Journal of Applied Technology and Innovation
Vol. 09, Issue 3, 2025, pp. 61-69
e-ISSN:2600-7304-ISSN
DOI: https://doi.org/10.65136/jati.v9i3.32

Enhancing Neural Network Models for MNIST Digit Recognition.

Vinnie Teh [1*], Jason Chin Yun Loong [1], Edward Ding Hong Wai [1], Liew Jie Yang [1], Chew Jin Cheng [1], Zailan Arabee bin Abdul Salam [1]

[1] School of Computing, Asia Pacific University of Technology & Innovation, Kuala Lumpur, Malaysia
*Corresponding Author: tp064168@mail.apu.edu.my

Abstract

Using the MNIST dataset, a standard in computer vision, this study tries to improve neural networks' digit recognition ability. Focusing on elements such as neural network architecture, hyperparameters (dropout rate and training epochs), and their effect on digit identification, it examines a variety of methodologies and strategies. The study identifies hyperparameter settings that significantly increase accuracy. Results indicate that the model with the highest accuracy, ranging from 80.96% to 98.67%, used the Adam optimizer, four hidden layers with Dropout, 0.1 learning rate, and 23 epochs. These discoveries improve MNIST digit recognition and have wider ramifications, including those for document analysis and financial transactions.

Keywords: *Multilayer Perceptron, Training Epochs, Dropout Rate. Overfitting, Underfitting*

1. Introduction

Recognizing handwritten digits remains a canonical difficulty and a crucial milestone in the constantly changing fields of deep learning and machine vision. The MNIST dataset, an extended collection of 28x28 pixel grayscale photographs showing handwritten numbers, has been used to test the effectiveness of different machine-learning patterns. This journal paper's main goal is to explore, innovate, and improve the performance of neural network models for MNIST digit recognition by utilizing a comprehensive combination of methods and methodologies. The MNIST dataset—often referred to as the "Hello World" of deep learning—represents digit recognition problems in the field of machine vision symbolically. This dataset, which consists of 70,000 compiled samples of handwritten digits from 0 to 9, has evolved into the benchmark of choice for assessing the performance of different machine learning techniques, particularly neural networks. The following factors merit careful consideration in the pursuit of this goal: the architectural complexities, which include the fine-grained control over learning dynamics through the wise selection of dropout rates and training epochs. The Multilayer Perceptron (MLP) and feedforward neural network with numerous hidden layers, serves as the fundamental tenet of this research. Based on the complexity of its architecture, this decision enables us to tap into neural networks' latent representational power for improved discriminative ability. Through algorithmic implementation, the expedition taken in this research has travelled through the landscape of several learning rules in hyperparameters guiding the neural network. While holding constant parameters across all trials, such as learning rate and number of hidden layer, we methodically modified the dropout rate and number of training epochs to investigate their effect on the neural network's performance.

Furthermore, the journal article is, in essence, a comprehensive investigation of the MNIST digit recognition challenge with an uncompromising dedication to excellence. The delicate coordination of hyperparameter optimization, advanced training techniques, and neural network architecture form the

core of our methodology. With implications across a wide range of areas, from automated document analysis to the precision-driven world of financial transactions, our findings, which have been hard documented and analytically tested, are poised to redefine the state of the art in MNIST digit recognition.

2. Literature Review

In the context of digit recognition, the first paper proposes the study of using a Random Forest Classifier (RFC) for digit classification, it focuses on elevating the accuracy of results with different hyperparameters based on the performance criteria of stratified-k fold cross-validation. The study was conducted from three key aspects which are the selection of materials, implementation of algorithms and parameter modification. The random forest classifier was implemented using Python programming on a device which uses Windows 11 pro with 12th Gen Intel Core i5-12600 processor and 16gb of RAM. Testing was performed by using a handwritten digit images dataset that contains 1797 images with 8x8 pixels grayscale digit. Two algorithms were employed in these papers which are decision tree and random forest. Decision trees act as a foundation of building blocks in more complex classification, while random forest used the majority voting of decision trees to make predictions and addressed overfitting effectively compared to a single decision tree.

