COMPSYS 302
Project 1
Final Report

Group 18

SignAI

Cecil Symes, csym531

Contents

Table of Figures

Abstract

Machine Learning commonly used buzzword in the world of technology, and this project aims to help clarify what it is and how it works.

The goal of this project is to not only create four working models, but to also learn the inner workings of a neural network, how they operate, and what can be manipulated to change a network's effectiveness.

Our pair had very limited prior experience with Python and PyTorch as well as machine learning. After an initial learning period, we were able to successfully create four models, alongside numerous testing and helper functions.

The results showed what we hoped for, with all models learning properly and achieving reasonable accuracy results. Models 1 & 2 attained 85% and 87% accuracy respectively. Models 3 & 4 were optimized versions of Models 1 & 2, and they achieved accuracies of 93% and 87%. These results reflected what we had hoped for in increasing the effectiveness of Model 1 with Model 3, and also showed us that some modifications have little effect with Model 2 and Model 4.

With more time, it could be possible to further increase the accuracy of our models, however this was outside the time scope of this project.

Introduction

Dataset

We decided to work with the Sign Language MNIST dataset on Kaggle (tecperson, 2017). This dataset follows the original MNIST size of 28 pixels by 28 pixels by 1 layer, as all images are grayscale. The dataset is in CSV format, with labels and pixel values in single rows. The dataset is a relatively small 101 MB, and consists of 27,455 training cases and 7,172 testing cases.

We chose this dataset because the only alternative dataset we could find was very large, with an excess of 1 GB of images, each with a size of 200x200 pixels (Akash, 2018). Combined with an 87,000 image training set, and a 29 image test set, we decided that the smaller MNIST dataset would be more useful for our purposes.

Our inexperience and unfamiliarity with machine learning meant we did not expect to create a complex neural network. Using the smaller MNIST dataset allowed us more flexibility to create smaller networks from scratch ourselves in comparison to using larger predefined models.

There were limitations with our smaller MNIST dataset, such as the size of the images being small, which prevented us from implementing too many layers. A larger dataset would be more practical and robust for real world applications. Some advantages of the MNIST style dataset were the ease of use due to the tabular CSV format, as well as the small compact size of the dataset.

Models

Our project consisted of four models, and we simply refer to them with a numbered notation. The structure of Model 1 and Model 2 were from two articles on towards data science, but the code was created from scratch by us (Dias, 2019), (Jain, 2019). Model 3 and Model 4 were created by us as well from scratch, and were also based off the structures of Model 1 and 2 respectively. Model 3 is modified version of Model 1, and Model 4 is a modified version of Model 2. They were optimized in order to gain higher accuracy and lower loss.

The reasoning behind implementing previously designed structures for Model 1 and 2 was because of our lack of experience with Python, PyTorch, and neural networks as a whole. The predefined structure allowed us to focus on implementation and understanding the network instead of worrying about mundane decisions like the number of layers or which activation function to use.

To illustrate the merits of this decision, consider the time to implement each model. Model 1 took over four days of coding to implement and majority of the time was simply understanding how to use PyTorch. Model 2 took less than a day to create, as the previous experience from Model 1 allowed us to avoid trivial syntax errors, and focus on creating the structure of the model. Model 3 and 4 were then fairly straightforward to create as well, with more of a time commitment required to test each and every modification to see if it was beneficial to the final accuracy and loss.

Initially in our design document we wanted to use larger more complex models, such as AlexNet, and VGG. However, we quickly realized that the sheer complexity of these larger sophisticated models was outside our understanding of machine learning, and also overkill for a small dataset like ours.

The focus of our project changed from using large, complex models, to developing a more intimate understanding of PyTorch. We believe this was a good decision, and that choosing to code our own models from scratch has taught us more than copy and pasting could.

Background & Literature Review

Background

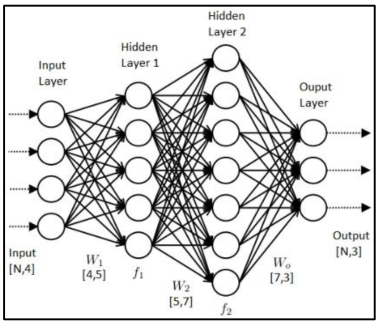

Figure 1: Deep Neural Network with Hidden Layers

Neural networks are interconnected systems of algorithms, weights, biases, and neurons, that use repeated training attempts to recognize patterns and make predictions. A neural network's ability to learn is based around the gradual iterating process of training, where an image is input, and the network's output is tested to see if it was correct or not. If incorrect, the network is back-propagated to decrease the weights of the incorrect neurons, and increase the weighting of the correct neurons. Over time this makes the network's predictions more accurate, and lends them their much more recognizable term of machine learning (3Blue1Brown, 2017).

