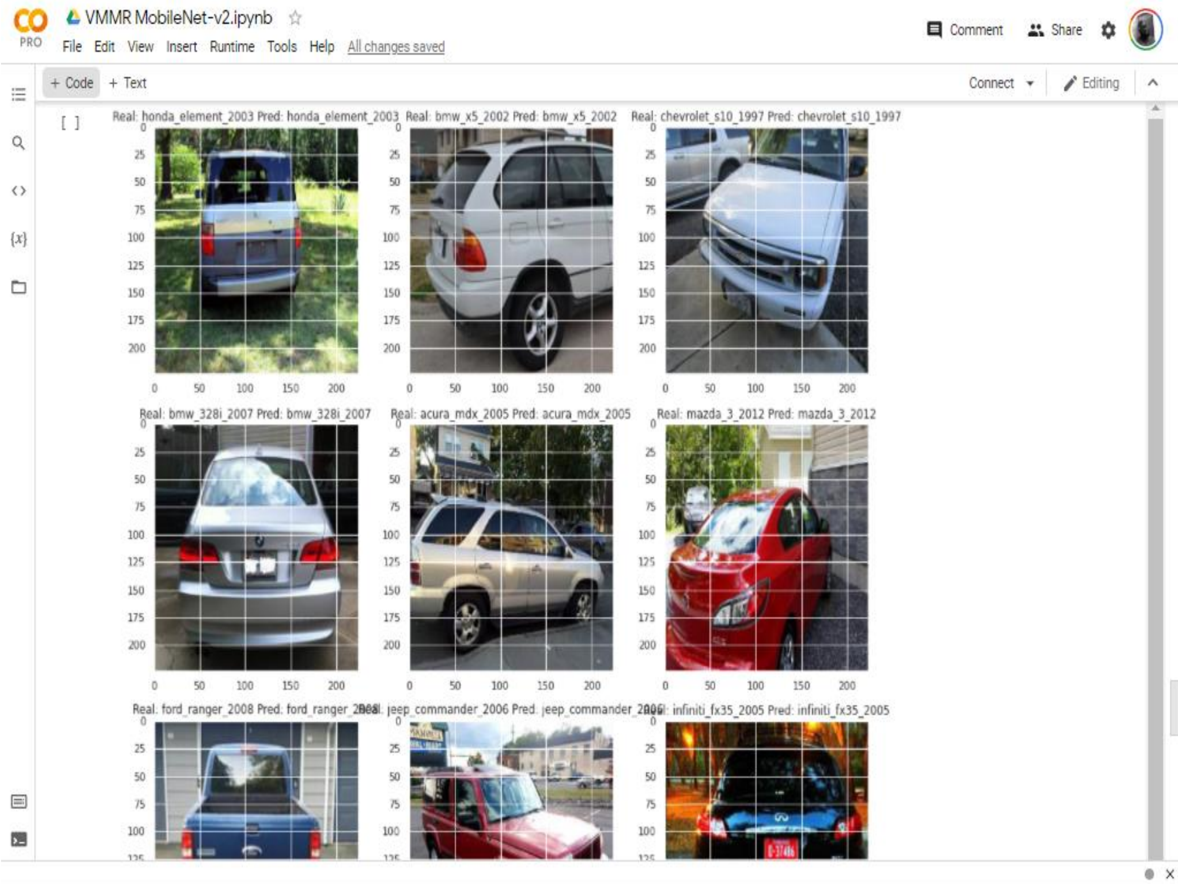

Figure 21: Test results (Real vs Predicted)

The last and final step is the loading of the tensorboard which is used to display the logged metrics in a graphical format. This is shown below in figure 22.

