Digit Classification

Zhansaya Orazbay, Luca Bazzetto

Abstract
The MNIST dataset, a dataset of handwritten digit recognition, comprises 70,000 images of digits (0-9). This project employs a Sequential Dense model with fully connected layers to classify these digits. The objective is to achieve high accuracy while evaluating the efficacy of dense networks for image classification.

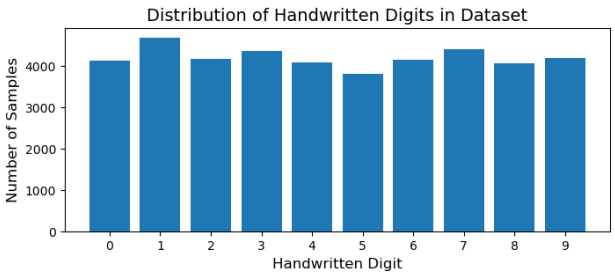
Contents

1 Introduction

The MNIST dataset is one of the most popular datasets in machine learning, often used for testing algorithms that classify handwritten digits. It contains 70,000 images of digits from 0 to 9, with a size of 28x28 pixels. However, for this project, we will utilize a sample of 42,000 images to train and evaluate our models. Each image represents a single handwritten digit, and the goal is to build a model that can correctly identify which digit appears in each image. Recognizing handwritten digits has real-world applications in areas like postal sorting, bank check processing, and document scanning, where automation can make things more efficient and accurate. For this project, a Sequential Dense model was used to tackle the classification task. By using fully connected layers to process and learn from the features in the images, the goal is to achieve a high level of accuracy in classifying the digits, while also exploring the potential of a dense network for image classification tasks.

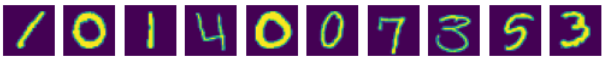
1.1 Exploratory Data Analysis

The first step in the exploratory data analysis was to load and inspect the dataset. We examined the first few rows of the dataset to understand its structure and ensure that the images were correctly aligned with their respective labels. The dataset contains 785 columns: one for the digit label and 784 columns for the pixel values of each image. We then performed a quick summary of the data, including checking for any missing values or inconsistencies, though the dataset appeared to be complete with no missing entries. The distribution of the labels can be visualized in figure 1.1 to confirm a balanced representation of digits across the dataset.



Furthermore, sample images from the dataset are presented in figure 1.1 to illustrate the di-

verse handwriting variations.



1.2 Preprocessing

The preprocessing of the MNIST dataset involved several key steps to prepare the data for modeling. First, the dataset was loaded into a pandas DataFrame, and the features (pixel values) and target labels (digits) were separated. The data was then split into training and testing sets using an 80-20 split. To prepare the target labels for the model, one-hot encoding was applied to the labels, transforming each label into a 10-dimensional vector, where the correct digit is represented by a 1, and all other positions are 0. This encoding is necessary because the model will output probabilities for each of the 10 digits. The pixel values of the images in the MNIST dataset range from 0 to 255. To make the data more suitable for training a neural network, the pixel values were normalised by dividing each value by 255.0, transforming the range to [0, 1]. The original input images are 28x28 pixels in size, but the model requires a 1D array as input. Therefore, the preprocessing function reshapes the images from a 2D array (28x28) into a 1D array with 784 elements ($28 * 28 = 784$). This flattened structure ensures that each pixel value is treated as a distinct feature in the training process. By applying these steps, the preprocessing function ensures that the input data is in an optimal format for the model, facilitating better training and classification results.

2 Methodology

We had always used a Sequential model where we added different Dense layers with different characteristics each. The number of layers in a network significantly influences its capacity to learn complex patterns. We experimented with 1 to 7 Dense layers. This range was chosen to strike a balance between model complexity and computational efficiency, allowing us to observe the effect of increasing depth on model performance. A variety of neuron configurations was explored, ranging from concise layouts like [64, 32, 16, 10] to more expansive ones like [256, 256, 256, 256, 256, 10], as well as configurations with varying numbers of neurons in each layer, such as [256, 128, 64, 32, 16, 10]. This diversity allowed us to assess how the number of neurons in each layer affects the model's abilities. The number of epochs across our models ranges from 10 to 30. A shorter training duration might lead to underfitting, while excessive training could result in overfitting. We aimed to find the optimal training duration for achieving the best model performance. The EarlyStopping was used to monitor the model's performance during training. Specifically, it tracks the loss and halts training if the loss does not improve for 3 consecutive epochs. This prevents the model from overfitting by stopping when no further improvements are observed. We explored batch sizes of 32 and 64 to understand how this parameter affects training efficiency. An appropriate batch size can help the model converge more quickly. Three common optimizers were used: 'adam,' 'rmsprop,' and 'SGD'. Each optimizer employs a different strategy for updating the model's parameters, and our goal was to identify the most effective optimizer for our specific dataset and model architecture. Three different activation functions were tested: 'relu,' 'tanh,' and 'sigmoid', using always the same in each layer for the same model. We applied three types of regularizers: 'l1,' 'l1l2,' and 'l2', to observe how the type and the values in each of them affect the model. For the weights, we explored five different initializers: he_normal ($\sigma = \sqrt{2/n\_input}$), lecun_normal ($\sigma = \sqrt{1/n\_input}$), lecun_uniform, glorot_normal ($\sigma = \sqrt{2/(n\_input + n\_output)}$) and zeros. This initialization can significantly impact the model convergence.

3 Models

3.1 Model 1

The first model tested was a Sequential Dense model consisting of five fully connected layers with decreasing number of neurons (256 → 128 → 64 → 32 → 16). The model begins with a Flatten layer. This layer takes the 28x28 pixel images and flattens them into a one-dimensional vector with 784 features. Each hidden layer used the ReLU activation function and L2 regularization with a regularization value of 0.01. The output layer used a Softmax activation function for multiclass classification and was compiled with the Adam optimizer.

