Hand-Gesture Based Interaction with Nao

Under the guidance of Prof. G. C. Nandi
Arpit Bahety (IIT2015089), Tanmay Jaiswal (IIT2015059), Ankur Dengla (IIT2015510)

CONTENTS

1. ABSTRACT

The project tackles the problem of hand gesture recognition in real time. The gestures being recognized are not images but videos that is the movement of the hand with respect to time is also taken into consideration (Example - Swiping left, zooming in, etc). The problem also incorporates the fact that the environment in which the gestures are being performed is not constrained by any factors i.e. the domain for this system is hand gesture videos captured by any standard webcam.
The first part of this project involved exploring different deep learning architectures for this purpose and figure out the best one for this application. The two main requirements for an architecture to be accepted were high accuracy and low latency. Finally, a 3D CNN based deep neural network was used.
The second part involved integration of this neural network to Nao robot so that it could be used for gesture based human-robot interaction. The main challenge was doing this considering the limited computing power of Nao. Hence, all computations were decided to

be done on an external system and thus, this involved creation of an interaction pipeline between Nao and the system.

2. INTRODUCTION

The problem of hand gesture recognition comes under the umbrella of Computer Vision in machine learning. Computer Vision basically involves giving computers the capability to process images and videos in the way humans do. The aim is to develop a theoretical or algorithmic basis for the machine so as to enable it to obtain visual understanding. It is an important area for research and innovation because of the wide range of possible applications that it can have such as automated video monitoring, video and image tagging, gesture based interaction with robots, etc.

Under the category of Computer Vision, an important sub-category is the recognition and interpretation of human movements or “gestures”. A gesture can be defined as any physical movement, large or small, by a person that is intended for communicating a specific message or instruction. In other words, gestures are mostly non-verbal means of communicating. According to this simple definition, we can say that hand gestures are gestures performed by only using the hands. The problem of hand gesture recognition basically involves the designing an algorithm so as to enable communication with the computer via hand gestures. This is an important and popular problem because of the applications that it can have such as in VR Systems, surgery using robots, and sign language interpretation systems to name a few.

Also, Kinect already can tackle this problem but it requires specialized hardware to be implemented. This system on the other hand aims at providing effective and efficient gesture recognition using input videos only from standard webcams that are easily available thus making the technology accessible by a wide range of people. By using video gestures instead of images, we also add a temporal dimension to the problem. This facilitates recognizing a much wider range of gestures. Also, the aim is to be able to use this system irrespective of the background i.e. at any location. The dataset used involves videos collected from the community. Thus, it has a variety of backgrounds, lighting and other varying factors.

After an effective architecture was obtained for hand gesture recognition, the module was integrated with Nao robot for the purpose of hand gesture controlled communication with it. Right now, this communication refers to controlling the movement of the robot with the help of hand gestures. This hand gesture based communication, if achieved effectively, can be developed further to enable people without any knowledge of programming to communicate with robots and thus, help towards the deployment of robots for a much wider range of applications than they are currently being used for.

3. LITERATURE REVIEW

Pavlo Molchanov et al [1] in their work on drivers' hand gesture recognition have used 3D Convolutional Neural Networks. To improve training they have applied spatio-temporal data augmentation. They've achieved 77.5 % accuracy on the VIVA challenge dataset.

The German AI company TwentyBN [2] studied several architectures and concluded on an architecture consisting of a 3D Convolutional Network followed by a recurrent layer (LSTM).
They achieved an accuracy of 82.34 % on the 20BN Jester dataset. [4]

Zhi-hua Chen et al. [3] in their paper on real-time hand gesture recognition segment the hand pixels on the basis of HSV values and then isolate each finger. Then, on the basis of the number and orientation of the fingers detected, they identify the gesture on the basis of a simple rule-engine. Their work, however, does not work when there is a high degree of variance in the background. Also, their work involves static gesture recognition.

Okan Köpüklü et al. [5] propose a technique called Motion Fused Frames in their paper that fuses motion information into a static image to represent the temporally dependent data. These frames are then appended to the existing RGB samples as extra channels. Using MFF, they were able to obtain 96.28% accuracy on the Jester dataset.

William T. Freeman et al. [6] in their paper propose a method to recognize hand gestures which is based on a pattern recognition technique. This was developed by McConnell [7] employing histograms of local orientation. Orientation histogram as a feature vector for gesture classification and interpolation has been used in their methodology.

