Mechatronic Systems Engineering
School of Engineering Science
SIMON FRASER UNIVERSITY

MSE 491: Project Report

Real-Time Sign Language detection

[Group #26]

Name	Student ID	Contribution %
Harsimranjeet Singh	301289434	Equal
Syed Salman H Abdi	301277622	Equal

MSE 491- Application of Machine learning in Mechatronic Systems

Prepared for Dr. Mohammad Narimani

23rd April 2021

Table of Contents

List of Figures

Introduction

Motivation

Modern day machine learning and speech recognition go hand in hand, and it is one of it's largest applications. Prominent examples of this that come to mind are Siri and Google Assistant which are virtual assistants that recognize our speech both in terms of content and who the specific speaker is. This technology is prevalent on almost all smartphones which go to show how far the computing power of our phones in addition to optimization in code have come that allows this to be possible.

While the speech recognition branch of machine learning has taken off, it has left behind its counterpart which is sign language recognition. The same convenience is not accessible to those who have a disability. This problem requires a software solution since the hardware peripherals required to implement a sign language neural network are already present on modern day smartphones.

Aim

The main goal of this project is to establish a real time sign language classification neural network by using a regular off the shelve camera and microcontroller with menial processing power such as a Raspberry Pi Zero. This system will be a proof of concept and our network will receive input that will need to be classified between 6 different trained sign language words/phrases i.e., 'Hello', 'Thank You', 'Good Job', 'No', 'Yes', and 'I Love You' in the American version of English signed language.

Methods

1. Theory

In machine learning image classification is assigning a class label to an image by prediction in this context and object localization is when a bounding box is drawn around one or more objects in an image. Together these two procedures are known as object recognition i.e. the sign that is expressed using our hands is the object that needs to be recognized and then matched to one of the trained classes that the network needs to assign the object to.

R-CNN

The R-CNN or Region Based Convolution Neural Network is a specific type of neural network that employs the 4 following steps – input image collection, convolution with kernel, pooling, and SVM. The process works by scanning the input image for possible objects and using an algorithm called Selective Search to generate region proposals. The convolution neural network is run on all these region proposals. Then the output of each CNN is fed into an SVM to classify the region and a linear regressor that tightens the bounding box around the object if an object was identified.

The main operator that is used in one type of image classification is convolution. Convolution is a multiplication of two arrays, they can be of different sizes but have to be of the same dimension and will produce another array of the same dimension. The 1st array is usually a matrix representation of our image, and the 2nd array is the kernel that is usually a smaller matrix that sweeps around the image matrix. The kernel determines characteristics such as overfitting if a large kernel is used, and other transformations that are determined by the kernel matrix form.

YOLO Model

The YOLO (You Only Look Once) Model was developed by Joseph Redmon involves the use of a single neural network to classify images. The Model works by first splitting the input image into a grid of cells, where each cell is responsible for predicting a bounding box if the center of a bounding box falls within the cell. Each grid cell predicts a bounding box involving the x, y coordinate and the width and height and the confidence. A class prediction is also based on each cell. This type of algorithm is known for being incredibly fast with a maximum of 45 frames per second which is enough data to convert a video into real time sign language detection.

2. Steps to train and test our Model

There are many steps that were followed to create a model that can detect sign language hand gestures in real-time. The project created the Model by using two different APIs, TensorFlow, and Yolo. The report gives detailed information about these APIs. To develop models, we had to follow the steps below. Figure 1 shows the state flow of how the models were created.

1. Data collection
2. Labeled the images to create the dataset for YOLO and TensorFlow.
3. Train the Model with the Labelled images with YOLO and TensorFlow.
4. Test the models.
5. Implement the Model on a microcontroller (Raspberry Pi).

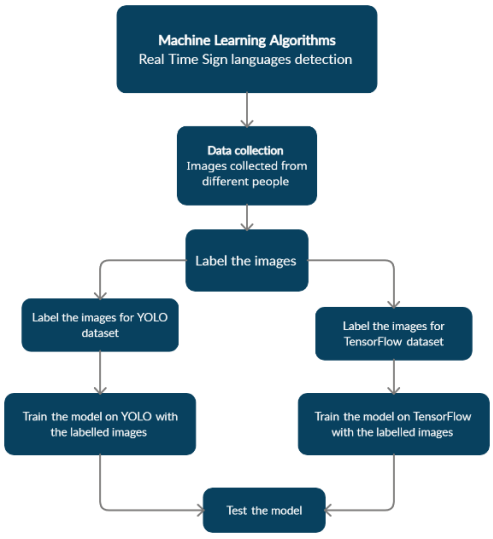

Figure 1: State flow of the models created

2.1 Data collection

For this project, we collected our own data in order to train and test the Model. This was done in order to ensure we can have a decent collection of data for our Model to be trained from. We collected approximately 500 images with four different people to have more diversity for our training our model. The pictures are taken by using a python programme that can collect multiple pictures using the PC webcam. The pictures are taken for 6 hand gestures (6 classes) for sign language: 'Hello', 'Thank You', 'Good Job', 'No', 'Yes', and 'I Love You'. Figure 2 shows an example of the images that was taken to create our own dataset for the project.

