Convolutional Neural Networks and MNIST

Tre' R. Jeter

October 12, 2023

Abstract

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision and classification tasks. These deep learning models are specifically designed to automatically learn and extract features from images or structured data with grid-like patterns. CNNs have demonstrated remarkable success in various domains, from image recognition to medical diagnosis and natural language processing. Their architecture, characterized by convolutional layers, pooling layers, and fully connected layers, allows them to capture hierarchical patterns and spatial dependencies, making them well-suited for tasks such as object detection, image classification, and even more complex problems like autonomous driving.

1 Background

The MNIST dataset is a well-known benchmark dataset in ML. It consists of a collection of 28×28 pixel grayscale images of handwritten digits, ranging from 0 to 9. MNIST was created as a standard reference dataset for testing and evaluating various ML algorithms, particularly for digit recognition tasks. With its 60,000 training samples and 10,000 testing samples, MNIST has played a crucial role in the development and evaluation of numerous image classification algorithms and has become a fundamental resource for researchers in the field.

2 Preliminary Experiments

2.1 Setup

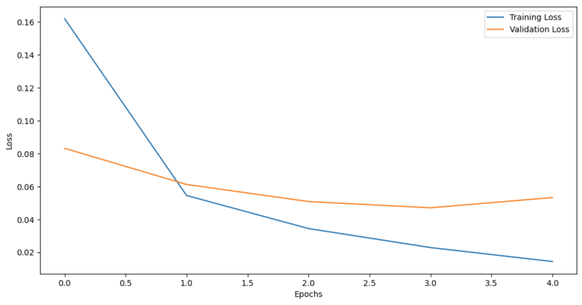
MNIST is already known to be split into a training set and a testing set. In a preliminary experiment, the training set is further split to create a validation set. The training set undergoes an 80/20 split meaning 80% of the original training set remains for training, and the other 20% makesup the new validation set. Next each set of data (training, validation, and testing) is normalized by dividing each pixel by 255. This places each pixel within a range of 0 to 1.

The Keras framework is used to construct a basic CNN model. The first layer is a convolutional layer with 32 filters with size 3×3 and a ReLU activation function. The input size is the dimensions of an MNIST image which is (28×28×1). The second layer is a max pooling layer with a pool size of 2×2. The third layer is a flattening layer to ensure the output is for a classification task (reduce dimensions). The fourth layer is a hidden layer with 128 neurons with a ReLU activation function. The final layer is the output layer with 10 neurons for 10 possible classification outputs and a softmax function. The model is compiled with the Adam optimizer with the *sparse_categorical_crossentropy* loss function.

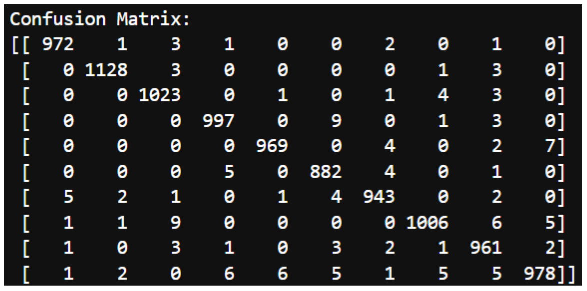
2.2 Results

The basic CNN model was trained for 5 epochs. The training accuracy reached ≈99.5% while the validation reached ≈98.5%. Figure 1a displays the loss for training and validation over the 5 epochs. Notice that the validation loss seems to be trending upward. This shows that the model, given more epochs, would probably overfit. However, the testing accuracy reached ≈98.6%. Figure 1b shows the testing sets confusion matrix.

I ran this experiment a second time with the same architecture and model to present the confusion matrices of all three sets of data. Figure 2 depicts the confusion matrices of the training, validation, and testing set. Each set in this second run reached ≈99.4%, 98.7%, and 98.6%, respectively. The resulting loss plot is shown in Figure 2d. It is noted that because these two preliminary experiments only ran with 5 epochs, an early stopping criteria was not defined.



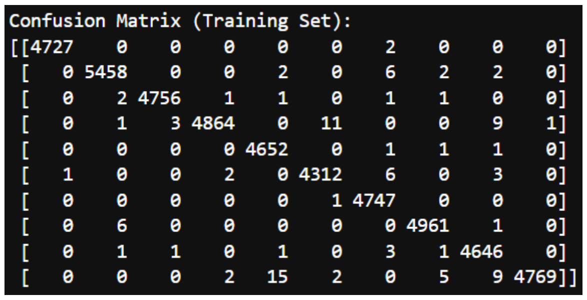
(a) Loss vs. Epochs.



```
Confusion Matrix:
[[ 972    1    3    1    0    0    2    0    1    0]
 [   0 1128    3    0    0    0    0    1    3    0]
 [   0    0 1023    0    1    0    1    4    3    0]
 [   0    0    0  997    0    9    0    1    3    0]
 [   0    0    0    0  969    0    4    0    2    7]
 [   0    0    0    5    0  882    4    0    1    0]
 [   5    2    1    0    1    4  943    0    2    0]
 [   1    1    9    0    0    0    0 1006    6    5]
 [   1    0    3    1    0    3    2    1  961    2]
 [   1    2    0    6    6    5    1    5    5  978]]
```

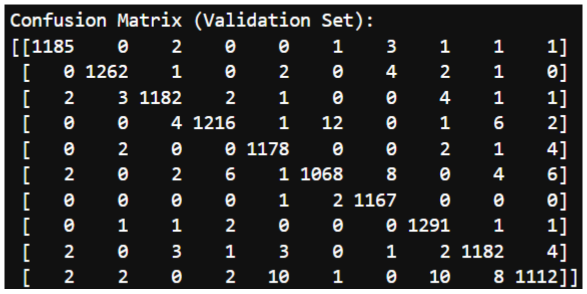
(b) Confusion Matrix on Test Set.

Figure 1: Preliminary Results with Basic CNN.



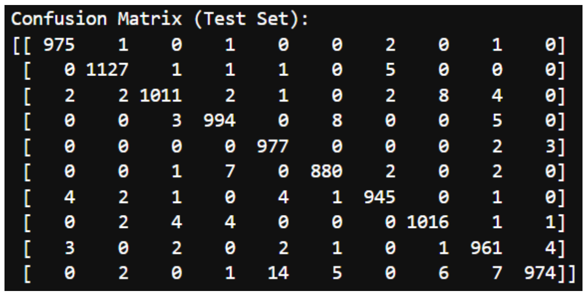
```
Confusion Matrix (Training Set):
[[4727    0    0    0    0    0    2    0    0    0]
 [   0 5458    0    0    2    0    6    2    2    0]
 [   0    2 4756    1    1    0    1    1    0    0]
 [   0    1    3 4864    0   11    0    0    9    1]
 [   0    0    0    0 4652    0    1    1    1    0]
 [   1    0    0    2    0 4312    6    0    3    0]
 [   0    0    0    0    0    1 4747    0    0    0]
 [   0    6    0    0    0    0    0 4961    1    0]
 [   0    1    1    0    1    0    3    1 4646    0]
 [   0    0    0    2   15    2    0    5    9 4769]]
```

(a) Confusion Matrix on Training Set.



