## Comparison of metrics between the three solutions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Accuracy | Training Loss |
| Network Trained on MNIST dataset | 0.97 | 0.97 | 0.97 | 0.97 | 0.99 | 0.01 |
| Pre-built Network Trained on AddNIST dataset | 0.79 | 0.80 | 0.80 | 0.80 | 0.94 | 0.17 |
| Network Trained on AddNIST dataset | 0.89 | 0.89 | 0.89 | 0.89 | 0.91 | 0.27 |

Figure 1: Comparison of metrics between the three solutions

## Part 1: Convolutional Neural Network trained on MNIST dataset

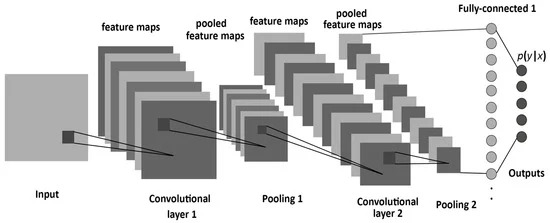


Figure 2: Convolutional neural network architecture

To predict MNIST handwritten digits, I chose to construct a convolutional neural network with one 2D convolution layer. After feature extraction, the feature map is downsampled using max pooling and flattened into a one-dimensional input tensor. I then applied a batch normalization layer to make the network more stable during training and reduce the number of epochs required to train the neural network. The flattened and normalized output is then passed into the first fully connected feature selection layer, with a ReLU activation function to avoid the vanishing gradient problem. A dropout layer is added between the two feature selection layers to prevent overfitting. For multi-class classification, the softmax activation function suits us best and is used in the final dense layer to produce the model’s prediction.

Compiling the model, I chose stochastic gradient descent with a learning rate of 0.01, and 0.9 momentum, as it yielded slightly higher accuracy than the Adam optimizer. Converting the data labels to one-hot encoding and using the categorical cross-entropy loss function also yielded the best results, when compared to sparse categorical cross-entropy and Kullback Leibler divergence loss. When training the model, I chose the number of epochs and batch size that provided the fastest training speed, without sacrificing accuracy.

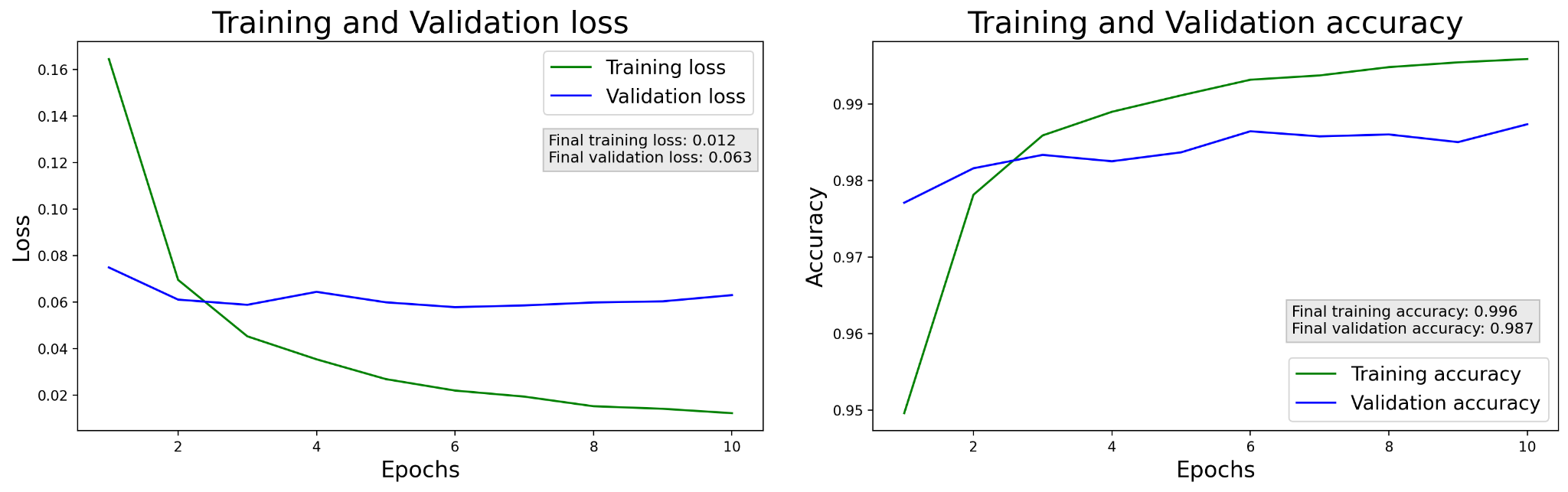


Figure 3: Part 1 Training and Validation accuracy and loss

This was the most successful, yet the most manual approach of the three, achieving 99% accuracy while training, 98% accuracy on the validation dataset, and 97% accuracy on the testing dataset. See below the classification report for Part 1, with precision, recall, and F1 scores for all predicted classes.

