COMP 551 - Assignment 3

Sophia Osborne, Jayden Metcalfe, Negin Esmaeilzadeh Kiabani

April 1, 2024

1 Abstract

In this assignment we are implementing different versions of a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN) in order to classify image data from the Sign Language MNIST dataset. We used MLP models with 0, 1, and 2 hidden layers. Afterwards, we compared the classification accuracy of the various models with a CNN. Hyperparameter tuning was used to find the optimal choices for all models. Further experiments involving the L2 regularization and the model architecture were conducted.

2 Introduction

For this assignment, we used the Sign Language MNIST dataset to classify hand signs representing the letters of the alphabet. We implemented classes for the Multi-Layer Perceptron, Optimizers, and the multiple types of Layers that could be used. Throughout the experiment we used Stochastic Gradient Descent. The dataset was initially split into train and test so, we carved out a validation dataset from the training and test data to help select optimal values for hyperparameters such as learning rate, batch size for Mini-batch Stochastic Gradient Descent, and number of hidden units. We created three MLP models: one with no hidden layers, another with one hidden layer, and a third with two hidden layers. The CNN model was implemented using Keras and Tensorflow libraries and had three convolutional layers and two fully connected layers. Unless otherwise stated, the Rectified Linear Unit (ReLU) function was used as the activation function for all models. Throughout the various experiments, we compared the test accuracies for the different models using different hyperparameters and regularization techniques.

When it comes to using neural networks for classifying sign language image data, one study was done using videos of hand gestures. The researchers pinpointed six locations on the hand and tracked them across the various movements for classification purposes. They reported perfect prediction accuracy. However, it is unclear whether this was achieved with test data or not [**3**]. Another study used CNNs to extract features and an ANN for classification. The researchers used data to classify images for twenty different Italian hand gestures and achieved a 91.7% accuracy result [**4**]. A group of students at Stanford used the Sign Language MNIST dataset. When using a CNN, they achieved training, validation, and test accuracies above 95% [**2**].

3 Datasets

The dataset is based on an original MNIST dataset that contained images of handwritten numbers. The goal of the sign language dataset is to make computer vision tools more accessible to deaf or hard-of-hearing individuals. Within the training and test set, there are 26 labels (0-25) for each letter in the alphabet (with no cases for J or Z since they require gesture motion) [**1**]. Each letter had around 900-1300 cases. The Before the validation split, the training data contained 27, 455 cases and the test set contained 7172 cases. The images are 28x28 pixels with a numerical value between 0-255 representing the grayscale of each pixel [**2**].

Since the training set is generated through augmentation and is relatively easier to learn compared to the test set and these two sets seem to be from different populations, a validation set was carved from the existing training and test data. The pre-processing techniques were partially based on what was described by Sayak [**5**]. As observed in Figure 1, after comparing the use of normalization, standardization, and both, it was determined that using both methods led to the highest accuracies.

4 Methods

4.1 Multi-Layer Perceptrons

A multi-layer perceptron is a type of feedforward artificial neural network with layers of non-linear basis functions. Regular perceptrons only work with linearly separable data. MLPs were developed to combat this issue. An MLP contains input and output layers, often with hidden layers in between. The number of units in these layers can vary and they have learnable weights. While being trained, they will use backpropagation methods and their hidden layers have non-linear activation functions such as sigmoid or the Rectified Linear Unit (ReLU) function in order for the model to

