COMP 551 Assignment 3

Matthew Barg, Jack Kelly, Cleo Norris

April 4, 2024

Abstract

In this project we investigated the performance of neural networks (or multi-layer perceptron) implemented from scratch and convolutional neural networks (CNN) using existing libraries on the Sign Language MNIST dataset to classify image data. After implementing the neural networks with different hyperparameters, we experimented with creating an optimal neural network. We found that the neural network with the highest performance had 1 hidden layer and 64 hidden units in each layer with a test accuracy of 52.89%. This can be compared to the performance of the optimal convolution neural network, which had 3 layers of a convolution, batch normalization and max pooling, followed by 2 fully connected layers of size 64 with a test accuracy of 90.88%.

1 Introduction

The goal of this assignment was to implement a basic neural network and its training algorithm from scratch, as well as gain experience with convolutional neural networks. This was accomplished using the Sign Language MNIST dataset. In general, as we increased the depth of our MLP, the performance increase varied. While increase was shown, over-fitting tended to occur at a certain depths for some of the models.

In existing literature, the Sign Language MNIST dataset has been used, among other things, to improve sign language recognition systems. One such example is an article titled “Optimization of Transfer Learning for Sign Language Recognition Targeting Mobile Platform” which develops “a system that can interpret American Sign Language alphabets in [a] real time environment and thus can potentially act as a communication device between the signer and a non-signer” [5]. This methodology included experimentation with Inception V3 and MobileNet models.

It has also been used in an article titled “Two-Stream Mixed Convolutional Neural Network for American Sign Language Recognition” which proposes a Two-Stream Mixed (TSM) method to “improve the correlation of feature expression between two time-consecutive images for the dynamic gestures” [4]. It was found that applying TSM improves feature capture ability. This methodology included adding the TSM block before classifiers such as LeNet, AlexNet, ResNet18, and ResNet50.

2 Datasets

The dataset observed was The Sign Language MNIST dataset, which contains training and test data for images of hand gestures forming 24 classes (English letters minus J and Z). This data is stored as a matrix with rows representing each image, and each column being a grayscale value between 0-255. There are 784 columns since each image is 28x28 pixels. We noted that the size of the testing data is about a quarter of the size of the training data, as the training data has 27,455 images and the test data has 7,172 images. This testing/training split is consistent with what we have worked with in the last two assignments and standard protocol for training models.

To understand the distribution of this dataset, our exploratory analysis involved first examining the raw data, formatting it for normalization, and then normalizing it by subtracting the mean (centering) and dividing by the standard deviation. We then created a bar plot to observe the distribution between class labels, confirming that there are indeed no class 9 or 25 labels (Figure 1).

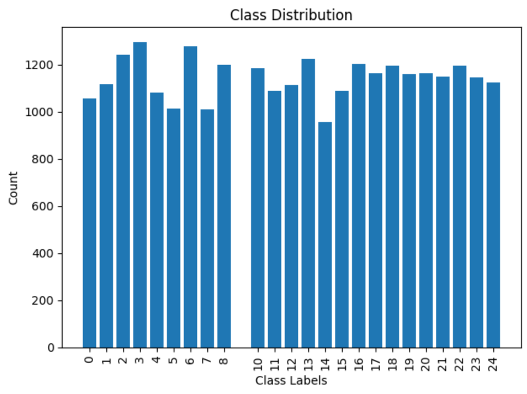

Figure 1: Class Distribution

3 Results

3.1 Create Three Different MLP Models

In our initial implementation of the MLP, we observed the vanishing gradient problem, so we added batch normalization to the forward method of the LinearLayer class. Although it is not shown in the final report, it is important to note that experimentation was done with adjusting the learning rate from 1e-1 to 1e-2. This proved to show extremely poor results and was discarded from our analysis.

In the tables below, Model 1 refers to the MLP with no hidden layers, Model 2 refers to the MLP with a single hidden layer having ReLU activations, and Model 3 refers to the MLP with two hidden layers having ReLU activations.