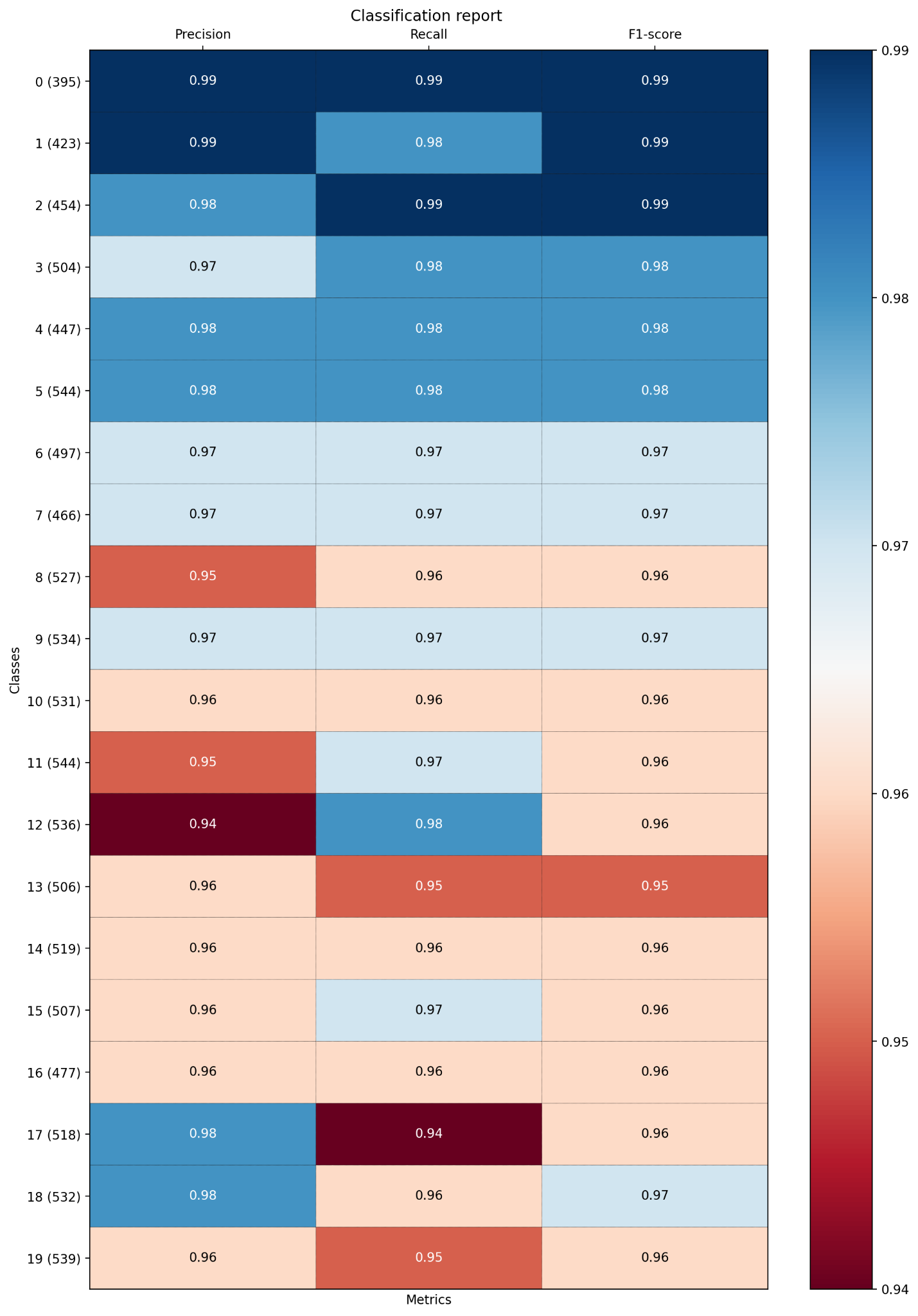


Figure 4: Part 1 Classification report

## Part 2: Pre-built DenseNet169 network trained on AddNIST dataset

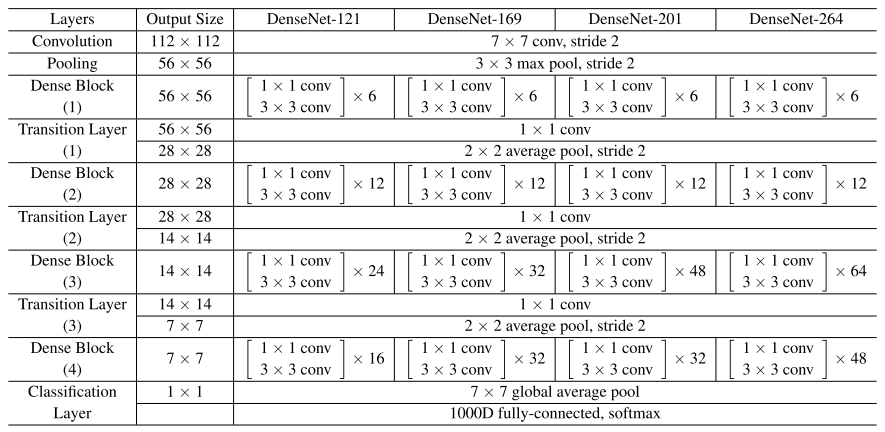


Figure 5: DenseNet architecture

To predict digit combinations from the AddNIST dataset using a pre-built deep neural network, I chose DenseNet169. I took the top layer off and added two layers above the pre-trained model, Input and Lambda, which take the images and resize them to fit the DenseNet model. I froze all layers except the convolutional layers in the final dense block (Dense Block (4)), to drastically reduce the training time. I used global average pooling instead of flattening the output of the pre-built network because flattening would make the next fully connected (FC) layer too computationally expensive, as the flattened output tensor shape would be (None, 81,536). The output is then passed into the first FC layer for feature