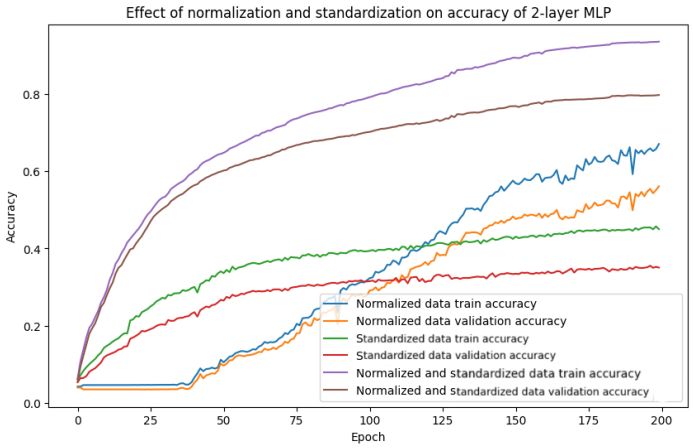


Figure 1: Comparing accuracies using standardized and normalized data

work with more complex data. On the forward pass, the loss is calculated and then during backpropagation, gradient descent is calculated and the weights are updated accordingly. The softmax function is used as the activation function for the output layer when doing multiclass classification, such as with the Sign Language MNIST dataset.

In this assignment we implemented MLP as a class object with forward, backward, fit, predict, loss, check gradient, and scoring functions. A Neural Layer class was defined alongside many sub-variations of layers such as Linear, ReLU, Sigmoid, Leaky-ReLU, and Softmax. A main Optimizer class was defined with subclasses for variations of gradient descent. Maximum number of iterations (epochs) and gradient norm were used as stopping criteria during optmization. The fit function stores training and validation accuracies and losses for future use. The function for checking the gradient computes the difference between gradients that are analytically derived versus those that are numerically calculated. It then gives the sum of all gradient differences relative to each each $w_{i,j}$ within the layers. A relatively small difference indicates that the gradient calculation is correct. The MLP class is also able to account for the L2 regularization penalty if it is given as an input. In subsequent experiments, we determined the optimal hyperparameters for various MLP models.

4.2 Convolutional Neural Networks

Convolutional Neural Networks are also a type of feedforward networks. They are helpful for determining whether a certain pattern appears in an image, regardless of the position. With each forward pass, a filter slides over the input and multiplies the inputs by weights. Padding can be added so the filter can go "outside" the image and prevent centre pixels from being scanned more frequently. They may contain convolutional layers followed by dense, fully connected layers. For this assignment, max pooling was also used after each convolutional layer to compute the maximum over a certain area and subsequently reduce the dimensionality of the feature map. A CNN may also incorporate activation layers using functions such as ReLU. A final fully connected softmax layer exists at the output for prediction purposes. For the puorose of this assignment tensorflow library's CNN is used in the related sections.

5 Results

5.1 Monitoring the model performance

A sample model was used to monitor the loss function during the model fitting/training process. A function for checking the gradient was implemented and used to check the summation of all differences between the numerically derived and analytically derived gradients. Figure 2 shows a graph sampling the results. As we can see the loss function drops significantly during training and the total difference of the gradients is quite small.

