Hand Gesture Recognition using RF-Sensing

(SCSE20-0528)

Submitted by: Tan Jun Hao
Matriculation Number: U1821560G

Supervisor: A/P Luo Jun
Examiner: Douglas Leslie Maskell

School of Computer Science and Engineering

A final year project report presented to the Nanyang Technological University
in partial fulfilment of the requirements of the degree of
Bachelor of Engineering (Computer Engineering)

2021

Abstract

Hand gesture recognition is the interpretation of hand gestures using mathematical algorithms. There are many forms of hand gestures, but hand sign gestures will be the focus for this project. Ultimately, this project aims to find out if hand signs gestures can be recognized using deep supervised learning algorithm. It will be able to build a new form of communication between humans and machines and it might even make the keyboard and mouse obsolete in the future.

This project consists of gathering of data of different hand gestures from the radar sensor and implementation of a neural network model to able to classify different hand sign gestures. The model is programmed in Python using the PyTorch library and deep learning algorithm called ResNet and achieved an accuracy of 87.4%.

Acknowledgements

First of all, I would like to express my great gratitude to Associate Professor Luo Jun for his continuous support and guidance for the duration of the project. His suggestions have helped me a lot though this journey and I am deeply grateful.

I would also like to thank my examiner, Associate Professor Douglas Leslie Maskell for his time in reviewing this report and grading this project.

Lastly, I would like to thank the School of Computer Science and Engineering for giving me this opportunity to work on this project.

Table of Contents

Acronyms

CNN	Convolutional Neural Networks
IR-UWB	Impulse Radio Ultra-Wide Band
PRI	Pulse Repetition Interval
CW	Continuous Wave
FMCW	Frequency Modulated Continuous Wave
ADAM	Adaptive Moment Estimation
SGD	Stochastic Gradient Descent
GPU	Graphic Processor Unit
CPU	Computer Processor Unit

List of Figures

List of Tables

1. Introduction

1.1 Motivations

Gestures are originated from any body parts and these gestures can help be a form of communication between humans and machine. There are many studies on gesture recognition such as emotion, hand or even posture recognition. The goal of gesture recognition is to be able to use mathematical algorithms to interpret the human gesture so that users can control and interact with devices without physically touching them. In the future, we might even make the keyboard and mouse obsolete when there are more accurate ways of recognizing gestures. In this project we will focus on hand sign recognition.

As for hand gesture recognition, majority is done using camera-based recognition, but it has a major drawback which is lack of privacy [1]. Another method is using wearable devices such as gloves. This method can get an accuracy of up to 95% however the gloves are causing discomfort to the users [1]. Unlike the previous methods, radar-based recognition can not only solve the disadvantage mentioned above but is also able to excel in environment with different lighting conditions.

Neural networks have been a rising star in the machine learning field due to their popularity in their performance in pattern recognition. Pattern recognition is not only limited to images, but it also includes audio and text. However, there are many considerations involved and it is not easy to come up with a deep model with high accuracy.

In this project, we will explore dynamic motion of hand sign gestures as shown in Figure 1 with the use of deep supervised learning classification method – CNN to correctly detect the different gestures.

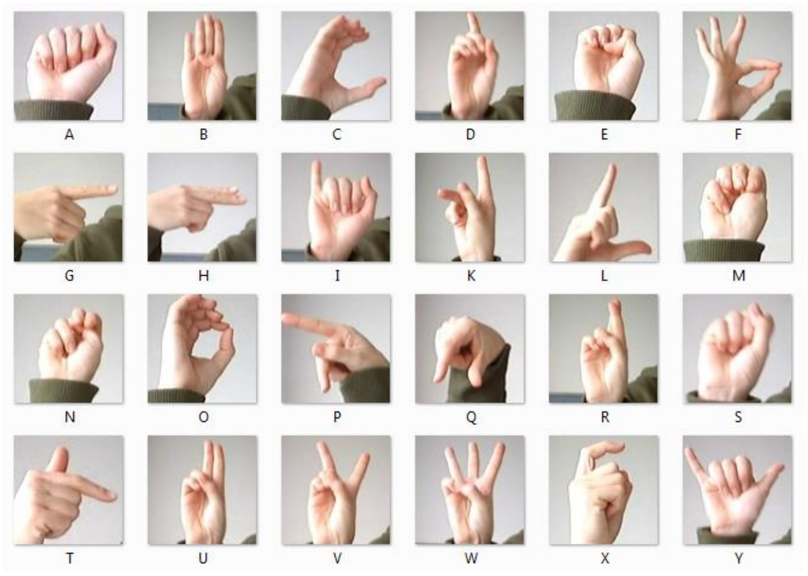


Figure 1: American Sign Language

1.2 Objectives and Scope

The first half of the project consist of gathering data from the IR-UWB sensor using Raspberry Pi and sending it over to the local computer for data processing.

The second half of the project consist of implementation of ResNet to get a high accuracy for detecting and classifying the hand gestures. This is done by using Python 3.9.2 and PyTorch library.

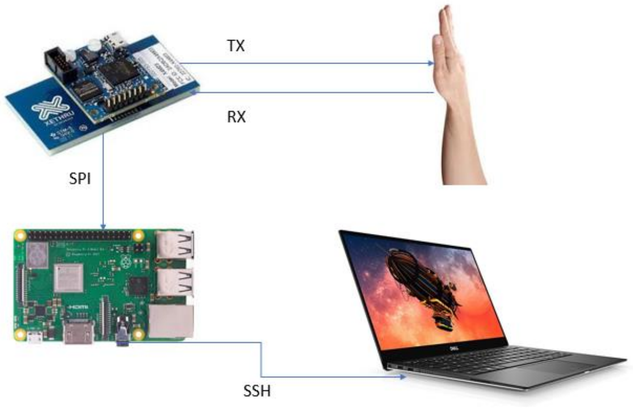


Figure 2: Data Collection Flow

1.3 Organisations

The report is organised into the following sections:

- Chapter 2: Literature Review for Radar and ResNet
- Chapter 3: Method and Implementation (Data Collection, Model Implementation and Training of model)
- Chapter 4: Conclusion

2. Literature Review

2.1 Radar Signal

Radar is widely known as a target detection technology used in military. Radar is an acronym for Radio Detection and Raging and as the name suggests, it make use of radio waves to detect objects. The radar sensor consists of transmitter and receiver and to determine parameters such as range and velocity of the object.

