Fine-tuning a CNN for the MNIST Dataset

Unit 4 Assignment: Deep Learning

Juliana Noronha

University of Maryland, Global Campus

DATA 655: Deep Learning and Neural Networks

Dr. Jeremy Bolton

February 11, 2025

Introduction

Objective

The objective of this analysis is to evaluate the impact of varying convolutional neural network (CNN) architectures on the classification accuracy of the MNIST handwritten digit dataset. Specifically, this paper investigates how changes in the number of convolutional layers, kernel sizes, stride lengths, pooling strategies, and the inclusion of batch normalization and dropout layers influence model performance.

Problem Domain

The MNIST (Modified National Institute of Standards and Technology) dataset is a smaller subset of the larger National Institute of Standards and Technology (NIST) database and has been a cornerstone in evaluating classification models for optical character recognition (OCR). It has been referenced in over 7,500 papers (Papers with Code - MNIST Dataset, n.d.) and is often referred to as the "Hello World" dataset of machine learning (Tutorials, n.d.). OCR is just a small subset of the larger computer vision problem in computer science, for which there are countless potential applications and use-cases. Accurate OCR can replace the need for manual handwritten verification or transcription. More generally, image classification or computer vision can be used for technologies like self-driving cars and drones, accessibility augmentations, crop monitoring, and medical diagnostics.

Method Rationale

Convolutional neural networks (CNNs) were developed specifically to classify image data. Yann LeCun and his team are credited with the invention of the convolutional neural network, with their groundbreaking paper "Handwritten Digit Recognition with a Back-Propagation Network" (LeCun, 1989). The concept of a convolutional layer was developed

specifically for the task of handwritten digits, but has since been applied to general image classification or computer vision tasks. CNNs use filters to scan the two-dimensional space of an image and identify the presence or absence of key shapes. Unlike traditional machine learning algorithms that rely on handcrafted features, CNNs learn hierarchical representations that improve classification performance. These characteristics mean that a CNN would be well-suited for most image datasets.

Analysis

Data

The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0-9), each of size 28x28 pixels (mnist, n.d.). 60,000 images are designated as training data and the remaining 10,000 images are designated as testing data. The digits were written by employees of the United States Census Bureau and have been size-normalized and centered in a fixed-size image (Papers with Code, n.d.). When imported directly from the tensorflow data repository, the training feature data comes as a three-dimensional 60,000 length tensor of 28x28 arrays, containing values ranging from 0 to 255, where 0 is black and 255 is white. The label data is simply a 60,000 length tensor, containing values ranging from 0 - 9.

Exploratory Analysis

First, the training labels are plotted as a barchart to evaluate whether there are any glaring class imbalances (Figure 1). Ones are the most prevalent, at 6742 occurrences, while fives are the least prevalent, at 5421 occurrences. However, the distribution is relatively uniform, with each number label representing about 10% of the dataset.

Next, the images themselves were inspected. Figure 2 displays the first 10 instances of each of the labels in the dataset. As one might suspect, there is great variation between each

hand-written digit. For example, two of the ones have serifs, while the rest are straight lines (although, some of the straight lines are indeed at an angle). One of the sevens has a line cutting through it. Additionally, digits employ different degrees of curvature. For example, some twos have a loop connecting the bottom portion to the top while others have a sharp angle. Two of the nines have rounded bottoms, while the rest are a circle connected to a straight line. So far, these are simply stylistic differences that may or may not pose issues for the CNN. Finally, there are extremely ambiguous digits, such as the 10th one, which a human might reasonably interpret as a two, or the 7th five, which is almost a straight line and might reasonably be interpreted as a one with a sloppy lower serif.

Preprocessing

The image data has already been centered, normalized and converted to gray-scale. There are no striking class imbalances to handle. The only pre-processing step required here is to scale the pixel values from 0 - 255 to 0 - 1, so that the larger values do not have undue influence to the model.