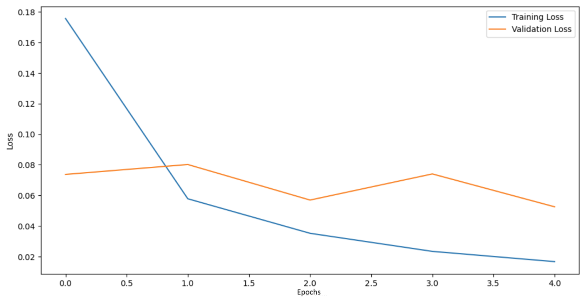
```
Confusion Matrix (Validation Set):
[[1185    0    2    0    0    1    3    1    1    1]
 [   0 1262    1    0    2    0    4    2    1    0]
 [   2    3 1182    2    1    0    0    4    1    1]
 [   0    0    4 1216    1   12    0    1    6    2]
 [   0    2    0    0 1178    0    0    2    1    4]
 [   2    0    2    6    1 1068    8    0    4    6]
 [   0    0    0    0    1    2 1167    0    0    0]
 [   0    1    1    2    0    0    0 1291    1    1]
 [   2    0    3    1    3    0    1    2 1182    4]
 [   2    2    0    2   10    1    0   10    8 1112]]
```

(b) Confusion Matrix on Validation Set.



```
Confusion Matrix (Test Set):
[[ 975    1    0    1    0    0    2    0    1    0]
 [   0 1127    1    1    1    0    5    0    0    0]
 [   2    2 1011    2    1    0    2    8    4    0]
 [   0    0    3  994    0    8    0    0    5    0]
 [   0    0    0    0  977    0    0    0    2    3]
 [   0    0    1    7    0  880    2    0    2    0]
 [   4    2    1    0    4    1  945    0    1    0]
 [   0    2    4    4    0    0    0 1016    1    1]
 [   3    0    2    0    2    1    0    1  961    4]
 [   0    2    0    1   14    5    0    6    7  974]]
```

(c) Confusion Matrix on Test Set.

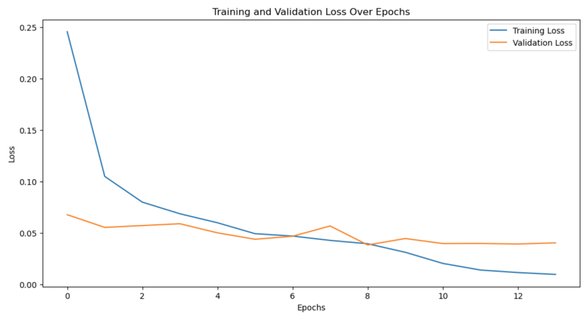


(d) Loss vs. Epochs.

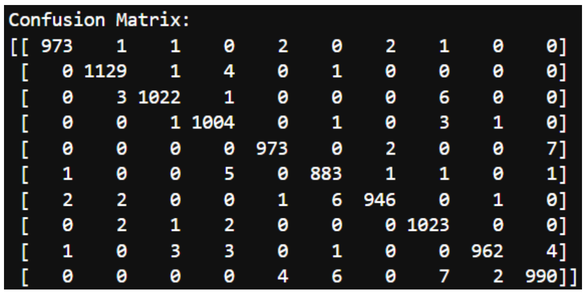
Figure 2: Secondary Results with Basic CNN.

3 Increasing the Depth of the CNN

In the second experiment, the aim is to make the CNN model more powerful by increasing its depth. The dataset is processed the same as before. The training set is further split into training (80%) and validation (20%). Each set of data is also normalized the same as before. The model architecture in this scenario consists of 10 layers. The main differences are the addition of 2 batch normalization layers, a second convolutional layer (64 filters, 3×3, ReLU), a second max pooling layer, and a layer for dropout (0.5). A learning rate scheduler is also implemented to adjust the learning rate during training. At every 10 epochs for a total of 30 epochs, the learning rate is adjusted from 0.001 to 0.0001 to 0.00001, respectively. An early stopping criteria is defined to monitor the validation loss with a patience of 5 meaning that if the validation loss does not drastically change for 5 consecutive epochs, the training will end early. The model is compiled with the same features as the previous experiments: Adam optimizer and sparse_categorical_crossentropy loss function. Due to the early stopping criteria, training halted after 14 epochs with training and validation accuracies reaching $\approx$99.7% and 99.2%, respectively. The testing accuracy came out to $\approx$99.1%. This experiment shows that a deeper model is able to learn more about the data than the original basic model from the previous experiments. Each set of data had an accuracy increase of 0.7%, 0.5%, and 0.5%, respectively. The resulting confusion matrix for the test set and loss curves are depicted in Figure 3.



(a) Loss vs. Epochs.



```
Confusion Matrix:
[[ 973    1    1    0    2    0    2    1    0    0]
 [    0 1129    1    4    0    1    0    0    0    0]
 [    0    3 1022    1    0    0    0    6    0    0]
 [    0    0    1 1004    0    1    0    3    1    0]
 [    0    0    0    0  973    0    2    0    0    7]
 [    1    0    0    5    0  883    1    1    0    1]
 [    2    2    0    0    1    6  946    0    1    0]
 [    0    2    1    2    0    0    0 1023    0    0]
 [    1    0    3    3    0    1    0    0  962    4]
 [    0    0    0    0    4    6    0    7    2  990]]
```

(b) Confusion Matrix on Test Set.

Figure 3: Results with Deeper CNN Model.

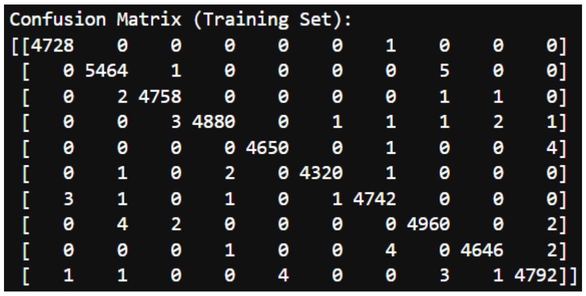
I ran this experiment a second time with the same architecture and model to present the confusion matrices of all three sets of data. Figure 4 depicts the confusion matrices of the training, validation, and testing set. Each set in this second run reached $\approx$99.7%, 99.3%, and 99.2%, respectively. The resulting loss plot is shown in Figure 4d. Interestingly enough, the previous comment in the preliminary experiments about the model possibly overfitting as the validation loss curve seemed to be increasing seems to be incorrect. With just 5 epochs in the preliminary results, the loss curve seems to end while increasing. However, in the experiments with a deeper model, we can see that at around epoch 5, the loss increases, but in subsequent epochs it begins to decrease and level off. This tells me that the preliminary experiments could possibly be capable of reaching the same levels of accuracy, but lacked the necessary time to train long enough to reach those levels.

4 Deeper Model + K-Fold Cross Validation

The next experiment implements the same model and architecture with added K-Fold Cross Validation. The model is ran with 5 folds, reporting the training and testing accuracies. Table 1 presents the training and testing accuracies for each fold. Early stopping occurred at epochs 17, 7, 7, 6, and 6, respectively for each fold. The average accuracy for training and testing were $\approx$99.77% and 99.11%, respectively. The resulting confusion matrices for the test set in each fold are depicted in Figure 5.