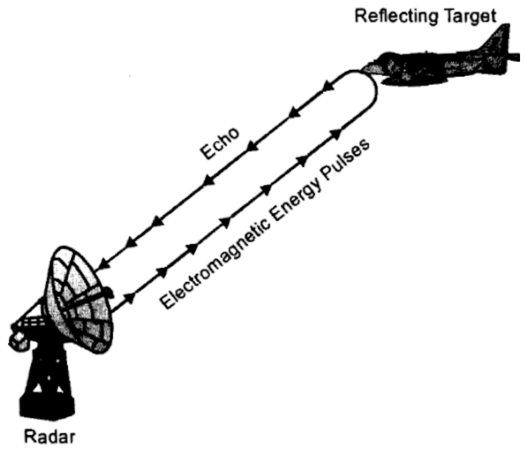


Figure 3: Radar

Radar sends out pulses repeatedly and waits for the pulse to be received before sending the next one. The rate of pulse to be emitted can be called Pulse Repetition Interval (PRI). The data is received in a 2D matrix and each time the radar receives a new pulse, the data is recorded and added to a new row of matrix [2]. With reference to Figure 4, the Fast time is the measurement of data at each pulse and the Slow time is the number of pulses receive. If the object is moving away as shown by the green boxes, the time taken for the pulse to be received is increasing. Slow time just updates every PRI. If there are multiple channels, the matrices are stacked together to form a data cube.

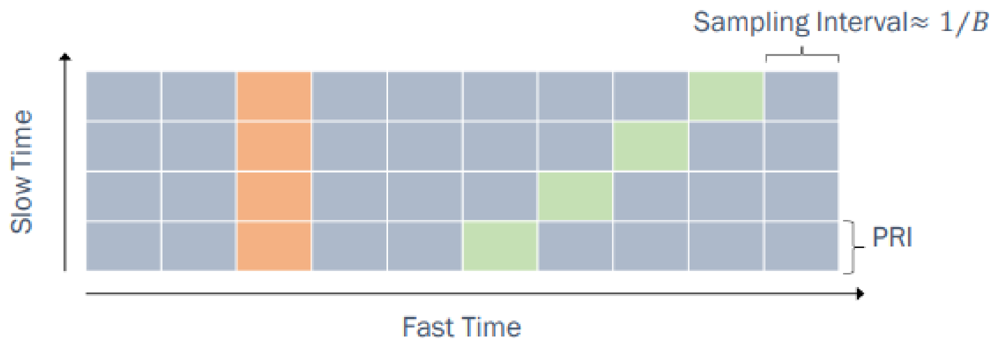


Figure 4: Fast and Slow Time

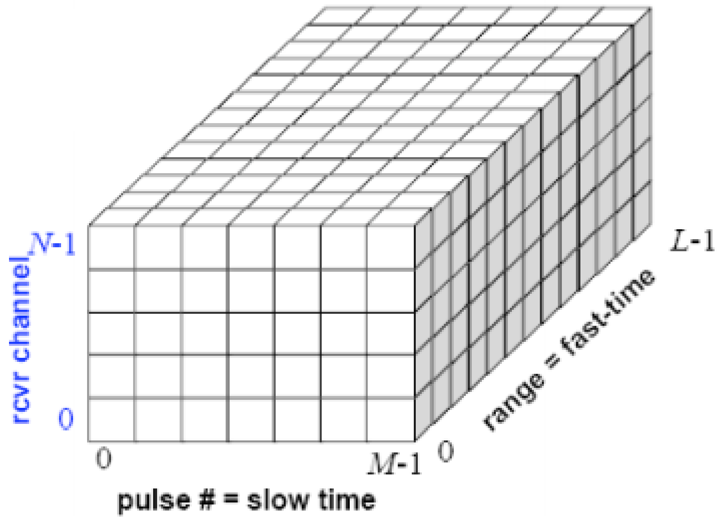


Figure 5: Data Cube

2.2 Different Radar Sensors

The three signals that can be emitted are the Continuous Wave (CW), the Frequency Modulated Continuous Wave (FMCW) and the Impulse Radio Ultra-Wide Band (IR-UWB). A radar using CW transmits a linearly modulated CW and calculates the distance based on the difference between the transmitted and received signal frequencies. FMCW on the other hand uses frequency modulation, where the carrier frequency changes over time. The biggest disadvantage is that CW and FMCW radars are more expensive than IR-UWB radars and slow data acquisition. IR-UWB radars, on the other hand, has a high range resolution, and it is less subjected to outside interference [3].

Hence, the hardware for this project is the Novelda Xethru X4 UWB pulsed radar.

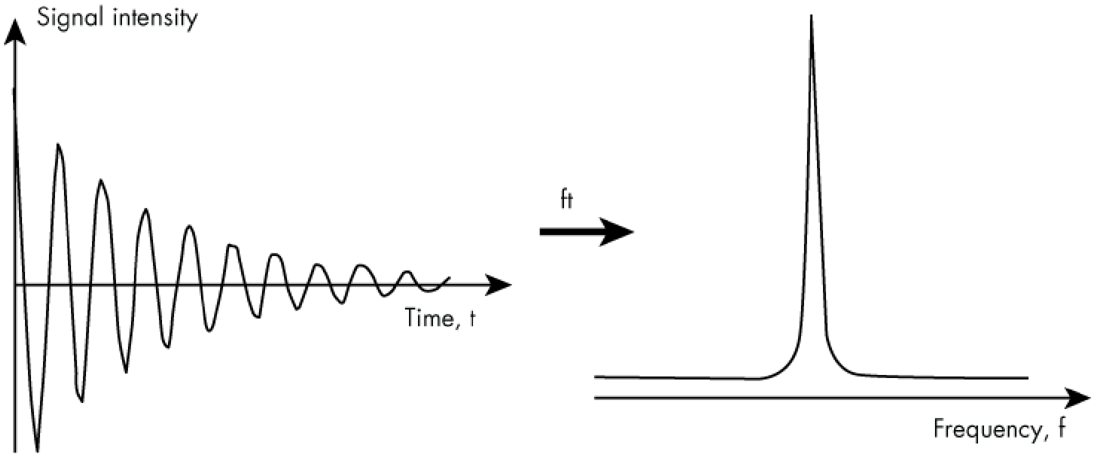


Figure 6: Left: CW Right: Pulse

2.3 ResNet

ResNet, also known as Residual Network [4], and has CNN as a building block. CNN are design from the reference of a biological neural networks. It consists of convolutional, pooling and fully connected layers. The convolutional layer applies convolutional filters to the input and passes it to the next layer. It learns the features without losing information. The pooling layer reduces to a spatial dimension of a network and it can result in a lower chance of overfitting. Fully connected layers help to provide the activation values to the features.