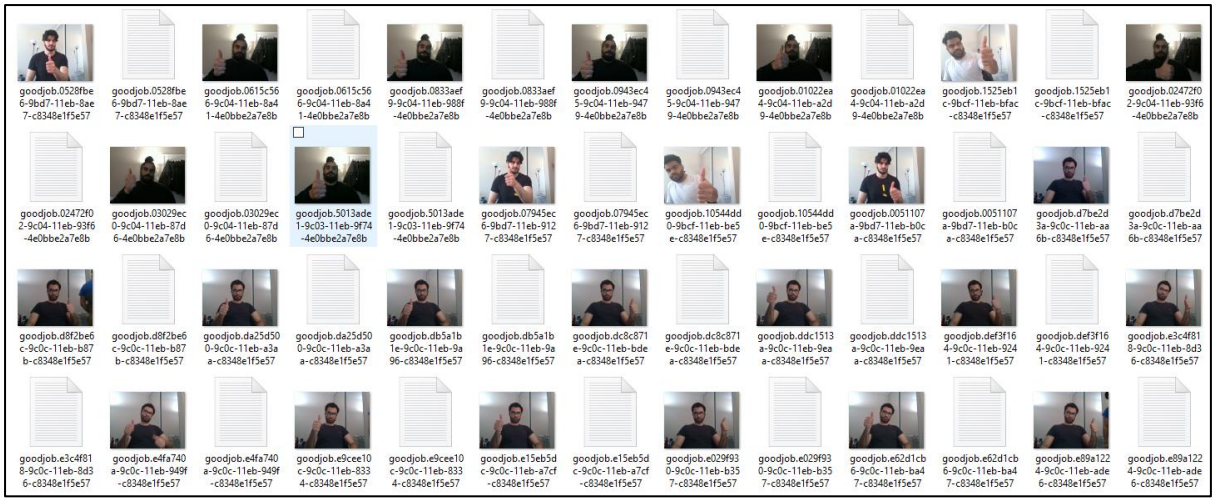

Figure 2: Collected images from multiple people for creating dataset

2.2 Labelling images

For Training the Model, we collected your own dataset. These images are labeled according to the API we are using. For labeling the data, we are using a premade labeling tool [7] to create the labeled file for our collected images, as shown in Figure 3. The tool creates files that will be used by the API to train the model to detect the object. YOLO and TensorFlow API both use different file types to be trained. Figure 4 shows the example of a few of the images that are being labeled to create training data set for our models. The steps for labeling the images are as follows.

1. Load the image.
2. Chose the file type you want to create with the images.
3. Draw a box around the part of the picture you want to label. That part of the image is the one you want the model to learn and get trained on detecting.

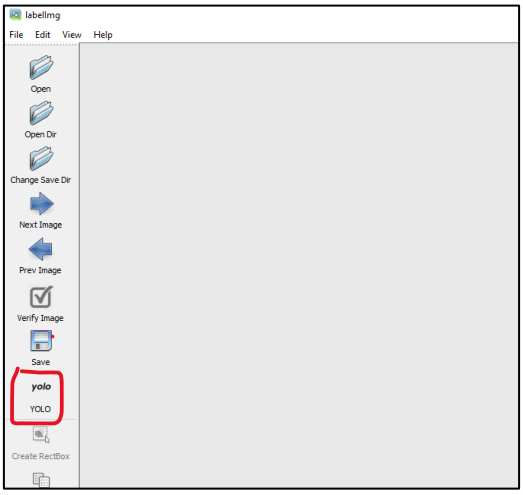

Figure 3: Image labeling tool [7]