During the layer 1 (256 Neurons), the weights are concentrated around zero, typical of the initial weight distribution with He normal initialization. As the number of neurons decreases, the weight distribution broadens, showing that the model is fine-tuning its parameters and becoming more diverse as it learns more complex features at deeper layers.

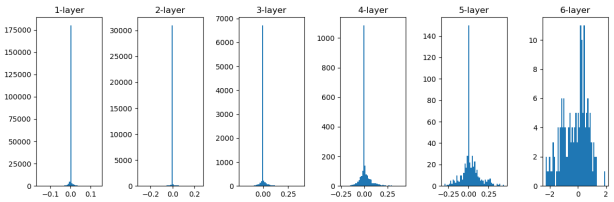


Figure 1: Model 1 weights

The model achieved a final training accuracy of 94% and validation accuracy of 94%, showing that it learns well and generalizes effectively to new data. The training loss is 0.59, and the validation loss is 0.57, the model demonstrates learning with decreasing loss.

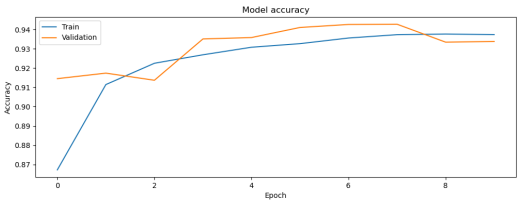


Figure 2: Model 1 accuracy

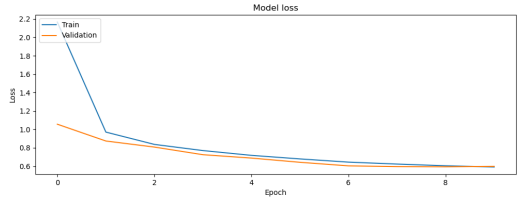


Figure 3: Model 1 loss

3.2 Model 2

Model 2 has the same architecture as the first model, but each layer has 256 neurons, providing a larger capacity for learning.

The weight distribution in the first layer is tightly centered around zero, similar to Model 1, as it uses He normal initialization. Unlike Model 1, Model 2 shows a sudden increase in variation by the final layer. The weight distribution is much wider and includes both negative and positive values. This shows that as the model adds more neurons, the weight values become more diverse, allowing it to represent more complex patterns.

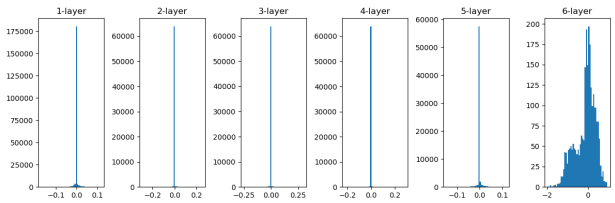


Figure 4: Model 2 weights

While the model learns effectively, as shown by decreasing loss and increasing accuracy, the validation accuracy fluctuates and has the overfitting.

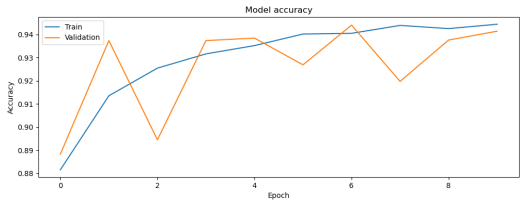


Figure 5: Model 2 accuracy

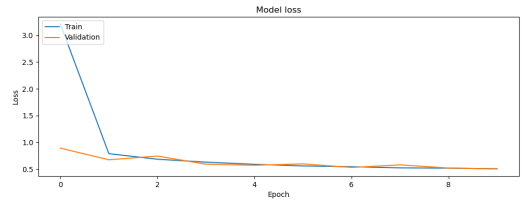


Figure 6: Model 2 loss

3.3 Model 3

Model 3 has the same architecture, but uses L1 regularization with a learning rate of 0.01. Each hidden layer contains 256, 128, 64, 32, and 16 neurons, as the Model 1.

The weights are adjusted differently due to the L1 regularization. It tends to promote sparsity, meaning that more weights are set to zero compared to L2 regularization. As the layers deepen, the weight distributions become more varied, but with L1 regularization, many smaller values are pushed toward zero, causing a larger number of inactive connections. Therefore the model is less flexible, leading to its lower performance.

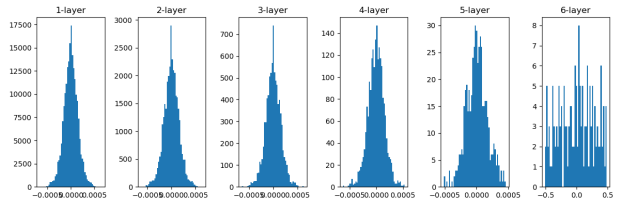


Figure 7: Model 3 weights

The model performs poorly. Probably the accuracy results from random guessing (since there are 10 possible classes). This indicates that the model is not learning effectively, and the L1 regularization might be too restrictive, causing the model to fail to capture meaningful patterns. The loss values are significantly higher. The model is struggling more with fitting the data.

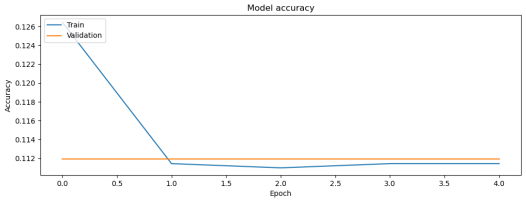


Figure 8: Model 3 accuracy

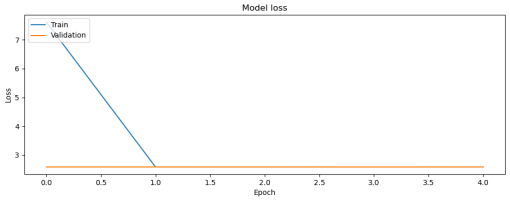


Figure 9: Model 3 loss

3.4 Model 4

Model 4 uses the same architecture but L1L2 regularization.