A deep neural network is a neural network more than two layers, giving it "hidden" layers between the input and output (techopedia, 2018) (Fig. 1).

A convolutional neural network is a neural network at least one convolutional layer (Saha, 2018). This layer reduces image size and complexity and allows for faster training on more complex images. This lends CNNs to be useful with image classification, or "computer vision".

A recurrent neural network consists of a single layer rather than multiple. The single layer instead memorizes the current output and uses it to help predict the next output (aishwarya.27, n.d.). As such, they are useful for sequence prediction tasks, such as speech recognition (Brownlee, 2018).

Literature Review

The first article was "American Sign Language Hand Gesture Recognition" (Dias, 2019). The article had a clear goal established, a method outlined, and justification. The results of the model are also provided. The article also covers a dynamic hand motion neural network, in order to pick up sign language as a motion and not just a static image, but this was irrelevant for our project. The article gives ample detail for their model, a confusion matrix, a convenient diagram of their structure, and a useful image showing the output of the first convolutional layer, visualizing what convolution does.

The second article we referenced was "Sign Language Recognition In Pytorch" (Jain, 2019). The article went into great depth with technicalities, with many code snippets and explanations to guide the reader. The article also provides an accuracy vs loss plot to help visualize the results.

Methodology

Model Design

Model 1

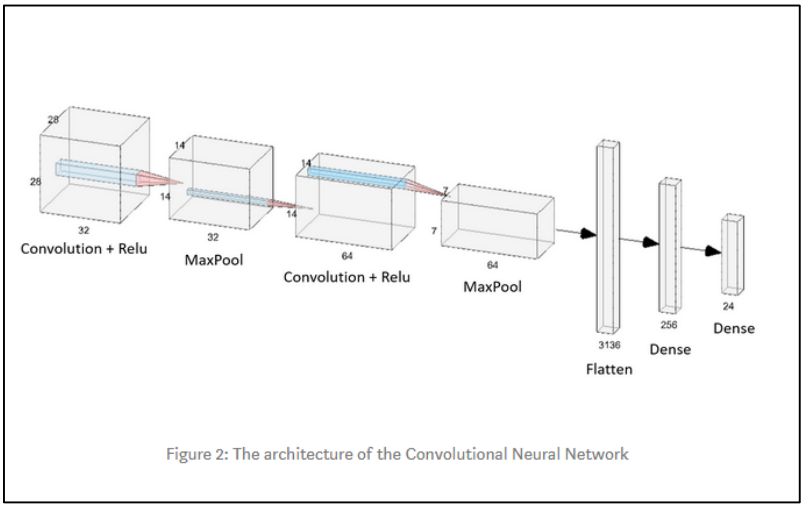

Figure 2: Structure of Model 1

Model 1 has a structure as shown in Figure 2.

1. Convolution layer 1 + ReLU
2. MaxPool
3. Convolution layer 2 + ReLU
4. MaxPool
5. Flatten
6. Fully Connected Linear Layer 1 (Dense)
7. Fully Connected Linear Layer 2 (Dense)

Something to note is that valid padding is added during convolution to maintain the image size throughout.

Model 2

Model 2 goes into more specific detail on the structure but doesn't have a graphical drawing like the first article.

1. Convolution layer 1
2. MaxPool
3. Convolution layer 2
4. MaxPool
5. Convolution layer 3
6. Dropout layer
7. Flatten
8. Dense
9. LogSoftmax Activation Function

Differences to note are the inclusion of a dropout layer with a 0.5 probability after the last convolution layer, and the lack of padding on the images.

Modifications to Model 1 & 2

Here are some of the potential modifications we brainstormed initially to try and increase accuracy of Model 1 and 2, as well as observe the effect of each parameter.

Model 3: Change Model 1's:

- Hyper-parameters
  - Batch size

  - No. of epochs
  - Learning Rate
- Percentage usage of dataset for training and testing
- Number and type of layers
  - Number of convolutional layers
  - Number of maxpool layers
  - Replace ReLU function with sigmoid
- Number of fully connected layers
- Add a dropout layer

Model 4: Change Model 2's…

- Hyper-parameters
  - Batch size
  - No. of epochs
  - Learning Rate
- Percentage usage of dataset for training and testing
- Number and type of layers
  - Number of convolutional layers
  - Number of maxpool layers
- Number of fully connected layers
- Change logsoftmax
- Remove a dropout layer

Modification Results

For Model 3, the modifications highlighted in Figure 3 were made and accuracy noted. The final changes we settled on were to add a dropout layer, convert 2nd ReLU to a Sigmoid, and added a final Linear Fully Connected layer (FC Layer).

The dropout layer prevents the network from forming weak connections and finding patterns that aren't there, and helps to prevent overfitting.

