Darlington O. Omoifo

Obstacle detection in autonomous vehicles using deep learning

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme

Thesis

18 April 2018

Metropolia

Acknowledgement

First, I am grateful to God for every of his mercy, help and direction towards me and in completing this project. I wish to express my sincere appreciation to my lecturers here at Metropolia UAS and Texas Tech University for their immeasurable impartation of knowledge.

I am ineffably indebted to my university and employer Helsinki Metropolia UAS, under whose auspices I studied and conducted this research under Robusta project sponsored by Business Finland® (Tekes). I am particularly appreciative for having my project manager, Olli Alm, who doubled as my supervisor. His valuable, sincere and conscientious guidance; belief in my abilities, timely disposition, and constant encouragement were pivotal to bringing this project to fruition.

I take this opportunity to record my gratitude to all lecturers whom I had to learn under including, in no particular order, Dr Jaana Holvikivi, Dr Vuori Jarkko, Dr Timo Salin, Anne Perälampi, Ulla Patola, and Peter Hjort. Your guidance these past four years has been priceless and invaluably precious.

I extend my gratitude to every Finn who had to indirectly foot the cost of my tuition through their taxes. Notably, I acknowledge, with a deep sense of reverence, my gratitude towards my parents, Mr & Mrs Dele Omoifo, and my siblings, who as always, stood by me these past years.

Besides, I place on record my sense of gratitude to everyone who have, in one way or the other, lent their assistance during my studies, and more so, towards this project. Finally, gratitude goes to all my friends and classmates; in Finland and the U.S, who have facilitated, in the least of manners, my completion of this project and its report.

Darlington O.

Author	Darlington O. Omoifo
Title	Obstacle detection in autonomous vehicles using deep learning
Number of Pages	50 pages + 6 appendices
Date	18 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology (IT)
Professional Major	Software Engineering
Supervisor	Olli Alm, Senior Lecturer
Instructor	Olli Alm, Project Manager

This project was carried out to enhance the intrinsic intelligence and surveillance of an autonomous vehicle (AV) with remote control capability in Robusta project – sponsored by Tekes – the Finnish Funding Agency for Technology and Innovation, done at Helsinki Metropolia UAS. The scope included developing four software used namely in training deep learning models used in object recognition, real-time obstacle recognition, real-time obstacle detection, and object detection.

Five models were trained with the model trainer (MT) software using CIFAR-10 dataset. The autonomous vehicle obstacle detector (AVOD) prototype was developed to be used in the AV's internal intelligence system. Using this prototype, possibly with a lidar: not included in this report, obstacle detection – recognition and localization of driving scene impediment object – could be carried out by the AV. Given it is a real-time application, the software's processed frames per second (fps) should nearly equal the camera's fps. An object detector mobile (ODM) Android® application was developed for object detection.

Using the trained model obstacle recognizer (TMOR) a model recorded 85.6% prediction accuracy with 10,000 test images. The AVOD software prototype could, in real-time, detect in a frame all objects in its trained classes (21) – processing 30fps at 100.0% the camera's capacity.

Keywords	deep learning, remotely controlled vehicles, machine learning, obstacle detection, object recognition, convolutional neural network

Contents

List of Abbreviations

List of Abbreviations

AV	Autonomous Vehicle. It is a vehicle with self-driving capability. In this project, the definition is stretched to include one with remote control access or one that can be controlled or driven remotely.
AVOD	Autonomous Vehicle Obstacle Detector This is a real-time obstacle detection software developed in this project that has its prototype as a component of the Autonomous Vehicle's internal intelligence system. A Caffe model was used in its development.
CNN	Convolutional Neural Network. A deep learning algorithm used in object recognition.
CNTK	Microsoft® Cognitive Toolkit. An open source Microsoft computer vision library. It is a dedicated deep learning library toolkit.
CUDA	Compute Unified Device Architecture. A parallel computing platform that readily provide the kernel using a graphics card - graphics processing unit (GPU).
CV	Computer Vision. The branch of artificial intelligence that deal with the ability of machines to recognise objects.
DL	Deep Learning. A branch of ML that deals more with artificial neural networks.
FC	Fully Connected [layer]. The last layer (after alternating convolution–max-pooling and sometimes dropout layer) in a CNN architecture where the outputs are turned in to the output layer.
FPS	Frames Per Second. This is the video frames recording or streaming rate of a video camera. In this project, it also reflects the rate at which any of the Software AVOD, TMOR, ODM frame processing rate per second.
GPU	Graphics Processing Unit. The processing unit that is optimised for rapid response to the commands of the computer with readily available kernel and multiple number of threads compared to the central processing unit of a computer.

K-NN	K-Nearest Neighbor. An unsupervised learning algorithm for detecting closely related members of a class. E.g. members of a network or graph.
ML	Machine Learning. The branch of computer science that deals with non-explicit programming of computers in making them do their assigned task.
MT	Model Trainer. The name of the software developed in this project that handles training of a model used for object recognition.
NMS	Non-Maximum Suppression. Used to avoid false positives or failure of detections.
ODM	Object Detector Mobile. This is a mobile android application that was developed in this project and used the Caffe model. Unlike the AVOD, it is not real-time, hence not used for obstacle detection but simulates a visualization of the initial stage of obstacle detection which is object detection on mobile platform.
PCA	Principal Component Analysis. A statistical procedure used for transformation of observations. It is often used in ML for obtaining principal components.
R-CNN	Region-based CNN. A CNN algorithm that is optimised for reporting on regions where an object is detected or where objectiveness is high.
ReLU	Rectified Linear Unit. This is where the linearity is removed from the CNN.
ResNet	Residual Network. Tends to ease the training of the network as deep neural networks are harder to train.
RoI	Region of Interest. This is the component of what constitutes detection areas –high objectiveness. Also, this is used in RPN, NMS and R-CNN.
ROS	Robot Operating System. This is a computer operating system (OS) like Microsoft® Windows® or Ubuntu® capable of running on other OS such as Linux or Windows. However, it is dedicated development OS for robot construction, test running and to more advanced artificial intelligence approaches including controls, and optimised virtual environments.

RPN	Region Proposal Network. An optimised approach to R-CNN where a network of the regions built and used in the R-CNN. It is optimised and faster compared to R-CNN without it.
SGD	Stochastic Gradient Descent. An optimised stochastic approximation of gradient descent in minimizing an objective function.
SPP	Spatial Pyramid Pooling. Helps fix the issue of fixed input size in CNN e.g. 300x300 images only accepted as input.
SS	Selective Search. An object detection algorithm used for small weight images as it is rather slow in computation.
SVM	Support Vector Machine. A supervised learning model with associated algorithms optimised for classification and regression analysis.
TMOR	Trained Model Obstacle Recognizer. The developed software that used any of this project's trained models in object or obstacle recognition.
VGG	Visual Geometry Group. A computer vision lab - have a model implemented in this project.

1 Introduction

Humans are in an ever-increasing quest to minimize or eliminate errors, increase precision, and possibly obliterate accidents. Though, much is increasingly sought within man's capability, hitherto, some of man's least conceivable solutions have come from integrating machines in daily tasks at various strata of human endeavour. This growing call continually stretches scientists and engineers, in science and technology respectively, to forage new frontiers in research and technological advancements in making discoveries and improving on those already made. This is where machine learning (ML), particularly deep learning – exemplified in computer vision, comes handy and invaluable.

1.1 Motivation

It is appropriate to now state the motivation behind this project. Firstly, curiosity on how computers can harness perception was a motivation. Secondly, cancer research using machine learning was another allure; done during my ML online classes. But, it was more of this – a quest to meet a real-world need vis–a–viz resources within my reach amplified in my work as a trainee working on remotely controlled vehicles with self-driving capability in the Business Finland® sponsored Robusta project at Helsinki Metropolia UAS – University of Applied Sciences. Besides, machine learning researches or projects had been worked on by nearly 35 students at Helsinki Metropolia UAS, from search on theseus.fi – Finnish thesis database, at the time of writing. While it may yet not have been an easy path to toll, the thought of foraging a different path – one that could be considered thought-provoking, innovative, equally needful, and a societal solution that attends to many problems, was very compelling.

1.2 Goals

The aims of this project are to:

- Train a deep learning algorithm in recognising an object, possibly encountered in a driving scene;
- Implement, using established library, a classifier algorithm (CNN using Microsoft Cognitive Toolkit – CNTK) and a visual display software (R-CNN using open source computer vision – OpenCV 3.4.0) for categorization;
- Develop software that enables a user to test the class (name) of a given object within those pretrained (defined) or otherwise;
- Develop AV's obstacle detection prototype capable of processing 99.5% of its camera device's captured frames per second;
- See visually the obstacle (object) detection and recognition performance

Training of the models (5) used ten classes including, airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The data source was from the CIFAR-10 dataset [1; 2]. In developing the AVOD software 21 classes were of interest, some of which are: car, bicycle, person (a human), bus, motorbike, dog, and a horse (or a moose). The model used was pretrained by Caffe, the model provider, to recognise 21 classes of images including those mentioned above. These were likely to been seen within and outside the vehicle. [3 – 8.]

1.3 Challenge & Task

The challenge is how does the vehicle build intricate and intrinsic intelligence when the remote driver losses control or intentionally drives recklessly. Also, another problem was how does the camera infer what it is seeing for the intelligence system to infer what it is in building parts of its artificial intelligence in answering the question – an obstacle or not?

Obstacle. An obstacle, for this project, is an object in a video sequence's frame or in a photograph that exhibits any of the following characteristics:

- Capable of impeding the autonomous vehicle's continuation on its current path of travel;
- Endangers itself when the vehicle contacts it;
- Endangers the vehicle's occupants by portending possible collision;

In this project, it is permissible to use the term 'Object' interchangeably with 'Obstacle' on a broad note – an obstacle is first perceived an object prior to being proven by the autonomous vehicle's intrinsic intelligence system to be an obstacle.

The task included developing four (4) software namely: Model Trainer (MT) App, Trained Model Obstacle Recognizer – TMOR App, Autonomous Vehicle Obstacle Detector (AVOD) App, and Object Detector Mobile (ODM) App. The task was to work on obstacle recognition using MT and TMOR software; obstacle detection using AVOD software, and object detection using ODM software. CIFAR-10 [1] dataset with 60,000 images were used: for the model training (50,000 images) and validation (10,000 images). The AVOD and ODM software's model was provided by Caffe®. The MT produced models were trained to recognise any of 10 classes of objects mentioned in section 1.2 using the CIFAR-10 dataset.

The AVOD and TMOR software were designed to send the detections and recognition respectively to the AV's intelligence system which then makes decision such as stop, turn left or right, take control from the remote driver or an any other action determined

by the engineers. The details of how the AV makes its decisions and controls were implemented chiefly in C++ and Python; both programming languages, using ROS actions, and ROS messages. However, details of the AV's control and intelligent response are outside the scope of this report, and therefore not discussed further.

Models were trained using Microsoft Cognitive Toolkit (CNTK) in recognizing 10 classes of objects presented in section 1.2 using the CIFAR-10 dataset – the algorithm was CNN [9] – [12]. The trained models were required for speed (real-time) with object recognition and ease of running the models on a relatively simple device such as a Raspberry Pi – a minimalistic computer. This was prior to the release of Intel© Movidius® – a simple hardware optimized for running deep network [13]. However, the available dataset classes (labels) of the models were limited (10 classes) and did not provide a considerable spectrum for objects encountered in a driving scene. Hence, and in addition to other reasons such as visualization – detection, the AVOD software was developed with 21 classes using Caffe model which used R-CNN (deep learning) algorithm in its model training.

Another gain of the obstacle detection achieved in the AVOD is that the size could also be used in calculating the object's distance from the camera if some other factors of the camera were known. However, this advantage was not explored in this project and in line with not presenting the intelligence details, other advantages are not further elucidated. The AVOD software could process over 99.5% of all received frames in real-time having 30fps while camera's framerate was 30fps [8], able to process 4K video stream frames (60fps) or reprogrammed videos of up to 200fps by slightly increasing worker threads, if performance did drop. Details of the task will be presented in chapters 3 & 4: Architecture & Software Implementations; and Intrinsic Intelligence, Developed Software Use & Results.

1.4 Industries Utilized

Deep Learning (DL) and Machine Learning (ML) in general have a wide range of application across industries. They try to address some problems which include: self-driving cars' vision challenge, adult content recognition, digital cameras' facial (or smile) recognition, industrial safety – alarming a worker before an object gets to her/him, mammograms analysis – in the field of medicine, surveillance – territorial integrity, crime detection and analysis; scene analysis, and many more. In addition, object recognition can be used in simulating perception for the visually impaired. [14 – 30.]

2 Deep Learning Background

To better appreciate what deep learning is, it is appropriate to have a good synopsis of what its parent field of study – machine learning is. Machine learning as a research field has evolved in virtually half a century ago. Its application and increasing ability to attend to some of life's most demanding situations that require high intelligence, critical thinking, and possibly coordination has been novel thus far. This has seen it being used in task requiring human intelligence. Some of these are best demonstrated via the lens of the big five (5) players; probably for their investment and partnership in furthering AI research, Microsoft©, Amazon©, IBM©, Google©, and Facebook© - claims USA TODAY ® [31].

In computer science a substantial number of algorithms (mathematics) [10] – [16] is involved. Each approach to solving a problem is presented as an algorithm. An algorithm can be explained as a process, procedure, rule or set of it, to be followed in calculations or other problem-solving operations, especially by a computer. Algorithms are defined procedures, path optimized for computing calculations, and data processing; particularly, large data sets. Also, an algorithm is employed in automated task, including logical reasoning that is autonomous. Briefly some algorithms in wide usage or considered in machine learning task include linear regression [30,5–6]. Also, obstacle (object) detection or recognition using support vector machine (SVM), principal component analysis (PCA) and k-nearest neighbour (k-NN). These algorithms, though have their application in object recognition, do not compare favourably with convolutional networks like CNN, or R-CNN. However, these algorithms for their usage and bases in machine learning are discussed further. [32 – 45.] This will be after enunciating what machine learning could really mean.

2.1 Machine Learning

Deep learning (DL) is a subset of machine learning (ML). On a basal note, a general overview of what it entails will help elucidate the complexities that may, otherwise, be lying in having a grasp of what deep learning is. Broadly, machine learning is a scientific approach to solving problems or providing solutions to challenges using machines [9; 10]. For such machines to be able to handle given problems, they must have learnt about similar problems –supervised learning or have no clue (*or model*) – unsupervised learning. Generally, with machine learning, a computer is not explicitly programmed but an algorithm is given it to train a model on, with samples. Given the trained model, it can

then infer or predict about closely or unrelated sample (test) input as being in the class of any of the pretrained classes or not, and if present rightly or wrongly classify it as appropriate.

2.1.1 Machine Learning Definition

Now, it is plausible to more closely look at what machine learning is. Michell [9,2; 15,1-10; 16,1-13] explains that a machine is said to learn from experience (E) relating to a given task (T), with its performance measured as (P), if its performance at the given task 'T'; as assessed by 'P', improves with an increase in experience 'E'. This buttresses the earlier overview that machine learning is the ability of a machine to get better at tasks it is assigned to do with learning. Furthermore, learning makes better. This is the key clause here. In learning the machine essentially follows a given algorithm.

2.1.2 Supervised Learning

This is when the machine was given samples; 'taught' what was right, and which was not; and builds a model, by means of an algorithm, on that. Using this approach, the machine typically produces a model. This model can be used, not for training, but predicting a test input (an image in this project). This way, it is said they had been trained – learnt from experience. In this instance, the training is said to be supervised [11,5-11; 12,10] – machine shown the right answer (precisely a model – not necessarily a certain answer) to solve problems like those it had seen during training.

2.1.3 Unsupervised Learning

In unsupervised learning, the computer has samples (*or rightly a population*) to work with but is not pretrained. This implies it has an algorithm that can at the instant infer what is appropriate, e.g. where should a node in a network be – k-nearest neighbour (k-NN) or k-Means algorithm is an example. Once done, a result is obtained, and no model is given or produced – pattern, clusters and correlation are the only reference here and no label to samples given. This implies the result is what is desired, and it cannot be used inferring where a new node should be if expansion is, for instance, desired. In such situations the algorithm will have to be run again on all the dataset including the already known or established nodes. This is a good approach in graphs and data centre design. In unsupervised learning, at the start, the computer has no model on what the right answer could be from prior knowledge of similar dataset – no pretraining – to get the desired result, unlike in supervised learning where a pretrained model is used in getting the result. [11,5-11; 12,10.]

2.1.4 Defining Deep Learning

Deep learning is of the family of machine learning methods. It is centred on a set of algorithms that learn data representation by having high-level abstraction of data so presented using model architecture having multiple non-linear transformations. This is opposed to task specific algorithms. DeepLearning [43] argues that this moves Machine Learning closer to ones of its original goals – towards artificial intelligence (AI). [42 – 45.]

2.2 **Machine Learning & Deep Learning Breakthroughs**

Of the senses of perception, harnessing the novelty of human (*or animal)* vision and making machines able to mimic same for use in comparable accuracy to humans has eluded humanity for centuries. This was so because of a combined effect of the complexities involved from the computation cost from both ends – time (required threads for computation) and memory —both acting as determinants or limiting factors. Before continuing, a programming application interface that is optimized and not native is discussed under CUDA in the next paragraph.

CUDA. This is a programming platform that is often used in machine learning. It allows for parallel computing and provides a ready application programming interface (API) to the programmer. Compute Unified Device Architecture (CUDA) combines various architectures, as DirectX3D for instance would typically be for Windows ® platform. CUDA can run on all machines – supposedly with Nvidia®. It gives software developers and software engineers an interface to use a CUDA ® enabled graphical processing unit (GPU) and have a rich access to the compute kernels with increased number of threads – smallest units of computer's processing engine room. This removes the complexity associated with other platforms. [7.] This could be an invaluable and incredible tool in deep learning, from this project's work and results.

However, with the advent of GPU – supposedly an accidental computer evolutional breakthrough by result hungry, determined, and resolute computer science researchers, machines are ever becoming more capable; it is alluded they have a limit – Moore's rule –but computing power has increased. Using CUDA [7; 14; 15] and its optimized game engine processors – graphical processor unit (GPU), it is possible to have an increasingly huge number of available threads – in the kernel, to process the intensively large amount of data. At the time of writing, CUDA is widely implemented or used in the implementation of NVIDIA® GeForce GPUs used in most game engines and known for its unified ability

to seamlessly compute huge amount of data with some of the best processing power in the range of 2.4 trillion executions per second; yes, may be expensive and do require a good processor alongside, such as an i5 or an i7 intel® Quadcore™ processor used in this project.

Nevertheless, the ability of CUDA® to readily provide the kernel, as well as its advantages, to the programmer and expose its multi-threading capabilities does not necessarily require GPU [15,4-6; 16,1-12], if the amount of processing required is minimal. A case for this would be if the image sizes are very small and the training or design set is relatively small; and another could be the training is done by an external server in which case it could be the Google Cloud Platform – precisely using the Google Vision API ® or any other cloud vision API provider.

Briefly introducing images, they are matrices of width, length, depth and possibly channels (coloured images). This implies they are simply numbers in the range of 0-255 (2^8 – a byte). Using this knowledge, image data can be processed using optimized algorithms [17; 18; 19] which range from Support Vector Machines (SVM) possibly with Gaussian kernels though computationally expensive, Logistics Regression, Principal Component Analysis (PCA), Single Vector Decomposition (SVD), Convolutional Neural Network (CNN) – adding layers of unit neural networks on each other; multiple layer neural networks, and its progeny Region-based Convolution Neural Network (R-CNN). The latter does have an approached that is optimized in that vast amount of data contained in the image is tripped to regions; possibly using a stride employing average or max pooling, having the most significant bits in the region. Thus, helping to check, effectively, the amount or number of useful (desired) regions to process in training, validating, or testing the algorithm. [19 – 29;32 – 41.]

2.3 Algorithms

Algorithms are crucial part of computer science. They are at the heart of most machine learning problems and consequently, deep learning, and even unsupervised learning. Section 2.3.1 will expound on what an algorithm is.

2.3.1 What Is an Algorithm?

To reiterate, an algorithm, as stated in opening paragraph of this chapter, on page 4, states the rudimental steps to be followed; meticulously or otherwise, in solving computer

calculations, data processing and or automated reasoning tasks. For a software engineer, it is the code path to be followed – including conditions, to achieve a desired result. This can span a simple greeting program to a graph that has all airports of the world and calculates the shortest weighted path between any two given airports – including stopovers.

In the following subsections, some algorithms will be briefly discussed that are central to machine learning and closely related to deep learning. This will be selective to suit those that tend to weigh in on artificial neural networks or object detection or recognition where the core of this project draws most of its strength.

2.3.2 Artificial Neural Network – ANN

This is the bedrock of deep learning. Briefly, Haykin [10,2] defines a neural network as an expansive parallel distributed processor comprising simple processing units, which has an innate propensity for storing experiential knowledge easily source when needed. In his model, he further argues that it is much like the brain in two respects. An artificial neural network is often called a neural network – NN, for brevity. [23,12-32; 29,2-3; 46,2-3.]

- The environment gives the network knowledge through the experience it relates in a learning process.
- Inter-neuron connection strengths, called synaptic weights accounts for the storage units for the acquired knowledge.

An algorithm comes handy in machine learning and by necessary implication, deep learning. The process through which a neural network or any machine learning process acquires its experience is called, an algorithm. ANN is the foundation of neural networks and derivative algorithms such as CNN and R-CNN.

2.3.3 Support Vector Machine - SVM

SVM is a feedforward network. It is used in object recognition [37] – [42]. Its developer is Vapnik. [10,318.] It is essentially an implementation of the method of structural risk minimization. Its principle is on sum over the training error of the test data, as well as the Vapnik–Chervonenkis (VC) dimension. The training error is the observed difference in the Euclidean distances [47,8] of the actual data value and the observed or reported value from the algorithm. Simply put, the error is how far off from the actual data value the observed value is. Though not directly related to R-CNN or deep learning but it does aid in understanding the design and complexity of deep net architectures.

It chiefly employs the inner-product kernel between a supposed support vector (x_i) and the vector (x) from the input space, in its construct. Given the inner product kernel generation, a handful of algorithms could be obtained. Three of these include:

- Polynomial learning machines
- Radial-basis function networks
- Perceptron with a single hidden layer (2-layer)

Unlike a backpropagation algorithm – used in training a multilayer perceptron, this is used to train a simple layer perceptron. It also has a wider and more generic application in training a learning process. As explained previously, it is a feedforward algorithm. [46 – 54.]

To optimally design an SVM, one could imagine the construction of a decision boundary, often called decision surface, that in the input space is non-linear, though its image in the feature space is guaranteed to be linearly separable. This a critical area of interest in R-CNN, the foundation of object detection. [10,332-335] It is based on two assumptions. However, we state the mathematical expression first.

With a training sample $\{(x_i,d_i)\}^{N}_{i=1}$, the Lagrange multipliers are sought that maximize the objective function given below – objective function:

$$Q(\alpha) = \sum \alpha_i - 0.5 \sum \sum \alpha_i \alpha_i d_i d_i K(x_i,x_j) \quad (1a)$$

Given these two constraints:

$$\sum \alpha\, d = 0$$

$$0 <= \alpha_i <= C$$

C is a user defined positive parameter. Now, we can quickly state the optimum solution for a weighted vector (W_0):

$$W_0 = \sum \alpha\, d_i x_i \quad (1b)$$

Where N_s is the number of support vector units. [11,24-30.]

As a quick overview, Table 1 below expounds on inner product of kernels.

Table 1. Synopsis of Inner Product Kernels adapted from [10,333].