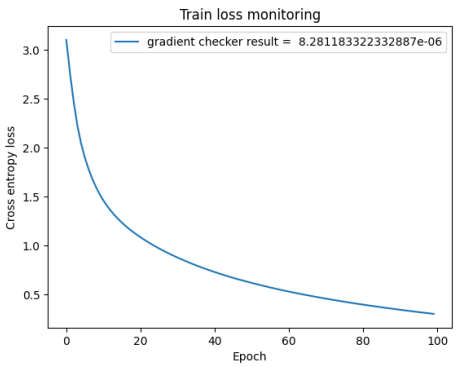
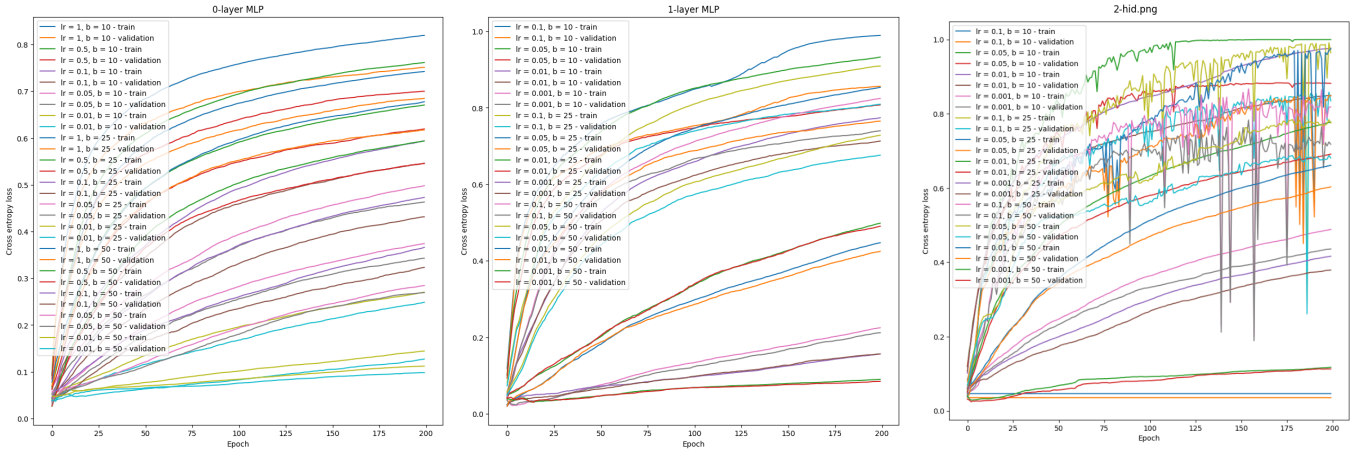


Figure 2: Monitoring the train loss and gradients calculation

5.2 Determining optimal learning rate and batch size

The first set of experiments were done on MLPs. Three different models were built. One had no hidden layers (and therefore was essentially multiclass regression), a second had one hidden layer, and a third had two hidden layers. Using the carved out validation set and a grid search with 3 different values for batch size ([10, 25, 50]) and 4-5 different values for learning rate ([1, 0.5, 0.1, 0.05, 0.01] for 0-layer MLP and [0.1, 0.01,0.05, 0.001] for other MLPs), the optimal learning rate and batch size for mini-batch Stochastic Gradient Descent was determined for each of the the three models. Figure 3 represents the graphs for the various combinations of hyperparameters for each of the models. As shown, smaller learning rates and higher batch sizes lead to a decrease in accuracy for all models and instability, especially in the 2-hidden layer model. For the model with no hidden layers, a larger amount of hyperparameter combinations, especially the ones that include a small learning rate, lead to slower training. Therefore, the models do not rely on the same hyperparameter values for increased accuracy. The optimal learning rate and batch size can be found in Table 1.



(a) no hidden layer (b) 1 hidden layer (c) 2 hidden layer

Figure 3: Optimal learning rate and batch size for 0, 1 and 2-layer MLP

Table 1: Accuracy scores for MLP models and hyper-parameters

Model	Learning Rate	Batch Size	Train Accuracy	Validation Accuracy
0 hidden layers	1	10	81.97	75.12%
1 hidden layer	0.1	10	98.92	85.59%
2 hidden layers	0.05	10	100.00	88.21%

5.3 Testing various MLPs with different numbers of hidden units

After determining the optimal learning rate and batch size for each model, the MLP models were tested with different numbers of hidden units: 32, 64, 128, and 256. Each model was fit using the hyperparameters determined in Section 5.1. Table 2 shows the complete results for each model with the varying numbers of hidden units. As indicated, the 0 hidden layer model achieved the highest validation accuracy with 32 hidden units. The test accuracy was 73.94%. 256 units led to the highest validation accuracy for the model with 1 hidden layer. The test accuracy for this model was 89.91%. The 2-hidden layer model had the highest validation accuracy with 128 hidden units. When scored on the test data, the accuracy was 89.86%. Figure 4 highlights the training and validation accuracies for each model when tested with different units, and the test accuracies when using the optimal number of hidden units.