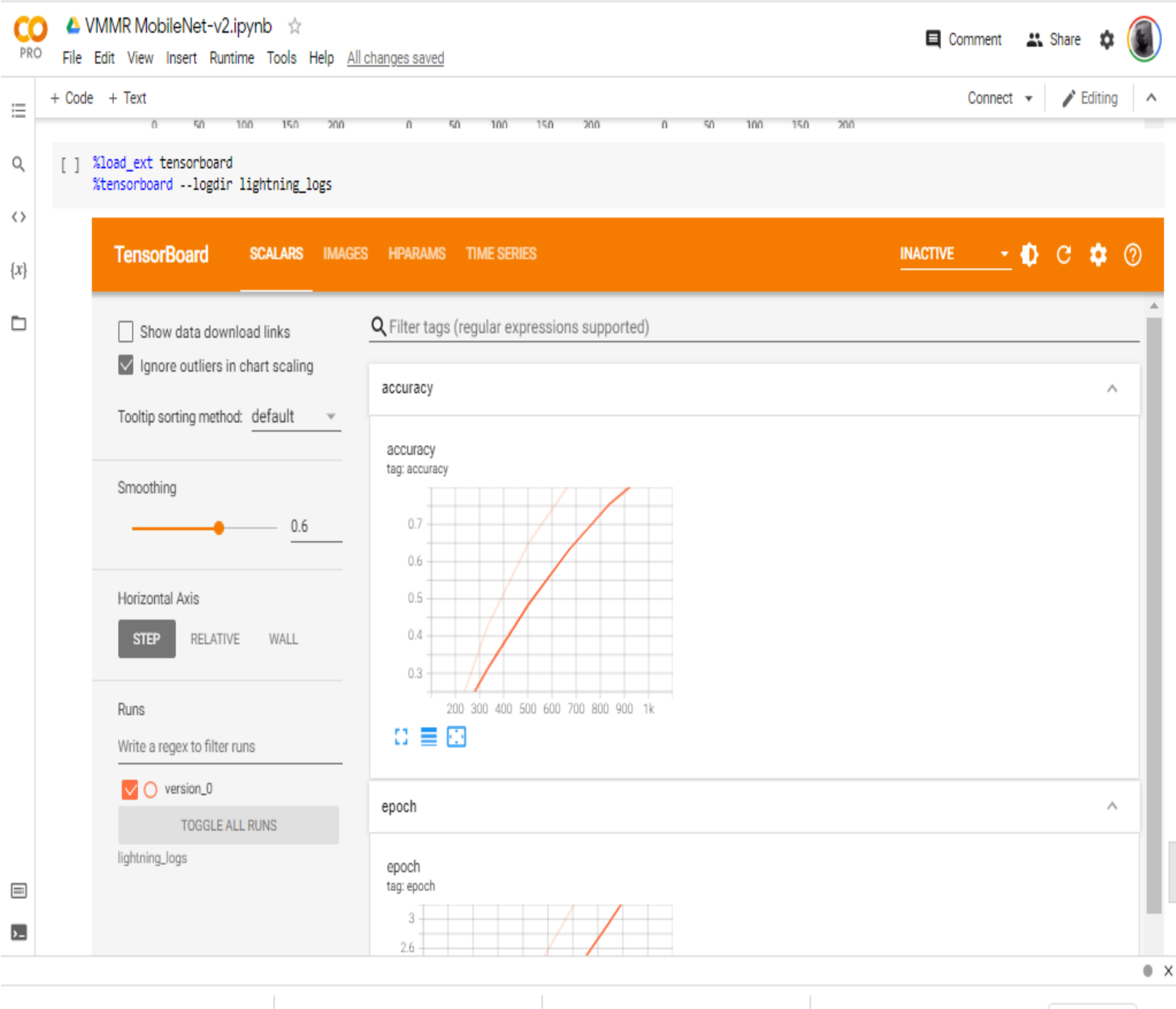

Figure 22: Tensorboard showing graphs

The same process is applied for every other model, the only difference is the creation of the Vehicle Recognition class which defines the model being trained which are shown below for both the VGG-16 and ResNet-50 models in figure 23.

