TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Computer Systems

Irase Ohiomomon Ohiokpehai 196390IASM

Deep Neural Network Based Object Classification on Low Power Edge Computing Hardware

Master Thesis

Supervisor
Mairo Leier
PhD
Secondary Supervisor
Uljana Reinsalu
PhD
Third Supervisor
Olutosin Ajibola Ademola
M.Sc

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia kool
Arvutisüsteemide osakond

Irase Ohiomomon Ohiokpehai 196390IASM

Sügaval närvivõrgul põhinev objektide klassifitseerimine väikese energiatarbega servaarvutuse riistvaral

Magistritöö

Juhendaja
Mairo Leier
PhD
Sekundaarne juhendaja
Uljana Reinsalu
PhD
Kolmas juhendaja
Olutosin Ajibola Ademola
M.Sc

Tallinn 2021

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Irase Ohiomomon Ohiokpehai .....
(signature)

Date: 10 May, 2021

Annotatsioon

Viimasel kümnendil on arvutinägemise valdkonnas tehtud palju edusamme ja selles uuringus kasutatakse ühte väljatöötatud meetodit tegeliku elu probleemide lahendamiseks. Sügav närvivõrk nõuab aga kõrge arvutusvõime ja suure mäluga süsteeme, mis võivad olla piiranguks. Selle uurimistöö eesmärk on välja töötada tõhus madalate kuludega ja energiasäästlik sügava närvivõrgu mudel, mis on võimeline klassifitseerima erinevaid sõidukiklasse, näiteks autosid, kaubikuid, veoautosid, haagiseid, haagiseid, mootorrattaid, busse ja teisi. Selle odava süsteemi saavutamiseks rakendaksime siirdeõpet olemasolevatele mobiilsete sügavate närvivõrkude mudelitele, kasutades kohandatud andmekogumit. Väljatöötatud mudel oleks kasutusel TPU-põhisel Google Corali kiirendi riistvaral. Selles uuringus vaadatakse üle mõned sõidukite klassifitseerimisel praegu rakendatavad algoritmid, seejärel võetakse treenitud närvivõrgu mudel olemasolevate kaaludega, treenitakse seda meie kohandatud andmekogumiga ja kontrollitakse mudelit. Eesmärk on klassifitseerida ja tuvastada korraga rohkem kui üks sõiduk (objekt). See tees illustreerib, millised peamised hüperparameetrid mõjutavad treenitud närvivõrgu mudeli jõudlust.

Abstract

There has been a lot of advancement in the last decade in the field of computer vision and this research makes use of one of the developed methods to solve a real life problem. However, deep neural network require systems with high computational abilities and large memory which could be a constraint. The purpose of this research is developing an efficient low cost and energy efficient deep neural network model that is capable of classifying different classes of vehicles such as cars, vans, lorry, trailer, trailer-trucks, motorcycles, buses and others. In order to achieve this low cost system we would apply transfer learning on existing mobile deep neural network models using a custom data-set. The developed model would be deployed on a TPU based Google Coral accelerator hardware. This research reviews some currently implemented algorithms in the vehicle classification space then proceed to take a trained neural network model with existing weights, train it with our custom dataset and verify the model. The goal is to classify and detect more than one vehicle(object) simultaneously. This thesis illustrates which key hyper parameters affect the performance of a trained neural network model.

Acknowledgements

This work is dedicated to my wife and two children, Uwaye, Nathan and Aide for their encouragement during the period of this research.

I will like to thank my supervisors Dr Mairo Leier, Dr Uljana Reinsalu and Olutosin Ajibola Ademola for their guidance, explanations and technical input through out the course of this work. I really enjoyed working with all of you.

List of abbreviations and terms

API	Application Programming Interface
RAM	Random Access Memory
IDE	Integrated Development Environment
DNN	Deep Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CSV	Comma Separated Values
TF	Tensor Flow
SGD	Stochastic Gradient Descent
RMS	Root Mean Square
v1	Version 1
v2	Version 2

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Background

The study on various methods to classify different classes of vehicles is not new however, we believe advancements made in two very interesting areas of research which are embedded systems and computer vision can be put to good use in solving this problem. The computer vision space is an area that has advanced tremendously since 2012 [1] and the advancement in this field could be used to a large extent with good results to achieve the task at hand. There has also been a lot of development in the field of embedded systems with the introduction of specialized hardware accelerators which are used to speed up the execution of computer vision algorithms[2] such as a CNN object detection model. These convolutional neural networks require specialized hardware for fast execution of deep neural network algorithms because CNN are repeatedly executing the multiple-accumulate(MAC) operations massively and these accelerators can handle the massive matrix multiplications required for neural networks at very fast speeds while consuming much less power.

1.1.1 Motivation

This thesis that is initiated by the larger research project, bridges two very interesting areas of research, computer vision and embedded systems which the writer believes are the bedrock of artificial intelligence hence the reason why working on this topic is alluring. The sudden renaissance of CNN was sparked by the annual Imagenet Large Scale Visual Recognition Challenge(ILSVRC) [3] where different institutions demonstrate new advancement and algorithms to improve large scale object recognition. AlexNet won the Imagenet competition in 2012 [1], since then CNN have been adopted to solve many real life problems which involve object detection and object classification. The choice for using Convolutional neural networks is because they have very good accuracy levels in computer vision field hence the reason why they are widely used in solving Object detection problems including the one this research is solving.

1.1.2 Objectives of this research

The objectives of this work are;

- To collect and prepare a custom dataset from different sources for vehicle classification.
- To select and train an existing object detection model with the new custom dataset(using transfer learning).
- Optimize the model for deployment on an edge based TPU accelerator.
- Test the performance of the model.

It was noticed that in some dataset most vehicles types are broadly grouped under 'cars' however in this research we have created a custom dataset containing specific classes of vehicles such as car, bus, van, truck, lorry, trailer, truck-trailer, motorcycle, long-low and long high. We have also developed an object detection CNN model suited for an embedded device to classify different types of vehicles. As will be later seen in section 3.3 the output model was deployed on a coral edge TPU accelerator and tested to see its performance in detecting and classifying the different vehicle types.

The output of this work can also impact customer charging systems for parking lots where car owners are charged for parking and in ships which are used to move cars where the car owners are sometimes charged different prices for the conveying space based on the type of vehicle. Currently, human attendants are needed to identify and verify the type of vehicles. This problem is one of the reasons this research seeks to develop a low cost solution in which the output model could be integrated into a system with a camera to view the car and classify what type of car it is. Such Proposed system would have to be a low cost energy saving system comprising a hardware with limited number of interfaces to connect the camera and limited processing capabilities to accelerate neural processing in achieving acceptable accuracy in vehicle type detection.

1.2 Literature Review

To the best of our knowledge there is no similar research done previously which entails developing a CNN model for detecting different types of vehicles which runs on an embedded device. This work will develop a model(using transfer learning), validate and optimize it for embedded systems. There are previous researches which have proposed different methods and algorithms one of such employs a sensor which comprises a magnetometer and microphone for vehicle classification and emergency vehicle detection [4]. This

method requires the noise made by the car in order to carry out its classification task which is not suitable for this task since we would also be classifying stationary vehicles at times. One study makes use of principal component analysis(PCA) for vehicle classification, it uses adaptive multi-class principal component analysis [5]. One other method uses an eigen vector which is generated for each car and compared with an earlier generated vector subclass for each vehicle type [6]. Another method makes use of the structural based features of the vehicles to differentiate them [5]. Zhen Dong also developed a method for vehicle classification using a semi-supervised neural network which makes use of the frontal view of vehicles to classify them [7]. Our task will view vehicles from all angles, reason why this method can't be adopted for the current task. Using sparse representation, vehicle classification can also be achieved based on comprehensive sensing, this method was proposed by [8]. One other method combines an artificial neural network and K nearest neighbor techniques in the classification of vehicles, this method employs the use of acoustic signals generated by moving vehicles to classify them [9]. One other paper developed a method for vehicle identification from visual based dimension estimation, a vehicle mask is generated which is basically the binary representation of the vehicle gotten from a calibrated camera by getting the 2D projection of a simple 3D model [10]. Ma used edge based features in determining the class of vehicles. This style augments edge points to repeatable and discriminate features from a fixed angle when the vehicles are viewed [11]. [12] developed a novel vehicle matching algorithm that computes and gives two vehicle observation of either being the same or different. A support vector machine model has also been used to propose a method for vehicle classification, it was achieved by generating eigen vectors from the frontal view of the vehicles [13].

1.3 Overview

Some of the above mentioned methods showed good results however they are not suitable for the current task since we require a low energy saving and low cost device which will not be able to provide the required computational power and resources required by most of these methods to achieve accurate results. Our method employs a small and efficient neural network to tackle the vehicle classification problem because we would like to fit our solution into a low cost embedded device. For this reason, MobileNet and MobileDet, which are convolutional neural network models were selected because these model characteristics closely align with the objective in this vision application project.

Convolutional neural networks is not a recent discovery of the millennial but was highlighted by a lot of research groups in the 1980s who came up with a lot of prepositions on how the CNN could be used in solving image recognition problems[1]. It did not have as much success and attention in implementation in the earlier years because at the time there

were some constraints such as limited computational resources and thus the computational power needed by neural network was unavailable. However, this narrative has changed since 2012 and Deep neural networks have had the best results in solving image recognition problems and will continue to get all the attention and focus by most research groups until another method better than CNN is probably brought to fore for solving image vision task.

For persons not so familiar or hearing about convolutional neural network for the first time,the question to be addressed is what is a convolutional neural network and what does it comprise of. It is important to note that the whole essence of neural networks (artificial neural networks CNN) is they are designed to be similar as the human brain. The human brain is made up of individual cells called neurons and these neurons learn by processing information through the human five senses but majorly through vision. In similar fashion, CNN comprises of neurons connected together to form a network with a primary function of pattern recognition[14]. a basic network of neurons is a feed-forward network which has an input layer, hidden layer and an output layer as shown in figure 1 however CNN are far more complicated than this. As earlier mentioned the network of neurons is set up to learn pattern recognition and the learning pattern could either by supervised learning or unsupervised learning[14].

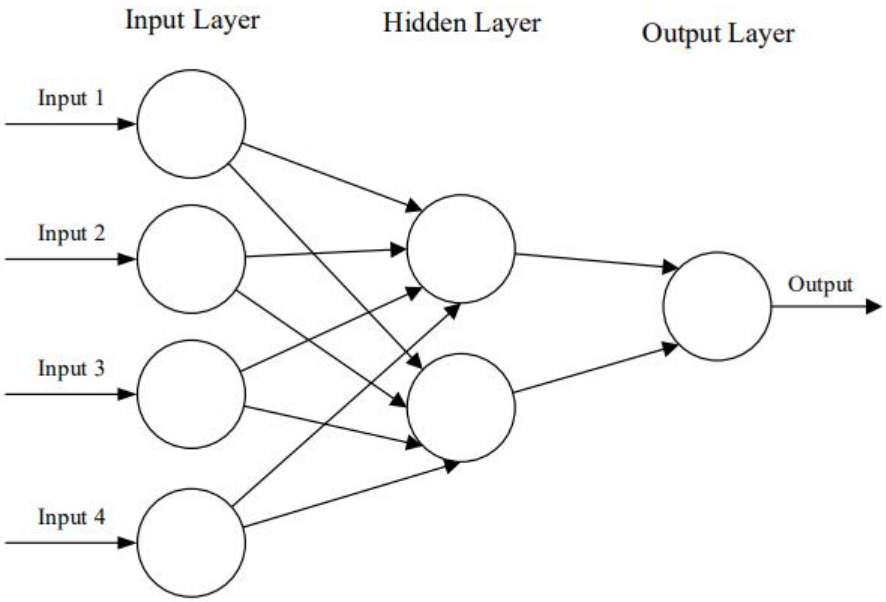

Figure 1. Basic Feedforward Neural Network[14]

For supervised learning, the input to the network is labelled, we could assume our network of neurons is a child learning about different type of cars, where each car type is shown to the network until its able to spot the difference between the various car classes fed to it. The expected end result of the learning process is that when a picture which was not

among the labelled ones is shown to it, it should be able to predict the car type with the least acceptable error rate[14]. Its also important to note that overfitting should be avoided while training, this issue arises when the neurons are trained in such a way that they are not flexible enough to make correct predictions outside the labels which it has been trained with.

Convolutional neural networks contain neurons which self optimize by learning[14]. As regards the architecture of CNN, they have three layers which include , the convolutional layer, the pooling layer and the fully connected layer as shown in figure 2. The convolutional layer receives the pixel values from the input and extracts feature maps from the images, the pooling layer downsizes the extracted samples while the fully connected layer attempts to give class points for regression and classification[14].

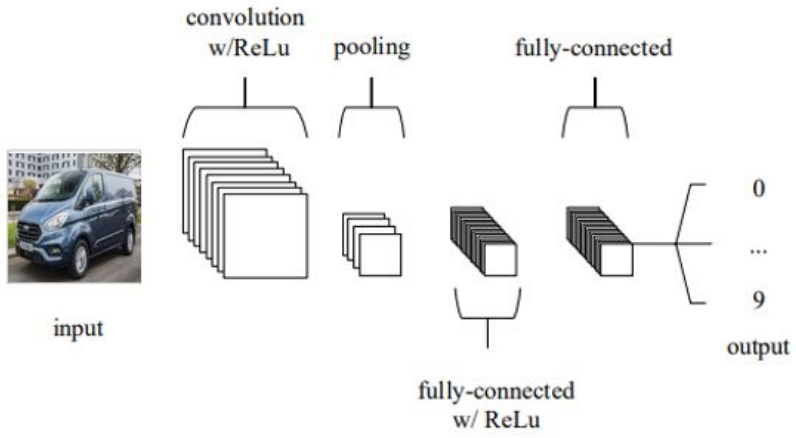

Figure 2. Basic CNN Architecture[14]

[15] showed that the input into a convolution layer which is an image can be represented as a product of three inputs which are, the image's width w, the image's height h, and the color channels which will be represented as P_{in}. Therefore the input into a convolution layer can be shown as $P_{in} \times w \times \mathrm{h}$. Then after the convolution is applied on the input the output which will have a size of $\mathrm{P}_{\text{out}} \times w_{\text{out}} \times h_{\text{out}}$ can be shown as,

$$\text{out}\left(P_{\text{out}_j}\right) = \text{bias}\left(P_{\text{out}_j}\right) + \sum_{k}^{P_{\text{im}}-1} \text{weight}\ \left(P_{\text{out}_j}, k\right) \star \text{input}\ (k) \tag{1.1}$$

The architecture of the models we selected, MobileDet and MobileNet are based on convolutional neural networks. Also both models were developed for mobile devices taking into account the limited resources of computation and size of mobile devices.