The weight distributions show some sparsity (with weights pushed toward zero), which is typical of L1 regularization, but the L2 component prevents excessive sparsity, maintaining the model's ability to learn more complex features compared to L1 regularization alone (used in Model 3).

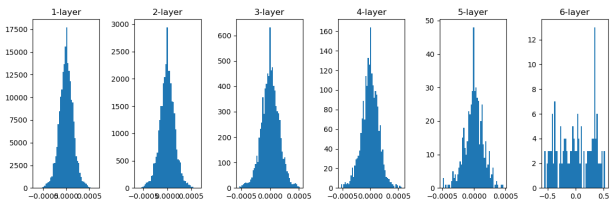


Figure 10: Model 4 weights

The accuracy for Model 4 is very similar to Model 3. These accuracies are almost identical, suggesting that the model is not overfitting but is failing to capture meaningful patterns in the data. The loss values are slightly

better than Model 3, which used only L1 regularization, but still relatively high. This suggests that the L1L2 regularization combination may not have been optimal.

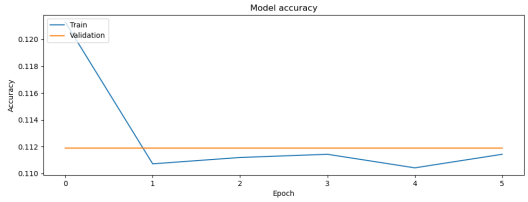


Figure 11: Model 4 accuracy

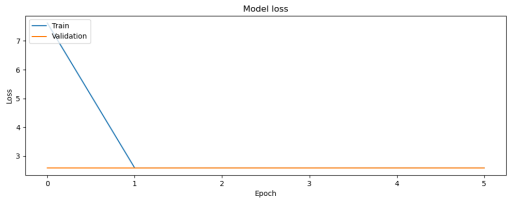


Figure 12: Model 4 loss

3.5 Model 5

Given the strong results of Model 1, we decreased the number of layers and neurons per layer ($64 \rightarrow 32 \rightarrow 16 \rightarrow 10$) while increasing the number of epochs (30), the batch size (64), and the L2 regularization term (increased to 0.1). This was done to determine if the model could achieve similar results with less training time.

The weight distributions show that the model has a higher concentration of weights around zero, but the spread increases as the layers become deeper. The increased L2 regularization value (0.1) helps prevent the weights from growing too large, encouraging better generalization compared to models with smaller regularization values.

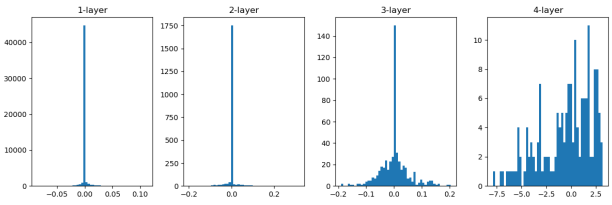


Figure 13: Model 5 weights

The model performs well, with both training and validation accuracies reaching around 82%. This is an improvement over the models with L1 and L1L2 regularization (Models 3 and 4) but still falls short of the performance of Models 1 and 2. Loss curve shows a steady decline for both training and validation loss, indicating that the model is improving. However, the model's loss seems to stabilize at a point where both training and validation loss are close to 1. This might indicate that the model is reaching its maximum capacity for this task.

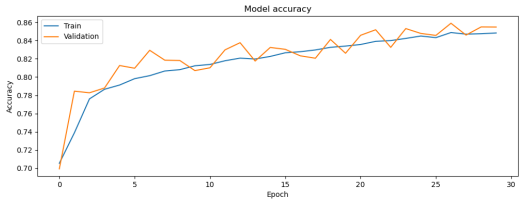


Figure 14: Model 5 accuracy

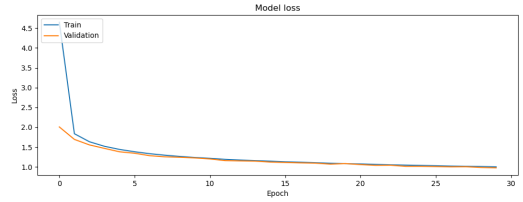


Figure 15: Model 5 loss

3.6 Model 6

The L2 regularization value was decreased to 0.001 for this model, which shares the same architecture and hyperparameters as Model 5. This change was made because the learning steps may have been too large in the previous model.

The regularization value of 0.001 allows more flexibility for the weights to reach larger magnitudes than in Model 5, which had a higher regularization value of 0.1. The model's weights have become less constrained, helping it to better capture complex features.

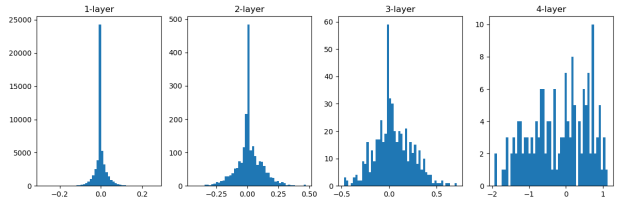


Figure 16: Model 6 weights

Model 6 achieved a high validation accuracy of 97% and a low loss of 0.19, indicating strong performance and successful generalization without overfitting. The reduced regularization value likely enhanced the model's ability to identify patterns within the data, contributing to its superior performance compared to the first and second models.