Changes	Accuracy (4 epochs unless otherwise stated)
Add dropout layer	83%
Add dropout layer 2nd ReLU to sigmoid	93% 83, 87, 92 Average = 89% 10epoch= 92%, 94%, 93%, 94%,93% Average = 93%
Add dropout layer Both ReLU to sigmoid	91%,89% 10epoch= 91%,92%,89%
Add dropout layer 2nd ReLU to ReLU6	82%
Add dropout layer 2nd ReLU to logsigmoid	77%
Add dropout layer 2nd ReLU to sigmoid Learning rate 0.1	15%
Add dropout layer 2nd ReLU to sigmoid Learning rate 0.005	10 epochs => 53%
Add dropout layer 2nd ReLU to sigmoid Learning rate 0.0005	10 epochs => 92%, 94%,92%
Add dropout layer 2nd ReLU to sigmoid Added 2nd FC layer	10 epochs => 93% 94% 92%,95%

Figure 3: Modification & Testing of Model 3

Changes	Accuracy (10 epochs unless otherwise stated)
Standard	88, 86 87 at 50 epochs,
Remove dropout layer	88, 87, 86
Logsoftmax to softmax	5%
Add sigmoid to layer 2	89% 89% 89%
Add sigmoid to layer 2, add dropout layer, remove layer 3	90%, 91%, 91%
Add sigmoid to layer 2, add dropout layer to layers 2 and 3	89% 90% 86%
Add sigmoid to layer 2, add dropout layer to layers 2 and 3, extra FC layer	85%, 90%, 89%
Add sigmoid to layer 2, add dropout layer to layers 2 and 3, extra FC layer, Batch size =100 instead of 10	83, 84, 85%
Add sigmoid to layer 2, add dropout layer to layers 2 and 3, extra FC layer, Only training batch size = 100	83 85, 85
Add sigmoid to layer 2, add dropout layer to layers 2 and 3, extra FC layer, only test batch size = 100	89% 86% 89%
Add sigmoid to layer 2, add dropout layer to layers 2 and 3, extra FC layer, only learn batch size = 1	40%

Figure 4: Modification & Testing of Model 4

For Model 4, the modifications highlighted in Figure 4 were made and accuracy noted. The final changes we settled on were to add a Sigmoid function after the second Convolutional layer, add a Dropout layer after the second and third Convolutional layers, and add one final Linear Fully Connected Layer.

The Dropout layers help prevent overfitting and learning patterns that are otherwise noise, and not useful. For both Model 3 and Model 4 we are not fully sure as to why the Sigmoid and FC Layers help accuracy, but they were included regardless.

Model 3

After applying the modifications, the final structure for Model 3 is as listed below.

1. Convolution layer 1 + ReLU
2. MaxPool
3. Convolution layer 2 + ***Replaced ReLU with Sigmoid***
4. MaxPool
5. ***Added Dropout Layer***
6. Flatten
7. Fully Connected Linear Layer 1 (Dense)
8. Fully Connected Linear Layer 2 (Dense)
9. ***Added Fully Connected Linear Layer 3***

Model 4

After applying the modifications, the final structure for Model 4 is as listed below.

1. Convolution layer 1
2. MaxPool
3. Convolution layer 2
4. ***Added a Sigmoid function***
5. MaxPool
6. ***Added a Dropout layer***
7. Convolution layer 3
8. Dropout layer
9. Flatten
10. Dense
11. ***Added Fully Connected Layer***
12. LogSoftmax activation function