MobileNets provides a good network architecture and a set of two hyper parameters for low latency and small models that can be fitted in the design for embedded system vision application [16]. These models are smaller when compared with other models such as YOLO and SSD restnet [17]. MobileNet and MobileDet are also efficient for visual tasks and will suit this work [17].

2. Method

In this chapter the applications and tools used to carry out the experiments are briefly reviewed. The overall process and evaluation parameters are also explained.

2.1 Overview of tools and applications

Convolutional neural networks provide the best results when compared to other classification techniques as demonstrated by [1]. hence the reason why for this implementation we have chosen to use CNN as the algorithm for classification of the various classes of vehicles. A new convolutional neural network architecture will not be developed, however the road map is to take an existing object classification model with very good accuracy and train this model on a newly created dataset that has been developed uniquely for this research. The model chosen for the purpose of transfer learning is the Mobilnet model proposed by [16] and MobileDet proposed by [18] both from Google. MobileNet is a deep neural network model based on the architecture developed by [16]. The chosen approach is supervised learning based on transfer learning. The chosen models were created specifically for mobile and embedded systems at the cost of a minute decrement in accuracy because of the reduction in size of the models[16][18].

Currently there exists different datasets such as Coco and Imagenet however these datasets do not differentiate the different classes of vehicles hence there was need to create a custom dataset which would capture the different classes of vehicles. The dataset seperates vehicles into cars, truck, van, trucks, truck trailer, bus, log-low, long-high, lorry, motorbikes and persons, 11 classes in all. For the creation of the dataset to increase the supervised learning rate during transfer learning annotation of the dataset is carried out. Upon completion of the training, the models were optimzed for deployment on the accelerator. The chosen accelerator for the task, the Coral USB accelerator. The optimized neural network models which have been trained with the new data set and deployed on this device for testing and inference.

Its important to point out that there are two aspects to be dealt with, one is the training of the network and the second is inference. The first aspect, which is training the network involves feeding a lot of labelled images to the deep neural network which can be termed

forward projection to enable the network arrive at its initial model weight parameters after which the adopted weights are adjusted based on how well the predictions go, this could be termed back propagation[19].The second aspect is inference which involves using a trained model with its weight to carry out predictions on a set of new inputs. The former is mostly carried out on CPU and GPU while accelerators are specially designed for the latter.

2.1.1 Neural Network Models

For the task at hand we have a constraint of power and computational capacity because the solution is to be deployed on an embedded device which does not possess the computational capacity of a traditional central processing unit hence this is the reason for the use of MobileNet and MobileDet as the chosen models. MobileNets neural network was specifically designed for deployment on embedded and mobile devices[16].

The concept behind MobileNet is making use of depthwise seperable convolutional blocks instead of expensive convolutional layers[20] as depicted in figure 3. For the development of MobileNet, instead of following the general trend of developing a deeper and more complicated architecture a neural network and a set of two hyper parameters were developed to come up with a low latency minute model that can be easily deployed on a mobile or an embedded device[16]. BN stands for batch normlization which is a technique that is used to address the issue of internal covariate shift when training convolutional neural networks[21]. Batch normalization acts as a standardizer by regularizing the inputs to a layer for every mini-batch[21]. It also helps to reduce the number of training steps required for training the neural network[21]. ReLU stands for rectified linear unit[22]. It is an activation function employed in building alot of CNN architectures[22]. The seleceted activation function is a key parameter to the success of training a neural network[22]. This activation function enables gradients to flow when it has an input that is positive[22]. The MobileDet model has a similar neural network architecture with MobileNet as depicted in figure 3 however the difference for MobileDet is its first 1 X 1 convolution is joined with its subsequent depthwise convolution to form a regular convolution[18].

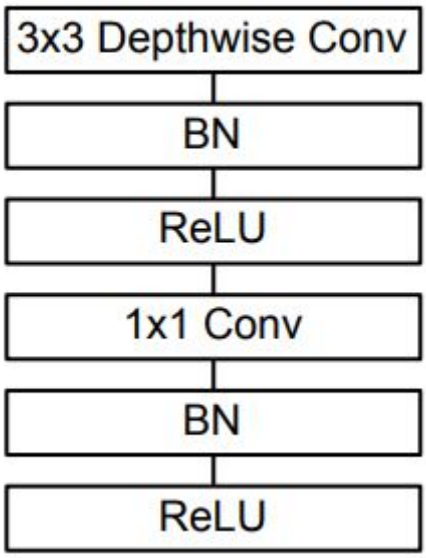

Figure 3. Depthwise Separable Convolution[16]

To achieve object detection task of this work both models are fused with the one stage single shot detector(SSD) which will enable the placement of the bounding boxes on the model's prediction during inference[18]. Two stage detectors such as faster RCNN will be too slow for this task[18].

2.1.2 Method of Optimization

DNN has shown the best result for computer vision task and more recent works keep producing deeper networks with layers which can vary from 7 layers to hundreds or even much more[23].

A deep neural network could require large amount of megabytes to store weights, also during inferencing billions of floating points calculation are involved[23] hence the embedded device in the scope of this project will have several constraints such as not been able to provide the computational power required for such calculations and the device does not have the storage space to warehouse such large network hence researchers have come up with several methods which has enabled the resizing of the DNN model to a smaller size.

One of such method which is used for reduction of model size which is employed for this implementation is quantization which instead of making use of 32 bit floating point, 8bit floating points are used for weights and activation[23].

Studies have also shown that the datatype in which the model weights are computed and stored affects the training, performance and efficiency of the developed deep neural network[19]. Datatype such as 32 bit floating point are mostly used during training while lower numeric precision datatype could be adopted during inference[19].

Figure 4 shows the range of the model size after quantization of the trained neural network model. This enabled the deployment of the model on the coral USB accelerator. The size of most models before quantization during the experiments was about 15MB however after quantization model sizes reduced to less than 5MB.

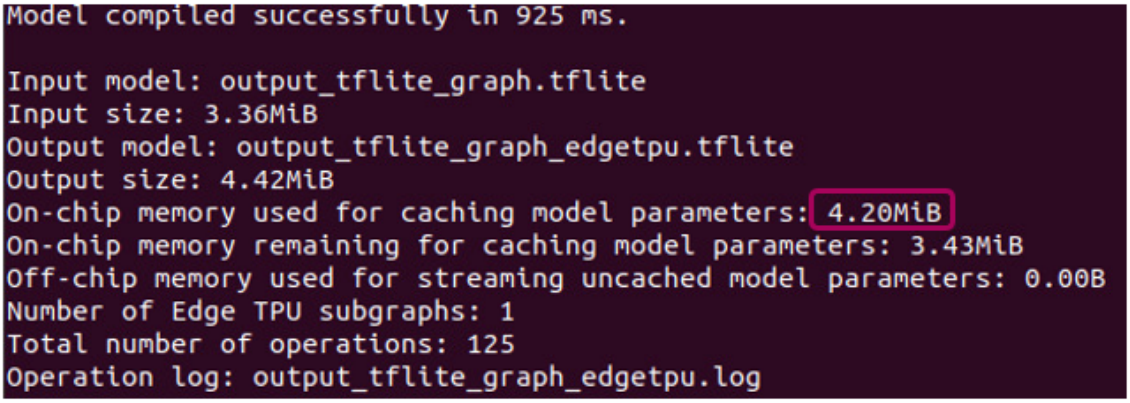

Figure 4. Model Size After Quantization

2.1.3 Google Coral TPU Accelerator

In achieving the goal of this project to develop a neural network model for vehicle classification small enough in size to fit into an embedded device with low power consumption, a specialized embedded hardware which is designed primarily for handling deep neural network is required because this will help in achieving better performance of the trained model during testing and usage[2]. The reason why we can't use just any type of embedded hardware is because deep neural network algorithms carry out massive multiply and accumulate operations (MAC) which deep neural network accelerators are designed specially for. In order to deal with the huge amount of MAC operations these accelerators possess massively parallel processing engines, each parallel processing engine has a multiplier and adder in it arithmetic and logic unit (ALU)[2]. Most of the MAC operations are carried out in the convolutional layer and fully connected layer of the DNN[19].

Figure 5. Google Coral TPU Accelerator[24]

The accelerator used for this implementation is the Google Coral TPU accelerator. The Coral TPU accelerator provides edge TPU as a co-processor attached to a computer. This accelerator helps with speeding up the inferencing of the trained neural network model however it only supports tensor flow lite models. The Coral TPU accelerator is a USB 3.0 Type-C socket that is supported on Windows, Mac and Linux operating systems. It is an application specific integrated circuit that is capable of performing four trillion operations per second that was designed by Google to accelerate the inferencing of TensorFlow lite models. The benefit of using the TPU accelerator is that it enhances local processing and reduces latency as there is no need for constant internet.

2.1.4 Dataset Preparation

In renaissance and success of convolutional neural network in the computer vision field the improvement and quality of the current datsets available for training these neural networks have played a major role. The key success of deep neural networks is their ability to learn from a lot of labelled images[25]. CNN which have been pre-trained on large dataset such as ILSVRC have been seen to have very good results[25] This goes to show that our dataset also determines how well the trained model will perform. Annotation involves placing bounding boxes in a picture which contains the class or classes the model is meant to detect after its been trained[26].

As earlier mentioned in section 1.2, to the best of our knowledge this research is novel because our model is trained specifically for classifying different classes of vehicles on an embedded device. Most of the datasets available have all vehicles classified under the broad category of cars, bus and truck but ours captures more vehicle categories. In order to achieve the goal of developing a trained model, a new custom dataset was developed by getting pictures from google web and all from a private source. A total of 42,000 images formed the custom data set, all annotated to reflect the 11 classes as shown in figure 6. The classes of vehicles captured in the dataset are 10 and they include car, bus, van, truck, machine, long-low, long-high, trucktrailer, trailer, motorbike and lorry including person.

The data set was split in a ratio of 70 percent to 30 percent respectively, 70 percent was used for training while 30 percent of the images was employed for evaluation of the trained model.

Figure 6. 11 Classes in the Custom Dataset

2.1.5 Annotation of Dataset

The annotating tool used in this research was LabelImg[27]. It was developed to be used in a python IDE. LabelImg was used in this research to place bounding boxes on all the identified 11 classes in the pictures of the custom data set. The below figure 7 shows what the LabelImg application looks like when launched and how the custom dataset was annotated.

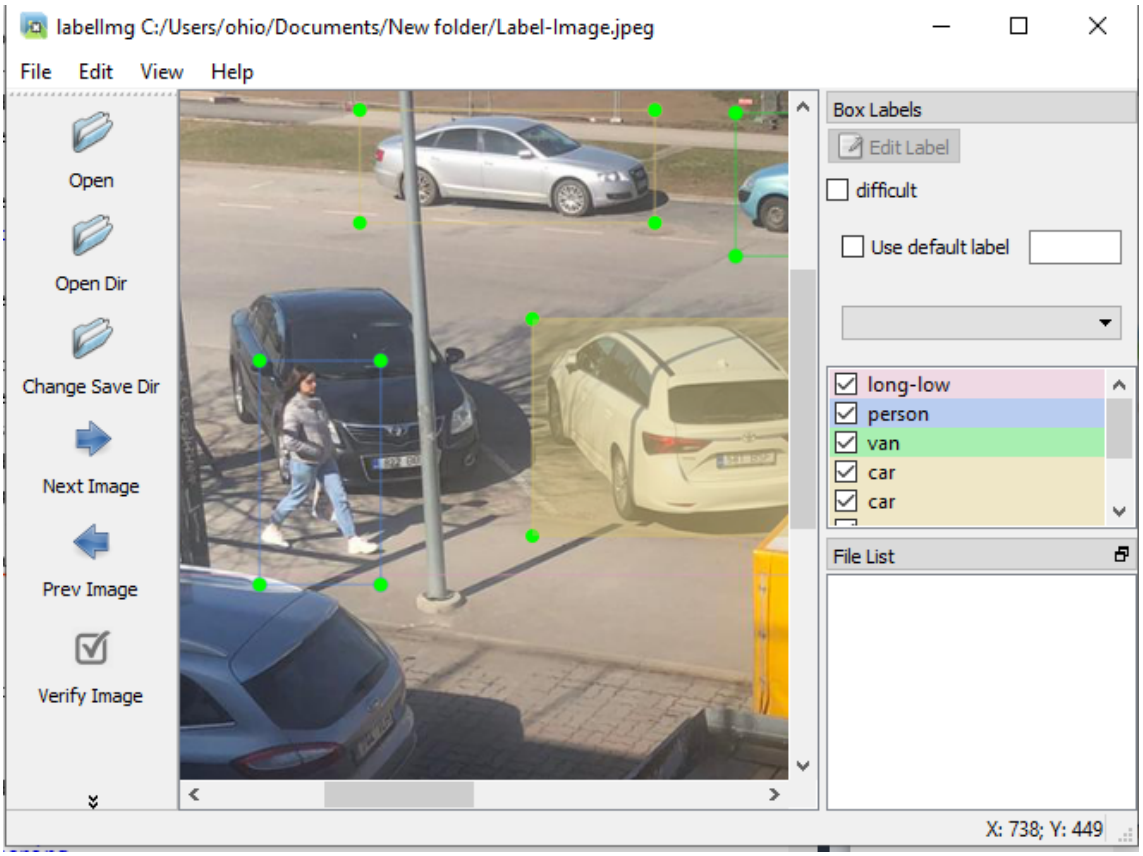

Figure 7. LabelImg annotating tool[27]

As annotations involve placing bounding boxes on identified classes[26] in order to have excellent results during inference all the 42,000 images were annotated using LabelImg before commencement of the training phase.

