# COMP 551 A3: Classifying Image Data with MLP and CNN models

Mira Kandlikar-Bloch, Trystan Vaillancourt, and Gaspard Ratovondrahona Rasoarahona

April 2, 2024

## 1 Abstract

In this project we implemented a Multi-Layer Perceptron model from scratch to test the performance of three instances of MLPs with 0-2 layers in categorizing an image dataset of sign language letters from the Sign Language MNIST dataset. Among these three models, we found that the 1-layer MLP performed the best at classifying the image data in comparison to the 0-layer and 2-layer MLPs. We performed experiments to test the efficacy of different activation functions and L2 Regularization in increasing the accuracy of our models. We then used the Keras TensorFlow library to implement a Convolutional Neural Network. We tuned our CNN and tested the performance of the CNN on the same dataset. We then created an MLP with an optimal architecture. Overall, we found that the CNN outperformed all of our MLPs, even the one with an architecture tuned to be as accurate as possible.

## 2 Introduction

We began this project by loading and pre-processing the image dataset and implementing our MLP as a Python class. We initialized three different models: an MLP with no hidden layer, an MLP with a single hidden layer, and an MLP with two hidden layers. The initial hidden layer MLPs use the ReLU activation function for all hidden layers.

For all three models, we did hyperparameter testing. For the non-hidden layer MLP, we tested a set of learning rates to determine the best one. For the one and two hidden-layer MLPs we tested the same set of learning rates against 32, 64, 128, 256 hidden units per layer. We compared the test accuracy of all three models and found that our single-layer MLP performed the best.

We then created two copies of our 2-layer MLP one with Sigmoid and one with leaky-ReLU activation functions. We compared the test accuracies of these models to our original ReLU model and found that the Sigmoid performed the worst, and ReLu the best.

We then took our 2-layer ReLU model and added regularization to the weights in layers 1, 2, o, and trained this model. We found that this L2 regularization had little to no effect on the accuracy of our model, we concluded that this was because our model was not overfitting the training data.

Using the Keras library in TensorFlow we created a Convolutional Neural Network. We tuned this model and experimented with the number of hidden units and filters. Ultimately we found that the CNN was the most accurate at predicting the image data among all of our models. Finally, we compared all the models by training them as a function of epochs and testing their accuracy to find the optimal number of epochs for training.

The Sign Language MNIST dataset has been used for analysis in other papers such as "A Comparative Analysis of American Sign Language Recognition Based on CNN, KNN, and Random Forest Classifiers" by Tushar Kumar Mahata at Daffodil International University. In this paper, their CNN was found to have a test accuracy of 96.2%, which gives us a benchmark for the accuracy our CNN implementation should approach.

## 3 Dataset

The dataset we used for training and testing our models in this project is Sign Language MNIST. This image dataset is comprised of 24 classes, one for each sign language letter excluding J and Z because of their dynamic nature in sign language. Each image in the dataset is composed of 784 pixels, meaning that each image was

treated as a vector with 784 features for use in our MLPs. This dataset is pre-split into training and test data, with 27,455 and 7,172 samples respectively. The distribution of the 24 classes was uniform for the training set, and slightly less uniform for the test set. To pre-process this dataset, we separated the "label" columns from both the train and test subsets, which gave us X and y sets for both train and test. We then took the y vectors for both train and test sets and turned them into matrices of one-hot-encoded vectors using the LabelBinarizer class. We standardized both the X train and test sets. Finally, we created a validation set by carving out subsets of both the train and test data and combining them. The reason behind this choice is that model had a much harder time predicting the test data, compared to the validation data, which was initially a subset of training data. We found later that this training set was generated artificially from the test set, which could explain why it was easier for our models to predict the training set.

This was all the preprocessing necessary for our MLP, however, for the CNN implementation, we had to do a further step. Since CNN takes matrices of pixels as input instead of vectors, we transformed the input vectors with 784 features to matrices of 28 by 28 pixels. This is the data we used to train, validate, and test our CNN.

# 4 Results

## 4.1 Experiment 1: Finding the optimal units per layer and hyperparameter tuning

In experiment 1 we were asked to test each of our three models using 32, 64, 128, and 256 hidden units per layer, to see which gives the best model performance. We opted to test these units per layer against learning rates of 0.1, 0.05, 0.01, and 0.005, over 2 epochs.