SVM Type	**Inner-Product Kernel $K(x,x_i)$, i = 1, 2, …, N**	**Insights**
Radial-basis function network	$Exp(-(2\delta^2)^{-1}\|\|x - x_i\|\|^2)$	$\|\|\Delta x\|\|^2$ – Euclidean distance, width δ - specified as common to every kernel

Polynomial learning machine	Table 1. $(\mathbf{x}^T\mathbf{x}_i + 1)^P$	The user specifies, a priori, power **p**
Single hidden layer	$\tanh(\beta_0\mathbf{x}^T\mathbf{x}_{ii} + \beta_1)$	Mercer's theorem partly for β_0 and β_1

Table 1 above depicts a balanced overview of the three Inner-Product Kernels described earlier. We can see the transpose of the training dataset (X) and an instance of the training set Xi as a matrix product in both polynomial learning curve and single hidden layer SVMs. The insight explains the difference or change in X as the Euclidean distance. This further explains how the Euclidean distance is computed – square of the 'deviation' – how far apart, from the test data instance. [10,332-334.]

Figure 1 below further enunciates the single hidden layer as a characteristic SVM architecture.

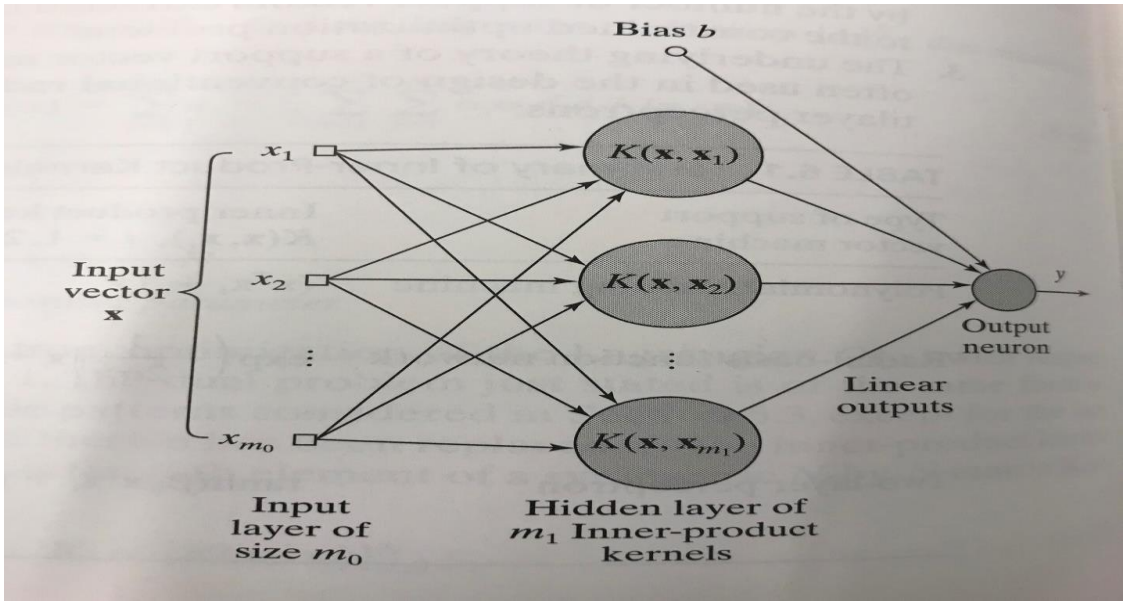


Figure 1. SVM Single Layer Architecture. Reprinted from [10,334].

In the figure above, it is evident that each training instance is tested against the entire training set in the hidden layer K(X,Xi). The hidden layer accounts for the complexity here in neural networks. Also, in the input layer, the input vector is accessed from x_{mo} through X_1 in same order as the hidden layer kernels arrangement. [10; 34 – 37.] As may be suggestive, the multi-layer architecture uses another approach or algorithm [46; 49,3] called, backpropagation [47; 52; 53].

2.3.4 Principal Component Analysis

This is a classical unsupervised learning algorithm. However, it is also used in object recognition or detection though not optimized for real-time applications. [32 – 36.] The

ingenuity of neural networks is in the ability of the network to get better at tasks (improve), with increases in learning or experience, much as training might be computationally expensive. Here, eigenvectors are extensively used. Just before that, quickly reviewing the intuition behind self-governing or organising could be helpful; this is an unsupervised learning – the machine has no labelled sample but gets better at the task afterwards – training or learning iteration. In PCA, as well as other unsupervised learning algorithms, the approach is to identify significant features and or patterns in the input data. Can one ask how it then learns? Yes, it is a good question to ask. It does follow a given set of rules (algorithm) of a local nature (in situ) that mimics a mapping of input to output. Using this pattern, the synaptic weight of the neural net is continuously modified (improved on). [10,392; 30,5-6; 51,1-11; 53,17-20.]

The key reason or rather answer to how self-governing algorithms work and a case study in point with PCA is highlighted in Turing's 1952 article on Chemical basis of Morphogenesis. It states that, "Global order can arise from local interaction". By this, it is explanatory that stripes, spots, and spiral natural patterns may arise naturally in a body if the body is in a uniform and or homogenous state. This is what the human intelligence; as exemplified by the brain, builds on not to mention the much talked about neural networks or artificial intelligence. [10; 12; 57.] The interactions can be further seen in two main levels of interaction at which network organisation is carried out. They include:

Activity. Each activity in some patterns is produced as an accommodation mechanism incurred by a network's input signals.

Connectivity. Network synaptic weight are altered in consonance with neuron signals registering itself in the activity pattern. This can be accounted for in response to synaptic plasticity. [10.]

Briefly, we look at some of the basic principles central to self-governance – unsupervised learning. The motivation here is that this unsupervised learning governing principle will, hopefully, not be mentioned hereafter.

2.4 Unsupervised Learning Governing Principles – PCA Case Study

1. Synaptic weight tends in the path of self-amplification when changes occur.
2. Scarce resources propel competition among synapses; the fittest survives – synaptic plasticity at play.
3. The changes in synapses' weight lead to cooperation.

4. Structure and layering (order) observed in activation patterns amount to redundant information incurred by the network in the form of knowledge. This is a prerequisite for a system to be able to self-govern or organise. [10; 15; 30.]

In summary with PCA, kernel PCA is quickly summarised showing how to effectively approach it.

- Given the training examples (x denoting the data vector) in $\{x_i\}_{i=1}N$, the N * N matrix can be computed using:
  - $K = \{K(x_i,xj)\}$, with (2)
  - $K(x_i,x_j) = \phi^T(x_i,x_j)\ \phi(x_j)$ (3)
- Using $K\alpha = \lambda\alpha$, (4)
  - solve the eigenvector problem – K (kernel matrix), α (eigenvector), and α (eigenvalue)
- Normalize computed eigenvector requiring,
  - $\alpha_k^T\ \alpha_k = (\lambda_k)^{-1}$ for k = 1, 2, …, p; (5)

    when arranged in descending order, p is the least non-zero eigenvector of matrix K.
- Extracting principal component of test point x, the projections are thus computed,
  - $a_k = (\delta q_k)^T\ \phi(x_j)$, (6)

    $a_k = \sum_{i=1}^{N} \alpha_{i,j}K(x_i,x_j)$, for k = 1, 2, …, p with $\alpha_{i,j}$ being the jth element of the eigenvector α_k. [2.]

A figure can be very graphical and better elucidate explanations. So, Figure 2 below buttresses these four key steps to computing PCA.

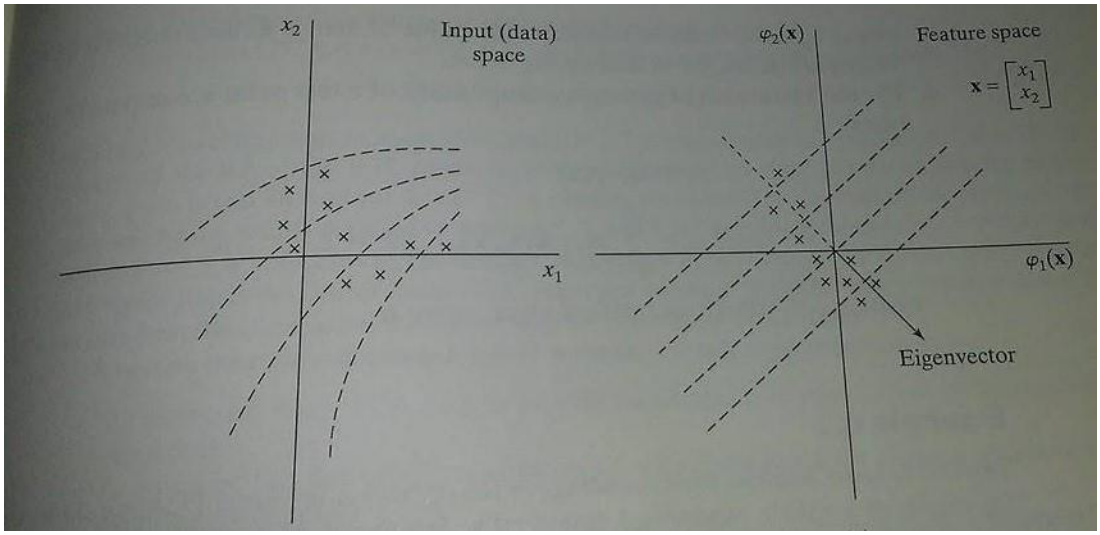


Figure 2. Kernel PCA illustration: 2-dimensional input and feature space respectively. Reprinted from [10,435].

In Figure 2 above, to the left depicts the 2-dimensional input while on its right shows a feature space mapping in the PCA illustration given.

2.5 Neural Networks via Feed-forward Networks

Unlike some linear regression algorithms discussed earlier including SVMs and PCAs, which chiefly were linear combinations of models using fixed basis function for classification and encompassing regression algorithms, feed-forward neural network, also known as multi-layer perceptron does not show the inherent limitations expressed or observed in SVMs or PCAs – dimensionality constraint [59,226; 60,15-17]. Feed-forward neural net is the most successful model that easily adapts on training or during same using a basis-function. The multi-layers do consist, essentially, of multiple layers of logistic regression models having continuous nonlinearities as opposed to multiple layers of perceptron. [59; 60; 61.] So, the name does not explicitly tell it all.

Bishop [59] argues that the attempt to propose a mathematical identity for the representation of information processing in biological systems led to the coinage of the term, 'neural network'. He equally alludes to the fact that some may have no plausible biological correlation whatsoever. He proposes that neural nets should be used as an effective model employed in statistical pattern recognition at the very level of data construction or extraction.

In proffering a solution to a nonlinear optimisation problem, an evaluation of the likelihood function, its log, as expressed in the network parameters, can help unveil an appropriate analysis. In doing that, the technique of error backpropagation [59,226-228; 62,52-54] needs be introduced. When extended error backprop can give room for evaluating other derivatives such as the Jacobian and Hessian matrices [59]. Mathematically, the regression and classification for linear models is of the general form stated below as explained in a preceding paragraph in this section (on page 12).

$$y(x,w) = f\left(\sum_{j=1}^{M} \omega_j \emptyset_j(x)\right) \quad (7)$$

the summation for which '*f*' is a function is a nonlinear activation function [59,227] for classification, and an identity for regression. The basis function is defined thus,

$$b = \emptyset_j(x) \quad (8)$$

The aim here is to is to make the basis function depend on parameters that can be varied, alongside the constant $\{\omega_j\}$ while training. This is one of several ways to achieve same result. Neural nets are structured to follow the algorithm presented in equation (7) above. This makes the linear combinations of inputs parameters for the non-linear function each unit of the basis function. [17; 18; 62,52.]

Now, a quick consideration of how a neural network model is constructed is presented. This involves transforming the model in various functional approaches. If we took the input variables as x_1, x_2, …, x_D, the linear combination of the input parameters can be expressed as below:

$$a_j = \sum^{D}_{i=1} \omega_{kj}^{(1)} x_i + \omega_{j0}^{(1)} \quad (9)$$

The (1) describes the layer is the first layer and *j* ranges from 1 through M. [50.] In this case, w_{ji} and w_{j0} refer to the weight and bias parameter respectively. As explained for equation (7), the a_j is the activation function. To further expound on Figure 1 given in page 10 above, we quickly see how SVM relates to a neural network. Figure 3 below presents an explanation of equation (11) given after it.

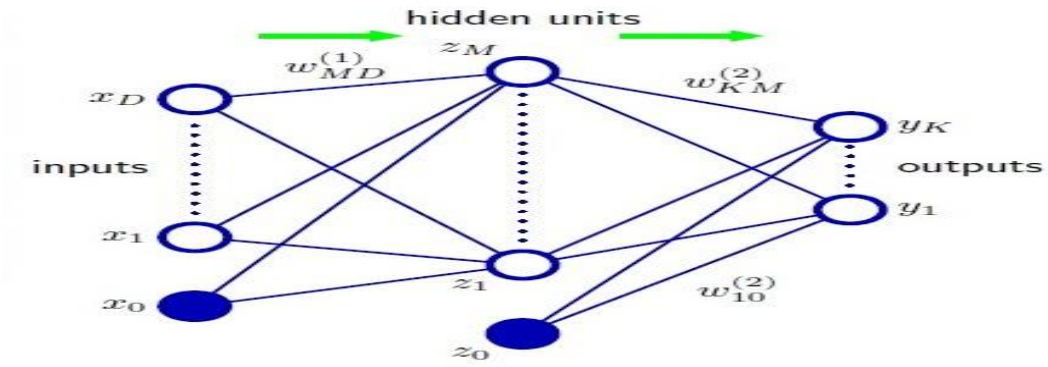


Figure 3. Network diagram for a two-layer neural net explaining equation 11. Reprinted from [59,228].

$$y_k = \sigma(a_k) \quad (10)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \quad (11)$$

The nodes represent the input, hidden, and output layers while the edges denote the weight nodes, for instance, the weight between x(0) often one (1) bias unit, and hidden layer z_m will be $w^{(1)}{}_{m0}$ as the edge (or link) between X_D and Z_M connotes. Back to equations 10 and 11, equation 10 is a standard regression problem where $y_{(k)} = a_{k.}$ On the other hand, the prominent activation function in logistic regression, is used, for a multiple binary classification problem, which is expressed in equation (11) above. However, if a multiclass problem was presented, a softmax activation would be required, as stated below.

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \quad (12a)$$

$$= \frac{\exp(a_k)}{\sum_j \exp(a_j)} \quad (12b)$$

Incorporating the various stages above, the prevailing network functions given in equation 13 below is obtained.

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \quad (13)$$

Expectantly, this is the last algorithm that will be discussed prior to getting into Region-base Convolutional Neural Network (R-CNN) [63; 64; 65]. However, quickly explaining this, the weight and bias of the various units have been grouped into a vector represented here by w. This controls the nonlinear function made of the sets of input and output variables, $\{x_i\}$ & $\{y_i\}$ respectively.

A network diagram can be gotten from the preceding function as shown in Figure 3 above. Having to analyse the function above requires forward propagation of data into the network. They do not represent probabilistic models as such, as the internal nodes are more representative of deterministic variables than stochastic ones – stochastic gradient descent (SGD). However, probabilistic interpretation can be given to neural nets – in a different instance and algorithm.

A property of feed-forward networks, worth considering, as it plays a role in Bayesian model comparison, is their weight-space symmetries [62,49]. Bishop [59] explains that this property ensures the weight vector (w), and its component units, all give rise to the same mapping function from input to output. Reviewing Figure 3 above, if all signs were changed and with an initial 'tanh' activation function, the sign of the activation (a) function is now reversed, as it is an odd function. [59; 60.] To explain this further, let us quickly look at Figure 4 and Figure 5 below.

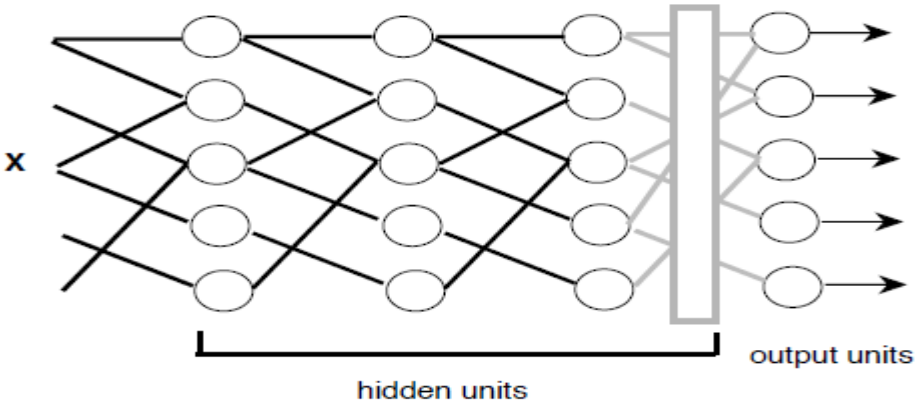


Figure 4. A Layered, Feed Forward Network. Reprinted from [60,47].

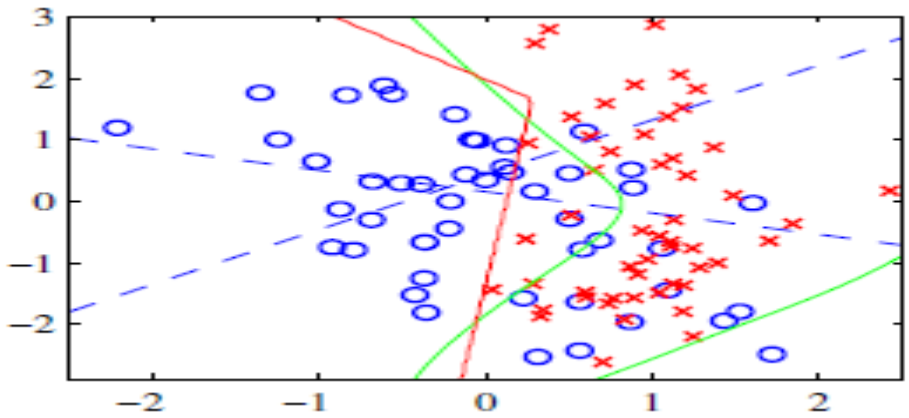


Figure 5. A solution to a two-class classification task having synthetic data using a neural network with two inputs. Reprinted from [59,232].

Figure 4 shows a layered feedforward network with three hidden layers in threshold logical units network (TLUs). It has 5 input units and same number of output units. Figure 5 above presents a logistic sigmoid activation function. The dashed lines (blue) indicate a *z* value of 0.5 contours per hidden unit. Also, the decision surface is represented with *y* = 0.5 per hidden unit in the network. Besides, the green line is a benchmark for comparing the optimal decision boundary obtained across the distributions employed in the data generation. [59 – 62.]

Error Backpropagation

To get a good grasp of backpropagation, a pictorial view of a multilayer perceptron might help ignite the curiosity. Figure 6 below depicts it diagrammatically.

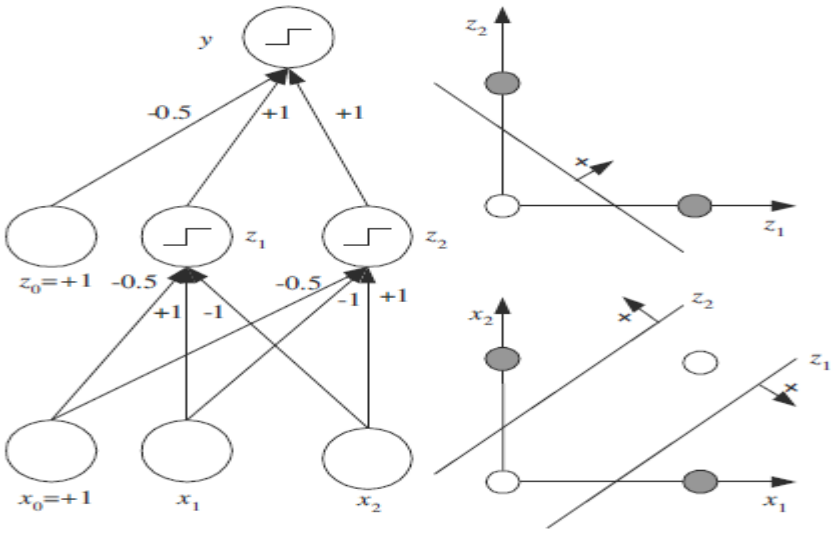


Figure 6. Multilayer perceptron that unravels the XOR problem. Modified from [63].

In Figure 6 above, both hidden and input units have their threshold activation functions having threshold at 0.

LISA [60] reiterates that backpropagation can be seen as a special case of a chain rule derivative. This can be viewed as taking a convenience path to reach a goal or a given destination. Considering Figure 4 on page 16, should an error arise there could be several ways of making a correction to any observed error in the pattern. A generalised approach could be to use the Widrow-Hoff method of gradient descent which is viable and generalised. [60,52.]

Shai et al. [61,269] maintains that backpropagation efficiently calculates the gradient. He stresses that obtaining the error inherent or observed in the neural net can be daunting, as it relates to the network's parameters. The distinguishing feature in training a multilayer perceptron compared to training a perceptron is the nonlinearity of the output's function compared to that of the input. Ethem [62,249] concurs with LISA [60,35] on the chain rule in their mathematical expression stating the algorithm used in computing the backpropagation. Taking the hidden units as inputs, the second layer is taken for perceptron. Mathematically, using the chain rule, the gradient (δE) can be calculated thus:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}} \qquad (14)$$

Z_h is input, the first-layer weights w_{hj}, y_i expected result on y. Using the chain rule stated above, using equation (14) above the error on the gradient can be calculated.

2.6 Convolutional Neural Network

Convolutional neural networks (CNN) are neural networks that are layered or built on each other. That explains the term, ‘convolution’, in its name. They are widely used in object recognition and detection tasks. They have a handful of advantages that make them tick when compared to other techniques [20,1].

2.6.1 What Is a Convolutional Neural Network (CNN)?

To begin, let us consider the terms in isolation or uniquely. The first pick here could be, ‘neural’. See Figure 7 & Figure 8 below.

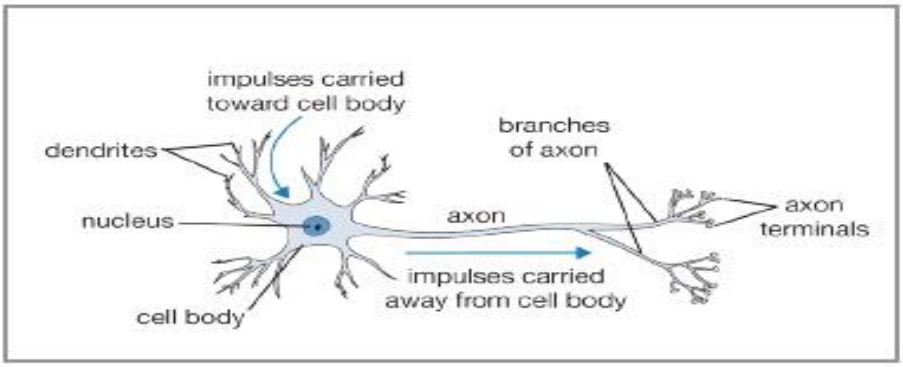


Figure 7. A neuron with axon and dendrites. As depicted in Shai [61,2].

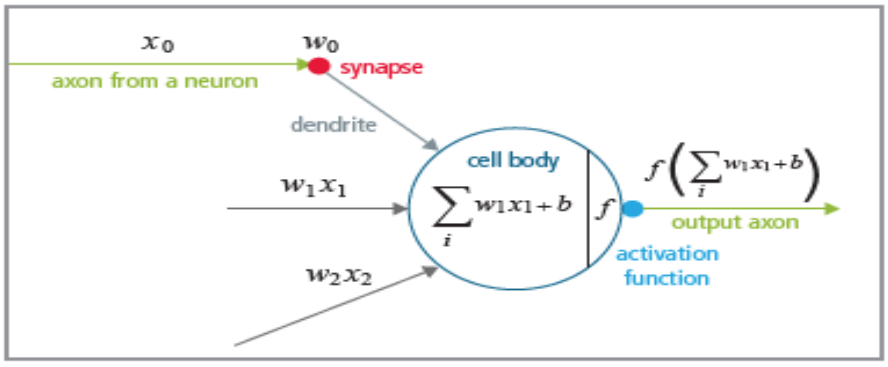


Figure 8. An artificial neuron with axon and dendrites. Adapted from Shai [61,2].