4. BACKGROUND

4.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are a class of deep learning algorithms that are essentially feed-forward neural networks which primarily have applications in image and video recognition. The convolutional layer is the main point of difference between a CNN and a Conventional Neural Network and provides the added advantage of implicit feature extraction.

CNN basically consists of 4 segments -

1. Convolutional layer
2. Pooling layer
3. Fully-connected layer
4. Output layer

4.1.1 Convolutional Layer:

The main building block of a CNN is the Convolutional layer. The convolution step is as follows:

Typically each convolution operation is followed by the application of an activation function eg. ReLU
A conventional neural network could learn features and classify inputs, but practically it is not feasible. The number of neurons needed would be high even in shallow networks. For example, for an image of size 75 x 75, the size of weights for each neuron in the second layer would be 5625. Convolutional layer solves this problem as, regardless of the image size, a filter of size, say 3 x 3 can be used to extract features, with the total number of parameters being just 9. Thus convolutional layer allows the network to be deeper with fewer parameters.
The convolution operation is denoted in Figure 4.1.

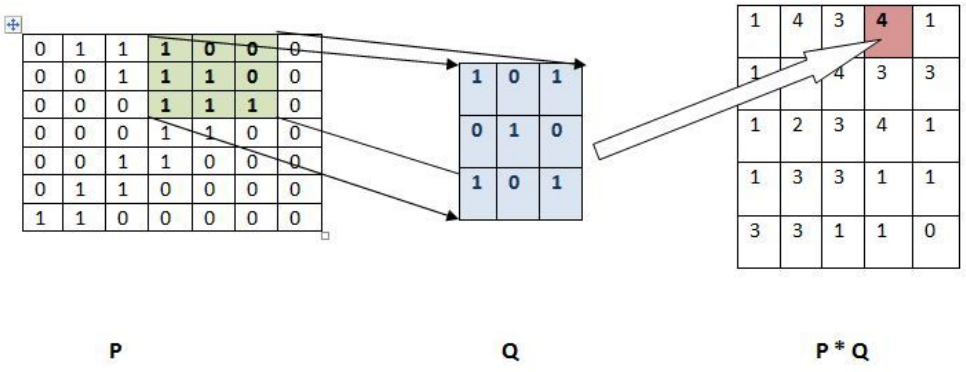

Fig 4.1 : Example of convolution operation

4.1.2 Pooling Layer:

Pooling layer progressively reduces the size of the output of the convolutional layer and this helps reduce the number of parameters in the network and thus the computation. An example is illustrated in Figure 4.2.

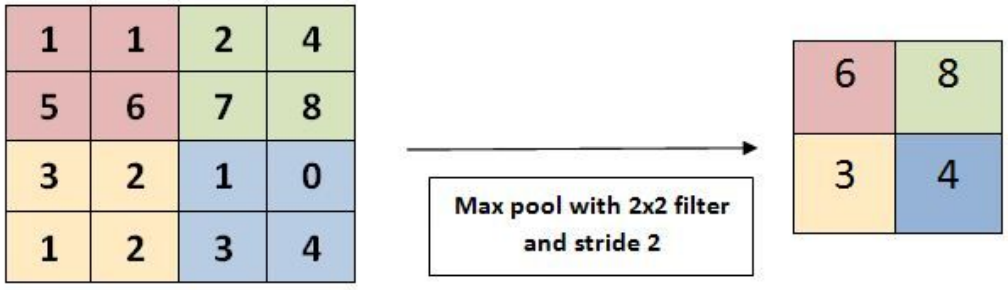

Fig 4.2 : Example of Max Pooling

4.1.3 Fully Connected layer:

This is basically the conventional neural network where every neuron in one layer is connected to every other neuron in the next layer.

4.1.4 Output Layer:

Output layer consists of neurons equal to the number of classes. Typically the softmax activation is applied here.

4.1.5 3D CNN:

3D CNN is mainly used for classifying videos. The difference here is that the input is a 3D tensor as compared to a 2D tensor, having the two dimensions of image height and image width similar to a 2D CNN with an added temporal dimension (image depth).
The steps to creating input for 3D CNN is as follows:

1. Create frames of a video at discrete intervals of time, where each frame is a 2-D image.
2. Stack these frames along the depth to obtain a 3-D input.