The study by (Ngan et al., 2023) used one fixed parameter of random_state and three hyperparameters of n_estimators, max_depth, max_features by modifying four different values on each to optimize the random forest classifier's performance in digit classification. While random_state affects the producibility of the model, which is useful for comparing the effects of other attributes, n_estimators affects the diversity and accuracy of the model, which in charge of capture more patterns in the data, however might also leads to overfitting if the value is set to be too high, max_depth affects its generalization of the model as higher value brings more accuracy and creates deeper decision tree but also leading to the risk of overfitting at the same time, max_features provides more features to choose when building decision tree as lower value of this can reduce overfitting but reduces accuracy at the same time. Stratified K-folds cross-validation that divides the dataset into a more balanced subsets is being used as its performance criteria to ensure a fair evaluation of the model, addressing potential class imbalance issues.

The result of testing the three parameters with different values resulting the importance of hyperparameter adjustment, especially the sole setting of each hyperparameters as 125 in n_estimators, max_dept to 'None' and max_features to 4 got the higher accuracy. It is worth noting that these parameters weren't the highest nor lowest value that were used for testing, highlighting that simply using higher or lower value doesn't guarantee improved accuracy. Interestingly, even when these parameters that achieved the highest accuracy individually were combined, they did not yield the highest overall accuracy, showcasing that each attribute has a unique impact on the model's performance. In summary, hyperparameter adjustment significantly enhances digit recognition accuracy using Random Forest Classification. This approach contributes to making a model to achieve the best result without being too time-consuming or costly, making it a suitable approach for newcomers in machine learning practitioners.

In parallel, the literature review by Lead et al. (2021) focuses on finding the most accurate architecture for the task by investigating the handwritten digit recognition based on various pre-trained deep learning models. Five pre-trained models from PyTorch were being applied to the study which are from GoogLeNet, MobileNet v2, ResNet-50, ResNeXt-50 and Wide ResNet-50 using a MINST dataset. After pre-processed with the data, Lead et.al (Lead et al., 2021) applied 70000 28x28 pixels of grayscales images dataset that contained labelled images from 0 to 9 in the testing part. The dataset was split into two datasets, 60000 for training and 10000 for testing images on digit classification. During the training process, CNN Resnet-18 was used for its training architecture and algorithm, it processes input images, predicts the outcomes, and then learns to improve from the feedback by comparing the result with the actual one. The evaluation involved using confusion matrices to analyze the top 9 loss images and found that the confusion patterns are quite similar with other digits. The result of accuracy and training time is being considered while evaluating these models and it returned that the Wide ResNet-50 achieved the least error percentage on Top-1 error (0.5278%) and Top-5 error (0.0079%). To be noted, MobileNet v2 also achieved a commendable top-1 error rate of 0.5754% and top-5 error (0.0079%) in just 498 seconds. The transition to CIFAR-10 dataset with same configuration revealed that ResNeXt-50 achieved

the least error rate on Top-1 error(14.0460%) and Top-5 error(0.5300%). To be noted, MobileNet v2 performed its versatility by achieving a top-1 error rate of 15.2780% and top-5 rate of 0.5380% with a smaller model size and faster training. Based on the outcome, it emphasizes how neural network architecture for digit identification is always evolving. The objective of study which is to enhance the neural network for MNIST digit recognition was aligned with Mobile v2's consistent performance across datasets which demonstrated its potential for further investigation of digit recognition.

3. Materials and Methods

3.1 Selection of Materials

1) Source code: The Python programming language, renowned for deep learning research, has been used for the study. Besides, Python has a large selection of tools and frameworks made particularly for developing neural networks and machine learning algorithms. The Pandas library handled data manipulation, which was used to load and handle datasets, rendering it simpler to deal with the MNIST dataset and carry out data preprocessing while NumPy enabled numerical computations to carry out normalization. Keras, which has TensorFlow as its backend and benefits from the fast calculation capabilities of TensorFlow, allowed for the flexible development of neural network models to evaluate architectural alternatives for digit classification.