Neural is the biological analogue of a neuron (*not minding the adjectival and noun differences*). A neuron, in its functionality, generates synaptic electric impulses for communications or propagating impulses (messages) from synapses to dendrites as Figure 7 makes evident. Neurons help animals, humans included, in communication, or more precisely relating with their environment – vision, irritability, task or smell and their ilk. The ability to have or obtain that from the environment (perceive) and interpret same is called perception. In computer vision, that passes for perceptron; see Figure 8 – a unit of inter-

connected nodes (*neurons*) that help propagate an understood message. I would argue that this ability to so interpret the perceived information is what has eluded computer science in general prior to the discovery of the evolutionary breakthrough in computer science – the GPU. When considering 'Convolution', the brain serves as an example. With its sinuous fold that most likely house neurons and the clear visible ridges of the surface, the thought of neurons interaction or layers can be suggestive. [66.]

Network on the other hand needs no introduction. We all know it is a collection of nodes that can at least interact. A case in point would be a DAG – a directed acyclic graph; the nodes point or move in one direction. An airport may not explain the DAGs, but it does present an analogy of a graph, as it were.

With the key terminologies elucidated, Samer et al. [67,1-2] states that a neural network is a system of interconnected 'artificial neurons'. As in a weighted graph, the connections (edges) have numeric weights. These are adjusted during training to ensure an appropriately trained network will respond correctly; in its major part – mostly likely pass; though not expected to be perfect, when presented with an image or object to recognise. The diagrams that readily depict this are Figure 1 & Figure 3 above; if we took the labelling such as inner kernel products of (in the case of Figure 1). However, Figure 9 below demonstrates a generic artificial neural network architecture.

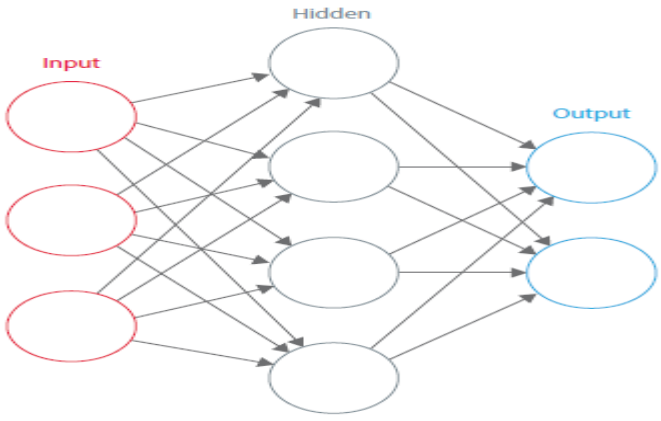


Figure 9. An artificial neural net architecture. Courtesy Shai [61].

The last figure before now shows input, hidden, and output layers; and 3, 4, and 2 units in each respectively. Analysing this architecture, the layers are arranged following a pattern where the first (hidden layer) enables it to detect a set of primitives in its previous layer (the layer that connects to it – input layer above). Same can be said for the second layer (output layer in this case). The network can in practice, be made up of multiple

layers (hidden layers), prior to having the output. With increases in the layer comes complexity in terms of time and computation cost – memory as well as time complexity. On the average, one typically sees 5 to 25 layers (including input and output layers) in most CNN architecture with industrial applications. [61,1.]

In computer vision or neural net computational models in general, a propagated signal, for instance, x_o, in interacting with adjoining dendrites does so multiplicatively. That implies it is a weighted interaction ($w_o x_o$) of a synapse's strength with its adjacent dendrite's synaptic strength. If the threshold is ever reached, at whatever time that occurs, the neuron fires a signal through its efferent neuron when it deduces the threshold has been reached on surpassed. Unlike in biological systems, the exact timing is not so much of interest nor critical in computational modelling – the frequency of firing should be strong enough to communicate any meaningful information. This is essentially what an activation function ***f*** such as a sigmoid does. [61,2.]

A CNN is a unique instance of an artificial neuron depicted above. As explained on page 21, a CNN has one or more convolutional layers, with each usually – not necessarily, having a subsampling layer. These subsampling layers are either followed by another subsampling layer or a fully connected layer – always at the end of the network just prior to the network's output layer. Explaining Figure 8 on page 20, we saw how a neural network mimics the human brain and its functionality. That is a ready motivation for studying CNN. Figure 10 below presents a block diagram of a CNN.

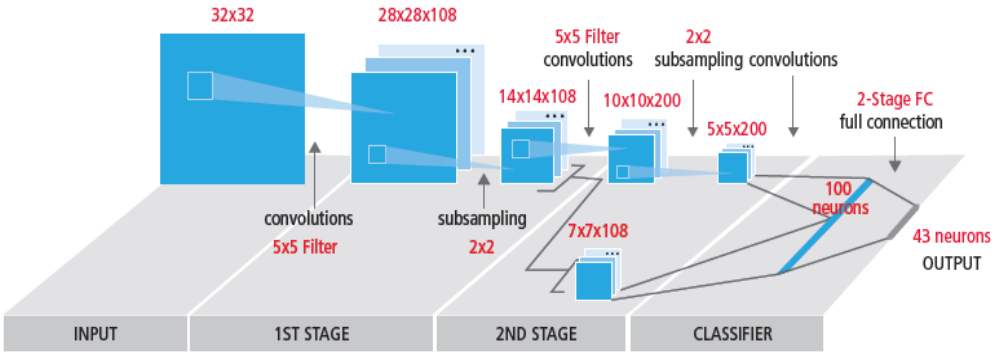


Figure 10. An example CNN block diagram. Reprinted from [67,3].

Figure 10 above helps explain the extractor feature of convolutional layers, though they are not manually designed. Convolutional layers usually have a field day at doing this – obtaining the local feature, by limiting the receptive fields of the hidden layers to be essentially local. These are depicted in the block diagram in Figure 10 above. [66.] [67.]

However, there is much more to say on motivation. Of late, CNN finds application in image and pattern recognition, object detection, speech recognition and analysis, natural language processing, and video analysis amongst others. CNNs usually have their weights of the convolutional layers employed in feature extraction, as well as the fully connected (FC) layers determined along the course of training. Over the years, the structure and architecture of CNNs have improved tremendously and this accounts for huge savings on memory and time in reducing the cost of computation. This has greatly enhanced performance of applications requiring local correlation (an example would be a picture or sound input application). [10 – 12;14 – 28.]

Furthermore, CNNs record some of the best results in terms of performance and accuracy of classification hovering over ninety-seven percent (97%) [61,3]. A common term for this is, 'Correct detection rates' – CDRs. Some of these data are found on MNIST database of handwritten digits, and NORB dataset of 3D objects with CDRs 99.77% and 97.6% accuracy in classification respectively [17] – [21].

2.6.2 Vision Algorithm Pipeline

This is often used in object detection or more precisely object recognition [53;68]. An example pipeline algorithm is presented in Figure 11 below, which shows a vision algorithm pipeline with 4 distinct stages.

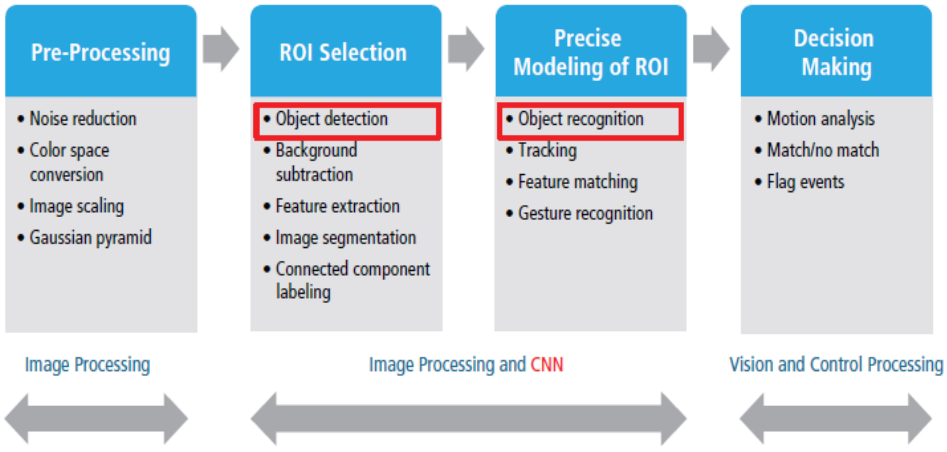


Figure 11. An image recognition vision algorithm pipeline. Reprinted from [67,4].

The first stage in Figure 11 is concerned with 'cleaning' the data – pre-processing, while the stage following handles down-sampling and presenting the regions of interest – ROI. The actual modelling takes place in the third stage and the classification then follows in the closing stage of the pipeline.

Assembling multiple and varied layers in a CNN requires building architectures that are complex and able to handle complex problems. The most common layers are reflected in the Figure 10. They include:

- convolutional layer;
- pooling or subsampling layers;
- non-linear layers; and
- fully connected layers.

Convolutional Layers

This layer obtains various features of the input. Often, the first of this layer identifies and collates low-level features such as edges, lines, and corners. Likewise, higher level abstraction in this layer obtain from its precursor higher-level features or identifiers. Using Figure 12 below, we can quickly deliberate on the process of 3D convolution used in CNNs. [67.]

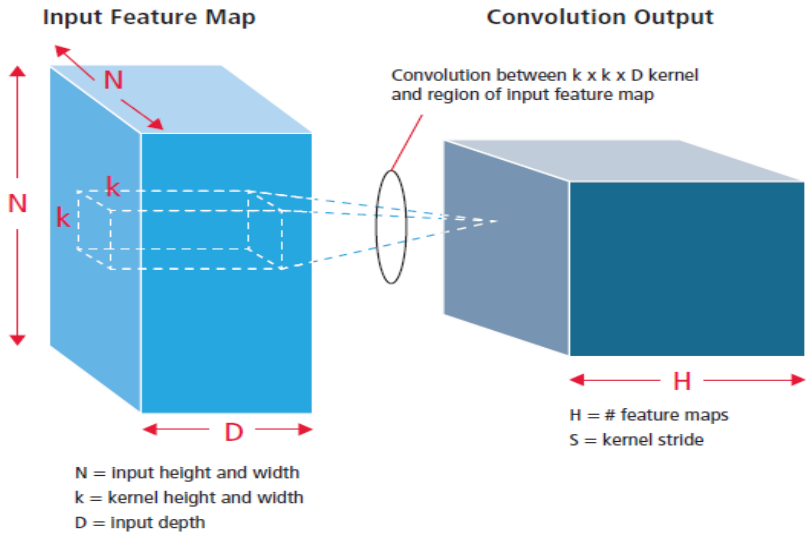


Figure 12. A convolutional process in pictures. Reprinted from [67].

Explaining Figure 12 above, the input is of size N * N * D and convolved using H kernels, each having size k * k * D independently. Using H kernels, H features are produced – understandably so, as each kernel produces one feature. Commencing from the top-left corner, every kernel is shifted rightward from left. This is done element by element – sequentially. On getting to the top-right edge, the kernel is moved downwards elementwise. This is then followed by the rightward movement from left. This is continued until the kernel gets to the bottom-right end (corner). For a case where N = 32 and k = 5, there would be 28 unique positions – horizontally or vertically. This defines the path of movement for the kernel as it moves from the top-left corner to the extreme bottom-right

corner. Sequel to these positions, each output's feature would contain 28 x 28 elements. The 28 is obtained by (N – k +1). Using a sliding window, the resulting matrices – input elements k * k * D and kernel elements k * k * D, multiplied elementwise and stored. The accumulator – multiple is needed in creating an element of an output. [67,4-5.]

Pooling Layers

Often the information to process is inadvertently huge and some of it is not in any way useful or required. This layer looks at downsizing (pooling) the regions that portend most significant bits. There are two ways to do pooling [52] and a third, though different, could be added:

- max pooling
- average pooling
- selective search [68]

In the first 2 instances, the input is divided into regions of non-overlapping 2D spaces as Figure 12 below would suggest.

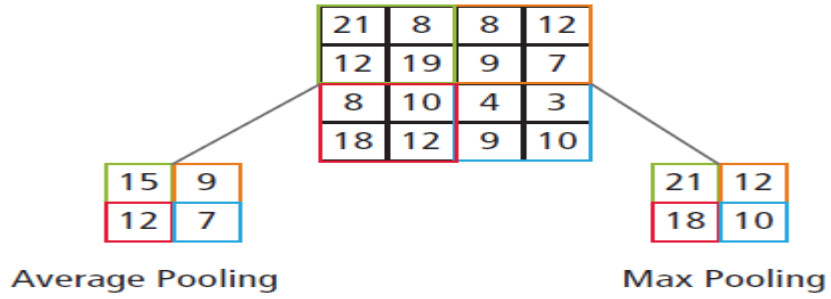


Figure 13. Max pooling & average pooling representation. Excerpt from [58].

In the first two rows, using a stride of 2, it is evident the maximum value for the first 2 x 2 grid is 21 and the second is 12. Same can be said for the last two rows 18 and 10. This is max pooling represented. On the other hand, using similar stride of two as depicted on figure 10, the average of the fist grid (integer) is 15 (*60/4*). Likewise, each of the other 2 x 2 strides is calculated. That is average pooling. Selective search is not exactly a pooling algorithm but an algorithm that is in the class of ROI algorithm where the region with the most probability of containing the required data is returned after the search [68] – see Figure 14 for details. Hence, selective search (SS) it is an algorithm that is very useful in object detection or objectiveness of a region.

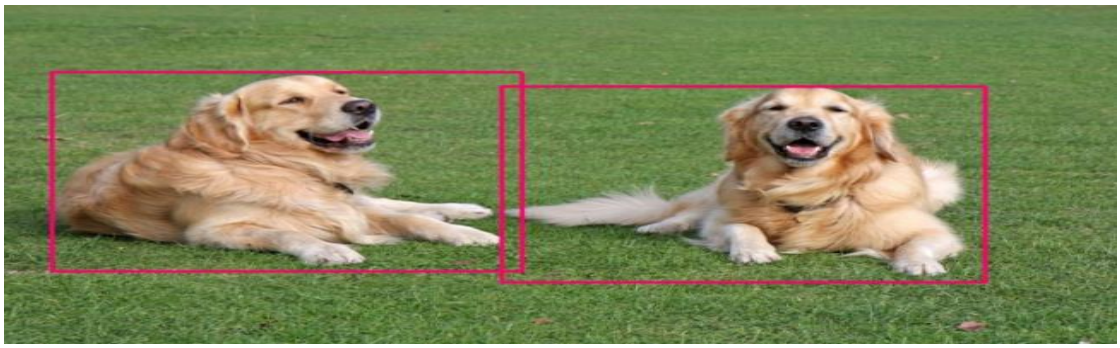


Figure 14. Object detected using selective search algorithm (SS). Image courtesy of [68].

Another approach to getting same image would be to use the sliding window [63] – a technique used in max pooling or average pooling. The sliding window is inundated with significant drawbacks. However, with Girshick's work [58], though he did not propose a network of the regions in his work, this algorithm has been enhanced into the region proposal network (RPN) algorithm [64]. Figure 15 below expounds on this pictorially.

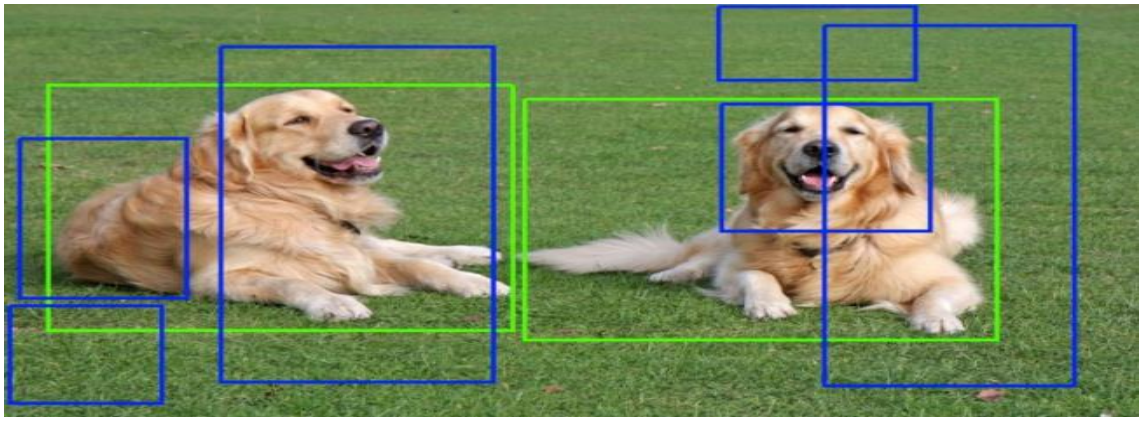


Figure 15. Region proposal objectiveness suggestion. Image copied from [68].

It is however improved on by the work of Shaoqing et al. [64] titled, 'Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In Figure 12 the regions suggested or proposed can be clearly seen. There are many regions, including some, that do not wholly show or cover the entire object. Of these, the strongest in suggestion reflecting their probability of objectiveness is presented as the object the algorithm employing them is to process or identify.

Non-linear Layers

To trigger unique identification of likely features on each hidden layer, neural networks, including CNN, do rely on non-linear 'trigger' functions. Some of the function that may be used by CNN include:

- rectified linear units ReLUs;
- hyperbolic tangent;

- absolute of hyperbolic tangent;
- sigmoid; and
- continuous trigger (non-linear) function

These can be used in the efficient implementation of the desired non-linearity. [67.]

ReLUs

The function implemented by a ReLU is of the order y = max(x,0). So, both the input and output sizes of this layer are essentially the same. The decision function is increased by a ReLU. It does this without altering the receptive fields of the convolution layer. With ReLU, the time required to train is shorter. So, we can say it is easy per time complexity to train. It is many times faster compared to most other functions. Figure 16 below enunciates the functionality of ReLUs. [64–67.]

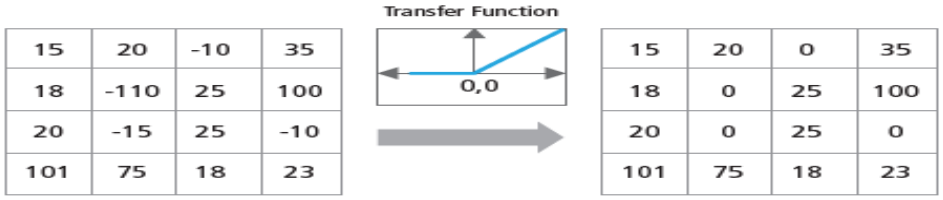


Figure 16. Showing how a ReLU functions. Reprinted from [67,6].

The transfer function is plotted over the arrow as depicted in Figure 16. The maximum or higher of the values (x,0) is shown on the right to the arrow as indicated in Figure 16 above. One more algorithm we will be considering is the last on the list.

Continuous Trigger (Non-linear) Function

This, in each of its features, operates elementwise. This sums up those above it on the list save the first as depicted on figures 17 – 20 below. The plots on figures 17 through 20 are modified from Samer et al. [61, 6].

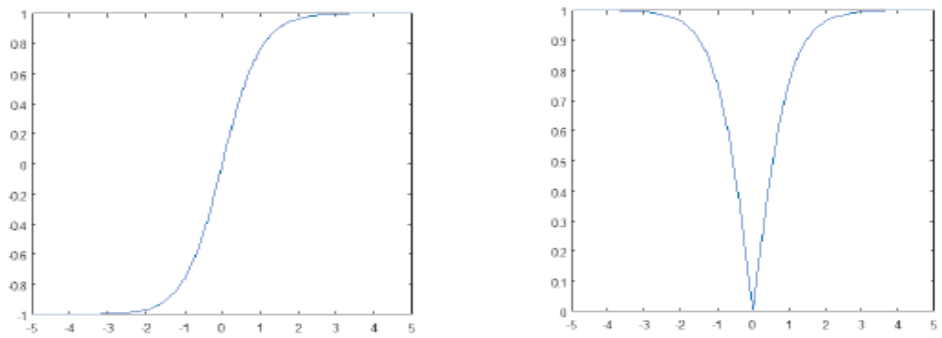


Figure 17. Hyperbolic tangent function plot. Figure 18. Absolute of hyperbolic function plot.

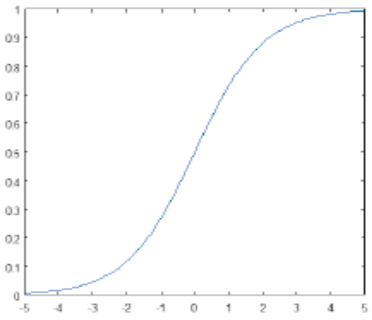


Figure 19. Sigmoid function plot.

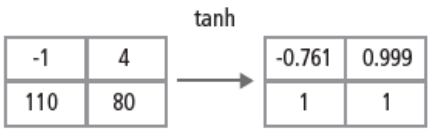


Figure 20. Tanh processing – pictorially.

These are essentially optimization function often used in machine learning. Particularly useful in activation for most ML processes is the sigmoid function.

Fully Connected Layer

Usually, this is the final layer that is employed at the final stages of a CNN. This layer sums over the weights of the previous layers' features and indicates the blend in determining the target in the output result [59,267-361; 65,2-9; 67,6-8]. Samer et al. [67,7] strongly holds that the weights of the previous layer's elements are used in computing each output's feature up to each layer in the fully connected layer. Citing their work and reprinting a figure, Figure 21 below, they demonstrate what a fully connected layer could look like.

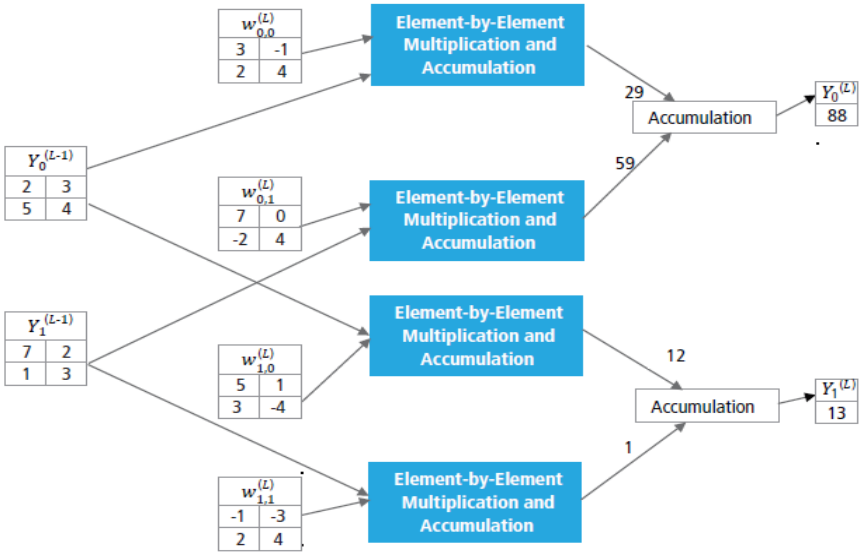


Figure 21. Plausible way of processing a fully connected layer in a CNN – copied from [67].

Besides, all the reviewed similar work in this field, Figure 21 above significantly explains, pictorially, a fully connected layer well.

3 Architecture & Software Implementations

This chapter presents details of the four (4) software developed during this projects in the sub-headings **Project Implementation**, **Methodology in Model Training**, , **Mobile App (ODM) Design – Android® Platform**. It also explicitly details the algorithm of the model training software. For the obstacle (object) detection software, a deep learning algorithm, possibly Region-based Convolutional Neural Networks (R-CNNs) was used by the model provider Caffe® [6]. Besides, architecture of the trained model, and the methodology are explained.