2.1.6 Gradient Descent Optimization Algorithms

Gradient descent is one of the most common known way of optimizing neural networks[28]. Different algorithms exists which are employed to optimize gradient descent for neural networks[28]. Two of such algorithms are Momentum and RMSprop which are used for optimizing stochastic gradient descent in this research.

Momentum is represented by equation 2.1, where $J(\theta)$ is the objective function, (η) is the learning rate and (γ) is the momentum term. Momentum helps SGD to head in the right direction and dampens oscillation while the learning rate (η) determines the size of step taken[28].

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.1}$$

RMSprop can be represented by equation 2.2. RMSprop is an adaptive optimizer, it works by adapting the learning rate (η) based on the parameters. 0.9 represents (β) in equation 2.2 a value suggested by Hinton [28]. $E\left[g^2\right]$ represents the moving average of the squared gradient and $\frac{\delta C}{\delta w}$ represents the differentiation of the cost function with respect to the weight.

$$\begin{aligned} E\left[g^2\right]_t &= 0.9 * E\left[g^2\right]_{t-1} + (1 - 0.9)\left(\frac{\delta C}{\delta w}\right)^2 \\ w_t &= w_{t-1} - \frac{\eta}{\sqrt{E\left[g^2\right]_t}} \frac{\delta C}{\delta w} \end{aligned} \tag{2.2}$$

2.2 Deep Neural Network Frameworks

Due to the complicated nature of convolutional neural networks special frameworks or environments are required for their set up, training and testing.In this section we review some of the most popular DNN frameworks available for working with neural network models. Examples of such frameworks specially suited for deep neural network processes include;

- Pytorch - This is an open source deep neural network framework developed by Facebook for performing functions such as implementing new CNN architectures and training of CNN models[29]. Some benefits of using Pytorch is that it supports APIs for programming languages in Python, Java and C++[30][29]. It also offers

flexibility during the creation of a neural network architecture[30]. Pytorch also allows the training of a neural network model on distributed environments and also allows the user to work with multiple GPUs[29]. Pytorch can be run on both CPU and GPU[29].

- Caffe(Convolutional Architecture for Fast Feauture Embedding) - is also an open source state of the art deep neural network framework developed by scientist from UC Berkeley[31]. Caffe was built as an integrated toolkit for testing, training, fine-tuning and implementation of models[31]. It was originally built in C++ but also has a python interface[30]. Some benefits of Caffe is that it has well documented illustration for training and fine-tuning neural network models[31]. It has a feature that allows for the binding of Python and MATLAB, a benefit for researcher who want to work with both languages[31]. It also supports training for both CPU and GPU.
- MXNet - It is a deep neural network library developed by Apache software foundation. This library is supported by web service providers such as Azure(Microsoft) and Amazon Web Service(AWS)[30]. This DNN library core language is C++ however MXNet supports as much as 8 languages and they include Scala, Julia, Python, R, Lua and Go[32]. One benifit of MXNet is its coverage of more programming languages as it offers the flexibility of exporting trained models for use in different languages[30].
- TensorFlow - The TensorFlow library is an open source application developed by Google for working with deep neural networks[33]. Tensor flow supports deep learning APIs such as Keras. Its an environment where different libraries, dependencies and algorithms such as numpy, pandas are made available for the special type of numerical computation required by deep neural network[34]. TensorFlow supports distributed training and also supports the use of multiple GPUs[33]. TensorFlow was designed primarily for machine learning system that run on large scale and in variegated environments[35]. The possibility of being able to map the nodes of the dataflow graph of TensorFlow across different computational devices running on one system such as a system which has GPUs, CPUs with multicore or a system running on a cluster provides flexibility to developers who want to use it[35].

2.2.1 Selected Framework

The framework used for this research was TensorFlow. TensorFlow supports training on CPU and GPU. One of the reason why this platform was selected was because TensorFlow provides a wide range of pre-trained quantized aware models trained on similar dataset made available in the TensorFlow zoo. The TensorFlow API is also flexible and easy to use. It also provided a monitoring interface, Tensor Board, which was used in monitoring

the training process. This made it easy to track changes during the experiments carried out during this research. There are currently two versions of Tensor flow, version 1 and 2. The version 1.15 of Tensoflow was used because more quantized pre-trained models are available for TensorFlow 1 version.

2.3 Overview of process

One of the early steps taken during this work in order to get good results involved gathering various pictures to form the custom dataset. The pictures contained in the data set were acquired from google web pictures and other private sources using an edge based camera. Data preparation is the most time consuming aspect when dealing with the training of convolutional neural network so this was the first aspect tackled. It also involved annotating the pictures in order to capture the 11 classes to be predicted in this work. Then separating the 42,000 annotated pictures into a training and test folder respectively. Training had 70 percent while 30 percent was separated for evaluation. The pictures were annotated using LabelImg. Figure 8 shows the data preparation process.

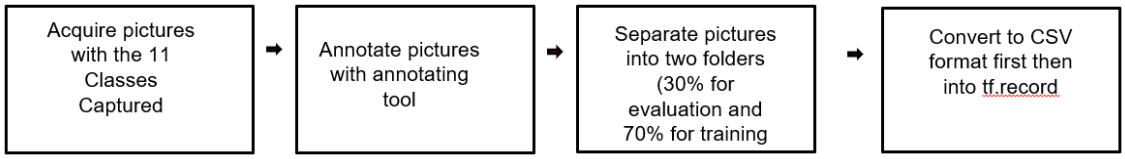

Figure 8. Data preparation

The experiments were carried out in python environment, version 3.6. For the embedded device used during this thesis, Google Coral TPU accelerator, it is only compatible with models saved in the edge TPU tflite format. The edge TPU tflite files used were gotten from models trained on the Tensorflow neural network framework.

The method of approach used was transfer learning which means selecting a pre-trained neural network model which already has defined weights. Training it with a new custom dataset that has been annotated showing the different classes of vehicles. Since its Tensorflow API being used, models which had been pre-trained on the tensor flow API were chosen.

Three quantized-aware object detection models were used to carry out the research and they are SSD-MobileNet V1, SSD-MobileNet V2 and SSD-MobileDet. These models were chosen because when optimized and compiled their sizes are small enough to fit into the coral usb-accelerator. The checkpoint files of these three models are available on the coral website and could also be gotten from the TensorFlow zoo. All 3 pretrained models

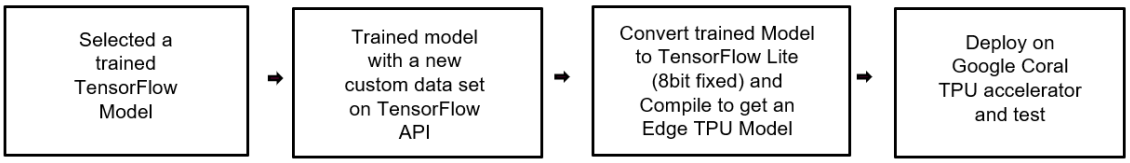

Figure 9. Conversion to tflite format[24]

were initially trained for detecting 90 objects on the coco dataset.

The next phase was preparing the Tensorflow API on a Linux operating system(Ubuntu) with 12GB RAM and later on Google colab with 15GB RAM. The set up of the Tensorflow API in both environments involved installing the required packages and libraries needed to train and carry out inference on a neural network. Packages such as numpy, tfslim, matplotlib were required for the training of the imported pre-trained models. The next phase involved converting the annotated files into CSV format and then the CSV files were converted to tf.record files. tf record is the file format required by the tensor flow models to receive the files for training.

Each model's pipeline configuration files were edited to reflect the new environment path and respective classes. Other parameters which were configured during the experiment for each model were the batch size, checkpoint path, training record and other hyper-parameters. Its ensured that the hyper-parameter changes were documented for tracking result during the experiment.

The first experiment involved training the last layers of the pretrained model to see the result. The second experiment involved training all the layers of the models in order to obtain lower loss values. The outcome is shown in Chapter 3. Once training of each selected model was completed, each model's checkpoint files was converted to the frozen graph and .pb file format.

The MobileNet SSD version 1, version 2 and MobileDet used for this experiment are quantized aware versions. The trained output files were converted to the edge TPU tflite format because the chosen hardware for inference testing, coral TPU accelerator, which only supports edge TPU tensor flow lite model format. Once the tensorflow model was compiled, it was deployed on the TPU accelerator. The below figure 9 shows the process flow.

2.4 Model Evaluation Parameters

For the evaluation of the models trained with the custom data set there are a few parameters which will be used as a metric to know how good or how well the trained model turned out.

2.4.1 Loss

In the field of computer vision when evaluating object detection models one parameter is the loss. As mentioned by [15], the loss function can be represented as shown in equation 2.3

$$\text{loss } = \sum_{i-1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{2.3}$$

where the loss function is a binary cross entropy for convolutional neural network. The loss function is adjusted by a process called back propagation [15]. Back propagation can be defined as an algorithm which is used to get the gradient of the loss function based on the input variables[36]. The adjustment of the loss function is an iterative process,as the weights of the model are continuously adjusted until the loss is minimum[15].

2.4.2 Precision

Another way to evaluate the performance of a trained neural network object detection model is to the calculate the precision. The precision can be determined by comparing the number of positive matches between the reference objects and the output objects [37]. The quality of matching for a model can be computed using equation 2.4

$$\text{precision } = \frac{\#\text{ of correctly detected objects}}{\#\text{ of all detected objects}} = \frac{N_D - UM}{N_D} \tag{2.4}$$

Where N_D is the number of all detected objects and UM is the number of unmatched objects in the output of the model[37][38]. The mean average precision(mAP) is computed at the end of each model training based on an intersection over union threshold of ratio 0.5:0.95, 0.50 and 0.75. IOU thresholds are used to determine the average precision evaluation results of a neural network model during its training.

2.4.3 Intersection over Union

Intersection over union is used to measure the level of overlap between two bounding boxes. The first bounding box represents the one predicted by the trained model and the second bounding box is the object's real bounding box[39]. This is also one of the hyper parameters set up in the architecture of the convolutional neural network before training is commenced. This parameter is captured as the 'IoU' threshold.

Intersection over union can be computed using equation 2.5

$$IoU = \frac{TP}{FP + TP + FN} \tag{2.5}$$

where TP is true positive, FP is false positive and FN is false negative [39]. True positive corresponds with correctly matched objects as discussed in 2.4.2, false positive represents the number of unmatched objects in the output of the model[38].

3. Experiments and Results

In this chapter, the experiments and results of this research will be outlined. Table 1 outlines the models considered for this section. SSD MobileNet version 1 and 2 have speeds that are suitable for this research and their quantized aware models were available including ModileDet which was built for mobile devices and also incorporates full convolution in its architecture [18]. The availability of a quantized aware model was an important factor taken into consideration during the selection of models because after training, the quantized aware models can be deployed on an embedded device with limited space.

Table 1. Potential Models For Experiments[40]

MODEL	INFERENCE SPEED(ms)	mAP	QUANTIZED AWARE
EfficientDet	39	33.6	Not yet
SSD MobileNet v1	29	18	Yes
SSD MobileNet v2	29	22	Yes
SSDLite MobileNet v2	27	22	Not yet
SSD MobileNet v3	43	15.4	Not yet
SSD MobileDet	113	24	Yes

List of selected convolutional neural network models for experiments;
i) MobileNet SSD v1
ii) MobileNet SSD v2
iii)MobileDet SSD

The above listed models were selected for the experiments in this research. The quantization of a non quantized model is outside the scope of this research. The training parameters such as number of training steps, evaluation steps and the dataset used were all the same for all three highlighted models during the first two phases of testing. The approach adopted for the test was to first train the last layers of the model with the new custom data set, this idea was gotten from the coral website[24] which recommended 500 to 1000 training steps for training the last layers. The Coral website also recommended at least 50,000 training steps for training all the layers of the model.

3.1 Experiment and Result of Training Last Layers of Models

The first experiment carried out was training the models with the number of training steps set at 1000 and evaluation step 100. This was done in order to train only the last layers of the model with the new dataset. The below results shows what was obtained for all the three chosen models

3.1.1 MobileNet SSD v1 Last layers Training

As shown in figure 10 after 1000 steps the loss was still above 10 and the goal is to achieve a loss that is less than 1 to get good results. With this outcome, it shows the training steps for training MobileNet SSD v1 needed to be incremented further in order to achieve a loss far less than the one achieved.

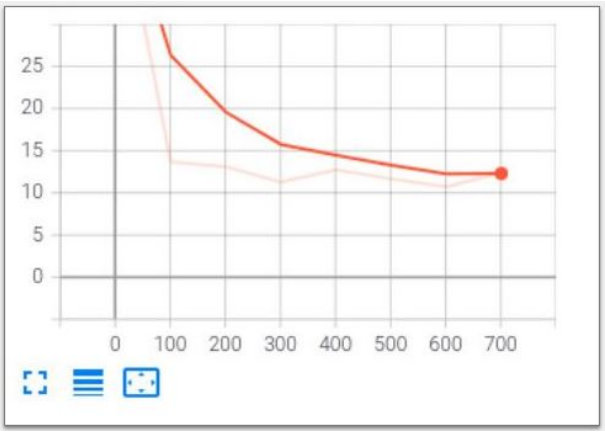

Figure 10. MobileNet SSD v1 last layer training Loss graph

3.1.2 MobileNet SSD v2 Last Layers Training

The outcome of training MobileNet SSD v2 for 1000 training steps is shown in figure 11 where the loss was at 7 at the end of the training. It also showed that more training steps were needed to drive the loss value lower.