2) Machine: A computer with Windows 11 (Version 22H2) operating system equipped with AMD Ryzen 7 Pro 3700U w processor and 16 GB of installed RAM was used to carry out this study.

3) Dataset: The MNIST dataset has been used in this study because this dataset serves as a model for several image classification schemes, especially handwritten digit recognition. According to Daniel (2022), MNIST was created from an even bigger dataset, the NIST Special Database 19, that comprises handwritten uppercase and lowercase characters in addition to numbers. Besides, MNIST comprises 60000 handwritten digits for training the machine learning model, and 10000 handwritten digits for model testing. Each digit in the MNIST is retained as a 28x28 pixel grayscale image, and each data has 784 features.

3.2 Selection of Methods

The data selection methods are discussed below:

1) Data Loading: The training and testing datasets (train.csv and test.csv) are loaded using Pandas.

2) Data Preprocessing and Transformation: The label will be separated from the feature data in the training dataset. 20% of the data in the training dataset will be randomly selected to implement cross-validation to avoid overfitting. The remaining 80% of the data will be the training sets to train the neural network model.

3) Neural Network Architecture Implementation: The neural network's input layer includes 784 units (pixel units in a 28x28 image). For multi-class classification, the output layer has ten units, which are digits 0-9.

4) Model Training: Several parameters will be modified to train the Neural Network model and improve the performance of the model. To reduce an identified loss function, the internal model parameters which is weights and biases must be adjusted repeatedly during the training phase. The training data will be tested, and the validation data will be fitted into the model constructed.

5) Model Evaluation: The accuracy, loss, validation loss, validation accuracy and times per epoch will be observed. Accuracy: The percentage of accurately predicted labels in the test dataset. Loss: The percentage difference between the predicted label and the actual label in the test dataset.

   Validation Accuracy: The percentage of accurately predicted labels with the target label in the validation dataset.

   Validation Loss: The percentage difference between the predicted label and the target label in the validation dataset.

   Times per Epoch: Amount of time that is required to complete each Epoch in seconds.

6) Prediction: On the test dataset, predictions are made using the model that performs the best. The trained neural network is used to make predictions on a different test dataset, and the predicted labels and image IDs will be saved.

4 *Algorithm Implementation*

Neural networks are systems of interlinked neurons that resemble the layers of the human brain. Computers may use this to build an adaptive framework that constantly learns from errors.

1) Feedforward Neural Network: A Feedforward Neural Network is an Artificial Neural Network that does not have looping nodes. Input data is fed into the network during the forward pass, and computations pass via the hidden layers to produce an output in output nodes. The network's prediction or categorization is represented in the output. Feedforward neural networks are trained by employing supervised learning.

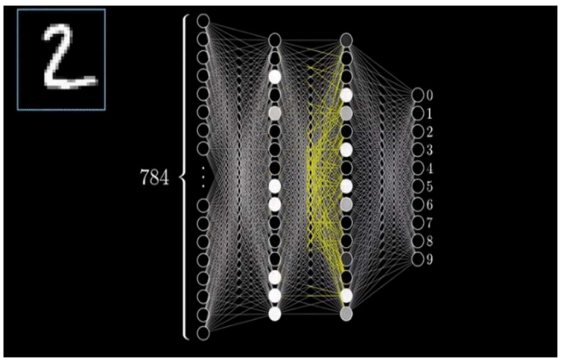

Fig. 1. Recognizing Digits using Feedforward Neural Network. (*Rubentak,2023*).

There are several processes the neural network performs to compute the data. Firstly, input is multiplied by the given weight values. For example, x1*w1 = 2*3 = 6. The demands for the signal intensity of the neuron are established by weights. The impact of input data on the result will be determined by weight value. Secondly, add the bias value to the product value in the prior phase. For example, 6+b1 = 6+1 = 7. Thirdly, the weighted sum will be calculated. Fourthly, by passing the weighted sum to an activation function, the corresponding weighted sum is converted into an output stream (Vihar, 2022). Fig.1 illustrates how the Feedforward Neural Network can learn to categorize the handwritten digits in the MNIST dataset by following these steps. A feedforward neural network's mean square error cost function is a smooth metric used to adjust weights and biases, allowing for incremental improvements for better performance with minimal effect on classified data points. Fig. 2 is used to compute the loss function in neural networks, which determines whether learning process adjustments are necessary.