3.1 Object Recognition vs Object Detection

Object recognition. This involves deciphering what instance of objects (classes or images) an object in a frame (photograph) belongs to, for instance, is it a car or a person? Here the answer, 'A car' or 'A person' suffices as an answer while outputting the confidence with respect to pattern matching result of the input's (image) data. [42; 45; 68.]

Object detection. This involves determining what objects are where (location or coordinates) in each frame – here the algorithm keeps track (in memory) of the locations and supplies the object recognition details alongside. So, object detection encompasses recognition and relates the coordinates or localization where an object was detected in the frame. [42; 58; 64; 68.]

Region Proposal Network

This is an R-CNN enhancing algorithm proposed by Ren et al. [65]. It uses Regions of Interest (RoI) in building a network which has, as against the entire image's dataset, the regions that are most likely to be of interest in determining its shape, colour – basically properties. Non-maximum suppression (NMS) could be used to avoid failure to detect regions or minimize detection rate or accuracy.

Faster R-CNN

Briefly Faster R-CNN combines RPN and Fast R-CNN in modelling a faster network, and one that is capable of real-time solutions in object detection and recognition. Microsoft Research earlier developed fast R-CNN® and used RoI (regions of interest) but did not necessarily build a network on them. It is a commonly used algorithm in object detection in real-time application, though hard to replicate or train. However, other models using similar approaches have equally comparable results and some can even be faster and

applicable in real-time calls like Caffe model – used in this project. Here spatial pyramid pooling (SPP) could be helpful. [50; 58]

3.2 Project Implementation

Convolutional neural network (CNN) – a default algorithm for object recognition, was used in training the five (5) models. Using the CIFAR-10 dataset repository with 60,000 images of size 32*32. Fifty thousand (50,000) images were used for training and ten thousand (10,000) for validation (testing). The learning parameters and the epoch size – the number of samples (sample size) of the input, in this case images– either training, cross validation or validation (testing), were critical to having significant accuracy per 10,000 images tested and with confidence of each tested image. The library used was CNTK – this is the initial platform on which Faster R-CNN was first developed. This was a plausible reason for choosing it and it is well implemented in other languages such as BrainScript, C++ and C. Most libraries are usually confined to Python, nevertheless a fast and good programming language, and the performance of the library was satisfactory, though none other library was tried as a comparison. Alternative libraries would have included Google® Tensorflow, Keras, or PyTorch [1– 6; 56; 58; 65.]

The AVOD software used CV's deep neural network (DNN) library [6]. Also, it used Caffe® model – probably used RPN and Faster R-CNN, or essentially R-CNN in its training, and Caffe® provided weights for detection.

Briefly, we will, pictorially, have an overview of the training results. Prior to that, we look at the network architecture in details. Referring to Figure 11 (on page 21), there are four basic stages seen which include:

- Pre-processing
- Region of interest (RoI) selection
- Precise processing of RoI
- Decision making

3.2.1 Confidence

This is how well the features (or patterns) of a sample (input) image fits into a trained class' features (or pattern) mapping. It usually is expressed in percentage (0-100%) or floating numbers between 0 and 1 inclusive. This is a key feature in the model testing software for instance the model uses this property to infer its approximation of the input features as belonging to the label's class.

3.2.2 Accuracy

This is the proportion of inputs with correct label classification when passed to the model's network to classify. For instance, if given 1000 airplane images and it correctly classified 893, its accuracy will be 89.3% (***893/1000***). This is a critical decider in evaluating the training process during the evaluation of the trained model after training.

3.2.3 Hardware

The main hardware used was a GeForce GTX 1080 Ti. It was CUDA capable, with compute capability of 6.1 having 11GB memory. The computer for the (DNN) compute intensive software had 4 cores (quadcore) with 16GB RAM.

3.3 Trained Model Generic Architecture

Essentially, the trained models' architecture involves the layering of the deep net phases of convolution-pooling. Using Microsoft® Cognitive Toolkit (CNTK) library in python a CNN architecture was implemented. Referring to Figure 22 below (or see section 2.6), three hidden layers were used in the architecture having convolution and subsampling (pooling layer) – the more hidden (dense) layers the harder it is to train. After the third dense layer, the network is fully convolved. [1 – 5.]

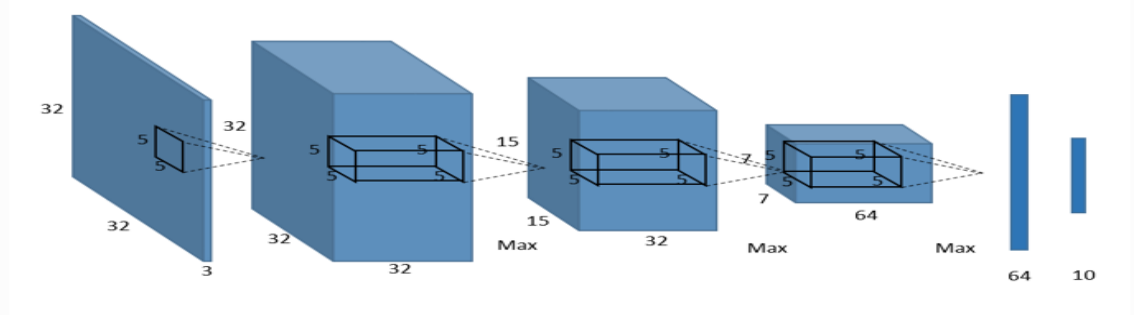


Figure 22. General architecture of a CNN as used in model training. Modified from [2].

In synopsis, the architecture has the Input (images) – Convolution – RELU – Sampling (Pooling) – Fully Connected layer (FC). This, unlike conventional neural nets scale well with even simple images with weight of 3072 (32x32x3) as used in this project's model training – data sourced from CIFAR-10. [1; 2; 3; 19.] Using this architecture, the ConvNet transforms the original input (an image), layer by layer, into class scores in the FC.

3.4 Methodology in Model Training

In this project, depending on the model being trained, dropout, batch normalization and various enhancement were done to each layer in the network to optimize the model's performance, after training, at predictions. CNTK library was used for all model trainings.

For ResNet (residual network) model training, convolution with batch normalization intrinsically modelled alongside, were used.
During training, each model was run through the network, each training varying the number of epochs (the round of getting all training samples to go through the network). Depending on the convergence of the network or otherwise, the epochs were either increased or reduced. The observations decided what was appropriate, and sometimes the learning rate had to be modified from the previous value to obtain a better result, much as there was always, as with all ML processes, a limit to how much modification could be made and the effect in each case – model overfitting or underfitting.

Furthermore, the subsampling process mostly used max-pooling – explained in ***Pooling Layers*** on page 23, to obtain the regions that had the most significant pixel per filter.

3.4.1 Convolution

The convolution is where the network is broken into pixels (or bits) and each is subsampled. It applies to all CNN model training irrespective of library used. It consists of the following:

- Filter shape – the nature of the filter (sub-matrix) e.g. (3,3) to be used in the choosing kernel;
- Number of filters – the units of such filters to be so used in the subnetwork;
- Activation function – This is the function used in activating the network; ReLU or None in DL;
- Initialization (init) – How the sub-network is to be initialised; this is often a random initialization in DL;
- Padding – Are we adding padding or not; this is a Boolean parameter: True or False;
- Stride – the matrix for iterating the kernel e.g. (1,1).

Each of the architectures function has varying number of parameters which could be seen as variadic in implementation. Each neuron in this network is connected to every other neuron in the network – a deep net architecture.

Non-linear layer or ReLUs in the network's implementation is part of the convolution layer. Referring to ReLUs in chapter 2, it employs an activation function or basically removes linearity from the image's data and introduces non-linearity.

3.4.2 Sub-Sampling

In this layer Max-pooling or average pooling (sometimes used only in the last network layer before FC) is used. It has parameters – filter shape, strides, and padding. Each parameter is as explained for convolution.

3.4.3 Dropout and Batch Normalization

These are performance optimization functions used between the convolution and sub-sampling layers of the network. Dropout takes a probability of discard rate of pixels along the network. It does help solve problems that may arise from overfitting. It is a regularization technique as explained in chapter 2. It drops units both hidden and visible in the ConvNet.

On the other hand, Batch normalization helps solve covariant-shift issues with deep learning. This occurs when the units used as predictors, also called the covariant, are altered between the training and production phases. Besides, it could arise when there is a dataset shift –when the joint distribution of the given input and their output data are altered – intentionally or otherwise. [1.]

The inputs used in the model training were the dataset (images) from CIFAR-10 [1]. Each image is a Height * Width * Colour (channels) – 32x32x3 – image with weight of 3072. The number of filters were mostly 64, and the hidden layers varied from 3 to 5.
CNTK library in Python was used in the training. For inputs to the CONV/FC, these layers enhance the network by each performing a transformation from two (2) perspectives – activation of input volume, as well as the parameters including the neuron's biases and weights.

3.5 Real-time Obstacle Detector (AVOD) Software Design

This is the autonomous vehicle obstacle detector (AVOD) software. Using open CV deep learning library (DNN) [6] and Caffe® model and weight downloaded from Caffe webpage [4] [5], with Python as the technology, a real-time software was developed. The application can, at its minimal, take input images and do a prediction on what they could be, per 21 trained classes, and draw a bounding-box around each class of image it identifies in the given input. Performing in real-time, it takes a live-stream video frame and analyses the content (images) in each frame of the stream. It then infers what it is, with label and confidence, and displays same, in real-time, to the user and in its prototype, sends just

the label and confidence to the AV's central intelligence as depicted in section 3.6 on architecture for inference or decision making.

3.6 Remotely Controlled Autonomous Vehicle Architecture

The architecture that supports the design and implementation is demonstrated in Figure 23 below. With this architecture, using received labels and confidence the AV's central intelligence system makes informed decisions per controls and safety.

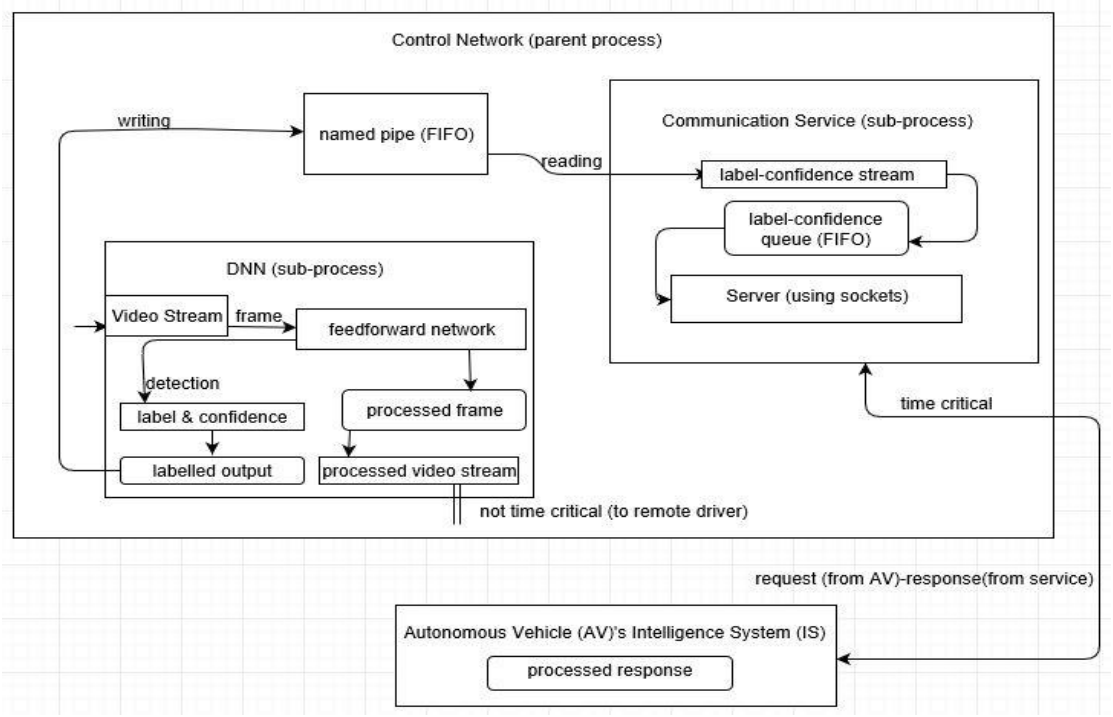


Figure 23. Autonomous Vehicle communication with neural network's service in prototype.

The architecture above has 2 sub-processes – DNN & Communication Service. The architecture demonstrated in Figure 23 has the AVOD software prototype as a component of the DNN and holistically the architecture is a component of the AV's intrinsic Intelligence (IS). The AV's IS continually queries the Communication Service in real-time with minimal or near-negligible latency. The DNN sub-process' processed frames could then be restreamed to the remote driver, if needed.

3.7 Mobile App (ODM) Design – Android® Platform

The mobile app mimics the AVOD cross-platform real-time software that was developed during this project. It is handy but with a smaller display for visualization. It was developed using Android Studio ® 3 with target APIs 23 & 26. It could capture a photo or read one from the user's gallery or take live video stream and analyse its content. Subsequently, it displayed the analysed frame and drew bounding boxes around objects detected, stating its prediction by a label and a confidence – expressed as a percentage.

4 Intrinsic Intelligence, Developed Software Use & Results

In this chapter, we will look at the developed software, what the AV's intrinsic intelligence means and its importance and how it utilises obstacle detection in its performance. Also, model training results, mobile and desktop software result from use are presented. Besides, the progress during training of each of the five models and result after training is illustrated in this chapter. The model used for the mobile and desktop software was Caffe® [. Furthermore, we literally see the software – all four (4) of them – used (with results in figures) and how well models trained in this project performed.

4.1 Developed Software

- Model Trainer – MT App
- Trained Model Obstacle Recognizer – TMOR App
- Autonomous Vehicle Obstacle Detector – AVOD App
- Object Detector Mobile – ODM App

Model Trainer App. This software was developed using CNTK library and enables the training of six (6) models – using the default library (basic), only sequential no batch normalization nor dropout (terse), dropout, batch normalized, ResNet and VGG models. This software enables altering training parameters and performance on the fly – at run time.

Trained Model Obstacle Recognizer (TMOR) App. This was wholly developed during this project and used the CNTK library. With the trained model and this software, any image in the 10 classes highlighted in section 1.2 can be recognized by the TMOR App.

Autonomous Vehicle Obstacle Detector (AVOD) App. This had 2 detection applications for video file and real-time live video stream analysis. This was entirely developed in this project using OpenCV 3.4.0 DNN library and Caffe model [6]. Using this software prototype the test AV can be conscientious, environmentally aware and its safety enhanced.

Object Detector Mobile (ODM) App. This was an application used for analysing captured images (photos) or used for detection of object content of video frames per class-scores (classes trained). This was not a real-time application in that it had considerably low frame processing rate. It was developed in this project.

4.2 Remotely Controlled AV's Intrinsic Intelligence

The used AV was designed to be remotely driven in this project. However, while not reporting on other security issues that could arise with such approach of driving a vehicle,

the concern of having a driver deliberately mishandling the vehicle and putting the lives of its passengers and other road users at risk was considered. In such a scenario, the intrinsic intelligence included the vehicle understanding its environment, and if needed, take control from the remote driver and manage its own movement. This could be stopping by itself, changing direction against the remote driver's ill-fated intention, or any other decision determined by the engineers as appropriate.

The above can be achieved by having the vehicle prioritize its awareness of its environment over the remote driver's control commands. If the AVOD software significantly predicted, for instance, it was approaching a stationery object or one it is moving considerably faster than, the vehicle intelligently decides not to carry out such commands from a remote driver, it ignored it and took corrective measure as explained above to avert any imminent danger.

4.3 Models Trained in This Project

Five models were trained during the project. When used, they could classify, or detect a class of an image it had been pretrained to detect it. For instance, the models trained could detect a horse's photograph and classify same according. This implies the classification is now faster and no training in this instance is require prior to classification as it had already been done. Below are the models and their performance output in diagrams. The model had their training implementation different from those provided on CNTK training [2] as the provided sample had training and prediction accuracy hovering around 20% at best. Worse yet, the desired epochs, more than the 5 or 10 provided made no difference and the training clearly and visually did not apply changes to the software, as maximum epoch was either 5 or 10 as used in the provided training application's hard-coded learning parameter. Predictions were far from desired. The workaround was sought – modified the software, significantly in its training application, developed a module and added some routines for saving trained model as those were not present on CNTK [2].

As in all results to follow, the 'Final Result' (validation result over 10,000 images) shows the test result and the last epoch gives a hint on the training performance (metric – the lower the better), and loss. There was no cross validation but 10,0000 of the 60,000 images were randomly selected for the test samples. The plot does show the training (Loss) and the Prediction Accuracy ('Final Result') and how well they compare. Also, the prediction, in a single instance using an image only, though not indicative of performance

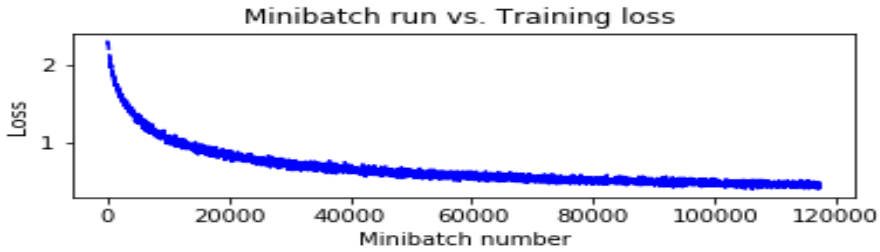
benchmark, it can, to some extent, suggest how the training went since the single instance used in top three prediction after each training as a mini-test was got from the training sample.

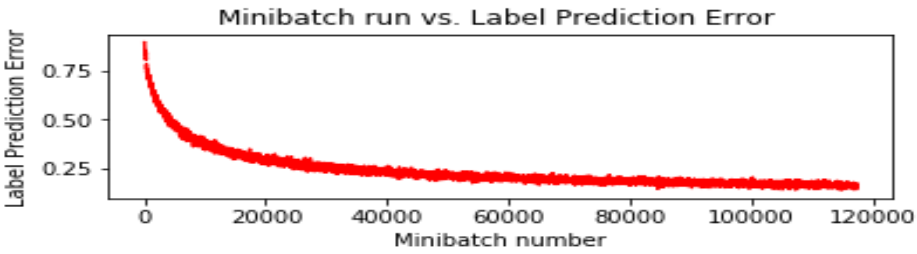
4.3.1 Terse Model

This is the default or minimalistic implementation of a CNN using the provided library. It has neither batch normalization, nor dropout. This has same implementation as the basic model save this used a more concise implementation called in ML sequential, and the basic model was trained over significantly larger epochs compared to the terse implementation. Terse, more precisely sequential implementation, encourages true parallelism and the computation can be faster as demonstrated in Figure 24 below.

```
Finished Epoch[149 of 150]: [Training] loss = 0.461662 * 50000, metric = 15.99%
Finished Epoch[150 of 150]: [Training] loss = 0.452852 * 50000, metric = 15.62%

Final Results: Minibatch[1-626]: errs = 17.9% * 10000
```






```
Top 3 predictions:
        Label: truck     , confidence: 99.63%
        Label: ship      , confidence: 0.33%
        Label: automobile, confidence: 0.02%
```

Figure 24. A Terse implementation of basic model.

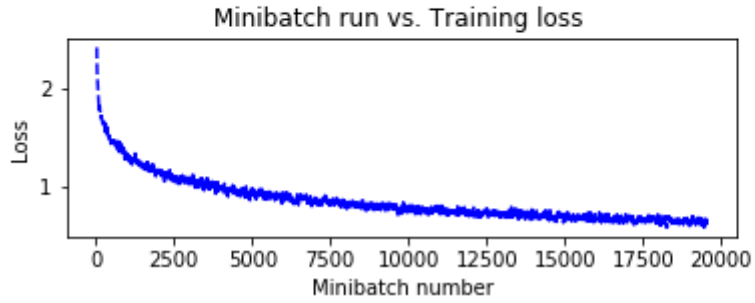
The prediction does compare to the loss function; though it did not come to levels comparable to the basic model. However, it is understandable given the basic model was trained in 1000 epochs – that is a much work. In addition, it does compare well, with well over 83% accuracy in the ten thousand (10,000) images tested.

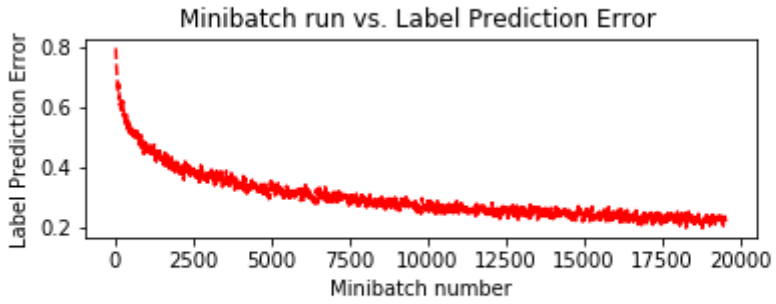
4.3.2 Batch Normalised Model

This approach is an optimal approach to having a deep learning algorithm converge during training in good time. It essentially helps models with saturating nonlinearities problem. [49,1-6.] Figure 25 below shows a model that is almost converging after 25 epochs, though the noise is yet pronounced as the loss function does try to somewhat fit (not necessarily overfit) the data. Figure 26 depicts a fully converged network with the loss almost steadying. The additional epochs, 20 more, in the 200 epochs do little to reduce the loss to values that produced a percent reduction in the test accuracy and bring it to 83.6% prediction accuracy.

```
Finished Epoch[23 of 25]: [Training] loss = 0.657829 * 50000, metric = 22.84%
Finished Epoch[24 of 25]: [Training] loss = 0.643797 * 50000, metric = 22.41%
Finished Epoch[25 of 25]: [Training] loss = 0.643571 * 50000, metric = 22.33%

Final Results: Minibatch[1-626]: errs = 21.6% * 10000
```





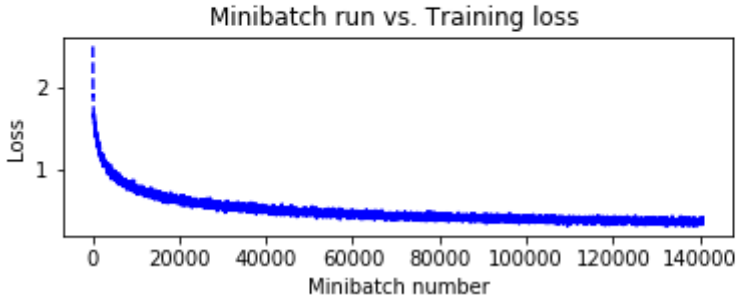

```
Top 3 predictions:
        Label: truck     , confidence: 99.71%
        Label: automobile, confidence: 0.25%
        Label: ship      , confidence: 0.02%
```

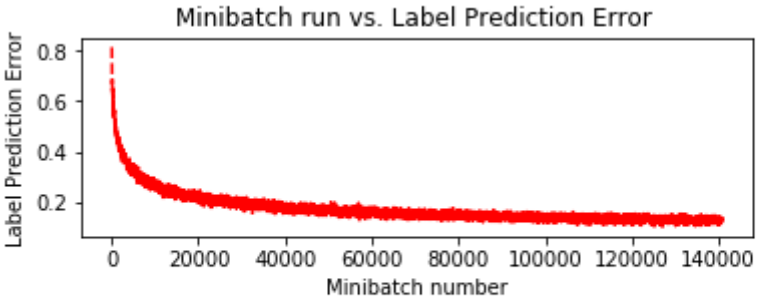
Figure 25. Batch normalized model.