For our 0-layer MLP (multi-class regression model), we only tested the learning rates as there are no hidden layers, and therefore no hidden units. From the set of learning rates above, we found that we got the best validation accuracy with a learning rate of 0.1. Below is a table comparing the training and validation accuracies of our 0-layer MLP against each learning rate.

Learning rate vs train vs validation accuracy for 0-layer MLP

| Rate | Train Accuracy | Validation Accuracy |
|---|---|---|
| **0.100000** | 0.978984 | 0.900052 |
| **0.050000** | 0.949372 | 0.878598 |
| **0.010000** | 0.838754 | 0.789901 |
| **0.005000** | 0.768749 | 0.731554 |

For our 1-layer MLP we tested each number of hidden units per layer against each learning rate. We got the highest validation accuracy with a learning rate of 0.1 and 256 hidden units per layer.

Validation accuracies at each Unit and Learning rate for 1-layer MLP

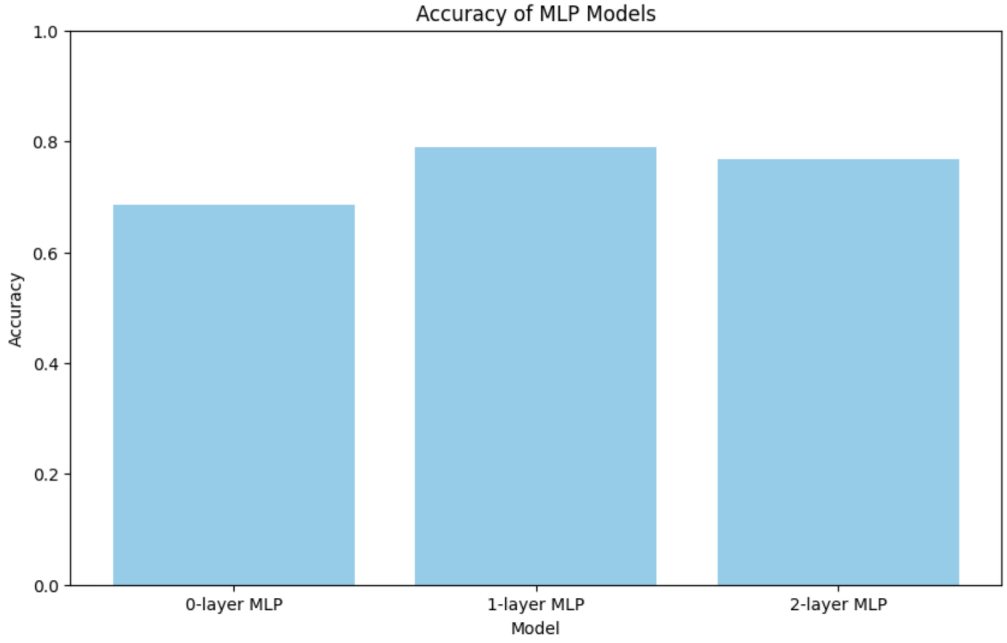
| | 32 | 64 | 128 | 256 |
|---|---|---|---|---|
| **0.100000** | 0.913396 | 0.924123 | 0.932496 | 0.932234 |
| **0.050000** | 0.823914 | 0.851648 | 0.887493 | 0.901884 |
| **0.010000** | 0.324961 | 0.399006 | 0.457352 | 0.524856 |
| **0.005000** | 0.181842 | 0.210099 | 0.257457 | 0.313710 |

For our 2-layer MLP, we tested hyperparameters in the same manner as the 1-layer MLP. We found the highest validation accuracy with 256 units per layer and a learning rate of 0.1.

Validation accuracies at each Unit and Learning rate for 2-layer MLP

| | 32 | 64 | 128 | 256 |
|---|---|---|---|---|
| **0.100000** | 0.409210 | 0.588435 | 0.726060 | 0.870487 |
| **0.050000** | 0.139194 | 0.213501 | 0.405285 | 0.564103 |
| **0.010000** | 0.040293 | 0.055468 | 0.066457 | 0.094192 |
| **0.005000** | 0.040293 | 0.040293 | 0.042125 | 0.067242 |

