ARTIFICIALLY INTELLIGENT ROBOTIC ARM

A Project Report

Submitted by

Deep Vyas	**111507062**
Shreeyash Pawar	**111407044**
Bharat Bodkhe	**111507010**
Omkaresh Pali	**111507038**

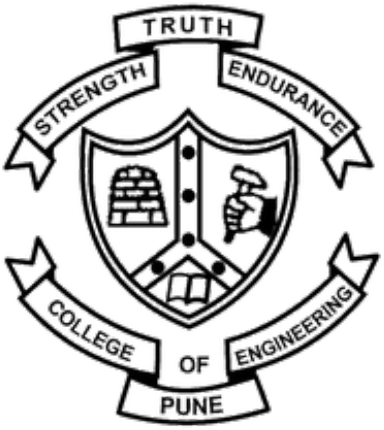
in partial fulfillment for the award of the degree

of

B.Tech Electronics and Telecommunication

Under the guidance of

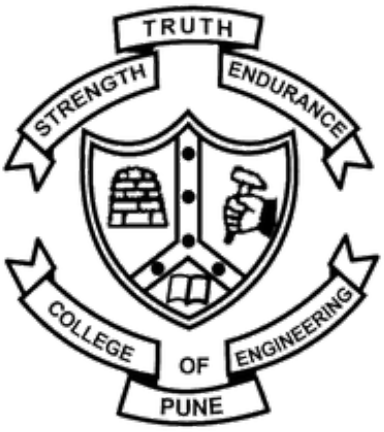
Prof. Dr. S.P. Metkar



DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, PUNE 2018-19

CERTIFICATE



This is to certify that the report entitled "AI Powered Robotic Arm" submitted by Deep Vyas (MIS No. 111507062), Shreeyash Pawar (MIS No. 111407044), Bharat Bhodke (MIS No. 111507010) and Omkaresh Pali (MIS No. 111407038) in the partial fulfillment of the requirement for the award of degree of Bachelor of Technology (Electronics and Telecommunication Engineering) of College of Engineering Pune, affiliated to the Savitribai Phule Pune University, is a record of their own work.

Prof. Dr. S.P. Metkar
Project Guide
Department of
Electronics and Telecomm,
College of Engineering
Pune

Dr. S.P. Mahajan
Head of the Department
Department of
Electronics and Telecomm,
College of Engineering
Pune

Date:

Place:

This report entitled

ARTIFICIALLY INTELLIGENT ROBOTIC ARM

By

Deep Vyas	111507062
Shreeyash Pawar	111407044
Bharat Bodkhe	111507010
Omkaresh Pali	111507038

Is approved for the degree of

B.Tech Electronics and Telecommunication

of

Department of Electronics and Telecommunication,

College of Engineering, Pune.

(An autonomous institute of Govt. of Maharashtra)

Examiners	**Name**	**Signature**
1. External Examiner	_____	_____
2. Internal Examiner	_____	_____
3. Supervisor (s)	_____	_____
	_____	_____

Date:

Place:

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature: _____

Student Name: _____

MIS Number: _____

Date:

Place:

Acknowledgements

We are grateful to our College and Department of Electronics and Telecommunication for providing us this great opportunity of working on this project. It is their visionary objective to encourage student for the curriculum oriented project that has created this extraordinary opportunity for us.

The authors sincerely thank our project guide Prof. Dr. S.P. Metkar for her guidance towards the design and development phases of the project. The key concepts and fundamentals could not have been understood to their fullest extent without her support. Our sincere thanks to all the faculties of department for their immense support. Special thanks to our faculty adviser Ms. Vanita Agarwal for her support. The authors also like to thank Mrs. Varada Kulkarni and Mr. Sushil Ronde for evaluating our project from time to time and for suggesting adequate improvements.

The team has greatly beneted from the faculty cooperation. It is their guidance and excellence in respective elds that makes the project a sustainable venture. The various facilities and labs made available to the team by the department have been instrumental in our consistent progress. The provision of the required resources by the department has really helped us throughout the tenure of the project. The team is immensely grateful to Head of the Department Dr. S.P. Mahajan. It is her support and encouragement to students to work on a project of their eld of interest that has made this possible.

Abstract

The advancements in computer vision, from traditional techniques to deep learning methods have been monumental, achieving human level accuracies in image classification. The use of such advancements hasn't reached the potential in aiding the people in process and manual industries. In our project, we aim to bridge this gap by assisting the people in warehouse industry . We aim to target picker to packaging work flow within a typical e-commerce warehouse, where a picker still manually sorts an item to respective collection bins, resulting in high inefficiencies, more human resource, and more pressure to deliver in tight e-commerce deadlines. To sort a product in real time conveyor belt operations, we can't use central cloud interfaces for computation as they introduce latency, require large bandwidth, and have high subscription costs to be realizable. If not for cloud, the system should be small enough to fit in dynamic closed spaces of warehouse processes. Hence, we deployed our AI model on the edge using a powerful single board computer with our optimized custom OS layer in middle. Our methodology has been to create a system in embedded Linux with installed libraries to support our application. On application side, we have contributed datasets, configured and trained models for our use case. Our results show the best practices in training of model using conveniences of transfer learning, and we conclude the best trade-off between accuracy and speed, to run on resource constrained embedded device. Future work is concerned with making the robotics more collaborative with humans, and integrating IOT protocols to interact with other devices to create a deploy able and scalable warehouse system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Object recognition is one of the most important and challenging problems in computer vision. The ability to classify objects plays a crucial role in scene understanding, and is a key requirement for autonomous robots operating in both indoor and outdoor environments. Recently, com- puter vision has witnessed significant progress, leading to impressive performance in various detection and recognition tasks. On the one hand, this is partly due to the recent advancements in machine learning techniques such as deep learning, fueled by a great interest from the research community as well as a boost in hardware performance. On the other hand, publicly-available datasets have been a great resource for bootstrapping, testing, and comparing these techniques.

But the aspect of process industries using deep learning as a solution has remained much unexplored. Human workers are still employed on manual task of sorting the product into respective categories for further processing which can be efficiently automated using combination of deep learning for computer vision and robotics, eliminating any need for human intervention thereby saving on huge operating costs and human resource.

As inventory management already being a tedious process, is made more cumbersome by legacy robotic management systems which in turn depletes

the human resource of a company. Recent advancements in field of computer vision in deep learning as a solution have remained unexplored in such industries. Moreover, as deep learning algorithms get computationally heavy, the large scale implementation in embedded systems for robotics gets severely limited. Our project focuses on this aspects and make sorting system most efficient in the market.

One of the challenging domains where object recognition plays a key role is service robotics. A robot operating in un- structured, domestic environments has to recognize everyday objects in order to successfully perform tasks like tidying up, fetching objects, or assisting elderly people. This is not only challenging due to the difficult lighting conditions and occlusions in real-world environments, but also due to the large number of everyday objects and products that a robot can encounter.

1.1 Workflow

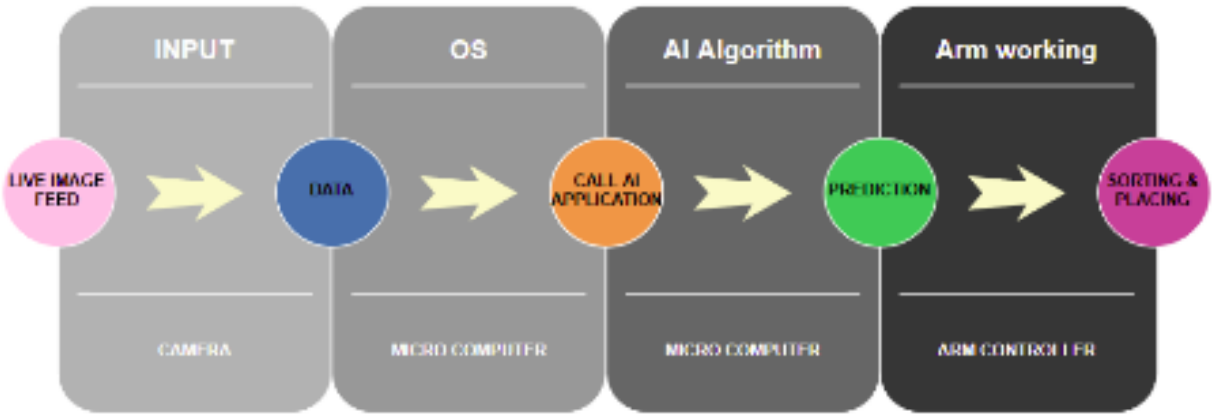


Figure 1.1: Project Workflow

Our Project workflow will go as shown in Figure 1.1,Customer will place item that is to be sorted and arm will put at it's appropriate position.

Chapter 2

Literature Survey

With advancement in AI for computer vision, image-processing techniques have improved exponentially. The field has become interdisciplinary with wide range of applications. In 2012 when Alex Krizhevsky proposed Alex Net [7], deep learning took a huge boom and a huge amount of research came up in past few years. Neural networks are extensively used in image processing algorithms. The Error rate of algorithms in vision has now became lesser than that of humans (5%) as neural networks keep on employing deeper layers in the past few years. There have been some recent advancements in the field of object detection algorithms and robotic arm used in warehouse solution. This chapter talks about some of this advancement that are going on this field.

- The earliest known industrial robot, conforming to the ISO definition was completed by "Bill" Griffith P. Taylor in 1937 and published in Meccano Magazine, March 1938[13]. The crane-like device was built almost entirely using Meccano parts, and powered by a single electric motor. From then many advancement has been taken place in this sector.
- A Review on Robot Arm Using Haptic Technology by Prof. A. Reshamwala, R. Singh discusses a system which utilizes human motion for

controlling the robot arm using Haptic technology[10].

- Various efforts in object sorting using have been made previously, M. Sabnis gave an approach to classify objects of by image processing algorithm on the parameters like color, shape and texture[9].

- Shubhangi Wanve, B.G.Gawalwad proposed a system that sorts object according to their color although in MATLAB. They used Arduino and Matlab for image processing which is a slower approach[12].

- J.D Gawande proposed a cost effective approach to design an automated system that can identify objects and relocate them if the object meets certain prescribed criterion using minimal image processing techniques[5].

- Saurin Sheth has given a minimal color based approach for object sorting based on microcontroller and processing to be done in MATLAB in real time[11].

On the contrary there have been few previous proposals to implement deep learning in industrial applications.

- M. Luckow proposed automatize use of deep learning in Industries in which he mainly focused on creation of automotive dataset for automatically recognizing vehicle properties[8].

- T. Wuest gave a machine learning approach to deal with industrial problems. None of these efforts explained the actual application deep learning could have in process of industry[14].

The focus of our implementation is going to be how we easily and accurately sort various products of the same category. As literature survey

continues more advanced feature may be part of this implementation such as intercommunication between such mass deployed systems is considered as future work.