Table 2: Validation accuracy scores for MLP models with various numbers of hidden units

Model	256 units	128 units	64 units	32 units
0 hidden layers	75.09%	74.99%	75.14%	75.21%
1 hidden layer	90.40%	88.44%	87.68%	81.86%
2 hidden layers	89.89%	89.90%	89.43%	89.33%

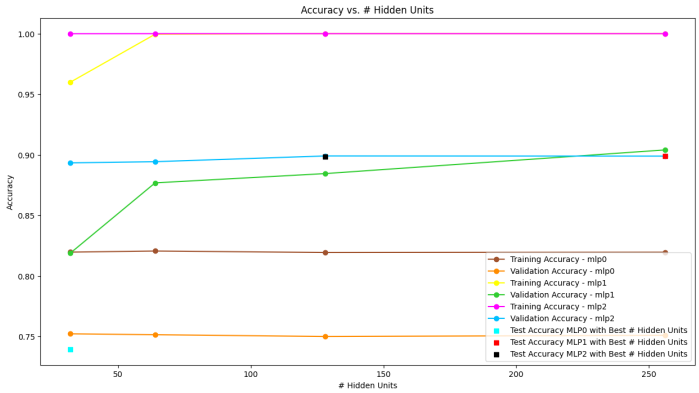


Figure 4: Accuracies when testing various numbers of hidden units

5.4 Comparing ReLU, Leaky-ReLU, and sigmoid as activation functions

For this experiment we compare MLP implementation using different activation functions. The functions being compared are ReLu, Leaky-ReLu and sigmoid activation. Leaky ReLu preserves some of the influence of negative inputs where ReLu sets negative inputs to zero. Sigmoid activation given that it is saturated at 0 and 1 is prone to vanishing gradient problems for deep networks. I tested all three activation functions using a batch size of 10 and a learning rate of 0.05 as this was found to be the optimal values for a two hidden layer MLP model as shows in Table 1. The accuracy of the models was compared when using 32, 64, 128 and 256 hidden units to analyze the impact of the activation functions for varying number of units. Table 3 shows the validation accuracy scores for each of the different combinations.

Table 3: Validation accuracy scores for MLP models with various numbers of hidden units and activation functions

Activation Function	**256 units**	**128 units**	**64 units**	**32 units**
ReLu	89.89 %	89.90 %	89.43%	89.33 %
LeakyReLu	88.79%	88.53%	87.81%	85.21%
Sigmoid	86.74%	83.07%	82.85%	82.34%

5.5 Using L2 Regularization

Here an MLP with two hidden layers was implemented but with L2 regularization. A batch size of 10 was used with 32 hidden units and a learning rate of 0.05. This model implementation was tested with a range of lambda sizes to asses the impact of adjusting this value. Values used were $\lambda = 0.1, 0.05, 0.01, 0.005, 0.001, 0.0001$. The results are shows in Table 4, from these results we see that accuracy increases as lambda tends towards zero. With values greater than 0.01 the model fails to complete running and exits with the very low accuracy of 3.66%. From the values tested the highest test accuracy achieved was 80.52%.

Table 4: Validation accuracy scores for MLP models with L2 Regularization

Lambda Value	**0.1**	**0.05**	**0.01**	**0.005**	**0.001**	**0.0001**
	3.66%	3.36%	3.36%	17.12 %	53.92%	80.52%

5.6 Training the Convolutional Neural Network

A CNN with three convolutional layers and two fully connected layers was constructed. Initially, the optimal number of hidden units was determined in a similar manner to Section 5.2 using initial hyperparameters modelled on the Kaggle notebook [**5**]. The number of filters, sizes of filters, and MaxPooling layer Pool sizes varied depending on the layer. The model was run for 70 epochs. The results showed that using 256 units in the Dense fully connected layers led to the highest validation accuracy and a test accuracy of 41.79%. The full set of training and validation accuracies is displayed in Table 5.