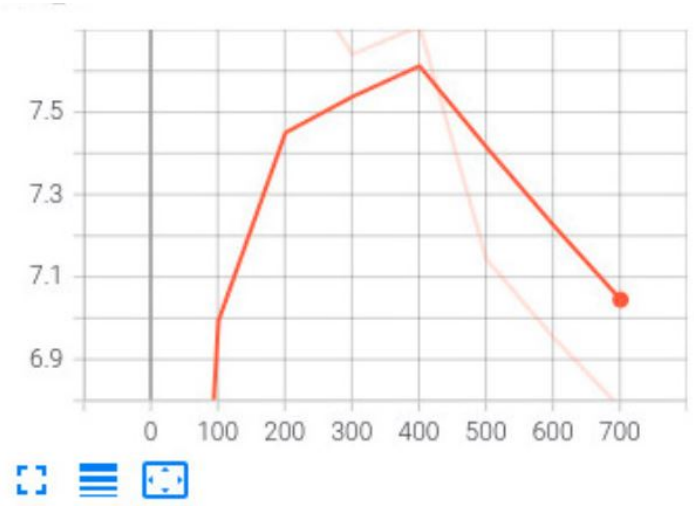

Figure 11. MobileNet SSD v2 Last layer training

3.1.3 MobileDet Last Layer Training

For MobileDet SSD model at the end of 1000 training steps the loss was not near zero however the loss was going downwards which was the desired trend as shown in figure 12.

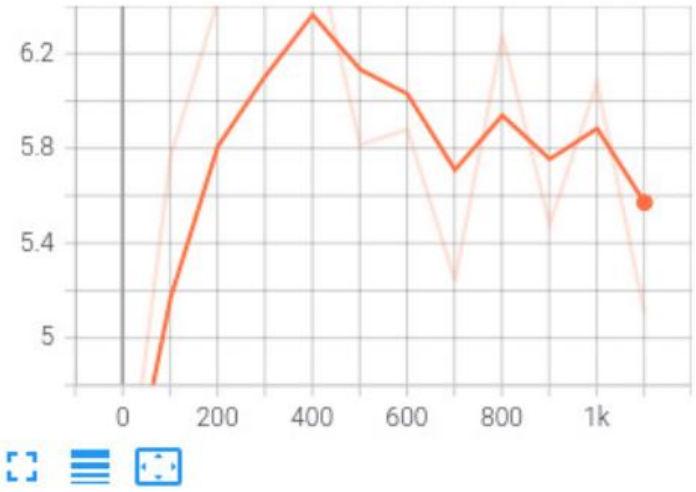

Figure 12. MobileDet Last layer training

3.1.4 Result Table of Last Layers Training

The Table 2 below shows a comparison between the three models result after training their last layers. It can be seen that the loss value and mean average precision values for all models are not optimum. It was concluded that more training was required to achieve better results.

Table 2. Result of last layer training for all models

MODEL	TOTAL LOSS	mAP
MobileNet v1	11.2	0.001420
MobileNet v2	7.03	0.000933
MobileDet	5.6	0.000047

3.2 Training All Layers

The below table 3 shows some hyper parameters for the three models before commencing the training of all the layers of the model. Most parameters highlighted in table 3 were the default parameters captured in the three models pipeline configuration files when they were trained to detect 90 objects in the Coco dataset. No changes were made aside the reduction in the batch size because of the RAM capacity of the local system the TensorFlow API environment was hosted on.

Table 3. All Layers Training Default Parameters

MODEL	STEPS	LEARNING RATE	OPTIMIZER	IoU THRESHOLD	BATCH SIZE
MobileNetv1	50000	0.2	momentum	0.6	2
MobileNetv2	50000	0.004	RMSprop	0.6	2
MobileDet	50000	0.8	momentum	0.6	2

3.2.1 Training All the Layers of MobileNet SSD v1

The results show that with the increase in the number of steps, the model is trained for much longer and the loss graph captured in figure 13 shows a steady decline in the total loss of mobilenet SSD v1 model being trained. Upon completion of training, the output tflite graph file in pb format is compiled into an edge TPU tflite file and deployed on the coral USB for inferencing to show the result seen in figure 14.

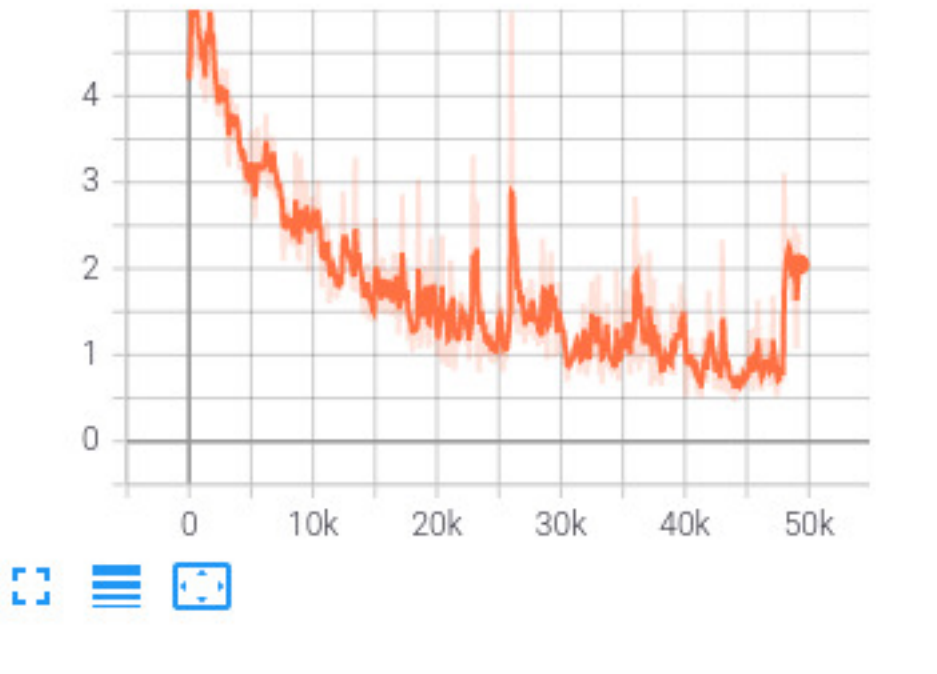

Figure 13. MobileNet SSD 50000 Step Loss

The trained model is tested on a sample picture to see the outcome, below is the result of the first inference as seen in figure 14. The result is poor because the bounding boxes are not properly placed and the classification result not accurate.

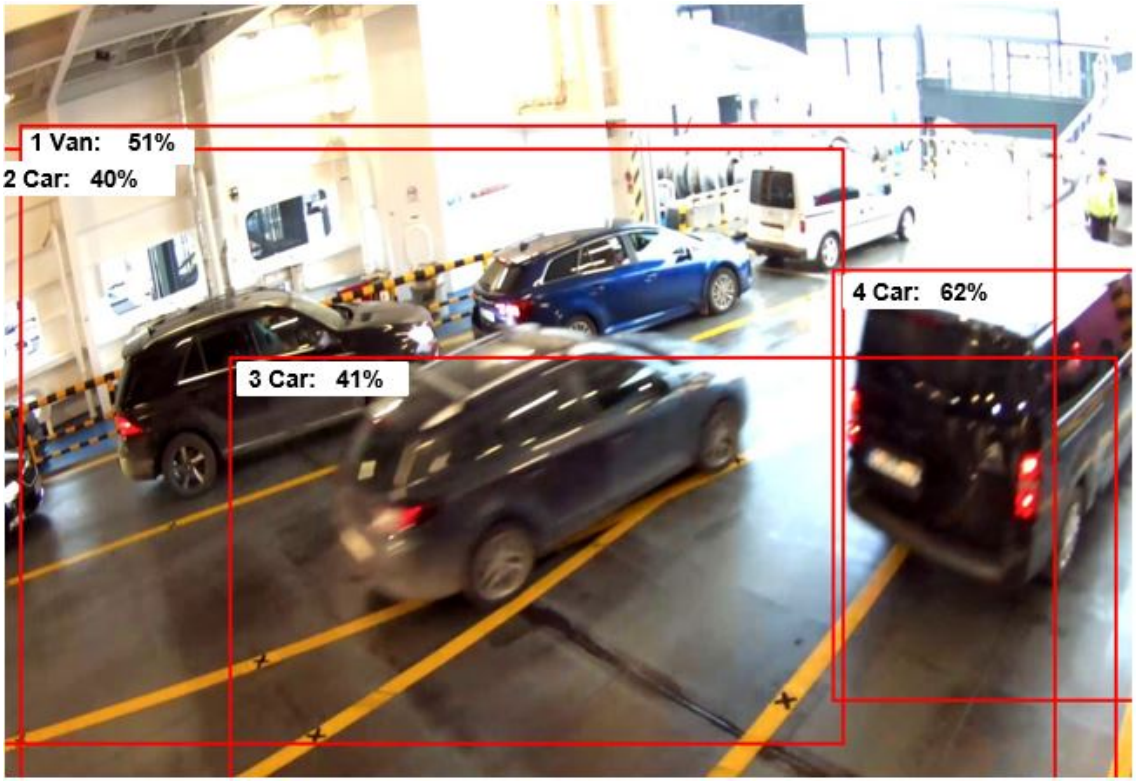

Figure 14. MobileNet SSD v1 50000 step model inference

3.2.2 Training All the Layers of MobileNet SSD v2

The model loss at the end of the training is shown in figure 15. When this loss is compared with the last layer training loss in figure 11, it is much smaller because there was a steady decline of the loss as the training went on.

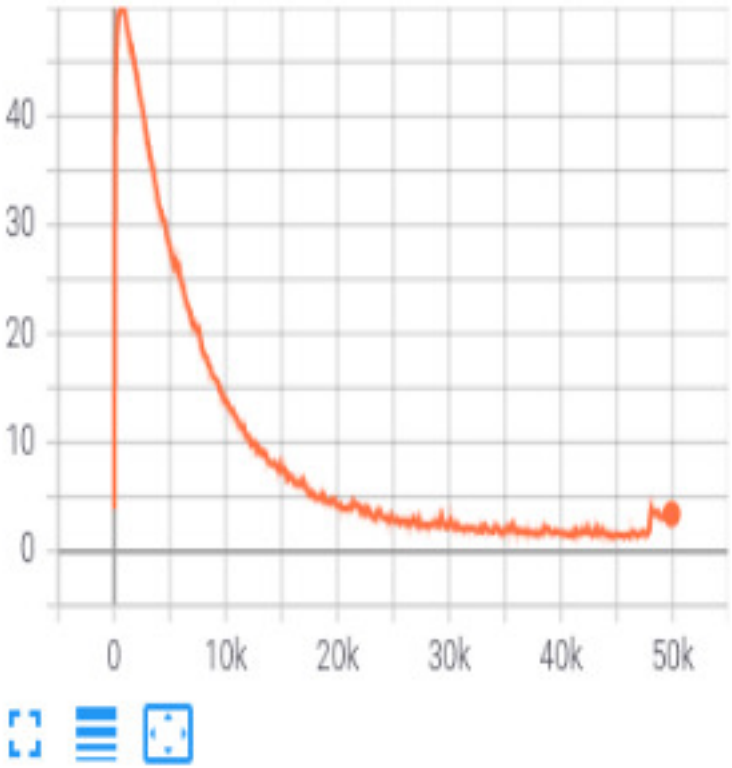

Figure 15. MobileNet v2 50000 Step loss

The model file was also compiled and inferencing carried out to see its performance as shown in figure16. Only one object is detected and its impossible to pick out which object was detected with the placement of the bounding box.The result is also poor as there are three cars that are clear enough for the model to pick out but was not detected.

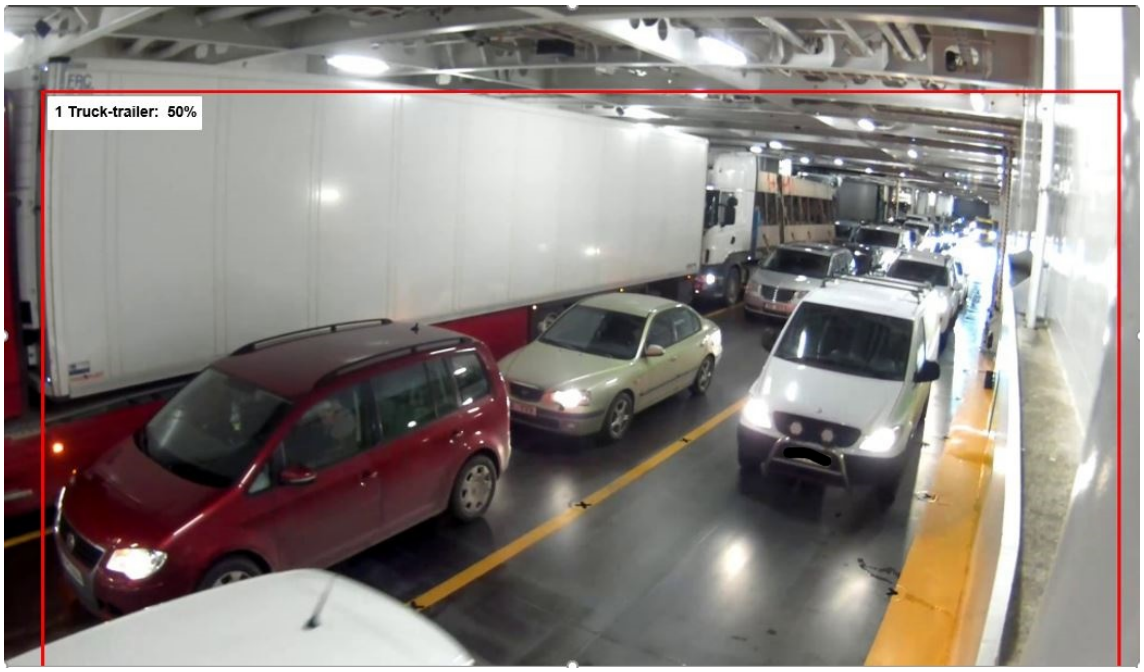

Figure 16. MobileNet SSD v2 50000 step model inference

3.2.3 Training All the Layers of MobileDet

The modileDet model loss was 1.94 after 50000 step training. The inference for this model is shown below in figure 17.From the inference result, only one object is detected and the bounding box placement is not precise. This can be attributed to the low mean average precision value obtained after training the model.

