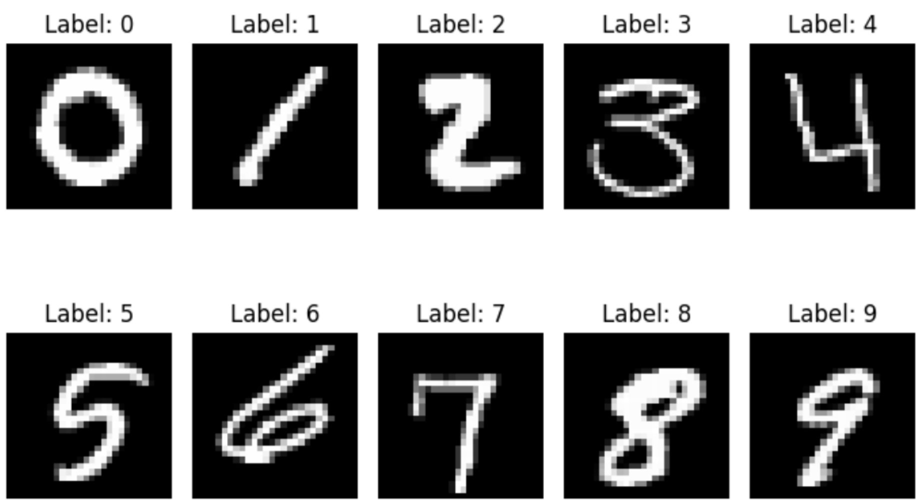
Classifying Digits with Neural Networks

Isha Singh



MSDS 422 – Practical Machine Learning

Irene Tsapara

26 February 2025

MANAGEMENT/RESARCH QUESTION

The goal of this machine learning research was to be able to develop an efficient model that will recognize different and unique handwritten digits. This is useful in the outside world to be able to understand when in work scenario one is not able to understand a specific numeric. The specific research is focused on making digit recognition more accurate using neural networks and their hyperparameters to improve performance. In daily life, this will be useful in scenarios such as bank forms or medical forms, where this model can correctly identify handwritten numbers. This is important and will make the process of understanding handwritten forms easier.

I. OVERVIEW AND METHODOLGY

The following research is based on the Kaggle Competition: Digit Recognizer. The main goal of this study is to classify handwritten digits (0-9) using neural networks. Different model factors will be evaluated by modifying hyperparameters such as the number of layers, neurons per layer, activation functions, optimizers, and batch sizes. By experimenting with different types of hyperparameters, the model will be created as a stronger output for classifying and recognizing digits.

The analysis will also reflect on the following factors: training time, validation accuracy, loss curves, confusion matrices, and ROC curves to compare model performances. Additionally, another main factor in this research will include a benchmark experiment that consists of a 2×2 crossed design to evaluate how the different architectural choices impact classification accuracy. The results will provide insights into the accuracy and precision of the different types of neural networks, and from that, it will be recognized as the most optimal model for handwritten digit recognition.

The dataset used has 42,000 training images and 28,000 test images, where each specific image is 28 x 28 pixels in grayscale with values that range from 0 to 255. The value 0 represents white, and 255 represents black. The training set includes a label column that indicates the digits 0-9; however, the test dataset incorporates only test set image data. Some of the observations that were noticed during the procedure were that no missing values were found within the dataset, each of the digits from 0-9 is explicitly there, and there were various different samples of handwriting styles.

To have the data become ready, data preprocessing and feature engineering were important. MinMaxScaler was used in order to normalize pixel values between 0 and 1. Furthermore, a few data augmentation techniques that were used are rotation, shifting, and zooming to enhance and create some sort of generalization across different and unique handwriting styles. However, the data augmentation was only used to gain more perspectives on the data through visualization.

Subsequently, EDA (Exploratory Data Analysis) was conducted in order to understand the samples visually and get a clear representation of different types of handwriting. The average and variance, samples, as well as pixel intensity distributions were examined.

The digit recognizer uses neural networks, and for that, several scenarios of architecture were tested to understand the best-performing model. A 2×2 crossed design, as required by instructions (4 models), was used to compare the number of layers vs. neurons per layer. This approach helped understand which type of model was the most effective for the neural network structure.