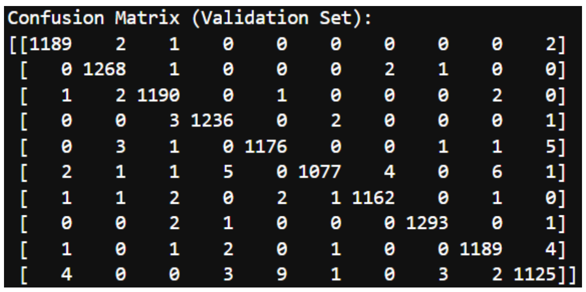
Data	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Training	99.92	99.61	99.80	99.82	99.69
Testing	99.27	98.97	99.09	99.25	98.95

Table 1: Accuracies with K-Fold Cross Validation.



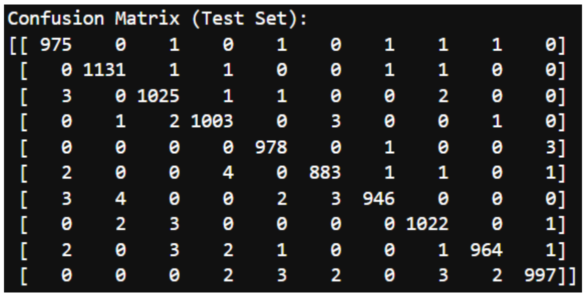
```
Confusion Matrix (Training Set):
[[4728    0    0    0    0    0    1    0    0    0]
 [   0 5464    1    0    0    0    0    5    0    0]
 [   0    2 4758    0    0    0    0    1    1    0]
 [   0    0    3 4880    0    1    1    1    2    1]
 [   0    0    0    0 4650    0    1    0    0    4]
 [   0    1    0    2    0 4320    1    0    0    0]
 [   3    1    0    1    0    1 4742    0    0    0]
 [   0    4    2    0    0    0    0 4960    0    2]
 [   0    0    0    1    0    0    4    0 4646    2]
 [   1    1    0    0    4    0    0    3    1 4792]]
```

(a) Confusion Matrix on Training Set.



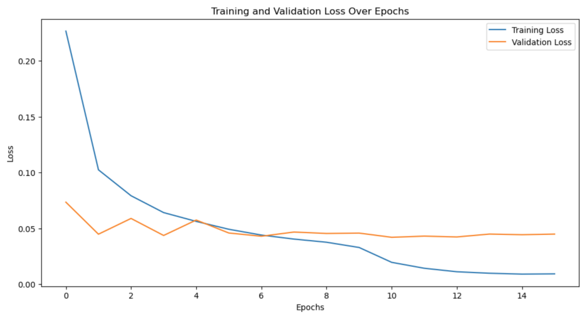
```
Confusion Matrix (Validation Set):
[[1189    2    1    0    0    0    0    0    0    2]
 [   0 1268    1    0    0    0    2    1    0    0]
 [   1    2 1190    0    1    0    0    0    2    0]
 [   0    0    3 1236    0    2    0    0    0    1]
 [   0    3    1    0 1176    0    0    1    1    5]
 [   2    1    1    5    0 1077    4    0    6    1]
 [   1    1    2    0    2    1 1162    0    1    0]
 [   0    0    2    1    0    0    0 1293    0    1]
 [   1    0    1    2    0    1    0    0 1189    4]
 [   4    0    0    3    9    1    0    3    2 1125]]
```

(b) Confusion Matrix on Validation Set.



```
Confusion Matrix (Test Set):
[[ 975    0    1    0    1    0    1    1    1    0]
 [   0 1131    1    1    0    0    1    1    0    0]
 [   3    0 1025    1    1    0    0    2    0    0]
 [   0    1    2 1003    0    3    0    0    1    0]
 [   0    0    0    0  978    0    1    0    0    3]
 [   2    0    0    4    0  883    1    1    0    1]
 [   3    4    0    0    2    3  946    0    0    0]
 [   0    2    3    0    0    0    0 1022    0    1]
 [   2    0    3    2    1    0    0    1  964    1]
 [   0    0    0    2    3    2    0    3    2  997]]
```

(c) Confusion Matrix on Test Set.



(d) Loss vs. Epochs.

Figure 4: Secondary Results with Deeper CNN.

I ran this experiment a second time with the same architecture and model to present the loss curve plots. Table 2 presents the training and testing accuracies for each fold. Early stopping occurred at epochs 16, 6, 7, 6, and 6, respectively for each fold. The average accuracy for training and testing were ≈99.75% and 99.24%, respectively. Figure 6 depicts the loss curves for each fold for training and validation sets. Because of the early stopping criteria, I am more inclined to think that the model begins to overfit in Folds 3-5 because of the validation loss increasing before being stopped. In the first experiment there was no early stopping criteria so the fact that the loss was increasing after the final epoch could not be fully trusted. The fact early stopping is implemented here and the loss is still increasing shows the model could be overfit.

Data	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Training	99.89	99.66	99.72	99.73	99.77
Testing	99.26	99.16	99.26	99.22	99.29

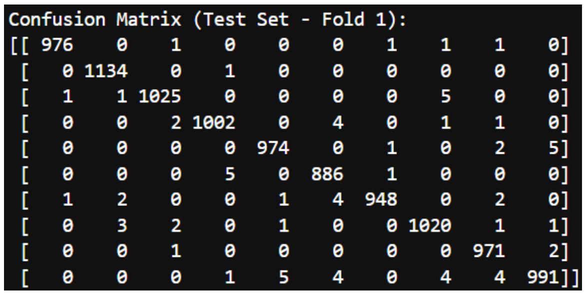
Table 2: Accuracies with K-Fold Cross Validation (Second Run).

5 Best Hyperparameter Search + K-Fold Cross Validation

The final experiment searches for the best hyperparameters while also implemennting K-Fold Cross Validation. The hyperparameters in question are the learning rate and the dropout rate for the model. This experiment uses the same model and architecture of the deeper CNN described above. The only difference is fine-tuning the learning rate and dropout to get the optimal results. Table 3 shows the combination of values tested during training.

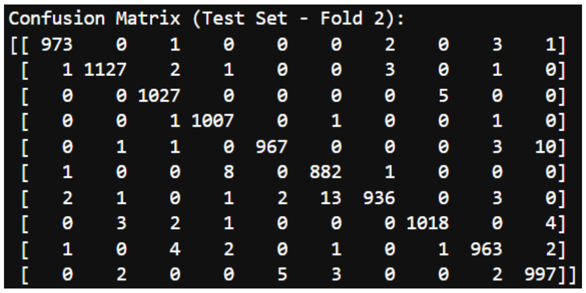
Learning Rate	Dropout
$[0.1, 0.01, 0.001, 0.0001]$	$[0.3, 0.5, 0.7]$

Table 3: Tested Hyperparameters.



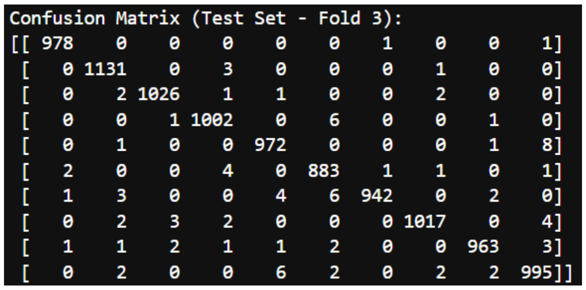
```
Confusion Matrix (Test Set - Fold 1):
[[ 976    0    1    0    0    0    1    1    1    0]
 [   0 1134    0    1    0    0    0    0    0    0]
 [   1    1 1025    0    0    0    0    5    0    0]
 [   0    0    2 1002    0    4    0    1    1    0]
 [   0    0    0    0  974    0    1    0    2    5]
 [   0    0    0    5    0  886    1    0    0    0]
 [   1    2    0    0    1    4  948    0    2    0]
 [   0    3    2    0    1    0    0 1020    1    1]
 [   0    0    1    0    0    0    0    0  971    2]
 [   0    0    0    1    5    4    0    4    4  991]]
```

(a) Confusion Matrix on Test Set (Fold 1).