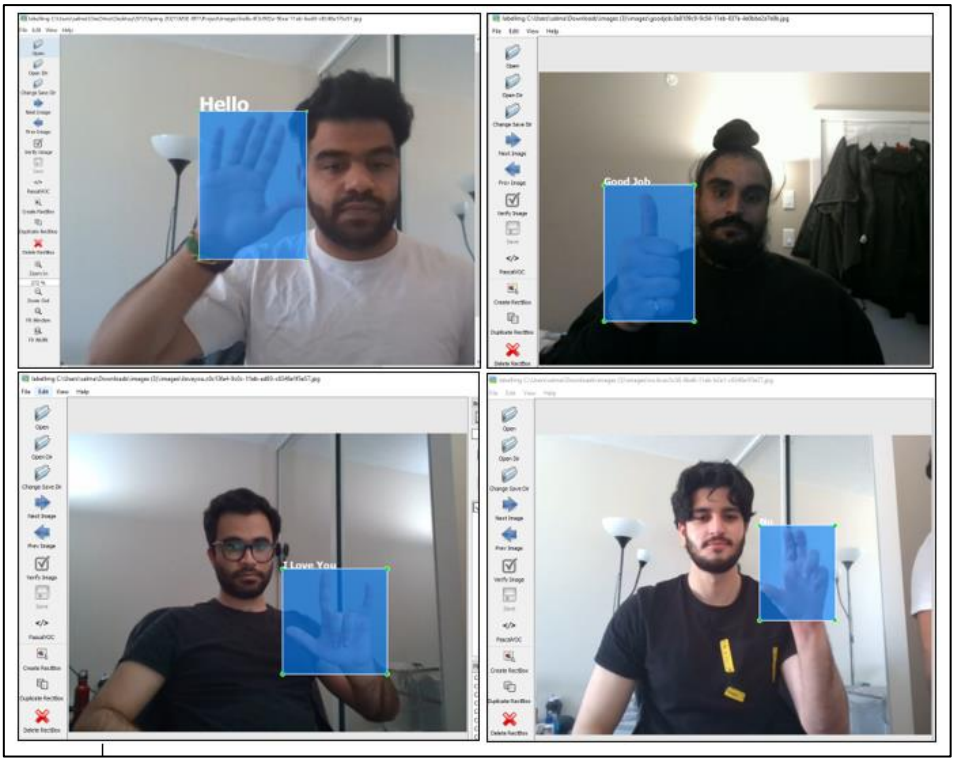

Figure 4: Example of labeled the images

2.3 Training the Model.

The model has been trained with two different APIs. The models were trained in google Collab due to the lack of hardware in our computers. The result section shows the result for both the Model and the demo that is provided with the with the report shows the result for the Model that is trained through YOLO.

2.3.1 With TensorFlow

For training the Model with TensorFlow, we used the TensorFlow library.

1. Installing the TensorFlow library
2. Then we used the TensorFlow transfer learning code from "Nicknochnack" [5][6] to train our model with the dataset. The code files are provided with the project report.
3. Train the data for 45000 iterations on our dataset.
4. We finally tested our Model with the real-time detection code from the sources. The code is using OpenCV to detect in real-time.

2.3.2 With Yolo

In this, we trained our Model with YOLO API. The steps that were followed to train the Model with our dataset are as follow:

1. Connected the google collab network with personal google drive.
2. Download the YOLO source darknet from and saved it in google drive to install all the required libraries for using YOLO [3] [4]. As shown in the picture below:

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

    #!git clone 'https://github.com/AlexeyAB/darknet' '/content/drive/MyDrive/yolo_custom_model_Training/darknet'
```

Figure 5: Source code for YOLO

3. Trained the model with the dataset we collected and labeled.
4. We use a similar OpenCV real-time detection source code to test our Model. The code is attached with this report.

Results and Discussions

Results from the TensorFlow model

After the model was trained, we run detection code which is using OpenCV. The code opens the webcam application which is integrated with the code and the train model to show the object, in our case the sign language hand gesture. It draws a box around the signs, which displays the name of the class and its percentage of accuracy. Figure 6 shows the screenshot of the result we got from TensorFLow model. In our case we were able to detect all the 6 classes that we trained the model on.

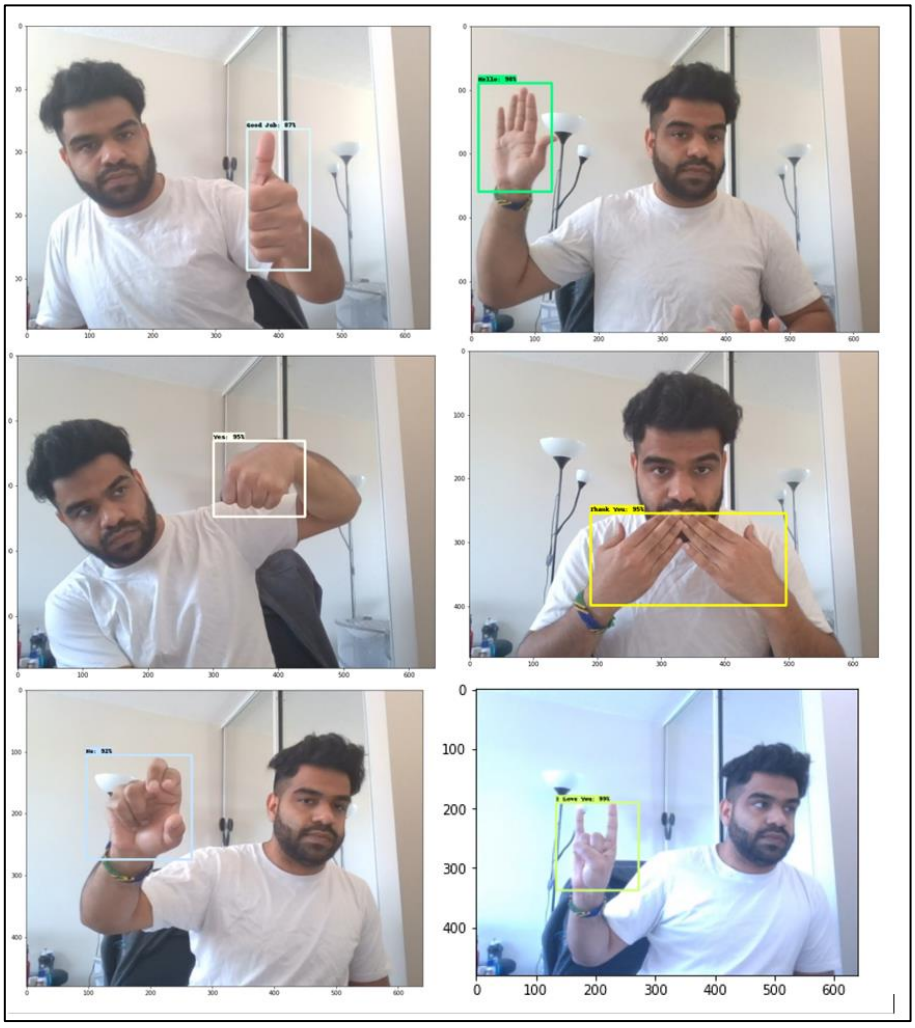

Figure 6: Results for TensorFlow model

Results from Yolo Model

We trained our model through a YOLO API with the dataset that was collected. We run a similar detection code in YOLO to test our model. Figure 7 shows the screenshot of the result that was detected. Our model was able to detect all 6 classes very accurately with multiple test subjects.

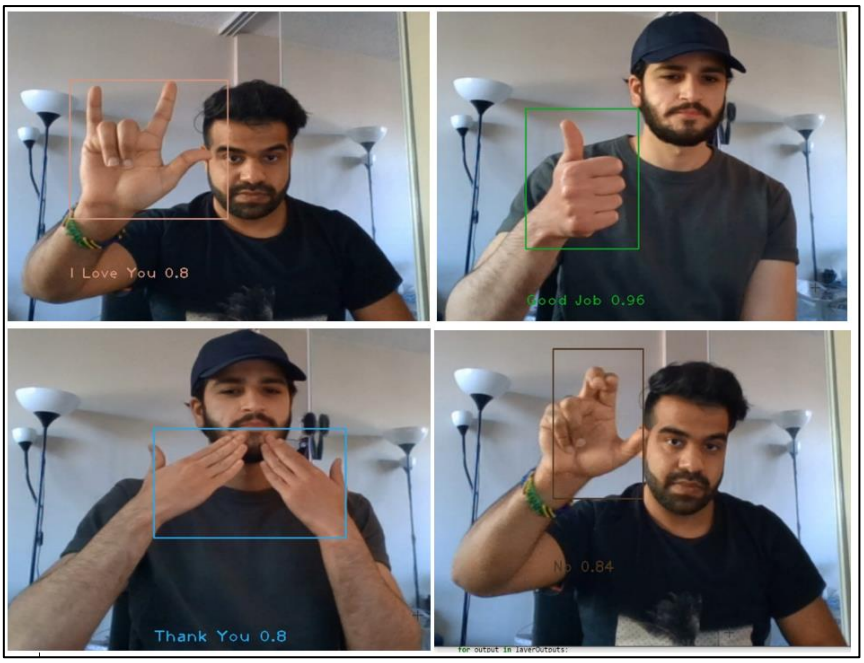

Figure 7: Results from YOLO model

Loss versus iteration for YOLO

Figure 8 shows the curve of loss versus iteration. As you can see in the plot, when the model started to train the loss is high but as the iteration for training increases, the loss drop drastically. The model was run for around 6000 iterations.

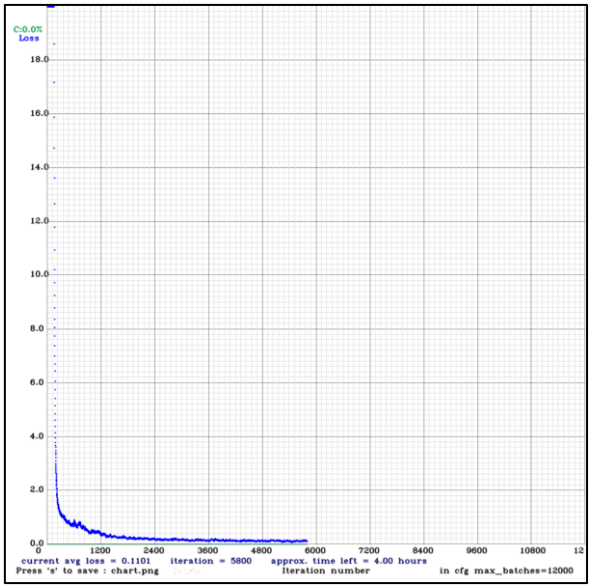

Figure 8: Loss vs iteration curve for YOLO

Yolo and TensorFlow result comparison

During testing of the two models, it was identified that the Yolo model produced must faster results than the Tensorflow API. This is because of the additional neural networks that are in Tensorflow that add to the computational overhead. The Yolo model can deliver more than 24 frames per second when used on a locally trained system that does not rely on a server for computation. Tensorflow is suitable for images where the dataset is not as abundant, and so this model needs to apply additional transformation to the image matrix in order to comprehensively study its characteristics.

Tested the Model on Raspberry Pi 4 with Pi Camera

