Assignment 3: Image Classification with Multilayer Perceptron and Convolutional Neural Network

Bonaventure Dossou, Mat Vallejo, Achraf Maine

March 28, 2024

Abstract

This paper explores the implementation and evaluation of neural networks using the Sign Language MNIST dataset. The setup and implementations explore fundamental concepts of multilayer perceptrons (MLPs) and convolutional neural networks (CNNs), along with design choices such as hyperparameter tuning, activation functions, regularization techniques, and model evaluation. Experiments are conducted to analyze the impact of model complexity, non-linearity, regularization, mini-batch gradient descent, and network layer depth on classification accuracy. The results highlight the importance of architectural choices and provide insights into the behavior of neural networks across various configurations, showing competitive accuracies across MLP and CNN models under certain conditions.

1 Introduction

The goal of this study is to examine the success of the MLP and CNN neural network architectures using a variety of model design choices. These variations in design include different layer depths, hidden units, activation functions, and the use of regularization. The dataset used for these experiments is the Sign Language MNIST dataset, explained in depth in Section 2. The dataset is considered robust for computer vision tasks and has been used in numerous other studies to evaluate computer recognition of sign language, including research on transfer learning approaches [1] as well as various other neural network architectures [2].

The Sign Language MNIST collection serves as an appropriate dataset for model training and analysis under different circumstances using both MLP and CNN models. We experienced the best model performance (highest accuracy) with MLP when using both layer normalization and batch normalization on a single hidden layer with 256 hidden units, the popular ReLU (rectified linear unit) activation function, and a softmax output layer. Our CNN model reached the highest accuracy using Adam (Adaptive moment estimation) with 3 convolutional layers, and 2 fully connected layers (each with a hidden size of 64).

2 Dataset and Models

2.1 Sign Language MNIST Dataset

The Sign Language MNIST is known as a challenging alternative to the well-known MNIST dataset of hand-drawn numerical figures for computer vision tasks. It consists of images of hands signing the American Sign Language (ASL) alphabet excluding the letters "J" and "Z" which require motion to convey.

The original dataset consists of a training set containing 27,455 images and a test set containing 7,172 images. We load these files into the Python coding environment for further preprocessing. From the training set, we derive a validation set for further model evaluation. Each image is classified in the dataset as the appropriate letter and labeled as such for supervised learning. Once we have our three distinct sets of data (training, testing, and validation) we normalize the data for numerical stability and better generalization across test data. Once this is done, the data has been properly preprocessed and is suitable for use in the models.

2.2 Multilayer Perceptron

Here we define a multilayer perceptron (MLP) with mini-batch gradient descent optimization for training the network. The framework comprises several classes representing different types of neural network layers. The

NeuralNetLayer serves as the base class, defining common attributes like gradients, parameters, lambda coefficients for regularization, and flags for L2 penalty normalization. Subclasses include *LinearLayer* for fully connected layers with random weight initialization and L2 regularization, *ReLULayer* implementing the ReLU activation function, *SoftmaxOutputLayer* for the output layer with softmax activation in classification tasks, *SigmoidLayer* implementing the sigmoid activation function, and *LeakyReLULayer* for the Leaky ReLU activation function. Each layer class provides methods for forward propagation (forward) and backpropagation (backward).

The *MiniBatchMLP* class represents the multi-layer perceptron model and utilizes the forward and backward methods of individual layers to perform forward and backward passes through the network. It also includes functionality to calculate the number of hidden layers in the network and compute regularization based on L2 norms of parameters. In addition to this, the *MiniBatchGradientDescentOptimizer* class implements mini-batch gradient descent optimization for updating the network parameters. It uses a specified learning rate (lr) for parameter updates based on gradients computed during backpropagation, updating parameters for each layer based on the mean gradient of the mini-batch. Test results are stored in a data frame and saved to CSV files for further analysis.

2.3 Convolutional Neural Network

For our second model, we design a Convolutional Neural Network (CNN) using PyTorch. The CNN architecture comprises three convolutional layers and two fully-connected (dense) layers. The optimization is performed using the Adam optimizer along with cross-entropy loss. The code conducts experiments with different batch sizes and dense layer sizes to find an optimal configuration.

The *BasicCNN* class initializes the model with specified parameters such as dense size, number of filters, kernel size, and image size. The CustomDataset class is used to preprocess and load the dataset for training, validation, and testing.

The training process involves iterating over various batch sizes and dense layer sizes. Within each iteration, the model is trained using the "train model" function, which performs forward and backward passes, updates model weights, and computes training, validation, and test metrics. Test results are stored in a DataFrame and saved to CSV files for further analysis.