The convolutional filter is also a 3D filter.

4.2 Nao

Nao is a humanoid robot created by the French company, Aldebaran for the purpose of academic research. It is autonomous and programmable. It can be made to do various tasks by using its various sensors and then redirecting the output to the actuators. The list of sensors available in Nao is -

- 25 degrees of freedom of motion
- Four microphones
- Two cameras
- Touch sensors
- Inertial measurement unit: to detect if robot is sitting or standing up

The sensor mainly being used in this project is the Nao camera. It can record videos ranging from resolution of k4VGA (1280*960 pixels) to kQQQQVGA (40*30 pixels) at any frame rate ranging from 1 to 30 fps.
A live feed of the video at kQQVGA (160*120 pixels) is obtained from the camera onto the server and then processed further.

5. METHODOLOGY

The aim of the project is the classification of basic human-hand gestures which has numerous applications including ‘gesture controlled robots’. For the purpose of a comparative study amongst different architectures, hitherto we have implemented two architectures.

5.1 2D CNN with Image Concatenation

Preprocessing: Video cannot be directly fed to a 2D CNN. To cater to this problem we have taken the following approach. Frames of video at discrete intervals of time are taken, where each frame is saved as a 2-D image of size 100 x 100. 30 frames of a video are taken. If the number of frames are less than 30, frames of zeros are appended.
All the 30 frames/images are concatenated to obtain a 3000 x 100 image. As the images are RGB, the input size effectively is 3000 x 100 x 3. The input is resized to 500 x 100 x 3. This is the input to the 2D CNN.

Architecture of 2D CNN:

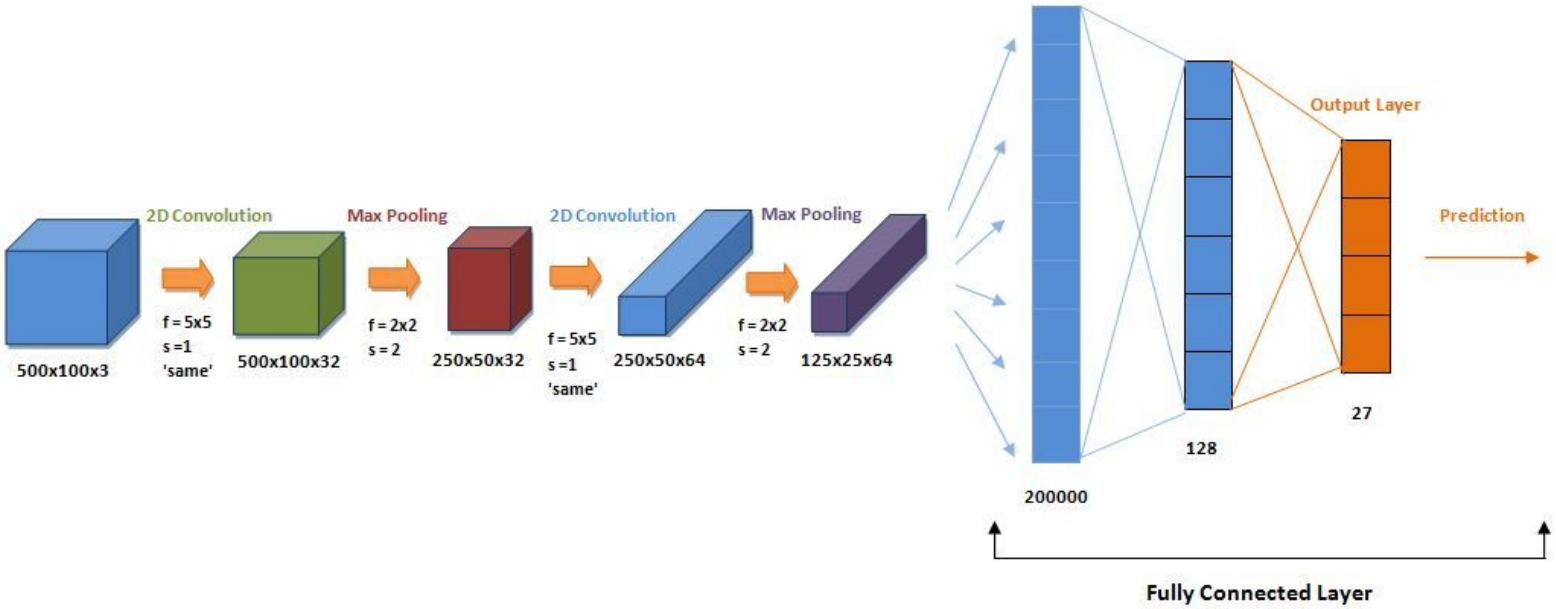

Fig 5.1 : Architecture of 2D CNN used

5.2 3D CNN

Preprocessing: Similar to the preprocessing of 2D CNN, Frames of video at discrete intervals of time are taken, where each frame is saved as a 2-D image of size 176 x 100. Each image is cropped at the center to output an image of size 84 x 84. 18 frames of each video are stacked sequentially along the depth to output a 3D tensor of shape 84 x 84 x 18. This 3D tensor is normalized using mean 0 and standard deviation of 1. This is the input to the 3D CNN.

Architecture of 3D CNN:

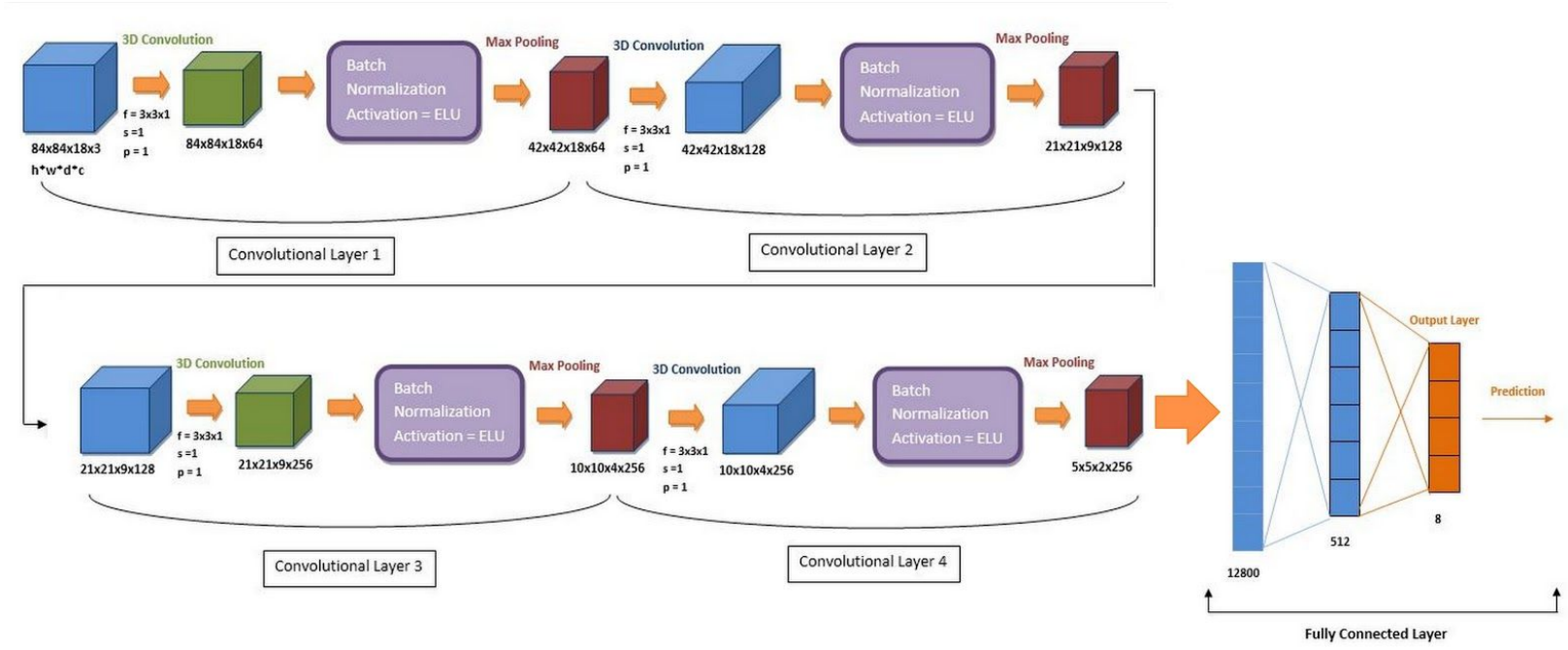

Fig 5.2 : Architecture of 3D CNN used

5.3 Integration With Nao

The main challenge with Nao was the limited computation power available on it. Thus, loading the neural network on the robot was out of the question as the 1 GB RAM available on it would be exhausted by allocating the weights of the network. Also the relatively low processing power of Nao would have a lot of latency in the prediction. Hence, there was a need to perform all the computations of predicting the gesture externally. This created the need of establishing a communication pipeline between Nao and the computer.

