HAND GESTURE CONTROLLED ROBOTIC ARM USING NVIDIA JETSON NANO DEVELOPER KIT

A PROJECT REPORT

Submitted by

SREEKAR C **2017504587**

SINDHU V S **2017504582**

BHUVANESHWARAN S **2017504516**

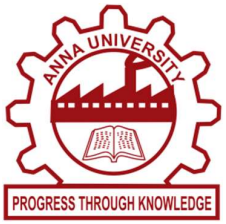
in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING



MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600 044

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2021

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"HAND GESTURE CONTROLLED ROBOTIC ARM USING NVIDIA JETSON NANO DEVELOPER KIT"** is the bonafide work of

SREEKAR C	**2017504587**
SINDHU V S	**2017504582**
BHUVANESHWARAN.S	**2017504516**

who carried out the project work under my supervision.

SIGNATURE

Dr.M.GANESH MADHAN
HEAD OF THE DEPARTMENT
Professor,
Department of Electronics Engg,
Madras Institute of Technology,
Anna University,
Chennai – 600 044.

SIGNATURE

Dr. V. SATHIESH KUMAR
SUPERVISOR
Assistant Professor,
Department of Electronics Engg,
Madras Institute of Technology,
Anna University,
Chennai – 600 044

ACKNOWLEDGEMENT

We consider it as our privilege and our primary duty to express our gratitude and respect to all those who guided and inspired us in the successful completion of the project.

We owe solemn gratitude to **Dr.T.THYAGARAJAN**, Dean, Madras Institute of Technology, for having given consent to carry out the project work at MIT Campus, Anna University.

We wish to express our sincere appreciation and gratitude to **Dr.M.GANESH MADHAN,** Professor and Head of the Department of Electronics Engineering, who has encouraged and motivated us in our endeavours.

We are extremely grateful to our project guide **Dr.V.SATHIESH KUMAR**, Assistant Professor, for his timely and thoughtful guidance and encouragement for the completion of the project.

We sincerely thank our panel staffs **Dr.G.KAVITHA, Mrs.S.ALAGU, Dr.N.ANITHA** for their valuable suggestions. We also thank all the teaching and non-teaching staff members of the Department of Electronics Engineering for their support in all aspects.

SREEKAR C **2017504587**

SINDHU V S **2017504582**

BHUVANESHWARAN.S **2017504516**

ABSTRACT

In tele operation mechanism, the surgical robots are controlled using hand gestures from remote location. The remote location robotic arm control using hand gesture recognition is a challenging computer vision problem. The hand action recognition under complex environment (cluttered background, lighting variation, scale variation etc.) is a difficult and time consuming process. In this paper, a light weight Convolutional Neural Network (CNN) model Single Shot Detector (SSD) Lite MobileNet-V2 is proposed for real-time hand gesture recognition. SSD Lite versions tend to run hand gesture recognition applications on low-power computing devices like Jetson Nano due to its light weight and timely recognition. The model is deployed using a Camera, Jetson Nano and a Raspberry Pi Controller. For the hand gesture recognition and data transfer to the cloud server, the Jetson Nano is used. The Raspberry Pi Controller receives the cloud information and controls the Robotic arm operations. The performance of the proposed model is also compared with a SSD Inception-V2 model for the MITI Hand dataset-II (MITI HD-II). The average precision, average recall and F1-score for SSD Lite MobileNetV2 and SSD Inception-V2 models are analyzed by training and testing the model with the learning rate of 0.0002 using Adam optimizer. SSD MobileNet-V2 model obtained an Average precision of 98.74% and SSD Inception-V2 model as 99.00%. The prediction time for SSD Lite MobileNet-V2 model using Raspberry Pi controller takes only 0.14s whereas 0.21s for SSD Inception-V2 Model. The model is then deployed in highly accurate robotic arm Niryo one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW:

Gesture recognition is a technology that focuses on understanding human gestures through mathematical algorithms. We may use simple gestures to control or communicate with devices without touching them physically. The interaction between the human beings and machines are carried out in more natural forms with the development in the field of Artificial Intelligence. Hand gestures are used to convey a non-verbal type of information's. Gesture-based implementation is associated with a number of fields in many applications, such as Human Machine Interaction, Virtual Reality, Robot Control, Telesurgery systems etc. Gesture recognition can be seen as a way for computers to understand human body language, to create a richer bridge between machines and humans, reducing the majority of keyboard and mouse inputs and communicating directly without any mechanical devices.

A robotic arm is a programmable mechanical arm consisting of motors that allow linear displacement or rotational motion. The robotic arms are widely used in industrial applications from welding, material handling, and thermal spraying, to painting and drilling, in medical applications for performing surgery and also in International Space Station experiments to capture free flying vehicles or modules, supporting extravehicular activity, and aiding in other scientific activities. In teleoperation, the doctors position the surgical robots from a remote location by an efficient Human Machine Interaction system.

In this project, a hand gesture recognition system using a single stage deep convolutional neural network (CNN) to control a robotic arm with 5 degrees of freedom (DOF) in a remote location is proposed. The hand gestures are recognized using a Single Shot Detector (SSD) Lite MobileNet-V2 based CNN model. The

MobileNet-V2 CNN model is used as a feature extractor. The information of the recognized gesture is transmitted using a Wi-Fi module and then to the 5 DOF Robotic arm in the remote location. The SSD Lite MobileNetV2 model predicts the input gestures at a faster rate when compared with the SSD Inception-V2 model.

1.2 ORGANISATION OF THE PROJECT:

The organization of the report is as follows: Chapter II elaborates the state of art related to the proposed work. Chapter III describes the image processing workflow. The hardware description is discussed in Chapter IV and implementation of the project is discussed in Chapter V. Finally, Chapter VI and VII concludes the report with results obtained and conclusion**.**

CHAPTER 2

LITERATURE SURVEY

This chapter reviews the literature proposed by researchers on Hand gesture robotic arm.

Saurabh et al. [1] developed a wireless gesture controlled robotic arm. The author captured the hand gestures using a webcam and recognized the gestures using co-relation technique in MATLAB. Robotic arm consists of three DC motors driven by a motor driver and controlled by a PIC 16F877A Microcontroller. The wireless communication is carried out by using a RF module.

Aggarwal et al. [2] proposed a wireless gesture controlled robotic arm with vision. An accelerometer-based device wirelessly (RF Signals) controls a robotic arm. The robotic arm and platform are coordinated with the operator's hand, leg movements and postures. The system is also fitted with an IP camera capable of transmitting video in real time to any Internet-enabled machine.

Sagayama et al. [3] proposed a hand movement recognition model with two deep learning classifiers: the Deep Belief Network (DBN) and CNN. The author analyzed the performance on hand movements of five distinct classes. For each hand gesture, the accuracy rate of DBN, CNN and HOG+SVM are obtained as 98.35%, 94.74% and 89.62%, respectively.

Howard et al. [4] proposed a MobileNet model for embedded vision applications. It comes with a compact architecture in which the network uses depth-wise separable convolutions. The author also introduced two hyper parameters called the width multiplier and resolution multiplier. These parameters help in resizing the model according to the application. The author illustrated the efficiency of MobileNet through a diverse range of applications like object detection, fine grain classification etc.

Othman et al. [5] utilized a SSD-MobileNet model for the detection of objects. The feature extraction is performed using the MobileNet architecture. The authors also reported few enhancements to the model such as default boxes, multi-scale characteristics and depth wise separable convolution layer added to the framework. It resulted in the improvement of efficiency. On the COCO dataset, the authors reported an overall accuracy of 73.00 %.

Wong et al. [6] proposed a deep convolutional SSD architecture (Tiny SSD) specifically developed to recognize objects in an embedded system in real time. A high performance and tailored non-uniform fire subnet module is integrated into the Tiny SSD. It concentrates on the supplementary convolution layers, designed specifically to reduce the effect of model dimension by retaining the detection performance. On the VOC 2007 dataset, the author reported an average accuracy of 61.3 %.

Rubin et al. [7] proposed an Inception-V2 based single stage detector (SSD Inception-V2) model for real-time hand action recognition. The author trained the model using custom-developed hand dataset (MITI-Hand Dataset (MITIHD)). The model is trained for ten classes. The author reported an Average Precision of 99.00 % and prediction time of 46 ms using a NVIDIA TitanX GPU.

Based on the extensive literature survey it has been identified that the real-time prediction time for the detection of hand gestures could be further reduced. In this paper, a 5 DOF robotic arm is positioned using the vision based real-time hand gestures. The real-time hand gesture recognition is implemented using a single stage CNN model (SSD Lite MobileNet-V2). SSD Lite is a lighter version of the conventional Single Shot Detector (SSD). MobileNet-V2 is a CNN algorithm used for feature extraction. The results of the SSD Lite MobileNet-V2 model is compared with the SSD Inception-V2 model. The computational time of SSD Lite model is comparatively lesser than the existing hand action recognition models.

CHAPTER 3

IMAGE PROCESSING WORKFLOW

3.1 CAPTURING THE IMAGE:

In order to develop a hand gesture based recognition system using computer vision, the most important operation is to acquire the image of the gesture in real-time. Since we are going to perform numerical operations on the image to recognize the gesture, we need digital images instead of analog images.

A typical digital camera includes a lens, an image sensor, support electronics, and may also include one or even two microphones for sound. The sensors are placed as 2D matrices on the imaging plane of the camera. These sensors pick up the light radiation of three different wavelengths (Red, Blue, Green) falling on them and produce electric signals due to the photoelectric effect. The signals are processed to obtain the 2D image matrix.

Image sensors can be CMOS (Complementary metal–oxide–semiconductor) or CCD (charge-coupled device) but CMOS sensors are the most dominant one. The two most important parameters to be considered in a digital camera are the image resolution and the frame rate. The image resolution gives information on the number of pixels available per square inch. A high-resolution image preserves much of the original details present in the actual surroundings captured by the camera. However, processing a high-resolution image requires much more computational power and consumes more time. Therefore, the image resolution should be kept as low as possible.

The frame rate is the number of images captured by the camera in one second. The frame rate should be kept as high as possible as this enables the device to take more decisions per second, providing a much smoother control of the system.

It is also required to have the shutter speed as fast as possible to prevent the motion blur of the image.

A single board computer system can acquire digital images in real time in two separate ways. One method is to use a camera module with a CSI (Camera Serial Interface), such as the Pi camera, and the other method is to use a standard USB web camera.

Figure 3.1 USB Web

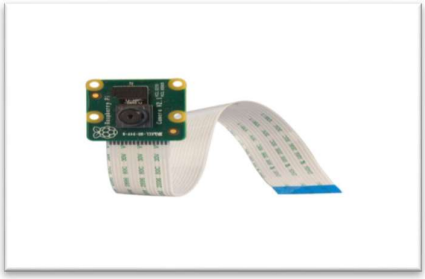


Figure 3.2 Pi camera

The image can be obtained in real-time using Python's OpenCV (Open Source Computer Vision) module by creating a VideoCapture() object.