Evaluation

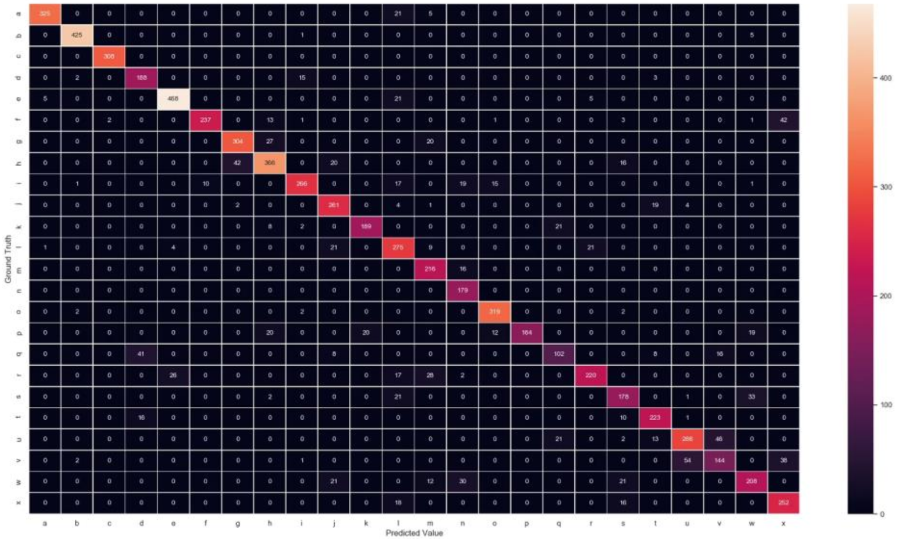

Figure 6: Model 1 Confusion Matrix

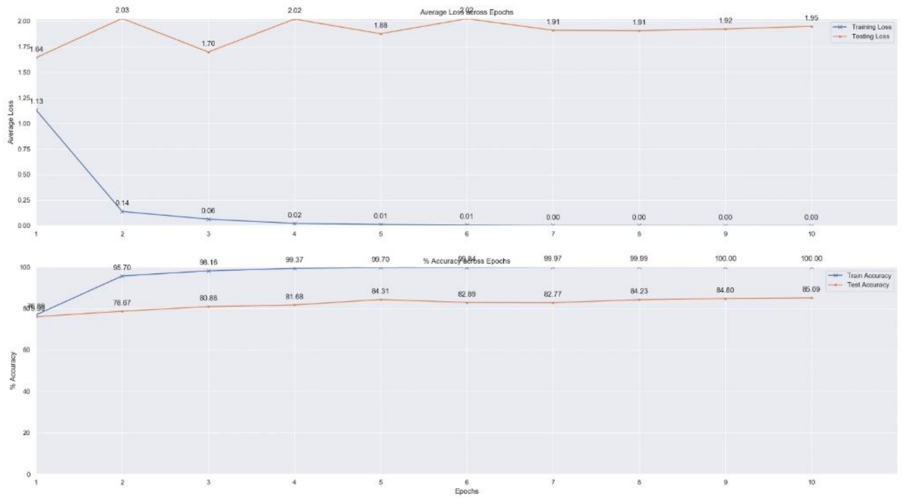

Figure 7: Model 1 Accuracy & Loss

	Precision	Recall	F1-Score	Support
a	0.9259259259259259	0.9818731117824774	0.9530791788856305	331
b	0.9860788863109049	0.9837962962962963	0.984936268829664	432
c	1.0	0.9935483870967742	0.9967637540453074	310
d	0.9038461538461539	0.7673469387755102	0.8300220750551875	245
e	0.9378757515030061	0.9397590361445783	0.9388164493480442	498
f	0.79	0.9595141700404858	0.8665447897623401	247
g	0.8660968660968661	0.8735632183908046	0.8698140200286123	348
h	0.8243243243243243	0.8394495412844036	0.8318181818181818	436
i	0.8085106382978723	0.9236111111111112	0.8622366288492707	288
k	0.8969072164948454	0.7885196374622356	0.8392282958199356	331
l	0.8590909090909091	0.9043062200956937	0.881118881118881	209
m	0.8308157099697885	0.6979695431472082	0.7586206896551724	394
n	0.9310344827586207	0.7422680412371134	0.8260038240917783	291
o	1.0	0.7276422764227642	0.8423529411764706	246
p	0.9815384615384616	0.9193083573487032	0.949404761904762	347
q	0.6978723404255319	1.0	0.8220551378446115	164
r	0.5828571428571429	0.7083333333333334	0.6394984326018809	144
s	0.7508532423208191	0.8943089430894309	0.8163265306122449	246
t	0.7574468085106383	0.717741935483871	0.7370600414078674	248
u	0.892	0.8383458646616542	0.8643410852713179	266
v	0.7771739130434783	0.8265895953757225	0.8011204481792717	346
w	0.602510460251046	0.6990291262135923	0.6471910112359551	206
x	0.7123287671232876	0.7790262172284644	0.7441860465116279	267
y	0.8811188811188811	0.7590361445783133	0.8155339805825242	332