The process of running the model on the Raspberry Pi involves running the python code on the NOOBs operating system which is based on the Linux kernel. This kernel allows us to install python in addition to the other dependencies required to run the Yolo model such as compiling and installing the OpenCV library from the source. Once all the dependencies are installed, a weights file needs to be exported from the trained model. This is a file that allows for a faster object detection when used on an embedded system since it removes the extra computational overhead that would have been required during a normal training and testing of a machine learning model.

As shown in the demonstration video, the Raspberry Pi 4 was able to handle the very resource demanding process of streaming video from its camera, breaking these images into arrays and then using a neural network to compute the result of the object detection. This result still has room for improvement as the frames per second were close to 2. Ideally at least 24 frames per second is needed as this is the point where the human eye is at its minimum threshold for smooth video viewing.

Figure 9: Raspberry Pi setup with a camera

Conclusion

Machine learning has paved the way for novel solutions and implementations that have taken out the strenuous part of analysing and characterising a system before trying to implement a solution to the system by using the computational power of computers to learn and study the data given to it to accomplish a measurable goal. By using machine learning in a difficult application such as hand gesture recognition where a lot of frames of data need to be studied and analyzed, we can quickly produce a system that can track any type of image stream and study it for the classes it has been trained for. Such a task using conditional logic would not be feasible at all. Additionally, there are nuances that need to be considered when using machine learning as it is very computationally expensive hence the product needs to be considered.

References

[1] R. Khandelwal, “Data Augmentation techniques in Python,” *Medium*, 11-Dec-2019. [Online]. Available: https://towardsdatascience.com/data-augmentation-techniques-in-python-f216ef5eed69. [Accessed: 09-Mar-2021].

[2] J. Redmon, *YOLO: Real-Time Object Detection*. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 17-Mar-2021].

[3] *Youtube.com*, 2021. [Online]. Available: https://www.youtube.com/watch?v=hTCmL3S4Obw&t=762s&ab_channel=JayBhatt. [Accessed: 22- Apr- 2021].

[4]"jakkcoder/training_yolo_custom_object_detection_files", *GitHub*, 2021. [Online]. Available: https://github.com/jakkcoder/training_yolo_custom_object_detection_files. [Accessed: 22- Apr- 2021].

[5] *Youtube.com*, 2021. [Online]. Available: https://www.youtube.com/watch?v=yqkISICHH-U&t=212s&ab_channel=NicholasRenotte. [Accessed: 22- Apr- 2021].

[6]"nicknochnack/TFODCourse", *GitHub*, 2021. [Online]. Available: https://github.com/nicknochnack/TFODCourse. [Accessed: 22- Apr- 2021].

[7] https://tzutalin.github.io/labelImg/