3.2 REGION OF INTEREST:

The image acquired by the camera will have a lot of unwanted details (trees, lights, fans and various backgrounds etc.,). These details don't contribute much to the decision-making process and thus, they can be ignored. This is done by obtaining what is called the Region of Interest(ROI). Cropping a subset of an image from the original image yields a ROI. After that, the sub-image is used for further processing. The lower and upper limits of the row and column values of the original 2D image matrix can be used to construct a rectangular ROI.

Figure 3.3 Obtaining the ROI

3.3 TRAINING DATA

Data is required to train a neural network. There are two types of data: Training data and testing data. By updating the model's parameters, training data is used to reduce the prediction error (loss). Testing data is used to ensure that a trained model is correct. Data from the testing should not be used to train the model. It's the information that the model has never seen before.

An input (image, text, audio, etc.) and a target value make up the training data. The target value may be an image, an integer, or a decimal number, depending on our task. The training data in our scenario consists of an image and a target value. The image is resized based on the neural network architecture used.

In our multi-class classification problem, the target value is gesture recognized. Examples of training data for classification and regression problems are shown below.

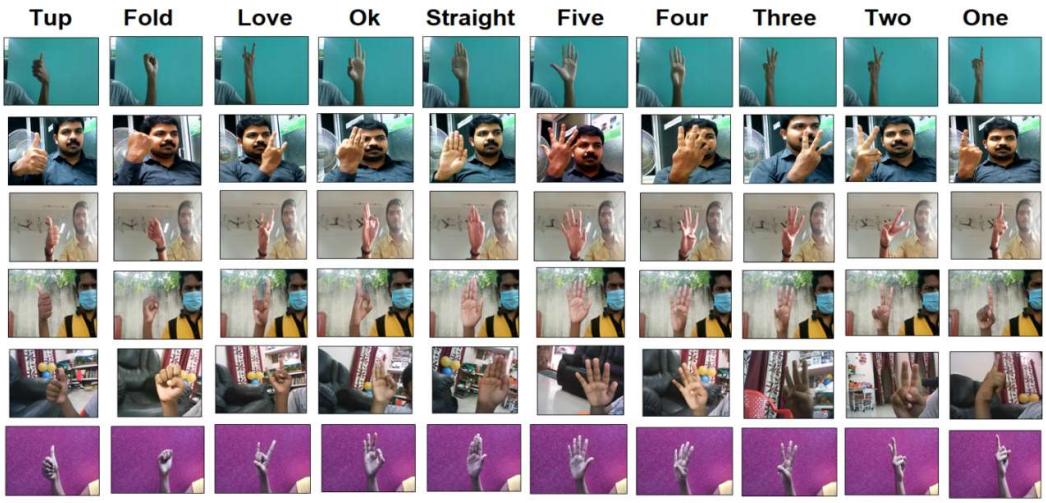


Figure 3.4 Few samples of MITI – HD Dataset

These CNN models are trained and tested using a MITI Hand Dataset–II (MITI HD-II). MITI HD-II is an improved version of MITI HD [7], [9] dataset. It is a custom-created dataset of hand posture obtained from different individuals with differing skin tones, complex profiles, different hand size, conditions of lighting, geometry, fast movements, and different age classes. To provide accurate results, the training data should be as large and diverse as possible. There are ten classes and about 970 samples per class.

3.4 NEURAL NETWORKS:

Neural Networks are function approximators. They are multi-layer networks of neurons classify things, make predictions, etc. It processes a multidimensional input and provides an output. Every input is a point in a multidimensional space and depending on the type of problem (Classification, Regression, etc.,) the neural network alters the multidimensional space.

A simple neural network consists of just the following components:

- A connection with a weight that transforms the input and gives it to the neuron.
- A neuron that includes a bias term and an activation function.

More complex neural networks are just models with more hidden layers and that means more neurons and more connections between neurons. And this more complex web of connections (and weights and biases) is what allows the neural network to learn the complicated relationships hidden in the data.

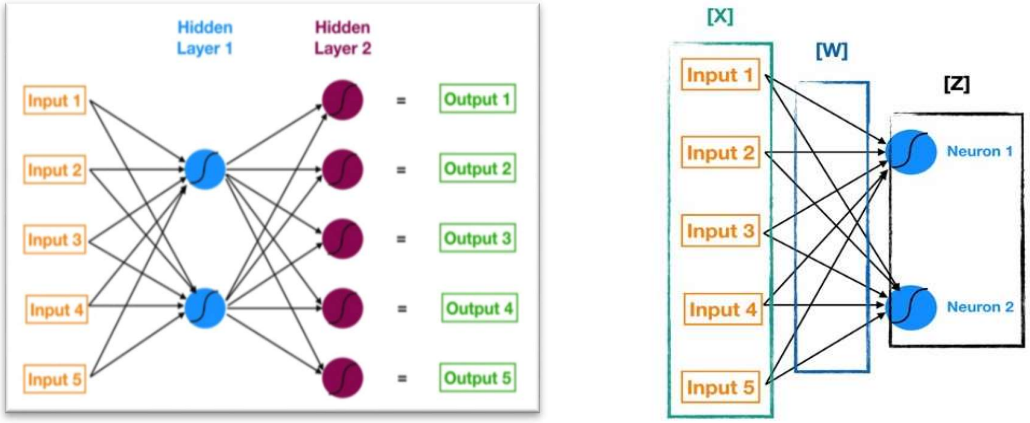
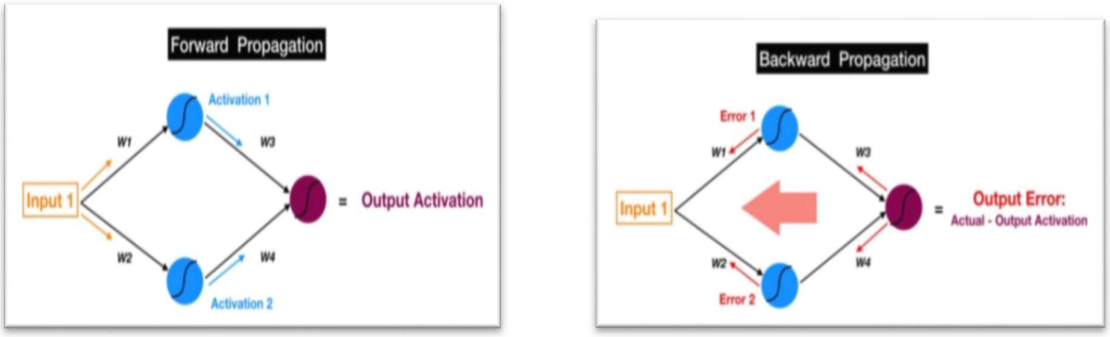


Figure 3.5 A simple example of a neural network with weights

Here, 'W' is **n by m** matrix of weights (the connections between the prior layer and the current layer),'X' is **m by 1** matrix of either starting inputs or activations from the prior layer, 'Bias' is **n by 1** matrix of neuron biases, and 'Z' is **n by 1** matrix of intermediates. By repeatedly calculating 'Z' and applying the activation function to it for each successive layer, we can move from input to output. This process is known as forward propagation. In a neural network, changing the weight of any one connection (or the bias of a neuron) has a reverberating effect across all the other neurons and their activations in the subsequent layers. The training process of a neural network, at a high level, is to define a cost function and use gradient descent optimization to minimize it.

The objective of forward propagation is to calculate the activations at each neuron for each successive hidden layer until we arrive at the output. The objective of back propagation is to calculate the error attributable to each neuron starting from the layer closest to the output all the way back to the starting layer of our model. The

magnitude of the error of a specific neuron (relative to the errors of all the other neurons) is directly proportional to the impact of that neuron's output (activation) on our cost function. So back propagation allows us to calculate the error attributable to each neuron and that in turn allows us to calculate the partial derivatives and ultimately the gradient so that we can utilize gradient descent.



a) **b)**

Figure 3.6 Propogation in neural networks a)Forward b) backward

3.5 CONVOLUTIONAL NEURAL NETWORKS:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take an input image and assign learnable weights and biases to different aspects/objects in the image, allowing it to distinguish from one another. As compared to other classification algorithms, the amount of pre-processing needed by a ConvNet is substantially less. Through the application of relevant filters, a ConvNet can successfully capture the Spatial and Temporal dependencies in an image. Owing to the reduced number of parameters involved and the reusability of weights, the architecture performs better fitting to the image dataset.

The ConvNet's task is to reduce the images into a format that is easier to process while preserving features that are essential for obtaining a good prediction. This is critical when designing an architecture that is capable of learning features while still being scalable to large datasets. The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need

not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well.

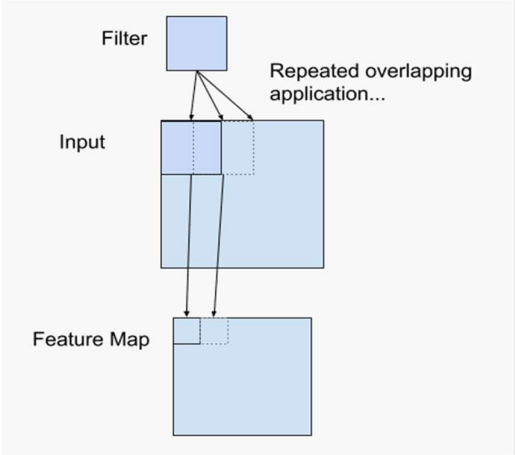


Figure 3.7 Filters in CNN

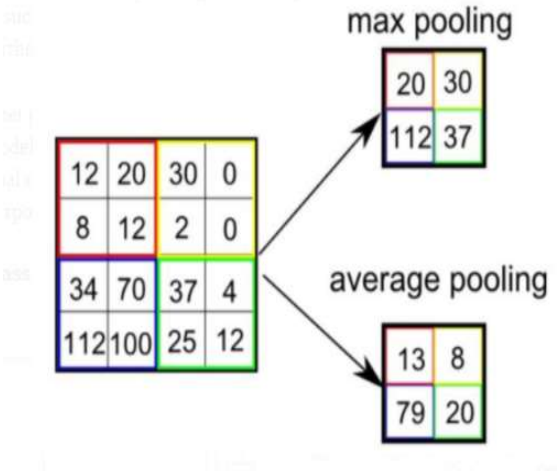


Figure 3.8 Types of pooling

The Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling.

I. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

II. Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism.

Hence, we can say that Max Pooling performs a lot better than Average Pooling.