Table 5: CNN accuracy scores for different numbers of hidden units

	256 units	**128 units**	**64 units**	**32 units**
Training	93.35%	87.42%	66.19%	80.93%
Validation	56.34%	52.79%	40.31%	48.74%

When testing for hyperparameters, the validation accuracy was noticeably lower than the training accuracy. Typically, this is indicative of overfitting. However, the fact that the validation set is a combination of the training and test data,

which are independent from one another, must be considered. It is expected that the accuracy with the test data would be lower which could explain why the inclusion of the test data in the validation set would lead to lower validation accuracies.

After the number of hidden units was determined, other hyperparameters were experimented on. A grid search was also used for optimal parameter selection. The options for number of filters were 32, 64, 128. The sizes of the filters were 2x2 or 3x3. We also compared setting padding to be 'valid' or 'same'. These are pre-set values for the Keras/TensorFlow CNN Sequential model that was used. Using 'valid' indicates that no zero-padding is added whereas 'same' indicates that the spatial dimensions should be preserved and so the output size matches the input size. Then, we compared having a dropout proportion of 0.2, 0.3 or 0.5 after each fully connected Dense layer.

Overall, 36 combinations of these hyperparameter options were compared. The top 5 combinations are shown in Table 6, ranked by their validation accuracy. The trends show that using a higher number of filters and larger filter lead to more optimal results. There is no true pattern within the options for padding, and the only observation for the dropout proportion is that 0.3 is not highlighted within the top 5 combinations.

No combination leads to extremely high accuracies. This is expected to be partially due to the aforementioned combination of training and test data that makes up the validation set. It also indicates that a mixture of the hyperparameters for each layer is likely more optimal than using the same values for each consecutive layer. The trends are a good start for looking into the ideal hyperparameters but a higher degree of computational power and specificity would likely yield more optimal results. Research shows that when the original test set is used as a validation set within the fit function, the accuracy improves [**5**]. However, the use of a true validation set is often more preferred in practice.

Table 6: Accuracy scores for top CNN hyperparameter combinations

	# of filters	**Filter size**	**Padding**	**Dropout**	**Train acc**	**Validation acc**	**Test acc**
1	128	(3,3)	same	0.2	84.92%	50.86%	24.18%
2	128	(3,3)	valid	0.2	83.96%	49.65%	42.46%
3	128	(3,3)	same	0.5	82.12%	49.64%	28.14%
4	64	(3,3)	valid	0.5	79.63%	47.86%	40.06%
5	128	(3,3)	valid	0.5	78.37%	47.01%	38.46%

5.7 Finding optimal MLP architecture

The aforementioned experiments were referred to when determining the optimal MLP architecture. It was found from Table 1 that the optimal MLP model has two hidden layers with a learning rate of 0.05. From Table 2 it is found that including 256 hidden units further increases the accuracy of the model. The test accuracy of this model was 89%. This significantly outperforms the CNN in the experiment above (results shows in Table 5). However, the MLP with this many hidden units took approximately 2 hours to fit to the data. As a CNN is better suited to image data its time complexity can be significantly reduced as it parses through the data as a grid rather than taking in all the pixels as individual inputs.

5.8 Comparing MLP and CNN train/test accuracy over multiple epochs

The CNN was implemented with three convolutional layers and two fully connected layers. ReLu activation is used at each layer with SoftMax function used to determine the class of the output as this is a multi-class classification problem. After 30 epochs the final test accuracy of the CNN was 90.41%. While 30 epochs was set as the hyperparameter, it appears from the graph that the CNN hits its peak at approximately 10 epochs and then its performance plateaus from there. The evolution of the testing and training accuracy of the CNN can be seen in Figure 5. For comparison, Figure 6 shows the training and testing accuracy of and MLP model over 200 epochs. A learning rate of 0.05 and batch size of 10 for consistency with models previously implemented in this project as these were determined hyper parameters. The number of hidden units is 32 as it is found to have limited impact on accuracy while significantly reducing model run time. With this implementation, the MLP has a lower accuracy than the CNN with a final test accuracy of 86.11%. The performance of this model appears to plateau after approximately 150 epochs.