Algorithm Intuition

Convolutional Neural Networks (CNNs) operate on the principle of spatially local feature extraction, leveraging convolutional layers to detect patterns in image data. The convolution operation, defined as , capturing spatial relationships between pixels. The key input parameters influencing CNN performance include kernel size, which determines the receptive field of each neuron; stride, which controls how much the filter shifts at each step; and pooling type and size, which reduce spatial dimensions while retaining essential features. Additionally, batch normalization stabilizes training by normalizing activations, and dropout helps prevent overfitting by randomly deactivating neurons during training.

Model Fitting

For model tuning, the following hyperparameters were tested: number of convolutional layers, kernel (or filter) size, stride length, pooling type, inclusion of a batch normalization layer and the inclusion of a dropout layer.

To test different combinations of these hyperparameters, a function was crafted that took model configurations as input and returned model results as output. For all model configurations, adam was used as the optimizer and accuracy was used as the evaluation metric. Sparse categorical cross-entropy was used as the loss function, since this is a multi-class classification problem and this loss function allows bypasses the need to one-hot encode the label (tf.keras.losses.SparseCategoricalCrossentropy, 2024).

All models tested had the following basic foundation: a 2D convolutional layer with 32 filters and a ReLU activation function, followed by an optional batch normalization layer, followed by a max or average pooling step. Then, a similar set of layers may be repeated 1-2 more times (convolution, optional batch normalization, pooling), but using 64 filters in the convolutional layer instead. The number of filters were increased from the first layer in order to capture more complex shapes. Finally, the architecture ends with a flatten layer, a dense layer with 128 nodes and a ReLU activation function, an optional dropout layer with 50% dropout, and finally, the output layer containing 10 nodes with softmax activations. The flatten layer is required to move from two-dimensional space (image) to one-dimensional space (the labels spanning from 0 - 9). A dense layer allows another opportunity to introduce non-linearity and the dropout layer, which may or may not be enabled during testing, potentially helps to prevent overfitting. Finally, the output layer must have 10 nodes, one for each class. The softmax

activation function constrains the output between 0 - 1, which ultimately represents the probability that the image should be classified as that class.

In addition to the number, type and order of layers, kernel size was also tested.

Results

Output

In total, four experiments were run on the data, with the following accuracy results:

Exp. #	# of Conv. Layers	Kernel size	Pooling type	Include Batch Norm Layer?	Include Dropout Layer?	Accuracy
1	2	3x3	max	No	No	99.1%
2	3	3x3	max	Yes	No	98.75%
3	2	5x5	average	No	Yes	99.11%
4	3	3x3	average	Yes	Yes	99.29%

All models performed extremely well, but the model created in experiment 4 had the best overall accuracy at 99.29%. Interestingly, experiment 2 seems to indicate that the inclusion of batch normalization layers without a dropout layer results in worse performance. However, the full search space was not exploited, so it is indeed difficult to pinpoint a cause for the slightly lower performance, or if the weights converged slightly differently by chance.

Model Properties

The model summary can be found in Figure 3. In total, the model had 13 layers (not including the input layer). There were three total convolutional layers, three batch normalization layers and three average pooling layers. Figure 4 offers a look into which filters were used in the first convolutional layer and Figures 5 and 6 display how those filters extracted features from an image of a seven and an image of a zero, respectively. These two numbers are very different --

sevens tend to be sharp and angular while zeros are a continuous circle. Some filters do an excellent job of outlining both shapes, despite how different they are (outlined by green on Figure 5) and others did very poorly (outlined by red). One can also understand intuitively how the filters are identifying features. For example, the filter in the bottom-right corner in Figure 4 is very bottom-heavy. This results in the filter highlighting the bottoms of outlines, as can be seen in Figures 5 and 6.

Evaluation

Running a classification report on the model, the model never falls below 98% for precision, recall, f1-score or accuracy (Figure 7). The only two appearances of 98% are for four's recall, and nine's precision. In other words, the model is slightly more likely to assign the label of "nine" to a digit that is not actually a nine, and it is less likely to correctly label all instances of four. Reviewing the confusion matrix (Figure 8), we see that these two lower scores are actually related to one another -- this model tends to over-classify fours as nines. It is also interesting that nines are never misclassified as fours. There are 64 misclassifications in total, and 17 of those (27%) are exclusively from nine being classified as four.