Introducing nonlinearity through activation functions like ReLU can better allow the MLP to understand the underlying relationships in the data as opposed to simply learning a linear mapping. Similarly, increasing the depth of the network can help the model to learn more complex relationships in the data. Thus, we expect that as we add more layers to the MLP and increasing the number of hidden units, the accuracy will increase.

However, this is only true up to a certain extent - by Model 3, we observe overfitting occurring with a high enough number of gradient steps. This is reflected by the increasing cross entropy loss as seen in the graphs, and the low accuracies as seen in the tables. We can also observe that stopping the training before the overfitting occurs results in higher accuracy - for example, accuracy improving from 27.38% to 36.64% when stopping at 55 gradient steps in Model 3 with 32 hidden units. An area of future work includes implementing a function to automatically select the best MLP model before overfitting occurs during training by checking that the current iteration's loss isn't higher than the previous iteration. Due to computational constraints, we were unable to train the 128 and 256 hidden unit models, illustrating the computational demands of neural networks. We discuss this paradigm further in our conclusions.

Model 1 (200 Epochs)	
Model Depth	Accuracy
No hidden layers	36.91%

Model 2 (200 Epochs)	
Model Depth	Accuracy
32 hidden units	48.77%
64 hidden units	52.89%

Model 3	
Model Depth	Accuracy
32 hidden units (200 epochs)	27.38%
32 hidden units (55 epochs)	36.64%
64 hidden units (200 epochs)	36.50%
64 hidden units (50 epochs)	40.00%

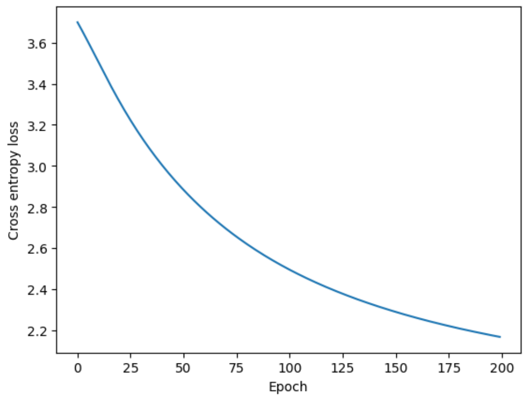

Figure 2: Model 1

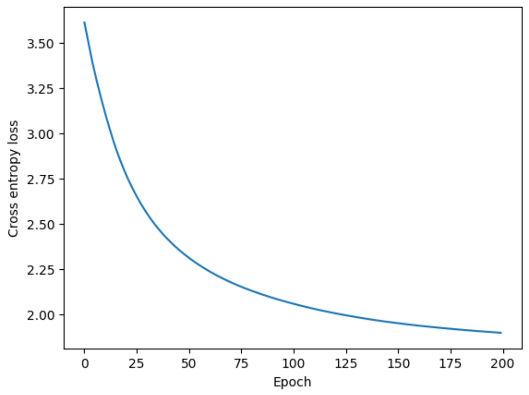

Figure 3: Model 2, 32 hidden units

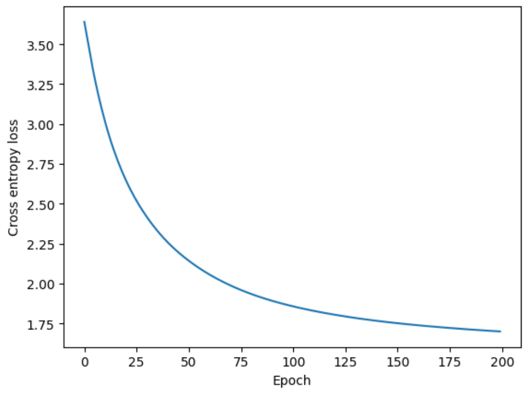

Figure 4: Model 2, 64 hidden units

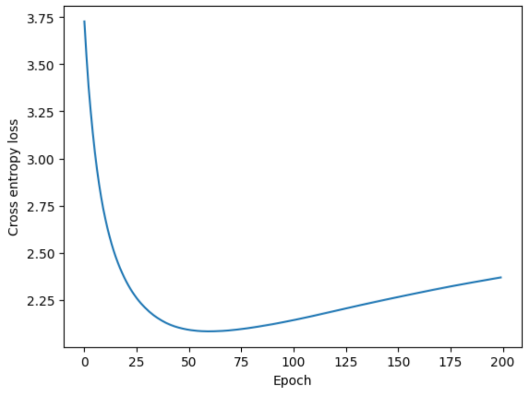

Figure 5: Model 3, 32 hidden units (overfit)