3 Implementations and Results

Multilayer Perceptron (MLP)

Task 3.1

For this task, we created three separate instances of the MLP model: One with no hidden layer, one with a single hidden layer, and one with two hidden layers. Each instance cycles over a variety of batch sizes (2, 16, 32, 64) and hidden units (32, 64, 128, 256). For the models with hidden layers, the ReLU activation function is used. All versions use a softmax function for output classification.

From Figures 3 and 4, we can see that both 0 & 1 hidden layers, perform best with a batch size of 2 and 256 hidden units. When it comes to the 2 hidden layers of MLP, the best performance is achieved with a batch size of 32 and a hidden size of 128. The highest accuracy is achieved with one hidden layer and 256 hidden units. The instance with no hidden layers performs relatively well considering its structural simplicity, and we see a notable decrease in accuracy in the instance with two hidden layers. This is likely due to a corresponding decrease in the model's ability to generalize across new data (overfitted). See Figures 3, 4, and 5 in Appendix A for further comparison.

Table 1: MLP — Mean and Max Accuracies

Model	**Mean Accuracy**	**Max Accuracy**
Model with 0 Hidden Layers		
Hidden Units	Mean Accuracy	Max Accuracy
32	0.523042	0.605549
64	0.527193	0.610290
128	0.496963	0.588818
256	0.527416	0.610987
Model with 1 Hidden Layer		
Hidden Units	Mean Accuracy	Max Accuracy
32	0.158166	0.243307
64	0.496933	0.569576
128	0.574777	0.613915
256	0.601884	0.612939
Model with 2 Hidden Layers		
Hidden Units	Mean Accuracy	Max Accuracy
32	0.064456	0.149052
64	0.317423	0.387479
128	0.485116	0.523006
256	0.179594	0.446737

Task 3.2

For this task, we take our two-layer model from Task 3.1 and create two new implementations, one with a sigmoid activation function and one with a Leaky-ReLU activation function. Task 3.1 established a best batch size = [32] which will be used here and for task 3.3 as well.

As shown in Table 2, the use of different activation functions demonstrates a significant shift in test accuracy. The Leaky ReLU seems to establish better learning with more hidden units, as does the sigmoid instance. However, sigmoid seems to struggle much more with learning in general, likely due to saturation at extreme values impacting gradient descent optimization. See Figures 6 and 7 in Appendix A for further comparison.

Table 2: MLP — Leaky ReLU vs. Sigmoid (2 Hidden Layers)

Model	**Mean Accuracy**	**Max Accuracy**
Leaky ReLU Accuracies		
Hidden Units	Mean Accuracy	Max Accuracy
32 units	0.094216	0.136921
64 units	0.320583	0.420106
128 units	0.511542	0.546012
256 units	0.523307	0.540296
Sigmoid Accuracies		
Hidden Units	Mean Accuracy	Max Accuracy
32 units	0.08639	0.131205
64 units	0.139081	0.222672
128 units	0.176003	0.248048
256 units	0.208081	0.285416

Task 3.3

For this task, we decided to implement the two hidden layer model using ReLU activation. In this test, we implement the training function with L2 Regularization to loop over various values for the weight penalty hyperparameter (lambda coefficient). We noticed that the normal L2 penalty term could get very large (and hence the loss value), so we implemented the normalization of the L2 penalty term (normalization by the total number of parameters). We train the model both with and without L2 penalty term regularization, to examine how the weight penalties (and normalization) interact with it.

There are two important remarks. First, using the L2 regularization showed an increase in performance (comparing the MLP with L2, and the previous MLP with ReLU experiments). Second, we can see that, as the network gets more dense (bigger with more parameters), L2 becomes more important in the sense that the regularization allows the network to get better as it gets more dense. This is the behavior expected with the L2 regularization. See Table 3, Figures 8 and 9 in Appendix A for further comparison.

Table 3: MLP — Accuracies With and Without L2 Penalty Term Regularization (2 Hidden Layers)