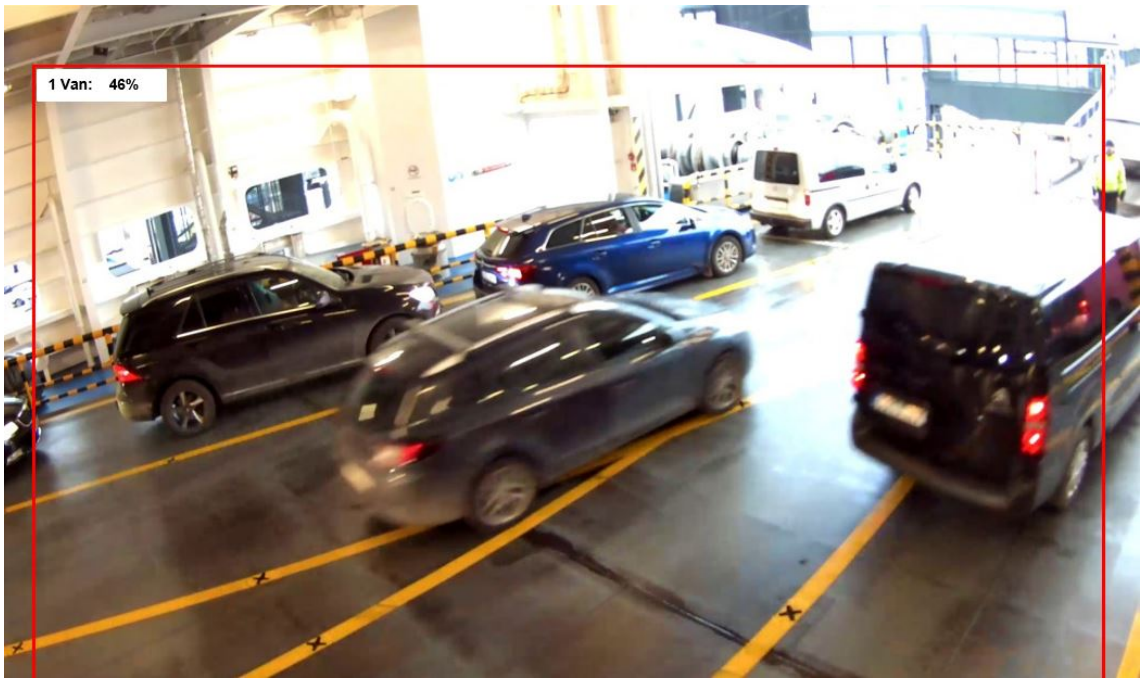

Figure 17. MobileDet 50000 step model inference

3.2.4 Comparing Model Result For All Layers Training

The Table 4 shows the final evaluation results for all three models after 50,000 training steps. The very low mean average precision values is brought to fore in the inference results shown in figures 14, 16 and 17 for all the models. Table 4 captures the total loss at the end of training, the total loss is a summation of each model's classification, localization and regularisation loss. We capture the localization and classification loss because the classification loss gives a pointer to how well the model is able to classify different objects and the localization loss with the placement of bounding boxes. At the start of the models training the goal was to achieve a total loss of less than 1 however from the results none of the model's loss was lower than 1.

,

Table 4. Result of all layers training for all models

MODEL	TOTAL LOSS	CLASSIFICATION LOSS	LOCALIZATION LOSS	mAP
MobileNet v1	3.37	1.72	0.57	0.00595
MobileNet v2	2.83	1.53	1.7	0.00736
MobileDet	1.94	0.87	0.30	0.12071

3.3 Hyper Parameter Tuning

From the results obtained in section 3.2 the model training went as required with the loss value steadily decreasing during training. However from the results shown in table 4 the precision values are low. This also translates to the results obtained during the inference. The bounding boxes are not properly placed and some of classification are incorrect as seen in figures 17, 16 and 14. An important point to note is the hyper parameters were not modified but the default values adopted. However, based on the results it was pertinent to tweak some of the hyper parameters in order to get a better result. [41] recommended an increase in batch size and [42] suggested a reduction in learning rate. The below table 5 shows some of the changes made to the hyper parameters of the models used during the experiment.

Table 5. Hyper Parameters Tuning

MODEL	STEPS	LEARNING RATE	OPTIMIZER	IoU THRESHOLD	BATCH SIZE
MobileNetv1	50000	0.079	momentum	0.6	32
MobileNetv2	50000	0.004	RMSprop	0.6	32
MobileDet	50000	0.079	momentum	0.6	32

3.3.1 MobileNet SSD v1 Hyper Parameter Tuning Result

The below inference results from figures 18, 19 and 20 show the improvement in the model's performance when changes were made to some of the model's hyper parameters. The model's learning rate was reduced from 0.2 to 0.079 and the batch size increased to 32. The lowest loss was also achieved after 23500 epochs which was 0.2 and this loss value was constant till the training ended . After 25000 training steps, the mean average precision value was 0.62. the inference result show that the bounding boxes are properly placed and the objects in the picture properly classified. Figures figures 18, 19 and 20 show that MobileNet SSD v1 performance improved tremendously with an increase in the batch size and reduction of the learning rate.

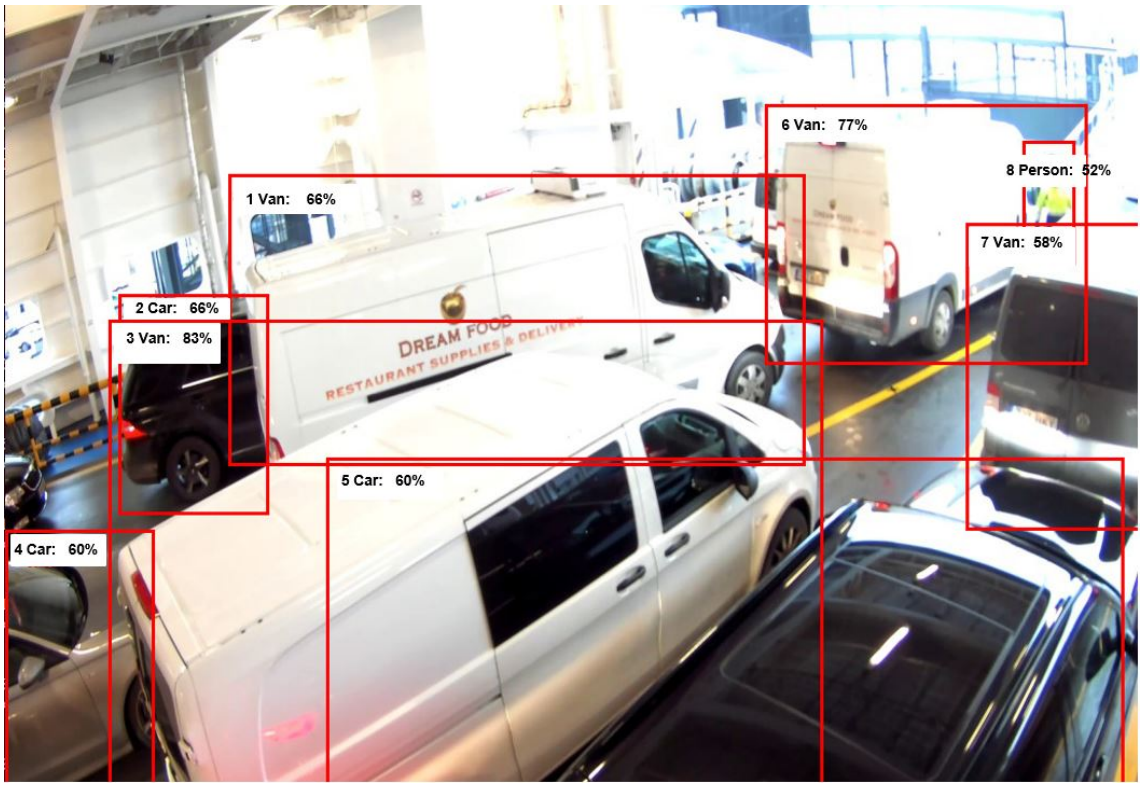

Figure 18. MobileNet v1 Hyper Parameter Change Inference result 1

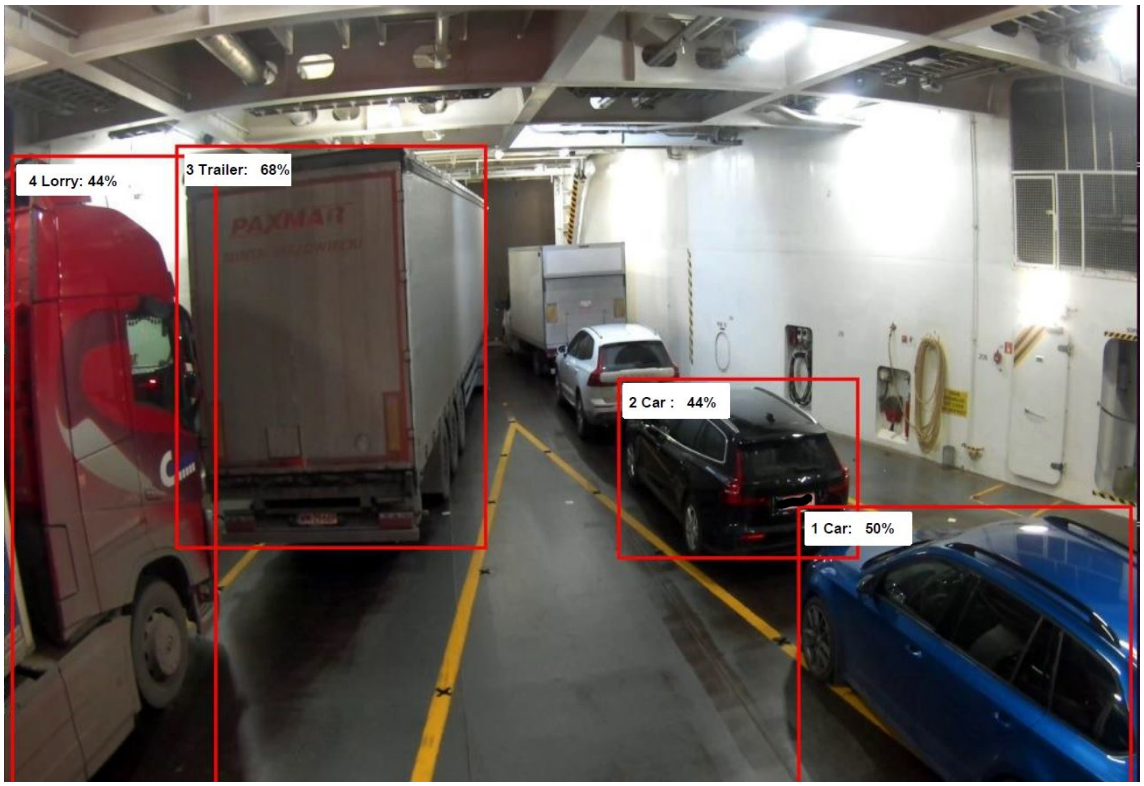

Figure 19. MobileNet v1 Hyper Parameter Change Inference result 2

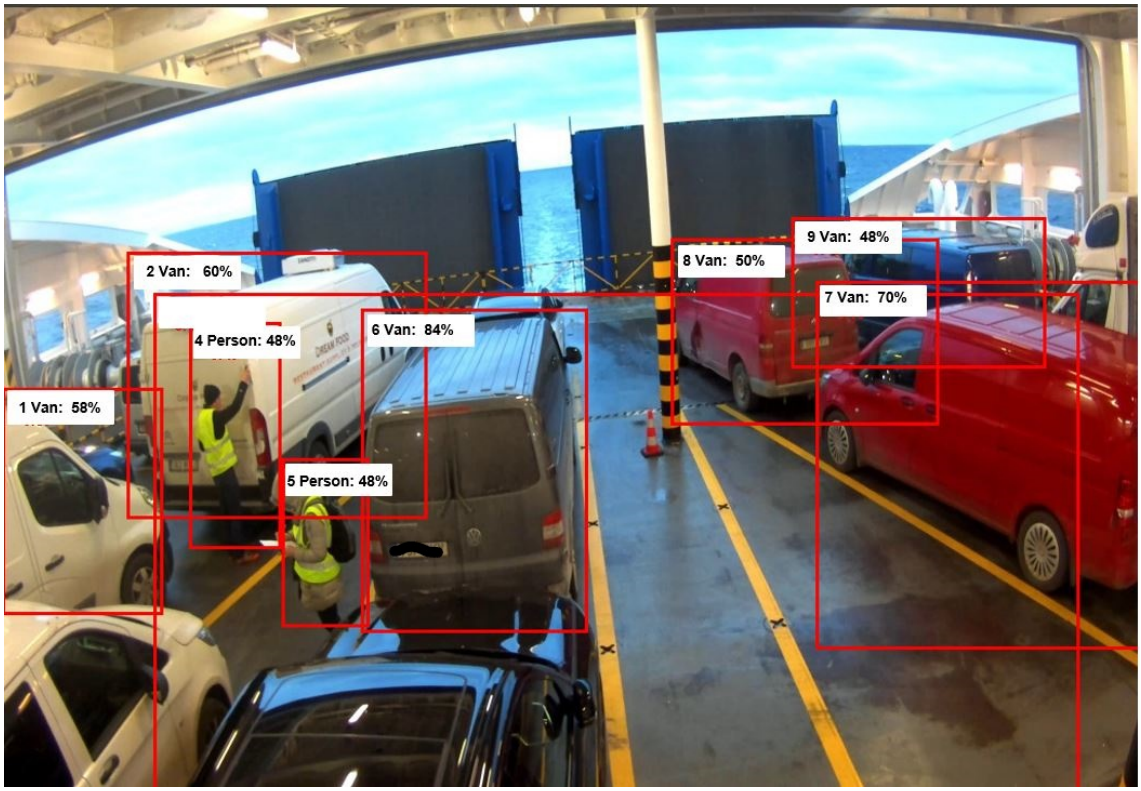

Figure 20. MobileNet v1 Hyper Parameter Change Inference result 3

3.3.2 MobileNet SSD v2 Hyper Parameter Tuning Result

There were also noticeable improvements in the performance of MobileNet SSD v2 after the batch size was increased to 32. Figures 21, 22 and 23 show the improvements. The lowest loss was also achieved after 50000 epochs. This model's performance is the least when compared with other two models. Some objects are still not detected and missed during inference however all the model's vehicle classification predictions are correct.