There are many impressive deep learning neural networks like the winner of the ILSVRC Challenge, AlexNet,ZFNet and GoogLeNet in 2012, 2013 and 2014 respectively. However, those algorithms are unable to deal with the phenomenon shown in Figure 7. The performance for a deeper layer network (56-layer) has a worse performance compared to a shallower network (20-layer). The winner of the 2015

ILSVRC Challenge, He Kaiming, from Microsoft Research has created ResNet to deal with the problem. The main difference is that the ResNet has a residual block (Figure 8). The main job of the block is to take the output from the previous layer to the layer ahead. This results in an increased performance compared to the plain network (Figure 9) and hence ResNet is used in the project.

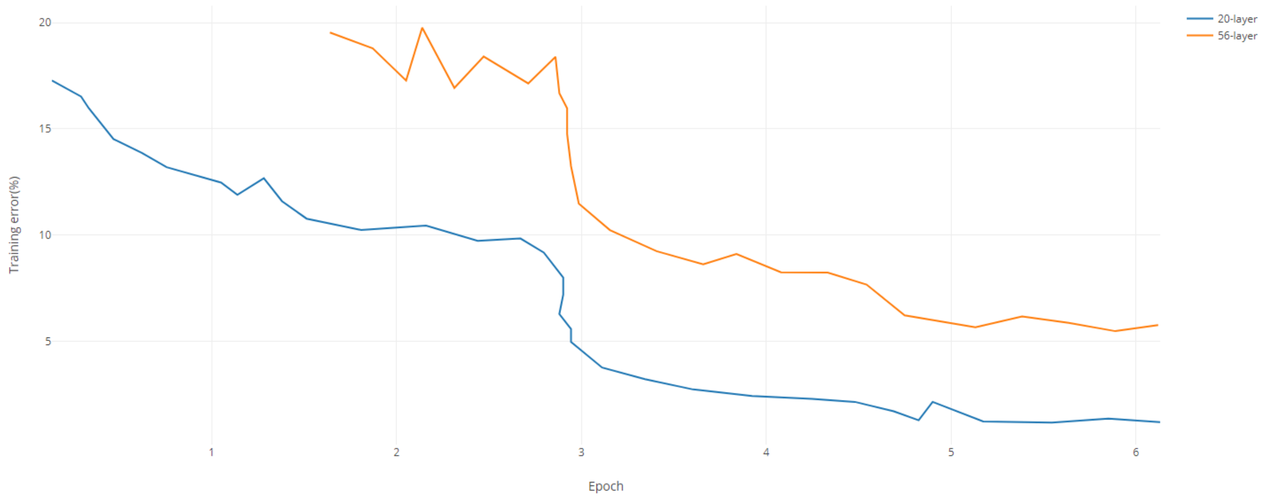


Figure 7: Performance Of 20-layer And 56-layer network

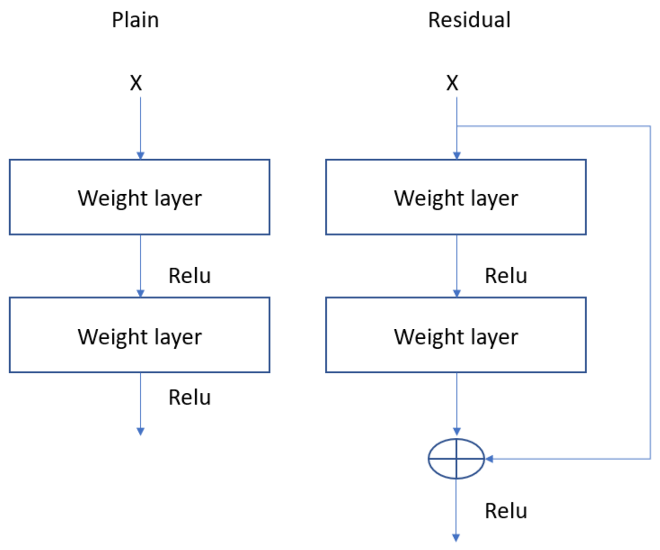
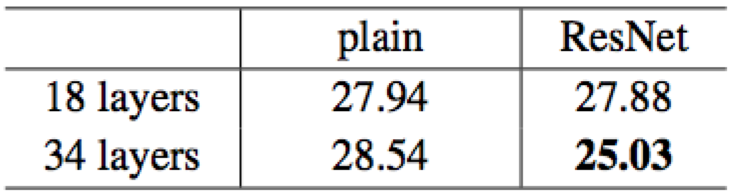


Figure 8: Plain vs Residual Block



	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Figure 9: Result - Plain vs ResNet

3.Implementation

3.1 Data Collection

The data collection procedure takes place in an environment with minimal external movement to prevent the noise from affecting the result. The experiment set up is shown in Figure 4. The palm is facing the sensor, which is 30 cm away and the position is called the starting gesture. The time is taken for each data to be 20 seconds long and the flow of events can be seen from Table 1. The reason why there is a 5 second buffer before and after the transition is to make sure that there is enough data to process and eliminate errors such as human errors when starting and stopping the experiment. 100 samples are collected for each gesture for deep learning. Each sample is saved into csv file.