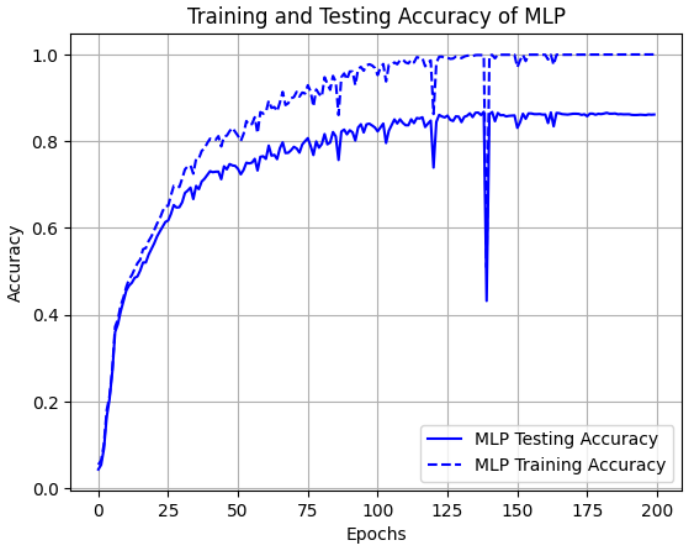


Figure 5: Training and Testing Accuracy of MLP

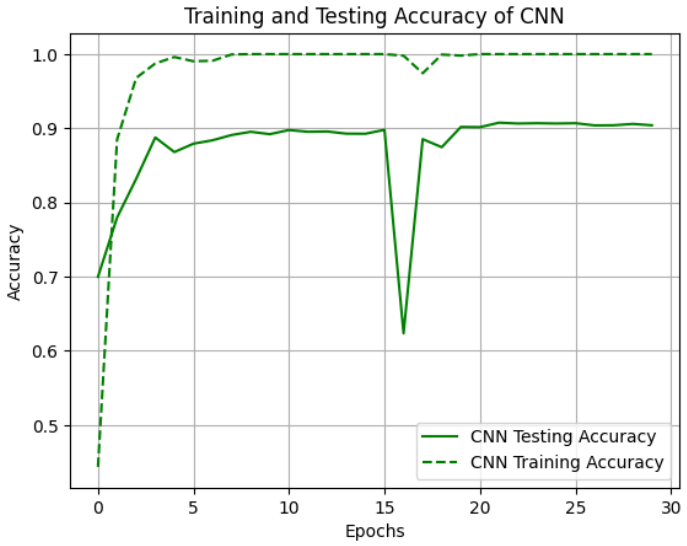


Figure 6: Training and Testing Accuracy of CNN

6 Discussion and Conclusion

6.1 Hyperparameter tuning conclusions

Reviewing the performance of the MLP models when tuned to an assortment hyperparameters, it is observed that different variations of the MLP class respond with varying degrees of sensitivity to certain hyperparameters such as learning rate, batch size and number of hidden units. Therefore, we conclude that ideally for each single experiment, best performance is achieved when an exhaustive search on all possible choices for each parameter is done. However, a reasonably sized grid search can also lead the model to a significantly better performance.

6.2 ReLU-type activation functions lead to increased MLP performance

In general, ReLU activation functions prove to be advantageous over sigmoid activation functions as ReLU-type functions lead to faster convergence, sparsity in activation, and they avoid problems such as saturation and vanishing gradients. The experiment in Section 5.4 reflects these generalizations. Additionally, Leaky-ReLU can address some limitations faced by the ReLU function. Leaky-ReLU can prevent neurons from becoming "dead" and it may be an equivalent or more suitable choice in less complex models.

6.3 L2 regularization does not benefit a well adjusted MLP

L2 regularization is useful when training data is limited compared to the number of features, or when the model is prone to over-fitting due to a high model capacity relative to the available data. An MLP trained on a large and diverse

dataset may learn robust representations without over-fitting, making additional regularization more unnecessary. In our experiments we observed that adding regularization may be helpful for really small lambdas ($< 10^{-4}$). However, larger lambdas can have the opposite effect on performance. For the MLP experiments, there were no significant signs of overfitting. Therefore, This shows that the model may already have a good balance between bias and variance, and additional regularization was not necessary.