Lambda	**With L2 Penalty Term Regularization**		**Without L2 Penalty Term Regularization**	
	Mean Accuracy	Max Accuracy	Mean Accuracy	Max Accuracy
32 hidden units				
0.00001	**0.096509**	0.122699	0.034969	0.072504
0.00010	0.046111	0.069158	**0.039317**	0.053960
0.00100	0.035686	0.056470	0.032373	0.059955
0.10000	0.030322	0.053263	0.037232	0.058979
0.30000	0.050162	0.066509	0.037370	0.055494
0.50000	0.023465	0.051868	0.035779	0.069158
0.70000	0.040014	0.046152	0.026768	0.040853
64 hidden units				
0.00001	0.132337	0.194367	0.190073	0.233687
0.00010	0.175782	0.259760	0.110141	0.134272
0.00100	**0.207448**	0.269939	**0.191701**	0.264083
0.10000	0.055630	0.081846	0.051216	0.087284
0.30000	0.027363	0.077524	0.030173	0.055772
0.50000	0.033533	0.069994	0.033632	0.067485
0.70000	0.039819	0.046291	0.029017	0.058561
128 hidden units				
0.00001	0.321818	0.408254	**0.324085**	0.412019
0.00010	**0.326532**	0.431679	0.295374	0.367401
0.00100	0.315555	0.403653	0.301049	0.399749
0.10000	0.079667	0.184049	0.067973	0.153932
0.30000	0.046807	0.126185	0.039515	0.106804
0.50000	0.032409	0.083659	0.036682	0.088818
0.70000	0.040141	0.078500	0.045105	0.067345
256 hidden units				
0.00001	0.085346	0.144869	**0.440224**	0.520915
0.00010	**0.160699**	0.271751	0.362560	0.426520
0.00100	0.111274	0.190323	0.110326	0.223090
0.10000	0.101885	0.299359	0.108219	0.308701
0.30000	0.048593	0.195482	0.046928	0.179727
0.50000	0.044497	0.128416	0.045240	0.155466
0.70000	0.024511	0.102900	0.037150	0.133993

Convolutional Neural Network (CNN)

Task 3.4

For this test, we implement a CNN using the PyTorch library. The network consists of three convolutional layers, a ReLU activation layer, and two fully connected layers. A softmax layer is added on top, to convert the output logits into probabilities. To train the model we use Adam optimization (Adaptive Moment Estimation) with backpropagation. We can see overall that the CNN outperforms MLP on all batch sizes and hidden sizes. The test returns excellent classification accuracy across all hidden size values as seen in Figure 1 below. This was expected, as a result. We believe CNNs often outperform MLPs in image-related tasks due to several key factors that make them more suitable for handling image data:

1. **Local Connectivity:** CNNs utilize the concept of local receptive fields, where each neuron in a convolutional layer is connected only to a small region of the input image. This design mimics the way the human visual system focuses on local spatial correlations, allowing CNNs to efficiently capture local features like edges, textures, and shapes.

2. **Parameter Sharing:** In a CNN, the same filter (weights) is applied across different parts of the input image. This means that regardless of where a feature appears in the image, the CNN can detect it using the same learned weights. This drastically reduces the number of parameters, making the network less prone to overfitting and more efficient in terms of computation and memory usage compared to an MLP, which would require a separate parameter for every connection.

3. **Hierarchy of Features:** CNNs are structured in such a way that they can learn a hierarchy of features. Lower layers might learn to detect simple features like edges and corners, while deeper layers can recognize more complex patterns by combining the lower-level features. This hierarchical feature learning is particularly suited to image processing, where understanding both low-level details and high-level semantics is crucial.

4. **Spatial Invariance:** Thanks to pooling layers, CNNs can achieve spatial invariance, meaning they can recognize objects regardless of slight variations in their appearance or position within the image. This is critical for image tasks where the same object can appear in different sizes, orientations, or positions.

5. **Efficiency on High-Dimensional Data:** Images, especially high-resolution ones, are high-dimensional data. MLPs, if applied directly to images, would require a vast number of parameters as each pixel would need to be connected to neurons in the next layer, leading to a combinatorial explosion of parameters and computational cost. CNNs, through their use of local connectivity and parameter sharing, are much more efficient at handling such high-dimensional data.

6. **Adaptation to the Task:** CNNs are specifically designed for tasks that benefit from understanding spatial hierarchies and patterns in data, which are prevalent in image processing. In contrast, MLPs lack these design features, making them a more generic tool that doesn't leverage the inherent spatial properties of images.

See Figures 10 and 11 in Appendix A for more details and a comparison with MLP.