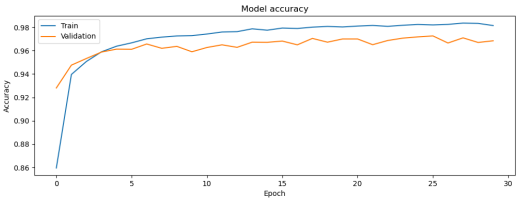


Figure 17: Model 6 accuracy

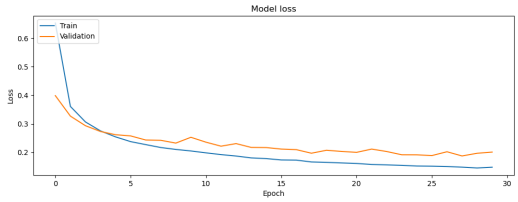


Figure 18: Model 6 loss

3.7 Model 7

This model shares the same architecture as Model 5, but with two key modifications. First, it utilizes the tanh activation function rather than ReLU. This aims to mitigate the vanishing gradient problem by restricting extreme values in the activations. Second, the initializer is changed to 'lecun_normal', which is better suited for models with tanh activation. This helps ensure that the weights are initialized to prevent saturation of the activation function, allowing for faster convergence during training.

The weight distributions in the deeper layers (4th layer) exhibit a wide spread with larger weights compared to the previous models, suggesting that the model is successfully learning complex features while still benefiting from the regularization and appropriate weight initialization.

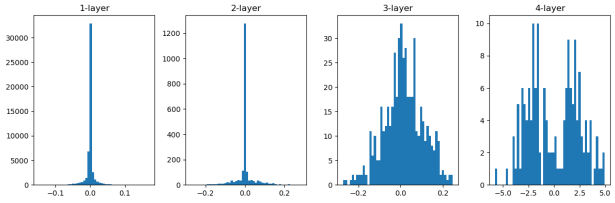


Figure 19: Model 7 weights

The training and validation accuracies are very high (both around 94.5%). Compared to previous models such as Model 5 with a regularization value of 0.1, the performance has significantly improved, which could be due to the use of the tanh activation function combined with the 'lecun_normal' initializer, contributing to better weight updates and smoother convergence. The training loss and validation loss curves exhibit a steady downward trend. The validation loss is quite close to the training loss, which is a positive indication of generalization.

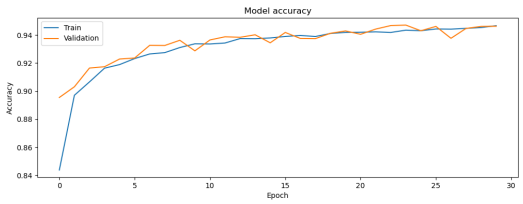


Figure 20: Model 7 accuracy

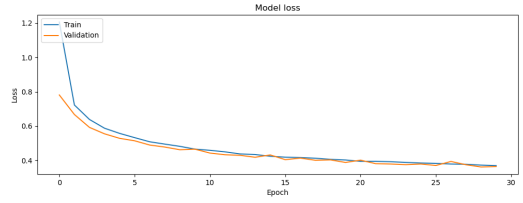


Figure 21: Model 7 loss

3.8 Model 8

This model has the same architecture as Model 7, but it uses the 'lecun_uniform' initializer instead of 'lecun_normal'. This change affects the weight initialization strategy, potentially influencing the model's learning dynamics.

The weight distribution across layers shows a varied spread, but the magnitude of the weights is more uniform compared to Model 7. The 'lecun_uniform' initializer contributes to a more balanced weight initialization, helping the model avoid large weight values early in training and potentially leading to better convergence. Draws samples from a uniform distribution within [-limit, limit], where $limit = sqrt(3/fan_in)$ (fan_in is the number of input units in the weight).

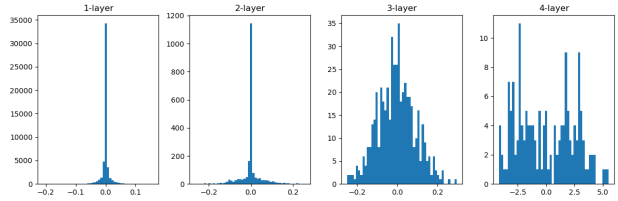


Figure 22: Model 8 weights

Model 7 performance is similar to this model. This means that the lecun initializer, whether lecun_uniform or lecun_normal, does not produce any significant changes in the model.

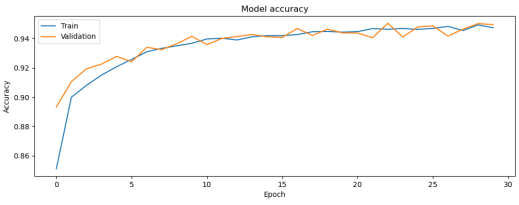


Figure 23: Model 8 accuracy

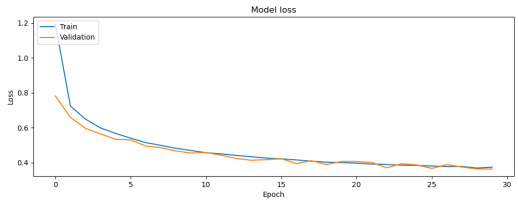


Figure 24: Model 8 loss

3.9 Model 9

Model 9 uses seven dense hidden layers instead of three but with a small number of neurons each (32) and L2 regularisation term equal to 0.1.

The weight distribution shows wider spreads compared to models with fewer layers. The model starts with smaller weights in the initial layers but gradually the weight distribution widens as more layers are added, reflecting the model's increasing complexity.

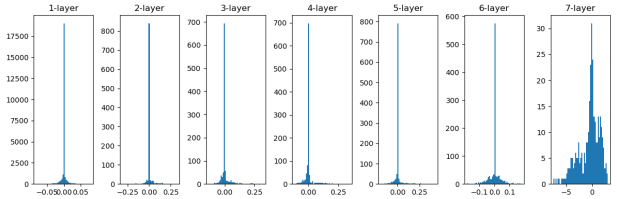


Figure 25: Model 9 weights

While Model 9 has an increase in the number of layers, the higher complexity and insufficient regularization (0.1) appear to be hurting its ability to generalize.