Figure 5: Model 1 Precision, Recall, F1-Score

Model 1

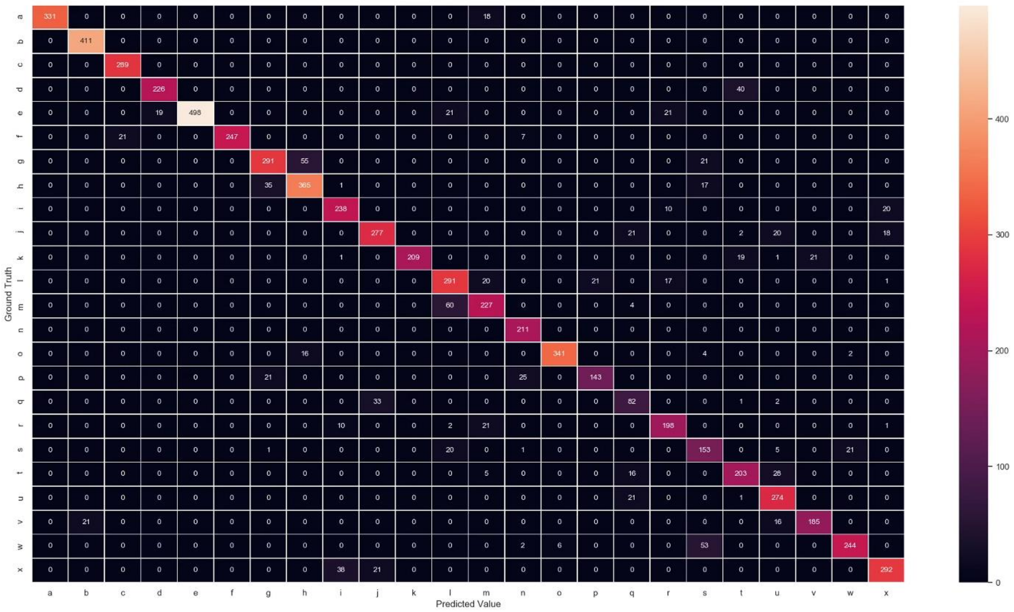

Figure 8: Model 2 Confusion Matrix

	Precision	Recall	F1-Score	Support
a	0.9484240687679083	1.0	0.9735294117647059	331
b	1.0	0.9513888888888888	0.9750889679715302	432
c	1.0	0.932258064516129	0.9649415692821369	310
d	0.849624060150376	0.9224489795918367	0.8845401174168298	245
e	0.8908765652951699	1.0	0.9422894985808893	498
f	0.8981818181818182	1.0	0.946360153256705	247
g	0.7929155313351499	0.8362068965517241	0.813986013986014	348
h	0.8732057416267942	0.8371559633027523	0.8548009367681498	436
i	0.8880597014925373	0.8263888888888888	0.8561151079136691	288
k	0.8195266272189349	0.8368580060422961	0.8281016442451419	331
l	0.8326693227091634	1.0	0.908695652173913	209
m	0.8314285714285714	0.7385786802030457	0.7822580645161289	394
n	0.7800687285223368	0.7800687285223368	0.7800687285223368	291
o	1.0	0.8577235772357723	0.9234135667396061	246
p	0.9393939393939394	0.9827089337175793	0.9605633802816902	347
q	0.7566137566137566	0.8719512195121951	0.8101983002832861	164
r	0.6949152542372882	0.5694444444444444	0.6259541984732824	144
s	0.853448275862069	0.8048780487804879	0.8284518828451882	246
t	0.7611940298507462	0.6169354838709677	0.6815144766146993	248
u	0.8055555555555556	0.7631578947368421	0.7837837837837838	266
v	0.9256756756756757	0.791907514450867	0.8535825545171338	346
w	0.8333333333333334	0.8980582524271845	0.8644859813084114	206
x	0.8	0.9138576779026217	0.8531468531468532	267
y	0.8319088319088319	0.8795180722891566	0.855051244509517	332