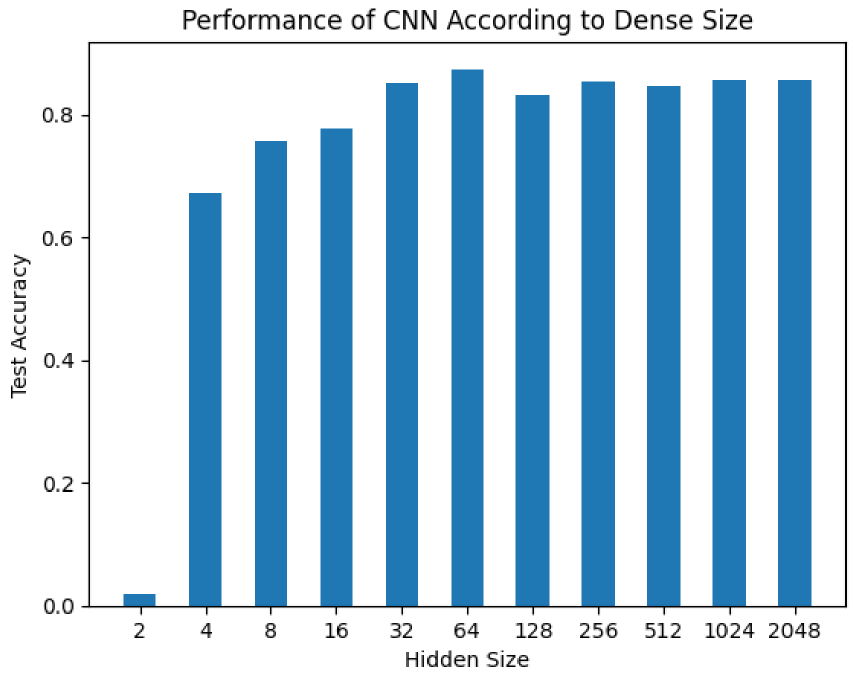

Figure 1: Task 3.4 CNN Accuracy

Fully-Optimized MLP

Task 3.5

To compare with the high accuracy of our CNN model, we implement another MLP using combinations of layer normalization (LN), batch normalization (BN), and dropout techniques to increase test accuracy and generalization across new data. Layer normalization normalizes the data across each independent layer's inputs (initiated after the activation function for ReLU layers), whereas batch normalization intuitively normalizes the data across each mini-batch. These methods allow for better stability, faster convergence, and better generalization to unseen (test) data. By using these methods, we prevent many of the issues encountered in Tasks 3.1-3.3 that either encumbered model learning or led to over-fitting. BN is applied on the batch input, before being fed to the sequence of layers. LN is applied to the input of each layer. Figure 2 shows a sharp increase in MLP accuracy in comparison to the previous tests when using LN and BN techniques, putting it closer to the competitive range of the CNN model in terms of accuracy.

We also implemented Dropout on top of BN and LN. Dropout was applied to the output logits of the network using two additional setups: MLP only with Dropout, and MLP with BN, LN, and Dropout. These setups performed neither better, nor as close to, the MLP with BN and LN. This is likely because the addition of Dropout *over-regularized* the network.

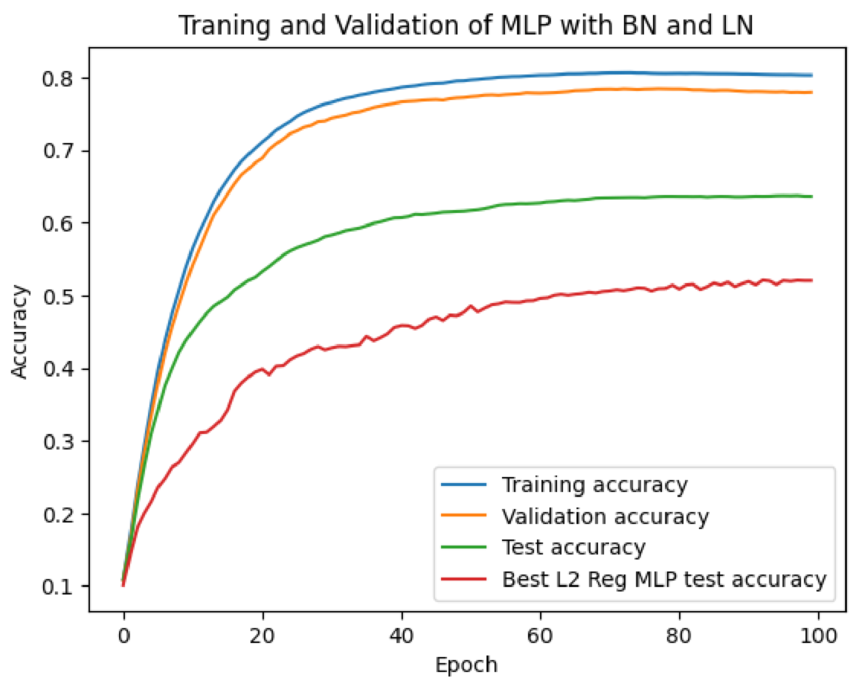

Figure 2: Task 3.5 Fully-Optimized MLP

4 Discussion and Conclusion

The experiments conducted in this paper focus on evaluating the performance of neural networks, specifically Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), using the Sign Language MNIST dataset. The study investigates various design choices such as hyperparameter tuning, activation functions, regularization techniques, and model evaluation metrics. The results shed light on the impact of model complexity, non-linearity, regularization, mini-batch gradient descent, and network layer depth on classification accuracy.