Figure 9 displays the first 25 misclassified images. Eight of these are fours that have been misclassified as nine. Some of these misclassifications are potentially forgivable, such as the two fours in the bottom row, but it seems the model has failed to pick up that fours can sometimes have an unconnected upper portion, whereas that should mostly never happen with a nine. Interestingly, upon inspecting the probabilities, the probability for four is always second highest to nine, and both probabilities are orders of magnitude away from the other numbers.

Conclusion

Summary

This analysis evaluated the impact of various convolutional neural network (CNN) architectures on the classification accuracy of the MNIST handwritten digit dataset. The results demonstrate that all tested CNN architectures performed exceptionally well, with accuracy exceeding 98%. The highest accuracy of 99.29% was achieved in Experiment 4, which incorporated three convolutional layers, batch normalization, dropout, and average pooling. The findings suggest that incorporating dropout alongside batch normalization may enhance performance, while model configuration decisions such as kernel size and pooling type can impact feature extraction quality. Finally, analysis of filter outputs and misclassified images provides insights into how CNNs learn to differentiate between handwritten digits, particularly in challenging cases like distinguishing fours from nines.

Limitations

Despite achieving high accuracy, this study has a few limitations.The evaluation focused on a limited set of hyperparameters and architectural choices, meaning that additional configurations might yield further performance improvements. The reliance on accuracy as the primary metric also does not fully capture performance nuances, such as the model's tendency to misclassify certain digits (e.g., fours as nines).

Improvement Areas

Future work can address these limitations in several ways. A more comprehensive hyperparameter search using techniques like Bayesian optimization or grid search could refine

architecture choices and further optimize performance. Additionally, incorporating other evaluation metrics could provide a more holistic understanding of the model's reliability.

References

Biswas, A. (2025, January 17). *The History of Convolutional Neural Networks for Image Classification (1989- today).* Towards Data Science. https://towardsdatascience.com/the-history-of-convolutional-neural-networks-for-image-classification-1989-today-5ea8a5c5fe20/

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1989). Handwritten Digit Recognition with a Back-Propagation Network. *Neural Information Processing Systems*, 2, 396–404. http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf

mnist. (n.d.). TensorFlow. https://www.tensorflow.org/datasets/catalog/mnist

Papers with Code - MNIST Dataset. (n.d.). https://paperswithcode.com/dataset/mnist

tf.keras.losses.SparseCategoricalCrossentropy. (2024, June). tensorflow.org. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