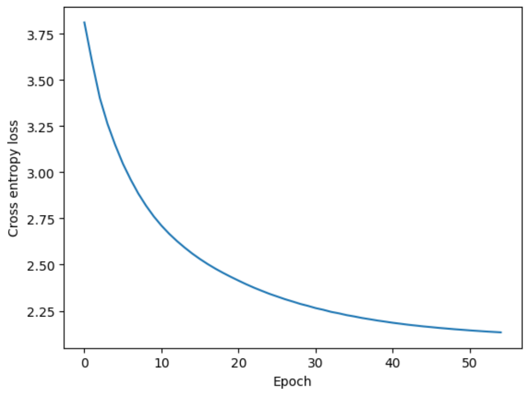

Figure 6: Model 3, 32 hidden units

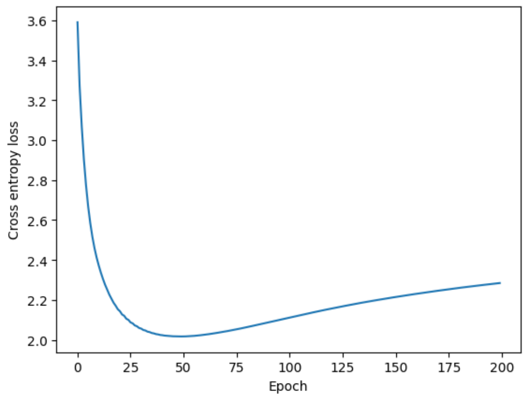

Figure 7: Model 3, 64 hidden units (overfit)

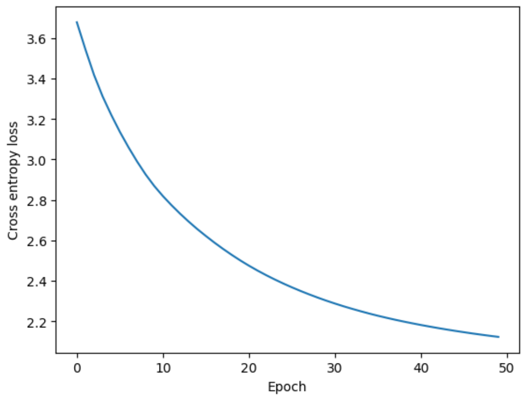

Figure 8: Model 3, 64 hidden units

3.2 Changing Activation Functions

Model 3 Accuracy Comparison from Adjusting Activation Functions	
Status	Test Accuracy
No Adjustment	40.00%
With Sigmoid Activation	37.35%
With Leaky Leaky-ReLU	48.66%

As we have seen in lectures, there is potential for the vanishing gradient problem to emerge during training for models with the sigmoid activation function, as it saturates at 1 for large positive and at 0 for large negative inputs. On the other hand, ReLU is known for solving the vanishing gradient problem as it doesn't saturate for positive input. Further, Leaky ReLU improves upon ReLU by solving the "Dying ReLU" problem in which a neuron stops learning once it becomes negative by providing a small gradient for negative inputs [1]. The theories learned in class are supported by the results show, an increase in performence with Leaky-ReLU activation function but a decrease with Sigmoid activation.

3.3 Adding L2 Regularization to Model 3

Model 3 Accuracy Comparison from implementing L2 Regularization		
Number of Hidden Units	Without	With
32	36.64%	39.99%
64	40.00%	44.20%

The results showed an average increase in accuracy of around 4% for both 32 and 64 hidden units. This increase can be explained by the general nature of regularization, which is reducing the likelihood of over-fitting and penalizing large weights by encouraging the model to keep the weights small. In our trainings of model 3, over fitting was apparent so this makes sense. It is important to note that the model 3 trials with regularization displayed in this table used only 100 gradient steps. This was due to the fact that testing with 200 proved to display over-fitting (as seen in the other models of this report).