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and back propagation is applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

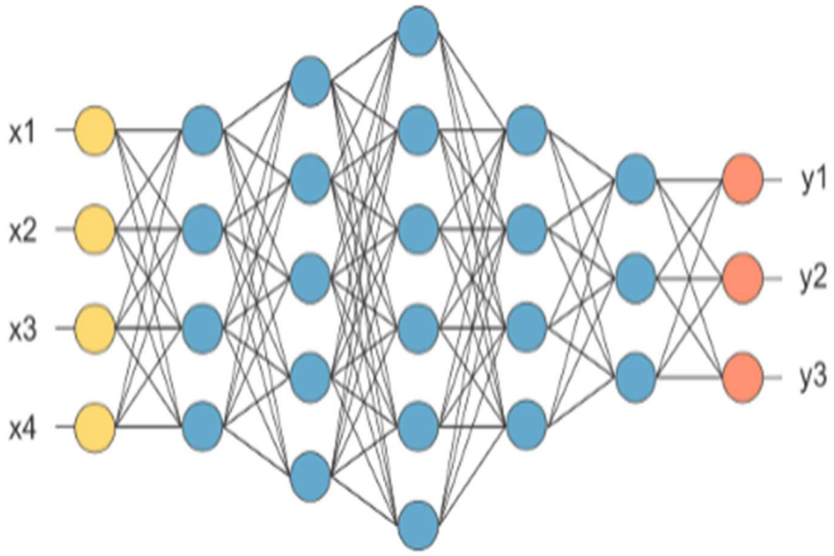


Figure 3.9 An example of fully connected layer

3.6 NEURAL NETWORK ARCHITECTURE:

The hand gesture recognition is performed using the single stage deep CNN. SSD Lite MobileNet-V2 [8] model and SSD Inception-V2 model [7] are used in this framework for gesture recognition. MobileNet-V2 and Inception-V2 CNN models are used as feature extractor. The SSD is a CNN based object detector. It uses a single forward pass network and a bounding box regression technique to classify and localize the object. Over Faster R- CNN, SSD offers significant speed gains. SSD is also used in real-time for action recognition. SSD completes the process of region proposal and classification in a single shot process. SSD Lite [6] model is the lighter version of the conventional SSD model. SSD Lite MobileNet-V2 is 20 times more effective and 10 times lighter than YOLO-V2. It outperforms YOLO-V2 architecture.

MobileNet is a compact architecture that constructs a lightweight deep convolutional neural network using deeply separable convolutions. Depth-separable convolution filters consist of filters for deep convolution and filters for point convolution. On each input channel, the depthwise convolution filter performs a single convolution, and the point convolution filter combines linearly, the 1 to 1 convolutions and the output of depthwise convolution. MobileNets concentrate on latency optimization, but also generate small networks. The depth-wise seperable convolution has separate layers for filtering and combining which drastically contributes in reducing the computation time and model size compared to a standard convolution process. It offers a computation reduction by a factor of $1/N + 1/Dk2$, where N is the number of output channels and Dk is kernel size. This is 8 to 9 times lesser compared to a standard convolution.

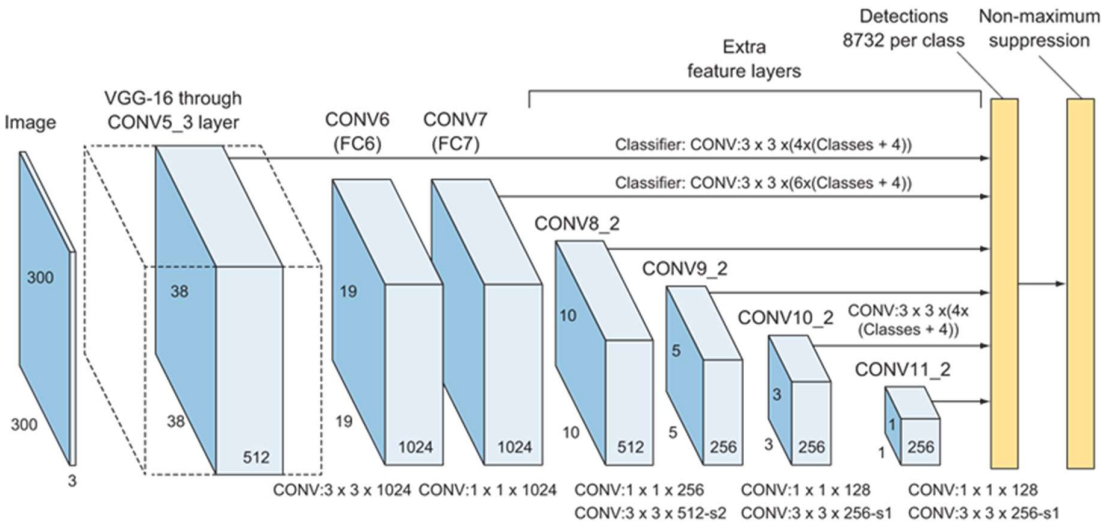


Figure 3.10 SSD Architecture

Numerous CNN models are tested in object detection problems. Some of them are Region-based convolutional neural network (R-CNN), You Only Look Once (YOLO), and Single Shot Detector (SSD). In this paper SSD Lite MobileNet-V2 model and SSD Inception-V2 model are used for Hand Gesture Recognition. The performances of the models are evaluated and compared with each other.

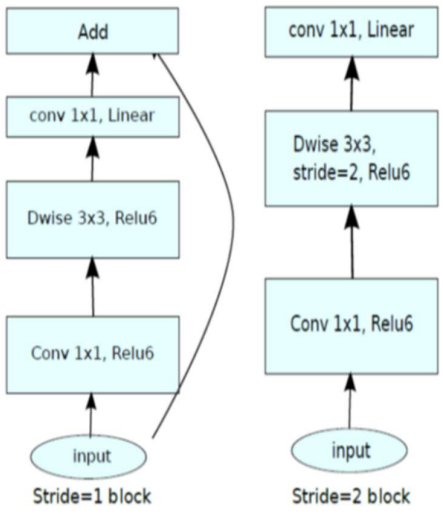
Inception-V2 is faster and computationally efficient than other similar architectures like VGG Net and Alex Net. The reduction of dimensionality is carried out effectively in the convolutions of the Inception-V2 model by integrating a smart factorization process. Inception-V2 consists of three layers. The first layer consists of three convolutional layers with dimensionality reduction filter bank. The second layer consists of five layers with deeper filter banks and last layer consists of two wider filter banks. It is then connected to fully connected layers. Out of these two CNN models, SSD Lite MobileNet-V2 model is highly preferred for remotely operated robots because of its light weight. This improves the speed of Hand gesture recognition.

The overall training objective loss function of SSD [7], [11] is calculated based on the weighted sum of the confidence loss Lcnf and the localisation loss Llcl. It is given by,

$$L(a,n,p,t) = 1\ K\ (L_{cnf}(a,n) + \rho L_{lcl}\ (a,p,t)) \ \ldots\ldots(1)$$

where ρ is the weight term and by cross-validation it is set to 1, K is the number of matched boxes. The loss is said to be 0 if the number of matched boxes default to zero. The term a is the priors, it is 1 when it matches the ground truth box and 0 otherwise. The term n is the number of classes, p is the predicted bounding box parameters and t is the ground truth bounding box parameter.

The trained models are tested using the test samples and their performances are evaluated by the parameters such as Average Precision, Average Recall, F1 score and Prediction time for various intersection over union (IoU) values. IoU determines the accuracy of hand action detection. IoU is unity when the predicted bounding box exactly overlaps the groundtruth bounding box. The prediction is considered to be true as long as the IoU is 0.5 and for the larger value of IoU's, the prediction is accurate.



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 3.11 MobileNetV2 Overall Architecture

It also introduces two global hyper parameters, the width multiplier for thinning the number of channels and the resolution multiplier for scaling the size of the input image.

3.7 TRAINING A NEURAL NETWORK:

A batch consists of multiple blocks of data points, usually in powers of 2.
We train a neural network by using two steps for every batch of data, feedforward, and backpropagation.

In the feed forward process, a batch of data is fed into a neural network with random weights and predictions are made. In the backpropagation process, the predicted values are compared with the target values and the error is calculated as the difference between the predicted values and the target values.

This error is minimized by finding the relationship between the weights of the neural network and the error value through chain rule of partial derivatives and modifying the weights accordingly to reduce the error. This method is known as the gradient descent technique.

Consider a function f(x,y)= x*y. The gradient is calculated as:

$$\partial f/\partial x=y,\ \partial f/\partial y=x$$

This simply means that if y is positive, f increases when x moves in the +ve direction and if y is negative, f increases when x moves in the negative direction. The same applies to y. Thus, to decrease the f, we have to move in the opposite direction ($-\partial f/\partial x$ and $-\partial f/\partial y$).

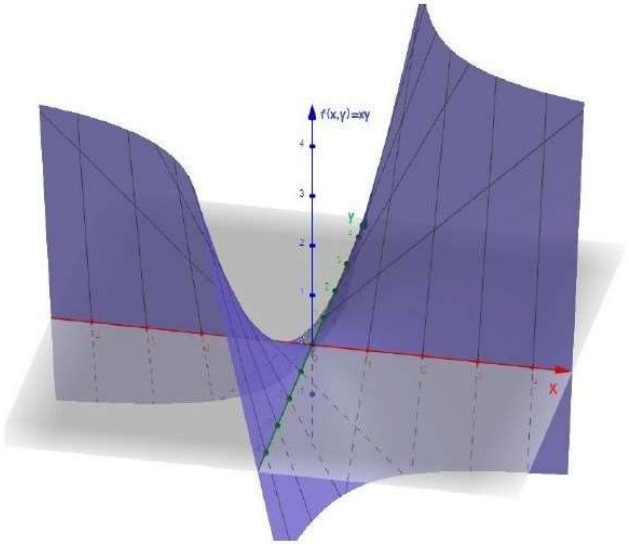


Figure 3.12 Parameter space of the function f(x,y)=xy.

The beauty of gradient descent is that it applies to every mathematical function as all we have to do is simply calculate the partial derivative of a function. For neural networks with multiple layers, chain derivative rule is used. It produces different results for different initial weights.

It is similar to rolling a ball down a slope. The ball gets settled down in the nearest valley called local minima. This local minima is not the absolute minimum. Therefore to overcome this, many different techniques are used.

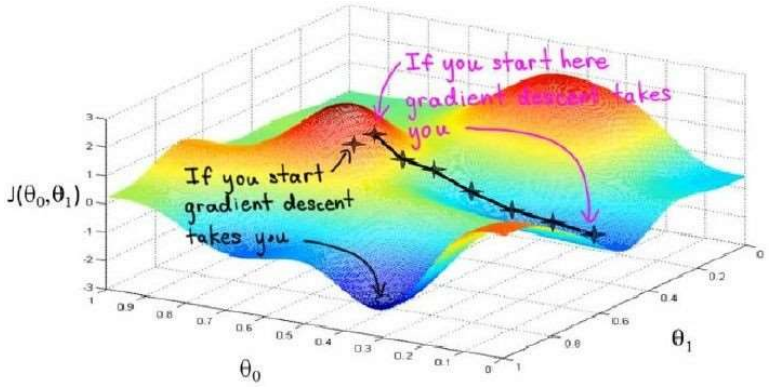


Figure 3.13 Local Minima of a neural network.