2) Multilayer Perceptron: A multilayer perceptron refers to an Artificial Neural Network that comprises input, output, and multiple hidden layers with numerous neurons to learn more complex patterns. "Perceptron" means the capacity to see and comprehend images, which mimics human perception (Carolina, 2021). However, the single-neuron perceptron cannot analyze non-linear data. Hence, this issue was resolved upon the introduction of the Multilayer Perceptron. Multilayer perceptron neurons can employ any activation function, as opposed to Perceptron neurons, which demand an activation function that imposes a threshold. The Multilayer Perceptron continually adjusts the network weights and lowers the cost function using backpropagation as its learning approach. The Multilayer Perceptron determines the Mean Squared Error gradient for every input and output set throughout every cycle after computing the weighted sums and applying them through all layers. The weights of the first hidden layer are subsequently modified using this gradient result, thereby propagating it back to the neural network's starting point.

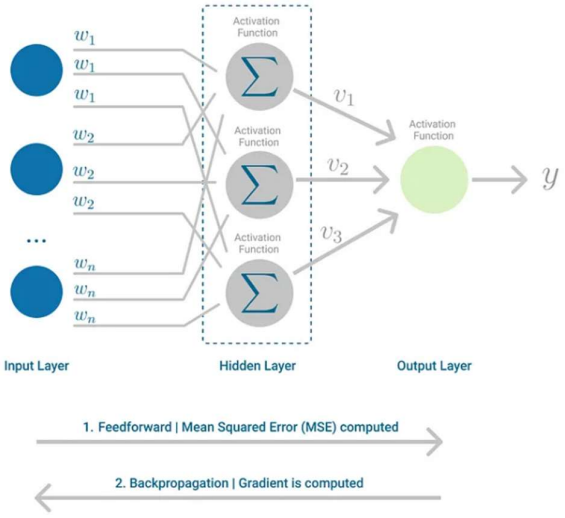

Fig. 2. Cross Entropy Loss (*Carolina, 2021*).

3) Optimizer Algorithm: Adam Optimizer

Adam is an approach for calculating adaptive learning rates that applies individual learning rates to different parameters. This is accomplished by gauging the gradient's first and second moments, which are then used to modify the learning rate for each weight.

5 Purpose

The main purpose of this study is to improve the Neural Network Model to accurately identify handwritten digits. Several neural network models are trained using various modified parameters in the Neural Network algorithm to evaluate the effectiveness of various models and determine which parameter values are best in digit recognition.

6 Parameters

The following parameters should be kept constant throughout the study to provide a fair evaluation of each hyperparameter, listed in Fig.3.

Constant Parameter	
Parameter	***Value***
Activation Function	SoftMax– output layer ReLU (Rectified Linear Unit)– hidden layers
Hidden layer	4
Learning rate	0.1
Batch size	100
Optimizer	Adam Optimizer

Fig. 3. Table of Constant Parameters.

The following parameters will be examined and modified within this study.

1) Number of Training Epochs: The training epochs represent the training iterations in the entire training dataset. It is important to strike a balance between enabling the model to converge to an accurate solution and avoiding underfitting or overfitting. Cross-validation and tracking validation accuracy processes should be performed to determine the optimal value of training.

2) Dropout Rate: Dropout is an approach applied to minimize overfitting during the training process for a neural network. It drives the network to acquire more enhanced features instead of solely being dependent on a particular group of neurons. The dropout rate is the hyperparameter specifying the probability that a neuron may be deactivated during training. The dropout rate has a value range between 0 and 1, with 0 denoting no dropout, indicating every neuron is active, and 1 denoting entirely dropout, indicating all neurons are deactivated.