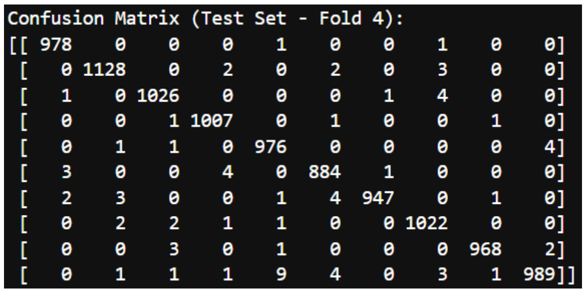
```
Confusion Matrix (Test Set - Fold 2):
[[ 973    0    1    0    0    0    2    0    3    1]
 [   1 1127    2    1    0    0    3    0    1    0]
 [   0    0 1027    0    0    0    0    5    0    0]
 [   0    0    1 1007    0    1    0    0    1    0]
 [   0    1    1    0  967    0    0    0    3   10]
 [   1    0    0    8    0  882    1    0    0    0]
 [   2    1    0    1    2   13  936    0    3    0]
 [   0    3    2    1    0    0    0 1018    0    4]
 [   1    0    4    2    0    1    0    1  963    2]
 [   0    2    0    0    5    3    0    0    2  997]]
```

(b) Confusion Matrix on Test Set (Fold 2).



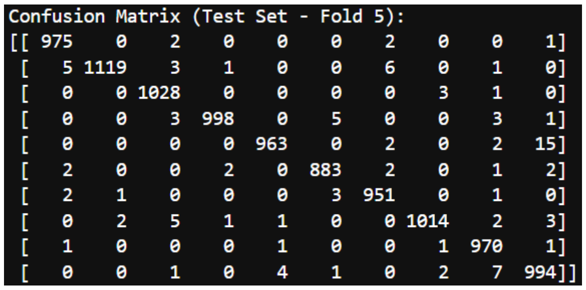
```
Confusion Matrix (Test Set - Fold 3):
[[ 978    0    0    0    0    0    1    0    0    1]
 [   0 1131    0    3    0    0    0    1    0    0]
 [   0    2 1026    1    1    0    0    2    0    0]
 [   0    0    1 1002    0    6    0    0    1    0]
 [   0    1    0    0  972    0    0    0    1    8]
 [   2    0    0    4    0  883    1    1    0    1]
 [   1    3    0    0    4    6  942    0    2    0]
 [   0    2    3    2    0    0    0 1017    0    4]
 [   1    1    2    1    1    2    0    0  963    3]
 [   0    2    0    0    6    2    0    2    2  995]]
```

(c) Confusion Matrix on Test Set (Fold 3).



```
Confusion Matrix (Test Set - Fold 4):
[[ 978    0    0    0    1    0    0    1    0    0]
 [   0 1128    0    2    0    2    0    3    0    0]
 [   1    0 1026    0    0    0    1    4    0    0]
 [   0    0    1 1007    0    1    0    0    1    0]
 [   0    1    1    0  976    0    0    0    0    4]
 [   3    0    0    4    0  884    1    0    0    0]
 [   2    3    0    0    1    4  947    0    1    0]
 [   0    2    2    1    1    0    0 1022    0    0]
 [   0    0    3    0    1    0    0    0  968    2]
 [   0    1    1    1    9    4    0    3    1  989]]
```

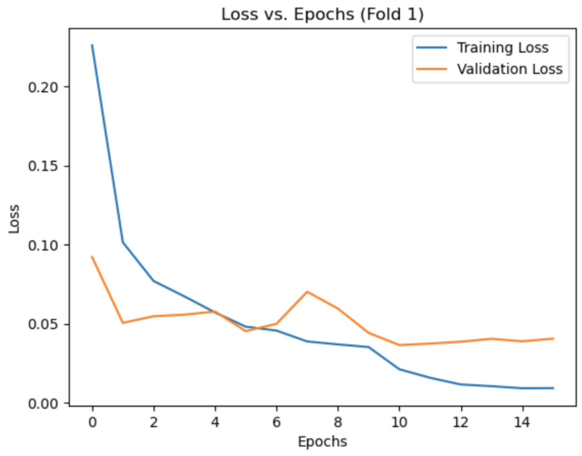
(d) Confusion Matrix on Test Set (Fold 4).



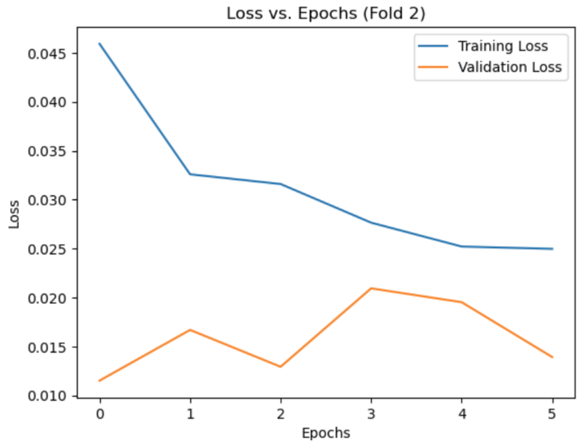
```
Confusion Matrix (Test Set - Fold 5):
[[ 975    0    2    0    0    0    2    0    0    1]
 [   5 1119    3    1    0    0    6    0    1    0]
 [   0    0 1028    0    0    0    0    3    1    0]
 [   0    0    3  998    0    5    0    0    3    1]
 [   0    0    0    0  963    0    2    0    2   15]
 [   2    0    0    2    0  883    2    0    1    2]
 [   2    1    0    0    0    3  951    0    1    0]
 [   0    2    5    1    1    0    0 1014    2    3]
 [   1    0    0    0    1    0    0    1  970    1]
 [   0    0    1    0    4    1    0    2    7  994]]
```

(e) Confusion Matrix on Test Set (Fold 5).

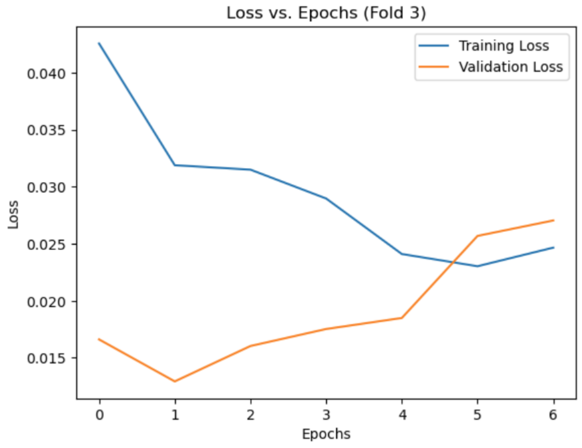
Figure 5: Confusion Matrices for the Test Set.