3.8 GPU SUPPORT:

A neural network consists of multiple neurons. A single neuron looks as shown below.

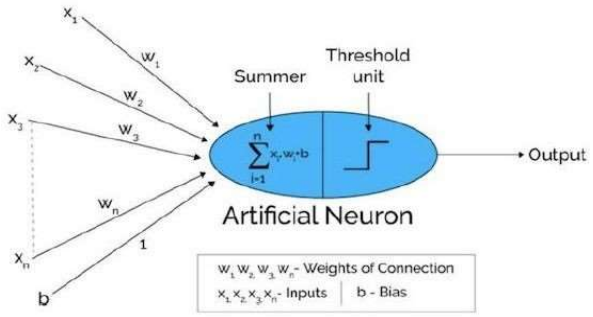


Figure 3.14 An artificial neuron

The inputs $X=x_1,x_2,x_3,.. ..... ,x_n$ are multiplied with the weights $W=w_1,w_2,w_3,….,w_n$ respectively and added with a bias b. Therefore,

$$output = w_1x_1+w_2x_2+w_3x_3+……..+w_nx_n+b$$

The operation could be achieved by treating the inputs, weights and the bias as vectors and performing matrice multiplication, y=X.W+b. The value y is further processed by an activation function which is necessary to produce nonlinearity in a neural network. Thus, neural network operations can be carried out as matrice operations.

Matrice operations can be accelerated through parallel computing. GPUs or Graphic Processing Units are hardware which is specialised for parallel computing operations. They consist of several cores each of which parallelly perform basic arithmetic operations like addition and multiplication.

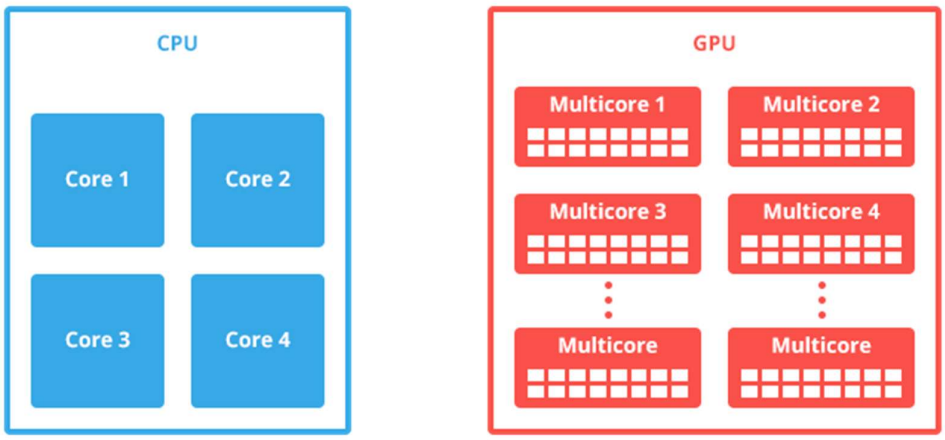


Figure 3.15 CPU vs GPU

NVIDIA organisation provides a special GPU software called CUDA which accelerates deep learning operations. Popular deep learning libraries like TensorFlow and pytorch use CUDA to speed up the training and testing processes.

A GPU can be more than 20 times faster in performing a feed-forward or backpropagation step compared to a CPU based on its capacity. NVIDIA's Jetson Nano is a single board computer which has GPU cores and cuda support. This makes it one of the best options available currently to develop mini AI prototypes.

Figure 3.16 NVIDIA Jetson Nano's processing core

Thus, in this scenario we are using TensorFlow-gpu optimized for NVIDIA Jetson Nano.

One of the biggest problems of neural networks is overfitting. Overfitting is the scenario where a neural network achieves 100% accuracy while training but has very low testing accuracy. The popular analogy used to describe this scenario is a person memorizing every question in the book and answering all the questions from it but fails miserably if a question is asked out of the book. The opposite of overfitting is underfitting where the model cannot converge no matter how long it is allowed to train.

Figure 3.17 The different scenarios in training a model.

Underfitting can be solved by using more complex architectures with more layers. However, overfitting is difficult to solve. One way is to use a huge amount of data with high diversity. Another method is to use dropouts.

One of the reasons for overfitting is the constant updating of the weights of only certain neurons in the neural network in comparison to the other neurons as shown below. To prevent this, the concept of Dropout is used, which de-activates some neurons in the network randomly during each feed-forward process.

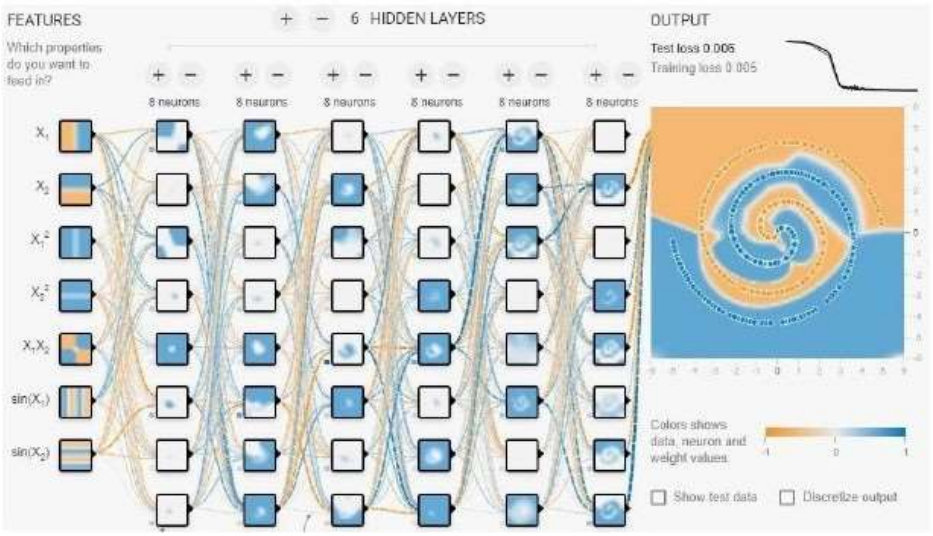


Figure 3.18 An overfitted model with unused neurons.

Dropout is active only during the training process. p is the probability of a neuron getting dropped out during the feed-forward process. The default value is 0.5. Dropout should take place only during the training process and should be deactivated during the testing process.

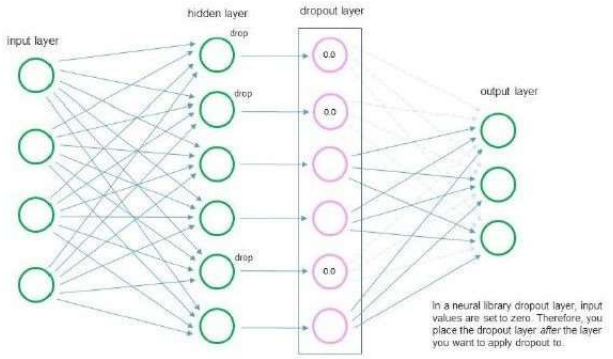


Figure 3.19 Random deactivation of neurons using dropout.

3.9 SAVING A TRAINED MODEL:

Once a model is trained enough to make predictions with desired accuracy, it

can be used in the future. It is necessary to save a model as it is not feasible to train a model every time before use. A model is saved as a hdf5 file or PB file. Tensorflow's save() function can be used to save a trained model. It is possible to prevent overfitting by saving the model at various instances of time using a process called checkpointing.

Validation data is used to prevent overfitting. The model does not train on the validation data to update its parameters. Validation data is used to prevent this by measuring the error the model produces on it and saving the model every time the error is lesser than the previous error. This is known as checkpointing. Thus, when the error starts to increase, the model won't be saved in the device.

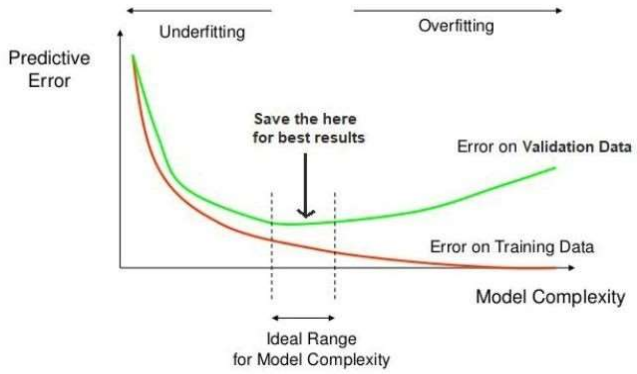


Figure 3.20 Change in validation accuracy over time.

3.10 TESTING THE TRAINED MODEL:

Once the model has been trained and saved, it can be loaded into a program to perform predictions. The saved hdf5 is loaded back using tensorflow's load() function. In testing, only feed forward takes place and the model's parameters won't get updated. Also, dropouts are deactivated while testing. Therefore, testing is less computationally complex compared to training a model. This is the final step of the Image Processing Workflow.

CHAPTER 4

HARDWARE DESCRIPTION

In order to build the prototype, several components are required which are assembled together. This chapter gives a description of each of the components used.

4.1 JETSON NANO DEVELOPER KIT:

NVIDIA Jetson Nana Developer Kit is a small, powerful single board computer that can run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. The Jetson Nano has a 64-bit quad-core Arm Cortex-A57 CPU running at 1.43GHz alongside a NVIDIA Maxwell GPU with 128 CUDA cores capable of 472 GFLOPs (Giga Floating Point Operations per Second), and has 4GB of 64-bit LPDDR4 RAM onboard along with 16GB of eMMC storage. It has 4x USB 3.0, USB 2.0 Micro-B port and 40 GPIO pins. The board can be connected to Wi-Fi through an external Wi-Fi chip. The official operating system for the Jetson Nano is called Linux4Tegra, which is a version of Ubuntu 18.04 that's designed to run on Nvidia's hardware. The Kit is an AI computer that brings the power of modern artificial intelligence to a low-power, easy to-use platform. Jetson Nano is supported by the comprehensive NVIDIA Jetpack SDK, and has the performance and capabilities needed to run modern AI workloads. JetPack includes:

- Full desktop Linux with NVIDIA drivers
- AI and Computer Vision libraries and APIs
- Developer tools
- Documentation and sample code

Jetson Nano Developer Kit requires a 5V power supply capable of supplying 2A current.