After tuning all of our models, we compared the accuracies of each model on our test data. We trained the 0-layer, 1-layer, and 2-layer MLPs with their respective hyperparameters, however, we decided to increase the number of epochs to 15. In comparing these three models we see that the 1-Layer MLP performed the best on the test data with a test accuracy of 79.3%. Our 2-layer MLP performed second best with a test accuracy of 76.7%. Finally, the 0-layer MLP performed the worst with a test accuracy of 69.4%. It makes sense that the 0-layer MLP performed the worst, as linear models have a harder time than non-linear models at predicting data that is not linearly separable. However, we were surprised to see that the 2-layer MLP performed worse than the 1-layer counterpart. We were expecting the best performance from our 2-layer MLP since MLPs with more layers are often better at generalizing as they can learn more complex data patterns. The below bar chart displays the comparison in test accuracy between the three tuned models.

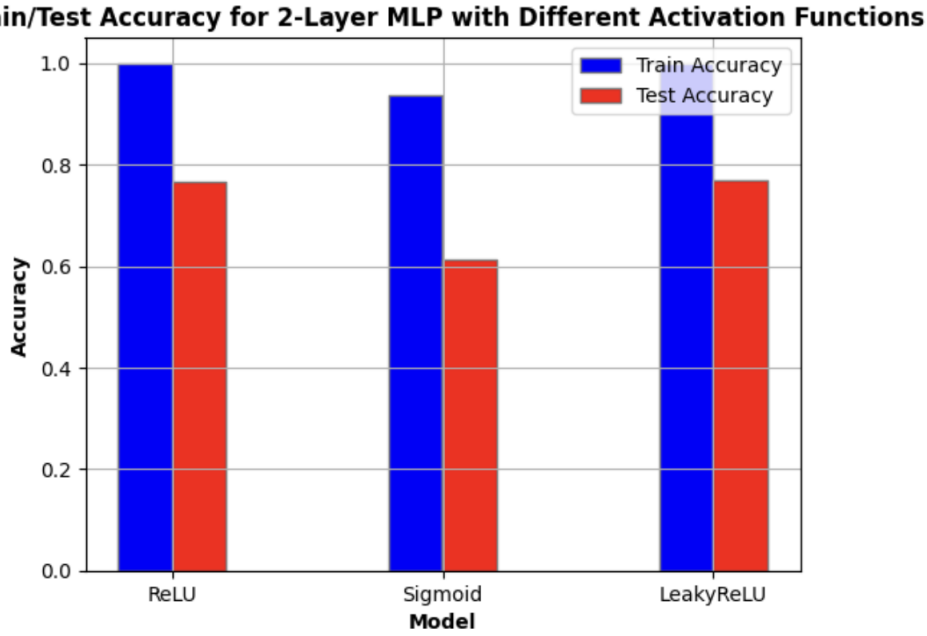



### 4.2 Experiment 2: Comparison of activation functions on 2-layer MLP

In experiment 2 we created two copies of our 2-layer MLP, one with Sigmoid as its activation function and another with Leaky-ReLU, with an $\alpha$ of 0.1. We trained both models with the same hyperparameters we got from tuning our 2-layer ReLU MLP in experiment 1, over 15 epochs. We then compared the accuracy of these two models with the test accuracies of the 2-layer ReLU MLP that we had previously trained. As expected, the model with the Sigmoid activation function performed the worst out of the three, with a testing accuracy of 61.2%. This is because of the vanishing gradient problem, where the function saturates and the gradient becomes very close to zero in deeper layers.

In comparing the ReLU and Leaky-ReLU MLPs, both of which mitigate the vanishing gradient problem, we found the same test accuracy of 76.7%. Considering our MLP is not very deep, it is unsurprising that there is no difference between these two activation functions, as the benefits of Leaky-ReLU generally start to appear in deeper networks where some units "die" when using ReLU because they never activate due to negative inputs and

negative bias terms. Since our network is not very deep and Leaky-ReLU behaves like ReLU for positive values, similar test accuracies make sense. Below is a chart displaying the training and test accuracies of all three models.