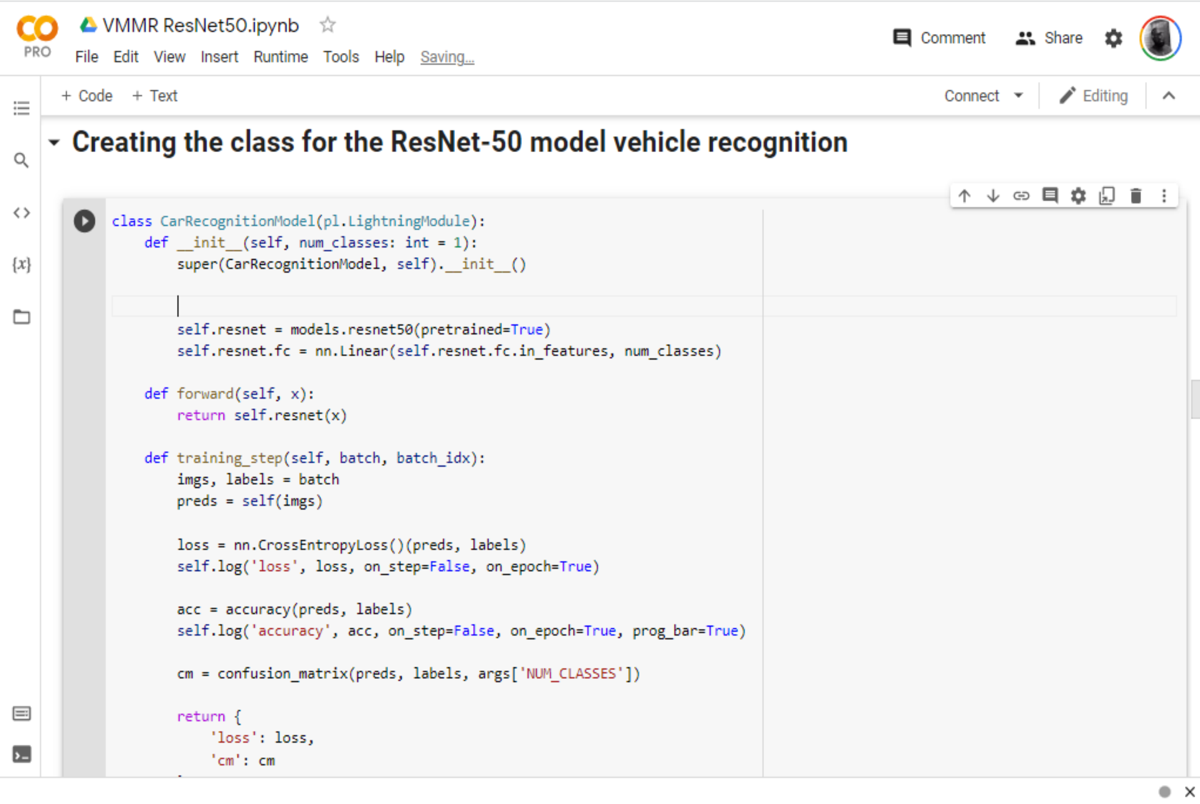

Figure 23: Creating the class for the ResNet-50 vehicle recognition model

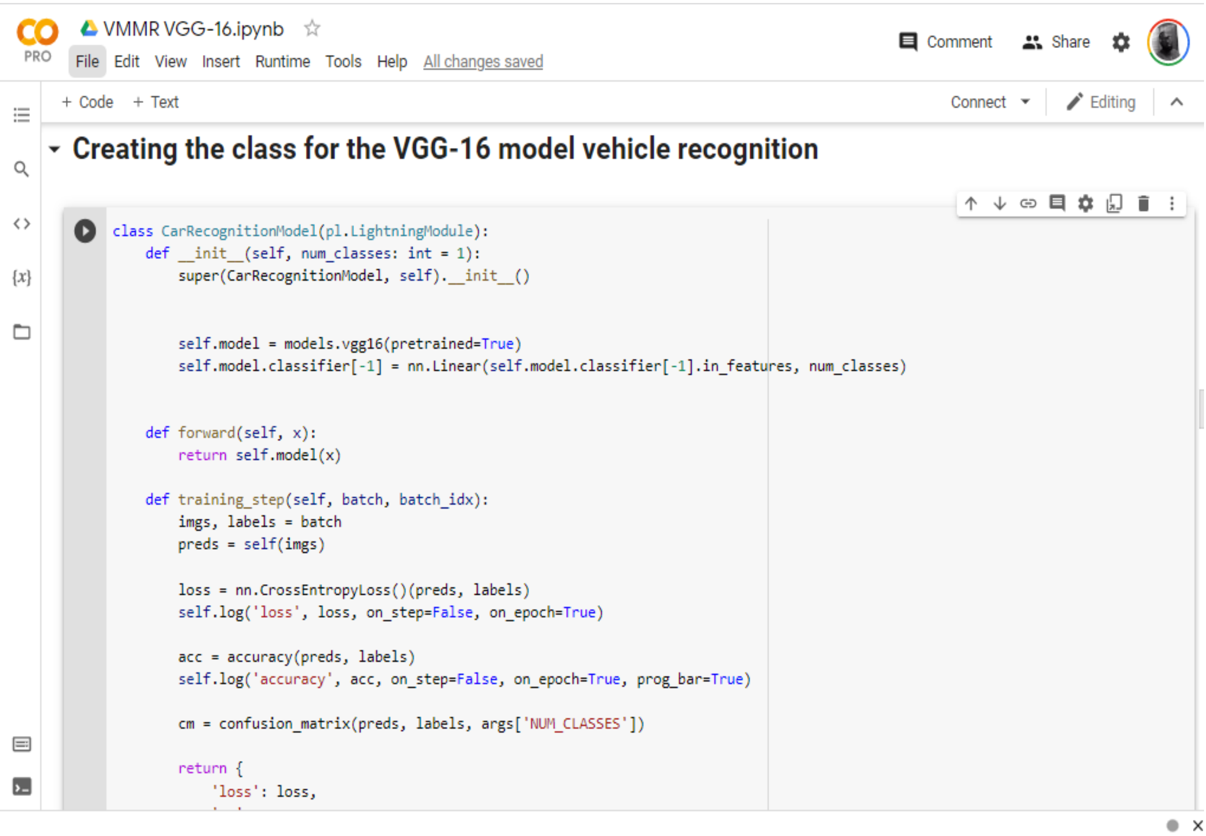

Figure 24: Creating the class for the VGG-16 vehicle recognition model

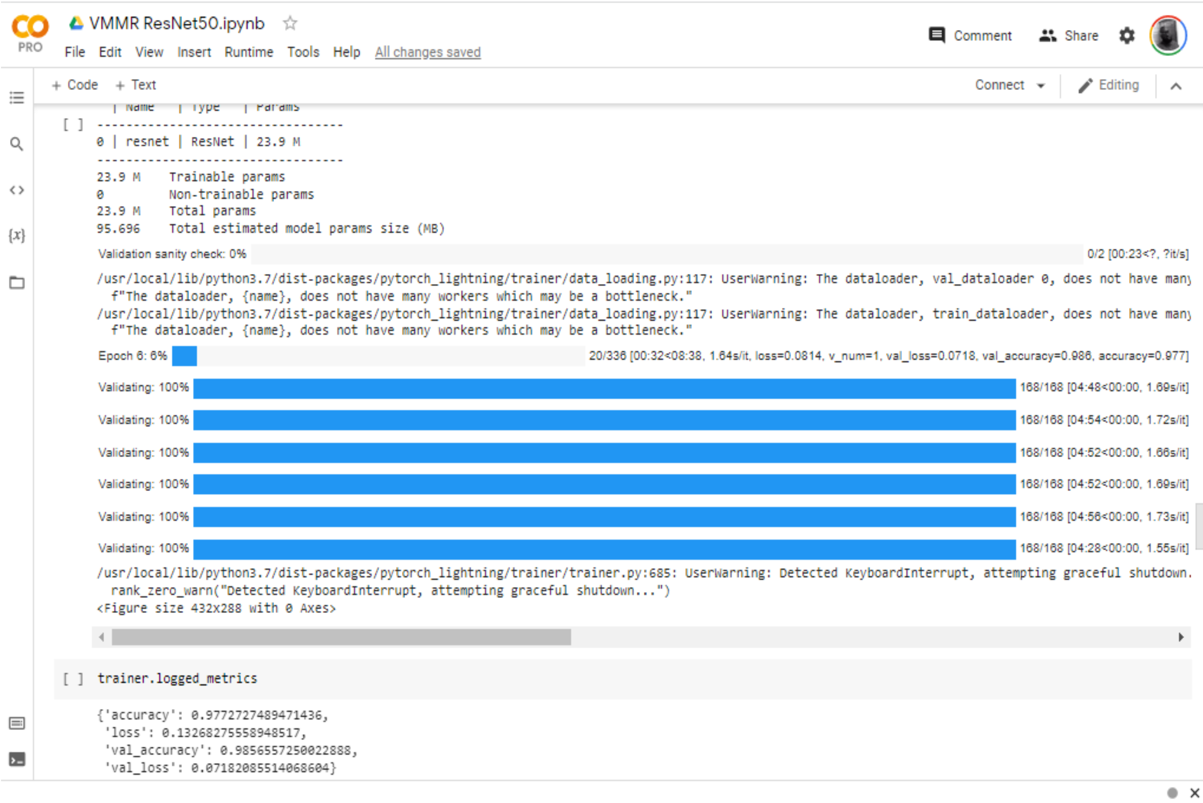

Figure 25: Training the ResNet-50 model