Afterward, some hyperparameters that were applied to create high accuracy and reduce overfitting were batch size, learning rate, and number of epochs. The models were tested and evaluated with several factors: validation accuracy, loss curves, confusion matrices, and ROC/AUC curves. Overall, from the entire research, the results showcased that the first model out of 4, with 2 hidden layers and 10 neurons per layer, had the best accuracy score of 0.903333.

The main goal of this research was to create a neural network with hyperparameter tuning to be able to recognize handwritten digits. By hyperparameter tuning and experimenting with different models, the research provides us with a lesson on choosing which parameters are best for the specific neural network model.

II. DETAILED STEPS AND CODE EXPLANATION

After adding hyperparameters, the accuracy changed, with Model 1's final revised accuracy becoming 0.9033. The loss was 0.3394, which meant there was a possibility of less overfitting. Model 2 followed similarly to Model 1, but its validation accuracy was lower. The extra 10 neurons added in comparison to the first model did not provide a strong boost, showing that simply increasing neurons does not necessarily improve learning.

Similarly, the last two models had a lot of issues. They were trained using the same template as Model 1, but due to having more layers, the validation accuracy was negatively affected:

- Model 3 - Test Accuracy: 0.4336
- Model 4 - Test Accuracy: 0.2295

Due to the extra layers, these models were unable to learn properly and ended up with low accuracy scores. This process helped create a starting point for the next step: hyperparameter tuning, which aimed to improve model accuracy. Throughout the procedure, hyperparameter tuning was applied by adjusting the learning rate to 0.0005 and switching to SGD (Stochastic Gradient Descent) for the third model. The learning rate for the fourth model was 0.0005, but the Adam optimizer remained.

The Adaptive Moment Estimation optimizer is believed to be better suited for deeper models and therefore it was decided to keep it for the 4th model. After adjusting the hyperparameters, the top two models in the competition remained as Model 1 and 2. Models 3 and 4 were not ideal for their recognition due to the low accuracy they had achieved.

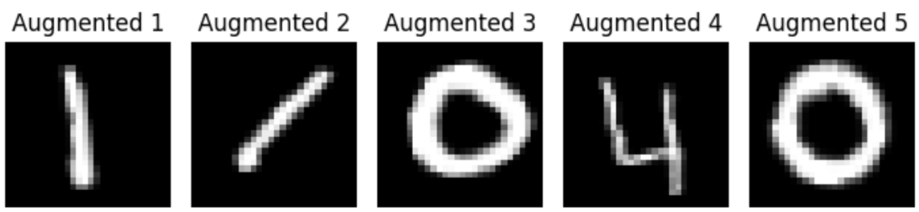
Some of the most misclassified numbers were 5 in Model 1 and Model 2. This may have happened because 5 can be misinterpreted as either 3 or 6. For Model 3, the number 3 was misclassified frequently, possibly due to its similarity to the numbers 8 or 5. Lastly, Model 4 misclassified 9 the most, possibly because it looked similar to 4 or 7.

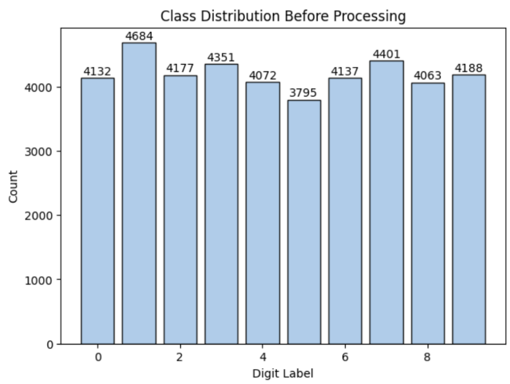
Overall, each model had different misclassification patterns because each architecture processed features differently. Simpler models generalized better, while deeper models struggled to differentiate certain digits, especially when not tuned correctly.