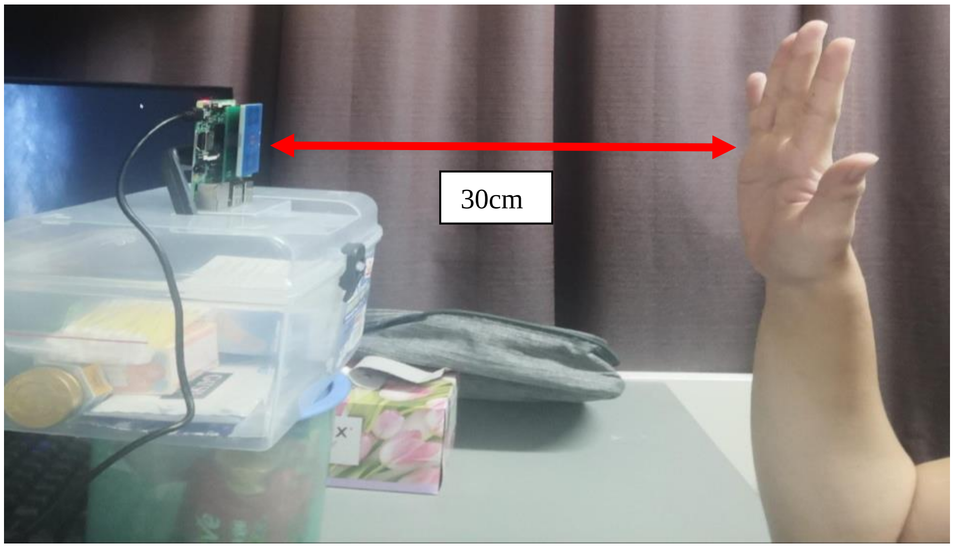


Figure 10: Experiment Setup

Time	Gesture
0 – 5s	Starting Gesture
5 – 15s	Transition to Gesture X
15 – 20s	Gesture X

Table 1: Flow Of Data Collection Event

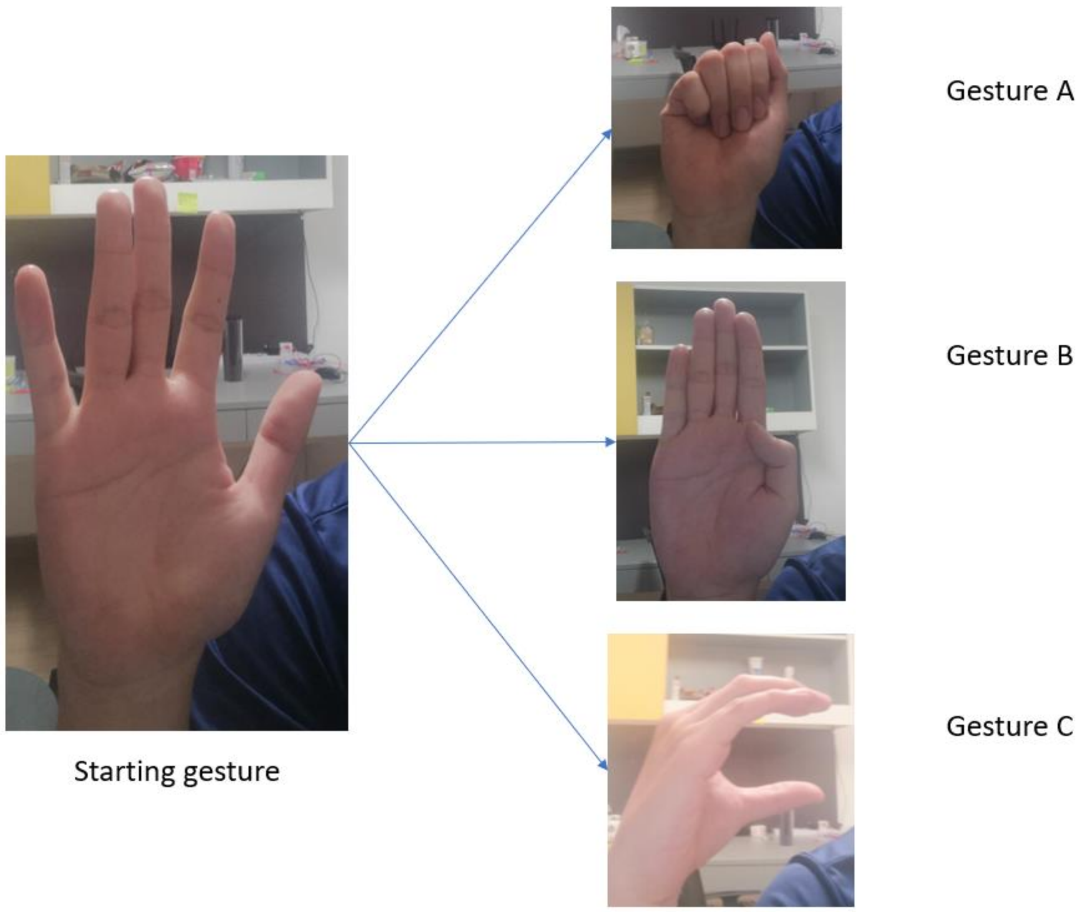


Figure 11: Starting To Gesture A, B, C

The following are the hardware and software needed for this part:

Hardware:

1. Raspberry Pi
2. Xethru X4 radar sensor

Software

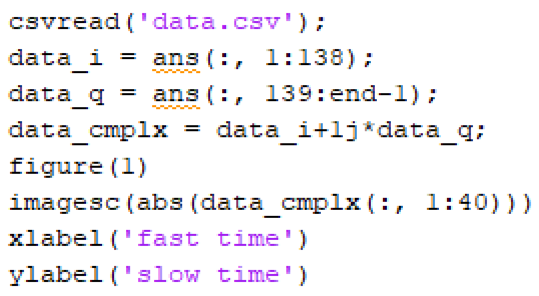
1. Raspbian OS
2. MobaXterm 20.3
3. MATLAB

MobaXterm is a free application for Windows that is used for remote computing. It can establish a stable SSH connection and transfer files from the remote computer. With this software, all the data samples collected is then transferred to another computer with larger RAM space and processor for deep learning.

MATLAB is used to visualize the data to make sure that data has similar radar images for the same gesture (low intra-class distance) and dissimilar radar images for different gesture (high inter-class distance).