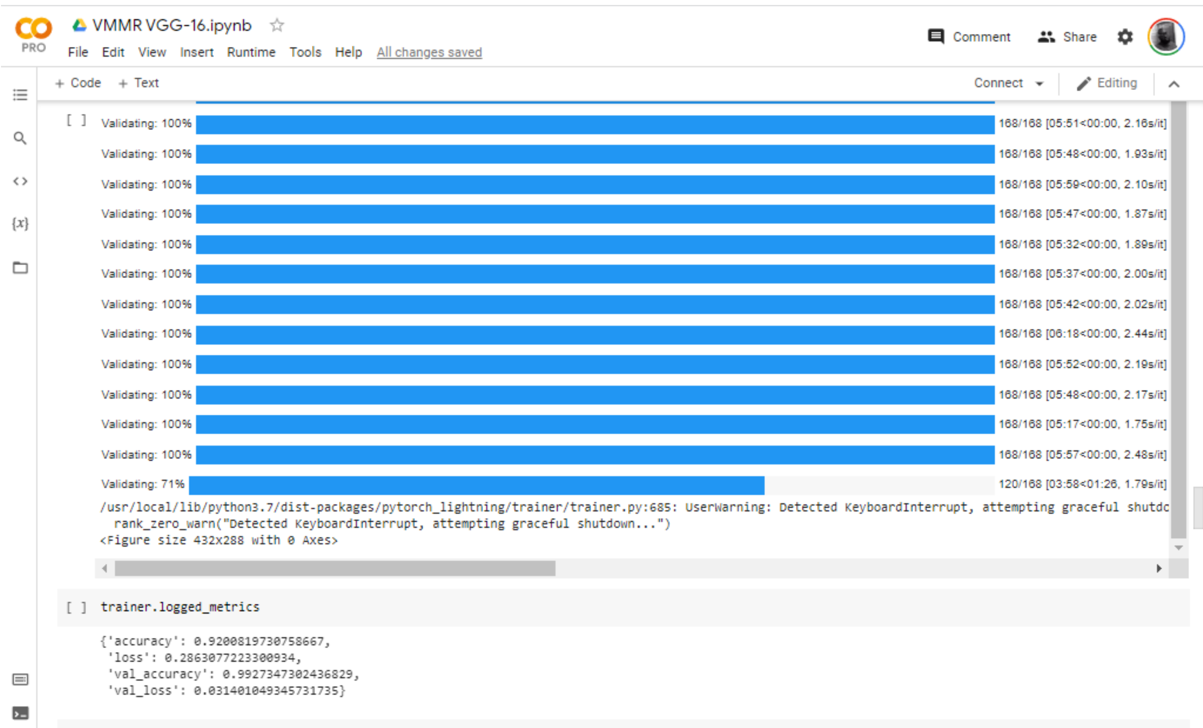

Figure 26: Training the ResNet-50 model

3.5 Running the GUI application

The GUI application is built for user purposes in view of commercialization. The process below explain how to run the application.

STEP 1: Unzip the GUI application on your PC with any unzipping tool
STEP 2: Open up a Command Line interface (CLI).
STEP 3: Make sure the current working directory (CWD) contains the GUI application folder.
STEP 4: Install the requirements by running the line pip install -r requirements.txt in your cli. This is shown in figure 27.