2.1 Motivations

1. The industrial and warehouse operations for employees is a hectic job with strict deadlines and adverse working conditions for which people are generally over qualified for menial job of merely sorting the products into appropriate categories for further packaging.

2. Even a facility that makes heavy use of robots still employs people to do the actual picking for actual differentiation of similar dimensional products such as FMCG (Fast Moving Consumer Goods). Moreover, for an e-commerce store that are always on deadlines to deliver the product with 24/7 working warehouses, the employees are overworked by the companies with long working hours, which is appropriately covered by "BUISINESS INSIDER" media house.

3. The current advancements in AI for computer vision and deep learning methods are computationally heavy to be running on small embedded devices for complete IOT scalable systems. Hence, limiting the large scaling of such robotic systems as they are incapable of processing the data on board making it very inefficient to transfer data for a large warehouse where multiple systems needs to be employed, for processing on servers, especially when the data consists of live image feed and needs instant response when on typical conveyor belt.

4. The industry standard AI chips don't ship with an OS that also has

inbuilt packages and libraries for deep learning algorithms, hence we can't run the application out of the box.

2.2 Objectives

1. To automate management process efficiently using combination of robotics that employ AI on edge, which intelligently sorts the product categories using robotic arm.

2. To introduce automated picking system between the inventory storage and packaging process.

3. To reduce time delays for data transmission as data computing will be done on the board itself using high performance microprocessor.

4. To deploy our application on embedded Linux as a platform making it cheaper and customizable by open source communities.

Chapter 3

System Hardware

This chapter deals with the hardware aspect of project. It consists of robotic arm, its parameters like degrees of freedom, working envelope,speed and accuracy. Various robotic arm comparisons ,the arm that has been selected to the needs,its specification,application . Next comes the board which is Hickey 970, its first use,evolution and future advancements followed by the technical specification.

3.1 Proposed System

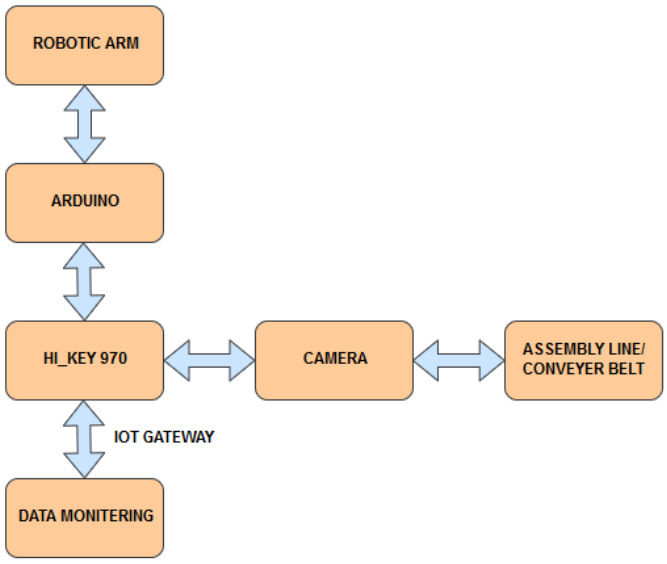


Figure 3.1: Block Diagram of Proposed System

Proposed system block diagram as shown in Figure 3.1 consists of various blocks like Robotic Arm, Hi-Key 970,Camera module and IOT Data monitoring.

3.2 Robotic Arm

Various parameters of Robotic arm are listed below,

3.2.1 Degrees of Freedom

Degrees of freedom of robotic arm controls movement It is the modes in which arm can move. DOF is equal to number of independent displacements.As shown in Figure 3.2, Arm can have more degrees of freedom than its dimension of working. Robot arm may have five to seven degrees of freedom. For e.g a robot having two arms has double DOF.

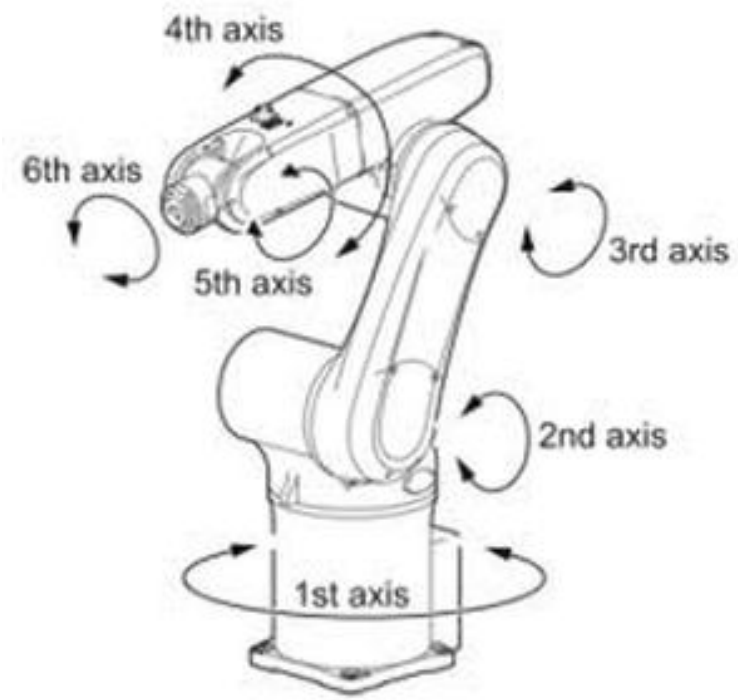


Figure 3.2: Robotic arm with 6 degrees of freedom

3.2.2 Working Principle

Range of motion of robotic arm is its working envelope It is shape created by range of arm.It is region in which arm can work freely in direction. Different modes of operation are shown in Figure 3.3

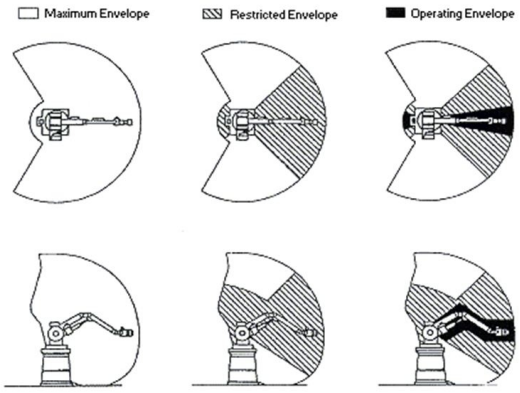


Figure 3.3: Modes of Operation

3.2.3 Payload

Payload is the amount of weight that the arm can lift. Pay load has to be taken into account when considering its application.

3.2.4 Speed and Acceleration

It consists of linear and angular movement. Robotic arm can reach a max velocity of 8m/s. Acceleration is the change in speed of the robotic arm considering its degree of freedom and working range.

3.2.5 Accuracy and repeatability

Accuracy is performing single task in specified time with no errors. Repeatability means getting the same output by performing it many times

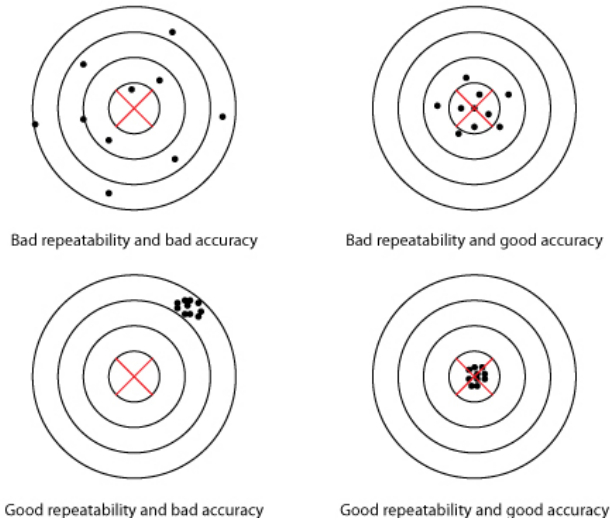


Figure 3.4: Accuracy and repeatability over 10 measurements

3.2.6 uArm Swift Pro

uArm Swift series are consumer level desktop robotic arms which are developed for makers and education purposes. Based on Arduino, uArm Swift series is open-source and DIY friendly. It's extremely easy to use, almost anyone can learn how to play with it in just a few minutes. It supports visual programming, which is beginner friendly. It also supports Arduino, Python programming.

uArm Swift series are open-source and DIY friendly robotic arms. With extension ports built-in on the base and end-effector,you can connect it with modules like OpenMV camera, Seeed Grove sensors, etc to expand it's possibilities. Whether you like to program with Arduino, Python.

Features

Offline Learning Mode, Open Source, Extendability, High Repeatability, Intuitive user interface, 3D printing, Laser Engraving and Drawing, Visual Programming.

Figure 3.5: uArm Swift Pro main body

Comparative Analysis

Specification comparison of different Robotic arm are shown in Table 3.1

Table 3.1: Specification comparison of Robotic Arms

Parameter	uArm Swift	uArm Swift Pro	Dobot M1	7Bot
Manufacturer	UFactory	UFactory	UFactory	UFactory
Size (mm)	150*138*281	150*140*281	640*230*681	310*280*150
Input Voltage	6V DC	12V DC	240V AC	7.5V DC
Speed	167m/s	100m/s	2000m/s	100m/s
Wireless	Bluetooth 4.0	Bluetooth 4.0	Bluetooth/WiFi	Bluetooth 4.0
Weight	1.2Kg	2.2Kg	4.5Kg	2.5Kg
DOF	4	4	4	6
Repeatability	5mm	0.2mm	0.02mm	0.2mm
Payload	500gm	500gm	1500gm	500gm
Range(mm)	50-320	50-320	20-400	90-340
Temperature(C)	0-40	0-40	0-65	0-50
Offline Learning	Yes	Yes	Yes	No
OpenMV Support	Yes	Yes	Yes	Yes
Software Support	Python	Python	Python/M1	Python
Price (Rs.)	78,000	96,000	1,56,000	80,000

3.3 HI-KEY 970

- HiKey 970 is development board made by Chinese company Huawei Technologies and launched at Linaro Development Conference
- It has HiSilicon Kirin 970 SoC(System on Chip) with HiAI architecture and dedicated NPUs(Neural Processing Unit).
- HiKey 970 has excellent capabilities of super edge AI computing which will provide developers powerful AI capabilities , Hardware acceleration with very promising performance .
- This board is designed for the applications based on Deep learning, Robotic applications, Automobile Industry and Real time big data high computing IoT applications.

Figure 3.6 shows Front side of Hi-Key 970 board.