Tutorials. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials

Appendix

Figure 1: Barchart of Training Labels

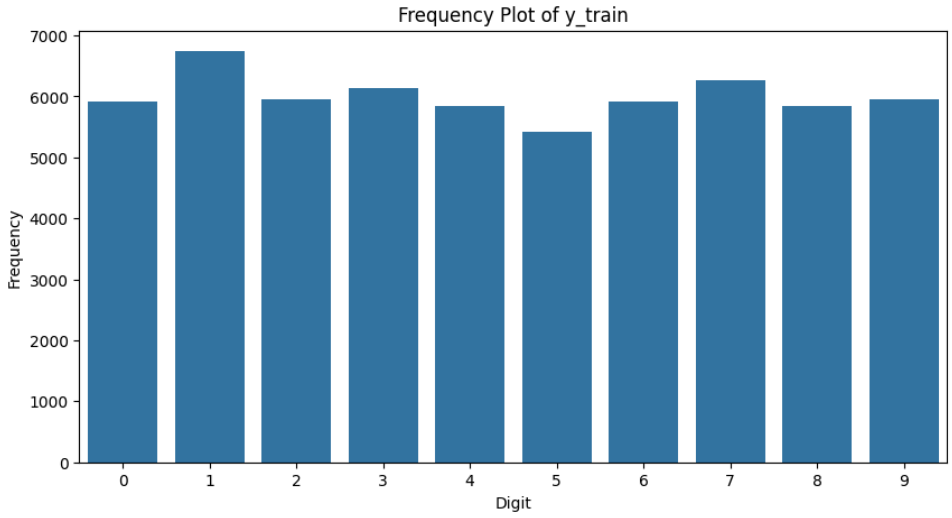

Figure 2: The first 10 instances of each label

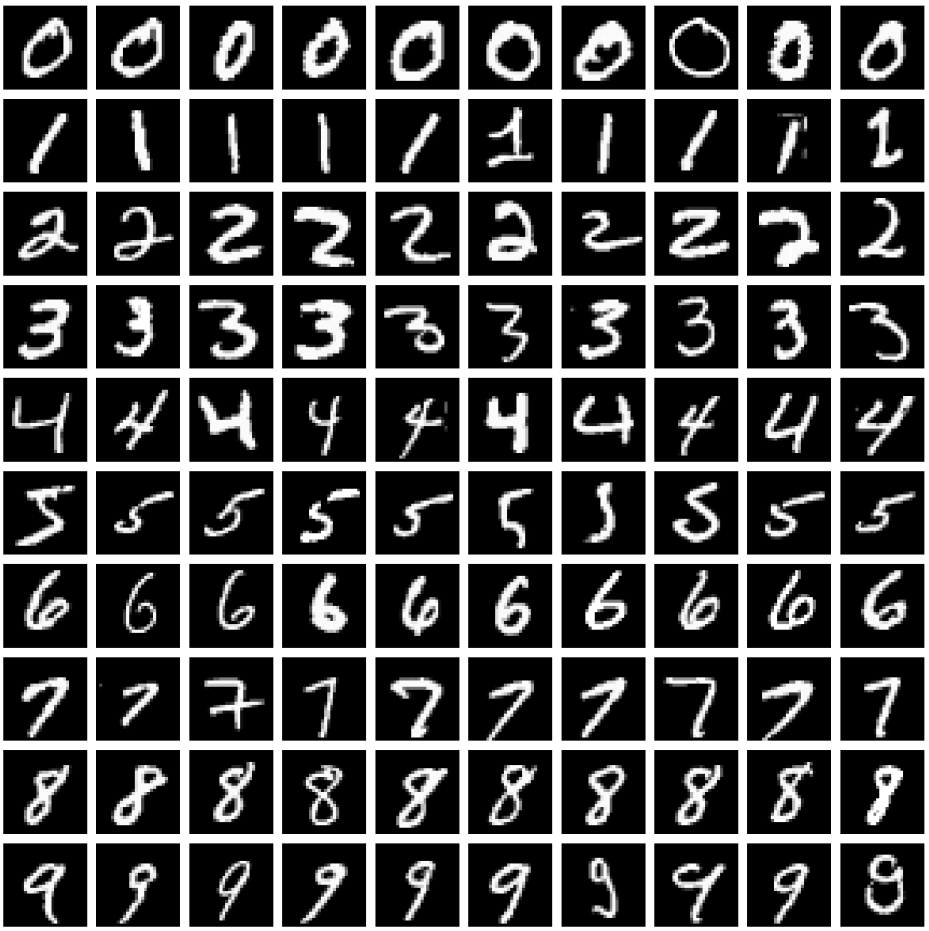

Figure 3: Model Summary

```
best_model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization_6 (BatchNormalization)	(None, 26, 26, 32)	128
average_pooling2d_6 (AveragePooling2D)	(None, 13, 13, 32)	0