In the context of MLPs, the experiments reveal that a single hidden layer with 256 hidden units using the ReLU activation function and softmax output layer yields the highest accuracy. However, the performance decreases notably with two hidden layers, indicating potential challenges in generalization with increased model complexity. Additionally, comparing activation functions such as Leaky ReLU and Sigmoid demonstrates significant differences in learning behavior, with Leaky ReLU showing better learning capabilities with more hidden units.

On the CNN side, the experiments show excellent classification accuracy across various configurations, highlighting the effectiveness of CNN architectures for image classification tasks. Adam optimization, utilized with backpropagation, contributes to the successful training of CNN models. Furthermore, the study compares the performance of fully optimized MLPs with techniques like layer normalization, batch normalization, and dropout, showcasing improved accuracy and generalization compared to earlier experiments. This demonstrates the importance of incorporating such techniques to mitigate issues like overfitting and enhance model learning.

In conclusion, the experiments provide valuable insights into the behavior and performance of neural networks under different configurations and design choices, offering guidance for choosing appropriate architectures and optimization strategies for image classification tasks.

5 Statement of Contributions

Mat Vallejo - Multilayer Perceptron (MLP), Report
Bonaventure Dossou - Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Report
Achraf Maine - Data Visualization

References

[1] Rathi, Dhruv. 2018. "Optimization of Transfer Learning for Sign Language Recognition Targeting Mobile Platform." *International Journal on Recent and Innovation Trends in Computing and Communication* 6 (4). `https://doi.org/10.48550/arXiv.1805.06618`.

[2] Bala, Diponkor, Mohammad Alamgir Hossain, Mohammad Anwarul Islam, Mohammed Mynuddin, Md. Shamim Hossain, and Md. Ibrahim Abdullah. 2022. "Effective Recognition System of American Sign Language Alphabets Using Machine Learning Classifiers, ANN and CNN." In *2022 IEEE Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, 1–6. Gwalior, India: IEEE. `https://doi.org/10.1109/IATMSI56455.2022.10119336`.

A Appendix

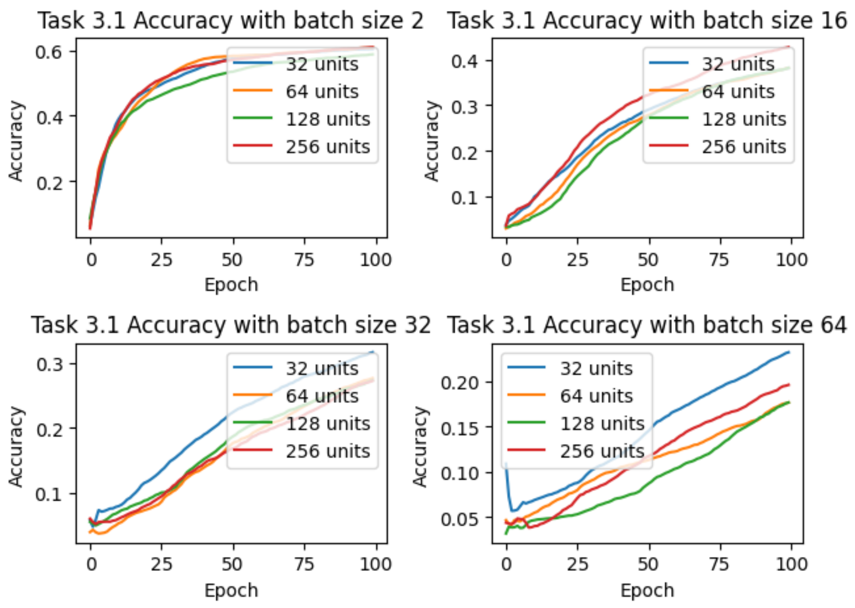

Figure 3: Task 3.1 (No Hidden Layers)