Figure 9: Model 2 Precision, Recall, F1-Score

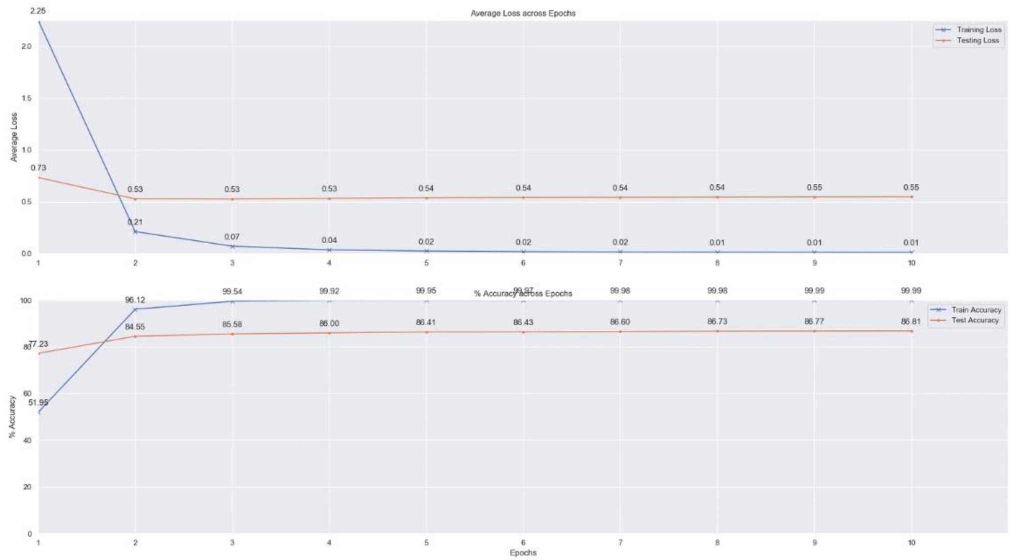

Figure 10: Model 2 Accuracy & Loss

Model 2

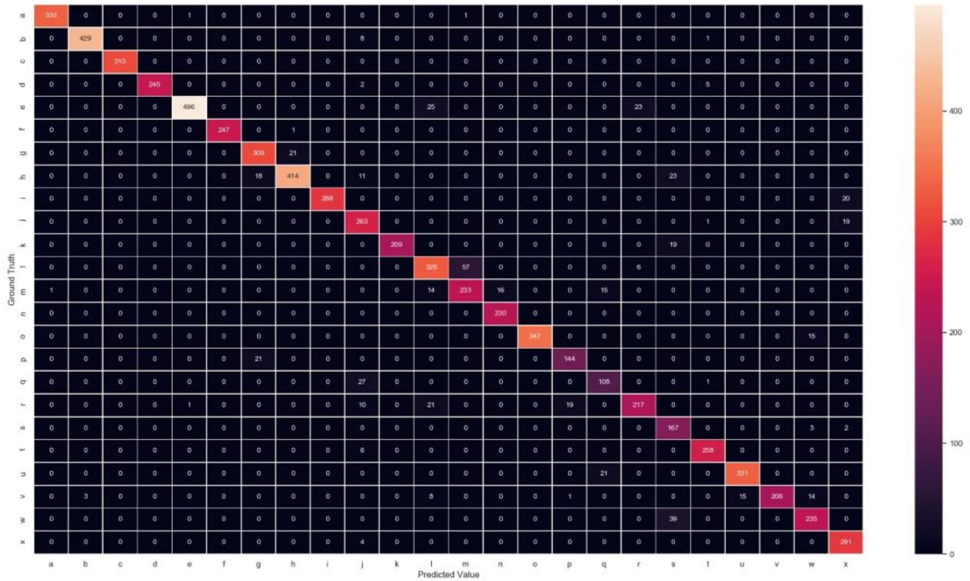

Figure 12: Model 3 Confusion Matrix

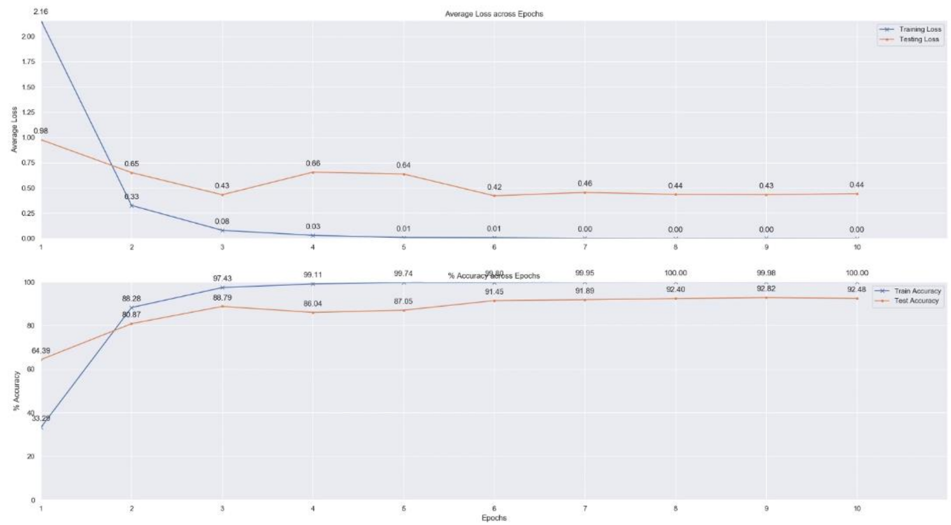

Figure 13: Model 3 Accuracy & Loss

	Precision	Recall	F1-Score	Support
a	0.9939759036144579	0.9969788519637462	0.995475113122172	331
b	0.9794520547945206	0.9930555555555556	0.9862068965517242	432
c	1.0	1.0	1.0	310
d	0.9722222222222222	1.0	0.9859154929577464	245
e	0.9117647058823529	0.9959839357429718	0.9520153550863724	498
f	0.9959677419354839	1.0	0.997979797979798	247
g	0.9363636363636364	0.8879310344827587	0.9115044247787611	348
h	0.8884120171673819	0.9495412844036697	0.9179600886917959	436
i	0.935064935064935	1.0	0.9664429530201343	288
k	0.9293286219081273	0.7945619335347432	0.8566775244299675	331
l	0.9166666666666666	1.0	0.9565217391304348	209
m	0.8380462724935732	0.8274111675126904	0.8326947637292464	394
n	0.8351254480286738	0.8006872852233677	0.8175438596491228	291
o	1.0	0.9349593495934959	0.9663865546218486	246
p	0.9585635359116023	1.0	0.9788434414668548	347
q	0.8727272727272727	0.8780487804878049	0.8753799392097265	164
r	0.7941176470588235	0.75	0.7714285714285715	144
s	0.8097014925373134	0.8821138211382114	0.8443579766536964	246
t	0.9709302325581395	0.6733870967741935	0.7952380952380952	248
u	0.9772727272727273	0.9699248120300752	0.9735849056603774	266
v	0.9403409090909091	0.9566473988439307	0.9484240687679083	346
w	0.8340080971659919	1.0	0.9094922737306843	206
x	0.8576642335766423	0.8801498127340824	0.8687615526802219	267
y	0.9864406779661017	0.8765060240963856	0.9282296650717703	332