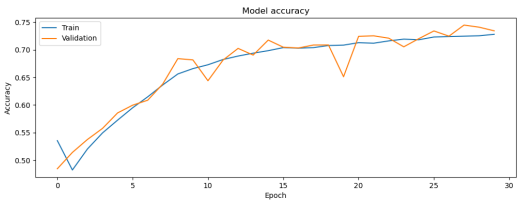


Figure 26: Model 9 accuracy

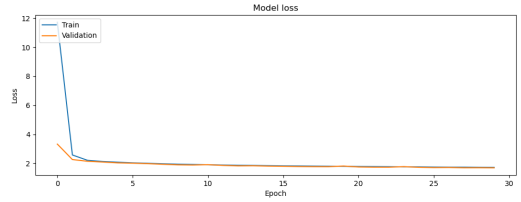


Figure 27: Model 9 loss

3.10 Model 10

The model uses the sigmoid activation function. It uses the 'glorot_normal' initializer, unlike the 'he_normal' initializer. It has six hidden layers with 512, 256, 128, 64, 32, and

16 neurons, respectively and 128 batch size. glorot_normal (also known as Xavier normal initializer) is a weight initialization method that draws samples from a truncated normal distribution centered on 0. Compared to he_normal, which is better suited for ReLU activations, glorot_normal is more conservative and could have contributed to Model 10's underperformance.

Previous models with 'he_normal' or 'lecun_uniform' initialization tended to perform better as they allowed weights to be adjusted more efficiently in the early stages of training. With different structure it struggles because its deep architecture doesn't match well with sigmoid activation. Other models with a similar number of layers and ReLU activations achieved more stable results and better generalization.

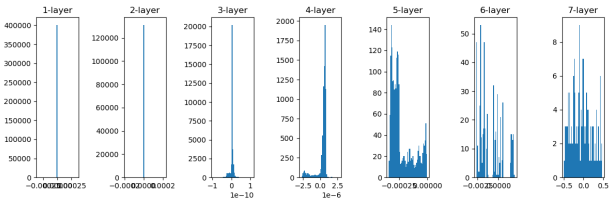


Figure 28: Model 10 weights

The accuracy for both training and validation is relatively low compared to the previous models. The model starts with an accuracy of around 0.11. It seems to have limited learning potential, and the gap between training and validation accuracy is also notable. Both training and validation loss remain high (around 2.302), which suggests that the model is struggling to optimize.

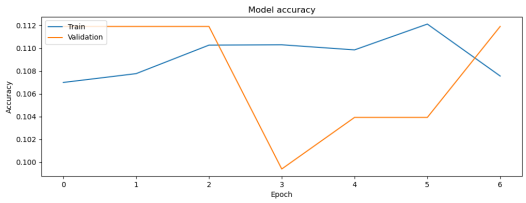


Figure 29: Model 10 accuracy

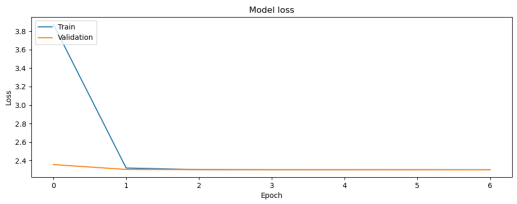


Figure 30: Model 10 loss

3.11 Model 11

Model 11 maintains the same architecture and hyperparameters as Model 10, but decrease in the batch size from 128 to 32. The batch size reduction affects the way the model updates weights during training, leading to a potentially more granular and accurate weight adjustment at the cost of higher variance in the gradient estimates.

The weight distributions in Model 11 show that the smaller batch size, combined with the Glorot Normal initializer, leads to less stable and more spread-out weight values. This could explain why the model struggles with convergence, as the high variance in gradient estimates disrupts the learning process.

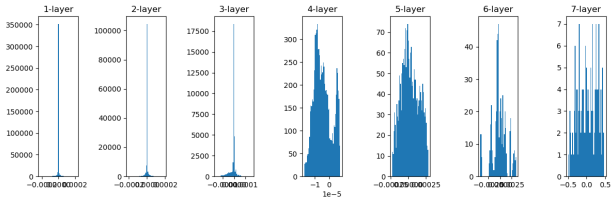


Figure 31: Model 11 weights

Model 11's accuracy is considerably lower. Despite using the same architecture, the smaller batch size may have led to more instability in the gradient descent process, preventing the model from learning effectively. Model 11 struggles to reduce the loss effectively due to training instability caused by the small batch size (32) leading to noisy gradient updates. The use of sigmoid activation, which causes gradients to vanish in deeper layers, makes it harder to optimize the weights.

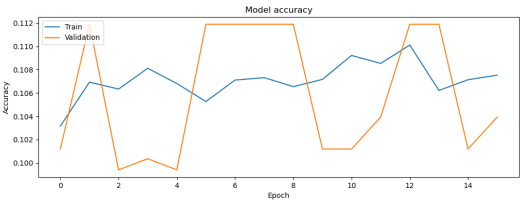


Figure 32: Model 11 accuracy

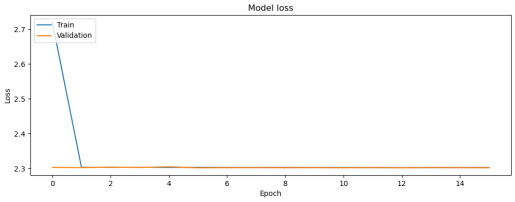


Figure 33: Model 11 loss

3.12 Model 12

It differs in the optimization algorithm. Instead of Adam, it uses RMSprop. Models using RMSprop generally show a bit slower improvement in validation accuracy but maintain a balanced learning rate across parameters.

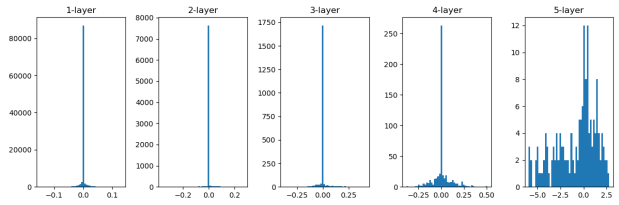


Figure 34: Model 12 weights

This model performs quite well, achieving both high training and validation accuracy of over 94%. The training and validation losses are quite low and show steady improvement. The loss is slightly higher compared to models that used Adam (e.g., Model 6, 7), but the model still performs excellently overall.