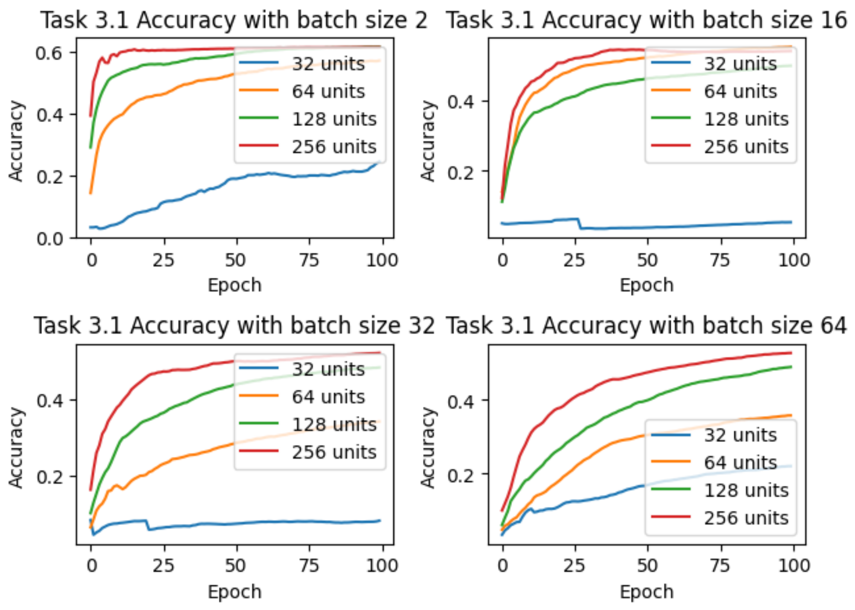

Figure 4: Task 3.1 (1 Hidden Layer)

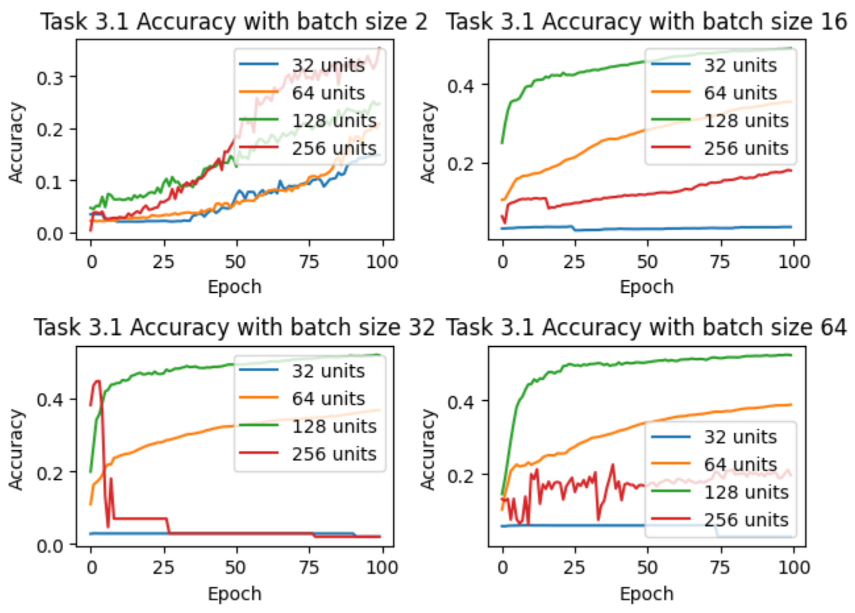

Figure 5: Task 3.1 (2 Hidden Layers)

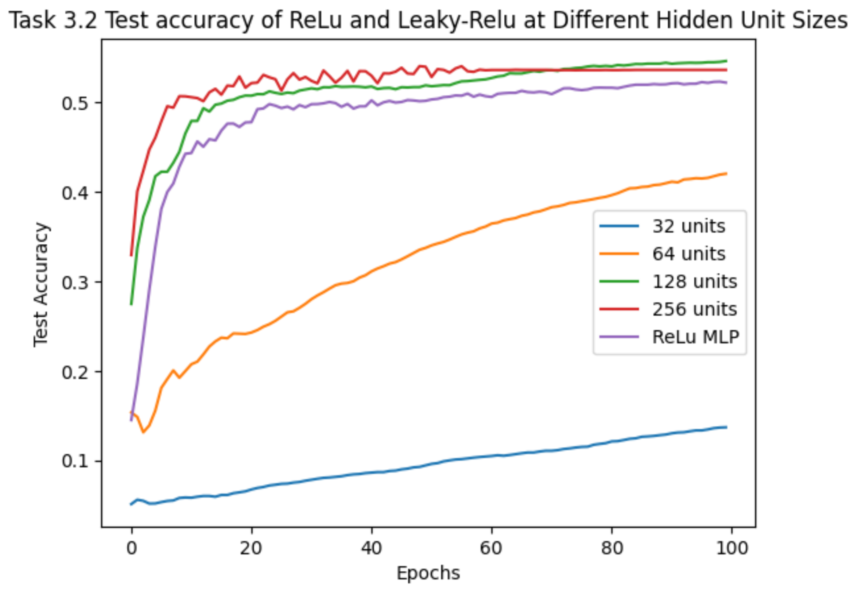

Figure 6: Task 3.2 (Leaky ReLU vs. ReLU)