Figure 11: Model 3 Precision, Recall, F1-Score

Model 3

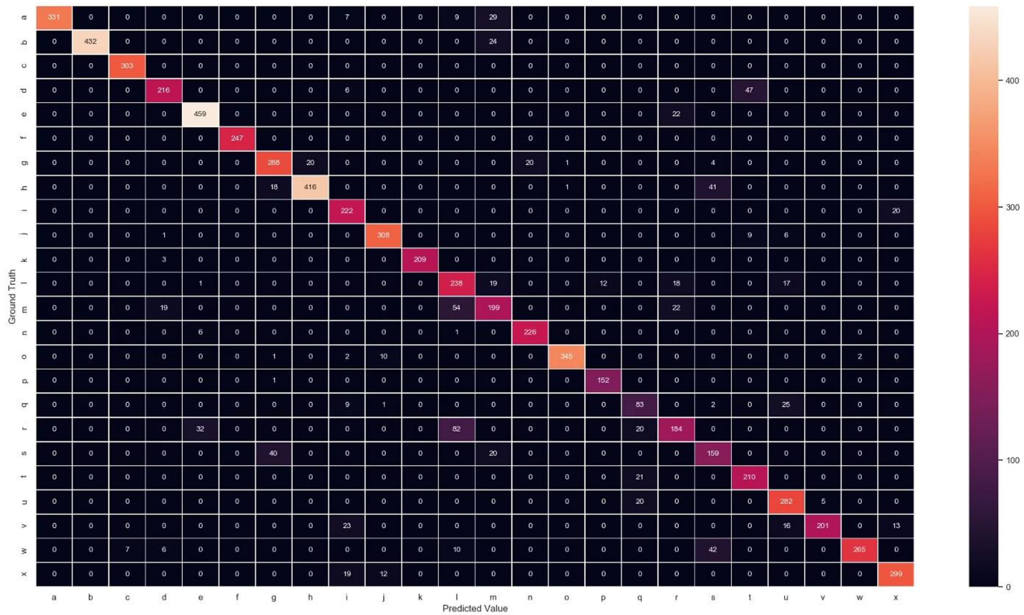

Figure 15: Model 4 Confusion Matrix

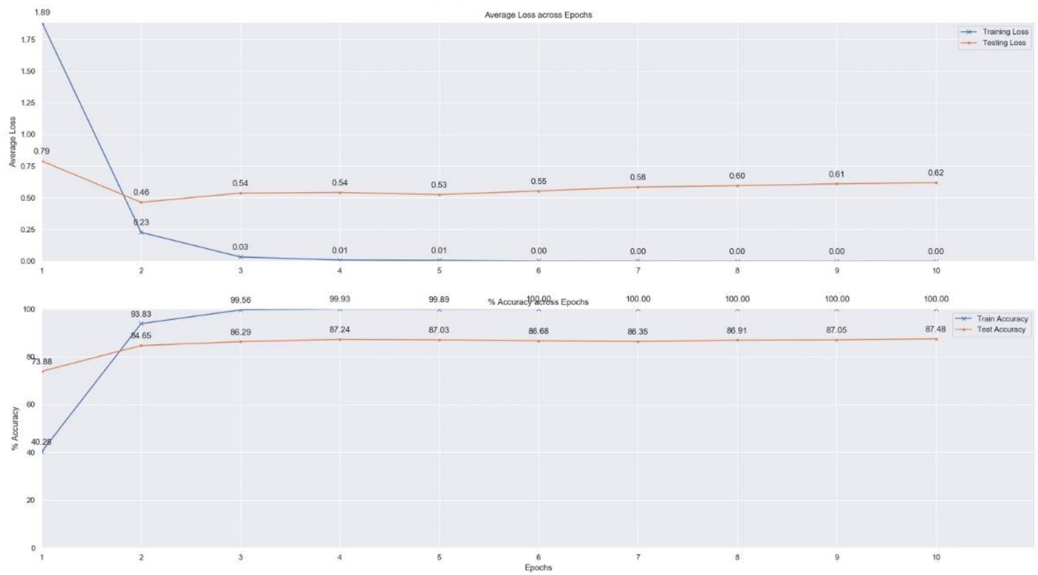

Figure 16: Model 4 Accuracy & Loss

	Precision	Recall	F1-Score	Support
a	0.8803191489361702	1.0	0.9363507779349364	331
b	0.9473684210526315	1.0	0.972972972972973	432
c	1.0	0.9774193548387097	0.9885807504078303	310
d	0.8029739776951673	0.8816326530612245	0.8404669260700389	245
e	0.9542619542619543	0.9216867469879518	0.9376915219611849	498
f	1.0	1.0	1.0	247
g	0.8648648648648649	0.8275862068965517	0.8458149779735683	348
h	0.8739495798319328	0.9541284403669725	0.912280701754386	436
i	0.9173553719008265	0.7708333333333334	0.8377358490566039	288
k	0.9506172839506173	0.9305135951661632	0.9404580152671755	331
l	0.9858490566037735	1.0	0.9928741092636578	209
m	0.780327868852459	0.6040609137055838	0.6809728183118741	394
n	0.6768707482993197	0.6838487972508591	0.6803418803418803	291
o	0.9699570815450643	0.9186991869918699	0.9436325678496869	246
p	0.9583333333333334	0.9942363112391931	0.975954738330976	347
q	0.9934640522875817	0.926829268292683	0.9589905362776027	164
r	0.6916666666666667	0.5763888888888888	0.6287878787878788	144
s	0.5786163522012578	0.7479674796747967	0.652482269503546	246
t	0.726027397260274	0.6411290322580645	0.6809421841541755	248
u	0.9090909090909091	0.7894736842105263	0.8450704225352113	266
v	0.9185667752442996	0.815028901734104	0.8637059724349156	346
w	0.7944664031620553	0.9757281553398058	0.8758169934640522	206
x	0.803030303030303	0.9925093632958801	0.8877721943048575	267
y	0.906060606060606	0.9006024096385542	0.9033232628398792	332

Figure 14: Model 4 Precision, Recall, F1-Score

Model 4

Overall Evaluation

Model Number	Accuracy after 10 epochs/%	Average Loss after 10 Epochs
1	85.09	1.95
2	86.81	0.55
3	92.48	0.44
4	87.48	0.62

Figure 17: Comparison Table of Overall Results

Link to Video Demo

https://webdropoff.auckland.ac.nz/cgi-bin/pickup/f95eeb730cc7e4bf83d773e74526f16b/1106853

Results & Future Work

Results

Out of the four models, the best performing model was Model 3. Model 3 performed better than Model 1, with a relative 8.68% increase up from 85.09% to 92.48%. The loss saw a substantial decrease of 77.34% from 1.95 down to 0.44.