Step for plotting a spectrogram of the data:

1. Read CSV file
2. Split the data into in-phase (real) and quadrature (imaginary) components
3. Add both components
4. Plot the spectrum with the absolute value



```
csvread('data.csv');
data_i = ans(:, 1:138);
data_q = ans(:, 139:end-1);
data_cmplx = data_i+1j*data_q;
figure(1)
imagesc(abs(data_cmplx(:, 1:40)))
xlabel('fast time')
ylabel('slow time')
```

Figure 12: MATLAB code for plotting data

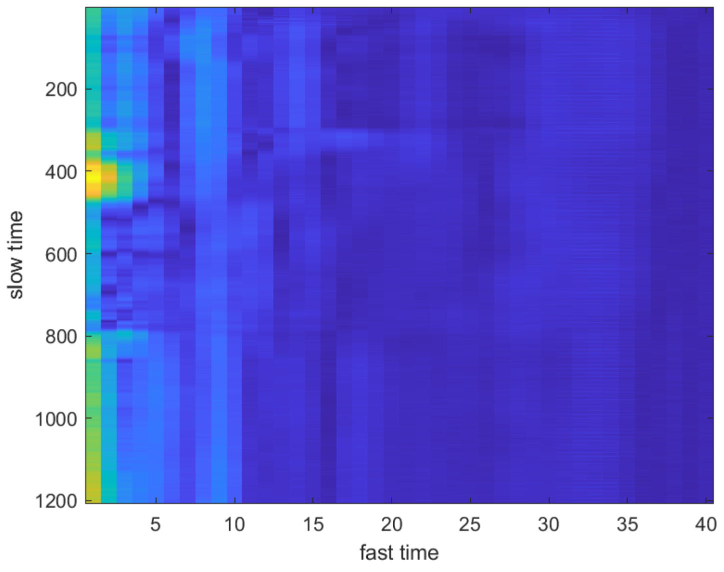


Figure 13: Spectrogram Of Gesture A

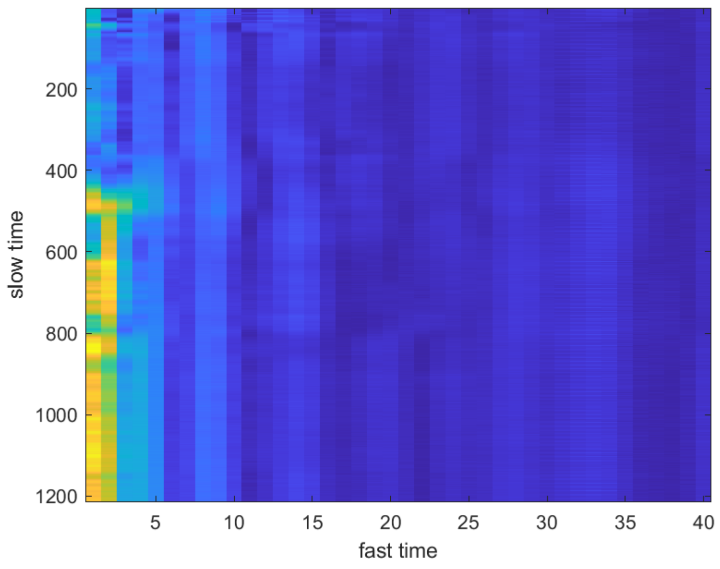


Figure 14: Spectrogram Of Gesture B

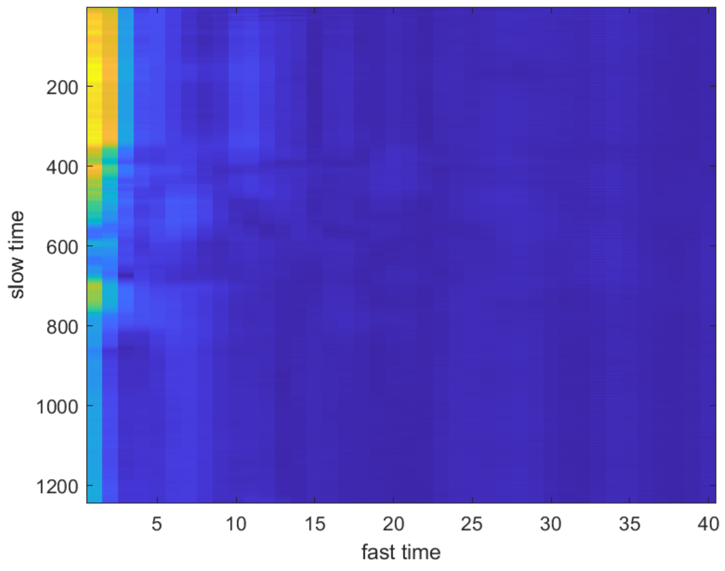


Figure 15: Spectrogram Of Gesture C

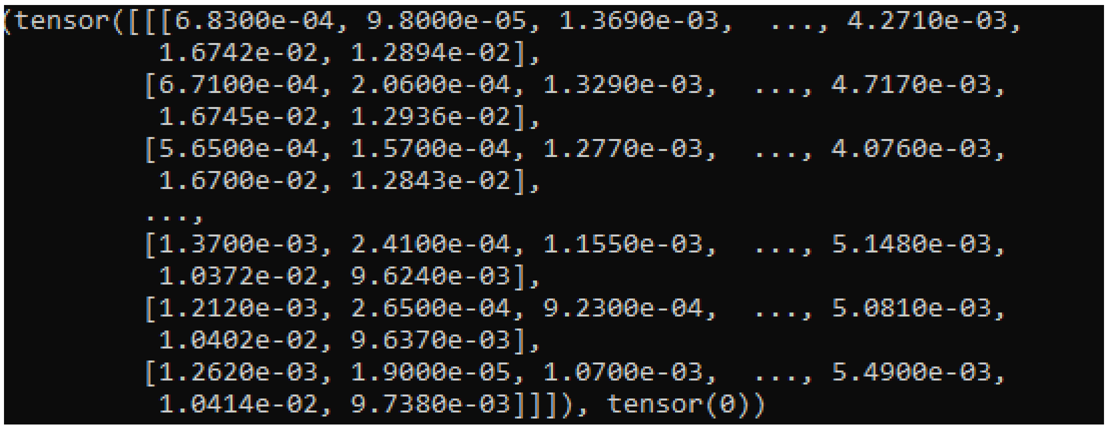
The files are also saved in their class labels so it will be easier in the next step which is loading and preparing of the data. (eg Gesture A will be A.csv)

3.2 Gesture Recognition

3.2.1 Loading and Preparing of data

Before creating a model for deep learning, the first step is to load and prepare the input for the model. To achieve that, Dataset class and Dataloader class from the torch library is used.