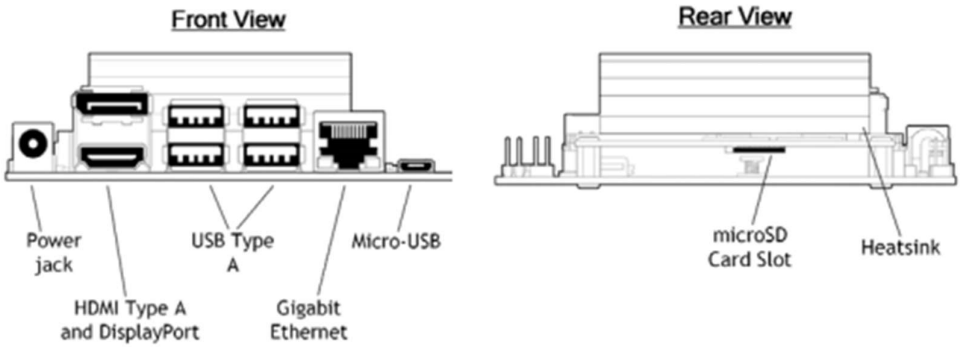


Figure 4.1 NVIDIA Jetson Nano Developer Kit

4.2 RASPBERRY PI 3B:

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. The latest Raspberry Pi 3 Model B+ has a faster 64-bit 1.4GHz quad core processor, 1GB of RAM, faster dual-band 802.11 b/g/n/ac wireless LAN, Bluetooth 4.2, and significantly faster 300Mbit/s ethernet.

The specifications include:

- 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU, 1GB RAM (LPDDR2 SDRAM), On-board wireless LAN - dual-band 802.11 b/g/n/ac
- On-board Bluetooth , 4 x USB 2.0 ports, 300Mbit/s Ethernet, 40 GPIO pins, Full size HDMI 1.3a port, Display interface (DSI), Micro-SD slot
- Combined 3.5mm audio and video jack, Camera interface (CSI)

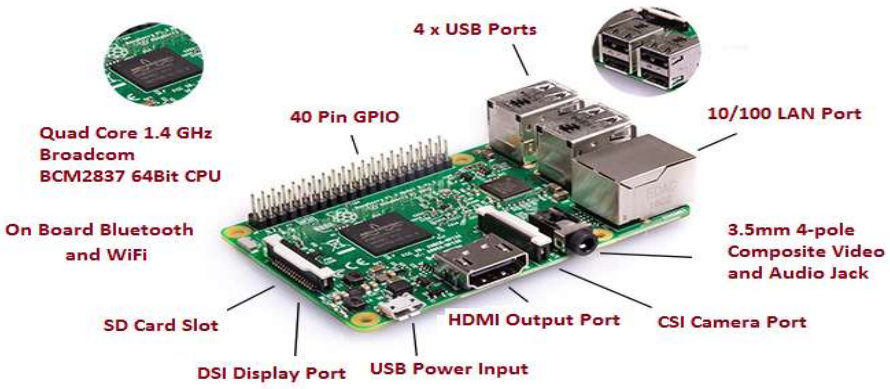


Figure 4.2 Raspberry Pi 3b

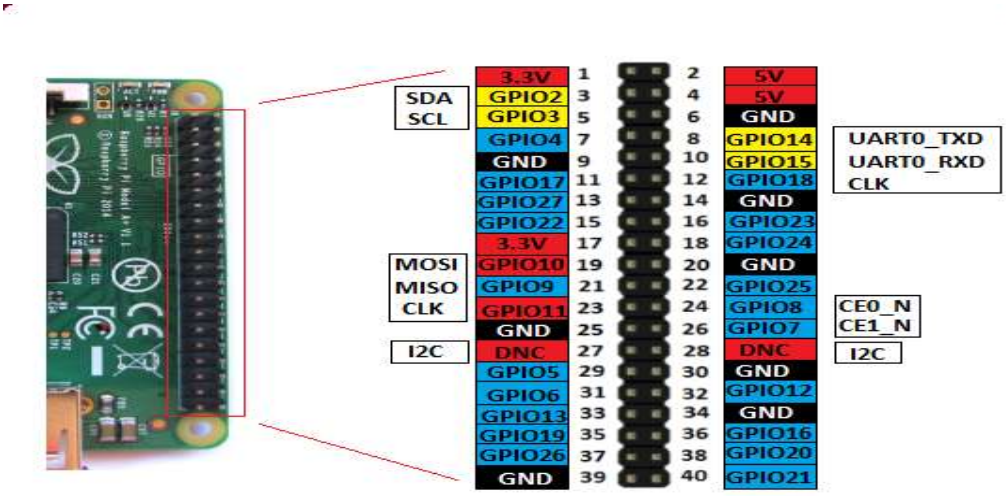


Figure 4.3 Raspberry Pi 3b Configuration

4.3 PCA9685 SERVO DRIVER:

The PCA9685 is a 16-channel I2C-bus controlled LED controller optimized for Red/Green/Blue/Amber (RGBA) color backlighting applications. Each LED output has individual 12-bit resolution (4096 steps) PWM controller with a fixed frequency. The controller operates at a programmable frequency from a typical 24 Hz to 1526 Hz with a duty cycle that is adjustable from 0% to 100% so the LED can be set to output a specific brightness. All outputs are set to the same PWM frequency.

With the PCA9685 as the master chip, the 16-channel 12-bit PWM Servo Driver only needs 2 pins to control 16 servos, thus greatly reducing the occupant I/O's. Moreover, it can be connected to 62 driver boards at most in a cascade way, which means it will be able to control 992 servos in total.

The PWM signal captured by the receiving channel is transmitted to the signal demodulation circuit, and a DC offset voltage is generated. Next, this voltage will be compared with the potentiometer's voltage, and then the voltage drop between them will be input into the motor driving integrated circuit to make the motor rotate clockwise or counter-clockwise. When the rotating speed reaches a certain value, it will drive the potentiometer R0 to spin by the cascaded gear reducer. The motor would not stop rotating until the voltage drop decreases to 0. The servo is controlled by PWM signal, i.e., the changed duty cycle decides where the servo rotates to.

It has following features:

- Contains an I2C-controlled PWM driver with a built-in clock
- 5V compliant, which can be controlled from a 3.3V microcontroller, which is good when it is required to control white or blue LEDs with a 3.4V+ forward voltage
- Supports using only two pins to control 16 free-running PWM outputs
    i. 3 pin connectors in 4 groups
    ii. 5. 12-bit resolution for each output - for servos, that means about 4us resolution at an update rate of 60Hz

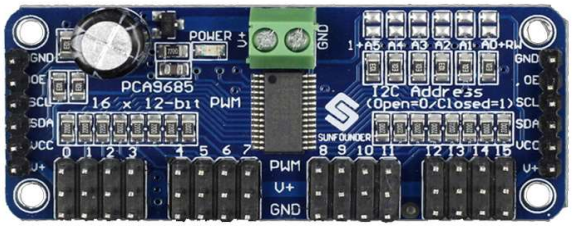


Figure 4.4 PCA9685 Servo Driver

4.4 WEB CAMERA:

A webcam is a video camera that feeds or streams an image or video in real-time to or through a computer to a computer network, such as the Internet. Image sensors can be CMOS or CCD, the former being dominant for low-cost cameras. Webcams are capable of providing VGA-resolution video at a frame rate of 30 frames per second.

Various lenses are available, the most common in consumer-grade webcams being a plastic lens that can be manually moved in and out to focus the camera. As a camera system's depth of field is greater for small image formats and is greater for lenses with a large f-number (small aperture), the systems used in webcams have a sufficiently large depth of field that the use of a fixed-focus lens does not impact image sharpness to a great extent.

Digital video streams are represented by huge amounts of data, burdening its transmission (from the image sensor, where the data is continuously created) and storage alike. The webcams come with built-it ASIC to do video compression in real-time.

Support electronics read the image from the sensor and transmit it to the host computer. Typically, each frame is transmitted uncompressed in RGB or YUV or compressed as JPEG. Some cameras, such as mobile-phone cameras, use a CMOS sensor with supporting electronics "on die", i.e. the sensor and the support electronics are built on a single silicon chip to save space and manufacturing costs. Most webcams feature built-in microphones to make video calling and videoconferencing more convenient.

Figure 4.5 Web Camera

4.5 ROBOTIC ARM:

4.5.1 STANDARD ROBOTIC ARM:

The robotic arm has five degrees of freedom and has five MG995 servo motors with a gripper. **MG995** is a **servo motor** that is popular for its performance and low price. The motor is used in many applications mainly being robotics and drones. **MG995** has three terminals as mentioned in pin diagram and the function of each pin is given below.

Table 4.1 MG995 Pin description

Pin	Name	Function
1	Signal pin (Orange pin)	The PWM signal which states the axis position is given through this pin.
2	VCC (Red pin)	Positive power supply for servo motor is given to this pin.
3	Ground (Brown pin)	This pin is connected to ground of circuit or power supply.

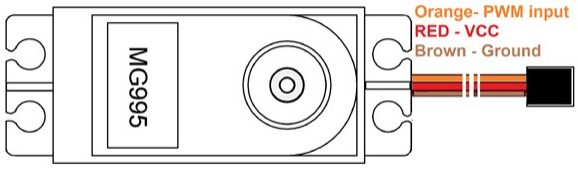


Figure 4.6 MG995 Servo Motor

Since MG995 is a servo motor providing precise rotation over 180º range its applications are many and in them a few are stated below

- The servo is suited for designing robotic arm in which wear and tear of motor is high. Being metal geared, the servo has long life and can be installed on system like robotic arm were motor work is huge.
- The servo is also suited to be used in drones and toy planes. Having a satisfying torque which is enough to overcome air resistance and control wings of plane, the servo is preferred in toy planes and drones which need precision control no matter the condition.

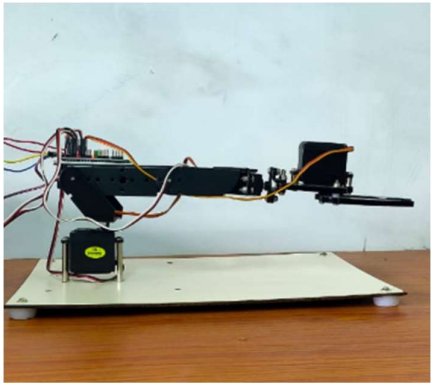


Figure 4.7 Robotic Arm

The robotic arm moves to fixed positions based on the gestures recognized by the SSD model.

4.5.2 NIRYO ONE:

Niryo One is a robot arm created for robotic learning. With its three Dynamixel XL servomotors and its NiryoSteppers, it facilitates the learning of robotics in an environment true to the industrial reality: its 6 axis allow it to reproduce all the movements required for the most advanced uses.