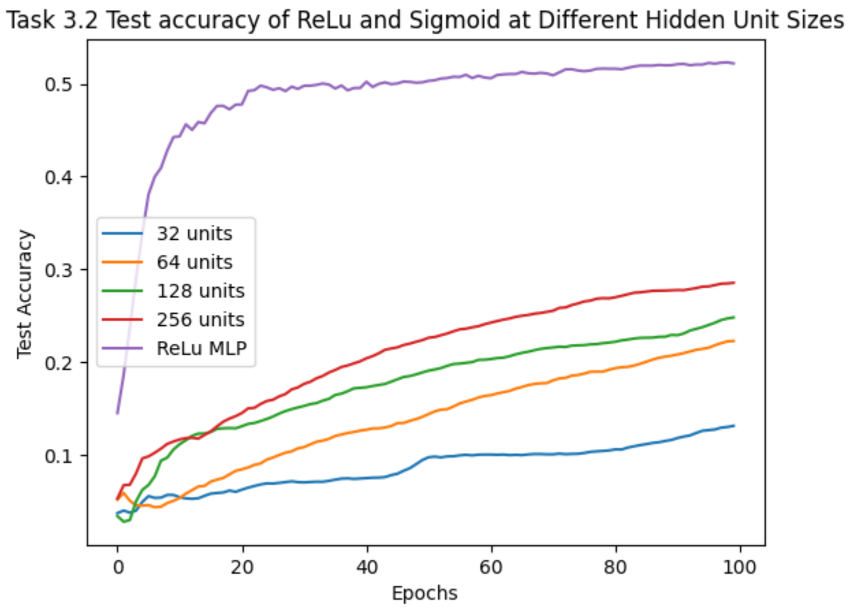

Figure 7: Task 3.2 (Sigmoid vs. ReLU)

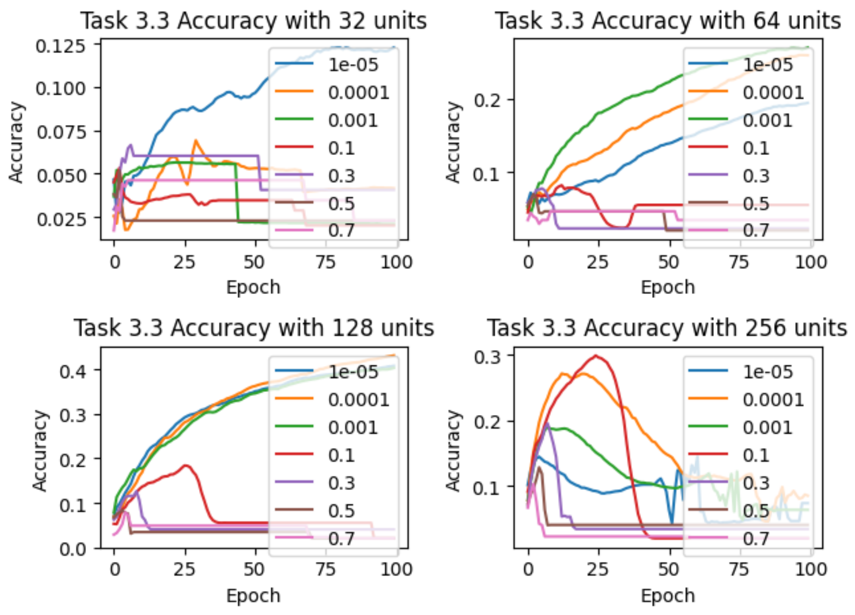

Figure 8: Task 3.3 (With L2 Penalty Term Regularization)