3.4 Create CNN

Keras CNN	
Number of Hidden Units	Test Accuracy
32	81.57%
64	80.17%
128	82.78%
256	82.82%

Using a Convolutional Neural Network (CNN) results in a significant improvement in the prediction accuracy on the MNIST sign language dataset in comparison to a fully connected MLP. This is unsurprising, as CNNs have a number of properties which make them much better suited to fitting to image data than regular neural networks (NN). One of these is that the usage of convolutions allows the CNNs to consider the spatial aspect of image data. This way, the CNN can learn spatial patterns and dependencies while also being invariant to translation. Regular NNs have no such capacity, as each pixel value is just treated as a singular input feature. Additionally, convolutional layers also serve to reduce to the dimensionality of image data, which is often large (our data was monochrome images with 784 pixels and high colour resolution images could be much larger) and can share the same filter weights across the whole image, furthering reducing the number of parameters to be trained.

3.5 Optimal MLP Attempt

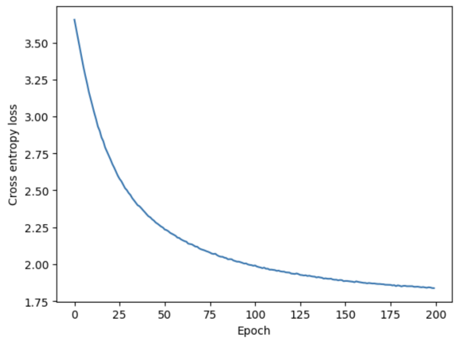

Figure 9: Model 2, 64 hidden units, Dropout layer added

When creating our optimal MLP, we wanted to build off from the highest-performing model of the three models previously constructed and tested in section 3.1. Naturally, we chose Model 2 with 64 hidden units as it showed the highest performance out of the three (52.89% accuracy). To attempt to elevate the MLP's architecture, we added a dropout layer with a dropout rate of 0.2. We chose to do this because dropout is known to enhance generalization and robustness and by result reduce any over-fitting occurring. Surprisingly, the accuracy gave a result of 47.10%, a decrease of 5.79%. This can possibly be explained by the dropout rate possibly being too high which may have led to under-fitting, or a lack of any over-fitting present at all in Model 2. It's also possible that additional hidden units (128,256) could have improved this model however, as mentioned before this was not an option for us. This led us to believe that the original Model 2 was the best that could be produced with our MLP implementation

3.6 Experiments

Keras CNN Experiments	
Architecture	Test Accuracy
Max pooling	88.72%
Max pool + BN	90.88%
Max pool + BN + Dropout	90.20%

During experimentation with the CNN, several dense layer unit sizes were experimented with (32, 64, 128, 64), each with 3 convolution layers and 2 fully connected layers with ReLu activation functions and a softmax output layer. A dense layer size of 256 units produced the best test accuracy of 82.82%. Other techniques such as batch normalization, max pooling and dropout were also added sequentially to the 64-unit model to reduce over-fitting and improve accuracy. First, max pooling was added after each convolution layer, improving our test accuracy to 88.72%. Max Pooling drastically reduces the number of trainable parameters in the model (in this case from 2,000,000 to 60,000) by utilizing a sliding kernel and taking the maximum of the convoluted values inside the kernel. It also prioritizes prominent features by taking the max value, helping the model to sort through noisy data. Next, batch normalization was added, a procedure where the samples in a mini-batch are standardized to have zero mean and unit variance. This procedure reduces the sensitivity to learning rates and reduces gradient explosion, improving the test accuracy to 90.88%. One issue that arose during the training process was the over-fitting of the model to the training data, as evidenced by Figure 9, showing that the training accuracy of the model quickly approaches 100% while the validation accuracy slowly improves. To remedy this, a 50% dropout layer was added before the softmax layer. Dropout is designed to combat over-fitting and increase the generalizability of models by randomly setting the weights of a certain percentage of nodes in a given layer to 0 during training. Interestingly, dropout did not improve the test accuracy or reduce over-fitting as evidenced by Figure 9. As a result, our optimal architecture was one that implemented max pooling and batch normalization, but not dropout.