Model in Figure 25 is almost converged after 25 epochs. However, the loss is still far from the desired results. Let us see the impact of further training in Figure 26.

```
Finished Epoch[179 of 180]: [Training] loss = 0.361219 * 50000, metric = 12.64%
Finished Epoch[180 of 180]: [Training] loss = 0.366098 * 50000, metric = 12.61%

Final Results: Minibatch[1-626]: errs = 17.4% * 10000
```





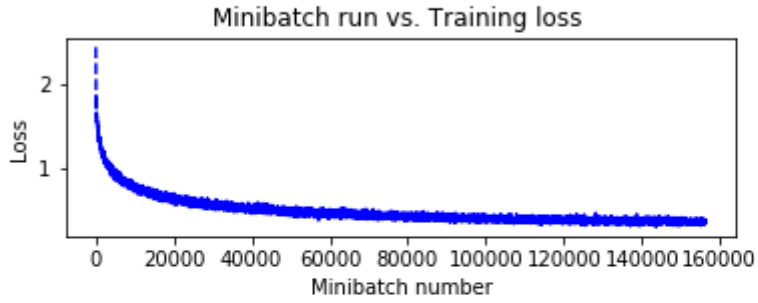

```
Top 3 predictions:
        Label: truck     , confidence: 99.27%
        Label: ship      , confidence: 0.64%
        Label: automobile, confidence: 0.05%
```

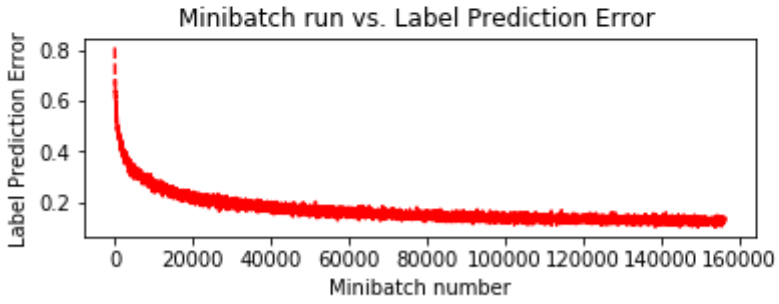
Figure 26. Normalized model with more epochs.

This model from the training represented in Figure 26 does significantly better than the model with 25 epochs. Also, the loss is significantly reduced, and the accuracy is further enhanced by just over 4 percent from the 25-epochs-trained-model before it. To see how epochs and training impact on test and accuracy, let us see the effect of further training on the normalized model in Figure 27.

```
Finished Epoch[199 of 200]: [Training] loss = 0.361857 * 50000, metric = 12.75%
Finished Epoch[200 of 200]: [Training] loss = 0.359550 * 50000, metric = 12.66%

Final Results: Minibatch[1-626]: errs = 16.4% * 10000
```






```
Top 3 predictions:
        Label: truck     , confidence: 99.45%
        Label: ship      , confidence: 0.21%
        Label: cat       , confidence: 0.17%
```

Figure 27. Normalized model with 200 epochs.

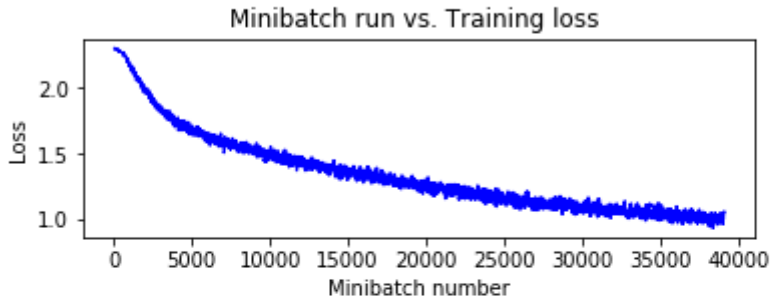
The model obtained after the training depicted in Figure 27 showed significant convergence. It is very much comparable to the 180 epochs model of Figure 26 though it marginally improves on the loss and struggled to improve the test result but with a tangible 1 percent gain on prediction accuracy.

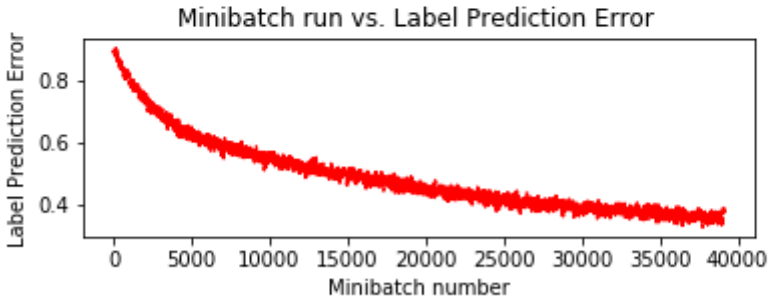
4.3.3 Dropout Model

This was another promising model and optimised the network by discard an input in layer using a probability function. In the training, the probability was varied using values 0.4; 0.2; 0.25; and 0.3. The model depicted used 25 percentage (0.25) in its training. It had predictions comparable to its loss function's output. Also, there is no overfitting – a good performance benchmark. Figure 28 shows a model trained with dropout optimization.

```
Finished Epoch[49 of 50]: [Training] loss = 1.015125 * 50000, metric = 35.39%
Finished Epoch[50 of 50]: [Training] loss = 1.011064 * 50000, metric = 35.56%

Final Results: Minibatch[1-626]: errs = 31.2% * 10000
```





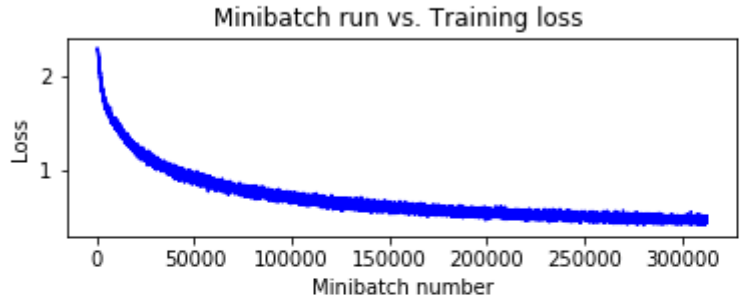

```
Top 3 predictions:
        Label: truck     , confidence: 97.44%
        Label: automobile, confidence: 1.97%
        Label: ship      , confidence: 0.25%
```

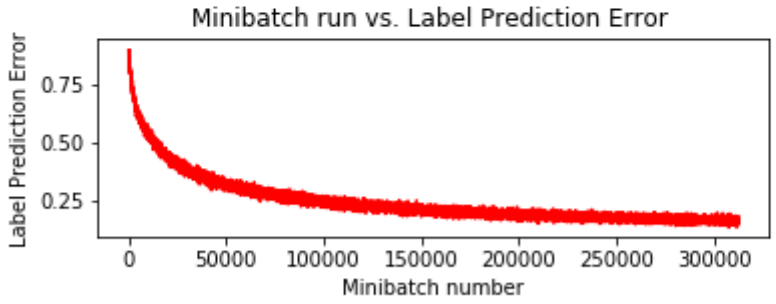
Figure 28. Dropout model.

The model in Figure 28 above had a good 'loss' value, after training, as it was as high as 2.1 for 50,000 images trained at training commencement. However, it did not appear to have steeply fallen and slight fluctuations showed in the output though a steep fall in error with minibatch numbers. Also, the individual (a single image inferred here) test correctly predicted Truck as Figure 28 does show. It did suggest more epochs was likely required. Figure 29 shows a model trained with dropout having more epochs than Figure 28.

```
Finished Epoch[399 of 400]: [Training] loss = 0.469115 * 50000, metric = 16.15%
Finished Epoch[400 of 400]: [Training] loss = 0.468135 * 50000, metric = 16.11%

Final Results: Minibatch[1-626]: errs = 17.7% * 10000
```






```
Top 3 predictions:
        Label: truck     , confidence: 89.95%
        Label: ship      , confidence: 6.57%
        Label: cat       , confidence: 1.63%
```

Figure 29. Dropout with more epochs.

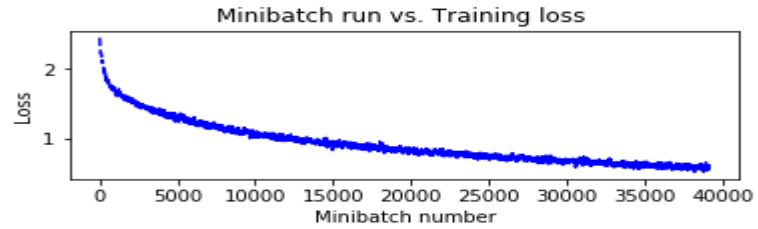
Model from training shown in Figure 29 almost halved the test error from the previous 50-epochs-trained-model in Figure 28. Also, the convergence is more evident here. Besides, the loss and predictions are comparable in the figure above. Surprisingly, much as it did far better in the training error for 10000 images, the individual test with a truck's image did not outperform or even equal with the 50-epoch's model, as it lags by nearly 8 percent (8%). This is a case in point where an error could have occurred. Hence, true performance or accuracy can be estimated with more test or test data passed through the model's feedforward network.

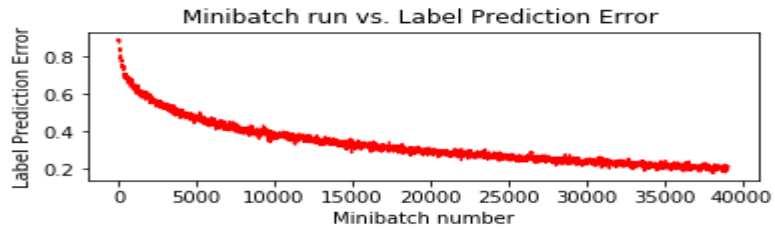
4.3.4 Residual Network (ResNet) Model

This is a complex model and as explained in section 3.4.3, its implementation did include batch normalization. Having gone through 50 epochs as shown in Figure 30, it converged save the loss level was still considerably high per expectation of having loss less than 0.45 – the standard set by this project – not a requirement.

```
Finished Epoch[49 of 50]: [Training] loss = 0.592660 * 50000, metric = 20.70%
Finished Epoch[50 of 50]: [Training] loss = 0.589663 * 50000, metric = 20.64%

Final Results: Minibatch[1-626]: errs = 23.4% * 10000
```





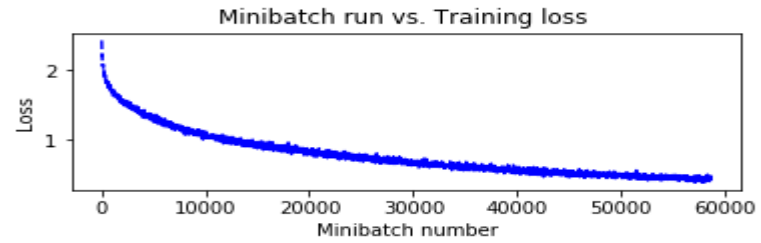

```
Top 3 predictions:
        Label: truck     , confidence: 96.43%
        Label: ship      , confidence: 3.18%
        Label: automobile, confidence: 0.17%
```

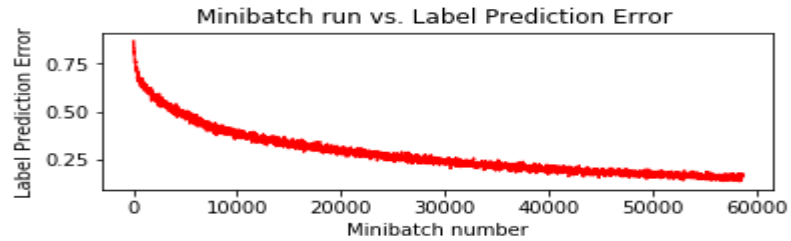
Figure 30.ResNet model with 50 epochs.

The error rate of model in figure above was above the benchmark, though it is converged. Let us see the effect, if any, of further epochs on the model trained in Figure 31.

```
Finished Epoch[74 of 75]: [Training] loss = 0.445538 * 50000, metric = 15.54%
Finished Epoch[75 of 75]: [Training] loss = 0.444391 * 50000, metric = 15.48%

Final Results: Minibatch[1-626]: errs = 20.1% * 10000
```






```
Top 3 predictions:
        Label: truck     , confidence: 64.87%
        Label: ship      , confidence: 35.06%
        Label: automobile, confidence: 0.05%
```

Figure 31. ResNet model.

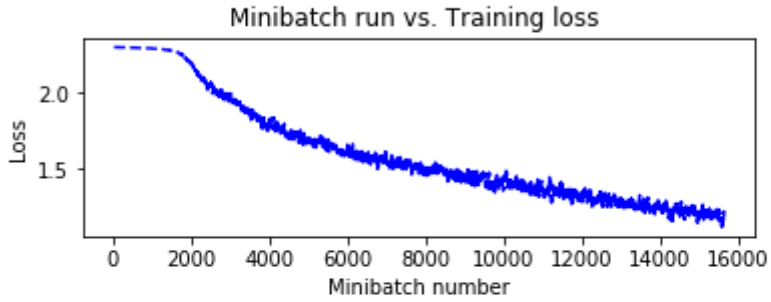
After the training shown in Figure 31, the model diagrammatically did not appear to show much difference. The bulk test error still stayed above the metric, though it fell by 2.1 percent (2.1%). Notwithstanding, the loss significantly reduced from 0.59 per 50,000 to 0.44 per 50,000 – a noticeable improvement.

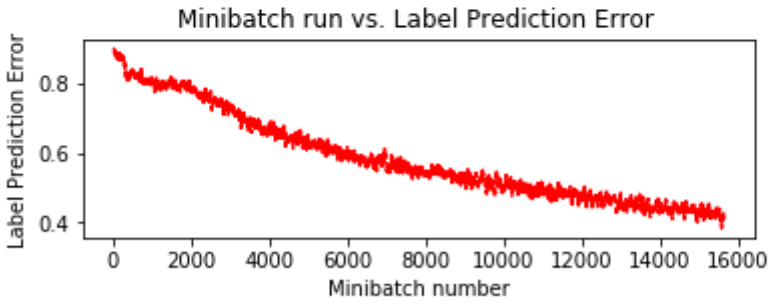
4.3.5 VGG Model

The VGG model was run in two epoch phases. First, the model had 20 epochs as shown in Figure 32 below. During this phase the loss was considerably high, with values well over 1.1 per 50,000 training images.

```
Finished Epoch[19 of 20]: [Training] loss = 1.227158 * 50000, metric = 43.97%
Finished Epoch[20 of 20]: [Training] loss = 1.199090 * 50000, metric = 42.64%

Final Results: Minibatch[1-626]: errs = 39.0% * 10000
```





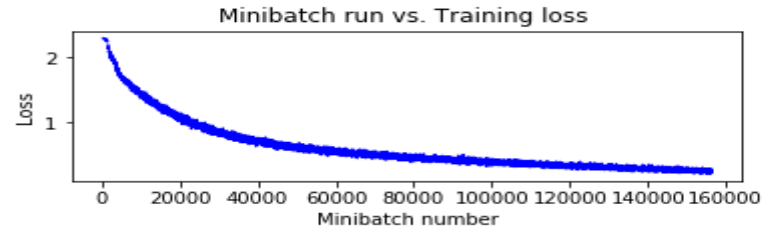

```
Top 3 predictions:
        Label: truck     , confidence: 89.87%
        Label: automobile, confidence: 3.74%
        Label: ship      , confidence: 2.71%
```

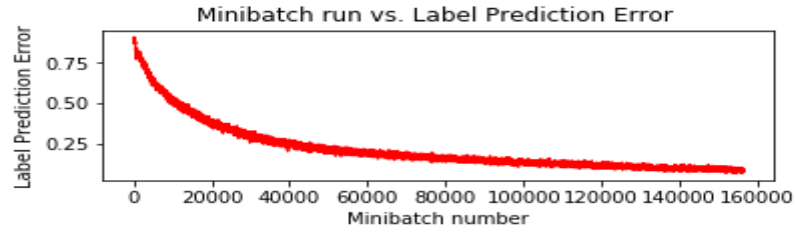
Figure 32. VGG model.

The model in the preceding figure did not perform optimally in predictions. Also, it was seen the predictions are not exactly coherent with the loss function and the convergence though imminent did not clearly indicate it had been reached. Again, the individual test, though not a benchmark as to a good performance when used in isolation, did not give an accuracy up to 99%. Having these observations, the model was further trained as Figure 33 below indicates.

```
Finished Epoch[199 of 200]: [Training] loss = 0.256062 * 50000, metric = 9.00%
Finished Epoch[200 of 200]: [Training] loss = 0.251540 * 50000, metric = 8.80%

Final Results: Minibatch[1-626]: errs = 14.4% * 10000
```






```
Top 3 predictions:
        Label: truck      , confidence: 98.74%
        Label: automobile, confidence: 1.17%
        Label: ship       , confidence: 0.09%
```

Figure 33. Well trained VGG model.

The model with 200 epochs shown in Figure 33 reached convergence. Its predictions notably mimicked the loss function and it reduced the previous error by sixty-three percent (63%).

4.4 TMOR App Using the VGG Model

The VGG model trained was used by the TMOR App in recognizing some images downloaded from the web – see Appendices 1-4. results are depicted in Figure 34 below.

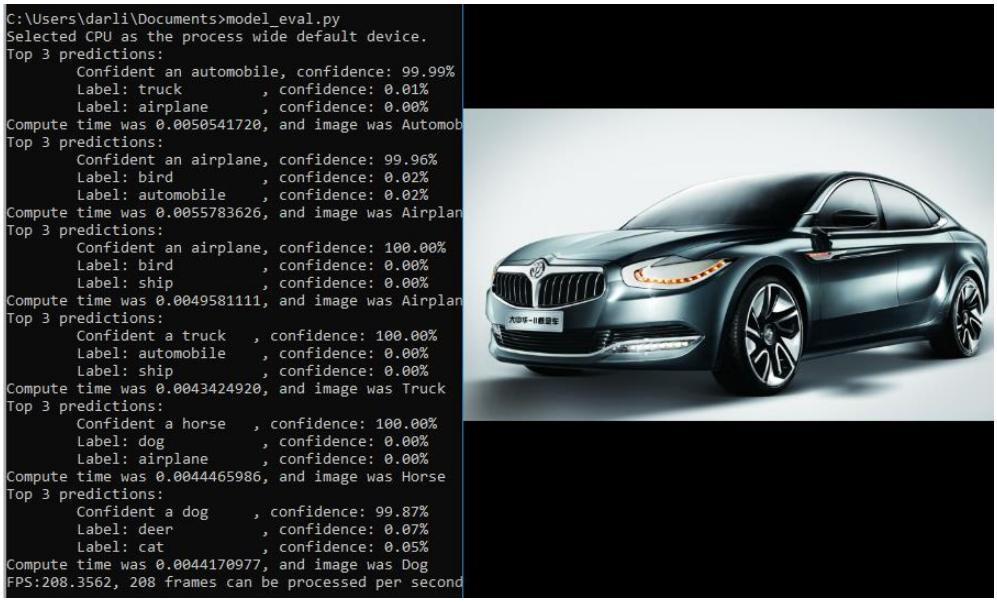


Figure 34. Testing a model with a car image.

The reason for choosing VGG as a representative model here was not far fetch – was the best performing model during training and in validation (testing). Part of the image to the right was the first test image passed through the model's network to classify. The value to be expected is the last displayed value in each test result and notably the FPS displayed. It did classify it with a triple nine (99.9% and above) accuracy. This result will be explained further in section 5.1 on **Trained Models**. The other part of Figure 34 shows only the computation results.

The VGG model above did show impressive confidence with test samples from dataset it had not seen previously. Some are with confidence above 95%.

4.5 Object Detector Mobile (ODM) App

The android application developed could analyse an image and predict the content of a video (its frames) in real time. When given an image, the application runs it through its network – a pretrained network using OpenCV DNN Caffe network, and it predicts one of the pretrained classes – the application gives it a 21-class confines to choose from with a given threshold of acceptable confidence values. Some results depicted with screenshots are shown in **Figure 35** and Figure 36 below (see Appendix 5).

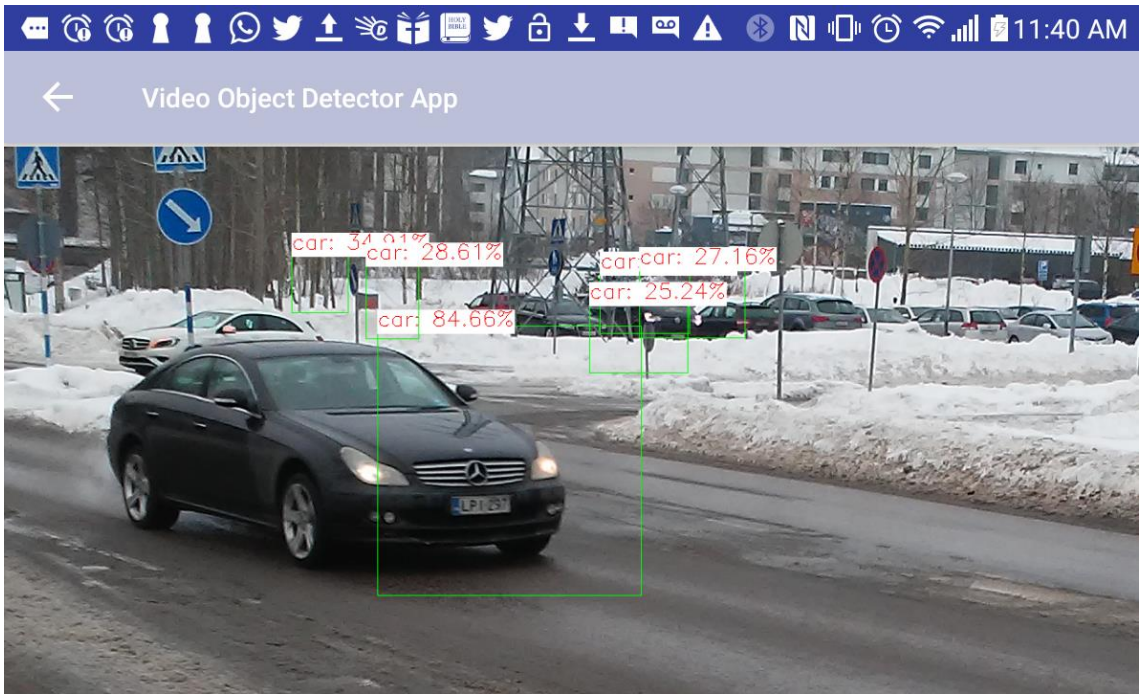


Figure 35. Android App object detection of a moving car.

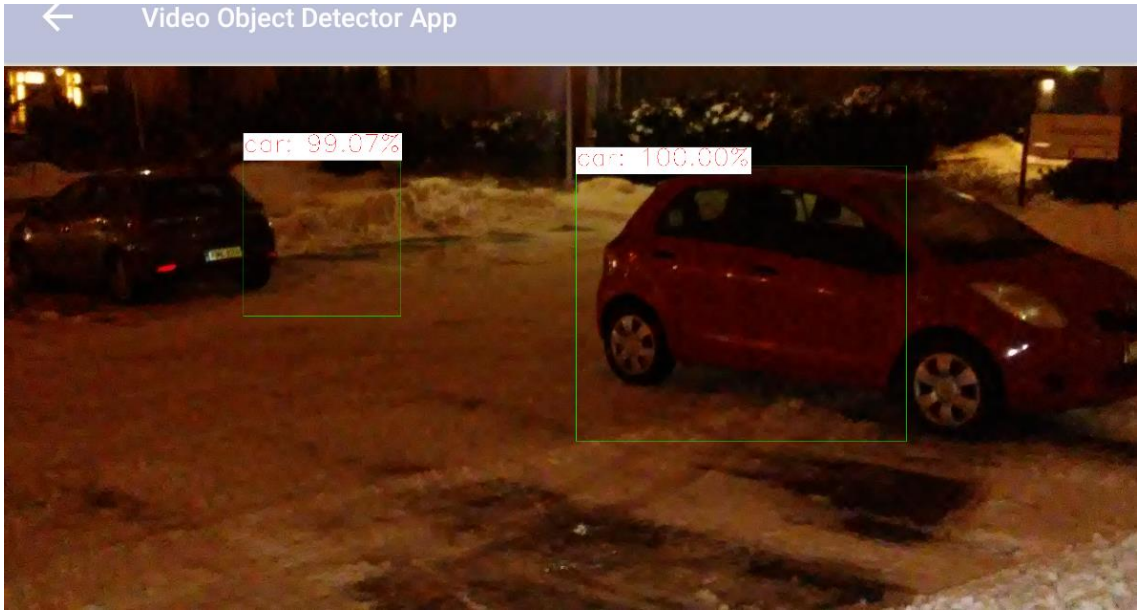


Figure 36. Android app detecting vehicles at night with label and confidence.