There are 2 types of dataset: map-style and iterable-style which the former was chosen since it can give the size ahead of time which allows easy parallel loading. CSV parsing and converting the data to tensor is done in the Dataset class. The class returns [data, label]. The size of data is 1 × 950 × 276. The data was 950 × 276 but the input needed to be 3D and hence the unsqueeze function is used to convert the data into a 3D data. The data is converted into tensor before passing it into the neural network.



```
(tensor([[[6.8300e-04, 9.8000e-05, 1.3690e-03,  ..., 4.2710e-03,
          1.6742e-02, 1.2894e-02],
         [6.7100e-04, 2.0600e-04, 1.3290e-03,  ..., 4.7170e-03,
          1.6745e-02, 1.2936e-02],
         [5.6500e-04, 1.5700e-04, 1.2770e-03,  ..., 4.0760e-03,
          1.6700e-02, 1.2843e-02],
         ...,
         [1.3700e-03, 2.4100e-04, 1.1550e-03,  ..., 5.1480e-03,
          1.0372e-02, 9.6240e-03],
         [1.2120e-03, 2.6500e-04, 9.2300e-04,  ..., 5.0810e-03,
          1.0402e-02, 9.6370e-03],
         [1.2620e-03, 1.9000e-05, 1.0700e-03,  ..., 5.4900e-03,
          1.0414e-02, 9.7380e-03]]]), tensor(0))
```

Figure 16: Example of one data

The data is split into training and test set so that we are able to see if the model overfits when the test performace is much worst than the training. Figure 17 is an example of the performace when the model is overfitted. The dataset is split to 80% training and 20% test. The most important factor to consider is how the split function worked, as out of the

400 datasets, they had to be similarly distributed across the training and test set(Table 2). Even though it is not exactly 80-20, the deviation is quite small.

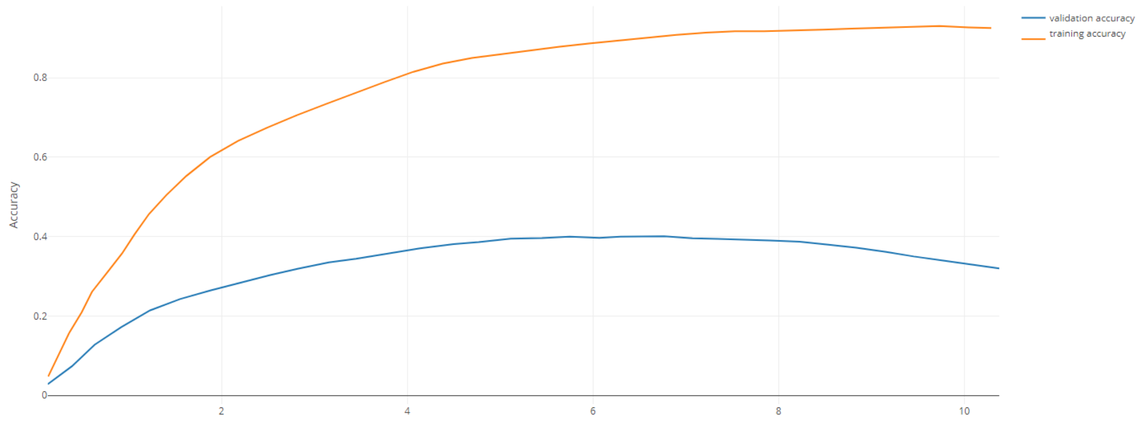
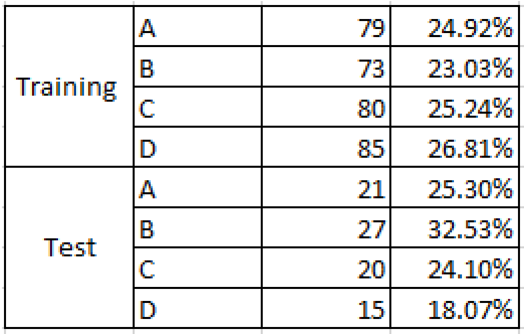


Figure 17: Overfitting



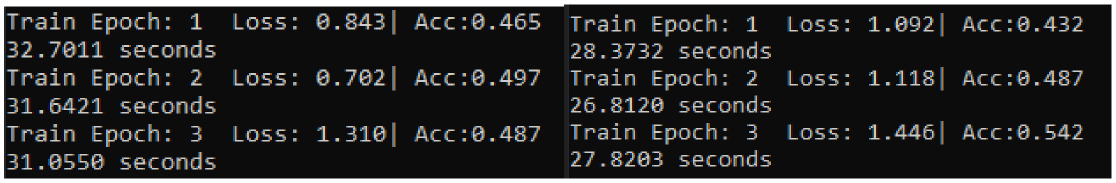
Training	A	79	24.92%
	B	73	23.03%
	C	80	25.24%
	D	85	26.81%
Test	A	21	25.30%
	B	27	32.53%
	C	20	24.10%
	D	15	18.07%

Table 2: Distribution Of Gestures For Training And Test Set

Library imported	Description
torch.utils.data.dataset	Reading the dataset from the path given
torch.utils.data.dataloader	Provide the model with batches of data (specify by user) instead of whole dataset for computation speed up
Torch.utils.data.random_split	Split the data into training and test set
pandas	Import CSV

Table 3: Libraries Needed For Loading Dataset.

One important hyperparameter to tune is the batch size as it affects the memory used for the computation and the speed of each epoch. As you can see from Figure 18 the time taken has drop from an average of 32 seconds to 27 seconds when the batch size is increased.



```
Train Epoch: 1  Loss: 0.843| Acc:0.465
32.7011 seconds
Train Epoch: 2  Loss: 0.702| Acc:0.497
31.6421 seconds
Train Epoch: 3  Loss: 1.310| Acc:0.487
31.0550 seconds
```

```
Train Epoch: 1  Loss: 1.092| Acc:0.432
28.3732 seconds
Train Epoch: 2  Loss: 1.118| Acc:0.487
26.8120 seconds
Train Epoch: 3  Loss: 1.446| Acc:0.542
27.8203 seconds
```

Figure 18 Left Batch Size=1 Right Batch Size =5

3.2.2 Model Implementation

For the model, ResNet is used. The main benefit of ResNet is alleviate the problems caused by deeper networks. Res18 and Res50 is being used due to the 2- and 3-layers residual block, respectively. With multiple testing, Res50 gives the best results with the limited data.

Adaptive Moment Estimation (ADAM) is chosen to be the optimizer due to the ability to implement adaptive learning rate for each of the parameters so that we do not have to manually adjust compared to Stochastic Gradient Descent (SGD). For example, if the gradient is small, the learning rate is increased for the corresponding parameters [5]. Since this is a classification project, Cross Entropy Loss is used for the loss functions [6].