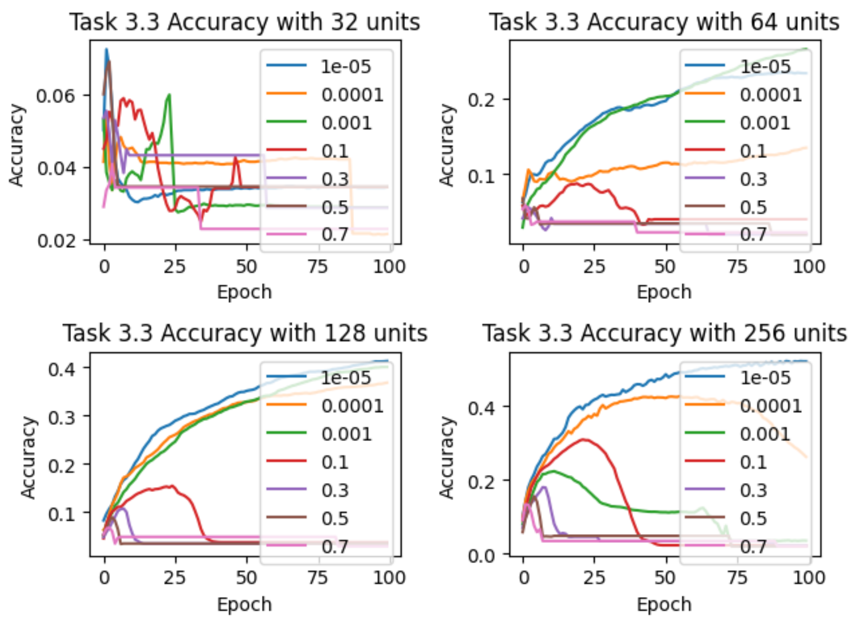

Figure 9: Task 3.3 (Without L2 Penalty Term Regularization)

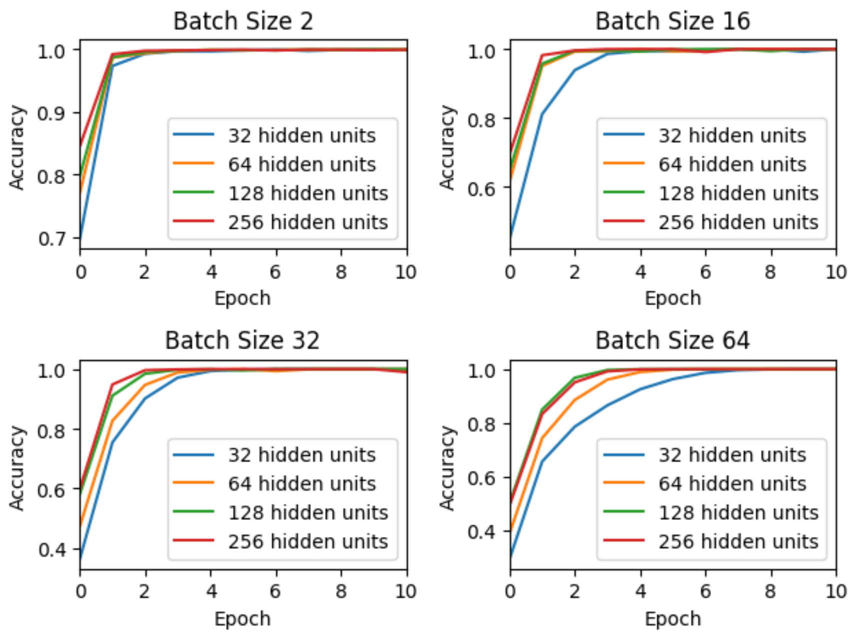

Figure 10: Task 3.4 (CNN Accuracy)

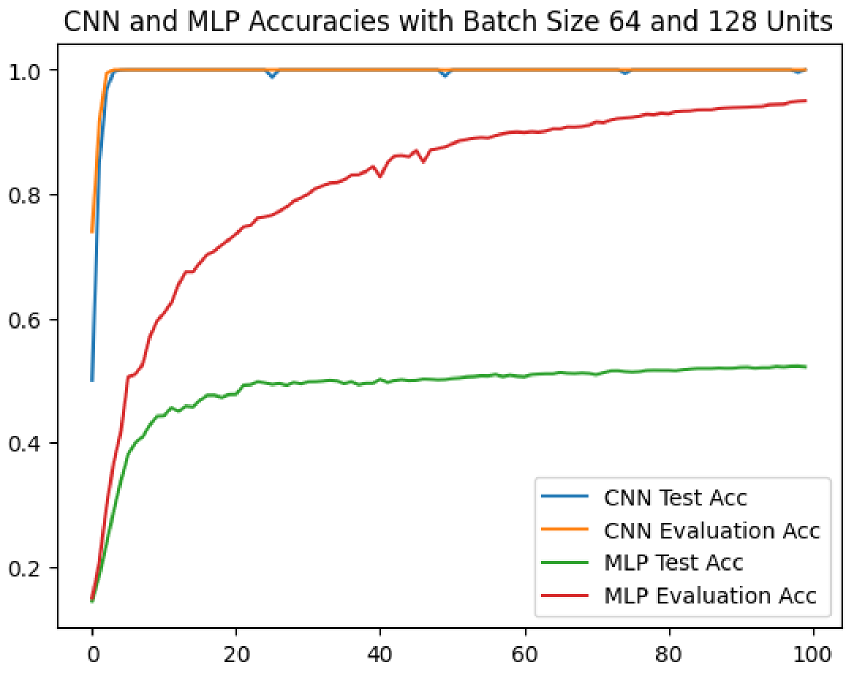

Figure 11: Task 3.4 (CNN vs. MLP)