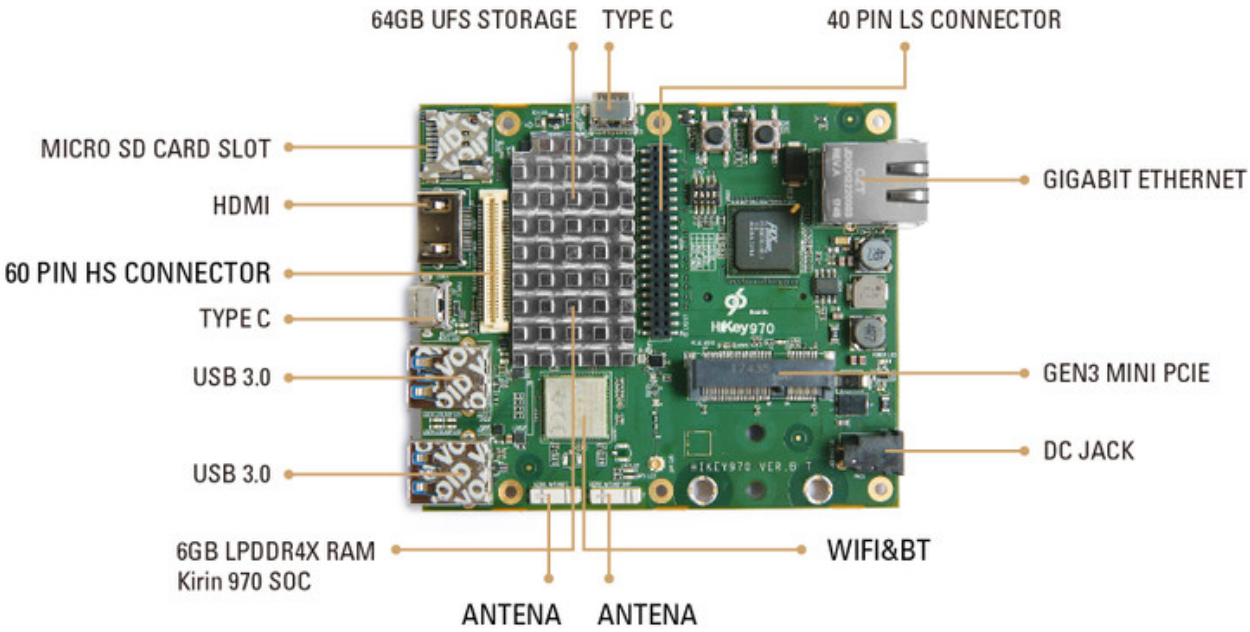


Figure 3.6: Front side of HiKey-970

3.3.1 Board Specifications

HiKey 970 Board Specifications are shown in the Table 3.2 below

Table 3.2: HiKey 970 Specifications.[4]

Component	Description
SoC:	HiSilicon Kirin 970 - HiAI Architecture, Dedicated NPU
CPU:	ARM Cortex-A73 MPCore4 @up to2.36GHz, ARM Cortex-A53 MPCore4 @up to1.8GHz
GPU:	ARM Mali-G72 MP12 GPU
RAM:	6GB LPDDR4X 1866MHz
Storage:	64GB UFS 2.1, Micro SD
Connectivity:	Bluetooth/WIFI/GPS
Video:	1080p@60Hz HDMI, 4 line MIPI/LCD port
Camera:	4 line MIPI port, 2 line MIPI port
Expansion Interface:	One 40-pin Low Speed (LS) expansion connector, UART, SPI, I2S, I2C x2, GPIO x12, DC power One 60-pin High Speed (HS) expansion connector 4L-MIPI DSI, USB, I2C x2, 2L+4L-MIPI CSI
User Interface:	Power/Reset 8 LED indicators 4 -user controllable 3 -for radios (BT and WLAN and GPS activity) 1 – for CAN
Power Source:	DC Power: +8V to +18V
OS Support:	Android, Linux
Mechanical:	105.26mm by 100mm meeting 96Boards Consumer Edition standard dimensions specifications.
Environmental:	Operating Temp: 0C to +70C RoHS and Reach compliant

Chapter 4

Object Detection Algorithms

Paul Vinola and Michel Jones invented the first efficient algorithm for face detection in 2001 since then object detection algorithms became very popular.They showed one of the first demonstration of human face detection in real time using computer vision techonology which was very advanced at that time and later it was implemented in OpenCV and face detection became very popular.Furthermore Navneet Dalal and Bill Triggs improved the traditional approach of pedestrian detection using Histograms of Oriented Gradients (HOG) algorithm in 2005.

Every year there was continuous significant improvement in object detection algorithms but when in 2012 Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton developed algorithm based on Deep learning in ImageNet Large Scale Visual Recognition Challenge (ILSVRC), it shook the world with very high accuracy of 85% which was 11% higher than of traditional approach.After which Convolutional Neural Network (CNN) based algorithms became trend achieving outstanding accuracy of 99%!

4.1 Traditional Algorithms

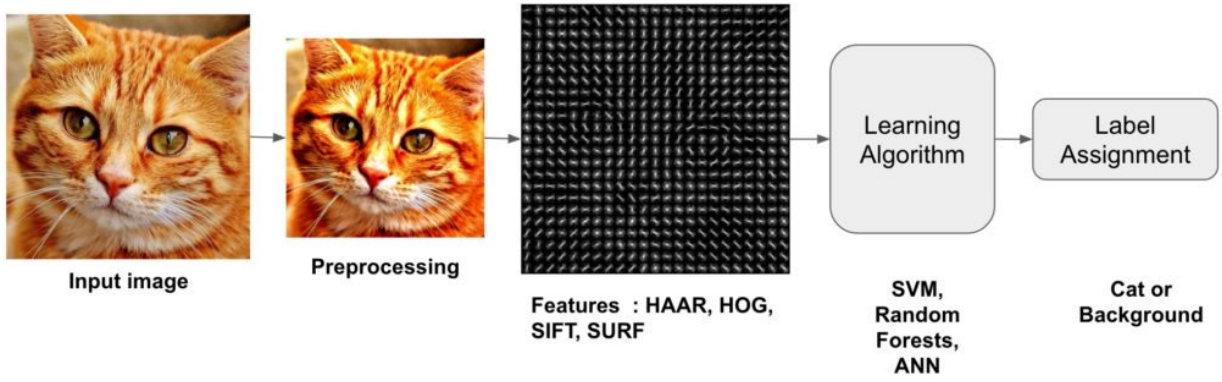


Figure 4.1: Block diagram of traditional object detection algorithm[3]

Various blocks in traditional object detection algorithm are shown in Figure 4.1

Preprocessing

Preprocessing includes normalizing contrast, brightness effects and mean image intensity.Gamma correction gives better results in Black and White images while for colour images color space transformation is used.Image is resized before sending to further feature extraction steps performed in preprocessing are generally trial and error basis.

Feature Extraction

First step in feature extraction is to remove redundant information which is not required for classification of objects.For example our aim is to detect shirt and coat buttons in image for that there is significant variation in RGB pixel values.This problem can be solved simply by using edge detection algorithms which can easily differentiate circular shape of buttons in these

images. In conclusion edge detection focuses on relevent information while neglecting unnecessary information for classification.Designing these features are important in performance of traditional computer vision algorithms.

Learning Algorithm For Classification

Output of Feature extraction step is given as input to learning algorithms and gives output class.Before classification we have to train our model on given dataset.Learning algorithms treat feature vectors as points in higher dimensional space and try to find distinguishing plane so that all examples of same class have a common side of a plane.

One such algorithm is SVM(Support Vector Machine) in which each feature vector is considered as point in higher dimensional space.For example consider your feature vectors are a 3780-dimensional, SVM will consider it as a point in a 3780 dimensional space and find the hyperplane separating different classes.As shown in diagram plane H3 differentiate two different classes denoted by black and white dots as shown in Figure 4.2

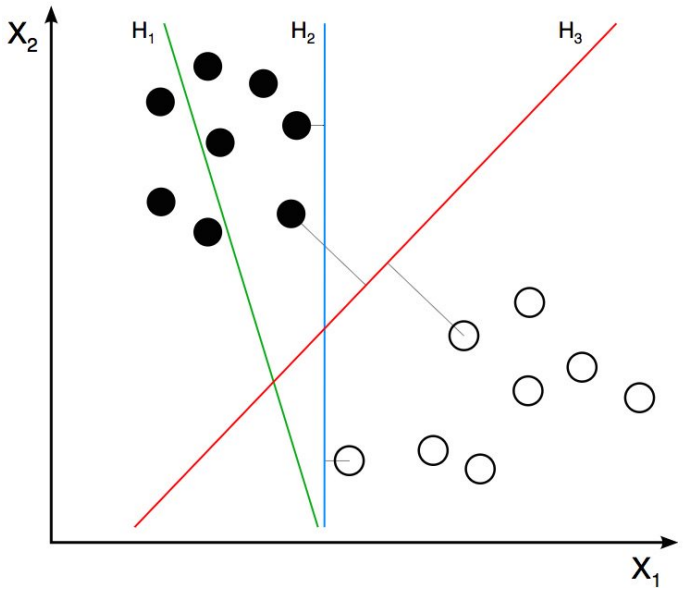


Figure 4.2: SVM Classification Example[3]

Optimizing

In some cases we can not find perfect plane differentiating classes in such cases SVM till finds hyperplane by tradding off some examples.This tradeoff is controlled by C parameter, so when value of C is small, we choose large margin hyperplane and when C is large, small margin hyperplane is chosen.The value of C is found using validation dataset for which algorithm was not trained on.

4.2 Convolutional Neural Networks

Convolutional neural network (CNNs) is one of the main categories to do object detection. CNN based model to train and test each image pass it through a series of convolution layers, filters, Pooling, fully connected layers and apply Softmax function to classify an object. Figure 4.3 shows how this operations takes place on image.

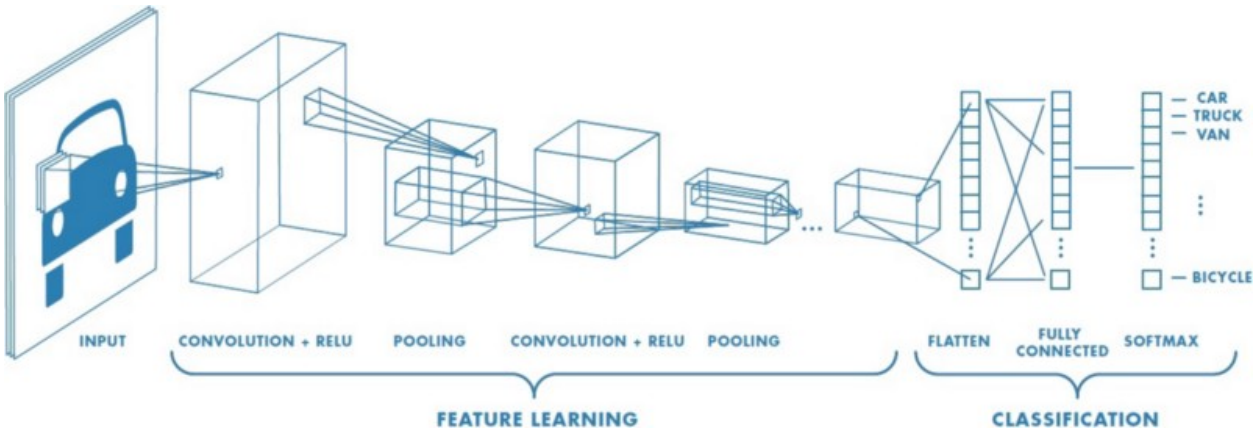


Figure 4.3: Operations in Convolutional Neural Networks

4.2.1 Blocks in Convolutional Neural Networks

Convolution Layer

Convolution is the first layer, it is mathematical operation that takes two inputs such as image matrix and a filter to extract features from an input image. The main advantage of Convolution is that it preserves the relationship between pixels.Different filters can be used to perform operation such as edge detection, blur and sharpening of image.