Like what is shown on Figure 35, the images (pictures) and video frames (still images in videos) were analysed in real-time, but considerably slow, and the results displayed on a mobile device as shown in Figure 36 showing detailed label and confidence.

4.6 Autonomous Vehicle Object Detector (AVOD) App

The AVOD application used real-time video stream frames. It takes a frame as shown in **Figure 37** below, analyses it and gives detections and confidence of objects detected.

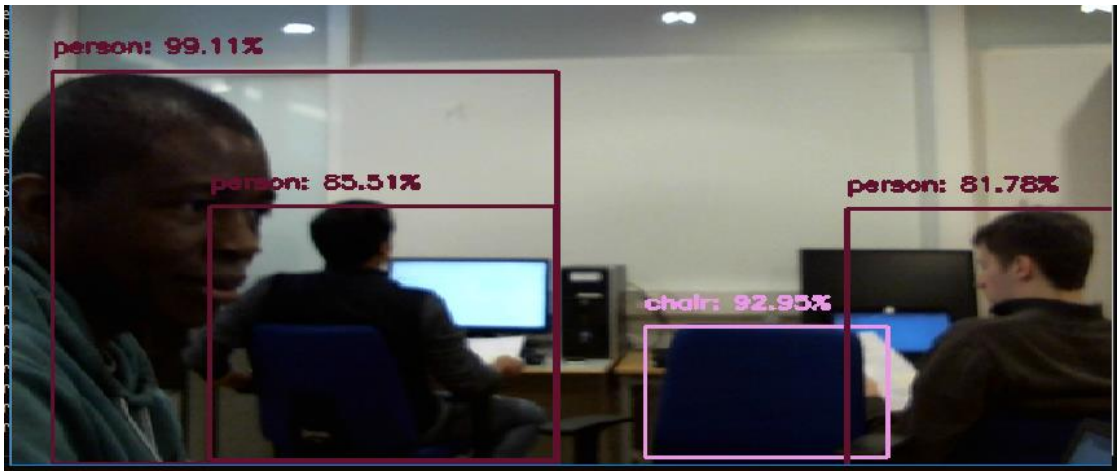


Figure 37.Desktop app – object detection.

The AVOD app on a laptop predicted person and chair as seen in Figure 37 above with confidence of 99.88% and 91.92% for person and chair respectively. That is an encouraging benchmark it may suggest. Notepad [69] was used in its development.

5 Discussion

In this chapter the observations from results in using the software are discussed. First, the observation from the MT app and the trained model object recognizer TMOR are discussed in **Trained Models**. Subsequently, other software developed are reviewed under the headings: **Object Detector Mobile (ODM) App Review**, and **Autonomous Vehicle Obstacle Detector (AVOD) App Review**. Appendices 1–6 have some additional details.

5.1 Trained Models

To discuss the model training phase Table 2 below depicts the training results. As observed in chapter Intrinsic Intelligence, Developed Software, models could predict with commendable confidence object recognised from dataset no yet seen.

Table 2. Implementation and epochs Impact on models training and prediction results.

Model	Epochs Run	Pred. Accuracy (%)	Pred. Error (%)	Training Acc. (%)	Tr. Error (%)
VGG9	200	85,60	14,40	91,18	8,82
ResNet	75	84,52	15,48	85,52	14,48
Basic	1000	84,50	15,50	89,54	10,46
Dropout	400	83,89	16,11	82,3	17,7
Batch Normalized	200	83,60	16,40	87,34	12,66
Batch Normalised	180	82,60	17,40	87,39	12,61
Terse Basic	150	82,10	17,90	74,38	25,62
ResNet	50	76,60	23,40	79,36	20,64
Dropout	50	68,80	31,20	64,44	35,56

Taking a closer look at the results from Table 2 above, it is indicative that model implementation does impact on training accuracy or prediction result. Besides, it also demonstrates more training or epoch did help the convergence of the network and training of the models. However, training indefinitely or to large number of epochs may not necessarily yield better results after a threshold is reached – a convergence point. This is further buttressed, with results from Figure 26 & Figure 27, where the increase was barely marginal – 20 more epochs yielded 1% performance (accuracy) increase.

Overall, the models trained having shown their individual performance and accuracy levels, the average prediction success rate, as shown in the next figure – Figure 38, stood at 84% accuracy per 10,000 images analysed (*Appendices 2 & 3 highlight this*).

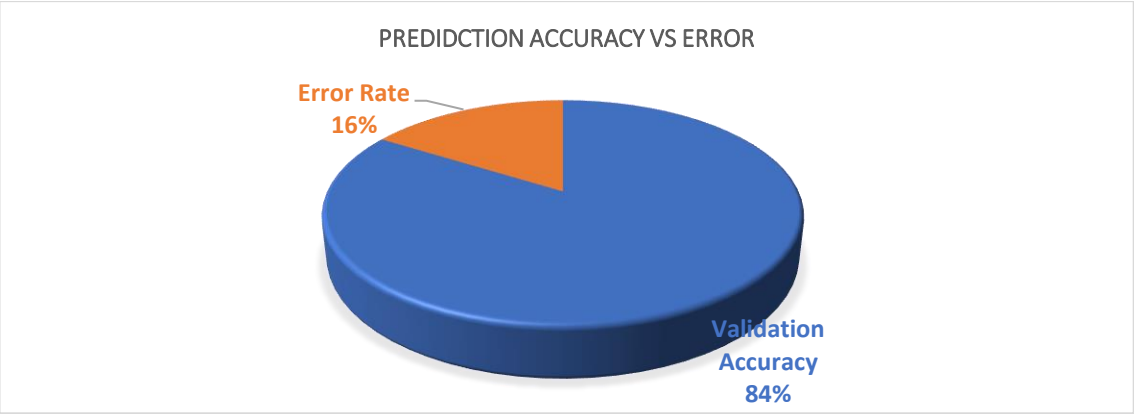


Figure 38. Models' average performance per 10,000 images tested.

On average prediction accuracy, Figure 38 shows how the models concertedly compare. A pie chart of success rate with accurate and inaccurate predictions. This implies the models, on the average predicted 84 of 100 predictions correctly, and incorrectly predicted 16 of 100 predictions. However, the manner or probability is not inferred – it could go on, in a 500 test images, with all 500 predictions and get them all right.

Figure 39, below further shows individual training, epochs and test results of the models trained.

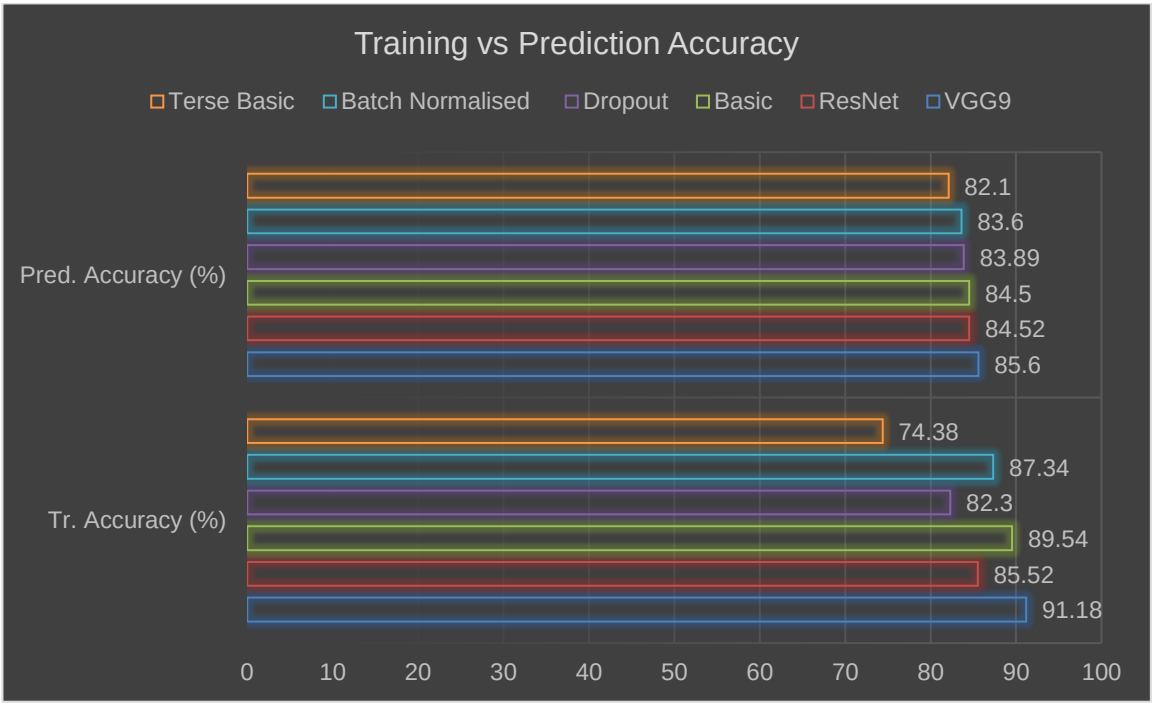


Figure 39. Effect of more epochs on performance with different models.

Model VGG had the best training accuracy and prediction which was closely followed and understandably so, by basic – for its epochs (1000). Terse Basic model was low with its training accuracy, though did better at predictions – summarised in Figure 39 above.

5.2 Object Detector Mobile (ODM) App Review

The mobile app (*Appendix 5*) did, for the video streams, prioritize fps for large confidence to give the user a real-time experience. This was a self-imposed requirement as the app on a mobile platform was increasingly stretched and the fps dropped further, giving the low processing power of a mobile device. To compensate for this, the app had to compromise on confidence, reduce the weight on the network and achieved fps of over 4 – 6.36fps. This also came at a further cost – the labels and confidence were so frequent – requiring the user to focus to see the labels as the screens are small. On images, it got the analyses in the images and displayed them as seen on Figure 35. It also could analyse capture photos and analyse photos from an android device's photo gallery.

5.3 Autonomous Vehicle Obstacle Detector (AVOD) App Review

This is a main real-time software, together with the TMOR app. This app could detect and categorize objects in real-time. However, the frames per second (fps) was initially low – 1.87fps. The engineering around this – assigning a dedicated thread to handle the frame processing improved it to 4.68 fps from 1.87fps. Further improvement using multi-threading (12 worker threads for network and a thread for screen) had proved not significantly helpful – produced 12.87fps and not fluid – until how the write-to-queue, using locks, functions was critically observed and understood. With this understanding, and further modification in the software's applications' program structure, the software could process all frames – 100.00% of frames it got from the camera stream which is per manufacturer specification 30fps – 29.88fps (see *Appendix 6*). The application's frame processing rate using a Logitech® c525 webcam to whole number was 30fps–the camera's maximum frame rate. [1–8; 54.] This is good and as stated in the conclusion this software can handle 4K video streams at 60fps. Also, with modified video or reprogrammed video with framerate up to 200fps, this application can process frames at over 97.5% – time between frames will be so small to impact somewhat, of the video's framerate though it may require slightly increasing the number of worker threads when framerate go beyond 100fps (*see Appendix 6 for further display*).

5.4 Autonomous Machine Ethics

Autonomously controlled machines including robots and vehicles may present some ethical questions as alluded to by Mark [70], Bryson [71], Gunkel [72,125] and Moor [73]. These questions do arise sequel to our placing autonomous machines among humans in our social space, reiterates Sandry [74]. Whether machine ethics apply or not depends on from what perspective one looks at it. M. Anderson & S. Anderson [75] argue that the

existence of a set of principles defined for a machine and the machine's ability to follow same should be the overriding goal of machine ethics. Whether these rules or instructions are defined (*explicitly programmed*), or the machine given room to make informed calculations from which to draw inference, yet, mishaps do happen, even so with non-humans – machines.

Ethical challenges emanating from autonomous machines including robots and vehicles centre chiefly on morals, even as Sandler [76,267-393] further buttresses. He adds that this could span from usage in warfare to government surveillance. However, who defines the morals or sets of principles and for who? Also, who defines ethics, looking at it from Moor's [73] perspective, who sees it as an emotional expression and argues that machines cannot have emotion, though Sandry [74] challenges that claim? Bryson [71] reiterates morality is hinged on freedom, without it there will be no action, which is the product of freedom. This new field of research as suggested by M. Anderson & S. Anderson [77] should essentially be geared towards making machines more ethically responsible towards those who will use it directly or indirectly and for whosoever it will encounter accidentally or otherwise. While calling for more responsibility, care and being meticulous, I strongly hold machine ethics should not be so overwhelming sway that it douses our willingness and ability to explore new frontiers or prevents advancing our course of discovery in every area of human endeavour, more so hampering our giving an autonomous machine a space in our sphere.

5.5 Call for Further Research

During this project, a tangible lesson learnt was not to be too concerned about making mistakes when you are learning or experimenting, and when mistakes occur find out why they did and attempt proffering a solution. Also, some task can be very demanding to the point of almost giving up. Only determination and self-motivation will keep one devoted to a task in reaching set goals. All of these have been demonstrated in this project. It is no gainsaying this project was challenging, complex and resource consuming, much as it met desired goals even surpassing some. However, further research in machine learning, and particularly deep learning application for those who are strongly self-motivated, challenge-seeking and of an innovative mindset is sought and encouraged.

6 Conclusion

This project was carried out to infer prediction accuracy and framerate from a self-trained deep learning image recognition model, and object detection usefulness in remotely controllable self-driving cars using a library. The AVOD software, was used to analyse frames from a vehicle attachable webcam. The VGG model could classify the images with prediction accuracy up to 85.6% per 10,000 images tested. VGG model, using TMOR, had confidence levels up to 100.0% when tested with images. However, the AVOD and ODM software's model's (Caffe model) prediction accuracy could not be inferred given the model was not trained during this project, though its confidence, *not prediction accuracy*, was often more than 90% with near images.

Three of the software developed were used in object detection (ODM), live video stream obstacle detection (AVOD), and obstacle recognition (TMOR). The fourth software developed (MT) was used in training models including VGG model which using TMOR could recognize objects in its trained classes in real-time expending nearly 6milliseconds on each frame and recognised far higher than 100 frames per second (100fps) – sufficient for a 4K video (60fps) camera. The project had targeted increasing AVOD's fps from 12.87fps to 23.88fps – high definition (HD) movies' fps. It did surpass that – processing at 100.0% of camera's framerate (30fps) [8] using 12 worker threads, a 13th thread for display and the main thread.

Notwithstanding, the models did give wrong predictions infrequently in the AVOD, ODM, and TMOR software. A notable observation was that, a trained model (VGG), and significantly Caffe model, when given an image not parallel to the camera, considerably tilted at an angle, say greater than 12°**,** though not measured, gave awkward classification sometimes with surprisingly high confidence. The MT's & TMOR's accuracy benchmark was set at 90% – prediction and training with 10,000 and 50,000 images respectively, but best model lagged by 4.4% (Figure 39) at predictions though in training recorded 91.2%.

The project was intellectually demanding, needing research, determination, commitment, and high computation power of a GPU. More research by students in DL object recognition, detection and model training is encouraged, and the scholar should not underplay the importance of a CUDA capable GPU. Projects as these can pose a challenge to get started – devotion and resolve will help actualise it. Having a supervisor or instructor who has worked on DL will be invaluable though not a requirement.

References

1. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. Ithaca, New York: Cornell University Library Feb 2015; 1-11.

2. CNTK. Recognize objects in images from CIFAR-10 data (convolutional network, CNN) [online]. Redmond, Washington: Microsoft®; October 2017.
   URL: https://cntk.ai/pythondocs/tutorials.html. Accessed; 12 December 2017.

3. Andrej Karpathy and Justin Johnson. CS231n Convolutional neural networks for visual recognition [online].
   URL: http://cs231n.github.io/convolutional-networks/. Accessed 9 January 2018.

4. Anaconda, Inc. Jupyter-notebook application*. Austin, Texas. December 2017.

5. Anaconda, Inc. IPython QtConsole application*. Austin, Texas. January 2018.

6. Deep Neural Networks Library (DNN). OpenCV 3.4.0 [online]. OpenCV team; 23 December 2017.
   URL https://opencv.org/releases.html. Accessed 5 January 2018.

7. CUDA. CUDA Zone [online]. Sanata Clara, California: Nvidia Inc.; 25 January 2018.
   URL: https://developer.nvidia.com/cuda-zone. Accessed 27th January 2018.

8. Logitech. Logitech c525 camera specification [online]. Newark, California: Logitech; January 2018.
   URL: http://support.logitech.com/en_us/product/hd-webcam-c525/specs. Accessed 21 January 2018.

9. Jason Bell. Machine learning – hands-on for developers and technical professionals. Indianapolis, IN: Wiley; 2014.

10. Haykin S. Neural networks – a comprehensive foundation. 2nd ed. Upper Saddle River, NJ: Prentice Hall; 1999.

11. Abdallah Meraoumia, Maarouf Korichi, Salim Chitroub, and Ahmed Bouridane. Hidden Markov models & principal component analysis for multispectral palmprint identification. 2015 5th International Conference on Information & Communication Technology and Accessibility (ICTA), Marrakech, 2015, pp. 1-6.

12. Wei Wang, Gang Chen, Haibo Chen, Tien Tuan Anh Dinh, Jinyang Gao, Beng Chin Ooi, Kian-Lee Tan, ShengWang, and Meihui Zhang. Deep learning at scale and at ease. ACM Transactions on Multimedia Computing and Communication Applications 2016;12(4,69):1-25.

13. Intel© Movidius®+ [online]. Santa Clara, California: Intel©; 7 November 2017.
   URL: https://www.movidius.com/.
   Accessed 8 January 2018.

14. Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Energy-efficient and improved image recognition with conditional deep learning. J. Emerging Technology and Computing Systems 2017; 13(3,33):1-21.

15. Kaoru Ota, Minh Son Dao, Vasileios Mezaris, and Francesco G. B. De Natale. Deep learning for mobile multimedia: A survey ACM Transactions on Multimedia Computing and Communication Applications 2017; 13(3,34):1-22.

16. Dumitru Erhan, Christian Szegedy, Alexander Toshev, Dragomir Anguelov. Scalable object detection using deep neural networks. CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition June 2014;1(1): 2155-2162.

17. Amirfarhad Nilizadeh, Wojciech Mazurzyk, Cliff Zou, and Leavens GT. Information hiding in RGB images using an improved matrix pattern approach. IEEE Conference on Computer Vision and Pattern Recognition Workshops 2017;183(1):1407-1416.

18. Zheng-Jun Zha, Yang Yang, Jinhui Tang, Meng Wang, and Tat-Seng Chua. Robust multiview feature learning for RGB-D image understanding. ACM Transactions on Intelligent Systems and Technology 2015;6(2,15):1-19.

19. Tom M. Mitchell. The discipline of machine learning. Pittsburgh: Pennsylvania, PA: Carnegie Mellon University; 2016.

20. Fatma S. Abousaleh, Tekoing Lim, Wen-Huang Cheng, Neng-Hao Yu, M. Anwar Hossain, and Mohammed F. Alhamid. A novel comparative deep learning framework for facial age estimation. EURASIP Journal on Image and Video Processing 2016;1(1,47):1-14.

21. Yao-Yi Chiang, Stefan Leyk, and Craig AK. A survey of digital map processing techniques. ACM Computing Surveys 2014;47(1,1):1-44.

22. Xinyu Ou, Hefei Ling, Ping Li, Fuhao Zou, and Si Liu. Adult image video recognition by a deep multicontext network and fine-to-coarse strategy. ACM Transactions on Intelligent Systems and technology 2017;8(5,68):1-25.

23. Bell S & Bala K. Learning visual similarity for product design with convolutional neural networks. ACM Transactions on Graphics;34(4,98):1-10.

24. Wang L, Wang Z, Guo S, and Qiao Y. Better exploiting OS-CNNs for Better Event Recognition in Images. IEEE International Conference on Computer Vision Workshop 2015;46(1):287-294.

25. Aykanat M, Kilic O, Bahar K, and Sevgi Saryal. Classification of lung sounds using convolutional neural networks. EURASIP Journal on Image and Video Processing 2017;1(1,65):1-9.

26. Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Energy-efficient and improved image recognition with conditional deep learning. J. Emerging Technology and Computing Systems 2017; 13(3,33):1-21.

27. Giuseppe Amato, Fabrizio Falchi, and Claudio Gennaro. 2015. Fast image classification for monument recognition. ACM Journal on Computing and Cultural Heritage. 2015;8(4,18):1-25.

28. Abdallah Meraoumia, Maarouf Korichi, Salim Chitroub, and Ahmed Bouridane. Hidden Markov models & principal component analysis for multispectral palmprint identification. 2015 5th International Conference on Information & Communication Technology and Accessibility (ICTA), Marrakech, 2015, pp. 1-6.

29. Oulasvirta, A., Nurminen, A., and Suomalainen, T. How real is real enough? Optimal reality sampling for fast recognition of mobile imagery. ACM Transaction on Applied Perception 2012; 9(4,21):1-18.

30. Tarr, M. J., Georghiades, A. S., and Jackson, C. D. 2008. Identifying faces across variations in lighting: Psychophysics and computation. ACM Transactions on Applied Perception 2008;5(2,10):1-25.

31. Marco della Cava. Five (5) big tech companies form group to foster good AI [online]. McLean, Virginia: USA Today; 29 September 2016. URL: https://www.usatoday.com/story/tech/news/2016/09/29/microsoft-google-and-others-form-ai-consortium---no-apple-tesla/91266190/. Accessed 15th January 2018.

32. Yann LeCun, Fu Jie Huang, and L´eon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition cvpr 2004; 2(4):97-104.

33. Aleix M. MartõÂnez and Avinash C. Kak. PCA versus LDA. IEEE Transactions on Pattern Analysis and Machine Intelligence February 2001; 23(2):228-233.

34. Nikhil Naikal, Allen Yang, and S. Shankar Sastry. Informative feature selection for object recognition via sparse PCA. Electrical Engineering and Computer Sciences University of California at Berkeley: Berkeley, California. April 2011;1-12.

35. Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape recognition and object recognition using shape contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence April 2002; 24(24):509-523.

36. Liefeng Bo, Xiaofeng Ren, Dieter Fox. Depth kernel descriptors for object recognition. Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on September 2011;(1):821-826.

37. Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06) 2006; (6):1-8.

38. Sabri Boughorbel, Jean-Philippe Tarel, and Franc¸ois Fleuret. Non-Mercer kernels for SVM object recognition. Research Gate 2004;1-11.

39. Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. IEEE Transactions on Pattern Analysis and Machine Intelligence, March 2007;29(3):411-426.

40. Massimiliano Pontil and Alessandro Verri. Support vector machines for 3D object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, June 1998;20(6):637-646.

41. Hao Zhang, Alexander C. Berg, Michael Maire, and Jitendra Malik. SVM-KNN: discriminative nearest neighbor classification for visual category recognition. Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference, June 2006; 2(1):2126-2136.

42. Yangqing Jia, Evan Shelhamer , Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell. Caffe: convolutional architecture for fast feature embedding. Proceedings of the 22nd ACM International Conference on Multimedia, November 2014, Orlando, Florida; 1(1):675-678.

43. DeepLearning [online]. 1 December 2015.
URL http://deeplearning.net/.
Accessed 20 February 2018.

44. Wikipedia. Deep Learning [online]. 8 March 2018.
URL: https://en.wikipedia.org/wiki/Deep_learning.
Accessed 11 March 2018.