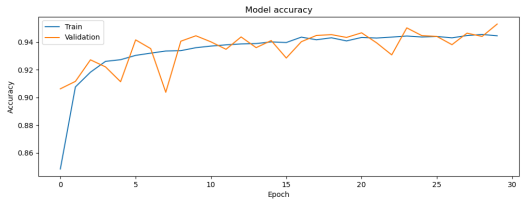


Figure 35: Model 12 accuracy

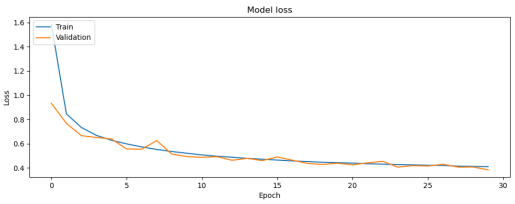


Figure 36: Model 12 loss

3.13 Model 13

Model 13 uses the SGD optimizer instead of Adam, which can result in a less adaptive learning rate, and potentially slower convergence for certain problems. The architecture, regularization, batch size, and activation function are the same as in previous models, with the exception of the optimizer.

The weights for all layers are distributed mostly close to zero with some spread. In higher layers, the weights show a wider distribution, which could be an indication of learning patterns becoming more complex as the network depth increases.

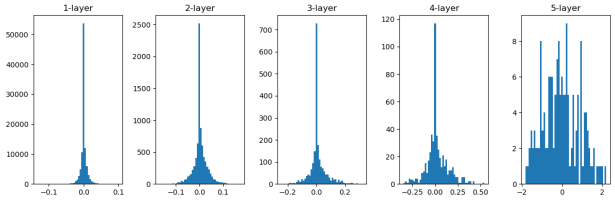


Figure 37: Model 13 weights

The SGD optimizer in Model 13 resulted in good accuracy and loss values, but it generally requires more epochs and adjustments for fine-tuning compared to the Adam optimizer used in earlier models. The loss curves in Model 13 steadily decreased, similar to other models. However, this decrease was more gradual than the rmsprop in the previous model.

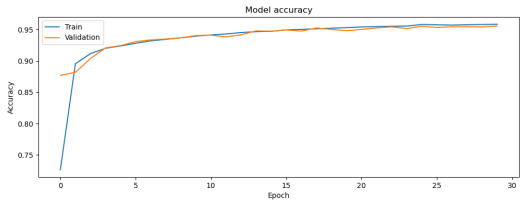


Figure 38: Model 13 accuracy

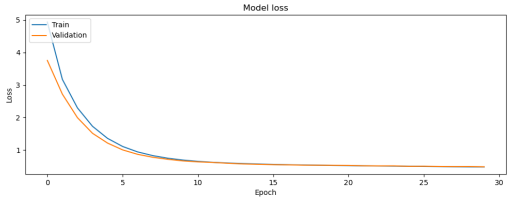


Figure 39: Model 13 loss

3.14 Model 14

Model 14, similar to Model 6 in other aspects, introduces batch normalization after each dense layer. This normalization technique adjusts layer outputs for improved training stability and faster convergence, which is particularly beneficial in deeper networks. The model utilizes the Adam optimizer, 'he_normal' weight initialization, and ReLU activation for hidden layers. Notably, the batch size (128) is significantly increased compared to other models to evaluate the effectiveness of batch normalization.

The weight histograms show a consistent spread with batch normalization applied, especially in the deeper layers. The histograms indicate that the weights across the network are better distributed and have avoided extreme values, which is often the case with batch normalization.

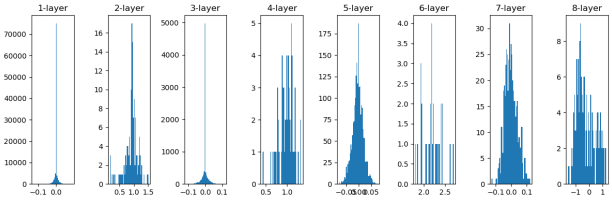


Figure 40: Model 14 weights

The model performs well with a slight gap between training and validation accuracy, indicating good generalization. The loss values are relatively low, with stable convergence.

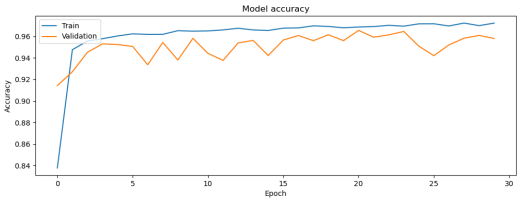


Figure 41: Model 14 accuracy

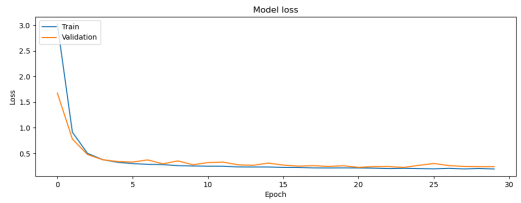


Figure 42: Model 14 loss

3.15 Model 15

Model 15, which uses the 'zeros' weight initialization, is identical to Model 12, which performs well with the rmsprop optimizer.

The use of the 'zeros' initializer may be contributing to poor performance, as it does not allow the network to start with meaningful weights, leading to issues in learning. In comparison, models using 'he_normal' have performed significantly better.