conv2d_7 (Conv2D)	(None, 13, 13, 64)	18,496
batch_normalization_7 (BatchNormalization)	(None, 13, 13, 64)	256
average_pooling2d_7 (AveragePooling2D)	(None, 6, 6, 64)	0
conv2d_8 (Conv2D)	(None, 6, 6, 64)	36,928
batch_normalization_8 (BatchNormalization)	(None, 6, 6, 64)	256
average_pooling2d_8 (AveragePooling2D)	(None, 3, 3, 64)	0
flatten_2 (Flatten)	(None, 576)	0
dense_4 (Dense)	(None, 128)	73,856
dropout_2 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1,290

Total params: 393,952 (1.50 MB)
Trainable params: 131,210 (512.54 KB)
Non-trainable params: 320 (1.25 KB)
Optimizer params: 262,422 (1.00 MB)

Figure 4: Learned Filters -- First Convolutional Layer

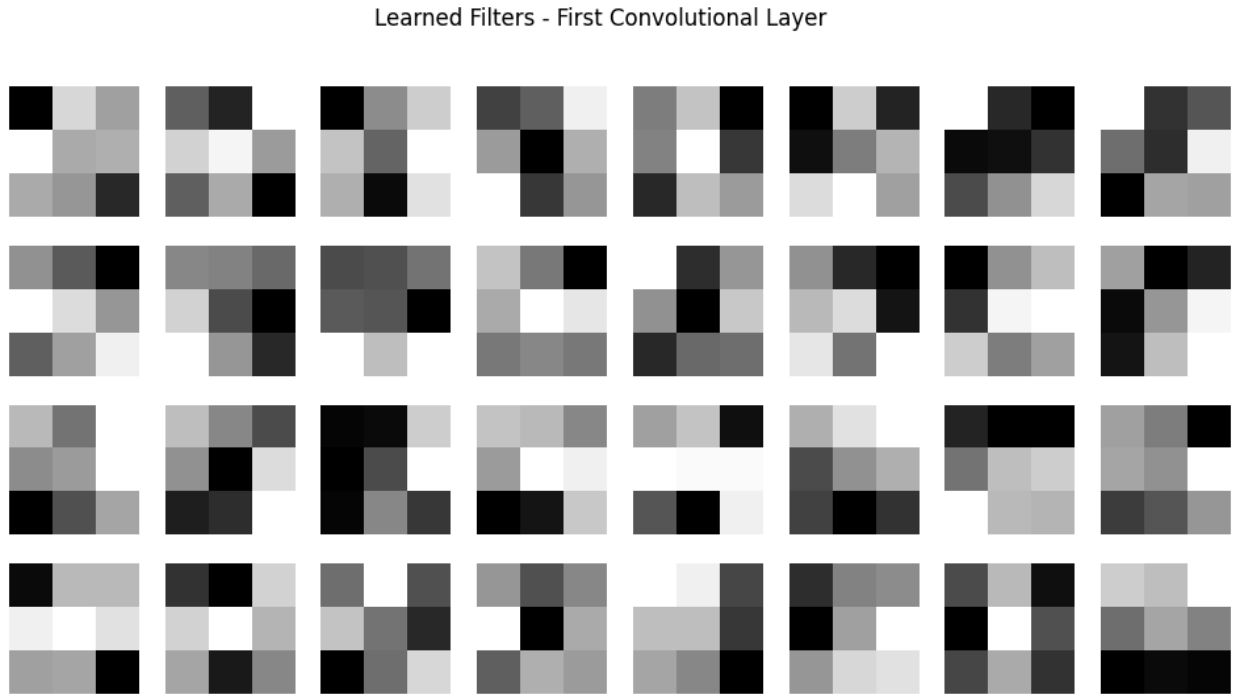

Figure 5: Feature Maps for a seven

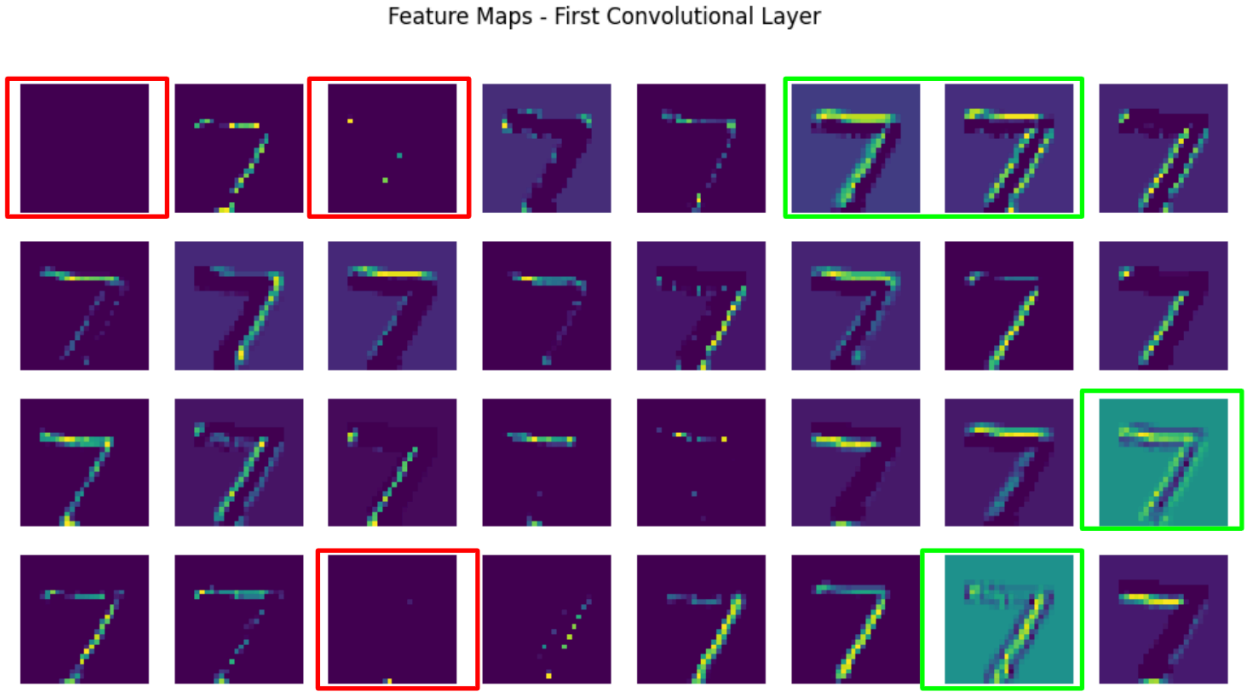

Figure 6: Feature Maps for a zero

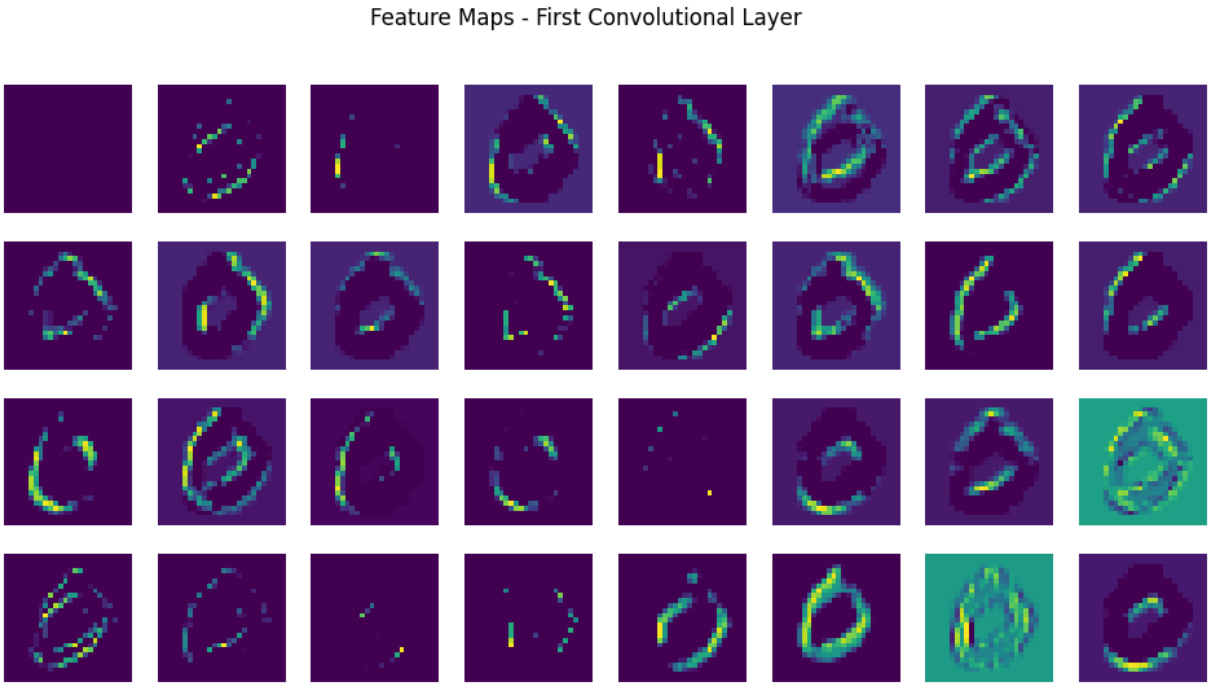