45. Omoifo, O. D. October 2017. Deep learning object recognition. Unpublished manuscript, Helsinki Metropolia Univ. of App. Sc., Helsinki, Finland.

46. Xiaorui Ma, Jie Geng, and Hongyu Wang. Hyperspectral image classification via contextual deep learning. EURASIP Journal on Image and Video Processing 2015;1(1,20):1-12.

47. Sheng-hua Zhong, Yan Liu, and Kien A. Hua. Field effect deep networks for image recognition with incomplete data. ACM Transactions on Multimedia Computing and Applications 2016;12(4,52):1-22.

48. Yan Liu, Yang Liu, Shenghua Zhong, and Songtao Wu. Implicit visual learning: Image recognition via dissipative learning model. ACM Transactions on Intelligent Systems and Technology 2016;8(2,31):1-24.

49. Shamir, L., Macura, T., Orlov, N., Eckley, D. M., and Goldberg, I. G. 2010. Impressionism, expressionism, surrealism: Automated recognition of painters and schools of art. ACM Transactions on Applied Perception 2010;7(2,8):1-17.

50. Parisa Pouladzadeh and Shervin Shirmohammadi. Mobile multi-food recognition using deep learning ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) - Special Section on Deep Learning for Mobile Multimedia and Special Section on Best Papers from ACM MMSys/NOSSDAV 2016 2017;13(3,36):1-21.

51. Ludmila I. Kuncheva and William J. Faithfull. PCA extraction for change detection in multidimensional unlabelled data. IEEE Transactions on Neural Networks and Learning Systems 2014;25(1):69-94.

52. Hu, W., Zuo, H., Wu, O., Chen, Y., Zhang, Z., and Suter, D. 2011. Recognition of adult images, videos, and web page bags. ACM Transactions on Multimedia Computing and Communication Applications 2011;7(1,28):1-24.

53. Benjamin J. Bala and Pawan Sinha. Region-based representation for face recognition. ACM Transactions on Applied Perception 2006;3(4):354-375.

54. Hays, J., Efros, A. Scene completion using millions of photographs. ACM Transactions on graphics 2007;26(3,4):1-8.

55. Vikky Mohane and Prof. Chetan Gode. Object recognition for blind people using portable camera. IEEE 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (WCFTR'16) 2016. Pp 1-4.

56. Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J. Guibas. Joint Embeddings of Shapes and Images via CNN Image Purification. ACM Transactions on Graphic 2015;34(6,234):1-12.

57. Zechao Li, Jinhui Tang. Unsupervised feature selection via nonnegative spectral analysis and spectral redundancy control. IEEE Transactions on Image Processing 2015;24(12):5343-5355.

58. Ross Girshick. Fast -RCNN. London: Microsoft Research September 2015;1-9.

59. Christopher M. Bishop. Pattern recognition and machine learning. Cambridge, Cambridgeshire: Springer; 2006.

60. LISA lab, University of Montreal. Deep Learning Tutorials. Release 0.1; September 2015.

61. Shai Shalev-Shwartz and Shai Ben-David. Understanding machine learning: From theory to algorithms. Cambridge, Cambridgeshire: Cambridge University Press; 2014.

62. Zhiyuan Shi, Tae-Kyun Kim. Learning and refining of privileged information-based RNNs for action recognition from depth sequences. London: Imperial College 2017;1-10.

63. Sanja Fidler. CS420: Introduction to image understanding - object detection sliding windows [online]. Toronto, ON: University of Toronto; 20 August 2017. URL:

http://www.cs.utoronto.ca/~fidler/slides/CSC420/lecture17.pdf. Accessed 09 October 2017.

64. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Microsoft Research Jan. 2016;1-14.

65. Nils J. Nilsson. Introduction to machine learning – an early draft of a proposed textbook. Stanford University, Stanford; 2005.

66. Ethem Alpaydin. Introduction to machine learning. 2nd ed. Cambridge, Cambridgeshire: The MIT Press; 2010.

67. Samer Hijazi, Rishi Kumar, and Chris Rowen. Using convolutional neural network for image recognition. IP Group, Cadence. Cadence Design Systems Inc.; 2015.

68. Satya Mallick. Selective Search. Learn OpenCV [online]. San Diego, CA: Learn OpenCV; 10 November 2017.
URL: https://www.learnopencv.com/
selective-search-for-object-detection-cpp-python/. Accessed 26th October 2017.

69. Do Ho. Notepad-plus-plus*. Release 7.5.2; 28 November 2018.

70. Coeckelbergh, Mark. The Moral Standing of Machines: Towards a Relational and Non-Cartesian Moral Hermeneutics. Philosophy & Technology – Dordrecht 2014;27(1): 61-77.

71. Joanna J. Bryson. Patiency is not a virtue: AI and the design of ethical systems. Ethics and Information Technology March 2018; 20(1):15–25.

72. David J. Gunkel. The machine question: critical perspectives on AI, robots, and ethics. Cambridge, MA: The MIT Press; 2012.

73. James H. Moor. The nature, importance, and difficulty of machine ethics. IEEE Intelligent Systems 2006; 21(4)18-21.

74. Eleanor Sandry. Re-evaluating the form and communication of social robots – The benefits of collaborating with machinelike robots. International Journal on Social Robotics 2015; 7(3):335-346.

75. Michael Anderson and Susan Leigh Anderson. Machine ethics: Creating an Ethical Intelligent Agent. AI Magazine 2007; 28(4):15–26.

76. Ronald L. Sandler, editor. Ethics and emerging technologies. New York, NY: Palgrave Macmillan; 2014.

77. Michael Anderson, & Susan Leigh Anderson, editors. Machine ethics. New York, NJ: Cambridge University Press; 2011.

*software – application environment for any of code writing, testing, or running applications developed during this project in the desktop or virtual environment.

†hardware – device optimized for running deep learning applications that increases performance up to 60 times what is obtainable on a conventional CPU. And the device is relatively cheap compared to average cost of a CUDA™ GPU. This hardware is optimized for running the AVOD or TMOR software with real-time capability.

Appendices

Appendix 1. Table of Figures

Appendix 2. Model Trainer (MT) Detailed Training Printouts

```
In [20]: %run TAE_basic_model_training.py
Reading map file: data\CIFAR-10\train_map.txt
Reading mean file: data\CIFAR-10\CIFAR-10_mean.xml
Reading map file: data\CIFAR-10\test_map.txt
Reading mean file: data\CIFAR-10\CIFAR-10_mean.xml
C:\Users\darli\Anaconda3\lib\site-packages\cntk\learners\__init__.py:340: RuntimeWarning: When providing the schedul
  warnings.warn('When providing the schedule as a number, epoch_size is ignored', RuntimeWarning)
Training 116906 parameters in 10 parameter tensors.

Learning rate per 200 samples: 0.01
Momentum per minibatch: 0.9
Finished Epoch[1 of 500]: [Training] loss = 2.251230 * 50000, metric = 82.89% * 50000 1.951s (25627.9 samples/s);
Finished Epoch[2 of 500]: [Training] loss = 2.061004 * 50000, metric = 75.28% * 50000 1.864s (26824.0 samples/s);
Finished Epoch[3 of 500]: [Training] loss = 1.908120 * 50000, metric = 69.70% * 50000 1.865s (26809.7 samples/s);
Finished Epoch[4 of 500]: [Training] loss = 1.769618 * 50000, metric = 64.64% * 50000 1.876s (26652.5 samples/s);
Finished Epoch[5 of 500]: [Training] loss = 1.677931 * 50000, metric = 61.57% * 50000 1.870s (26738.0 samples/s);
Finished Epoch[6 of 500]: [Training] loss = 1.607599 * 50000, metric = 58.87% * 50000 1.895s (26385.2 samples/s);
Finished Epoch[7 of 500]: [Training] loss = 1.554459 * 50000, metric = 56.74% * 50000 1.870s (26738.0 samples/s);
Finished Epoch[8 of 500]: [Training] loss = 1.508423 * 50000, metric = 55.08% * 50000 1.872s (26709.4 samples/s);
Finished Epoch[9 of 500]: [Training] loss = 1.470598 * 50000, metric = 53.49% * 50000 1.876s (26652.5 samples/s);
Finished Epoch[10 of 500]: [Training] loss = 1.442877 * 50000, metric = 52.54% * 50000 1.870s (26738.0 samples/s);
Finished Epoch[11 of 500]: [Training] loss = 1.411008 * 50000, metric = 50.97% * 50000 1.877s (26638.3 samples/s);
Finished Epoch[12 of 500]: [Training] loss = 1.386901 * 50000, metric = 50.00% * 50000 1.872s (26709.4 samples/s);
Finished Epoch[13 of 500]: [Training] loss = 1.355249 * 50000, metric = 48.61% * 50000 1.866s (26795.3 samples/s);
Finished Epoch[14 of 500]: [Training] loss = 1.323815 * 50000, metric = 47.45% * 50000 1.894s (26399.2 samples/s);
Finished Epoch[15 of 500]: [Training] loss = 1.297701 * 50000, metric = 46.52% * 50000 1.875s (26666.7 samples/s);
Finished Epoch[16 of 500]: [Training] loss = 1.268213 * 50000, metric = 45.09% * 50000 1.863s (26838.4 samples/s);
Finished Epoch[17 of 500]: [Training] loss = 1.252005 * 50000, metric = 44.40% * 50000 1.873s (26695.1 samples/s);
Finished Epoch[18 of 500]: [Training] loss = 1.226836 * 50000, metric = 43.83% * 50000 1.862s (26852.8 samples/s);
Finished Epoch[19 of 500]: [Training] loss = 1.211971 * 50000, metric = 43.05% * 50000 1.877s (26638.3 samples/s);
Finished Epoch[20 of 500]: [Training] loss = 1.185830 * 50000, metric = 42.17% * 50000 1.871s (26723.7 samples/s);
Finished Epoch[21 of 500]: [Training] loss = 1.166423 * 50000, metric = 41.15% * 50000 1.869s (26752.3 samples/s);
Finished Epoch[22 of 500]: [Training] loss = 1.154230 * 50000, metric = 40.91% * 50000 1.869s (26752.3 samples/s);
Finished Epoch[23 of 500]: [Training] loss = 1.136479 * 50000, metric = 40.19% * 50000 1.904s (26260.5 samples/s);
Finished Epoch[24 of 500]: [Training] loss = 1.121626 * 50000, metric = 39.57% * 50000 1.919s (26055.2 samples/s);
Finished Epoch[25 of 500]: [Training] loss = 1.106493 * 50000, metric = 39.04% * 50000 1.910s (26178.0 samples/s);
Finished Epoch[26 of 500]: [Training] loss = 1.090134 * 50000, metric = 38.49% * 50000 1.873s (26695.1 samples/s);
Finished Epoch[27 of 500]: [Training] loss = 1.080697 * 50000, metric = 37.91% * 50000 1.897s (26357.4 samples/s);
Finished Epoch[28 of 500]: [Training] loss = 1.065992 * 50000, metric = 37.40% * 50000 1.872s (26709.4 samples/s);
Finished Epoch[29 of 500]: [Training] loss = 1.057427 * 50000, metric = 37.22% * 50000 1.882s (26567.5 samples/s);
Finished Epoch[30 of 500]: [Training] loss = 1.048681 * 50000, metric = 36.63% * 50000 1.880s (26595.7 samples/s);
```

Image 1. Basic model showing convergence with steeply falling loss.

```
In [24]: %run TAE_pred_vgg.py
Reading map file: data\CIFAR-10\train_map.txt
Reading mean file: data\CIFAR-10\CIFAR-10_mean.xml
Reading map file: data\CIFAR-10\test_map.txt
Reading mean file: data\CIFAR-10\CIFAR-10_mean.xml
Basic Model Prediction WITH VGG9
Training 2675978 parameters in 18 parameter tensors.

Learning rate per 200 samples: 0.01
Momentum per minibatch: 0.9
C:\Users\darli\Anaconda3\lib\site-packages\cntk\learners\__init__.py:340: RuntimeWarning: When providing the schec
  warnings.warn('When providing the schedule as a number, epoch_size is ignored', RuntimeWarning)
Finished Epoch[1 of 200]: [Training] loss = 2.298223 * 50000, metric = 85.86% * 50000 5.608s (8915.8 samples/s);
Finished Epoch[2 of 200]: [Training] loss = 2.269297 * 50000, metric = 79.99% * 50000 5.356s (9335.3 samples/s);
Finished Epoch[3 of 200]: [Training] loss = 2.099337 * 50000, metric = 77.18% * 50000 5.374s (9304.1 samples/s);
Finished Epoch[4 of 200]: [Training] loss = 1.985987 * 50000, metric = 73.75% * 50000 5.395s (9267.8 samples/s);
Finished Epoch[5 of 200]: [Training] loss = 1.896656 * 50000, metric = 69.93% * 50000 5.433s (9203.0 samples/s);
Finished Epoch[6 of 200]: [Training] loss = 1.778256 * 50000, metric = 66.07% * 50000 5.421s (9223.4 samples/s);
Finished Epoch[7 of 200]: [Training] loss = 1.701244 * 50000, metric = 63.05% * 50000 5.416s (9231.9 samples/s);
Finished Epoch[8 of 200]: [Training] loss = 1.652821 * 50000, metric = 61.18% * 50000 5.362s (9324.9 samples/s);
Finished Epoch[9 of 200]: [Training] loss = 1.610633 * 50000, metric = 59.29% * 50000 5.349s (9347.5 samples/s);
Finished Epoch[10 of 200]: [Training] loss = 1.568056 * 50000, metric = 57.58% * 50000 5.345s (9354.5 samples/s);
Finished Epoch[11 of 200]: [Training] loss = 1.527095 * 50000, metric = 55.93% * 50000 5.328s (9384.4 samples/s);
Finished Epoch[12 of 200]: [Training] loss = 1.487198 * 50000, metric = 53.86% * 50000 5.340s (9363.3 samples/s);
Finished Epoch[13 of 200]: [Training] loss = 1.447777 * 50000, metric = 52.41% * 50000 5.351s (9344.0 samples/s);
Finished Epoch[14 of 200]: [Training] loss = 1.409334 * 50000, metric = 50.76% * 50000 5.424s (9218.3 samples/s);
Finished Epoch[15 of 200]: [Training] loss = 1.388840 * 50000, metric = 49.96% * 50000 5.321s (9396.7 samples/s);
Finished Epoch[16 of 200]: [Training] loss = 1.350448 * 50000, metric = 48.56% * 50000 5.317s (9403.8 samples/s);
Finished Epoch[17 of 200]: [Training] loss = 1.318652 * 50000, metric = 47.12% * 50000 5.359s (9330.1 samples/s);
Finished Epoch[18 of 200]: [Training] loss = 1.290904 * 50000, metric = 46.19% * 50000 5.335s (9372.1 samples/s);
Finished Epoch[19 of 200]: [Training] loss = 1.260824 * 50000, metric = 45.04% * 50000 5.339s (9365.0 samples/s);
Finished Epoch[20 of 200]: [Training] loss = 1.230425 * 50000, metric = 43.99% * 50000 5.380s (9293.7 samples/s);
Finished Epoch[21 of 200]: [Training] loss = 1.206871 * 50000, metric = 43.29% * 50000 5.345s (9354.5 samples/s);
Finished Epoch[22 of 200]: [Training] loss = 1.182963 * 50000, metric = 42.10% * 50000 5.327s (9386.1 samples/s);
Finished Epoch[23 of 200]: [Training] loss = 1.152332 * 50000, metric = 40.96% * 50000 5.337s (9368.6 samples/s);
Finished Epoch[24 of 200]: [Training] loss = 1.128602 * 50000, metric = 39.90% * 50000 5.347s (9351.0 samples/s);
Finished Epoch[25 of 200]: [Training] loss = 1.104563 * 50000, metric = 39.33% * 50000 5.341s (9361.5 samples/s);
Finished Epoch[26 of 200]: [Training] loss = 1.075919 * 50000, metric = 38.30% * 50000 5.387s (9281.6 samples/s);
Finished Epoch[27 of 200]: [Training] loss = 1.054931 * 50000, metric = 37.53% * 50000 5.348s (9349.3 samples/s);
Finished Epoch[28 of 200]: [Training] loss = 1.041165 * 50000, metric = 36.97% * 50000 5.336s (9370.3 samples/s);
Finished Epoch[29 of 200]: [Training] loss = 1.013648 * 50000, metric = 35.80% * 50000 5.423s (9220.0 samples/s);
Finished Epoch[30 of 200]: [Training] loss = 1.001088 * 50000, metric = 35.46% * 50000 5.414s (9235.3 samples/s);
```

Image 2. VGG model lost nearly 65% of its metric after 30 epochs.

Appendix 3. TMOR App Analysing Web Images

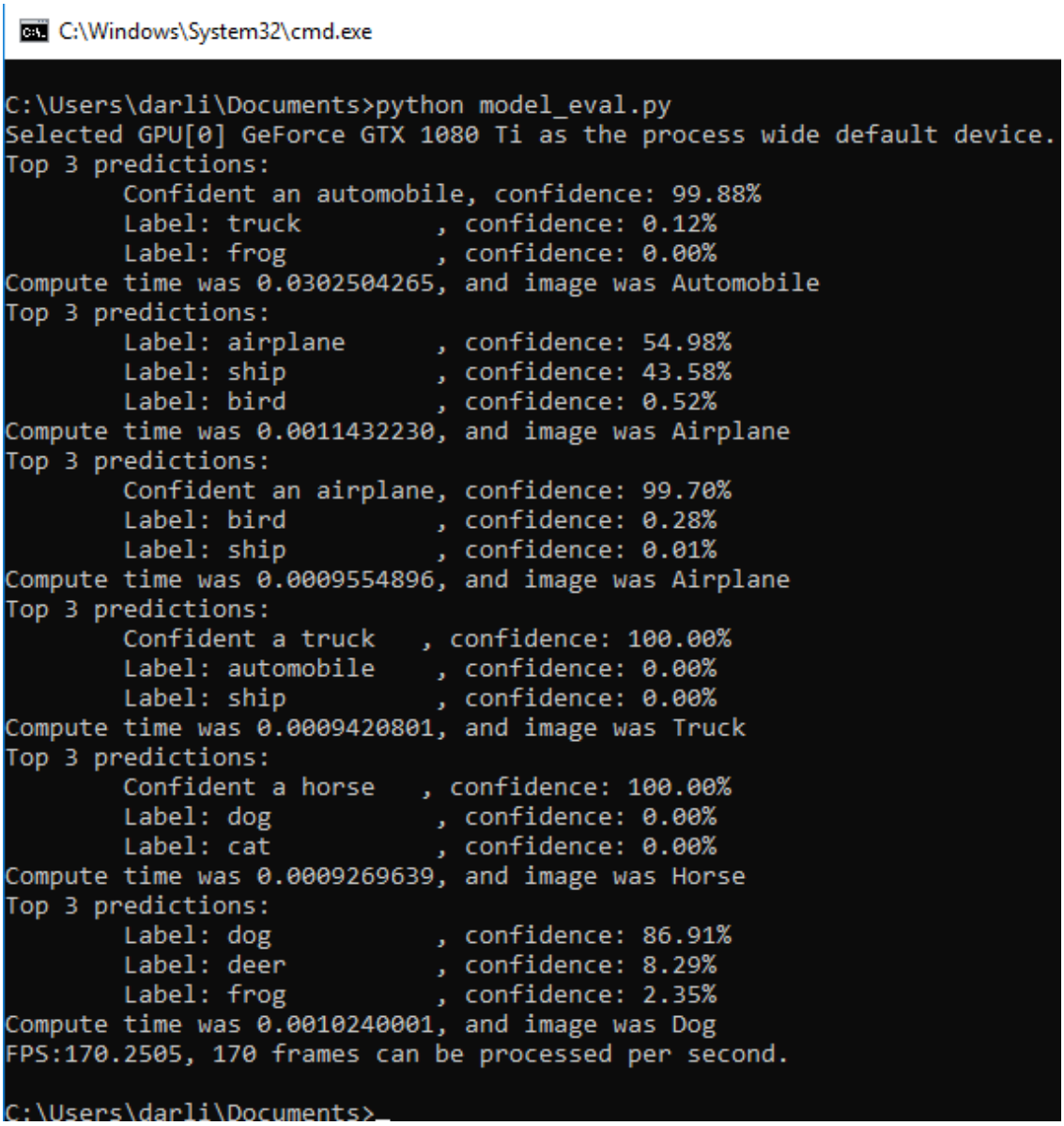


Image 3. TMOR using trained VGG model with FPS capability of 170 framerate (170fps).

Appendix 4. A Component Application of TMOR App

```
def evaluate_model(pred_op, image_path):
    label_lookup = ["airplane", "automobile", "bird", "cat", "deer",
"dog", "frog", "horse", "ship", "truck"]
    st_time = time.clock()
    bgr_image = np.asarray(Image.open(image_path), dtype=np.float32) -
127.5
    bgr_image = bgr_image[..., [2, 1, 0]]
    pic = np.ascontiguousarray(np.rollaxis(bgr_image, 2))

    result = np.squeeze(pred_op.eval({pred_op.arguments[0]:[pic]}))

    name =""
    names =
['Automobile','Airplane','Dog','Horse','Car','Ship','Truck','Bird']
    test_name = '_PDHCSTB'
    name = names[test_name.find(image_path[0])]
    # Return top 3 results:
    top_count = 3
    result_indices = (-np.array(result)).argsort()[:top_count]
    end_time = time.clock()
    print("Top 3 predictions:")
    vowels ='oiuea'
    for i in range(top_count):
        label = label_lookup[result_indices[i]]
        conf = result[result_indices[i]] * 100
        article = 'an'
        if vowels.find(label_lookup[result_indices[i]][:1]) == -1:
            article ='a'
        if  result[result_indices[i]] * 100 >= 99.0:
            print("\tConfident {:s} {:8s}, confidence:
{:.2f}%".format(article,label, conf))
        elif result[result_indices[i]] * 100 > 88.0:
            print("\tMost likely {:s} {:7s}, confidence:
{:.2f}%".format(article,label, conf))
        else:
            print("\tLabel: {:14s}, confidence: {:.2f}%".format(label,
conf))

    print('Compute time was {:.10f}, and image was
{:s}'.format(end_time-st_time,name))
    return (end_time-st_time)
```

Listing 1. An image evaluation function in any of 10 trained classes.

```
import numpy as np
import os, sys
from cntk.ops.functions import load_model
from PIL import Image
import scipy.misc
import cv2
import time

from IPython.display import Image as display
from resizeimage import resizeimage
```

Listing 2. Required libraries to run Listings 3 and 4 .

```
##Now load one of the trained models, her we use VGG, hopefully in
same directory
z = load_model('pred_vgg9.dnn')
#display any image if you like in next line
display("Car.png", width=50, height=50,embed=True)
```

Listing 3. Displaying a sample image.

```
#we assume all images listed here are in same directory as the code
sample and of size 32 x 32 (w x h)
eval_model(z,'_Car_small.jpg')
eval_ model (z,'P_sm.png')
eval_ model (z,'P_2.png')
eval_ model (z,'T_1_s.jpg')
eval_ model (z,'H_1_s.jpg')
eval_ model (z,'D_3_Boy_s.png')
eval_ model (z,'D_d_small.jpg')
```

Listing 4. Evaluating a model with images.

```
files =
['_Car_small.jpg','P_sm.png','P_2.png','T_1_s.jpg','H_1_s.jpg','D_3_Bo
y_s.png']

def test_many(model, files):
        ti = 0.0
        for i in range(len(files)):
            ti+=evaluate_model(model,files[i])
        print('FPS:{ :.4f}, {} can be processed per
second.'.format((len(files)/ti),int(len(files)/ti)))

z = load_model('pred_vgg9.dnn')

test_many(z,files)
# FPS:137.58552212270638, 137 frames can be processed per second.
```

Listing 5. Testing frames to get FPS.