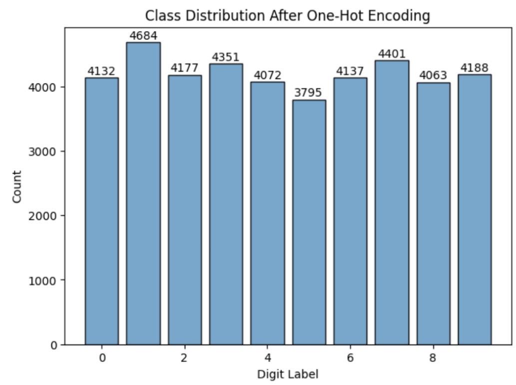
III. VISUAL AND TABULAR INFORMATION

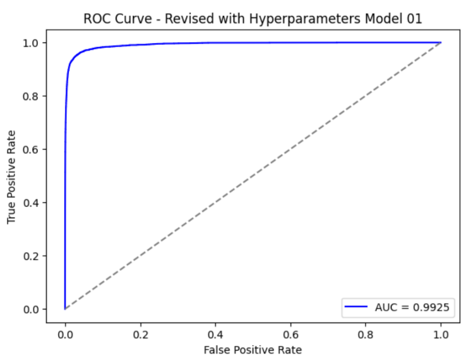
Model Performance Summary:

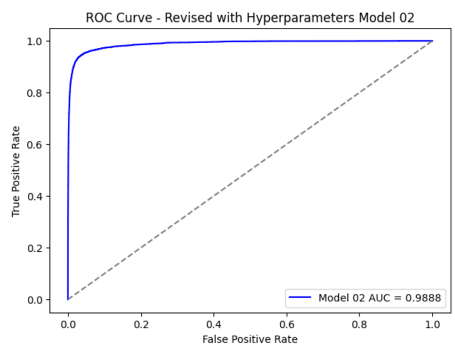
	Model	Test Accuracy	AUC Score	Training Time (s)	Loss
0	Model 01 (2L–10N)	0.903333	0.992504	191.998476	0.339371
1	Model 02 (2L–20N)	0.883571	0.988775	211.467213	0.541799
2	Model 03 (5L–10N)	0.433571	0.827469	84.689987	1.678385
3	Model 04 (5L–20N)	0.229524	0.576648	160.660718	1660.237915

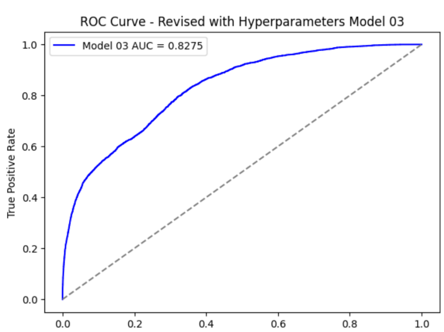

Augmented 1
Augmented 2
Augmented 3
Augmented 4
Augmented 5

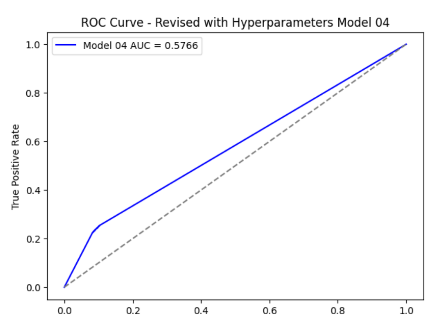

Class Distribution Before Processing
4132
4684
4177
4351
4072
3795
4137
4401
4063
4188
Count
Digit Label


Class Distribution After One-Hot Encoding
4132
4684
4177
4351
4072
3795
4137
4401
4063
4188
Count
Digit Label


ROC Curve - Revised with Hyperparameters Model 01
AUC = 0.9925
True Positive Rate
False Positive Rate


ROC Curve - Revised with Hyperparameters Model 02
Model 02 AUC = 0.9888
True Positive Rate
False Positive Rate


ROC Curve - Revised with Hyperparameters Model 03
Model 03 AUC = 0.8275
True Positive Rate
False Positive Rate


ROC Curve - Revised with Hyperparameters Model 04
Model 04 AUC = 0.5766
True Positive Rate
False Positive Rate

Since there were a lot of EDA visualizations created for the research, I selected the four most important that I believed best showcase key insights and effectively highlight the structure, impact, and significance. The visualizations include the model performance summary (showing accuracy and loss), class distribution before and after one-hot encoding, and ROC and AUC curves.