Figure 4.8 Niryo One Robotic Arm

There are many ways to program Niryo One (from high to low level):

- Program the robot with the learning mode: we can move the robot directly with our hands and tell it where it wants to go. With the free desktop application, Niryo One Studio, we can use visual programming (based on Blockly, similar to Scratch) to program the robot without having any programming knowledge.
- Using a Xbox controller to move the robot axis directly
- For developers, we can use the Python API to give commands to the robot using an easy-to-use programming interface. There is a modbus server running on the robot. We can develop our own APIs to connect Niryo One to any industrial device.
- We can use the digital pins on the back of the robot to make it communicate with other devices, such as Arduino and Raspberry Pi boards.
- For advanced developers, we can directly dive into the ROS code and program the robot using Python and C++.

Niryo One is accessible and versatile. To let it interact with its environment, several tools can be plugged on the universal adaptor on its hand.

- The Standard Gripper, provided with the Niryo One, is perfect for picking small or thin objects with precision.

- The Large Gripper is able to pick larger or further objects.
- The Adaptive Gripper deals easily with fragile items with uncommon shapes.
- The Vacuum Pump is very efficient for pick and place, allowing you to manipulate items with flat surfaces.
- The Electromagnet can pick and place small metallic pieces that couldn't be easily grasped by a gripper.

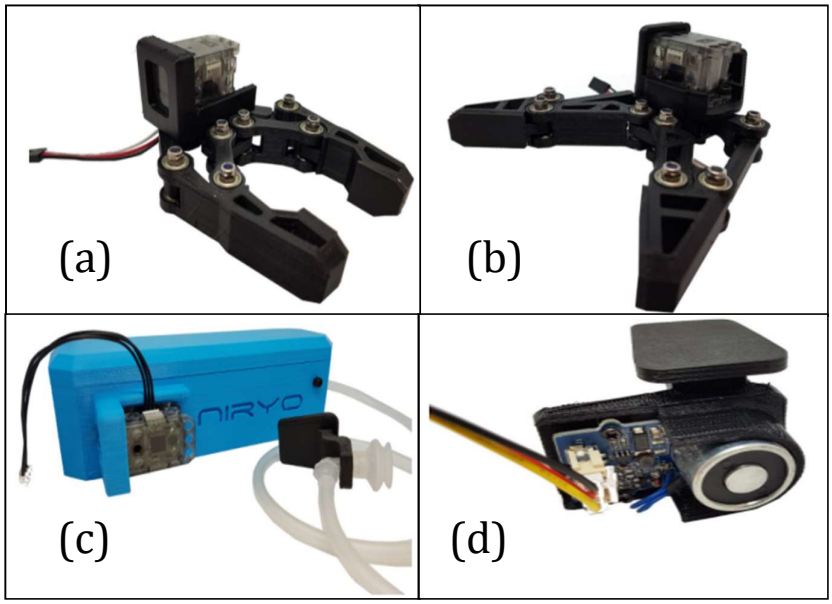


Figure 4.9 Types of grippers

a) Large gripper b) Adaptive gripper c) Vaccum pump d) Electromagnet

CHAPTER V

IMPLEMENTATION

Now that we have gone through the image processing workflow and the hardware description, it is time to combine our knowledge to build our final prototype. The various steps required to complete the project are given.

5.1 BUILDING THE PROTOTYPE:

Before implementing the neural network, it is essential to assemble the hardware components together to create the prototype.

The Jetson nano board requires an SD card with the Jetson OS flashed into it, we used a 32GB Sandisk SD Card and we flashed the Jetson OS into it using the Balena Etcher software.

Figure 5.1 SanDisk Ultra 32GB microSD card

Figure 5.2 Balena Etcher Software

There are 2 major ways through which you can power the board. The first method is to use a 10W micro USB mobile charger and power the board through its micro USB port. The DC barrel Jack of Jetson Nano can be used to power the board. To do so, a small jumper should be fit into Jetson's J48 pins.

5.2 CONTROLLING ROBOTIC ARM USING ARDUINO:

Before controlling the robotic arm with Jetson nano, first we try controlling the robotic arm using an Arduino. Since Jetson nano bit expensive, in order to avoid any mistakes we use Arduino beforehand.

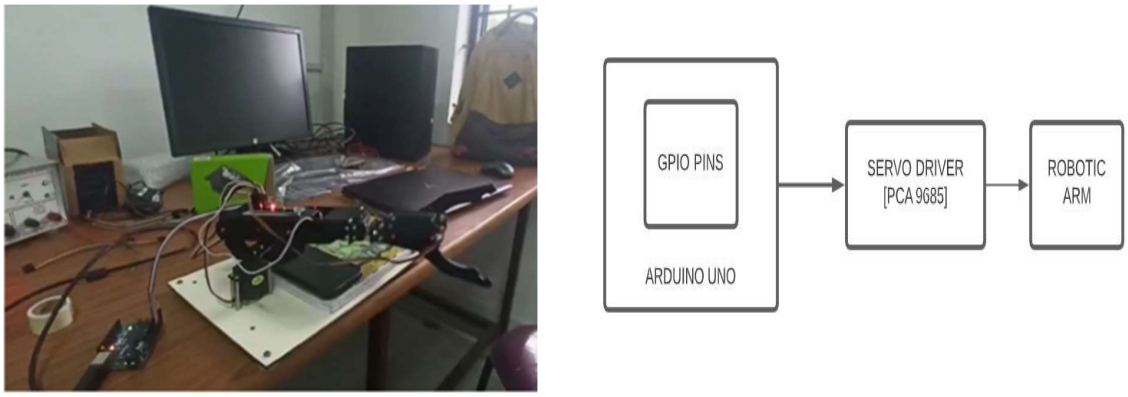


Figure 5.3 Controlling robotic arm using arduino

5.3 GESTURE RECOGNITION AND ROBOTIC ARM CONTROL USING JETSON NANO:

The Trained Model is then deployed onto Jetson Nano developer Kit. The Python library adafruit-circuit python-servokit is used to interface Jetson Nano with PCA 9685.

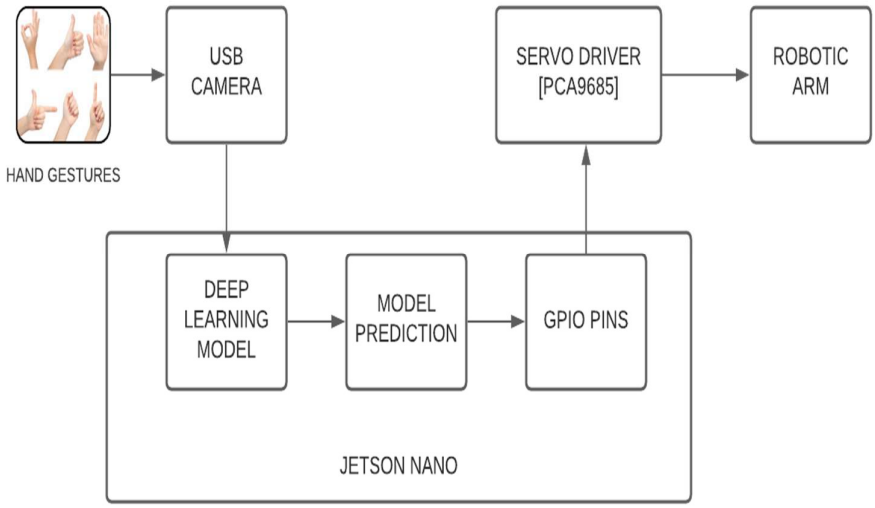


Figure 5.4 Block diagram of Jetson Nano implementation

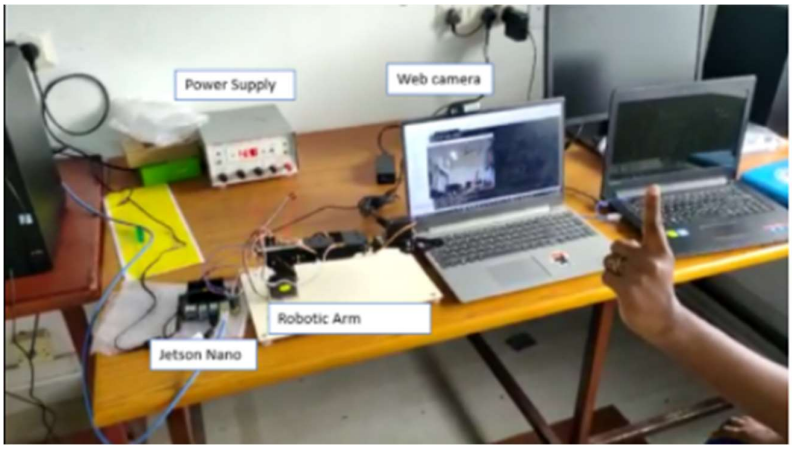


Figure 5.5 Experimental setup with Jetson Nano

The five servo motors that is used in the robotic arm are numbered as 1,2,3,4 and 5. The pulse width range (μs) differs for each servo and is determined using trial and error method. For all the DOF's of the robotic arm, the range of pulse width set for the servo motors is shown.

Table 5.1 Pulse Width Range of Robot Servo Motors

Servo Motor	Pulse Width (μs)
1	750-2200
2	650-2150
3	750-2700
4	500-1900
5	800-1670

Based on the gesture detected by the model, the corresponding GPIO signals are generated to move the Robotic arm to a specific position. For different positions, each servo motor in the robotic arm is rotated to a specific angle. The following table shows the various angles set for each servo during gesture identification.

Table 5.2 Angle Sets for Each Servo for Various Gestures

Gesture	Function	Servo 1	Servo 2	Servo 3	Servo 4	Servo 5
One	Init	90	90	145	90	180
Two	Handshake	180	50	100	180	0
Three	Hifi	90	90	90	180	180
Four	akr_gesture	130	180	140	120	180
Five	pick	90	30	110	135	90
Ok	yawn	90	180	130	180	0

Gesture	Function	Servo 1	Servo 2	Servo 3	Servo 4	Servo 5
Love	handraise	90	90	70	90	0
Tup	lift	180	180	150	80	0
Fold	punch	180	0	70	90	180
Straight	kochi	90	0	100	0	180

The robotic arm moves to a different positions corresponding to the gesture recognized.