- An image matrix of dimension $(h \times w \times d)$
- A filter $(f_h \times f_w \times d)$
- Outputs a volume dimension $(h - f_h + 1) \times (w - f_w + 1) \times 1$

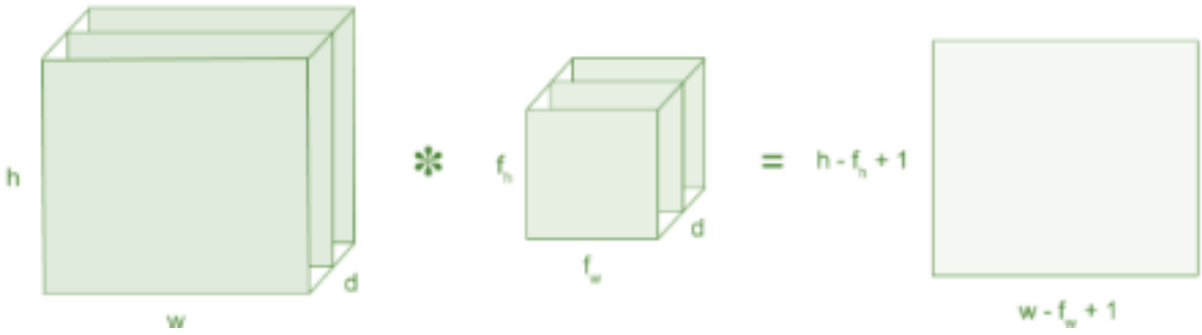


Figure 4.4: Convolutional Neural Networks:Features Extraction[2]

Padding

Filters does not fit image perfectly, we add rows and columns of zeros to input image to fit it perfectly.

Stride

Stride controls how fast filter moves over image. Consider when the stride is 1, filter is moved to 1 pixel at a time, when the stride is 2 filter is moved 2

pixels at a time.

Pooling

Pooling means taking dominant feature from available features to reduce number of features in case of big image. There are three types of pooling:

- Max Pooling: Take element with maximum value from chosen filter map
- Sum Pooling: Take sum of elements from chosen filter map
- Average Pooling: Take average of elements from chosen filter map

Flattening

We expand the convolved image pixel by pixel to feed in to network to apply activation functions and measure loss.

Fully connected layer

Output of flattening is provided as input to fully connected neural network as shown in Figure 4.5.
Each Layer is consists of many neurons. Three layer network:

- **Neurons:** Things that hold number between 0 and 1. This number is called activation (a_i). Activation in one layer determine activation in next layer.
- **Weights:** Weight is number assigned to connection between each neurons.
- **Weighted sum:** As shown in Figure 4.6 each neuron in next level, we calculate weighted sum according to this formula:

$$w_1a_1 + w_2a_2 + w_3a_3 + \cdots + w_na_n$$

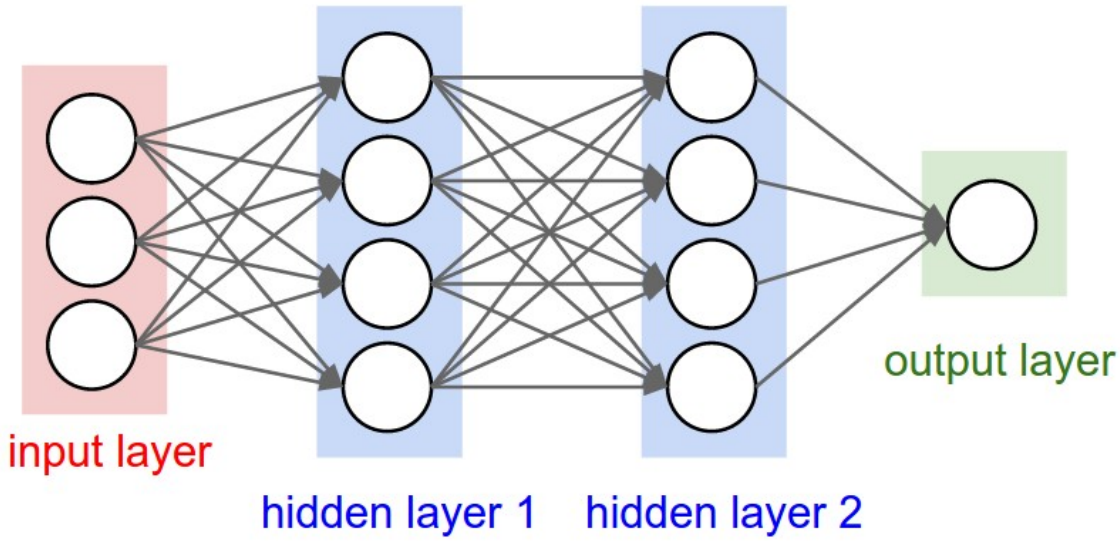


Figure 4.5: Layers:Convolutional Neural Networks

- **Activation Function:** We have used Rectified Linear Unit for a non-linear operation. The output is: $f(x) = max(0, x)$. The purpose of this unit is to introduce non-linearity in our ConvNet. As we want output to be only value between 0 and x.[2]

- **Bias:**It is parameter in neural network, which is used to adjust output along with weighted sum. It helps the model in a way that it can fit best for the given data.

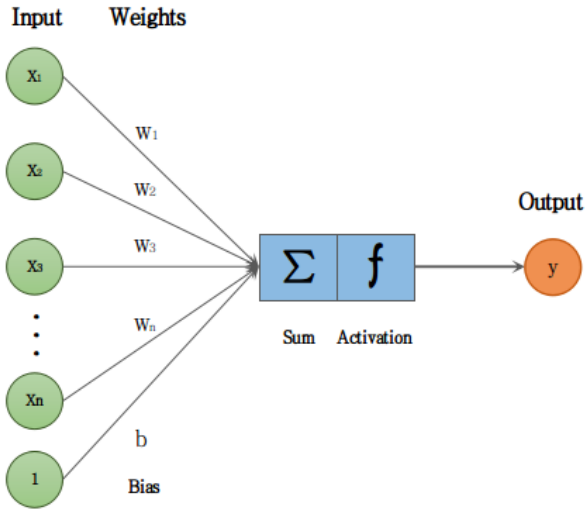


Figure 4.6: Weights:Convolutional Neural Networks

4.2.2 Convolutional Neural Network Models

Building a CNN model from scratch is tedious process. It also requires huge amount of data and time to train for good accuracy. "Transfer learning" is solving this challenge. Transfer learning is reuse of pre-trained model to solve new problem. Apart from less training data For example, Neural Networks usually try to detect edges earlier layers, shapes middle layer and some task-specific features in the later layers. With transfer learning, we use the early and middle layers and only retrain the later layers. Popular models for object detection include:

1. VGG
2. Inception-v3 model
3. Mobilenet ssdlite

VGG

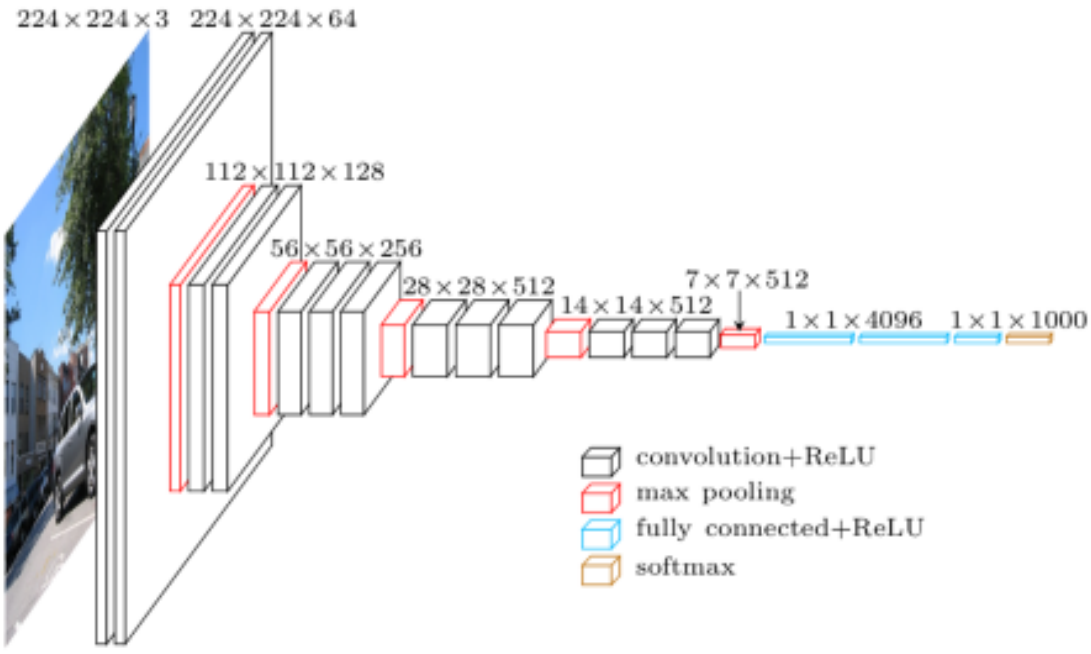


Figure 4.7: VGGNet: CNN model

First introduced in 2014 by Simonyan and Zisserman in their 2014 paper[6], Very Deep Convolutional Networks for Large Scale Image Recognition. This is one of the simplest network available, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Followed by max pooling, two fully connected layers, each with 4,096 nodes are passed through a softmax classifier.Training blocks of VGGNet are shown in Figure 4.7. The major drawbacks with VGGNet:

- More number of parameters to be trained lead to large training time.
- The weights of network architecture themselves are quite large.

Due number of fully-connected nodes and depth, the size of VGG is 533MB in VGG16 model and 574MB in VGG19 model. This makes deploying VGG on small arm devices like Hikey impossible.