The first visualization shows the model performance summary, which compares the four models that are revised with hyperparameters. The table consists of test accuracy, AUC score, training time, and loss. Model 1 performed the best with a 90.33% accuracy and a loss showing at 0.3394. Model 4 performed the worst with 33.95% and unfortunately with a very high loss of 1660.24. The results displayed that the simpler the models are, the better the performance.

The second visualization focuses on data augmentation, where it used the following factors: rotation, width shift, and height shift (Scikit-learn). This produced different variations for us to be able to understand how a digit looks when it is rotated.

The class distribution graphs show the outputs before and after one-hot encoding. One-hot encoding was used in order to convert categorical labels. In specific, to_categorical was used. The left graph represents the count of each digit from 0-9, and the right graph has a confirmation after using the method to_categorical and confirming nothing has changed after the conversion to binary numbers.

The last 4 visualizations represent the revised models' ROC curves. Model 1 and Model 2 performed the best with AUC scores of 0.99 and 0.98. Nevertheless, as mentioned in other sections, Model 3 and Model 4 did not perform well. Their AUC scores were 0.82 and 0.57, though Model 3 was subsequently better than Model 4.

IV. CONCLUSION

This research was a great way to understand how different neural network models handle handwriting recognition. The best-performing model was Model 01, which had 2 layers and 10 neurons per layer, achieving an accuracy of 90.93% after tuning the hyperparameters. The second revised model with hyperparameters had a slightly lower accuracy of 0.888810, which showed that increasing the number of neurons did not necessarily improve performance.

The third revised model had 5 layers and 10 neurons per layer, and the fourth model had 5 layers and 20 neurons per layer. Both had the lowest scores of about 0.324762 (Model 03) and 0.220238 (Model 04). These two models took the longest to train as they had issues with classification, and due to overfitting, they slowed down the procedure. This overall shows that adding more layers can impact the model's performance.

The most misclassified digits varied across each model, showing that with certain numbers, it was difficult to distinguish them correctly, likely due to similar pixel structures or overlapping features. For the future, it might be a good idea to focus on having better hyperparameters, experimenting with activation functions, or incorporating generalization functions such as dropouts. To conclude, this research showed the importance of learning how to pick hyperparameters wisely, as selecting too many or the wrong ones can cause issues with performance and efficiency.

V. REFERENCES

Chatgpt. Accessed February 27, 2025. https://chatgpt.com/.

“Classification: Roc and AUC | Machine Learning | Google for Developers.” Google. Accessed February 28, 2025. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

“Classification_report.” scikit. Accessed March 28, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

“Confusion_matrix.” scikit. Accessed March 1, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

“Digit Recognizer.” Kaggle. Accessed February 26, 2025. https://www.kaggle.com/competitions/digit-recognizer.

GeeksforGeeks. “Implementing Neural Networks Using Tensorflow.” GeeksforGeeks, February 28, 2022. https://www.geeksforgeeks.org/implementing-neural-networks-using-tensorflow/.

“MinMaxScaler.” scikit. Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

Radhakrishnan, Pranoy. “What Are Hyperparameters ? And How to Tune the Hyperparameters in a Deep Neural Network?” Medium, October 18, 2017. https://medium.com/towards-data-science/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a.

"Tensorflow 2 Quickstart for Beginners : Tensorflow Core." TensorFlow. Accessed February 27, 2025. https://www.tensorflow.org/tutorials/quickstart/beginner.

"Tf.Keras.Preprocessing.Image.Imagedatagenerator : Tensorflow V2.16.1." TensorFlow. Accessed February 27, 2025. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.