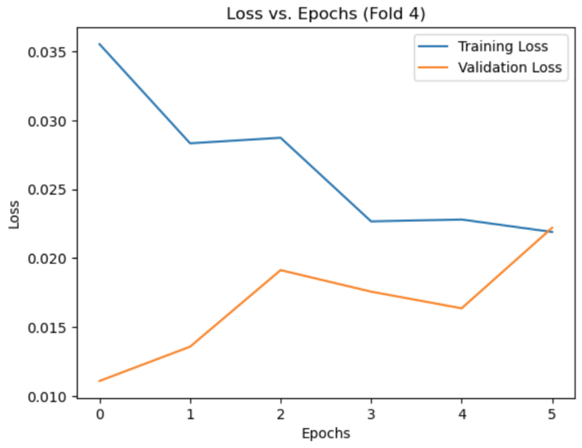
(a) Loss vs. Epochs (Fold 1).



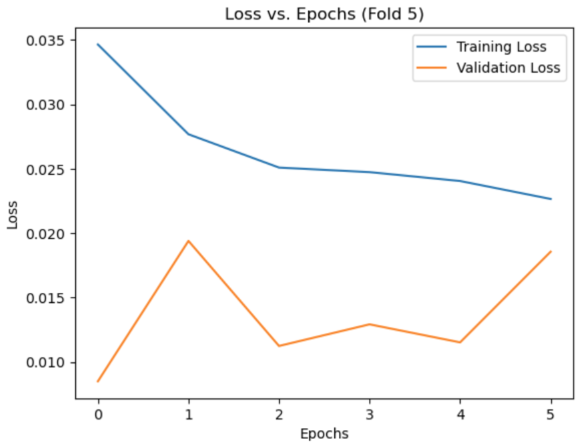
(b) Loss vs. Epochs (Fold 2).



(c) Loss vs. Epochs (Fold 3).



(d) Loss vs. Epochs (Fold 4).



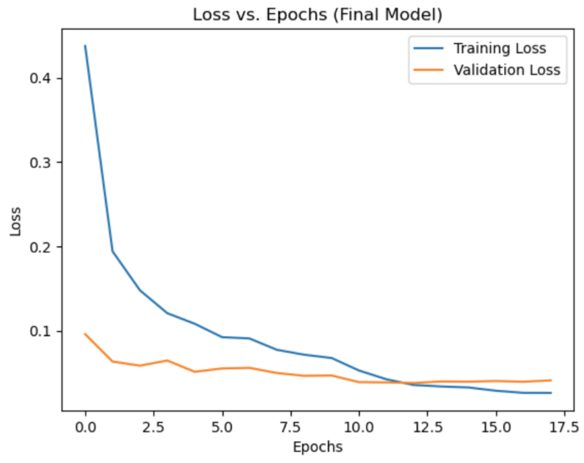
(e) Loss vs. Epochs (Fold 5).

Figure 6: Loss Curves for each Fold.

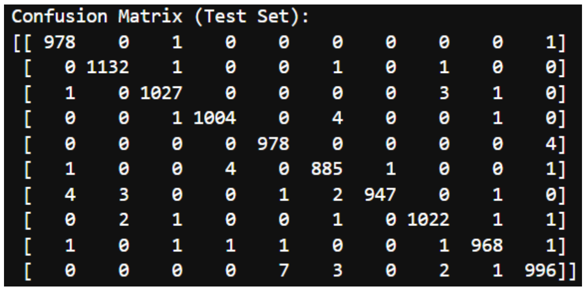
Each combination of hyperparameters was taken through the cross-validation phase meaning for every possible combination, a 5-Fold cross-validation was tested. Table 4 shows the performance of each combination at each fold. The hyperparameter combination yielding the highest average testing accuracy was when the learning rate = 0.001 and the dropout rate = 0.7. The final model used these hyperparameters to yield a final testing accuracy of ≈99.37% with early stopping around epoch 17. The final confusion matrix for the test set and loss curve plot are viewed in Figure 7.

Hyperparameters	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
LR: 0.1, D: 0.3	99.33	99.32	98.98	99.17	99.00
LR: 0.1, D: 0.5	99.09	99.39	99.43	99.30	99.36
LR: 0.1, D: 0.7	99.35	99.38	99.37	99.37	99.36
LR: 0.01, D: 0.3	98.99	98.96	99.00	99.17	99.30
LR: 0.01, D: 0.5	99.30	99.27	99.32	98.89	99.30
LR: 0.01, D: 0.7	99.28	99.41	99.40	99.35	99.27
LR: 0.001, D: 0.3	99.07	99.28	98.95	99.00	99.03
LR: 0.001, D: 0.5	99.28	98.83	99.37	99.31	99.05
LR: 0.001, D: 0.7	99.41	99.35	99.40	99.41	99.32
LR: 0.0001, D: 0.3	99.07	98.93	98.80	98.97	98.77
LR: 0.0001, D: 0.5	98.98	99.38	99.27	99.42	99.10
LR: 0.0001, D: 0.7	99.45	99.22	99.23	99.38	99.30

Table 4: Accuracies Over Each Hyperparameter Combination and at Each Fold.



(a) Loss vs Epochs (Final Model).



```
Confusion Matrix (Test Set):
[[ 978    0    1    0    0    0    0    0    0    1]
 [    0 1132    1    0    0    1    0    1    0    0]
 [    1    0 1027    0    0    0    0    3    1    0]
 [    0    0    1 1004    0    4    0    0    1    0]
 [    0    0    0    0  978    0    0    0    0    4]
 [    1    0    0    4    0  885    1    0    0    1]
 [    4    3    0    0    1    2  947    0    1    0]
 [    0    2    1    0    0    1    0 1022    1    1]
 [    1    0    1    1    1    0    0    1  968    1]
 [    0    0    0    0    7    3    0    2    1  996]]
```

(b) Final Confusion Matrix on Test Set (Final Model).

Figure 7: Results for the Final Model.

We can see that in this final model, the loss for training and validation both decrease and level off to where the early stopping criteria ends the training. Compared to the other experiments, this combination of hyperparameters proves to reach optimal performance.

6 Comparison to Literature

While this is not an exhaustive comparison, I compare my results to four papers that conduct similar CNN experiments on MNIST. The papers range from 2019 to 2023 in an effort to show how CNNs have evolved over recent years. For simplicity, I directly quote the model architectures and/or input the tables describing the model architectures of the various papers. They are all cited in the references in Section 8.

6.1 Model Design of Each Paper

Validation of Random Dataset using an efficient CNN model trained on MNIST handwritten Dataset [1]: Quoting directly from the paper, the model is designed with -

> *”4 convolutional layers, 32 filters of size 5x5 in first two and 64 filters of size 3x3 in the next two. ... There were 2 max-pooling layers each after 2 convolutional layers with strides of 2. ... The activation technique used after each convolutional network is Relu. There was a dropout of 0.25 after a pooling layer. After the features were extracted the final Matrix was flatten and fed as the input to the fully connected network. Activation technique used in the fully connected layer is Relu and Softmax with a dropout of 0.5.”*

CNN Model for Image Classification on MNIST and Fashion-MNIST Dataset [2]: Figure 8 is a table of each architecture combination used in this paper.