4. Result and Discussion

4.1 Discussion on Implementation

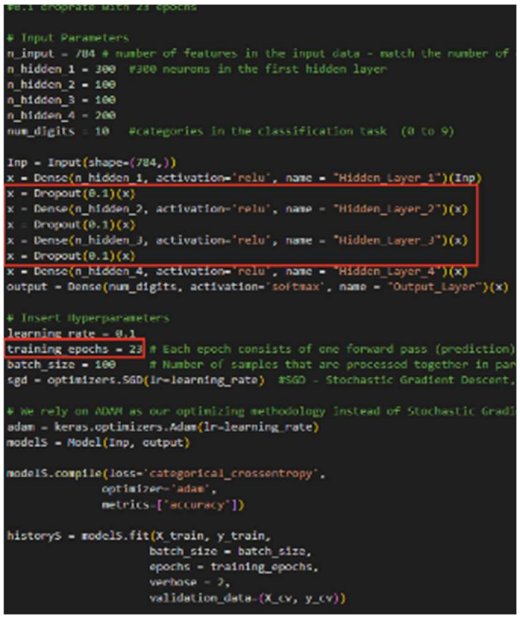

Fig. 4. Code to modify parameters.

Generalization describes a model's ability to adapt and appropriately respond to previously unobserved, new data. To avoid overfitting and underfitting, it is significant to achieve the ideal balance between the complexity of the model and adaptability (Evelyn,2023). The project aimed to configure a model that strikes the right balance between overfitting and underfitting. Thus, the model can be trained to identify handwritten digits more accurately using the MNIST dataset. The original code is from Kaggle. Blocks of code that adjust the parameters have been developed to experiment on while keeping the others constant to see the results of the specific parameter clearly. The use of Google Colab makes this process easier, where the code can be modified by multiple people at the same time. Any changes are easily saved and run in real time.

Fig. 4 is a sample of the code used to modify the two parameters chosen. The first highlighted block is where the dropout rate is changed while the second highlighted block is where the number of epochs is changed. Although the dropout rate can be modified separately for each layer, the same dropout rate is applied for all layers in order to get more consistent results for comparison and optimization. The rest of the code contains set parameters such as the number of neurons in the input layer, hidden layers, and output layer. It also has the activation function used for each layer, the learning rate, batch size, and optimizer used. The purpose of listing out these set parameters line by line is to give a picture that defines everything clearly.

4.2 *Results*

To find the optimal solution, parameters such as the number of epochs and dropout rate have been experimented with in this project. The main results that have been compared based on the training of the models are the accuracy and validation accuracy after the entire training is over. Both these results are utilized to ensure that the data does not go wrong, and a more detailed analysis is obtained. The average

time taken per epoch of the models is roughly the same, showing a very consistent time that is likely attributed to the performance of Google Colab.

Number of Epochs

Model	Number of epochs	Accuracy	Validation Accuracy	Average Accuracy
A	14	0.9946	0.9721	0.98335
B	17	0.9957	0.9771	0.98640
C	20 (Default)	0.9940	0.9724	0.98320
D	23	0.9967	0.9786	0.98765
E	26	0.9974	0.9761	0.98675

Fig. 5. Table of Results for Model trained with different number of epochs.

In training neural network models, the number of epochs is a crucial modified parameter. Research shows that when trained for 23 epochs, the model achieves its best average accuracy of 98.77%. Fig 5. Shows the outcome is better than the default value of 20 epochs and the other number of epochs, demonstrating that more training improves the model's overall accuracy. Underfitting and overfitting principles are implemented here as the model doesn't learn complicated patterns in the data when the number of epochs is too low (for example: 14 or 17 epochs). On the other hand, the model is more likely to overfit if the number of epochs is too high (for example: 26 epochs). When a model learns training data excessively or deficiently, it leads to poor accuracy and begins to pick up unimportant features that cause it to perform poor data in recognizing digit.

Dropout Rate

Model	Dropout rate	Accuracy	Validation Accuracy	Average Accuracy
F	0.1	0.9945	0.9815	0.98800
G	0.2	0.9920	0.9789	0.98545
H	0.3 (Default)	0.9864	0.9786	0.98250
I	0.4	0.9803	0.9758	0.97805
J	0.5	0.9723	0.9754	0.97385

Fig. 6. Table of Results for Model trained with different dropout rates.