Inception V3

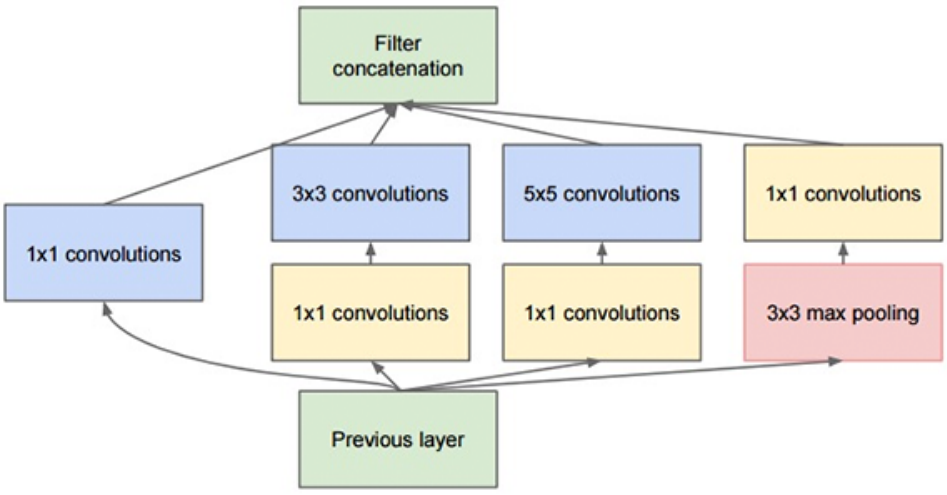


Figure 4.8: Inception: CNN model

The Inception model shown in Figure 4.8 is "Multi-level feature extractor". As each level it computes 1×1, 3×3, and 5×5 convolutions, max

pooling on previous layer as shown in figure above. Output of these filters are concatenated along channel dimension before fading into next layer. It is complicated model but we have control to choose parameters like 1×1, 3×3, and 5×5 convolutions, max pooling so model is small as it have less parameters compare to if we have performed simple convolution on each layer. The result of above implementation can be seen in size as it is reduced to 96MB.

MobileNet

As name suggest this is kind of neural network that designed to run on small hardware like mobile and is nearly as accurate as much larger convolutional networks like VGGNet.

Mobilenet uses depth wise separable convolutions and a pointwise convolution. They introduced two hyperparameters to trade-off between speed and accuracy. Two hyperparameters are: width multiplier and resolution multiplier. In MobileNets , each input channel is applied with depthwise convolutional filter. Further pointwise convolution applies a 11 convolutional filter, which combine the outputs the depthwise convolution. While in general convolution above two steps are merged into one.The effect of factorization has resulted in reduction in computation and model size.This layers are followed by a batchnorm and ReLU nonlinearity. Only final layer of fully connected layer has no nonlinearity and a softmax layer is applied for classification. Figure 4.10 shows comparative layers of regular convolution to that of used in mobilenetMobileNet has 28 layers, if we counted depthwise and pointwise convolutions as separate layers. Size of Mobilenet model is 50MB. Based on comparative study done on options available to implement object detection on edge device like arm, we conclude that deep learning based models have edge over traditional algorithm based models. We decided to go with

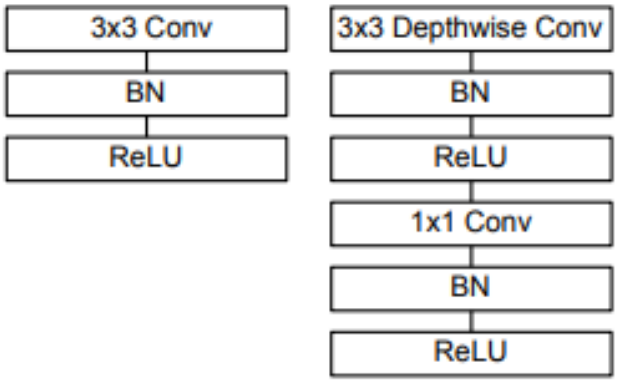
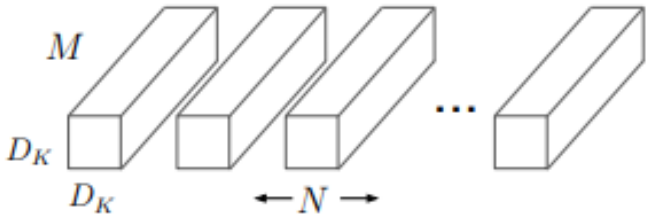
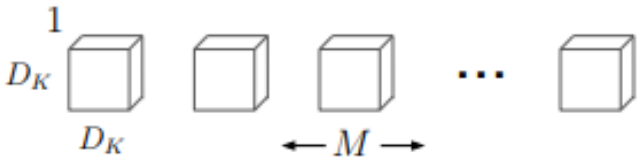


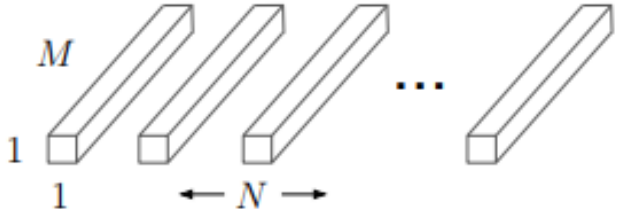
Figure 4.9: Mobilenet control flow



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 4.10: Mobilenet: CNN model

deep learning based model 'mobilenet', which we will use as transfer learning model for object detection.

Further original mobilenet model shown in FIgure 4.11 designed for object classification, is not very good for object detection. Since our aim object detection involves more complications than just classification, SSD adds many additional convolutional layers on top of the base network of mobile net as shown in Figure 4.12. In the case of object detection, we want to know not just high-level features but also lower-level ones, which is why we also read from the previous layers.

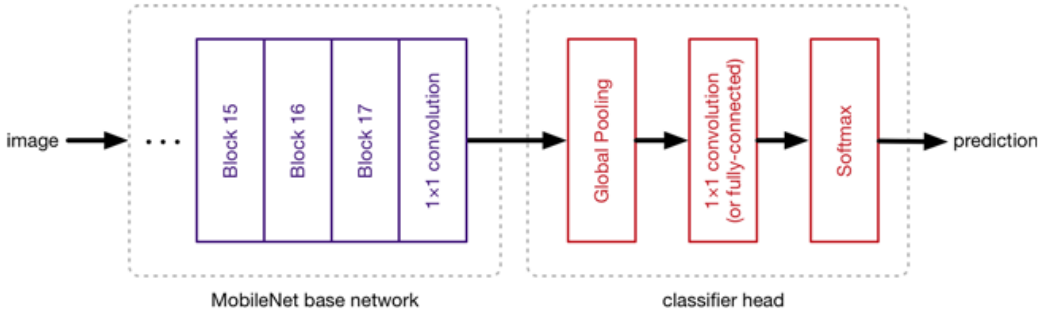


Figure 4.11: Mobilenet Original Architecture

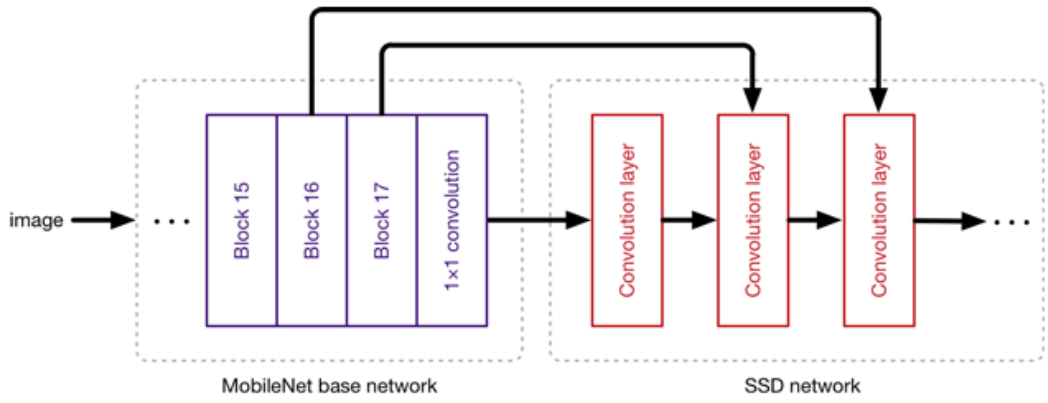


Figure 4.12: Mobilenet SSDLite combined Architecture

Chapter 5

System Software

A system is a set of various elements, which are connected inside a area, and may have abilities of input and output processing.

System on Board (SoB)

An SOB is a whole computer constructed on a single circuit board, with memory, microprocessor, I/O with other features needed for a functional computation.

System on Chip (SoC)

All the features or more are found in SoB along with the microcontroller itself when integrated into a single IC, it becomes a System on Chip (SoC),SoC integrates a lot of system components into a single silicon chip.

5.1 Micro controller vs Micro processors

- Key difference is the presence of external peripheral, where microcontrollers has RAM, ROM, E2PROM embedded in it while we have to use external circuits in case of microprocessors.
- Speed of Processing microcontrollers is 8 MHz -50 MHz, but actually processing speed of microprocessors is more tha 1 GHz , so it works

much faster than many microcontrollers.

- Micro controllers do not have Memory management unit like Micro processsos required by OS.

Since we required high computation power, we started with microprocessors

5.2 Operating System

Operating system acts as an interface between application and hardware . An operating system manages computer hardware and resources of software giving services for computer program.Figure 5.1 shows Operating system Interfaces with User programs and Hardware of the system.

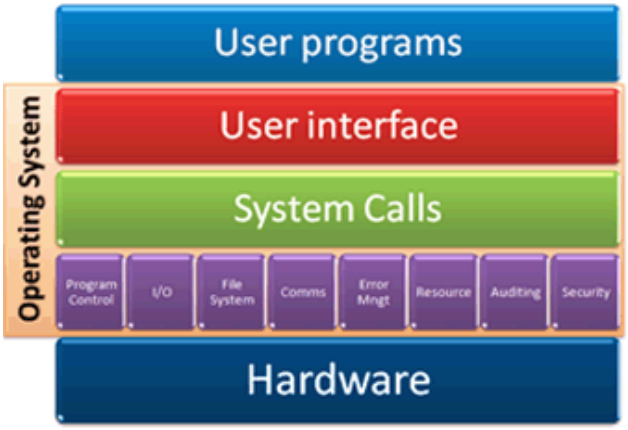


Figure 5.1: Operating System Interface

Components of operating system

5.2.1 Kernel

The kernel is a program that is the core of computer's operating system, with complete control over everything that happens in the program. It handles the start-up along with input/output requests from software, converting them

into data-processing commands for the CPU. It manages memory and other peripherals like Scanner,keyboards, monitors, printers, and speakers.

5.2.2 Bootloader

Bootloader is that code snippet that executes before OS starts running.It boots other OS, every OS has a specific bootloader.