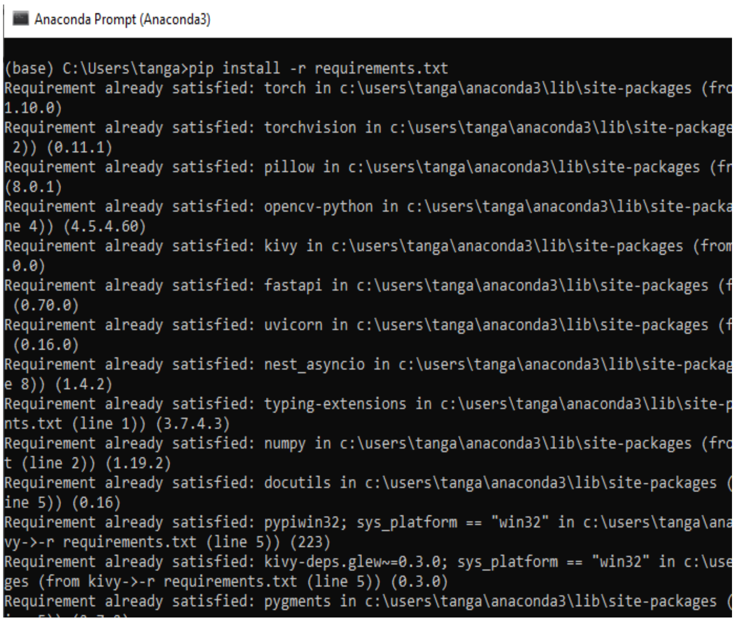

Figure 27: Installing the GUI application requirements

STEP 5: Run the server script by running the line python server.py as shown in figure 28.

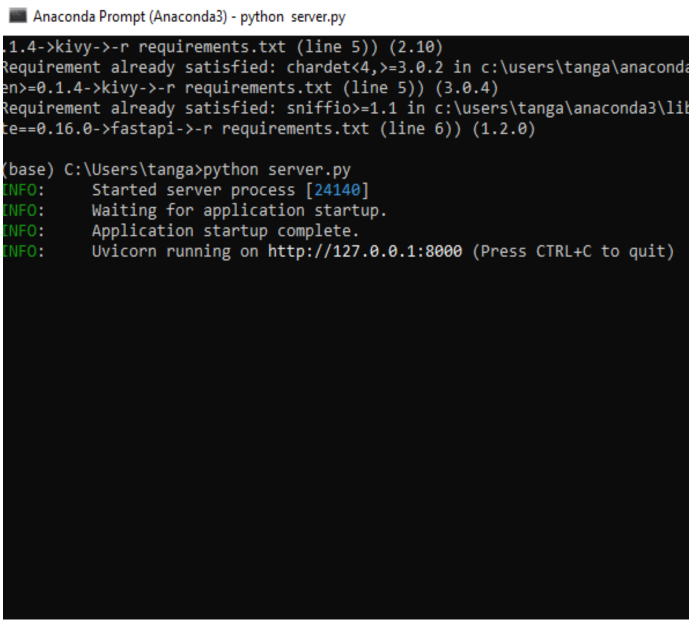

Figure 28: Running the server

STEP 6: Open another CLI window and run the line python main.py to open the GUI application as shown in figure 29