Model 4 almost no change from Model 2. Model 4 only saw a 0.77% increase in accuracy, and an 11.3% increase in loss. Both accuracy and loss fluctuate between different trainings, so there is no evidence to suggest our changes had an impact.

Overall, we managed to significantly improve Model 1, and whilst Model 4 did not see many improvements, it was a valuable learning experience.

Future Work

Given more time and experience, it would be interesting to push even further with Model 3 and especially Model 4. It would be worth trying to further increase Model 4's accuracy, and understand why the changes had no effect.

It would also be interesting to use the larger dataset with 200x200 images, and experiment with more complex networks, like AlexNet or ResNet.

Conclusion

Overall, this project has been an invaluable experience with Python, PyTorch, Github, and even MatPlotLib alongside other popular packages. My understanding and appreciation of machine learning has expanded, and I hope to apply this knowledge in my future career.

References

3Blue1Brown. (2017, October 5th). *But what is a Neural Network?* Retrieved from YouTube: https://www.youtube.com/watch?v=aircAruvnKk

aishwarya.27. (n.d.). *Introduction to Recurrent Neural Network*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Akash. (2018). *ASL Alphabet*. Retrieved from Kaggle: https://www.kaggle.com/grassknoted/asl-alphabet

Brownlee, J. (2018, July 23rd). *When to Use MLP, CNN, and RNN Neural Networks*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

Dias, R. (2019, Dec 14th). *American Sign Language Hand Gesture Recognition*. Retrieved from towards data science: https://towardsdatascience.com/american-sign-language-hand-gesture-recognition-f1c4468fb177

Jain, P. (2019, June 1st). *Sign Language Recognition In Pytorch*. Retrieved from towards data science: https://towardsdatascience.com/sign-language-recognition-in-pytorch-5d72688f98b7

Saha, S. (2018, December 16th). *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*. Retrieved from towarsd data science: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

techopedia. (2018, April 13th). *Deep Neural Network*. Retrieved from techopedia: https://www.techopedia.com/definition/32902/deep-neural-network

tecperson. (2017). *Sign Language MNIST*. Retrieved from Kaggle: https://www.kaggle.com/datamunge/sign-language-mnist