5.2.3 Drivers

Driver allows clear communication between device and OS. Hardware including scanners,cd rom,sd cards, dard drives uses device driver.

5.3 Embedded Linux

Linux itself is a kernel, but 'Linux' in regular use doesn't always mean so. Embedded Linux generally refers to a complete Linux distribution targeted at embedded devices. There is no Linux kernel specifically targeted at embedded devices, the same Linux kernel source code can be built for a wide range of devices, workstations, embedded systems, and desktops though it allows the configuration of a variety of optional features in the kernel itself.
The reasons why we chose Linux OS for the embedded device are,

1. Linux is a free and open source operating system
2. Linux can run on various platforms of CPUs.
3. Linux has a kernel which is very well structured.
4. Linux OS barely crashes unless there occurs any hardware failures.
5. Security aspect of Linux is high.

6. Linux kernel is easy to write to.

7. Linux supports wide range of software.

5.4 ARM Processors

ARM Processor has been used because of the following,

1. ARM processor supports a (Reduced Instruction Set Computer) based architecture.

2. More efficient than any other processor.

3. ARM instruction set allows comanies to build there own system

4. RISC architecture enables ARM to function a very high frequencies.

5. Has higher code density and faster processing speed

6. Kernel is loaded directly from Flash

7. Has debugging capacity present on chip

5.5 Library Porting

We got base OS for our Single Board Computer(hereafter referred as SBC), from our project sponsor company. that contained bootloader+Kernel+Root File system that are packed into single image file. But this OS image was a basic minimal system to run to top of hardware. Hence it didn't have the required software dependencies to run our application. So we needed to install our dependencies first. We fetched the final list of dependencies required from application side as:

Basic: Python3.5, numpy-1.15.4, pillow-5.2.0, pyserial-3.4, wheel-0.31.1, Cython-0.28.5, pip-18.0, lxml-4.2.4, Protobuf 3.0.0

Image processing library: OpenCV-3.4.1

AI framework: Tensorflow -1.9.0

These dependencies are pre-compiled as packages for PC architectures such as x86,x86-64 of Intel, for Linux and Windows OS's both. There's no such default packages available for ARM SoC's/SBC's, hence we had to recreate some libraries on target architecture. This process is called as library porting. There are mainly 6 steps involved in this process.

5.5.1 Dependencies

Finding and installing basic initial basic dependencies for the packages itself.

5.5.2 Fetching source

Downloading and fetching raw source files hosted on package servers.

5.5.3 Configuration

The downloaded source is preconfigured for PC based architecture. Hence, we have to make changes in this source to fir out target architecture i.e aarch64 . The detailed patches added has been explained in implementation chapter in library installation section. This is called configuration of source.

The output of this step is a config file that has all instructions for compiling the source, as weather to omit which part during installation or add another module for board support.

5.5.4 Compile

Start assembling/compiling of this source. There are two types of compilation.

- **Cross compilation:** Compiling of source for platform which is other than the one compilation is running
- **Native compilation:** Compiling of source on same platform on which the compilation is running.

We preferred native compilation as we get to know of errors in real time for easier debugging. This is the most time consuming step of all, as it is initialising all binaries and source files to be ready for installation. For our tensorflow compilation, this step took 14 hours on when using 4 cores of ARM Cortex-A73. For OpenCV it took 6 hours using same 4 cores. As using all 8 cores gave us memory errors and performance throttling. For python packages, this step outputs a "wheel" file. These files help in smooth installation of python modules.

Thus we get our pre-configured wheel for target architecture and OS. Obtaining this wheel does not require to do all 4 steps again for installation of a package next time. "whl" file is the standard built-package format used for python distributions. Hence, our wheel files for tensorflow and opencv were pushed in mainline as OS contributions and will be open sourced.

5.5.5 Install

Finally, we can installing wheel file using pip(Python-Inline package manager).

5.5.6 Testing

Test the working of and initialization of library using fixed set of commands(import opencv, import tensorflow). If we get no errors, then the library porting is completed.

Hence, our system is now made ready with linux OS and installed libraries for running application

Chapter 6

Implementation and Approach

This chapter deals with robotic arm interface , problem of operating system acting as both client and server . Then we look for solution by including drivers like CDC class, ttyUSB and Linux ACM driver then we move on to library porting in Hickey 970 and installation of various modules like python,openCV and tensorflow. Then we move on to the steps in implemetation which involves object detection and data collection and will be explored further.

6.1 Robotic Arm Interfacing

A operating system is designed to provide networking functionality. A network operating system supports client-server networking, and includes the programs needed to manage network resources to create a secure network environment.

6.1.1 CDC Class(Communications device class)

It is a composite Universal Serial Bus device class. The class may include more than one interface, such as a custom control interface or audio, or mass storage related interface.

The CDC is used for computer networking application similar to a network card, providing an interface for transmitting Ethernet or ATM machines onto some physical application. It is also used for modems, land lines, fax machines, and telephony applications for performing voice calls.

6.1.2 ttyUSB

Sometimes, the embedded computer does not come with a hardware USB interface. While it is possible to use a software-only USB , the additional constraints that is put onto the CPU and the small storage size often lead board designers to include a dedicated bridge between UART and USB.
Linux couples devices with similar functionalities under the same general device or interface names. For example, the UARTs present on system will be named /dev/ttyS0 and /dev/ttyS1 even if one of them is a legacy 16550 chip and the other one is a MAX3100SPI-controlled UART.

So, whenever /dev/ttyACM0 pops up, one can send it the escape sequence followed by AT commands, but there is a good chance that the device only pretends to be a modem and will happily send those characters to the core program without even considering cut off them. Hence we needed a driver that would solve all the above mentioned issues and ACM driver was used.

6.1.3 Linux ACM Driver

The drivers/ usb /class/ cdc-acm.c drivers works with USB modems and USB ISDN terminal Adapters that transform into the Universal Serial Bus Communication Device Class Abstract Control Model (USB CDC ACM) specifications. These two drivers were added to the kernel of the operating system Together with Linux acm driver and CDC Class the operating system was able to act as a server and as a client.

6.2 Library Installation

Huawei Hi-key 970 board operating system with Linux kernel doesn't have required dependencies that we require in this project, So first we have to finalized port libraries that can fulfill that dependencies.For example

- **Basic**: Python3 Numpy, pillow, pandas.
- **Image processing library**: Open CV
- **AI framework**: Tensor-flow and its 40 dependencies.

OpenCV, Tensorflow(or any AI libraries) are available as easily be installed by using pip(python-inline-package) manager on x86 systems. But not released for arch64 architecture.General pipeline for library porting:

1. Dependencies: Pre-requisites for the package to be installed
2. Source: Fetching raw files
3. Configure: Add patches for arch64 architecture
4. Compile: Compilation of sources
5. install: Install compiled files

6.2.1 Python Installation

sudo apt-get install python-dev python-pip

pip install numpy pillow pandas matplotlib image-tk

6.2.2 OpenCV Installation

Compiled from source.

Installed dependencies: Neon, gtk, proto-buffers, boost etc.

Building source using cmake : Added flags for Neon, VFVP3, TBB

Compiled using: make , this can take 4-5 hours with -j8 flag

Install: make install

Final post-processing and error resolving: linking path, adding patches

6.2.3 Tensorflow Installation

Compiled from source.

Installed dependencies: openjdk , python-wheel

Download and Install "Bazel": It is a software build system used by Google. Similar to "make".

Modify bazel source file to make it compatible with aarch64.

Compile Bazel and once done copy the binary into our system's bin folder

./compile.sh

sudo cp output/bazel /usr/local/bin/

Download TF source and configure it [1]

Compile: bazel build –flags . This takes 10-15 hours on 8 core

6.3 Application Implementation Steps

In our implementation, we have used Google's mobilenet architecture, TensorFlow's, Opencv as image processing library and python programming language to build and test object detector. Our object detector detects 4 different soaps namely:

1. Dove
2. Moti
3. Medimix
4. Pears

Our project implementation has mainly 7 steps:

1. Data Collection
2. Data Augmentation and labelling
3. Generating Training data
4. Configure training
5. Run the training
6. Exporting inference graph
7. Testing model

6.3.1 Data Collection

Convolutional neural network model required huge amount of data gathering this data is tedious task. Quality and features of collected data directly impact on accuracy of model. For good object detection our dataset contain images in different postures, which different angles, image of desired object which other undesirable objects, images in different lighting conditions with varying background. There are some images in which desired object is partially obscured or only halfway in Figure 6.1.

Figure 6.1: Collection of images

6.3.2 Data Augmentation and labelling

Despite of all of the above available data, the data has to have good diversity as the object of interest needs in varying sizes, lighting conditions if we desire that our network generalize well during the testing phase. We have applied technique of data augmentation to overcome problem of limited quantity and limited diversity of data, we generate our own data with the existing data which we have. Data augmentation include following operations:

Image resizing

Images collected from different sources are of varying sizes. The images fed to neural network will be required of a fixed size to get advantage of processing

data in batches.

Scaling

The most important aspect of image diversity is to have differently scaled object in images. When implemented in real life object size can be of varying to overcome that problem we did scaling.

Translation

To recognize object present in any part of image we do 'Translation' operation on image. Also, the object can be present partially in the corner or edges of the image. So, we shift the object to various parts of the image.

Rotation and Flipping

To make network recognize the object present in any orientation we must pass it as training data. In addition, to remove biasness of assuming certain features of the object is available in only a particular side we flip it.After pictures are gathered, it is time to label desired objects in given images. Before labelling we split our images in folder:

- Training data: 80% of imageset goes in this.
- Testing data: 20% of imageset goes in this.

Labeling

We have used LabelImg software to label images as shown in Figure 6.2. Output of labelling is given as xml file for individual image. This xml file contains label for image objects and their coordinates.

Figure 6.2: Labeling Images

6.3.3 Generating Training data

Tensorflow model take input for training and testing of model as TFRecords files. TFRecords are file stored in binary format. So firstly we converted our xml files to csv file. This created csv is given as input to generate TFRecords file.One of the last step to do before training is to create a label map file according to our requirement. Label map The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers.

```
item {
  id: 1
  name: 'Dove'
}
item {
  id: 2
  name: 'Moti'
```

```
}
item {
  id: 3
  name: 'Medimix'
}
item {
  id: 4
  name: 'Pears'
}
```

Above typed code is stored in .pbtxt file format.

6.3.4 Configure training

Last step is to configure object detection training pipeline. '.config' file, contains many info regarding configuration. We mainly added the number of classes and examples, and added the file paths to the training data.