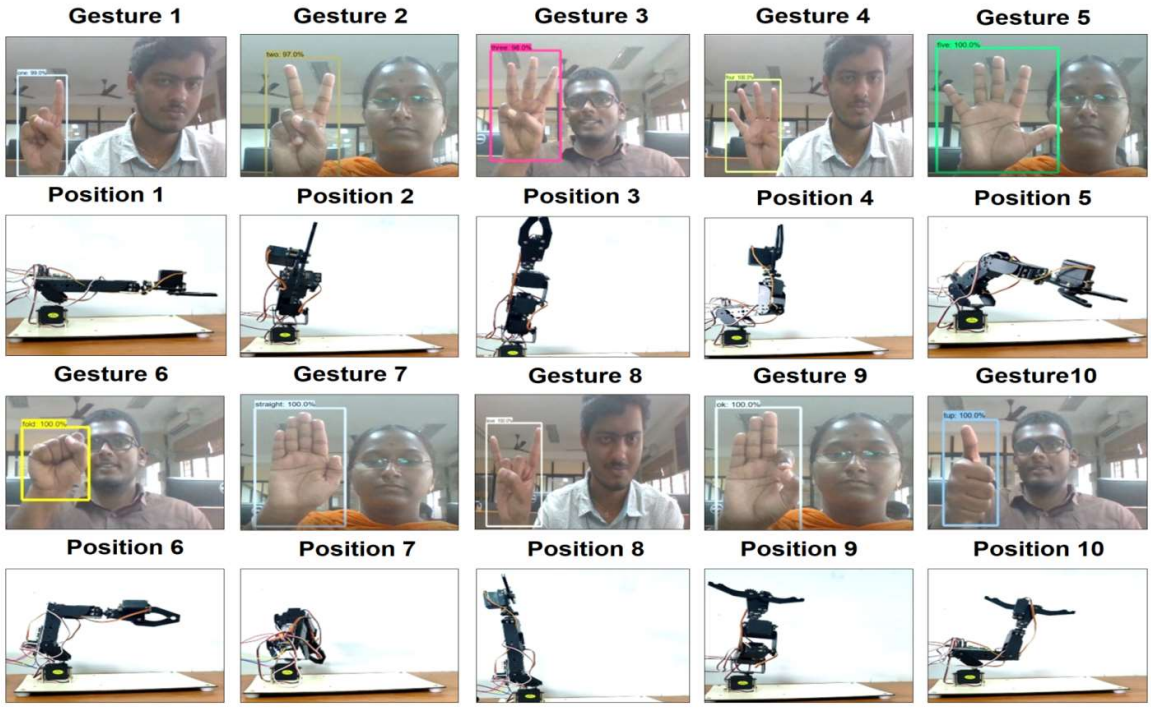


Figure 5.6 Robotic Arm Position for various gestures recognized

5.4 INTEGRATING WITH IoT:

To integrate with IOT, the Hand Gesture recognition was implemented in Jetson Nano and the robotic arm was controlled using Raspberry Pi. Both Jetson

Nano and Raspberry Pi were connected to different networks. The Raspberry Pi is remotely communicated with Jetson Nano using Message Queuing Telemetry Transport (MQTT) protocol and the MQTT broker is hosted on an Amazon Web Services (AWS) server. MQTT is an open source, lightweight and publish-subscribe network that transports messages between devices. Two types of networking entities are defined by the MQTT protocol: a message broker and several clients. The MQTT broker acts as a post office. Information is organized in hierarchy of topics. When a publisher wants to publish a message, it is sent to the MQTT broker under a specific topic and anyone who is subscribed to that topic will receive the message.

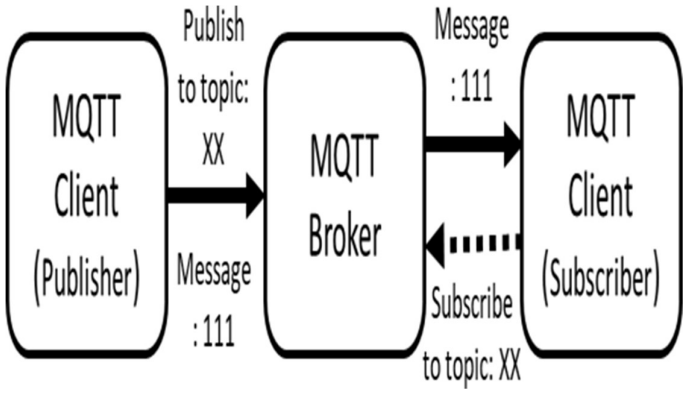


Figure 5.7 MQTT PROTOCOL

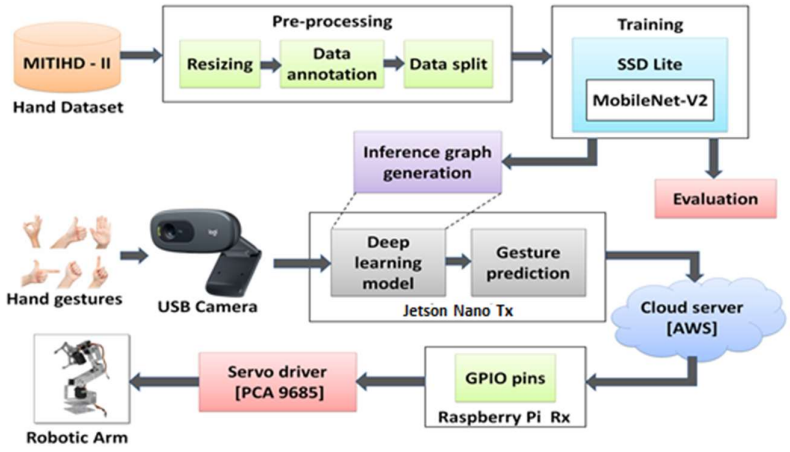


Figure 5.8 IOT integrated overall block diagram

PSEUDOCODE OF GESTURE DETECTION ON JETSON NANO TX

Input: Continuous frames of images V from web camera. f is the frame at an instant

Output: x1, y1, x2, y2, P_s, P_c for each f, where x1, y1, x2, y2 are bounding box coordinates. P_s, P_c is predicted score, class.

Load CNN Model into Session ()

WHILE TRUE:

f = Capture_Frame_From_Webcam ()

Gesture Recognition using SSD Lite MobileNet-V2

x1, y1, x2, y2, P_s, P_c = Predict (f)

IF P_s > threshold:

Publish P_c to MQTT broker under topic 'Gesture'

END IF

END WHILE

PSEUDOCODE OF ROBOTIC ARM CONTROL ON RASPBERRY PI RX

Input: Predicted class (P_c)

Output: Generate corresponding GPIO signals to control Robot.

Enable to MQTT broker under topic 'Gesture'

WHILE TRUE:

IF P_c is received:

Perform Desired Robotic arm Action ()

END IF

END WHILE

5.5 IMPLEMENTATION USING NIRYO ONE ROBOTIC ARM:

To obtain co-ordinates of the arm's position:

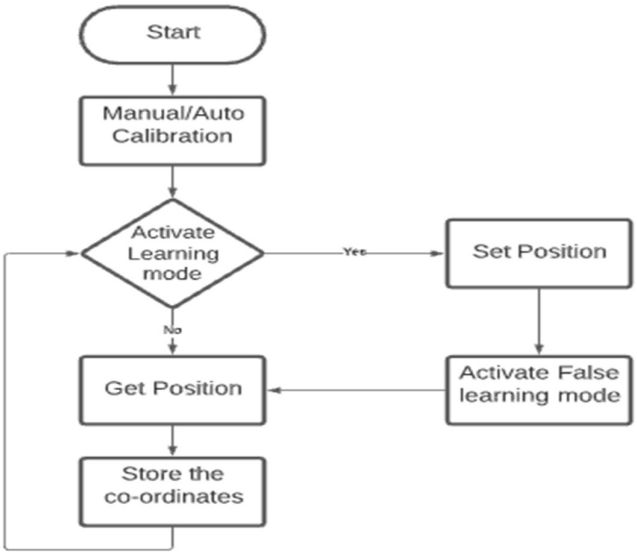


Figure 5.9 Flow chart for training

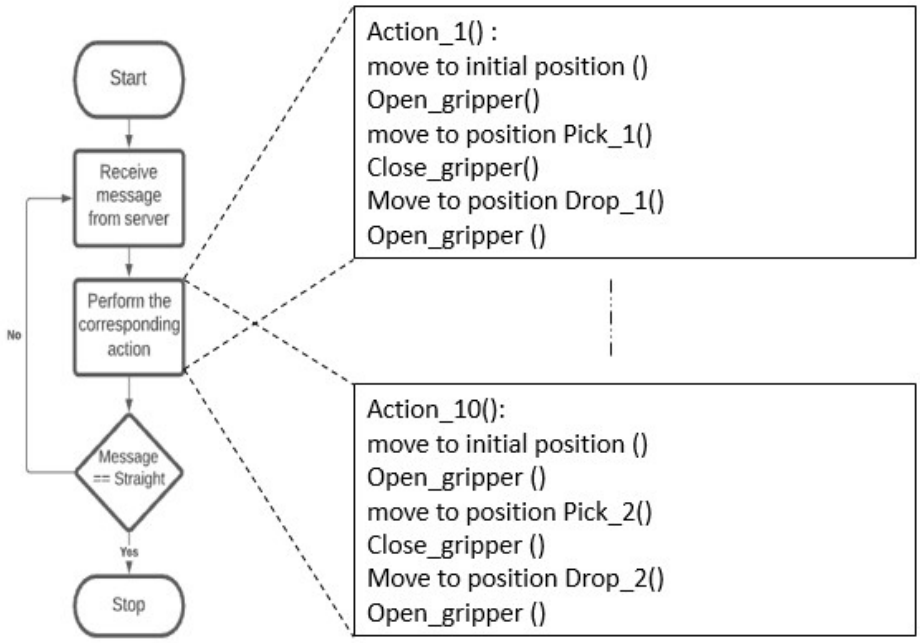


Figure 5.10 Flow chart for Execution

SETUP:

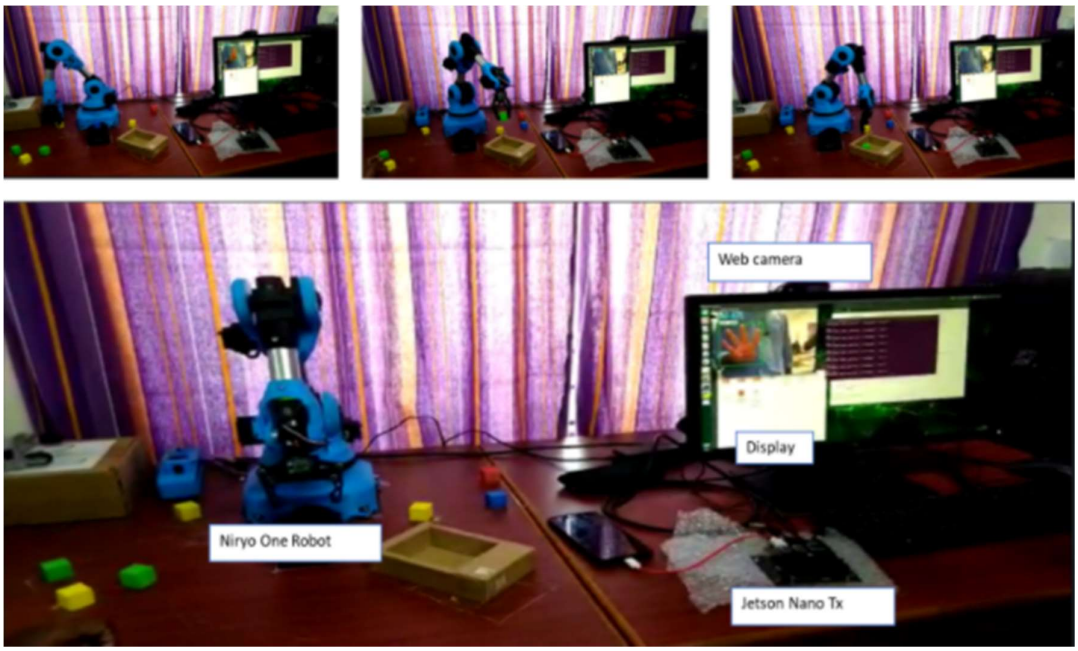


Figure 5.11 Niryo one Robotic arm setup and Robot in action

CHAPTER 6
RESULTS AND DISCUSSIONS

The SSD Lite Mobilenet-V2 and SSD Inception-V2 based hand gesture recognition models are evaluated. The performance of the models are assessed by Average Precision (AP), Average Recall (AR) and F1-Score. The prediction time of the model is calculated during the detection process on a Raspberry Pi processor. Average Precision of different models is calculated for various values of IoU threshold.