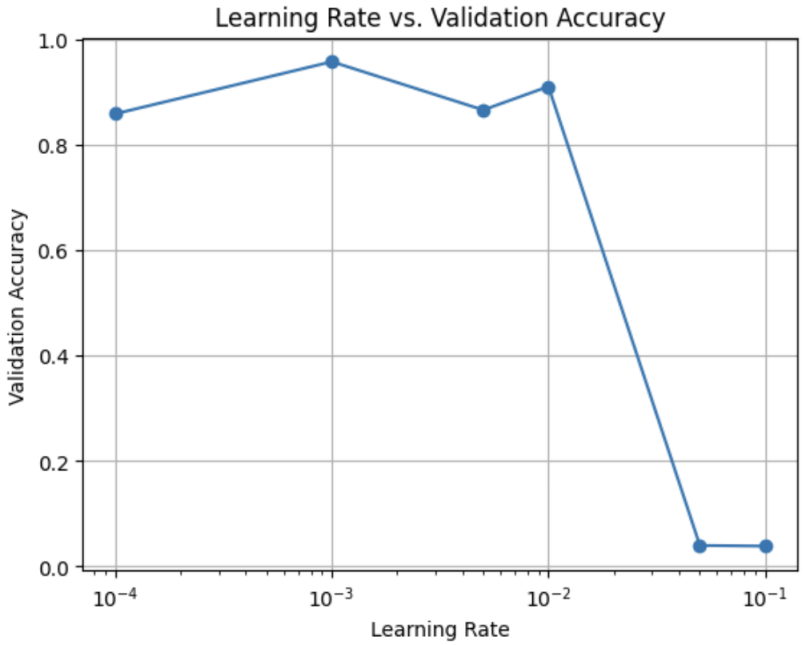

### 4.3 Experiment 3: L2 Regularization

In this experiment, we trained our 2-layer MLPs with L2 regularization. We decided to train the model with different values of $\lambda$ each time to see how it affected the model's performance on the testing dataset. The $\lambda$ values utilized were 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005 and 0.0001. For each $\lambda$ we trained our model with 5 epochs. We observed that for both $\lambda = 0.1$ and $\lambda = 0.05$, the test accuracy of our regularized model was inferior to that of our non-regularized model (2% test accuracy and 59% test accuracy for $\lambda = 0.1$ and $\lambda = 0.05$ respectively, versus 76% for the non-regularized model). We believe that this is because a high value of $\lambda$ induces a strong penalty during regularization, and prevents effective learning by the model. For $\lambda = 0.01$, 0.005, 0.001, 0.0005 and 0.0001, the test accuracy of the regularized model remained consistent compared to that of the non-regularized model, both having a test accuracy of approximately 76%. We believe that this is because our initial model was not overfitting the training data in the first place, explaining why L2-regularization did not yield any improvement.
Out of curiosity, we also applied L2 regularization to our 1-layer MLP (with different $\lambda$ values again). We observed similar behaviour. That is, poor accuracies for $\lambda = 0.1$ and $\lambda = 0.05$ and no significant difference between the test accuracies of our regularized models and the test accuracy of our non-regularized model for values of $\lambda \leq 0.01$ (around 79% accuracy for both regularized and non-regularized models). These observations led us to conclude that our non-regularized one-hidden-layer MLP was not overfitting the training data in the first place just like our non-regularized two-hidden-layer MLP. This conclusion is later confirmed in our experiments in section 4.6.

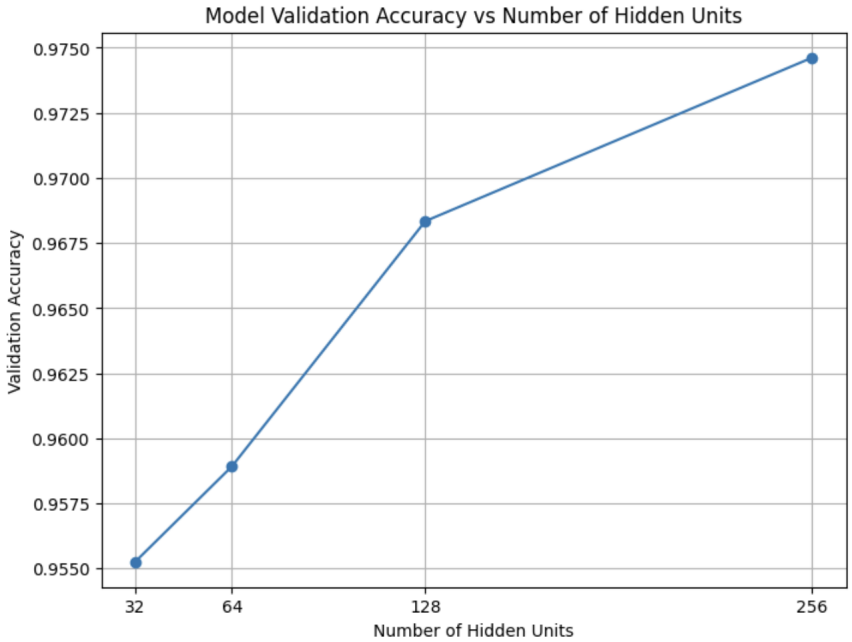
### 4.4 Experiment 4: CNN implementation and MLP comparison