VI. APPENDIX

LINK TO PANAPTO

- https://northwestern.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=482c157d-1369-46b4-8932-b29400136021&start=0

CHATGBT SCREENSHOTS

can you give a pusedo code of NN simple temple

Sure! Here's a simple pseudocode template for a basic neural network (NN) with one hidden layer:

yaml

```
# Initialize Neural Network
Initialize weights and biases

# Forward Propagation
For each input sample:
    Compute hidden layer activations:
        Hidden_Output = Activation_Function(Input * Weights_Input_Hidden + Bias_Hidde
    Compute output layer activations:
        Output = Activation_Function(Hidden_Output * Weights_Hidden_Output + Bias_Out

# Compute Loss
Compute loss between predicted output and actual target

# Backpropagation
Compute gradients of loss w.r.t weights and biases
Update weights and biases using gradient descent
```

what parameters are good for neural network

The choice of parameters for a neural network depends on the problem you're solving (e.g., classification, regression, image recognition). However, here are some general guidelines for key parameters:

1. Network Architecture

- **Number of Hidden Layers:**
  - For simple problems, 1-2 hidden layers are enough.
  - For complex tasks (e.g., image recognition, NLP), deep networks (multiple layers) work better.
- **Number of Neurons per Layer:**
  - Start with **2-3 times** the number of input features.
  - Experiment with increasing or decreasing the neurons.

2. Activation Functions

- **Hidden Layers:** Use **ReLU** (most common) or **Leaky ReLU** to avoid vanishing gradients.
- **Output Layer:**
  - **Sigmoid** for binary classification.
  - **Softmax** for multi-class classification.
  - **Linear** for regression problems.

3. Learning Rate (α)

- **Typical range:** `0.001` to `0.01`
- Too high → Fast convergence but may overshoot.
- Too low → Slow convergence or stuck in local minima.
- **Adaptive learning rates:** Use Adam optimizer (`learning_rate ≈ 0.001`).

4. Batch Size

- **Small batch (e.g., 32-64)** → More stable updates, better generalization.
- **Large batch (e.g., 256-1024)** → Faster training but can overfit.

7. Regularization

- **L2 Regularization (Weight Decay):** Helps prevent overfitting (`λ ≈ 0.0001` to `0.01`).
- **Dropout:** Randomly turns off neurons during training (`dropout rate ≈ 0.2 - 0.5`).

8. Number of Epochs

- **Start with 50-100 epochs** and use early stopping to prevent overfitting.

9. Loss Functions

- **Binary Classification:** `Binary Cross-Entropy`
- **Multi-class Classification:** `Categorical Cross-Entropy`
- **Regression:** `Mean Squared Error (MSE)`

What deos compile do?

In the context of neural networks (especially in **Keras** and **TensorFlow**), `compile()` is used to configure the model before training. It defines key settings like the optimizer, loss function, and metrics.

What `compile()` Does?

1. **Sets the Optimizer** → Defines how the model updates weights during training.
2. **Specifies the Loss Function** → Measures the difference between predictions and actual values.
3. **Defines Evaluation Metrics** → Helps monitor model performance.