The first step in creating this pipeline was making alterations to the neural network so as to make it more lightweight and thus, efficient to make faster predictions. This was done by reducing the number of classes of the 20bn jester dataset from 27 classes down to 8. The lower number of classes makes the neural network architecture required for prediction less complex. The 8 classes that were used for interaction with Nao along with their corresponding actions are -

- **No Gesture -** Do nothing
- **Swiping two fingers up -** Stand Up
- **Swiping two fingers down -** Sit Down
- **Pulling two fingers in -** Move Forward

- **Push hand back -** Move Backward
- **Swiping two fingers left -** Move left
- **Swiping two fingers right -** Move right
- **Shaking hand -** Say Hi

Since, the number of classes was reduced to almost one-third, a much lighter network could be used for the prediction process. This made the prediction process faster.

The next part was establishing the proper communication pipeline. The pipeline is described by the figure below -

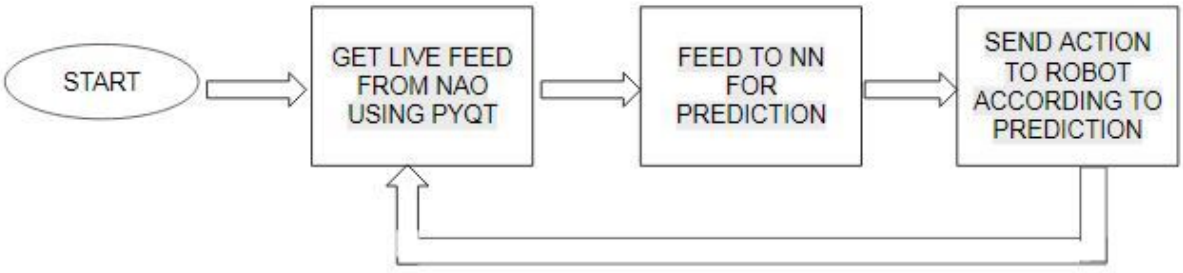

Fig 5.3 : Communication pipeline with Nao

The first step of the pipeline was to get the live feed from the Nao camera to the server to be processed. This was done via the PyQt4 [8] python library. Even though Nao can capture images at 30 fps, the server was limited by processing power and thus, the practically obtained fps on the server is low. The feed is kQQVGA (160*120 pixels). Frames are captured from the feed and then fed to the neural network architecture for the gesture to be predicted. The main challenge in doing this was that the NaoQi API [9] used for interacting with Nao is in Python 2.7 whereas our neural network architecture is in Python 3. The two codes in this case interact using socket programming. After the neural network makes its prediction, the NaoQi code then sends the corresponding action to the robot to perform.

6. RESULTS

The dataset being used for the purpose of this project is the 20BN-Jester dataset [4] which comprises of labeled videos of humans performing pre-defined actions. The dataset was collected from the community. The actions were performed in front of a

standard computer webcam. There are video samples spanning 27 classes. We used only 8 classes out of the 27 for this project. The classes chosen were the ones relevant to the actions that were needed to be performed. Each sample is a set of frames (JPG images) obtained from the video submitted at 12 frames per second. The height of each image is 100 pixels with variable width from 150 - 176 pixels. There are total 43,797 training samples and 5144 validation samples (with labels). The classes and the number of videos for each class are given in Fig 6.1.