In experiment 4 we used the Keras TensorFlow library to create a Convolutional Neural Network with 3 convolutional and 2 fully connected layers. For the convolutional hyperparameters, we started with arbitrary values. We set the number of filters to 8, the kernel size to 3, the number of strides to 1, the padding to 'valid' or no padding, and the activation function to ReLU. For each subsequent convolutional layer we doubled the number of filters, so the first convolutional layer had 8 filters, then 16, then 32. We also opted to add a max-pooling layer after the last convolutional layer, to speed up the training time of our model.
After creating our CNN we experimented with the learning rate to determine the best one. We set the number of units in the fully connected layers to 32, and all other parameters stayed as described above. We tested the same set of learning rates we had used for MLP, namely [0.1, 0.05, 0.01, and 0.005]. However, after running our experiment and getting poor results for larger learning rates, we decided to modify our set of learning rates. We ran tests on rates in the set [0.1, 0.05, 0.01, 0.005, 0.001, 0.0001], so we could include smaller learning rates than

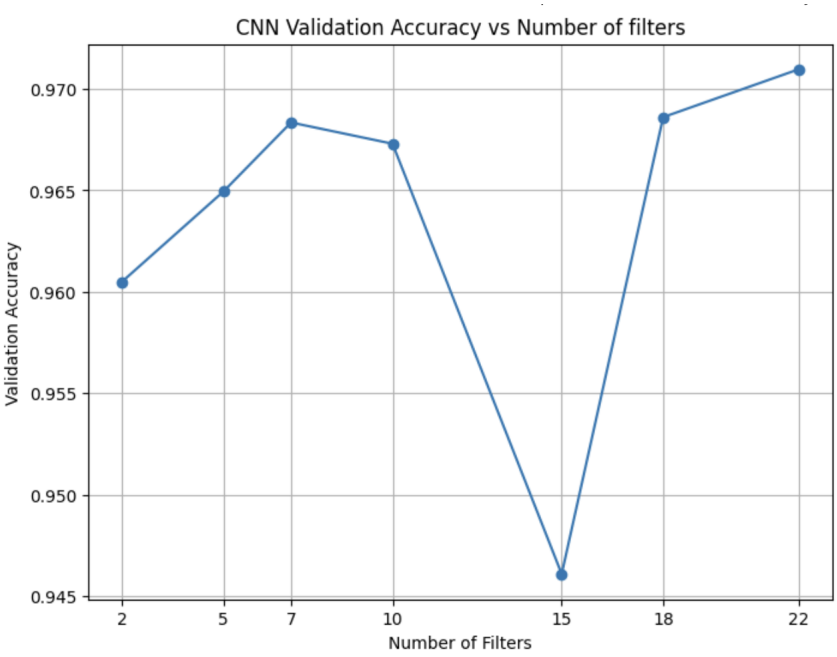
we did for MLP. We found that a learning rate of 0.001 gave us the best validation accuracy of 95%. The results of this experiment are pictured in the graph below.




We then went on to test 32, 64, 128, 256 hidden units in the fully connected layers, as we previously did with our MLPs. We kept all other parameters as described above and used our best learning rate of 0.001. We got the best validation accuracy of 90% with 256 hidden units when we trained over 2 epochs. The plot below shows the model accuracy vs the number of hidden units with the aforementioned hyperparameters.




After doing this testing we wanted to further investigate the effects of the CNN hyperparameters on accuracy. We decided to experiment with the number of filters initially passed to the CNN. We set the number of hidden units to 256 since that gave us the best accuracy in our previous experiment, and left all other hyperparameters as they previously were. We then varied the number of filters used in the set [2,5,7,10,15,18,22]. We found that this test was not very informative, as most of the numbers of filters we tested gave accuracies within 1% of 96%. Overall, 22 filters gave us the highest validation accuracy of 97%. Below is a plot of how the accuracy was affected by the number of filters.