The dropout rate determines the likelihood of neurons becoming inactive during each training iteration. Fig 6 and 7 shows higher accuracy and validation are generated by lower dropout rates, such as 0.1 and 0.2, while greater dropout rates, such as 0.4 and 0.5. Based on average accuracy, the ideal dropout rate is 0.1, which results in an average accuracy of 98.80%. Compared to the default rate of 0.3, which has an average accuracy of 98.25%. The lowest dropout rate of 0.1 results in a maximum validation accuracy of 98.15%. That represents a significant improvement, the same as the validation accuracy. Deactivation takes part in the deactivation of neurons during training. Lower dropout rates such as 0.1 and 0.2 keep more neurons active, allowing the model to retain and utilize a broader range of learned features. In contrast, higher dropout rates (0.4 and 0.5) deactivate a significant portion of neurons. The dropout rates must be balanced properly for a model to be generalized. The best option is to set a dropout rate of 0.1. to avoid overfitting and enhance the network's capability to generalize new data. In comparison to the default rate of 0.3, this rate results in a significant improvement in both average accuracy and validation accuracy.

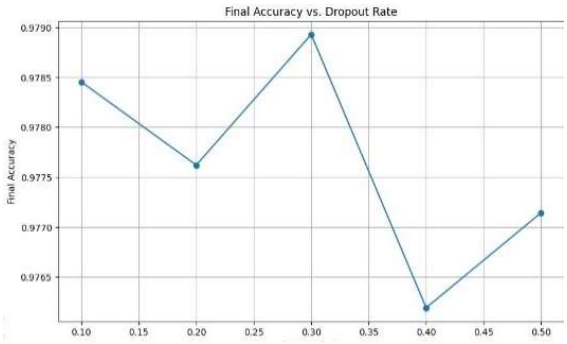

Fig. 7. Line Chart (Final Accuracy vs Dropout Rate).

Cross Tuning

After obtaining the best value to use for the number of epochs and dropout rate separately, the best value of the number of epochs will be tested with multiple different values of dropout rate and vice versa to confirm the chosen values for both modified parameters match one another well.

Model	Dropout rate with 23 epochs	Accuracy	Validation Accuracy	Average Accuracy
K	0.1	0.9944	0.9799	0.98715
L	0.2	0.9926	0.9785	0.98555
M	0.3	0.9875	0.9807	0.98410
N	0.4	0.9821	0.9775	0.97980
O	0.5	0.9730	0.9764	0.97470

Fig. 8. Table of Results for Model trained with different dropout rates with 23 epochs.

This is because certain over- or underfitting problems that might take place can be prevented when modified parameters are considered individually. Fig. 8 shows a significant amount of fluctuation in average accuracy when the dropout rate is changed. This indicates that having the right dropout rate can greatly affect the effectiveness of the model and its results. Based on the average accuracy, it is confirmed that a 0.1 dropout rate is optimal when paired with 23 epochs, proving that the greatest generalization performance is achieved with this configuration and adequately adapting to the training dataset while avoiding overfitting. This optimal model is efficiently trained across a reasonable number of epochs, while preventing high dropout that may hinder learning.

0.1 dropout rate with 14-26 epochs

Model	Number of epochs with 0.1 dropout rate	Accuracy	Validation Accuracy	Average Accuracy
P	14	0.9920	0.9748	0.98340
Q	17	0.9929	0.9783	0.98560
R	20	0.9946	0.9785	0.98655
S	23	0.9937	0.9773	0.98550
T	26	0.9952	0.9780	0.98660

Fig. 9. Table of Results for Model trained with different number of epochs with 0.1 dropout rate.

Fig. 9 shows that the average accuracy is consistent across the 5 different training epochs. All the epochs' accuracy and validation scores are high, showing that the model is effective overall. This is indicative of the robustness and efficiency of the model with a 0.1 dropout rate. However, 20 epochs and 26 epochs both achieve slightly high accuracy, therefore 23 epochs are taken as the optimum solution due to it being the average of them. This is to strike a balance to prevent overfitting, which could happen with more epochs, while ensuring the model has enough training iterations to achieve a good result.