- input_path:"home:/tensorflow1/models/research/object_detection/test.record"
- label_map_path: "home:/tensorflow1/models/research/object_detection /training/labelmap.pbtxt"

6.3.5 Running the Training

From the /object_detection directory, issue the following command to begin training:
python train.py –logtostderr –train_dir=training/ –pipeline_config_path=training /faster_rcnn_inception_v2_pets.config

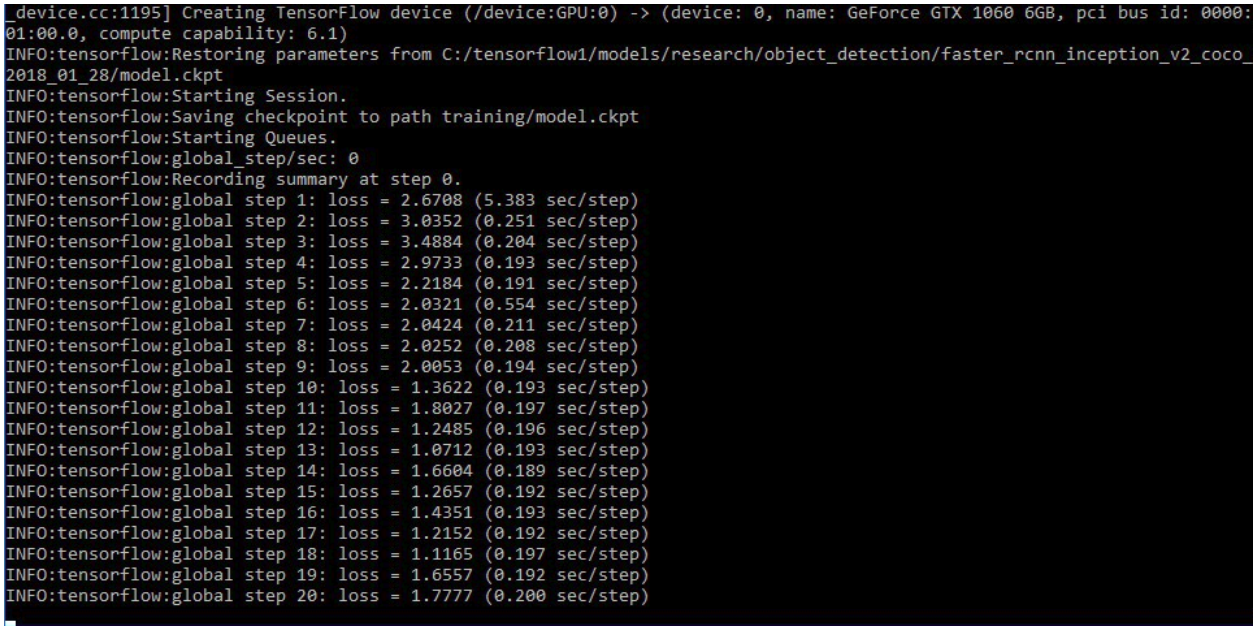


Figure 6.3: Model Training

TensorFlow will initialize the training, it usually takes few seconds for initialization before the actual training begins. Figure 6.3 shows command prompt after training begins. We can also view the progress of the training job by using TensorBoard. To view training process give following command: home:/tensorflow1/models/research/object_detection>tensorboard –logdir =training

One of the important graph is the Loss graph, which shows the overall loss of the classifier over time. The training process periodically saves checkpoints about every five minutes. The checkpoint at the maximum number of steps are used to generate the frozen inference graph.

6.3.6 Exporting Inference Graph

Next step is to generate the frozen inference graph (.pb file). From the /object_detection folder, we issue the following command, where “XXXX” in “model.ckpt-XXXX” is replaced with the highest-numbered .ckpt file in the training folder:

python export_inference_graph.py –input_type image_tensor –pipeline_config _pathtraining/faster_rcnn_inception_v2_pets.config–trained_checkpoint_prefix training/model.ckpt-XXXX–output_directory inference_graph

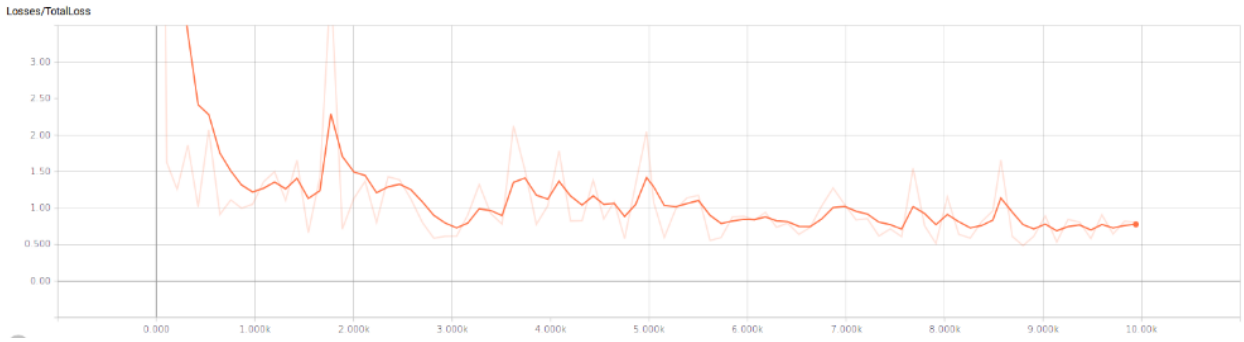


Figure 6.4: Loss as a function of steps

This will create a frozen_inference_graph.pb file in the /object_detection/ inference_graph folder. This .pb file is our object detection classifier.

6.3.7 Testing model

Now our model is ready for testing. We run object_detection_webcam.py file to test our build model. Following snippet shows the output:

Figure 6.5: Testing Model

Chapter 7

Result and Conclusion

In this chapter, we have summarized the results of models used. From starting phase we have made changes in our dataset and tuned our hyper parameters to obtain better accuracy and precision. Different parameters are:

1. Learning Rate
2. Loss function
3. Regularization loss
4. Mean Average Recall
5. Mean Average Precision

7.1 SSD MobileNet Test Results

7.1.1 Learning rate

In Figure 7.1 Y-axis shows learning rate and X-axis shows number of steps. As can be infer from above graph learning rate is relatively high and decreases continuously for better tuning and at end end it is relatively constant.

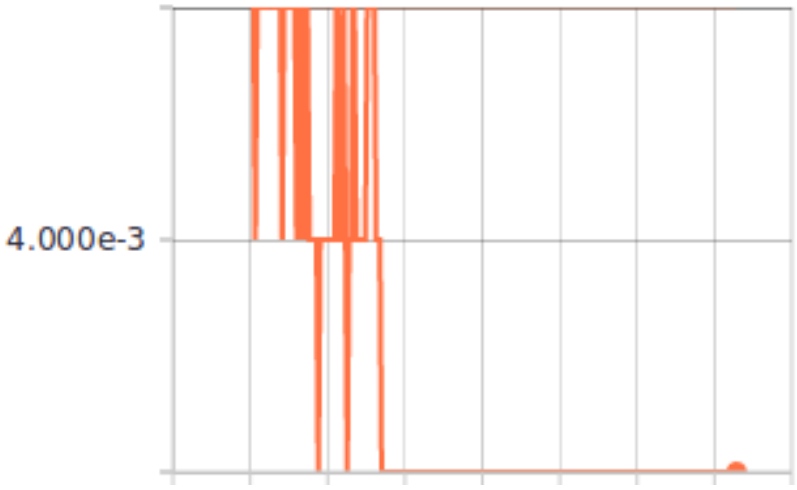


Figure 7.1: Variation in Learning Rates(SSD MobileNet)

7.1.2 Loss function

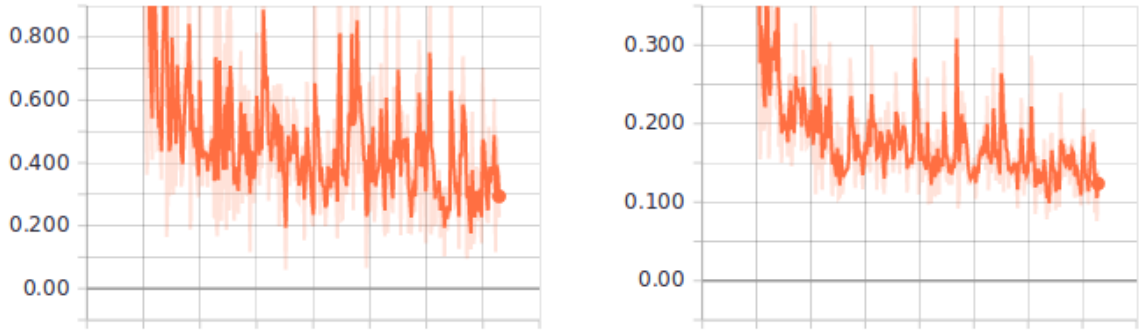


Figure 7.2: Classification and Localization Losses(SSD MobileNet)

In Figure 7.2 Y-axis shows value of loss and X-axis shows number of steps. Our aim is to minimize this loss. Ideally loss value should decrease as we increase number of steps but as we can see fluctuations in loss value which indicate model is not taring well and their is some issue with training data or hyper parameters.

7.1.3 Regularization loss

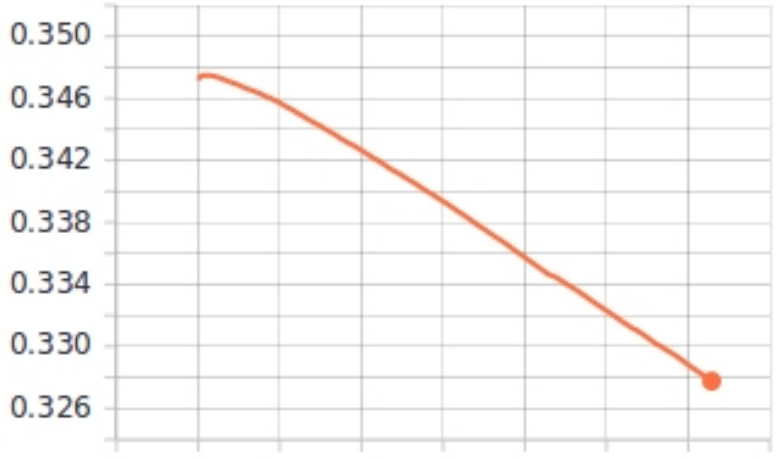


Figure 7.3: Regularization loss(SSD MobileNet)

In Figure 7.3 Y-axis shows value of loss, X-axis shows number of steps. Regularization methods are used to generalize model better, in order to drive the optimization algorithm in desired directions. Ideal aim is to push this loss close to zero.