We used only 8 classes out of the 27 and ended up with

Gesture	Number of Videos
Pulling Two Fingers In	5315
Shaking Hand	5314
Sliding Two Fingers Down	5410
Sliding Two Fingers Up	5345
Sliding Two Fingers Left	5244
Sliding Two Fingers Right	5262
Stop Sign	5413
No Gesture	5344

Fig 6.1

The results obtained were as follows -

2D CNN with Image Concatenation

In this case, the frames of the video were concatenated along the width in order to obtain a single image. Since the size of each frame was 100 pixels x 100 pixels, the final image obtained was of size 3000 pixels x 100 pixels. This was resized to 500 pixels by 100 pixels and fed to the neural network.
The results obtained for this network were not promising as the validation accuracy stagnated around the 8% - 15% mark after 15-20 epochs even after several tests whereas the benchmark for this dataset is around 96.6%. This may be due to the fact that

the architecture was very rudimentary and the limited number of parameters may not have been able to learn effectively. Also, another cause may be the fact that the image was resized to 1/6th of its original size which could have led to loss of important details in the image and hence, resulted in faulty classification.
These results facilitated the need to explore alternative architectures.

3D CNN

After the poor results obtained from 2D convolution, a 3D CNN architecture was employed.
The training process took a total of 5 hours for 11 epochs. It was trained on a Nvidia 1080Ti GPU with 10 GB RAM. A learning rate of 0.001 was used with Cross-Entropy being the loss function and Stochastic gradient descent as the optimizer to update the weights.

The following results were obtained -

Epoch	Top 1 (Accuracy)	Top 3 (Accuracy)	Loss
0	15.000	50.000	2.0500
1	75.129	92.330	0.7332
2	87.750	97.561	0.3868
3	90.880	98.357	0.2868
4	92.715	98.810	0.2275
5	94.199	99.202	0.1775
6	95.348	99.488	0.1430
7	96.012	99.580	0.0970
8	97.237	99.766	0.0824
9	97.547	99.857	0.0713
10	98.538	99.909	0.0478
11	98.898	99.968	0.0363