The term AP_{all} is the overall average precision, AP_{small} is the average precision for the size of the object in an image lesser than 32×32 pixels, AP_{medium} is the average precision for the object size lesser than 96×96 pixels and greater than 32×32 pixels and AP_{large} is for the object size greater than 96×96 pixels. Similarly, AR_{small} average recall for the object is lesser than 32×32 pixels, AR_{medium} and AR_{large} are the average recall for the object size smaller than 96×96 pixels and greater than 32×32 pixels, greater than 96×96 pixels, respectively. AR_1, AR_{10} and AR_{100} are the average recall values determined for various number of detections such as 1, 10 and 100.

Both the models are trained for 50,000 epochs with batch size as 8. SSD Lite Mobilenet-V2 and SSD Inception-V2 models are trained with a learning rate of 0.0002. The models are compared on the basis of Average Precision, Average Recall and F1 Score. It may be noted that SSD Inception-V2 performs better on terms of accuracy while SSD Lite MobileNet-V2 has lower prediction time.

Below Table shows the Average Precision of Single Stage Deep CNN models using MITIHD-II datasets for the various IoU ranges of 0.5, 0.75 and 0.5:0.95. The performance metrics obtained for SSD Inception-V2 is described as $AP_{0.50}$ is 0.993,

$AP_{0.75}$ is 0.991 and $AP_{0.5:0.95}$ is 0.884. The metrics of SSD Lite MobileNet-V2 is given as $AP_{0.50}$ is 0.987, AP0.75 is 0.968 and $AP_{0.5:0.95}$ is 0.804 respectively. The comparison of these two models shows that the SSD Inception-V2 has higher Average precision values when compared to the SSD Lite MobileNet-V2 model.

Table 6.1 Average Precision of Single Stage Deep CNN Models Using MITIHD-II for various IoU Ranges

CNN Models	Average Precision (AP)					
	0.5	0.75	0.5:0.95	$Small_{0.5:0.95}$	$Medium_{0.5:0.95}$	$Large_{0.5:0.95}$
SSD Inception-V2	**0.993**	**0.991**	**0.884**	**0.858**	**0.877**	**0.889**
SSD Lite MobileNet-V2	0.987	0.968	0.804	0.723	0.742	0.823

The Average Recall of Single Stage Deep CNN models using MITIHD-II datasets for the IoU range of 0.5:0.95 is illustrated in table below. The Average Recall of SSD Inception-V2 model is 0.911 for AR_1 and 0.914 for both AR_{10} and AR_{100}. For SSD Lite MobileNet-V2 model AR_1 is 0.839 and 0.842 for both AR_{10} and AR_{100}.

Table 6.2 Average Recall of Single Stage Deep CNN Models Using MITIHD-II for ranges 0.5:0.95

CNN Models	Average Recall (AR)					
	1	10	100	$Small_{0.5:0.95}$	$Medium_{0.5:0.95}$	$Large_{0.5:0.95}$
SSD Inception-V2	**0.911**	**0.914**	**0.914**	**0.875**	**0.898**	**0.919**
SSD Lite MobileNet-V2	0.839	0.842	0.842	0.764	0.791	0.861

Following demonstrates the comparison of the Performance metrics of Single Stage Deep CNN models using MITIHD-II dataset for the IoU range of 0.5. The AP0.5, AR0.5, and F1-Score0.5 are calculated as 99.27%, 95.58% and 97.39% for SSD Inception-V2 Model and 98.74%, 94.11% and 96.37% for SSD Lite MobileNet-V2 Model. Though the Precision, Recall and F1-score are higher for SSD Inception-V2 model, the prediction time of SSD Lite MobileNet-V2 Model is low (0.67s). This makes a significant contribution in the speed of communication between human and robot.

Table 6.3 Comparison of Performance Metrics Single Stage Deep CNN Models Using MITIHD-II Dataset

CNN Models	$AP_{0.50}$ (%)	$AR_{0.50}$ (%)	$F1\text{-}Score_{0.50}$ (%)	Prediction time (s) Jetson Nano
SSD Inception-V2	**99.27**	**95.58**	**97.39**	0.21
SSD Lite MobileNet-V2	98.74	94.11	96.37	**0.14**

The overall loss curve is plotted with number of training steps in x axis and loss in y axis. The graph is shown in fig.6.1. The loss is higher at the start and slowly decreases towards the end. It may be noted from the graph that after 50,000 steps SSD Lite MobileNet-V2 has higher loss than SSD Inception-V2 CNN model.

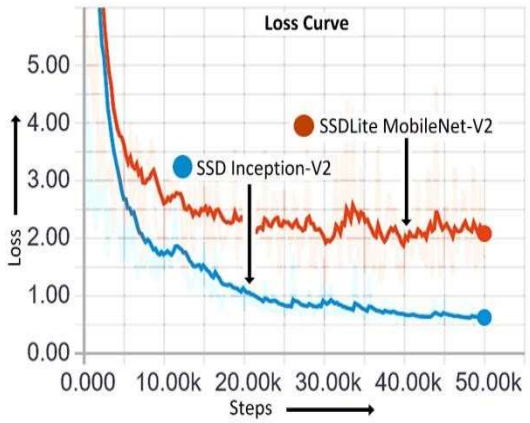


Figure 6.1 Loss Curve of Single Stage CNN models

CHAPTER 7
CONCLUSION

The remote location robotic arm control is performed using the single stage deep CNN hand gesture recognition model. The SSD lite MobileNet-V2 and SSD Inception V2 models are trained and tested using MITI HD-II dataset. SSD Lite model is preferred to run the hand gesture recognition applications on low power devices such as a Jetson Nano due to its light weight and speedy recognition. Jetson Nano controller is used for hand gesture recognition and to transmit the information to cloud. Raspberry Pi controller receives the information from the cloud and control the operations of Robotic arm. Both Jetson Nano and Raspberry Pi are remotely communicated using MQTT Protocol and the MQTT broker is hosted on an AWS server. For IoU threshold of 0.5, the Average precision, Average recall and F1 score of SSD Inception-V2 model is calculated as 99.27%, 95.58%, and 97.39% and the corresponding values for SSD MobileNet-V2 model is 98.74%, 94.11% and 96.37% respectively. The prediction time for SSD Inception-V2 Model using Raspberry Pi controller is 1.2s whereas SSD Lite MobileNet-V2 model consumes 0.67s. The MobileNet model runs faster at the cost of lower precision. Hence to run on low power edge devices, the SSD Lite Mobilenet-V2 model is preferred. Further reduction in prediction time will improve the speed of communication between humans and robots. This prototype can be extended into a Tele-surgery system which is considered as our future work.

REFERENCES

1. Saurabh A. Khajone, S. W. Mohod and V.M Harne, "Implementation of wireless gesture controlled robotic arm", International Journal of Innovative Research in Computer and Communication Engineering, vol. 3, no. 1, 2015.

2. Aggarwal, L.; Gaur, V.; Verma, P. Design and implementation of a wireless gesture controlled robotic arm with vision. International Journal of Computer Applications. 2013, 79, 39–43.

3. Sagayama KM, Viyasb TV, Hoa CC, Heneseyb LE (2017) Virtual robotic arm control with hand gesture recognition and deep learning strategies. Deep Learn Image Process Appl 31:50.

4. Howard, A., G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T.,Marco Andreetto, M., and Adam, H., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", Google Inc., 2017.

5. Othman N. A. and I. Aydin, "A new deep learning application based on movidius ncs for embedded object detection and recognition", 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), pp. 1-5, Oct 2018.

6. Wong, A., Shafiee, M.J., Li, F., Chwyl, B.: Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. arXiv preprint arXiv:1802.06488 (2018).

7. Rubin Bose. S, Sathiesh Kumar. V. "Efficient Inception V2 based Deep Convolutional Neural Network for Real-Time Hand Action Recognition". IET Image Processing, Vol. 14, no. 4, March 2020, pp. 688 – 696.

8. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. CVPR, 2018, doi: 10.1109/CVPR.2018.00474.

9. Rubin Bose. S, Sathiesh Kumar. V. "Hand Gesture Recognition Using Faster R-CNN Inception V2 Model". AIR 2019: Proceedings of the Advances in Robotics 2019, ACM digital library, July 2019, Article No.: 19, Pages 1–6.

10. D. Kang et al., "Room Temperature Control and Fire Alarm/Suppression IoT Service Using MQTT on AWS," 2017 International Conference on Platform Technology and Service (PlatCon), Busan, Korea (South), 2017, pp. 1-5, doi: 10.1109/PlatCon.2017.7883724.

11. S. Chen, J. Hong, T. Zhang, J. Li and Y. Guan, "Object Detection Using Deep Learning: Single Shot Detector with a Refined Feature-fusion Structure", In 2019 IEEE International Conference on Real-time Computing and Robotics (RCAR), pp. 219-224, 2019, August.

12. Kingma, D., & Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

13. https://www.jetsonhacks.com/2019/07/22/jetson-nano-using-i2c/

14. https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/using-the-adafruit-library

15. https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10

16. https://medium.com/tensorflow/using-tensorflow-lite-on-android-9bbc9cb7d69d

17. https://www.tensorflow.org/lite/guide

18. https://www.tensorflow.org/lite/tutorials

19. https://niryo.com/product/niryo-one/