Appendix 5. Detailed Screenshots of ODM Android® App

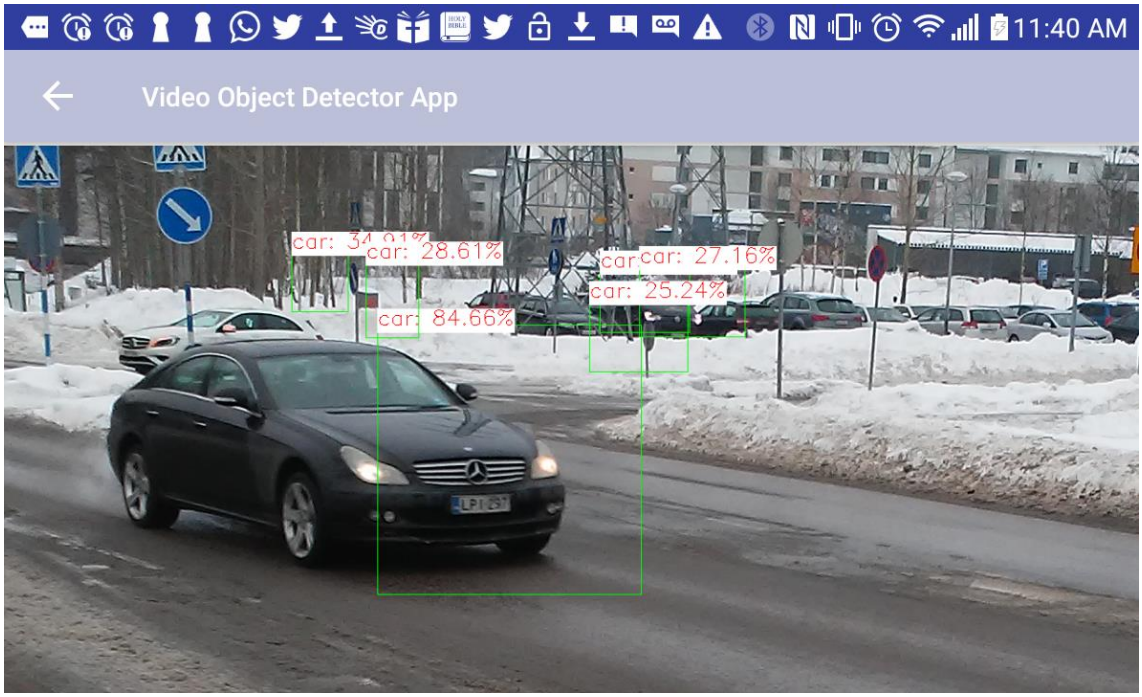


Image 4. ODM (mobile) app recognized and detected a moving car.

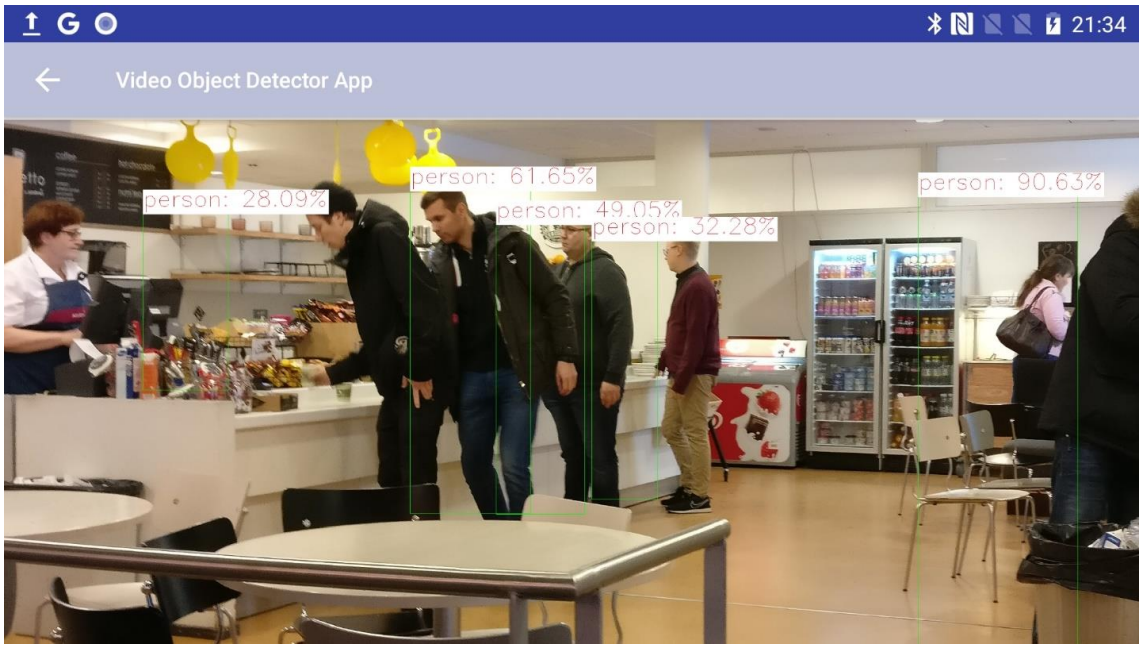


Image 5. ODM app recognizing and detecting persons in an indoor environment.

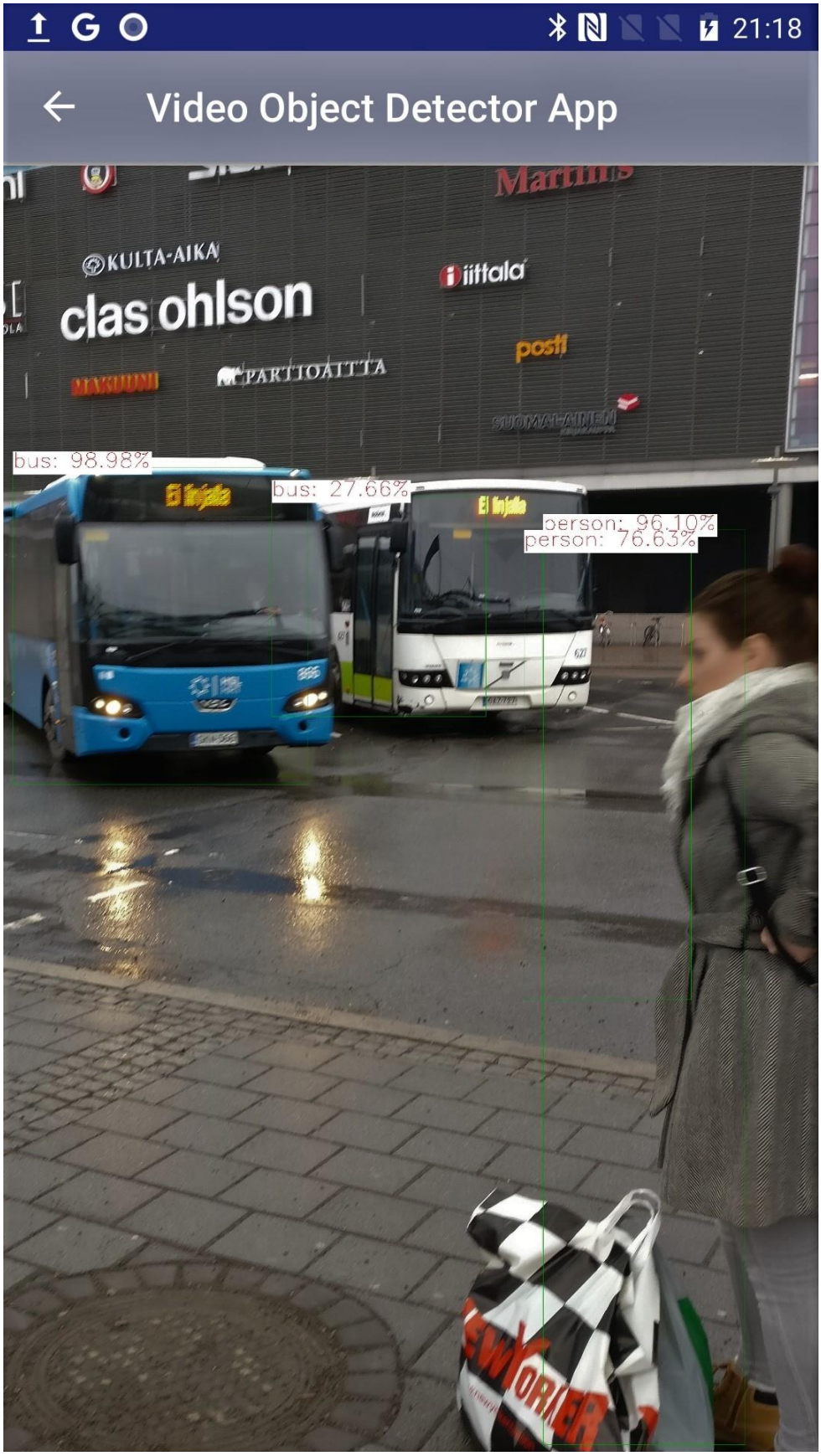


Image 6. ODM app detected a pedestrian on a sidewalk with buses.

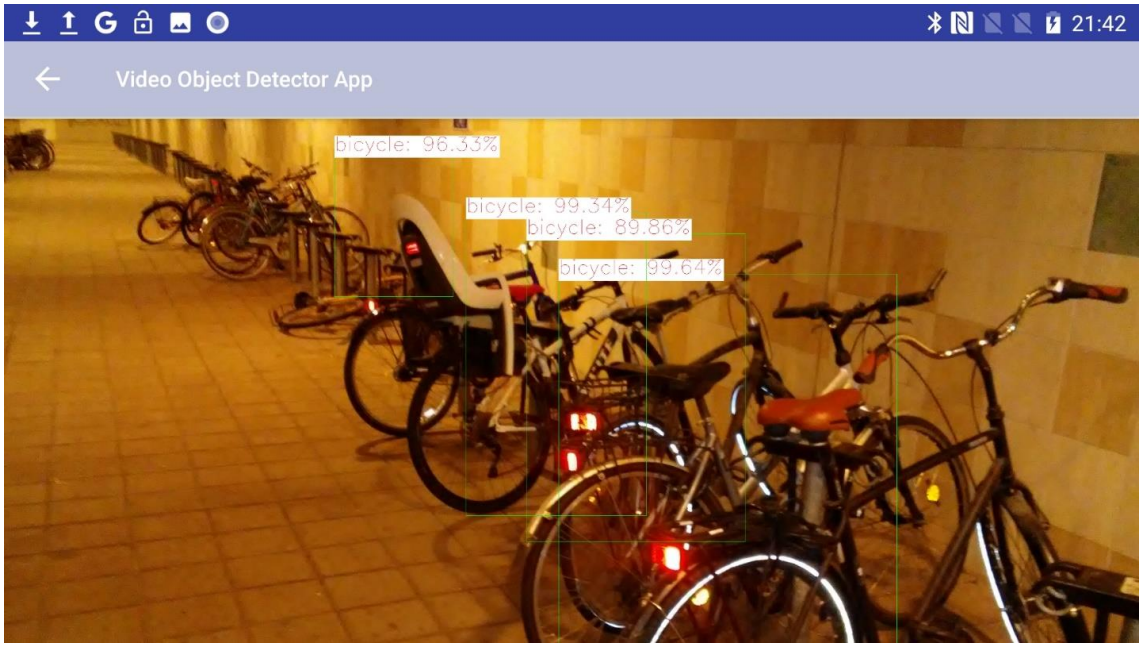


Image 7. Mobile app (ODM) detected parked bicycles at night with commendable confidence.

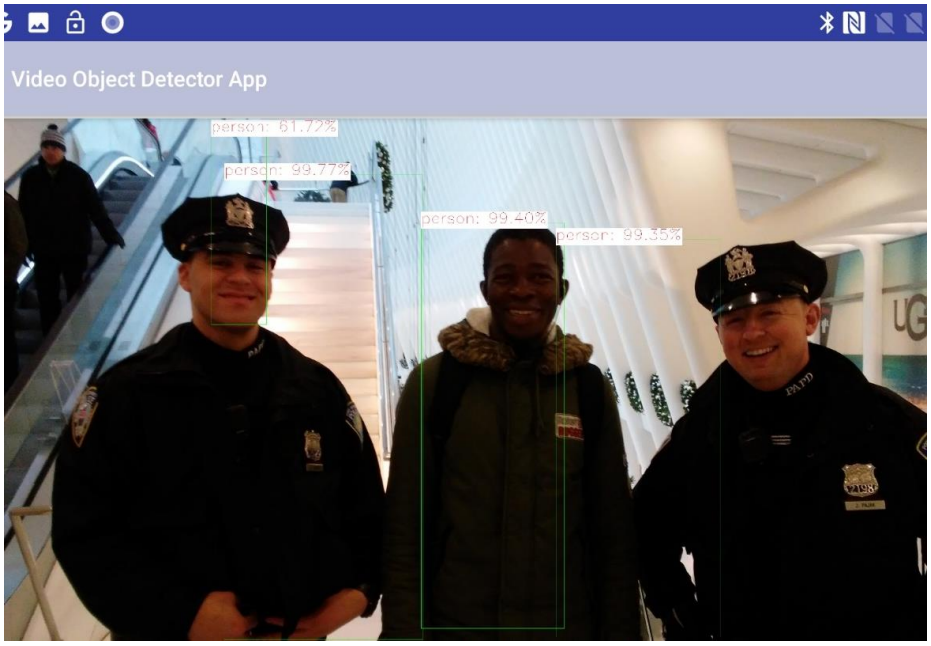


Image 8. The police officers are handsome with the ODM app.

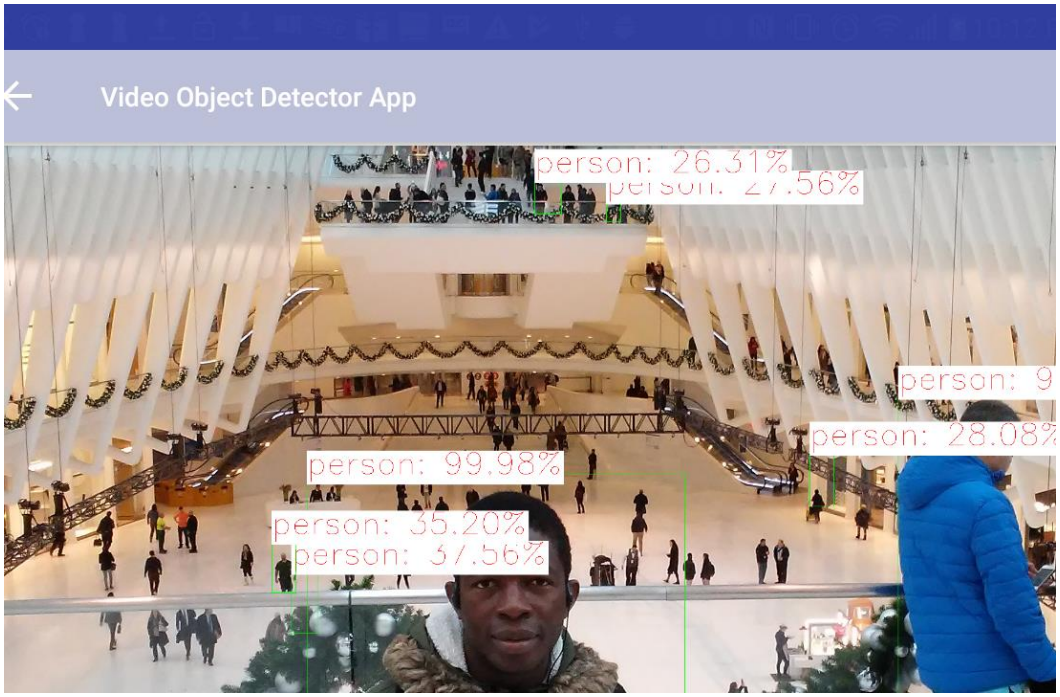


Image 9. The ODM app identified people in an indoor crowded area.

Appendix 6. Some Other Screenshots of AVOD App

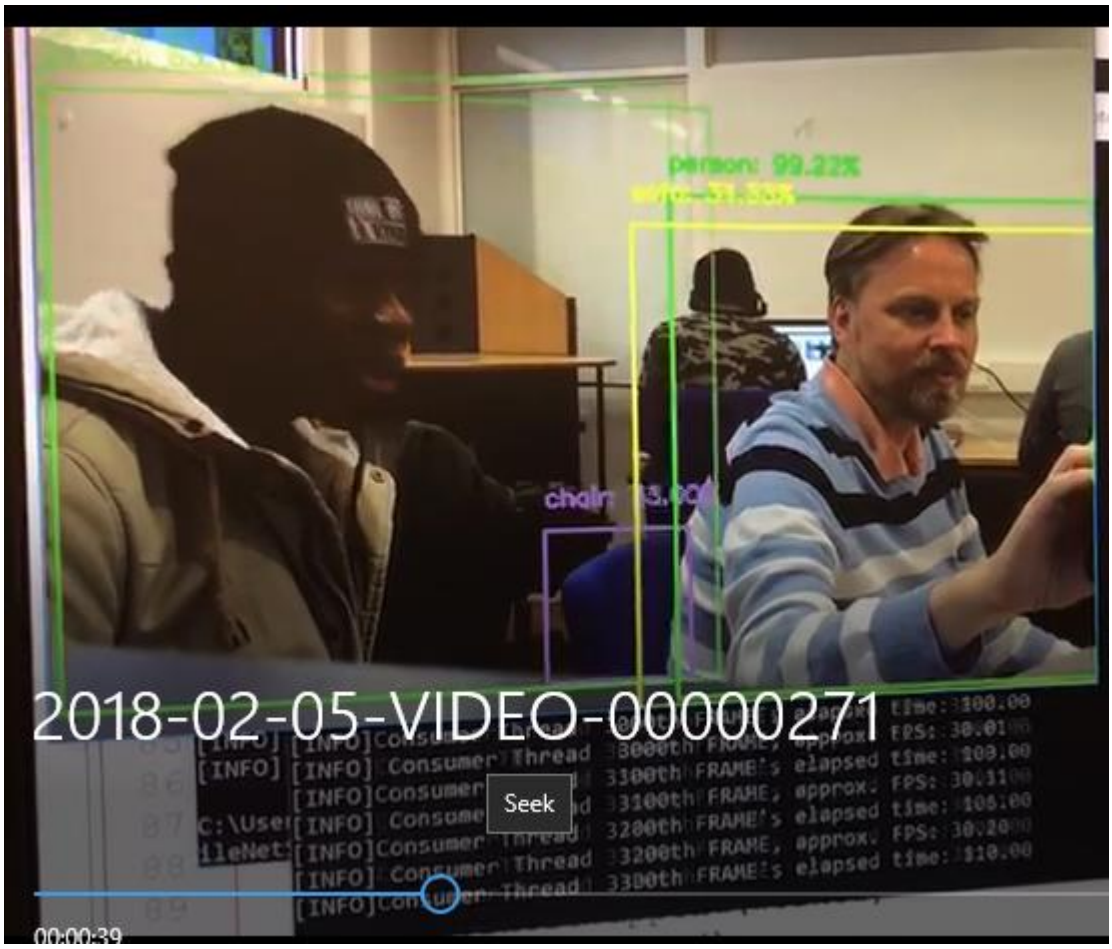


Image 10. AVOD App in real-time demo processing at camera's framerate (30fps).

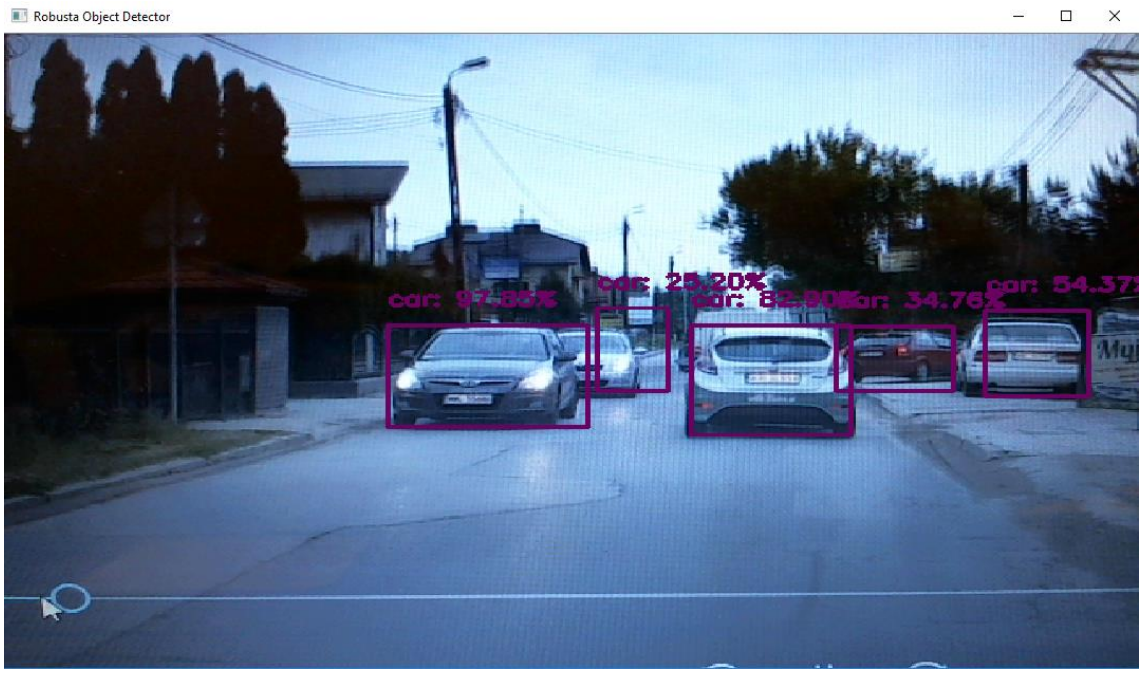


Image 11. AVOD App identifying moving cars.

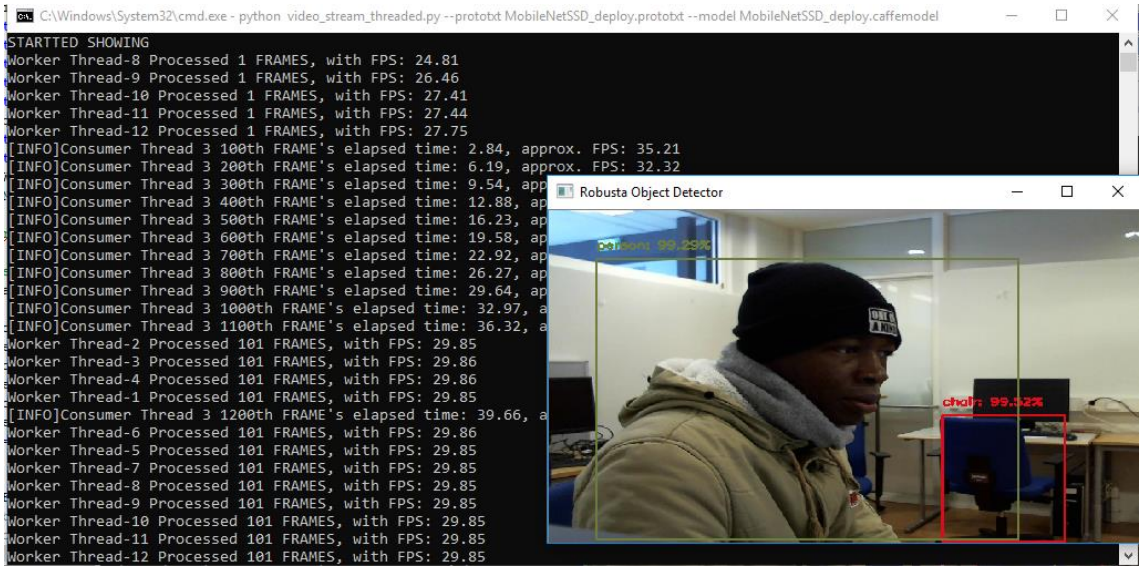


Image 12. AVOD app's FPS output with image capture.