Fig 6.2 Training Top 1 and Top 3 Accuracy Table

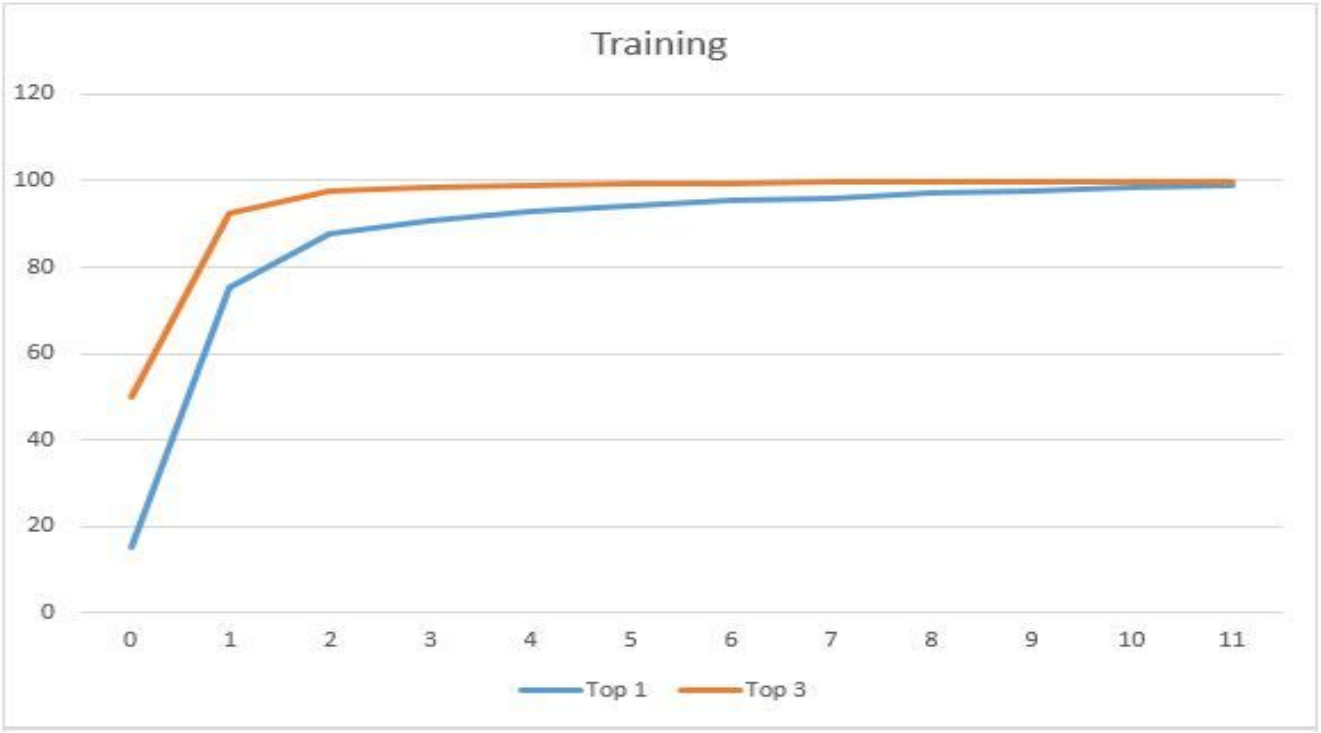

Fig 6.3 Training Top 1 and Top 3 Accuracy Graph (Number of Epochs vs Accuracy)

Epoch	Top 1 (Accuracy)	Top 3 (Accuracy)	Loss
0	60	90.000	0.9927
1	86.779	97.688	0.4020
2	89.908	98.341	0.3179
3	89.989	98.096	0.3180
4	90.790	98.556	0.3040
5	90.968	98.060	0.3240
6	91.444	98.065	0.3280
7	92.103	98.430	0.3170
8	92.900	98.776	0.2693
9	93.145	98.803	0.2869
10	92.383	98.504	0.3082
11	91.975	98.585	0.3800

Fig 6.4 Validation Top 1 and Top 3 Accuracy Table

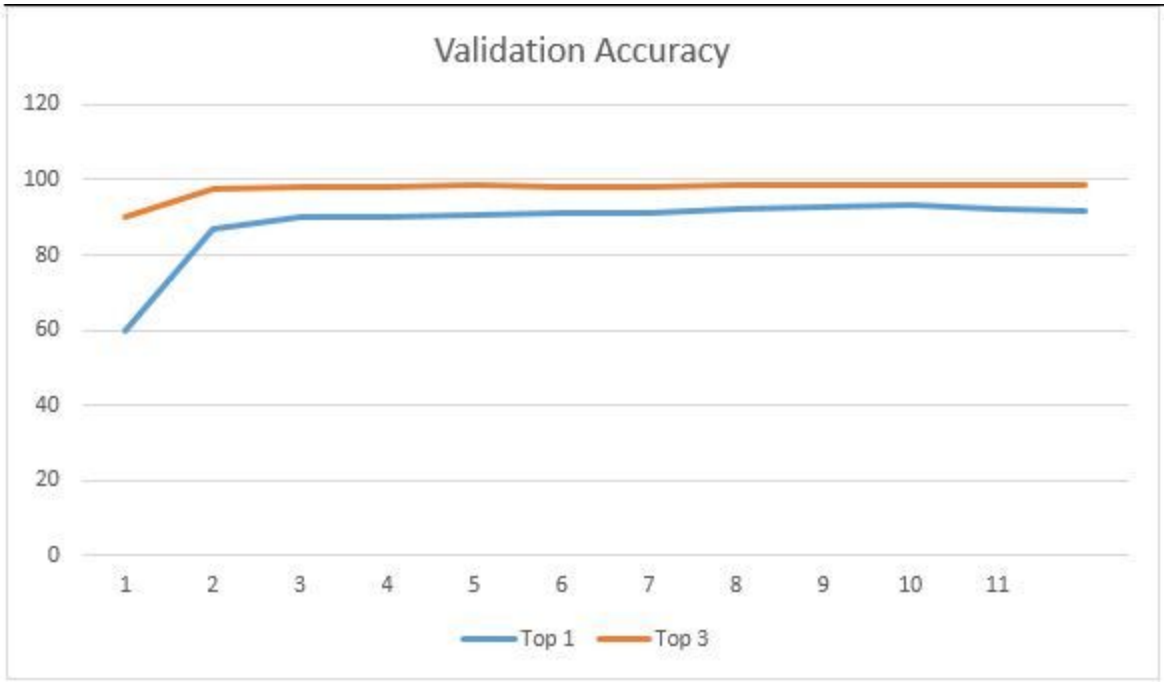

Fig 6.5 Validation Top 1 and Top 3 Accuracy Graph (Number of Epochs vs Accuracy)

Fig 6.6 Loss in Training and Validation (Number of Epochs vs Loss)