6.4 MLP outperformed CNN

Generally, CNN is better suited to handling image data and therefore parses the information at a much faster rate. However, our observation on this dataset was that MLP outperformed CNN in almost all experiments. There are a couple possible explanations for this observation:

1. CNN is more sensitive to data augmentation: CNNs typically require large amounts of labeled training data to learn meaningful features. If the dataset is small then data augmentation techniques may not be effective in enriching the dataset. On the other hand, MLPs may generalize better with limited data, as they are less prone to overfitting and may capture simpler patterns effectively. In this assignment we worked on the Sign Language MNIST dataset which has separate training and test sets. The training set of this dataset is synthetically expanded with data augmentation techniques to increase the size of the training set, whereas the test data is mostly comprised of real images.

Since the training data was simpler to work with as compared to the test set and the carved out validation set, the CNN could be prone to overfitting and have a more difficult time predicting the labels on the unseen data.

2. Neural networks are too sensitive to hyper-parameters tuning: A neural network's performance is highly dependent on its architecture and hyperparameters. The MLP had fewer hyperparameters to select which made it easier to find the optimal values. The CNN model had a larger amount of hyperparameters to optimize. Proper hyperparameter tuning - including the number of layers, number of neurons, learning rate, and regularization strength, can significantly impact the performance of both MLPs and CNNs. It would be interesting for a future investigation to look more into better defining hyper parameters for the CNN. Hence, it seems that future investigations should focus in on optimizing this more time efficient model and looking into varying the hyper parameter values for each layer.

6.5 Stopping criteria and Adaptive optimization algorithms can help improving time efficiency and performance

Despite our implemented MLP having gradient norm convergence as the early stopping criteria, we did not benefit from this property for the majority of the experiments since we set the maximum number of epochs slightly lower than the number required to get complete convergence in order to optimize the time cost. So the maximum epochs was usually the limit in our experiments. To advance this, we may need to implement some self updating optimization methods like ADAM for our MLP to allow for quicker convergence and make use of early stopping criteria which can save time and improve the performance of the model. Similar to MLP, we could also look into early stopping criteria for CNN models to reduce computational load.

7 Statement of Contributions

Jayden did part of the data pre-processing, wrote part of the report and did Experiments 1 and 4. Negin did part of the data pre-processing, implemented the MLP, Optimizer, and Layers classes, and determined optimal learning rate and batch size for the MLP classes. Sophia did Experiments 2, 3, 5, and 6 and wrote report sections relevant to them.

References

[1] tecperson. (2017). *Sign Language MNIST. Kaggle. https://www.kaggle.com/datasets/datamunge/sign-language-mnist*

[2] Khan, M., & Gondi, A. (2021). *Interpreting Sign Language using Image Classification Techniques. Stanford University, Department of Computer Science. https://cs229.stanford.edu/proj2021spr/report2/81997848.pdf*

[3] Mekala, P., Gao, Y., Fan, J., & Davari, A. (2011). Real-time sign language recognition based on neural network architecture. *2011 IEEE 43rd Southeastern Symposium on System Theory.* https://doi.org/10.1109/ssst.2011.5753805

[4] Pigou, L., Dieleman, S., Kindermans, P.-J., & Schrauwen, B. (2015). Sign Language Recognition Using Convolutional Neural Networks. In *Computer Vision - ECCV 2014 Workshops* (Vol. 8925, pp. 572–578). Springer, Cham. Retrieved March 28, 2024, from https://link.springer.com/chapter/10.1007/978-3-319-16178-5_40#citeas.

[5] Sayak. (2017). *Sign-Language Classification CNN (99.40% Accuracy). Kaggle. https://www.kaggle.com/code/sayakdasgupta/sign-language-classification-cnn-99-40-accuracy#Building-the-CNN-Model*