Layer Name	Res50
Con1	7×7, 64, stride = 2
Pool1	3×3, max pool, stride = 2
Conv2_x	$\begin{bmatrix} 1\times1, 64 \\ 3\times3,64 \\ 1\times1,256 \end{bmatrix} \times 3$
Conv3_x	$\begin{bmatrix} 1\times1, 128 \\ 3\times3,128 \\ 1\times1,512 \end{bmatrix} \times 4$
Conv4_x	$\begin{bmatrix} 1\times1, 256 \\ 3\times3,156 \\ 1\times1,1024 \end{bmatrix} \times 6$
Conv5_x	$\begin{bmatrix} 1\times1, 512 \\ 3\times3,512 \\ 1\times1,2048 \end{bmatrix} \times 3$
	Global average pool + 1000 fully connected with softmax

Table 4: Res50 Architecture

Library imported	Description
torch.nn	This library contains all the neural network blocks like convolutional layers to pooling layers
Torch.optim	Consist of the optimizer building block

Table 5: Libraries For Model Implementation

3.2.3 Training of model and results

While training a model, there are 3 hyperparameters that we change and modify to get better results (high accuracy, low loss). The parameters are epoch, learning rate and batch size. Epoch is number of times the entire dataset passes forward and backward, learning rate is the amount of parameters being updated and the batch size is the number of training sample in a batch. For the first few trainings, we start off with a low epoch of 5, learning rate of 0.01, batch size of 15 and we can see that that we are only able to get an accuracy of 55%. To achieve high accuracy, we deal with epoch first while keeping the other 2 hyper parameters constant.

As a result, as the epoch increases, the accuracy also increases. However, if the epoch is too high, the model will be overfitted even though the training accuracy is high. This is when the test accuracy comes in. If the test accuracy does not increase, any epoch higher than the current value will be deemed as overfitting.

Epoch	Accuracy	Loss
10	62.5%	0.5
20	79.2%	0.4
30	85%	0.29

Table 6: Results From Different Epoch

Next, we will be changing the learning rate from 0.1 to 0.001. As the learning rate decreases, the accuracy increases because the loss is decreasing much more when the learning rate is smaller.

Learning rate	Accuracy	Loss
0.1	52.2%	0.74
0.01	58.3%	0.76
0.001	66.6%	0.47

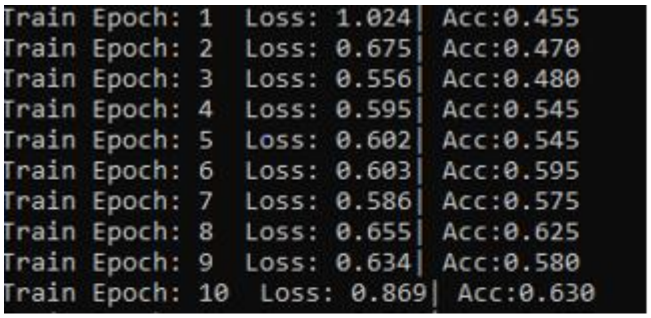
Table 7: Results From Different Learning Rate

Lastly, we will see how the batch size affects the accuracy. Increasing the batch size does affect the accuracy by a little but the not so much for the loss. However, the time taken has reduced for each epoch reduced when the batch size increases. Due to the memory constraint, the batch size is capped at 25.

Batch size	Accuracy	Loss
5	54%	0.83
15	60.8%	0.66
25	62%	0.74

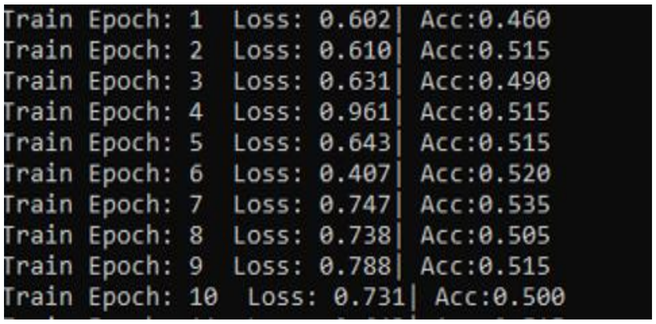
Table 8: Results From Different Batch Size

Other than the 3 hyper parameters, the different residual network are also tested to check for which network yield better results. Res18 has a lower accuracy compared to Res50 and it could be due to the shallower network that the Res18 have compared to Res50.



```
Train Epoch: 1  Loss: 1.024| Acc:0.455
Train Epoch: 2  Loss: 0.675| Acc:0.470
Train Epoch: 3  Loss: 0.556| Acc:0.480
Train Epoch: 4  Loss: 0.595| Acc:0.545
Train Epoch: 5  Loss: 0.602| Acc:0.545
Train Epoch: 6  Loss: 0.603| Acc:0.595
Train Epoch: 7  Loss: 0.586| Acc:0.575
Train Epoch: 8  Loss: 0.655| Acc:0.625
Train Epoch: 9  Loss: 0.634| Acc:0.580
Train Epoch: 10  Loss: 0.869| Acc:0.630
```

Figure 19: Res50



```
Train Epoch: 1  Loss: 0.602| Acc:0.460
Train Epoch: 2  Loss: 0.610| Acc:0.515
Train Epoch: 3  Loss: 0.631| Acc:0.490
Train Epoch: 4  Loss: 0.961| Acc:0.515
Train Epoch: 5  Loss: 0.643| Acc:0.515
Train Epoch: 6  Loss: 0.407| Acc:0.520
Train Epoch: 7  Loss: 0.747| Acc:0.535
Train Epoch: 8  Loss: 0.738| Acc:0.505
Train Epoch: 9  Loss: 0.788| Acc:0.515
Train Epoch: 10  Loss: 0.731| Acc:0.500
```

Figure 20: Res18

With the accuracy of 55%, more data for each gesture has been collected to increase the accuracy. The accuracy has increased to 67% when data samples increase from 50 to 100. This is because the network can get more features rather than relying on weak correlation when there is limited data.