Finally, we tested our tuned model with a learning rate of 0.001, 256 units per fully connected layer, and 22 filters. We ended up with a test accuracy of 90% for our tuned CNN. Comparing these results with our results from our best MLP model (79.3% accuracy), is it clear that the CNN performs much better. This is to be expected as CNN models are generally much better at image classification tasks than MLPs because they take into account the dimensional information from the image data, while MLPs must flatten images into vectors, thus losing valuable information.

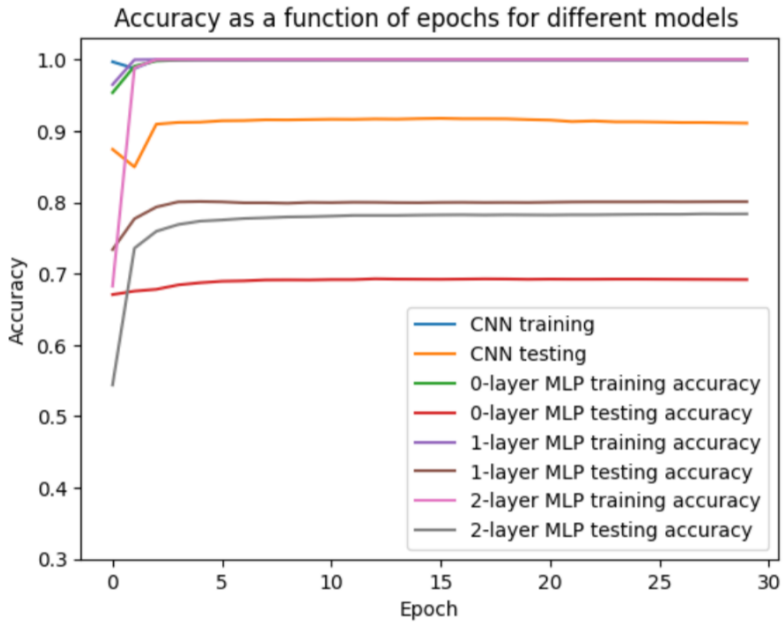
### 4.5 Experiment 5: Improving our best MLP and comparison with CNN

The goal of experiment 5 was to test different combinations of hyper-parameters based on previous experiments to find the MLP model architecture that gave the best accuracy and compare it with our tuned CNN model. The first analysis we did was to test the model depth. From section 4.1, we know that shallower models(excluding the zero-layer one) not only converged better than deeper models but they also perform better on the test data. We further tested models with 3, 4 and 5 hidden layers, 256 units per layer, 5 epochs, a learning rate of 0.1, and ReLU activation, and found that those models could not converge. We tried to make them converge using 8 epochs and Leaky-ReLU as the activation function but only the model with depth 3 converged. We concluded that the best depth for our architecture was the single-layer MLP. We further investigated the effect of the width on the accuracy. As observed in section 4.1, wider models tended to perform better. We experimented with widths of {128; 256; 512; 1,024} units per layer. We saw the highest accuracy with the 1-layer model using 1,024 units per layer. We opted to use ReLU as our activation function as section 4.2 showed no improvement with Leaky-ReLU for shallow models, and this model only has one layer.

The test accuracy obtained for our model with one hidden layer, 1,024 hidden units, and using ReLU as the activation function was 80.26%. This is better than any accuracy of our previous MLPs, however, it is still 10% less accurate than our tuned CNN. Again, this is because MLPs do not take the dimensional aspects of the data into account.

### 4.6 Experiment 6: Testing and training accuracy of all models as a function of epochs

In experiment 6 we plotted the testing and training accuracies of each of our MLPs and our CNN as a function of epochs. Ultimately, we were looking to see how much our models should be trained before they start to overfit the training data. We tested from 1-30 epochs for each model, and ultimately we found that though accuracy increased significantly at first, after 10 epochs the model's accuracy plateaued to the point where it was not worth the extra training time. This is what previously motivated our choice of 15 epochs on tests for our models in section 4.1. The graph below shows the model test and train accuracy vs the number of epochs trained for 0-layer, 1-layer, and 2-layer MLPS, and our CNN.