From both results obtained from cross tuning, the second model shows that when a dropout rate of 0.1 is used, the model's performance is unaffected by the number of epochs. This indicates that the number of epochs does not affect the results of the model as significantly as the dropout rate. This remarkable observation has proved that dropout rates serve as a more significant modified parameter than the number of epochs impacting the performance or accuracy of the model. The extensiveness of the model is directly affected by the dropout rate. The model may be unable to retain the training data (which can cause the risk of overfitting if there is excessive training data) if there is a greater dropout rate, which establishes more randomness and regularization. On the other hand, a model can retain more data and complexity when the dropout rate is lower.

All in all, overfitting or underfitting can be successfully regulated by the dropout rate, irrespective of how many numbers of epochs there are. Both tables have confirmed that a 0.1 dropout rate is the most optimal solution when paired with 23 epochs. With this in consideration, the model K will be chosen as it has the highest average accuracy. The Digit Recognizer will be able to recognize the handwritten digit more accurately and possibly faster too.

5. Conclusion

In this paper, it is demonstrated how the performance of neural network models for digit recognition to MNIST datasets can be improved by utilizing a comprehensive combination of methods and methodologies. From experimental results, it is found that the model K uses Adam Optimizer, 4 hidden layers with Dropout Layer, 0.1 learning rate, and 23 epochs get an average accuracy of 0.98715 compared with other results this is the highest accuracy. The average time taken per epoch for model K is 3.3 seconds, which is a very short time, indicating a model that can be trained very fast and has high performance. This is especially crucial as performance speed is highly valued in this technological era. Compared to the other results, the result of model K shows that the performance of neural network models has been improved to recognize MNIST datasets by changing the parameters of the model. Finally, it is possible to implement training with different optimizers such as Adamax and SMORMS3 to further improve the accuracy of the digit recognition (Amananandrai,2023).

References

1. Ngan, J.F., Keong, Y.Q., Raymond, J.M.C., Wong, K.W., Gan J.X. (2023). Digital Classification using Random Forest Classifier. Journal of Applied Technology and Innovation,7(3),1-6. http://jati.sites.apiit.edu.my/files/2023/07/Volume7_Issue3_Paper11_2023.pdf
2. Lead, M.S., Brennan, B.C.C., Gwo, Y. T. & Hui, T.C. (2021). MNIST handwritten digit recognition with different CNN architectures. Journal of Applied Technology and Innovation,5(1),1-4. https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2021/01/17192613/MNIST-Handwritten-Digit-Recognition-with-Different-CNN-Architectures.pdf
3. Ng,B.L. (2017). MNIST Dataset: Digit Recognizer. https://www.kaggle.com/code/ngbolin/mnist-dataset-digit-recognizer/notebook.
4. Daniel.E. (2022). MNIST — Dataset of Handwritten Digits. https://medium.com/mlearning-ai/mnist-dataset-of-handwritten-digits-f8cf28edafe#:~:text=MNIST%20is%20a%20widely%20used,standard%20benchmark%20for%20classification%20tasks.
5. Carolina,B. (2021). Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141#:~:text=A%20Multilayer%20Perceptron%20has%20input,use%20any%20arbitrary%20activation%20function
6. Turing. (2023). Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. https://www.turing.com/kb/mathematical-formulation-of-feed-forward-neural-network
7. Vitaly, B.(2018). Adam — latest trends in deep learning optimization. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
8. Vihar. (2023). Feedforward Neural Networks: A Quick Primer for Deep Learning. https://builtin.com/data-science/feedforward-neural-network-intro
9. Evelyn, M.(2023). What Is Generalization In Machine Learning?. https://magnimindacademy.com/blog/what-is-generalization-in-machine-learning/
10. Rubentak. (2023). Understanding Feed Forward Neural Networks with MNIST Dataset. https://magnimindacademy.com/blog/what-is-generalization-in-machine-learning/
Amananandrai. (2023). 10 famous Machine Learning Optimizers. https://dev.to/amananandrai/10-famous-machine-learning-optimizers-1e22