Data Augmentation is a method to increase the diversification of data when there is limited data. The data is being transformed into different formats but while retaining the features. Some techniques are changing the brightness, rotating, and resizing the images. Using the RandomAffine method from PyTorch library, we can do that to the data. Affine transform is a type of geometric transformation that preserves the center invariant. The transformation can consist of rotating the image (Figure 22, using the degree parameter), moving the image (Figure 23, using the translate parameter) and scaling of the image (Figure 24, using the scale parameter). The accuracy has increase from 75% to 78%. This method is usually done when there is no chance to collect more data and but when we still want to have a small boost of accuracy. Of course, collecting data would yield better results.

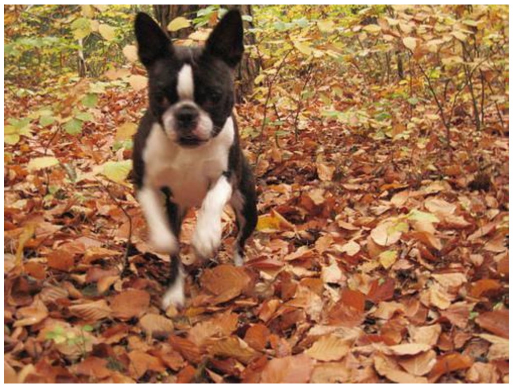


Figure 21: Original Image

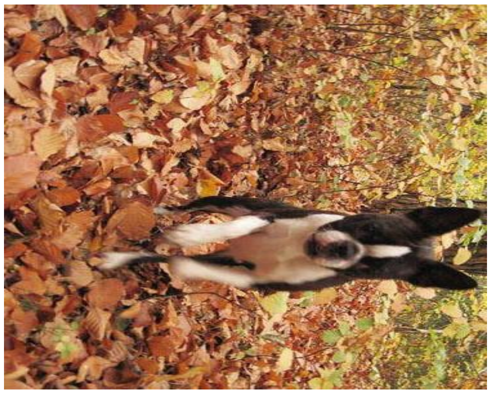


Figure 22: Rotation Of Image

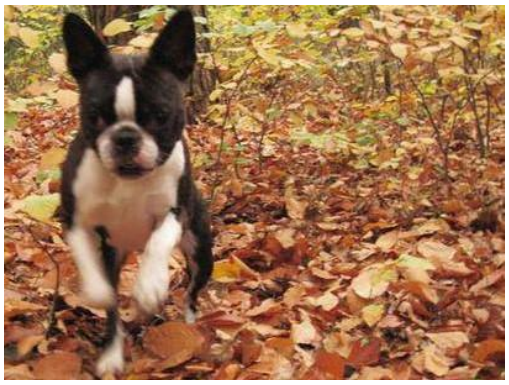


Figure 23: Shifting Of Image

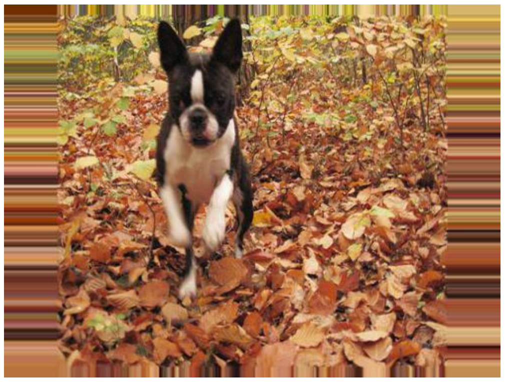


Figure 24: Scaling Of Image

Overall, to achieve the optimal values for the hyperparameters, different combinations of values were tried. With batch size = 10, epoch = 20, learning rate = 0.0005, the model can get a training accuracy of 87.4% and a test accuracy of 80.7% with 4 different gestures and 100 samples each [7]. Due to the memory constraint, the batch size cannot be further increased.

Chapter 4: Conclusion and Future Work

4.1 Conclusion

In this project, there were many challenges that were faced. From insufficient number of samples, to picking the optimal value for the hyper parameters. Data collection was the most important and challenging phase because it is a long and tedious process. There are times where the transition of hand gesture was too fast or the remote connection with the Raspberry Pi was lost halfway through and the collection of data must restart. There are a lot of parameter and model architecture to select from and choosing the right values takes multiple testing but however, with the help of a graphic processor unit (GPU), the time taken for each epoch was greatly reduced by more than 50 times as compared to a computer processor unit (CPU). The problems were addressed with trial and errors and extensive research. As a result, hand gesture recognition system with a training accuracy of 87.4% and a test accuracy of 80.7% with 4 different gestures (A, B, C, D) is successfully developed.

4.2 Recommendation in Future Work

- Reach more than 90% accuracy by increasing the number of datasets for each gesture
- Increase the number of gestures for recognition so that the model can recognize all the alphabets in sign language
- Implement real time recognition
- Collect data with different orientations so that it can detect the gestures from different angle

References

[1] S. Ahmed, K. D. Kallu, S. Ahmed, and S. H. Cho, “Hand Gestures Recognition Using Radar Sensors for Human-Computer-Interaction: A Review,” *MDPI*, 02-Feb-2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/3/527. [Accessed: 15-Mar-2021].

[2] M. Parker, “Radar Basics - Part 4: Space-time adaptive processing -,” *EETimes*, 28-Jun-2011. [Online]. Available: https://www.eetimes.com/radar-basics-part-4-space-time-adaptive-processing/#:~:text=Radar%20processing%20can%20occur%20over,to%20as%20%E2%80%9Cfast%E2%80%9D%20time. [Accessed: 15-Mar-2021].

[3] S. Ahmed and S. H. Cho, “Hand Gesture Recognition Using an IR-UWB Radar with an Inception Module-Based Classifier,” *Sensors (Basel, Switzerland)*, 20-Jan-2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7014526/. [Accessed: 15-Mar-2021].

[4] C. Shorten, “Introduction to ResNets,” *Medium*, 15-May-2019. [Online]. Available: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4. [Accessed: 15-Mar-2021].

[5] B. Roy, “optim.Adam vs optim.SGD. Let's dive in,” *Medium*, 16-Jan-2019. [Online]. Available: https://medium.com/@Biboswan98/optim-adam-vs-optim-sgd-lets-dive-in-8dbf1890fbdc. [Accessed: 15-Mar-2021].

[6] “CrossEntropyLoss¶,” *CrossEntropyLoss - PyTorch 1.8.0 documentation*. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed: 15-Mar-2021].

[7] K. Shen, “Effect of batch size on training dynamics,” *Medium*, 19-Jun-2018. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e. [Accessed: 15-Mar-2021].