Figure 7: Classification Report

Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	980
1	1.00	1.00	1.00	1135
2	0.99	1.00	1.00	1032
3	0.99	0.99	0.99	1010
4	1.00	0.98	0.99	982
5	0.99	0.99	0.99	892
6	1.00	0.99	1.00	958
7	1.00	0.99	0.99	1028
8	1.00	0.99	0.99	974
9	0.98	1.00	0.99	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

Figure 8: Confusion Matrix

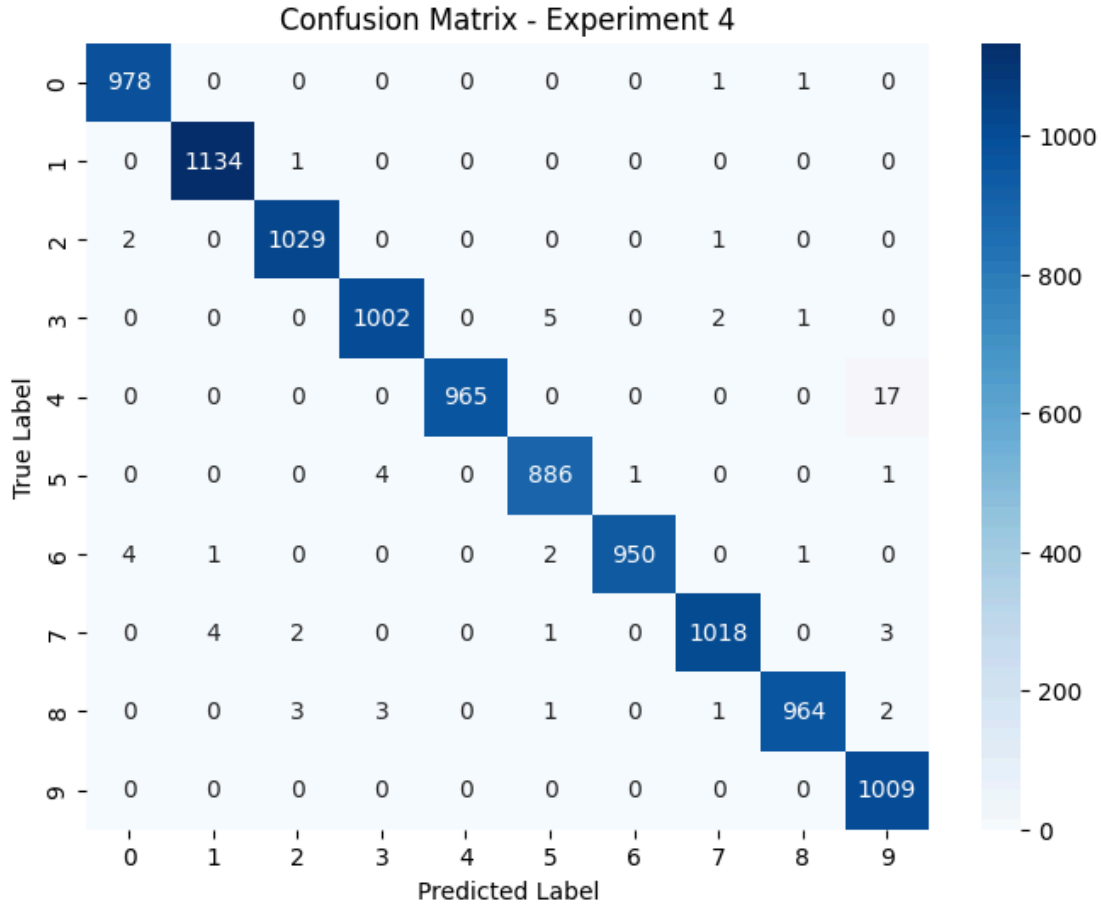

Figure 9: Misclassified Image Examples