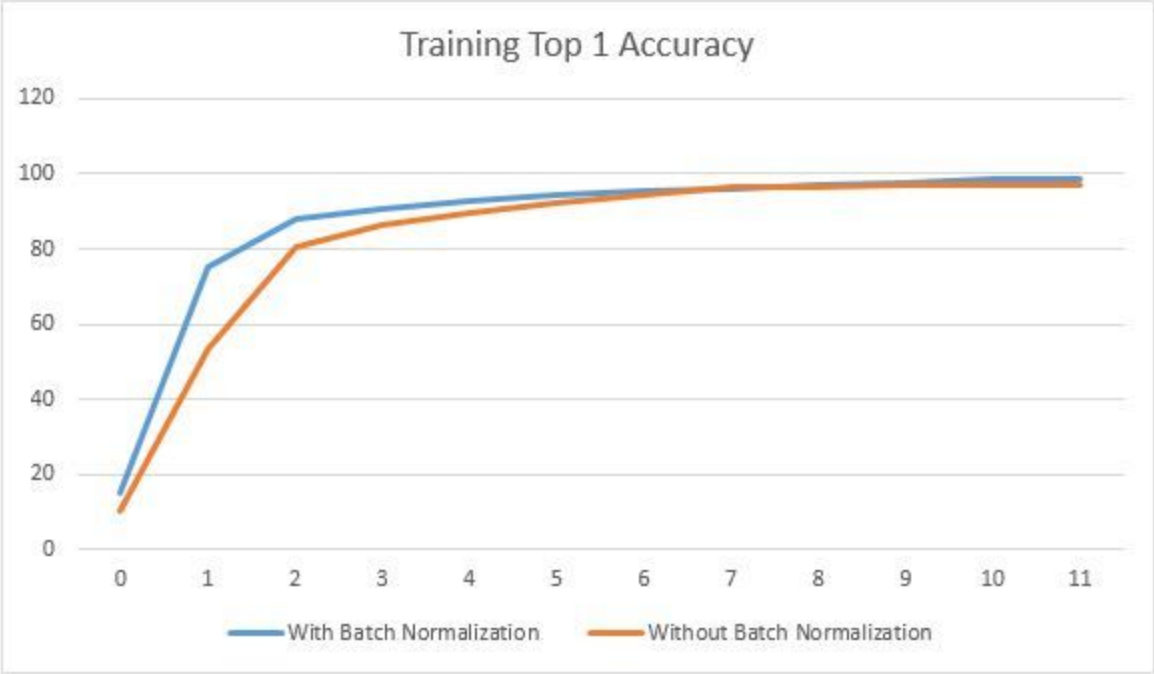

Fig 6.7 Comparing model with and without Batch Normalization (Number of Epochs vs Accuracy). Clearly the training process is faster with Batch Normalization.

7. CONCLUSIONS

In this paper we started with 2D CNN model which did not perform well. Next, we developed a 3D CNN model giving a top-1 training accuracy of 98.89% for our dataset and a top-1 validation accuracy of 91.975%. This model was then integrated with the NAO robot to recognize the gestures performed by humans and take actions accordingly.

8. REFERENCES

[1] Hand Gesture Recognition with 3D Convolutional Neural Networks - Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz - NVIDIA, Santa Clara, California, USA

[2]https://medium.com/twentybn/gesture-recognition-using-end-to-end-learning-from-a-large-vid eo-database-2ecbfb4659ff

[3] Real-Time Hand Gesture Recognition Using Finger Segmentation - Zhi-hua Chen, Jung-Tae Kim, Jianning Liang, Jing Zhang, and Yu-Bo Yuan - Department of Computer

Science and Engineering, East China University of Science and Technology - The Scientific World Journal Volume 2014

[4] https://20bn.com/datasets/jester

[5] Motion Fused Frames: Data Level Fusion Strategy for Hand Gesture Recognition - Okan Köpüklü, Neslihan Köse, Gerhard Rigoll - Institute for Human-Machine Communication Technical University of Munich, Germany

[6] Orientation Histograms for Hand Gesture Recognition - William T. Freeman and Michal Roth

[7] Method of and Apparatus for Pattern Recognition - Robert K. McConnell, Arlington, Mass.

[8] https://www.riverbankcomputing.com/software/pyqt/

[9] http://doc.aldebaran.com/2-1/naoqi/index.html