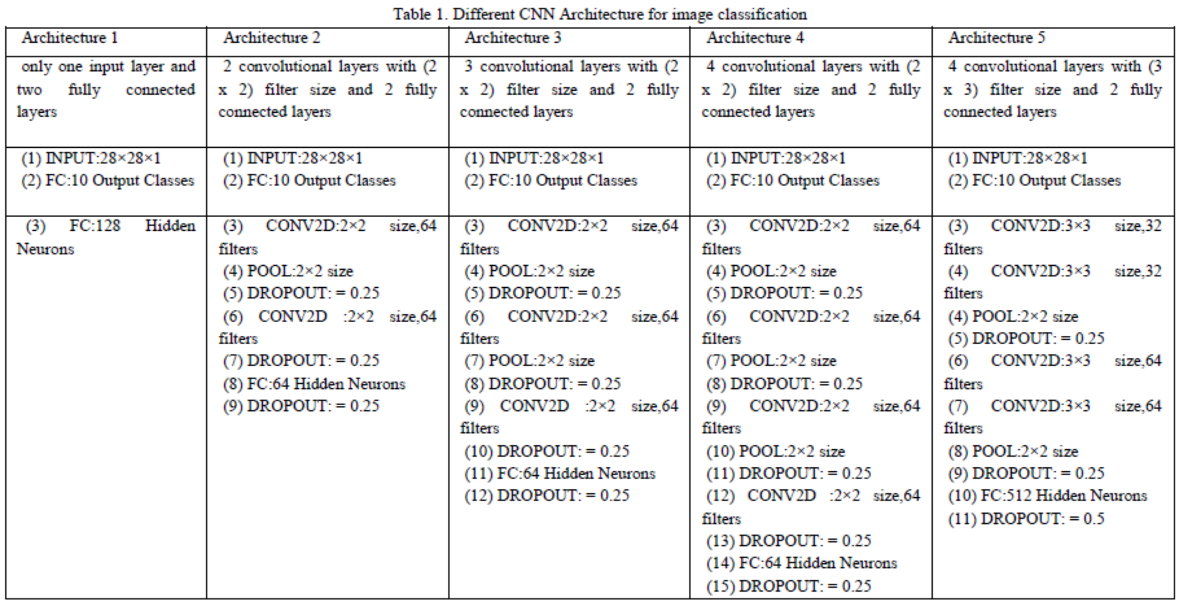


Table 1. Different CNN Architecture for image classification

Architecture 1	Architecture 2	Architecture 3	Architecture 4	Architecture 5
only one input layer and two fully connected layers	2 convolutional layers with (2 x 2) filter size and 2 fully connected layers	3 convolutional layers with (2 x 2) filter size and 2 fully connected layers	4 convolutional layers with (2 x 2) filter size and 2 fully connected layers	4 convolutional layers with (3 x 3) filter size and 2 fully connected layers
(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes
(3) FC:128 Hidden Neurons	(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D :2×2 size,64 filters (7) DROPOUT: = 0.25 (8) FC:64 Hidden Neurons (9) DROPOUT: = 0.25	(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:2×2 size,64 filters (7) POOL:2×2 size (8) DROPOUT: = 0.25 (9) CONV2D :2×2 size,64 filters (10) DROPOUT: = 0.25 (11) FC:64 Hidden Neurons (12) DROPOUT: = 0.25	(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:2×2 size,64 filters (7) POOL:2×2 size (8) DROPOUT: = 0.25 (9) CONV2D:2×2 size,64 filters (10) POOL:2×2 size (11) DROPOUT: = 0.25 (12) CONV2D :2×2 size,64 filters (13) DROPOUT: = 0.25 (14) FC:64 Hidden Neurons (15) DROPOUT: = 0.25	(3) CONV2D:3×3 size,32 filters (4) CONV2D:3×3 size,32 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:3×3 size,64 filters (7) CONV2D:3×3 size,64 filters (8) POOL:2×2 size (9) DROPOUT: = 0.25 (10) FC:512 Hidden Neurons (11) DROPOUT: = 0.5

Figure 8: Architectures Tested in Paper [2].

MNIST Handwritten Digit Recognition using Machine Learning [3]: Figure 9 is the output of *model.summary()* used in this paper.

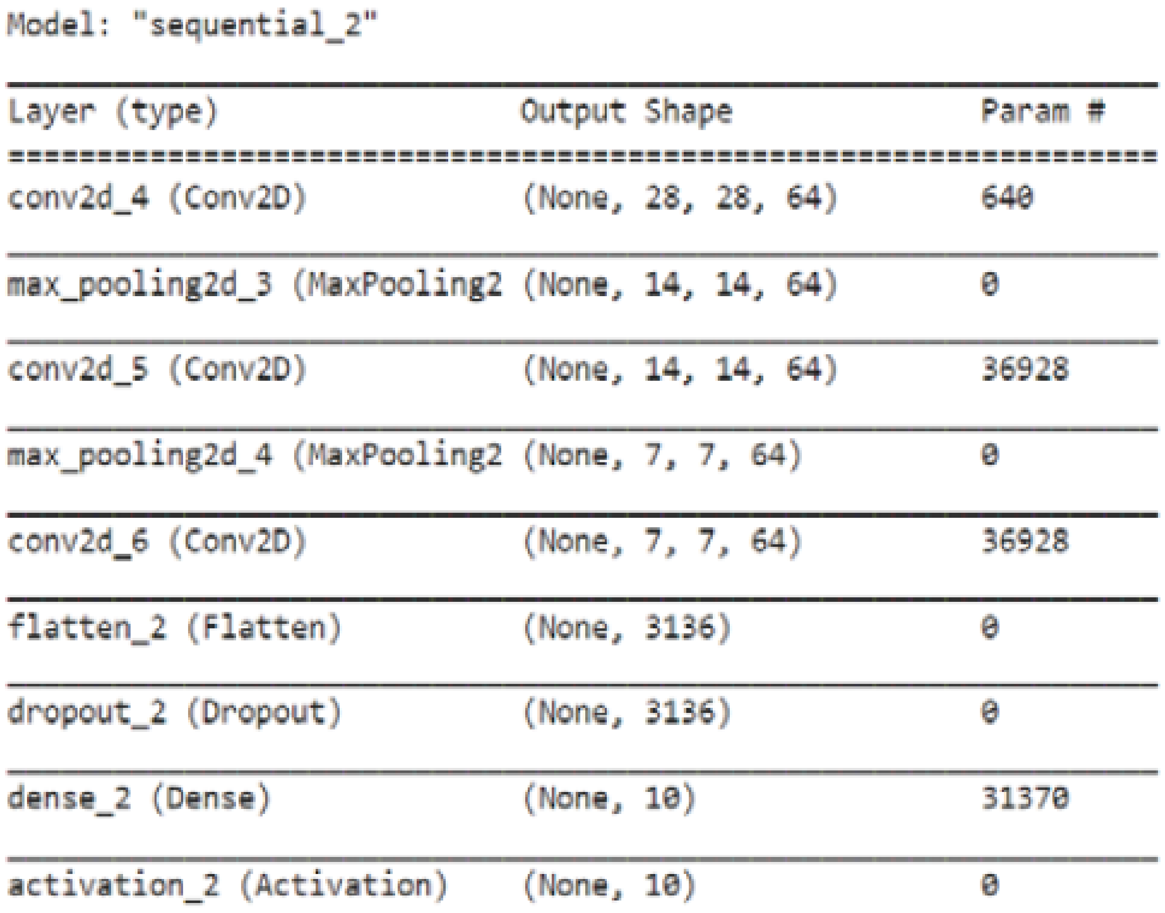
Towards the Distributed Wound Treatment Optimization Method for Training CNN Models: Analysis on the MNIST Dataset [4]: Quoting directly from the paper, the model is designed with -

> *”The CNN architecture consists of 25 filters or kernels of 12×12 for the convolutional layer. Next, the output of the convolutional layer is passed into a ReLU layer. Then, a fully connected layer with 10 outputs, one for each handwritten digit, is used for classification task. Finally, a softmax layer is used to obtain a vector of probability scores.”*

6.2 Performance Comparison

Table 5 depicts the training and testing accuracies of each paper and my own in this assignment. Note that paper [2] uses many combinations of architectures, so in the table [2-2] represents ”paper two, architecture two” and so on. Also note, results for architecture one in [2] are excluded because they are much more extensive and vary more from the simple work done in this assignment. A simple display of training and testing accuracy would not justify a thorough comparison.