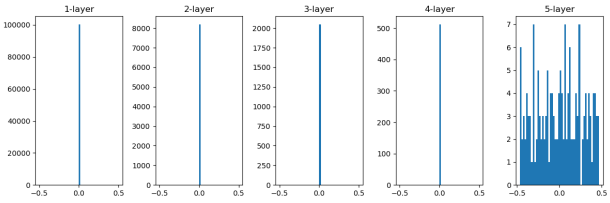


Figure 43: Model 15 weights

Model 15 exhibits poor performance with extremely low and nearly identical training and validation accuracy. The accuracy curves rise steeply at the start but immediately plateau. The loss values for both training and validation are unusually high and similar, which could indicate a problem with the 'zeros' weight initialization. Due to a lack of loss

improvement for more than 3 epochs, training stopped prematurely at epoch 11 instead of running for the full 30 epochs.

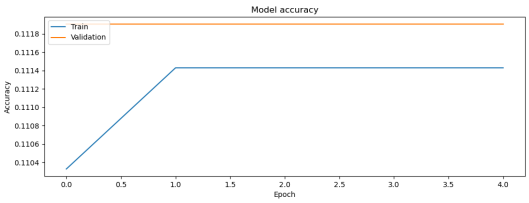


Figure 44: Model 15 accuracy

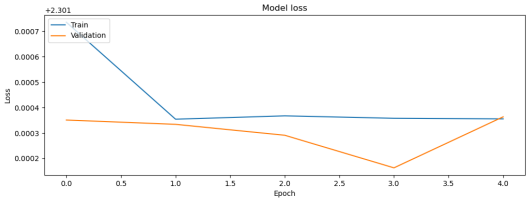


Figure 45: Model 15 loss

4 Data Augmentation

For two of our models, we employed data augmentation techniques to artificially increase the size of our training dataset. This involved applying transformations such as rotation, flipping, and scaling to the existing training samples. The data augmentation parameters were set as follows: randomly shift images horizontally by 10% of total width, randomly shift images vertically by 10% of total height, and randomly flip images horizontally). The figure 46 showcases the effects of data augmentation. The first row displays original images of handwritten digits, while the second row shows the same digits after applying data augmentation techniques. It is evident that the augmented images have undergone various transformations. Some digits are rotated, some are shifted slightly, and others are flipped horizontally. These alterations introduce variability into the dataset, effectively expanding the training data and exposing the model to a wider range of possible digit representations. This, in turn, can improve the model's ability to generalize and accurately classify unseen handwritten digits.

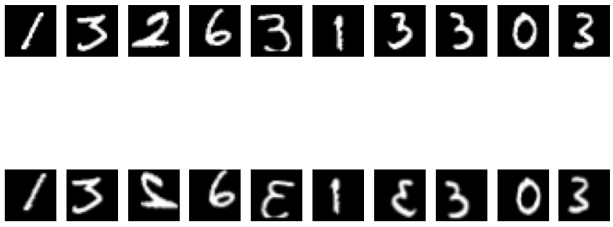


Figure 46: Data Augmentation

4.1 Model 1 D.A.

The model is constructed in the same way as Model 9, one of the worst-performing models, but it is trained on an augmented dataset.

The model's capacity to learn the underlying patterns in the data may be limited by the use of only 32 neurons in each of the six hidden layers. Additionally, the application of a 0.1 regularization term appears to limit the range of weights and keep them centered around zero across all layers, except the final layer which uses a softmax activation.

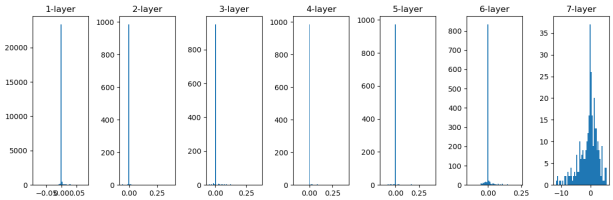


Figure 47: Model 1 D.A. weights

The accuracy plot shows that both training and validation accuracy increase; however, validation accuracy consistently lags behind training accuracy with noticeable fluctuations. The accuracy is approximately 50%, suggesting that the model's performance is similar to random guessing. The loss plot illustrates decreasing training and validation loss, with validation loss consistently exceeding training loss. The validation loss value, approximately 2, indicates poor model performance.

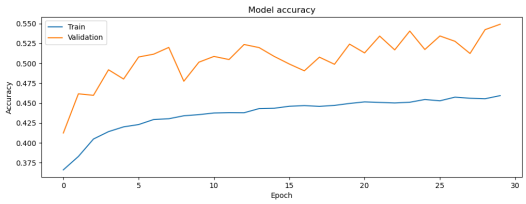


Figure 48: Model 1 D.A. accuracy

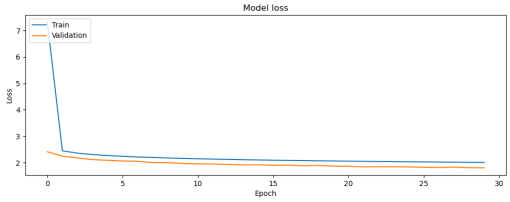


Figure 49: Model 1 D.A. loss

4.2 Model 2

This neural network model has been configured to match the architecture of our highest-performing models. It consists of 5 hidden layers with the following number of neurons in each: 256, 128, 64, 32, 16, and the final one 10. The model is trained using 30 epochs, a batch size of 64, and L2 regularization with a lambda value of 0.01.

The weights in each layer due to the he_normal initializer have a normal distribution centered around zero. The first and second layers exhibit a tighter distribution with a higher concentration of weights near zero because they are learning more general features. In the deeper layers (4th and 5th), the distributions become slightly wider, indicating a greater range of weight values with more specialized feature detection. The final layer (6th) shows a distinct distribution with a noticeable skew, due to its role in producing the final output with 10 classes.

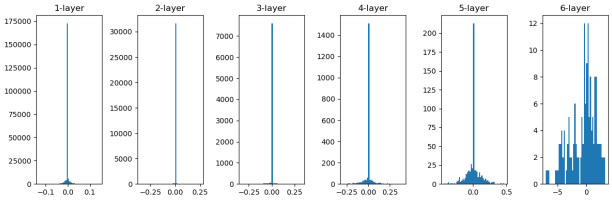


Figure 50: Model 2 D.A. weights

The plots reveal an interesting trend: validation loss is consistently lower than training loss, and validation accuracy surpasses training accuracy. This is likely attributed to the data augmentation applied to the training set, which effectively doubled its size. By increasing the diversity and difficulty of the training data, the model is forced to learn more general features, leading to improved performance on the unseen validation set.