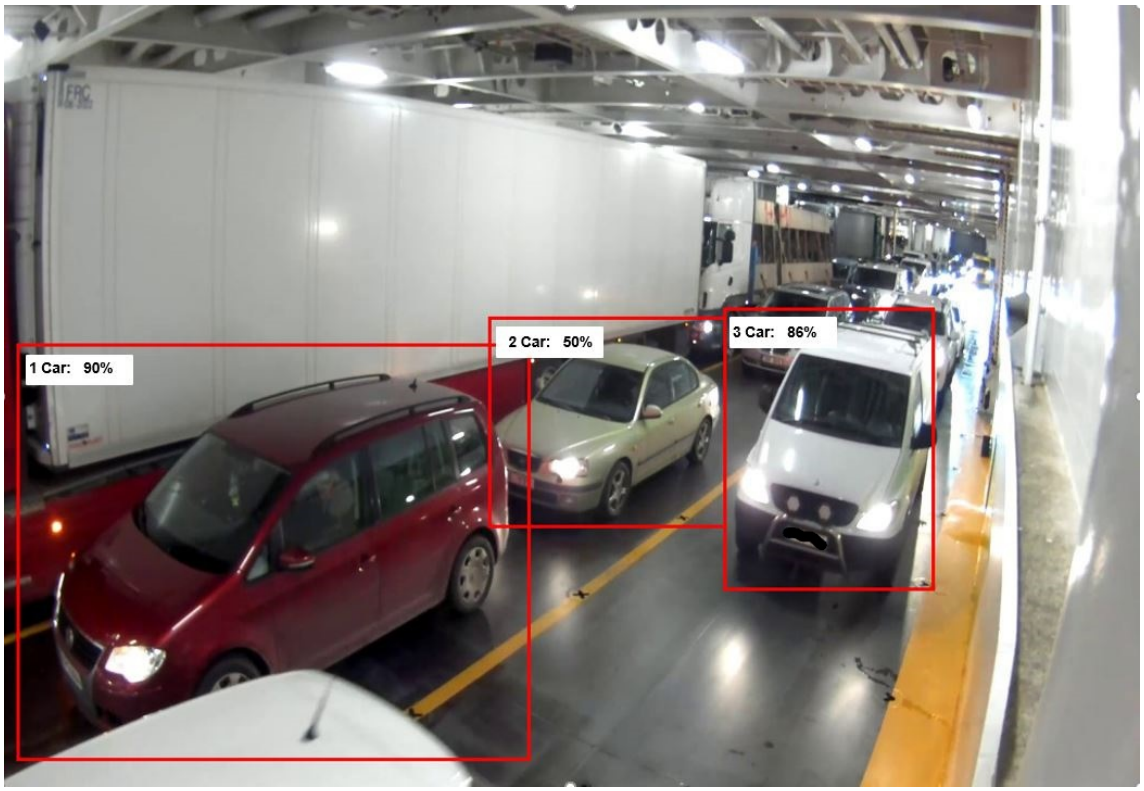

Figure 21. MobileNet v2 Hyper Parameter Change Inference result 1

Figure 22. MobileNet v2 Hyper Parameter Change Inference result 2

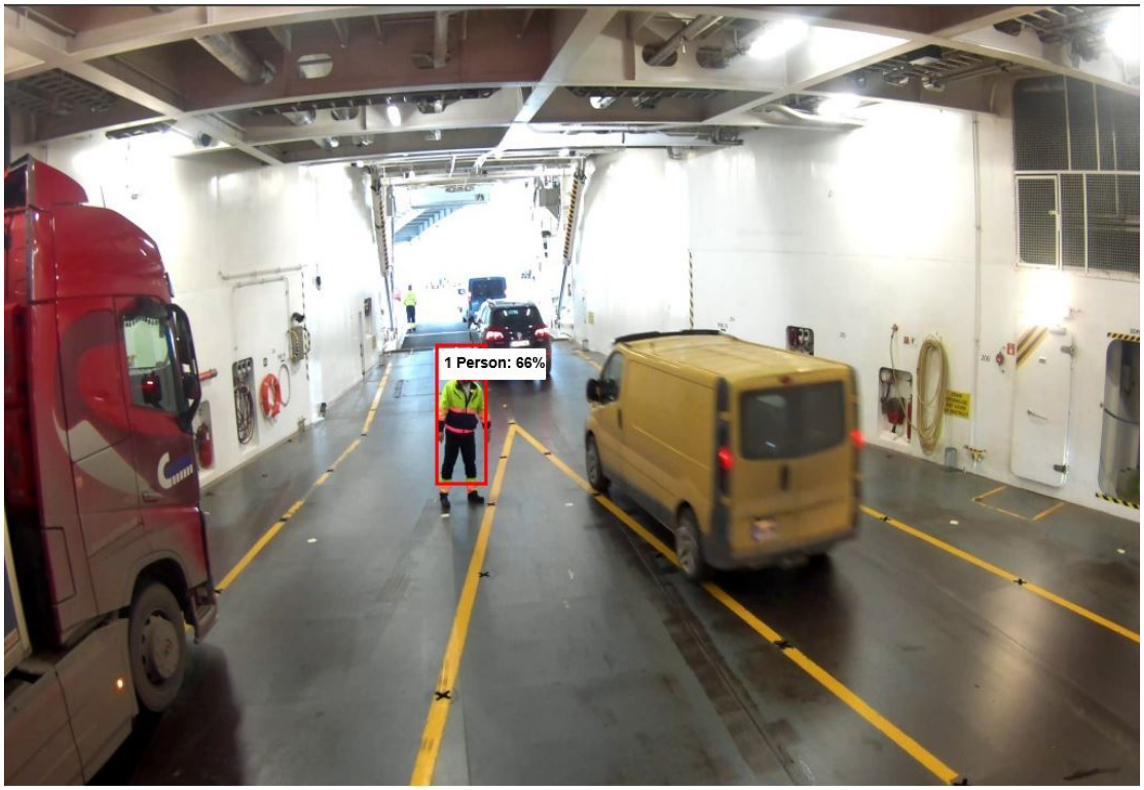

Figure 23. MobileNet v2 Hyper Parameter Change Inference result 3

3.3.3 MobileDet Hyper Parameter Tuning Result

The below inference results as seen in figures 24, 25 and 26 show the improvement for MobileDet model's performance when changes were made to some of the model's hyper parameters. The classification results were better and bounding box placement much more precise. The model's learning rate was reduced from 0.8 to 0.079 and the batch size increased to 32. The lowest loss of 0.7 was also achieved after 25000 epochs.