The highest performances are bolded. Notice that [2-3] and my own final CNN model report the same testing accuracy. This is interesting and shows that not just one architecture is capable of achieving respectable performance rates. In particular, [2-3] uses three convolutional layers whereas my final model only uses two. After the first two pooling layers, dropout is introduced at 0.25 and after the final convolutional layer, dropout is introduced again. In my final model, I only introduce dropout at 0.7 and right before the output layer.


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 3136)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                31370
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
```
Figure 9: Architecture Tested in Paper [3].

Models	Training	Testing
Basic $CNN_{(mine)}$	99.5	98.6
Final $CNN_{(mine)}$	–	**99.37**
[1]	–	68.57
[2 – 2]	98.86	98.96
[2 – 3]	**99.60**	**99.37**
[2 – 4]	99.02	99.03
[2 – 5]	99.26	99.29
Simple CNN [3]	98.68	–
Deep CNN [3]	99.23	–
BatchNorm CNN [3]	99.30	–
[4]	–	85.41

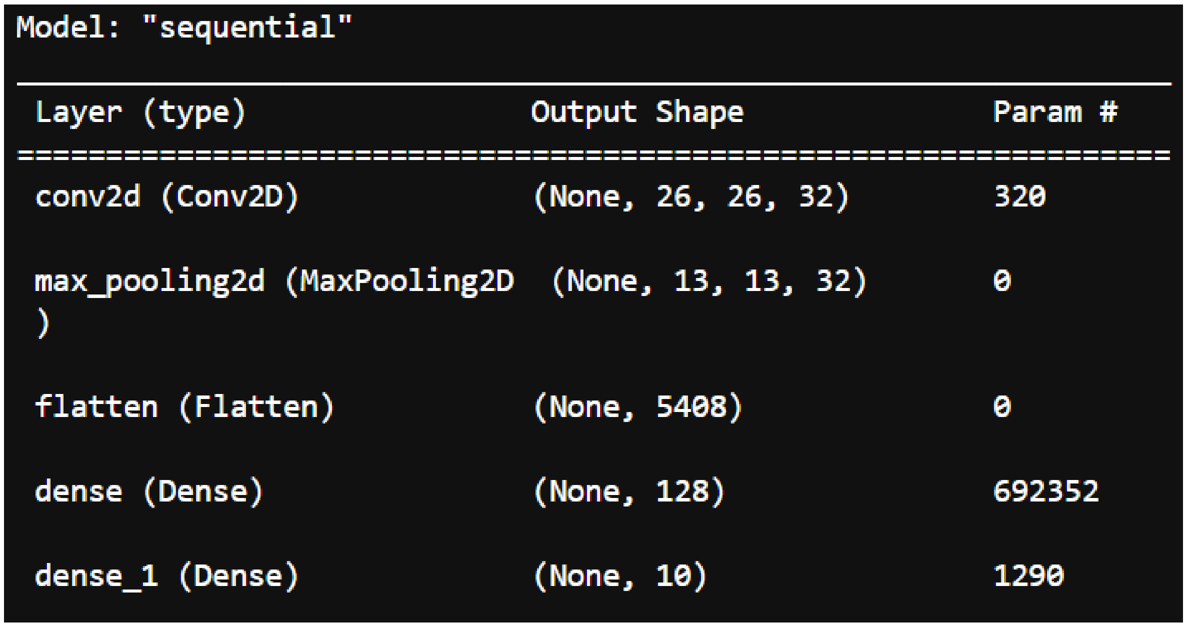
Table 5: Performance Comparisons.

6.3 How to Improve Experiments and Final Model

There are many factors that play a role in the possible improvement of my model. Although I trained with multiple learning rates and dropout rates, I do not include testing at multiple ranges of epochs or different batch sizes. [1] makes a note that they tested with multiple filter sizes in their CNN, but saw no noticeable difference in results. The results may not differ much, if any, but adjusting the filter size of my model could also possibly have some affect on the performance. [2], among many things, tested with multiple optimizers whereas I only experimented with the Adam optimizer. Adam is considered the best, but verifying the results for myself with other optimizers could make my experiments much more thorough. [4] uses an optimization method known as Wound Treatment Optimization (WTO) to train with a range of ”agents” in a distributed manner. While training, the authors also change the percentage of data that is used for training to watch the accuracy improve as 1.) number of agents increases and 2.) proportion of data used for training increases. This is another interesting angle that could improve my model if implemented properly.

7 Architectures Used in Each Experiment

This section illustrates the model summaries of each model used in this assignment. It is noted above in previous sections when a model architecture was reused. However, it is also noted that even though the same architecture is used, new results are generated and therefore reported as a new case.

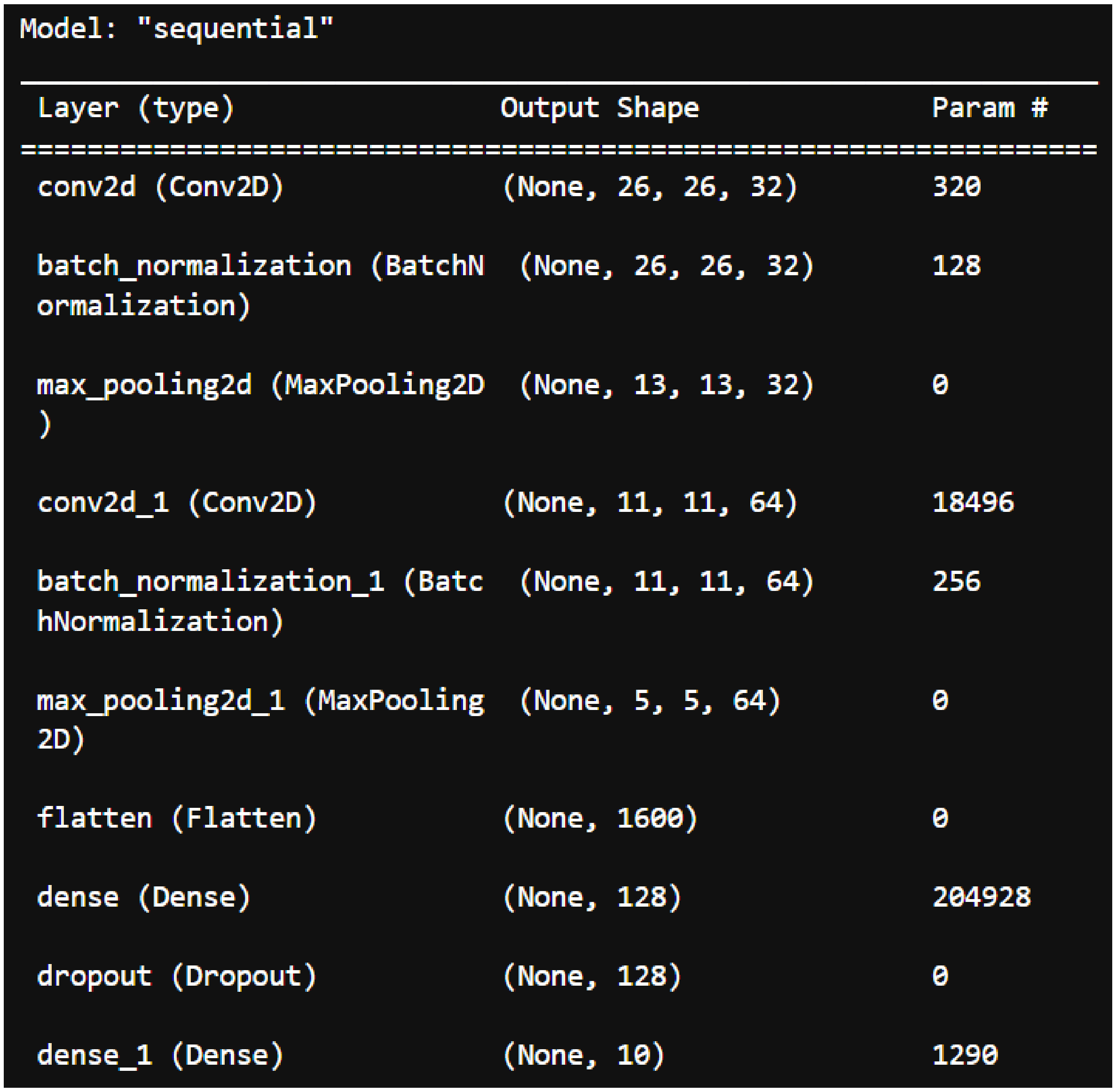


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 128)               692352

 dense_1 (Dense)             (None, 10)                1290
```