### 4.7 Additionnal Results

Some additional results that were obtained in this work are worth mentioning. The derived gradients were compared to gradients obtained from numerical approximation using the value $\epsilon = 10^{-8}$. We then compared the gradients for the biases and the weights using the mean squared difference between the back propagation derivation and the approximation. With a model with two hidden layers, we obtained the following results:

Mean Square error between Numerical Gradients and Derived Gradients for a two-layer model

| | Weights Error | Biases Error |
|---|---|---|
| **Output Layer** | 0.043123 | 0.001764 |
| **Layer 2** | 70491.290438 | 0.094834 |
| **Layer 1** | 711808.704007 | 0.000276 |

These results show a significant increase in the error of the gradients of the weights for deeper layers. This seems to suggest that the error for the weights accumulates from previous layers or even gets amplified.

## 5 Discussion and Conclusion

Overall, we had some expected and some unexpected results. The first unexpected result was in experiment 1. Our 1-layer MLP gave us the best performance on the test set. As previously mentioned this surprised us because deeper models generally learn patterns in the data better. Further, we know it is not the model overfitting, because we got better training and validation accuracies on the 1-layer MLP than on the 2-layer MLP. This is likely due

to the larger errors in gradients we observed in deeper layers. An interesting lead for future experiments would be to investigate the discrepancy between the numerical and derived gradients for deeper layers, and further to find a way to reduce this propagation error of the back-propagation algorithm. This would surely improve the accuracy significantly for deeper models.
In experiment 2, we found no difference in accuracy between ReLU and Leaky-ReLU, however, this was unsurprising. In the future, we would be interested in varying the leakage constant to see if it could have any impact on our results.
In experiment 3, L2 regularization did not result in any significant improvement to the testing accuracy of either one of our best two-hidden-layers MLP and best one-hidden-layer MLP. This led us to conclude that our models were not overfitting the training data in the first place. This conclusion was later confirmed in experiment 6 were we saw that our models were not overfitting the training data even after 30 epochs.
In experiment 4, when comparing the CNN with our MLPs, both the models from experiment 1 and our best architecture, were as expected, with CNN outperforming all MLP models. We were surprised to see our models never overfitted, even when trained on 30 epochs. While we would have been interested in comparing with many more epochs, it took quite a while to train our models with a large number of epochs.
In regards to training, there are a couple of things we would like to discuss. Firstly, to train our models, we added an extra criterion to stop the gradient descent. We passed our fit function the validation data and an integer number $k$. For each batch iteration, we logged the validation loss and stopped the descent if the average validation loss over the passed $k$ iterations was smaller than the validation loss of the current batch. The idea behind this criteria is that assuming the validation and the test data come from the same population set, the gradient will stop when the model starts overfitting. This stopping criterion was already tested in our previous work on logistic and multi-class regression and was shown to work very well for those models. However, it was quite inefficient for the models in this project. The training time was more than doubled when using this method, and most of the time, the descent was stopped too early.
In terms of large training time, another issue we encountered in this project was the extremely long training time. We found this was caused by the computations of the gradients in the linear layer. The weights gradient $\frac{\mathrm{d}L}{\mathrm{d}W^l}$ of dimension $K^{l+1} \times K^l$ to update the parameters at layer $l$ were computed using matrix multiplication of two 3-dimensional matrices of size $N \times 1 \times K^l$ and $N \times K^{l+1} \times 1$, where $N$ is the batch-size. The result of this multiplication was a matrix of size $N \times K^{l+1} \times K^l$. The weights $W^l$ were then updated using the mean of each $K^{l+1} \times K^l$ matrices. The solution to this problem was to compute the dot product of the transpose of the back-propagated gradient matrix of size $N \times K^{l+1}$, with the matrix of the current input at layer $l$ of size $N \times K^l$, instead of doing the matrix multiplication. Then, for the gradients not to overshoot, it was divided by $N$, which is equivalent to taking the mean of each single input. Using the dot product meant we could avoid broadcasting, which was taking a very long time and thus making the training very long. Fixing this took our training time from about 4 minutes per epoch to 7 seconds.

## 6 Statement of Contribution

Trystan worked on the model implementation, experiment 5, and he contributed to the report. Mira conducted experiments 1, 2, and 4, and wrote a majority of the report. Gaspard worked on the model implementation and conducted experiments 3 and 6.

## 7 References

1. Kumar Mahata, Tushar. “A Comparative Analysis of American Sign Language Recognition Based on CNN, KNN, And Random Forest Classifier.” Daffodil International University, 2021.