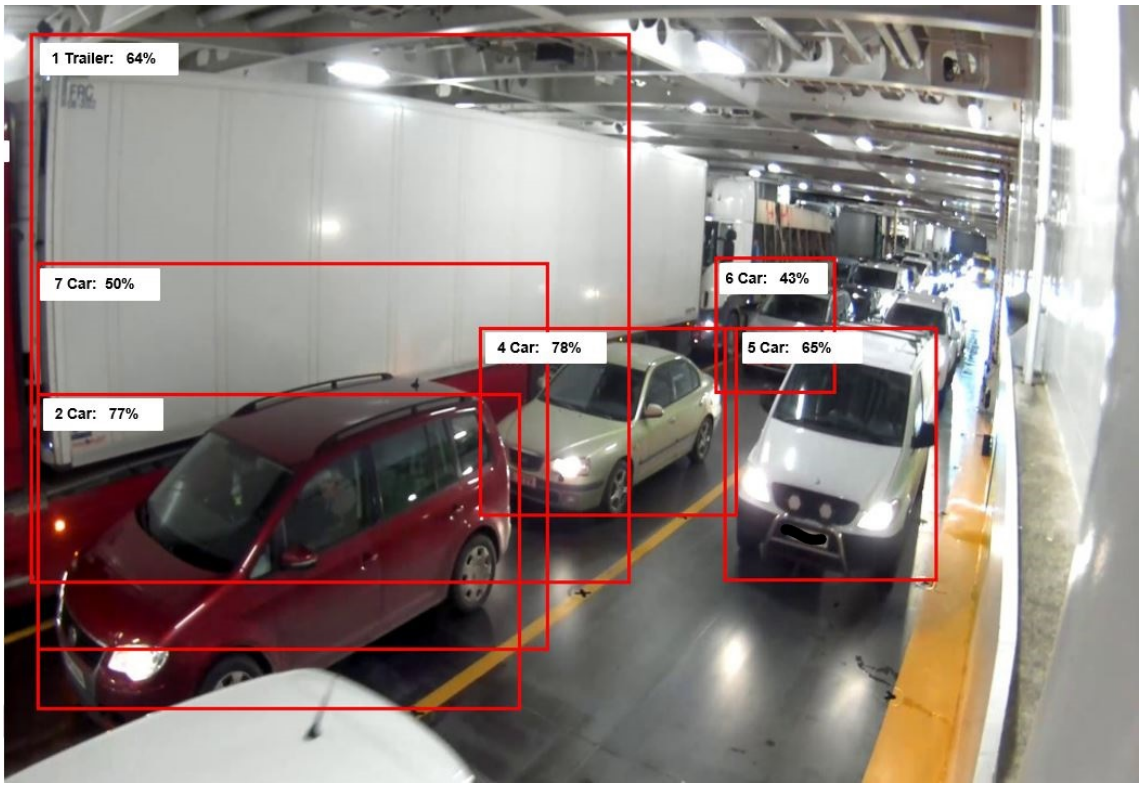

Figure 24. MobileDet Hyper Parameter Change Inference result 1

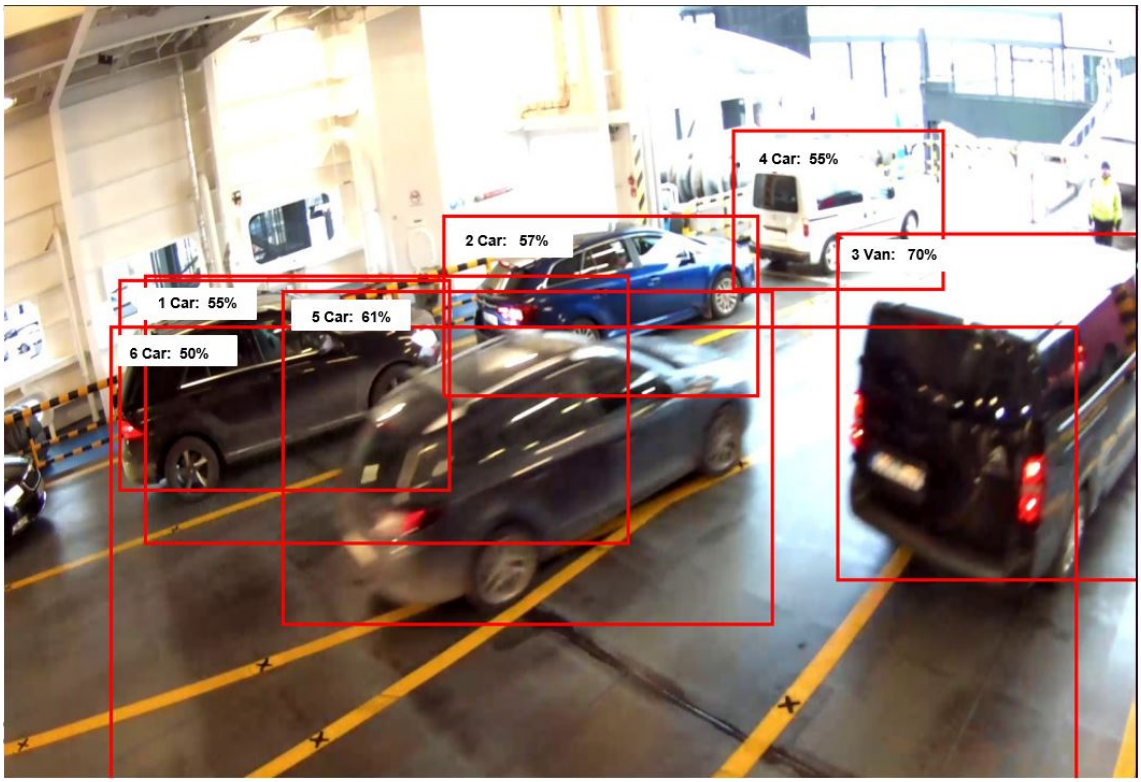

Figure 25. MobileDet Hyper Parameter Change Inference result 2

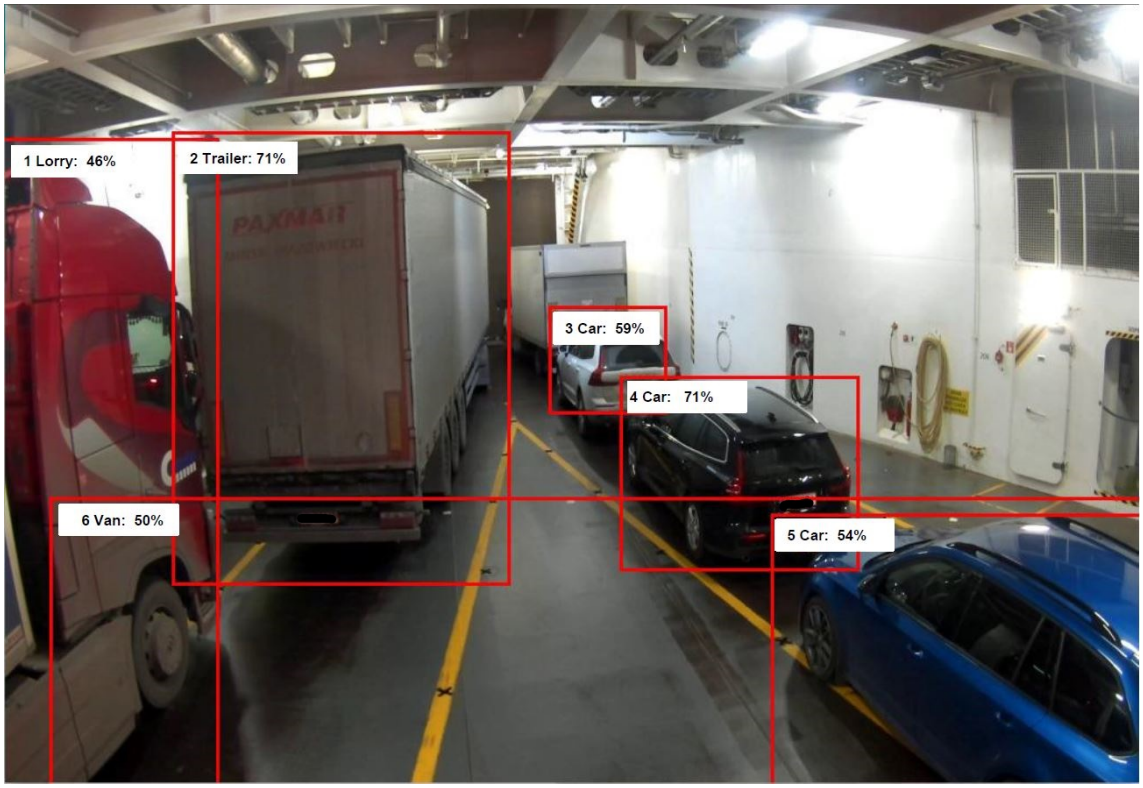

Figure 26. MobileDet Hyper Parameter Change Inference result 3

3.3.4 Table of Result For Hyper Parameter Tuning

Table 6 shows the final evaluation results for all three models at the end of training after the hyper parameters had been tuned. From the results we were able to achieve a loss less than one for MobileDet SSD and MobileNet SSD v1. The models with lower loss value had better prediction results. It can also be noticed that two models training steps were reduced to 25000 from 50000 earlier in section 3.2 because they achieved the lowest possible loss after 25000 steps hence there was no need training them further. The mean average precision values are much higher when compared with the values in Table 4. Table 6 clearly shows that the decision to tune the hyper parameters resulted in better model performance for all three models.

Table 6. Result After Tuning Hyper Parameters For Models

MODEL	TRAINING STEPS	TOTAL LOSS	mAP
MobileNet v1	25000	0.2	0.62
MobileNet v2	50000	3.2	0.28
MobileDet	25000	0.7	0.45

4. Analysis and Discussion

In this chapter the results obtained in chapter 3 will be analysed. The problems faced during each stage of the research, how they were resolved and some learning points will be discussed.

During the course of the experiment the following factors proved they were key to the success of the trained convolutional neural network models. These factors are listed as follows;

1) The quality of the dataset

2) The set hyper parameters of the model such as;

- learning rate
- Batch size

3) The length of time used to train the neural network.

These three mentioned factors were the basis for some of the issues that had to be tackled during the course of this research. In section 3.3, a reduction in the learning rate and an increase in the batch size led to improved inference results for MobileNet SSD v1 and MobileDet SSD models. The increase in batch size was done based on recommendation from [43][41] which proposed that higher batch size could lead to better result. The Batch size was increased for MobileNet SSD v2 but the learning rate left the same.[42] suggested a reduction in learning rate.

As earlier mentioned in Section 1, this experiment seeks to develop a model that will be able to classify and detect different classes of vehicles and should be small enough in size to fit into a mobile device such as the coral USB accelerator used in this experiment. Time was taken to ensure that the custom dataset contained all the classes of vehicles to enable the achievement of the end goal which is, to be able to detect these vehicle types and classify them. The newly created custom data set contained all the 11 classes and most pictures in the dataset were heterogeneous in that most of them contained not just one

vehicle class but contained as much as four to five classes in each picture.

4.1 Determining the Number of Training Steps

Looking at the results obtained from section 3.2, A higher number of epochs(50000) did not necessarily translate to a better inference result for all the models. When looking at figure 13 it can be seen that when the training steps had gotten to 20000 we had already achieved minimum loss for mobilenet SSD version 1. This was also reflected in table 6 because the minimum loss after making changes to the hyper parameters was achieved after 20000 epochs for MobileNet SSD v1.

The determination of the adequate number of training steps required for training a neural network model is still an active area of research as some researchers still use a trial and evaluate approach[44]. Despite the advances made in the field of neural network, determining some of the required parameters such as how many epochs are required is determined through a black box approach[45]. Therefore at the start of this research it was difficult to come up with an exact number to fix for the model training. Hence the choice of an experimental method. Based on an object detection tutorial on the coral website, where it was advised that to train the last layer of a model, 500 epochs could be used hence for this experiment training started with 1000 epochs. The coral website also recommended that to train all layer, 50000 epochs could be used hence the decision to start the next experiment in section 3.2 with 50000 epochs when the results seen in section 3.1 showed that the loss was still high after 1000 steps for all three models.

The results obtained from section 3.1 helped deduce that more training steps were needed to get a reduced loss value. Also the 3.2 helped note that 20000 training steps was adequate for MobileNet SSD version 1 as increasing past that number could result to over-fitting the trained model to the dataset[46] while more time was required for MobileNet SSD version 2 and MobileDet to achieve better results.

4.2 Hyper Parameters Adjustment (fine tuning the models)

The difference in results when section 3.2 and section 3.3 are compared goes a long way to show that the model hyper parameters have a huge part to play in how well a trained model turns out. However, there is no straight forward way to determine the optimum hyper-parameters for a neural network model[45]. Mastering the art of understanding the needed changes to be made sometimes requires years of expertise or trial and error method could be adopted[45]. The output performance of the model, model training time are

hugely dependent on hyper-parameters such as learning rate, batch size and weight decay and their effects are tightly coupled with each other[45]. During the course of this research using MobileNet SSD v1 as a case study, the below figures 27, 28 and 29 buttresses some of the improvements in the classification and detection noticed when changes to the learning rate and batch size was made.

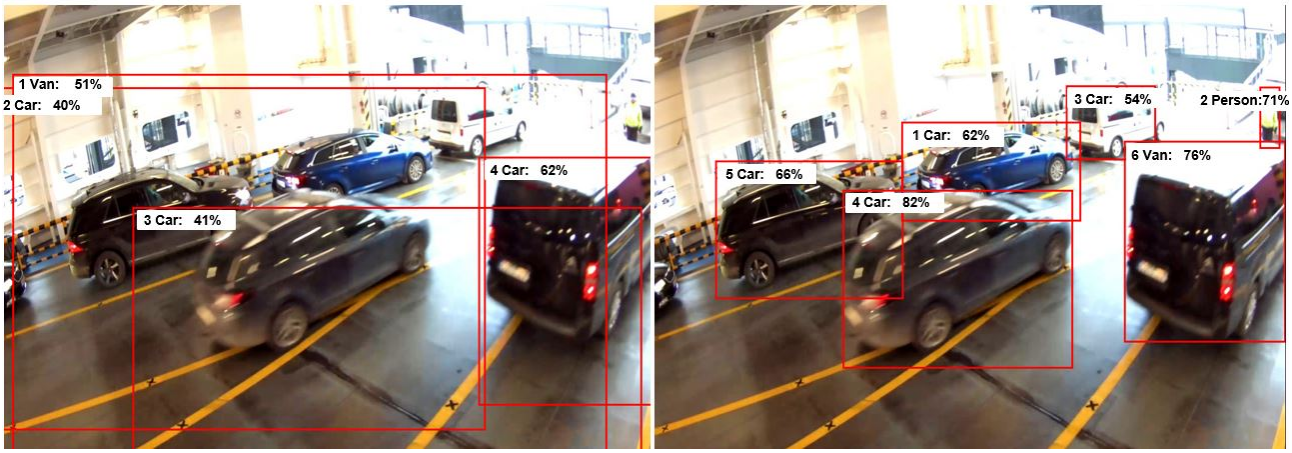

Figure 27. Finetune MobileNet v1 Result comparison 1

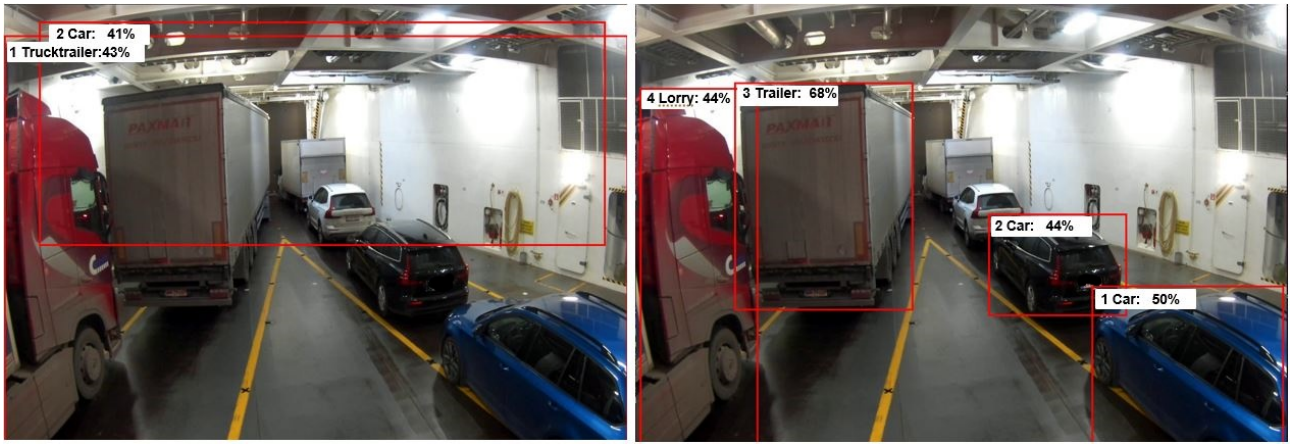

Figure 28. Finetune MobileNet v1 Result comparison 2

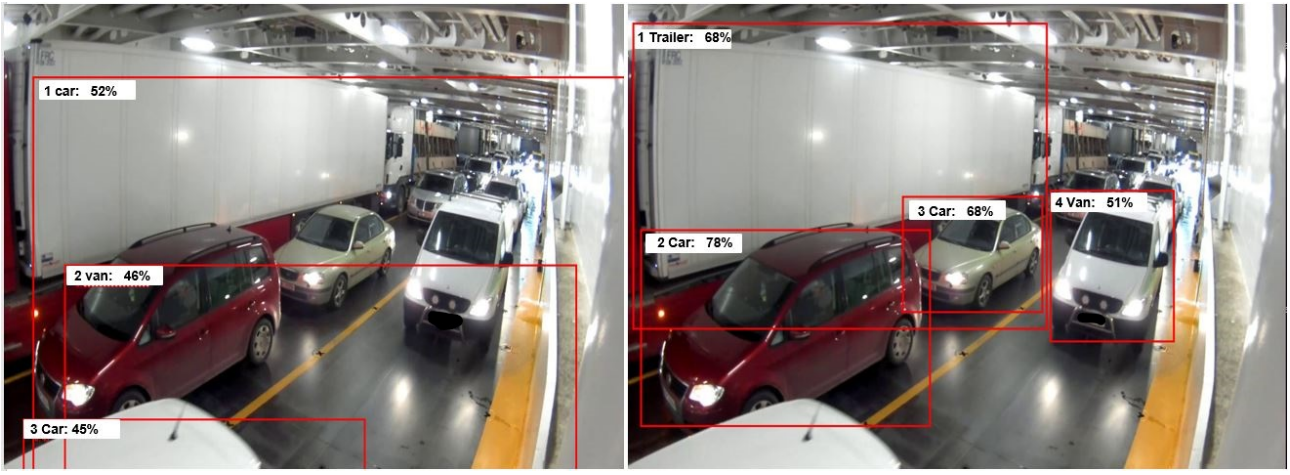

Figure 29. Finetune MobileNet v1 Result comparison 3

4.2.1 Batch Size Tuning

Batch size is an influential hyper parameter when configuring neural network for training[43]. During this research it was noticed that the increase in the batch size for MobileNet SSD version 1 resulted in the model performing better. The batch size determines how many images are used for training in one epoch and it impacts the time required for the model to converge [43]. During the experiment in section 3.2. The experiment as carried out in two python platforms as depicted in Table 7.