Figure 7.4: Soap Dataset used in SSD MobileNet

7.1.4 Precision and Recall

Precision

Precision is percentage measure that your predictions are correct.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Precision is percentage measures that how good a model is to find all the positives.

$$Recall = \frac{TP}{TP + FN}$$

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

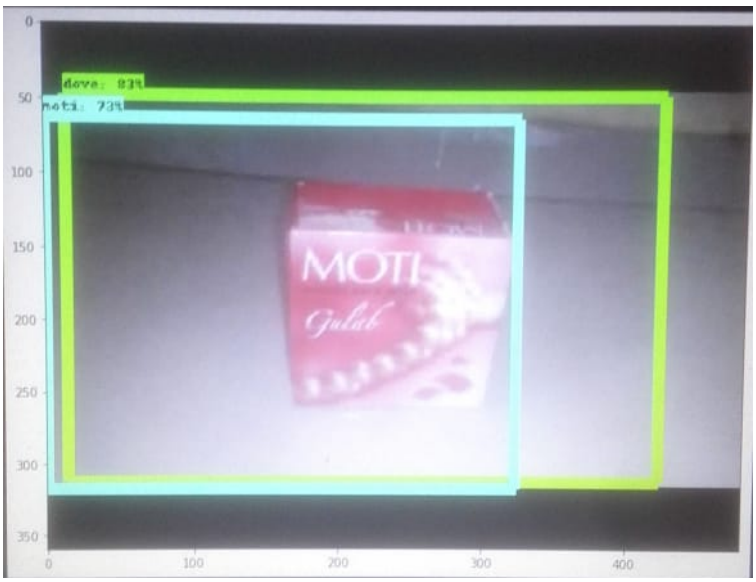


Figure 7.5: Object accuracy at real time in SSD MobileNet

For our case we have four classes so we find out mean average precision(mAP) and mean average recall(mAR):

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)}$$

$$mAR = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FN(c)}$$

Table 7.1 shows the quantity of test classes taken to calculate Mean average Precision and Mean average Recall values.

Table 7.1: Quantity of test classes(SSD MobileNet)

	Dove	**Pears**	**Moti**	**Medimix**
Total Quantity	30	30	30	30
True Positive	19	21	18	20
False Negative	5	4	4	5
False Positive	6	5	8	5

Mean Average Recall = 81.24%

Mean Average Precision = 76.5%

7.1.5 Conclusion from SSD MobileNet

Model was very biased. While collecting training data, in most of the images we kept environment constant that results in model over fitting. Steps for which model is run also effect the performance of model.

7.2 SSDLite MobileNet v2 Test Results

This time we take images in variety of environments,directions and added random noise to them. For all four classes we took exactly same number of images for training. This results in improved performance of SSD MobileNet.

7.2.1 Learning rate

In Figure 7.6 Y-axis shows learning rate and X-axis shows number of steps. Comparing with previous model we can conclude that this model converges at less no of steps compared to previous one.

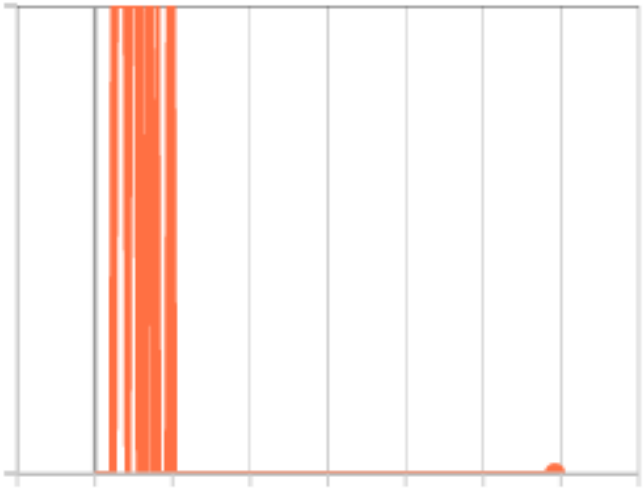


Figure 7.6: Variation in Learning Rates(SSDLite MobileNet v2)

7.2.2 Loss function

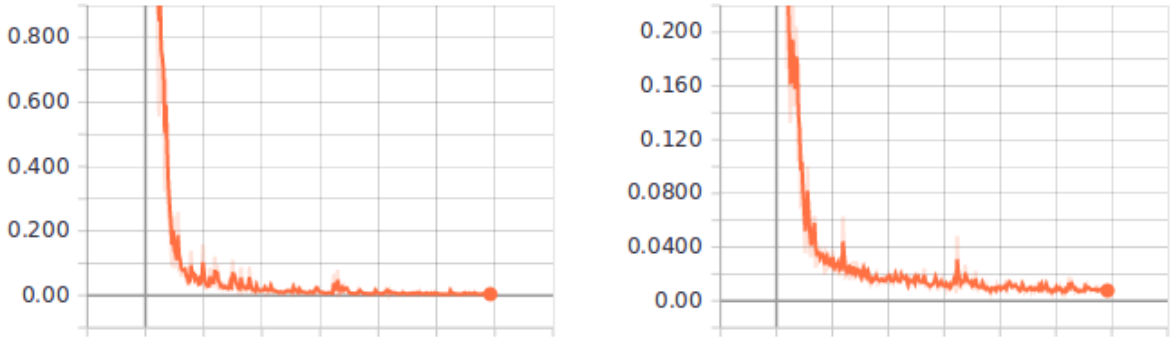


Figure 7.7: Classification and Localization Losses(SSDLite MobileNet v2)

In Figure 7.7 Y-axis shows value of loss and X-axis shows number of steps. It can be conclude from above that model is training relatively smooth. Value of loss function is reducing constantly.

7.2.3 Regularization loss

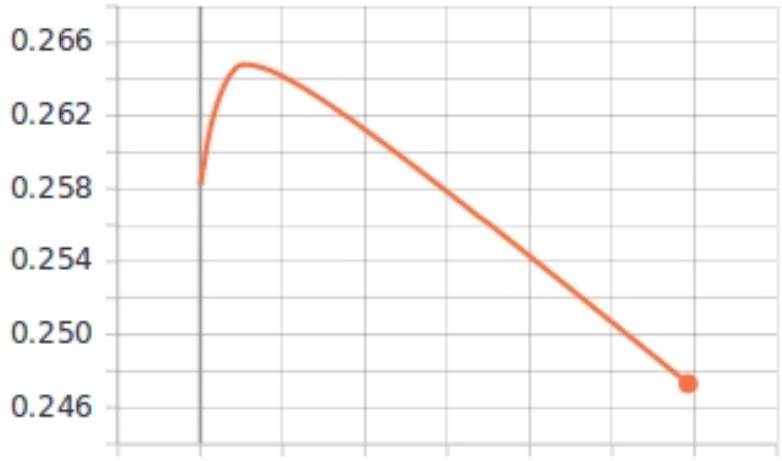


Figure 7.8: Regularization loss(SSDLite MobileNet v2)

In Figure 7.8 Y-axis shows value of loss, X-axis shows number of steps. Compared to previous values model itself start with low initial regularization loss. There is some fluctuations for some steps but after that model trained well to decrease overall regularization loss.

7.2.4 Precision and Recall

Table 7.2 shows the quantity of test classes taken to calculate Mean average Precision and Mean average Recall values.

Table 7.2: Quantity of test classes(SSDLite MobileNet v2)

	Dove	**Pears**	**Moti**	**Medimix**
Total Quantity	30	30	30	30
True Positive	26	24	27	27
False Negative	2	3	1	2
False Positive	2	3	2	1

Mean Average Recall = 93.81%

Mean Average Precision = 92.77%

Figure 7.9: Oject Accuracy at real time in SSDLite MobileNet v2

7.2.5 Conclusion from SSDLite MobileNet v2

From above values we can conclude that accuracy of deep learning model is directly proportional to quality of training data feed into network. Increasing number of training steps up to particular steps results increase in accuracy but after crossing that threshold steps model gets over fitted thus decrease in accuracy.

7.3 Conclusion

Trend in computer vision algorithms were studied. The approach of using deep learning was selected, and most suitable model was customized and implemented according to our use case of object detection. The custom operating system was configured with required dependencies for our target ar-

chitecture ‘aarch64’ on single board computer, our contributions were merged upstream with mainline OS which will be made open source. The application was tested on this platform and gave reasonable real time performance for classification, adequate for conveyor systems with near human level accuracy. The robotic arm was serially interfaced with SBC, for sorting of objects. The operating time of arm was found to be 5 sec on average for every pick and place operation.

7.4 Future Work

Model precision and accuracy can be further improved if we take decision after averaging results of several frames. This will impact our frame per seconds rate but obtained frame per second rate will still good enough for real time applications. Another scope of improvement is that we can define the region of interest for arm. So arm can pick up objects randomly place on conveyer belt. Current system requires huge training data for detection of new objects. This images have to be gathered in various environmental conditions. If we want to deploy system for real life problem solving this method will not be practical one. Our next target will be to develop software which will simulate different environmental conditions on very few images. Current robotic arms work individually, to solve complex real life problems we must be able to implement system of robotic arm who work in collaboration with each other. Our next step will be to connect this arms through IOT gateways to share crucial data and commands with each other. Further the arm can be optimized for the concept of collaborative bots which will add human - machine interface. It will also provide safety to workers working with robotic arms directly.

Bibliography

[1] Tensorflow installation. https://github.com/tensorflow/tensorflow.git, 2016.

[2] Deep learning. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-997 2017.

[3] Image recognition and object detection. https://www.learnopencv.com/image-recognition-and-object-detection-part1/, 2019.

[4] Hikey970 - 96boards. https://www.96boards.org/product/hikey970/, October, 2018. Specification Information.

[5] S. K. Laga J. D. Gavade. Cost effective approach for object sorting. *International Journal of Computer Applications*, 52(16), August 2012.

[6] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large -scale image recognition. 2015. ICLR conference.

[7] Alex Krizhevsky. Imagenet classification with deep convolutional neural. September 2012.

[8] N. Ashcraft E. Weill E. Djerekarov M. Luckow, M. Cook and B. Vorster. Deep learning in the automotive industry: Applications and tools. *2016 IEEE International Conference on Big Data (Big Data), Washington, DC*, 2016.

[9] R. Thorat G. Yeole C. Tank M. Sabnis, V. Thakur. Automatic object sorting using deep learning. *International Journal of Computer Engineering an Applications*, 9, May 2015.

[10] R. Singh Prof. A. Reshamwala. A review on robot arm using haptic technology. 4, April 2015.

[11] Rahul Kher Saurin Sheth. Automatic sorting system using machine vision. January 2010.

[12] B.G. Gawalwad Shubhangi Wanve. Automatic color object sorting system. *International Journal of Modern Trends in Engineering and Research*, July 2015.

[13] Bill Griffith P. Taylor. Meccano magazines. `http://www.nzmeccano.com/MMviewer.php`, March 1938. p172.

[14] Christopher Irgens Klaus-Dieter Thoben Thorsten Wuest, Daniel Weimer. Machine learning in manufacturing: advantages, challenges, and applications, production manufacturing research. June 2016.