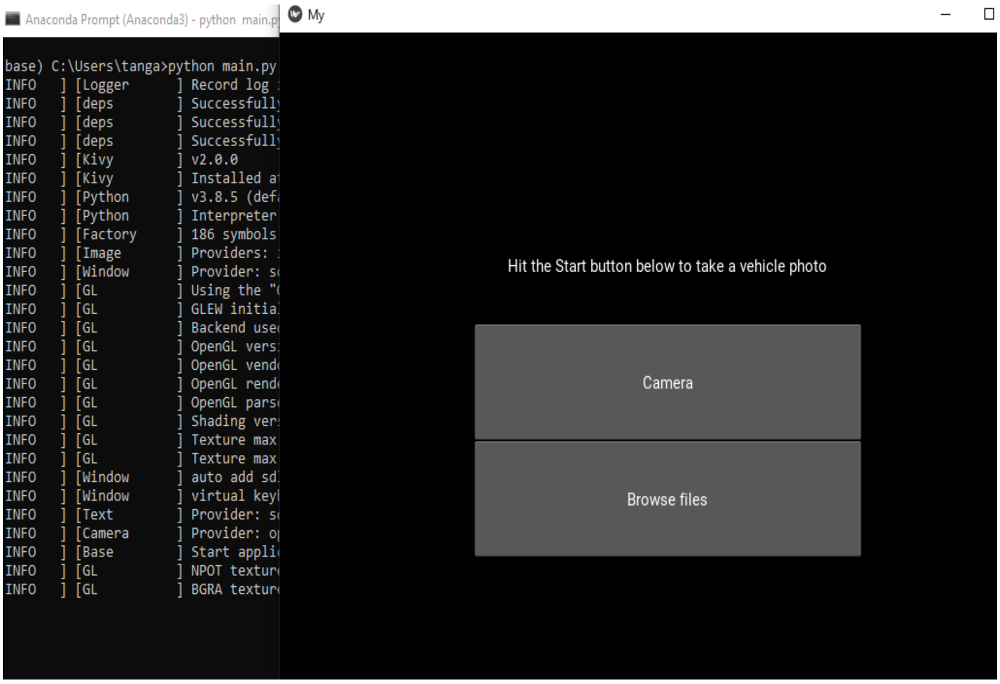

Figure 29: Opening the GUI application

STEP 7: Either choose to capture a new image with the camera or upload an image file. The list of trained classes are in the class_names file.

Output: The GUI application displays the top 3 predictions for the image passed through it as seen in figure

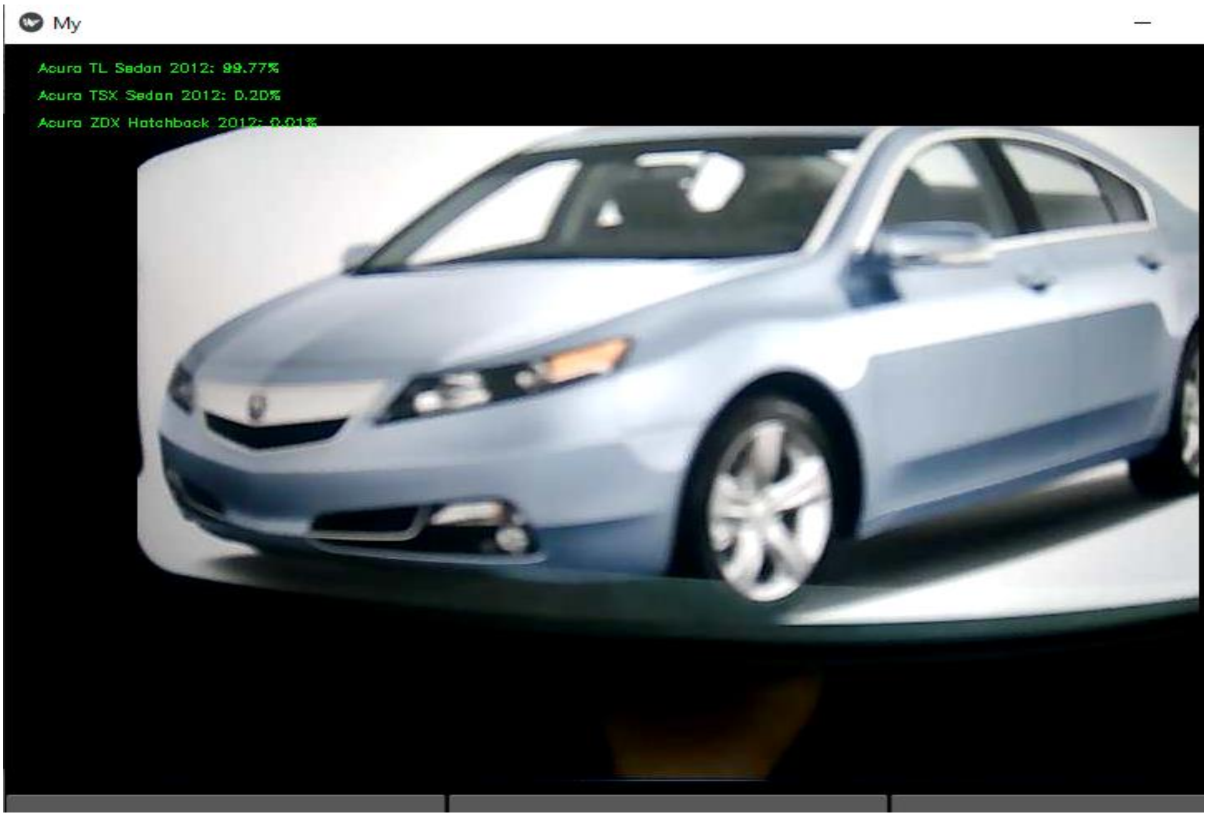

Figure 30: Output showing top 3 predictions testing with an unclear captured image