Table 7. Platforms Used for Tensor Flow API

Platform	Hardware	RAM(GB)	Avg Time Per 100 Steps
Ubuntu Linux	CPU	12	5sec
Google Colab Pro	GPU	15	100secs

The batch sizes for all three models was 2 and the model training could be executed on a Tensorflow API set up on a CPU based system with 12GB of RAM. When the batch size was increased to 32 the training was executed on a Tensorflow API running on Google Colab Pro with 15GB RAM because the CPU based TensorFlow API ran out of memory during training. It was noticed that the time taken for every 100 training steps was about 3 to 5 secs with a lower batch size however when the batch sizes of the respective models were increased the time for every 100 training step increased to approximately 100 secs as can be seen in figure 30. The increase in batch size translated to a longer training time for the same number of steps when previously executed with a lower batch size.

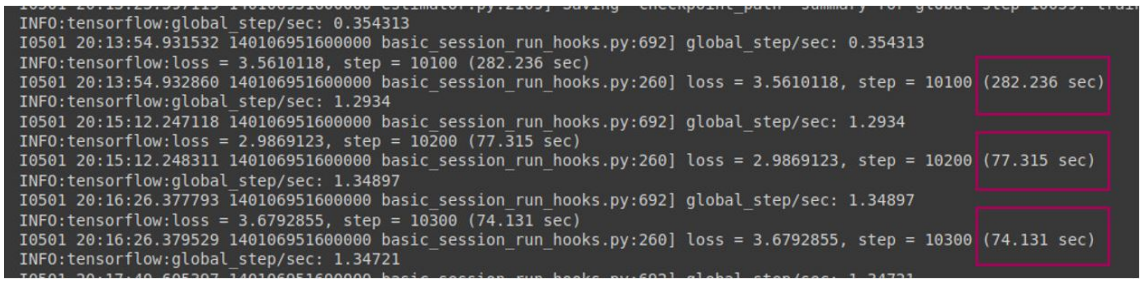

Figure 30. Increase in training step time due to batch size increase

From the results obtained in Section 3.3, an increase in the batch size led to improved results for all three models however in varying degree but MobileNet SSD version 1 showed the best improvement with an increase in batch size. As specified in many research papers [45][43] coming up with the best batch size for the task at hand requires some level of experimental approach to achieve a batch size that will help the model converge and not result in overfitting.

4.2.2 Learning rate Tuning and Monitoring

Learning rates determine how much the weights in a neural network change in reaction to an observed error in the training data[42]. [47] states that learning rate is the most important hyper-parameter to be tuned when training deep neural networks. During the experiment in section 3.3.1 the learning rate was reduced from 0.2 to 0.079 for MobileNet SSD v1 and MobileDet which lead to improvement in the results shown in figure 28. It was reduced based on discussion in [42] that a high learning rate doesn't necessarily translate to better results hence the reason to select a more conservative learning rate which eventually provided better result. The learning rate of MobileNet SSD v2 was not modified as it was considered to be moderate. [42] [47] stated the set learning rate determines the convergence or divergence of the model being trained hence the importance of knowing when to have a high or low learning rate.

In order to prevent over training of the models, the model loss values were monitored during training to notice when the validation error starts to rise in order to stop training based on recommendation from [46]. Data augmentation techniques such as random horizontal flip and ssd random crop were employed to variate the data set during training. The dataset was also sourced from different sources such as google and other private sources in order to have a heterogeneous dataset.

4.2.3 SGD Optimizer Analysis

From Table 3 two of the selected models, MobileNet SSD v1 and MobileDet made use of momentum algorithm for optimizing SGD while MobileNet SSD v2 made use of RMSprop. Its noticed that both models with momentum optimizer achieved lower model loss values and reached the value with fewer number of training steps when compared with the model using RMSprop. This is reflected in the results shown in table 3.3.4, training steps column. Also based on the inference results captured in section 3.3 it could be deduced that the momentum SGD optimizer is better suited for the task at hand.

5. Conclusion

In this work three convolutional neural network models were presented which are able to detect and classify different vehicle types with varying degree of performance. Convolutional neural network models were retrained to classify custom data classes such as 11 different vehicle types.

Based on the results obtained, MobileNet SSS v1 showed the best results in achieving the task at hand however this does not imply that this model with the best result for this work is better than MobileNet SSD v2 and MobileDet but has substantiated the notion that developing a well trained neural network is determined by a number of factors as shown in this work which include the dataset, the training time and the model's set hyper parameters. In the end, the results demonstrated that the tuning measures carried out during this research worked in favor of MobileNet SSD v1.

The output of this work was tailored to be deployed on the coral USB accelerator and the quantization of the trained model enabled the neural network to be deployed on the edge device for inferencing. However the output models can be configured for deployment on other edge computing devices for further test on its performance.

This work is unique because prior to the commencement of this research a neural network model developed for classifying vehicle types for a low power device was not readily available hence the output of this research could be used as a foundation for further research such as developing a vehicle classifying model on other neural network frameworks aside TensorFlow such as pytorch and caffe. Also, a novel CNN architecture could be developed for the vehicle classification task unlike the transfer learning approach which was used for this research. Another aspect for further research involves trying out new SGD optimizer algorithms such as madgrad to see how it affects the model results.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. DOI: 10.1145/3065386.

[2] Guanpeng Li et al. "Understanding error propagation in deep learning neural network (DNN) accelerators and applications". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Nov. 2017. DOI: 10.1145/3126908.3126964.

[3] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Apr. 2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[4] Ondrej Karpis. "System for Vehicles Classification and Emergency Vehicles Detection". In: *IFAC Proceedings Volumes* 45.7 (2012), pp. 186–190. DOI: 10.3182/20120523-3-cz-3015.00037.

[5] Zhen Dong and Yunde Jia. "Vehicle type classification using distributions of structural and appearance-based features". In: *2013 IEEE International Conference on Image Processing*. IEEE, Sept. 2013. DOI: 10.1109/icip.2013.6738890.

[6] Yu Peng et al. "Vehicle Type Classification Using PCA with Self-Clustering". In: *2012 IEEE International Conference on Multimedia and Expo Workshops*. IEEE, July 2012. DOI: 10.1109/icmew.2012.73.

[7] Zhen Dong et al. "Vehicle Type Classification Using a Semisupervised Convolutional Neural Network". In: *IEEE Transactions on Intelligent Transportation Systems* 16.4 (Aug. 2015), pp. 2247–2256. DOI: 10.1109/tits.2015.2402438.

[8] Xue Mei and Haibin Ling. "Robust Visual Tracking and Vehicle Classification via Sparse Representation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.11 (Nov. 2011), pp. 2259–2272. DOI: 10.1109/tpami.2011.66.

[9] Jobin George, Leena Mary, and Riyas K S. "Vehicle detection and classification from acoustic signal using ANN and KNN". In: *2013 International Conference on Control Communication and Computing (ICCC)*. IEEE, Dec. 2013. DOI: 10.1109/iccc.2013.6731694.

[10] A.H.S. Lai, G.S.K. Fung, and N.H.C. Yung. "Vehicle type classification from visual-based dimension estimation". In: *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*. IEEE. DOI: 10.1109/itsc.2001.948656.

[11] Xiaoxu Ma and W.E.L. Grimson. "Edge-based rich representation for vehicle classification". In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. IEEE, 2005. DOI: 10.1109/iccv.2005.80.

[12] Ying Shan, Harpreet S. Sawhney, and Rakesh Kumar. "Unsupervised Learning of Discriminative Edge Measures for Vehicle Matching between Nonoverlapping Cameras". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.4 (Apr. 2008), pp. 700–711. DOI: 10.1109/tpami.2007.70728.

[13] Yu Peng et al. "Vehicle Type Classification Using Data Mining Techniques". In: *The Era of Interactive Media*. Springer New York, Aug. 2012, pp. 325–335. DOI: 10.1007/978-1-4614-3501-3_27.

[14] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: (Nov. 26, 2015). arXiv: 1511.08458 [cs.NE].

[15] Rongjie Yu et al. "Convolutional neural networks with refined loss functions for the real-time crash risk analysis". In: *Transportation Research Part C: Emerging Technologies* 119 (Oct. 2020), p. 102740. DOI: 10.1016/j.trc.2020.102740.

[16] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: (Apr. 17, 2017). arXiv: 1704.04861 [cs.CV].

[17] Weiqi Sun et al. "Vehicle Following in Intelligent Multi-Vehicle Systems Based on SSD-MobileNet". In: *2019 Chinese Automation Congress (CAC)*. IEEE, Nov. 2019. DOI: 10.1109/cac48633.2019.8996181.

[18] Yunyang Xiong et al. "MobileDets: Searching for Object Detection Architectures for Mobile Accelerators". In: (Apr. 30, 2020). arXiv: 2004.14525 [cs.CV].

[19] Albert Reuther et al. "Survey and Benchmarking of Machine Learning Accelerators". In: (Aug. 29, 2019). DOI: 10.1109/HPEC.2019.8916327. arXiv: 1908.11348 [cs.PF].

[20] Aurelia Michele, Vincent Colin, and Diaz D. Santika. "MobileNet Convolutional Neural Networks and Support Vector Machines for Palmprint Recognition". In: *Procedia Computer Science* 157 (2019), pp. 110–117. DOI: 10.1016/j.procs.2019.08.147.

[21] Christian Szegedy Sergey Ioffe. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Machine learning*. Ed. by Sergey Ioffe. Vol. 1. Sergey Ioffe, 2015. URL: https://arxiv.org/abs/1502.03167v3.

[22] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: (Oct. 16, 2017). arXiv: 1710.05941 [cs.NE].

[23] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. “Adaptive Layerwise Quantization for Deep Neural Network Compression”. In: *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, July 2018. DOI: 10.1109/icme.2018.8486500.

[24] Google Coral. *retrain-detection*. Version 1. Dec. 2020. URL: https://coral.ai/docs/edgetpu/retrain-detection/ (visited on 12/05/2020).

[25] Hakan Bilen and Andrea Vedaldi. “Weakly Supervised Deep Detection Networks”. In: (Nov. 9, 2015). arXiv: 1511.02853 [cs.CV].

[26] Dim P. Papadopoulos et al. “Training object class detectors with click supervision”. In: (Apr. 20, 2017). arXiv: 1704.06189 [cs.CV].

[27] darrenl tzutalin. *LabelImg*. Version 1.8.1. Dec. 2018. URL: https://github.com/tzutalin/labelImg (visited on 01/15/2021).

[28] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: (Sept. 15, 2016). arXiv: 1609.04747 [cs.LG].

[29] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: (Dec. 3, 2019). arXiv: 1912.01703 [cs.LG].

[30] Amol Mavuduru. *Which deep learning framework is the best?* Version 1. June 2020. URL: https://towardsdatascience.com/which-deep-learning-framework-is-the-best-eb51431c39a (visited on 05/07/2021).

[31] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: (June 20, 2014). arXiv: 1408.5093 [cs.CV].

[32] Tianqi Chen et al. “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems”. In: (Dec. 3, 2015). arXiv: 1512.01274 [cs.DC].

[33] Martin Abadi et al. “TensorFlow: A system for large-scale machine learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[34] Fatih Ertam and Galip Aydin. “Data classification with deep learning using Tensorflow”. In: *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, Oct. 2017. DOI: 10.1109/ubmk.2017.8093521.

[35] Paul Barham Martín. “TensorFlow: A System for Large-ScaleMachine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation*. 2016.

[36] Yann LeCun et al. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1990. URL: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[37] Bahadır Özdemir et al. “Performance measures for object detection evaluation”. In: *Pattern Recognition Letters* 31.10 (July 2010), pp. 1128–1137. DOI: 10.1016/j.patrec.2009.10.016.

[38] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2 (Sept. 2009), pp. 303–338. DOI: 10.1007/s11263-009-0275-4.

[39] Md Atiqur Rahman and Yang Wang. “Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation”. In: *Advances in Visual Computing*. Springer International Publishing, 2016, pp. 234–244. DOI: 10.1007/978-3-319-50835-1_22.

[40] Bengemon. $TF_Object_Detection2020$. Version 1. Aug. 2020. URL: https://github.com/Bengemon825/TF_Object_Detection2020 (visited on 01/26/2020).

[41] Samuel L. Smith et al. “Don’t Decay the Learning Rate, Increase the Batch Size”. In: (Nov. 1, 2017). arXiv: 1711.00489 [cs.LG].

[42] D.R. Wilson and T.R. Martinez. “The need for small learning rates on large problems”. In: *IJCNN’01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. IEEE, 2001. DOI: 10.1109/ijcnn.2001.939002.

[43] Pavlo M. Radiuk. “Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets”. In: *Information Technology and Management Science* 20.1 (Jan. 2017). DOI: 10.1515/itms-2017-0003.

[44] Shivam Sinha et al. “Epoch determination for neural network by self-organized map (SOM)”. In: *Computational Geosciences* 14.1 (May 2009), pp. 199–206. DOI: 10.1007/s10596-009-9143-0.

[45] Leslie N. Smith. “A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay”. In: (Mar. 26, 2018). arXiv: 1803.09820 [cs.LG].

[46] Warren S. Sarle. *Stopped training and other remedies for overfitting*. Version 1. May 1995. URL: https://www.academia.edu/3337293/Stopped_training_and_other_remedies_for_overfitting (visited on 05/03/2021).

[47] Leslie N. Smith. “No More Pesky Learning Rate Guessing Games”. In: (June 3, 2015). arXiv: 1506.01186 [cs.CV].

Appendices

Appendix 1

0.1 Non-exclusive licence for reproduction and publication of a graduation thesis

I Irase Ohiomomon Ohiokpehai Jnr

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Deep Neural Network Based Object Classification on Low Power Edge Computing Hardware" , supervised by Mairo Leier

1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

08.05.2021

Appendix 2

0.2 GitHub Repository

The output files and code used for this project has been hosted in the below repository.

`https://github.com/irohio/MasterThesisMay2021`