Example

python Copy Edit

```python
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Breakdown of Parameters

- `optimizer`: Algorithm that updates weights to minimize loss (`'adam'`, `'sgd'`, `'rmsprop'`, etc.).
- `loss`: Function to compute the model's error (e.g., `'mse'` for regression, `'binary_crossentropy'` for binary classification).
- `metrics`: Measures performance (e.g., `'accuracy'` for classification, `'mae'` for regression).

Why is `compile()` Necessary?

- It prepares the model for training by linking it with a loss function and optimizer.
- Without `compile()`, the model won't know **how** to learn.

How much epochs

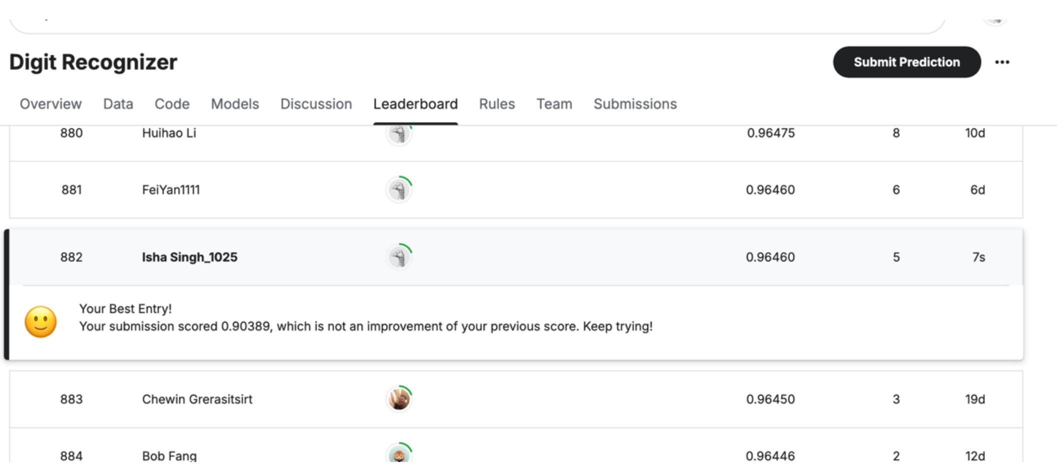
The **number of epochs** depends on the complexity of your dataset and the problem you're solving. Here are some general guidelines:

1. Start with a Baseline

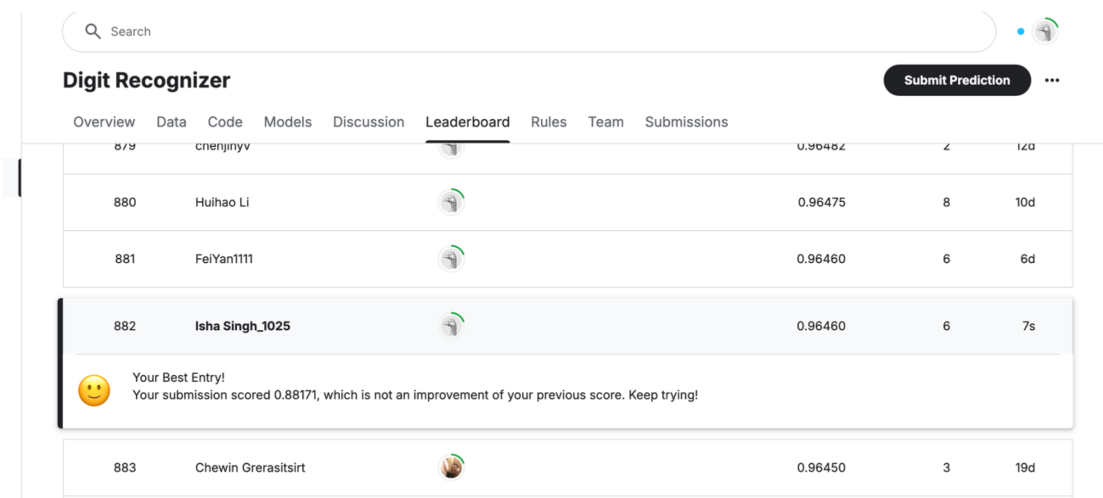
- **Small datasets:** `50 - 100` epochs
- **Medium datasets:** `100 - 300` epochs
- **Large datasets:** `300+` epochs

KAGGLE SCORES

MODEL 01



MODEL 02



MODEL 03

881	FeiYan1111	0.96460	6	6d
882	**Isha Singh_1025**	0.96460	7	4s

Your Best Entry!
Your submission scored 0.42314, which is not an improvement of your previous score. Keep trying!

883	Chewin Grerasitsirt	0.96450	3	19d
884	Bob Fang	0.96446	2	12d
885	gxingyu111	0.96442	1	1mo
886	Zixuan Zhang cc	0.96417	15	7d

MODEL 04

881	FeiYan1111	0.96460	6	6d
882	**Isha Singh_1025**	0.96460	8	7s

Your Best Entry!
Your submission scored 0.42314, which is not an improvement of your previous score. Keep trying!

883	Chewin Grerasitsirt	0.96450	3	19d
884	Bob Fang	0.96446	2	12d
885	gxingyu111	0.96442	1	1mo
886	Zixuan Zhang cc	0.96417	15	7d