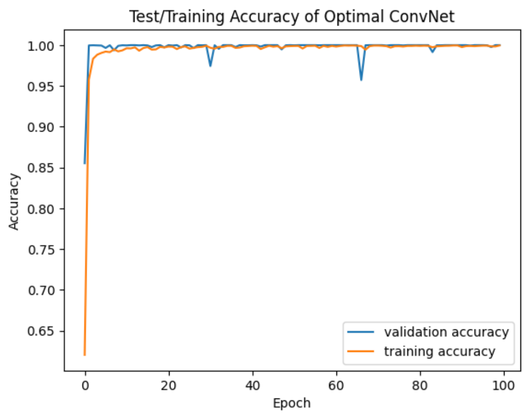

Figure 10: Optimal ConvNet, 100 epochs

4 Discussion and Conclusion

Unfortunately, our MLP implementation struggled to run with 128 or 256 hidden units without timing out (due to Google Collab's computing power refusing to increase). This left some uncertainty whether or not our models could have produced better accuracy results however, enough work was done to compare the results between the models and the effects of different layers and approaches.

One key takeaway was that irrespective of the depth and size of the MLP, CNNs perform much better on image data. This makes sense for several reasons mentioned earlier. Another related takeaway was the size of the parameter space that even relatively small neural networks can have and as a result the large amount of computational resources they require. Our inability to train the larger MLPs with layers of 128 and 256 hidden units highlights this reality and further necessitates the use of CNNs on image data. Even when using CNNs built with Keras, the training time and cost of these models significantly limited our experimentation capabilities. This highlighted the need for a measured approach during usage of compute resources when working with deep learning models.

One area of future investigation could be the effects of adding more images via artificial data augmentation, as done in the example CNN Kaggle [2]. Keras has an ImageDataGenerator function which randomly generates images which are zoomed in/out, rotated or otherwise mildly distorted. These artificially modified images could serve as additional training examples, which might reduce the model's tendency to overfit and improve test accuracy. Additionally, we could investigate the impact of different stride lengths, numbers of filters and filter sizes on the model's accuracy. Larger stride lengths result in more dimension reduction and less trainable parameters, lessening the possibility of over-fitting and reducing training time. Similarly, fewer filters results in less model layers and fewer parameters.

More experimentation with dropout layers to reduce over-fitting is another avenue to explore further. Placing dropout layers in different locations in the architecture or using different probabilities may increase their effectiveness, as problems due to over-fitting were clearly present in some of the models.

5 Statement of Contributions

There was an equal contribution from team members across cleaning data, implementing the methods, experimenting, and the write-up.

References

[1] *Dying ReLU*. Accessed: 2024-04-02. URL: `https://dkharazi.github.io/notes/ml/deep_learning/dying_relu`.

[2] Sayakdas Gupta. *Sign-Language Classification CNN 99.40 Percent Accuracy*. Accessed: 2024-04-04. 2020. URL: `https://www.kaggle.com/code/sayakdasgupta/sign-language-classification-cnn-99-40-accuracy`.

[3] Katherine (Yi) Li. *Vanishing and Exploding Gradients: Debugging, Monitoring, and Fixing*. Accessed: 2024-04-02. 2023. URL: `https://neptune.ai/blog/vanishing-and-exploding-gradients-debugging-monitoring-fixing`.

[4] Ying et al. Ma. “Two-Stream Mixed Convolutional Neural Network for American Sign Language Recognition”. In: *Sensors* 22.16 (2022), Article 5959. URL: `https://www.mdpi.com/1424-8220/22/16/5959`.

[5] Dhruv Rathi. “Optimization of Transfer Learning for Sign Language Recognition Targeting Mobile Platform”. In: *International Journal on Recent and Innovation Trends in Computing and Communication* 6 (2018), pp. 198–203. URL: `https://arxiv.org/ftp/arxiv/papers/1805/1805.06618.pdf`.