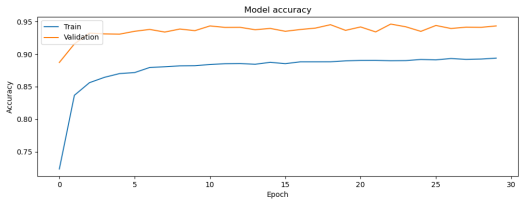


Figure 51: Model 2 D.A. accuracy

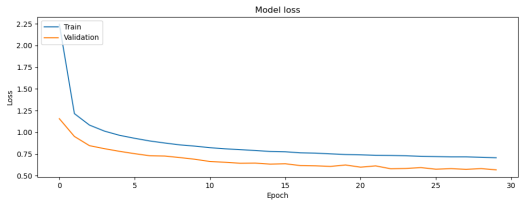


Figure 52: Model 2 D.A. loss

5 Observations

Models with L1 or L1L2 regularization (Models 3 and 4) performed poorly, suggesting that these regularization techniques were too restrictive for this task. Models with L2 regularization generally achieved good performance, with the value of the regularization parameter influencing the model's ability to generalize. Increasing the number of layers did not always improve performance. Models with fewer layers often achieved better results than deeper networks. The ReLU activation function generally produced good results, while the sigmoid activation function led to poor performance in deeper networks (Models 10 and 11), possibly due to the vanishing gradient problem. The Adam optimizer consistently led to good performance and fast convergence. RMSprop and SGD

optimizers also produced good results. The 'he_normal' initializer consistently produced good results, while the 'zeros' initializer led to poor performance (Model 15), highlighting the importance of proper weight initialization. Notably, Model 2 D.A., with an architecture based on the best-performing models, benefited from data augmentation, achieving excellent results. Interestingly, it exhibited a distinct difference between the training and validation results, likely due to the augmented data having exposed the model to a wider range of variations during training, leading to exceptional performance on the unseen validation data.

The detailed results of all models are provided in A.

6 Conclusions

This project provided valuable insights into the impact of various Dense layer parameters on the performance of a sequential model for handwritten digit classification using the MNIST dataset. By systematically exploring parameters such as the number of layers, neurons, epochs, batch size, optimizer, activation function, regularizers, and weight initializers, we observed how each parameter influences the model's ability to learn, generalize, and ultimately achieve high accuracy.

The results demonstrate the importance of regularization to prevent overfitting, the potential for vanishing gradients with sigmoid activation in deep networks, and the impact of weight initialization on model convergence. The success of Adam optimizer is consistent with its wide adoption in the field.

Data augmentation proved to be a powerful technique for improving model generalization, particularly when dealing with limited datasets. The inclusion of augmented samples led to notable performance gains in models that otherwise struggled to generalize.

Further exploration could involve a more extensive hyperparameter search, the use of different architectures (e.g., convolutional neural networks), and the evaluation of various data augmentation techniques.

A Results

Name	Accuray	Validation Accuracy	Loss	Validation Loss	Number of Layers	Neurons per Layer	Epochs	Batch Size	Optimizer	Activation Function	Regularizer	Lambda	Weight initializer
model1	0.94	0.93	0.59	0.60	7	256, 128, 64, 32, 16	10	32	adam	relu	l2	0.01	he_normal
model2	0.94	0.94	0.50	0.50	7	256, 256, 256, 256, 256	10	32	adam	relu	l2	0.01	he_normal
model3	0.11	0.11	2.59	2.59	7	256, 128, 64, 32, 16	10	32	adam	relu	l1	0.01	he_normal
model4	0.11	0.11	2.59	2.59	7	256, 128, 64, 32, 16	10	32	adam	relu	l1_l2	0.01	he_normal
model5	0.85	0.86	1.00	0.98	5	64, 32, 16	30	64	adam	relu	l2	0.1	he_normal
model6	0.98	0.97	0.15	0.20	5	64, 32, 16	30	64	adam	relu	l2	0.001	he_normal
model7	0.95	0.95	0.37	0.36	5	64, 32, 16	30	64	adam	tanh	l2	0.01	lecun_normal
model8	0.95	0.95	0.37	0.36	5	64, 32, 16	30	64	adam	tanh	l2	0.01	lecun_uniform
model9	0.73	0.73	1.72	1.69	8	32, 32, 32, 32, 32, 32	30	64	adam	relu	l2	0.1	he_normal
model10	0.11	0.11	2.30	2.30	8	512, 256, 128, 64, 32, 16	30	128	adam	sigmoid	l2	0.01	glorot_normal
model11	0.11	0.10	2.30	2.30	8	512, 256, 128, 64, 32, 16	30	32	adam	sigmoid	l2	0.01	glorot_normal
model12	0.95	0.95	0.41	0.39	6	128, 64, 32, 16	30	32	rmsprop	relu	l2	0.01	he_normal
model13	0.96	0.96	0.48	0.49	6	128, 64, 32, 16	30	32	SGD	relu	l2	0.01	he_normal
model14_ba	0.97	0.96	0.20	0.24	9	128, 64, 32, 16	30	128	adam	relu	l2	0.01	he_normal
model15	0.11	0.11	2.30	2.30	6	128, 64, 32, 16	30	32	rmsprop	relu	l2	0.01	zeros
model1_da	0.46	0.55	2.01	1.81	8	32, 32, 32, 32, 32, 32	30	64	adam	relu	l2	0.1	he_normal
model2_da	0.89	0.94	0.71	0.57	7	256, 128, 64, 32, 16	30	64	adam	relu	l2	0.01	he_normal