Figure 10: Architecture for Preliminary Results in Section 2.

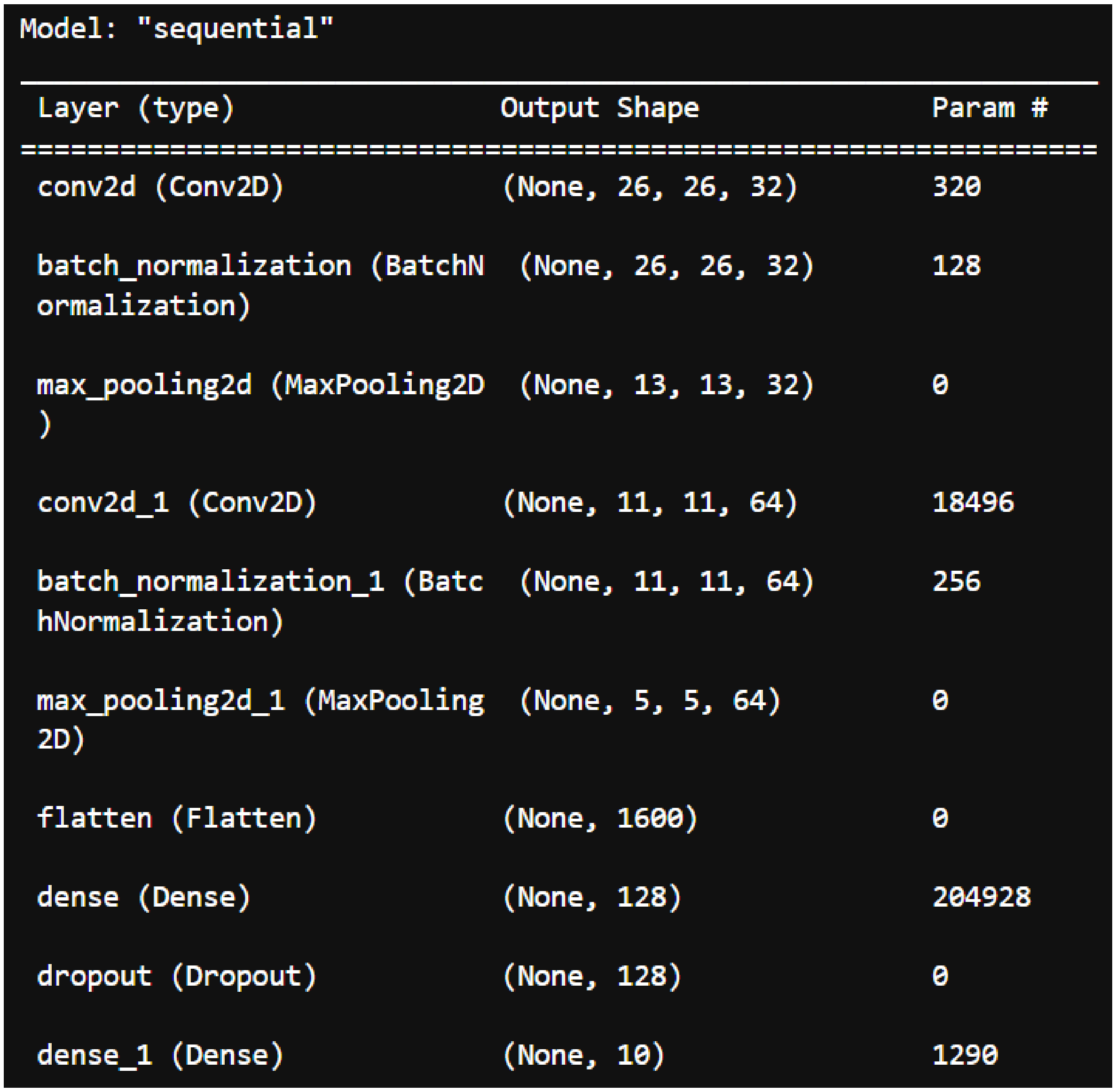
8 References

Below is a list of references used to assist with this assignment:
https://github.com/mattburtnz07/PyTorch-Convolution-Neural-Network/blob/master/beginners-guide-to-pytorch-cnn-s-by-a-beginner.ipynb
https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/
https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a
[1] Garg, A., Gupta, D., Saxena, S., & Sahadev, P. P. (2019, March). Validation of random dataset using an efficient CNN model trained on MNIST handwritten dataset. In 2019 6th International Conference on Signal Processing and Integrated Networks (SPIN) (pp. 602-606). IEEE.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (BatchN  (None, 26, 26, 32)       128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 batch_normalization_1 (Batc  (None, 11, 11, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 128)               204928

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290
```

Figure 11: Architecture for Deeper CNN Model in Sections 3 and 4.



```
Model: "sequential"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (BatchN  (None, 26, 26, 32)       128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 batch_normalization_1 (Batc  (None, 11, 11, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 128)               204928

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290
```

Figure 12: Architecture for Final Model in Section 5.

[2] Kadam, S. S., Adamuthe, A. C., & Patil, A. B. (2020). CNN model for image classification on MNIST and fashion-MNIST dataset. Journal of scientific research, 64(2), 374-384.
[3] Keerthi, T. (2022, April). MNIST Handwritten Digit Recognition using Machine Learning. In 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE) (pp. 768-772). IEEE.
[4] Ponce, H., Moya-Albor, E., & Brieva, J. (2023, March). Towards the Distributed Wound Treatment Optimization Method for Training CNN Models: Analysis on the MNIST Dataset. In 2023 IEEE 15th International Symposium on Autonomous Decentralized System (ISADS) (pp. 1-6). IEEE.