selection, after which batch normalization and dropout are applied, before getting the final prediction using another FC layer, with 20 neurons and the softmax activation function.

This model uses the Adam optimizer and the same categorical cross-entropy loss function as the first model, as these have yielded the highest accuracy. At 10 epochs and a batch size of 64, the model achieves 94% training accuracy, although the validation accuracy is only 77%, meaning that the model is severely overfitting.

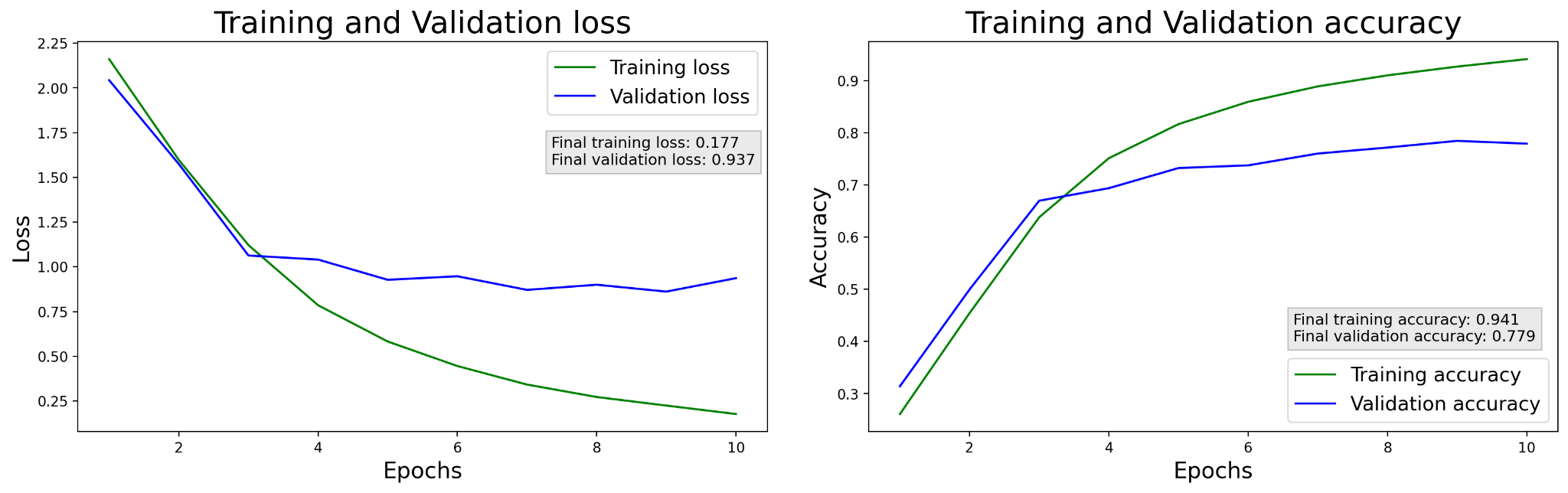


Figure 6: Part 2 Training and Validation accuracy and loss

The evaluation of the model on testing data once again proves that the model is overfitting. I will work on improving this, potentially by adding more dropout layers and constraining the model’s complexity. The model predicts many false positives for classes 15 and 16, as shown by the respective Precision of these two classes.

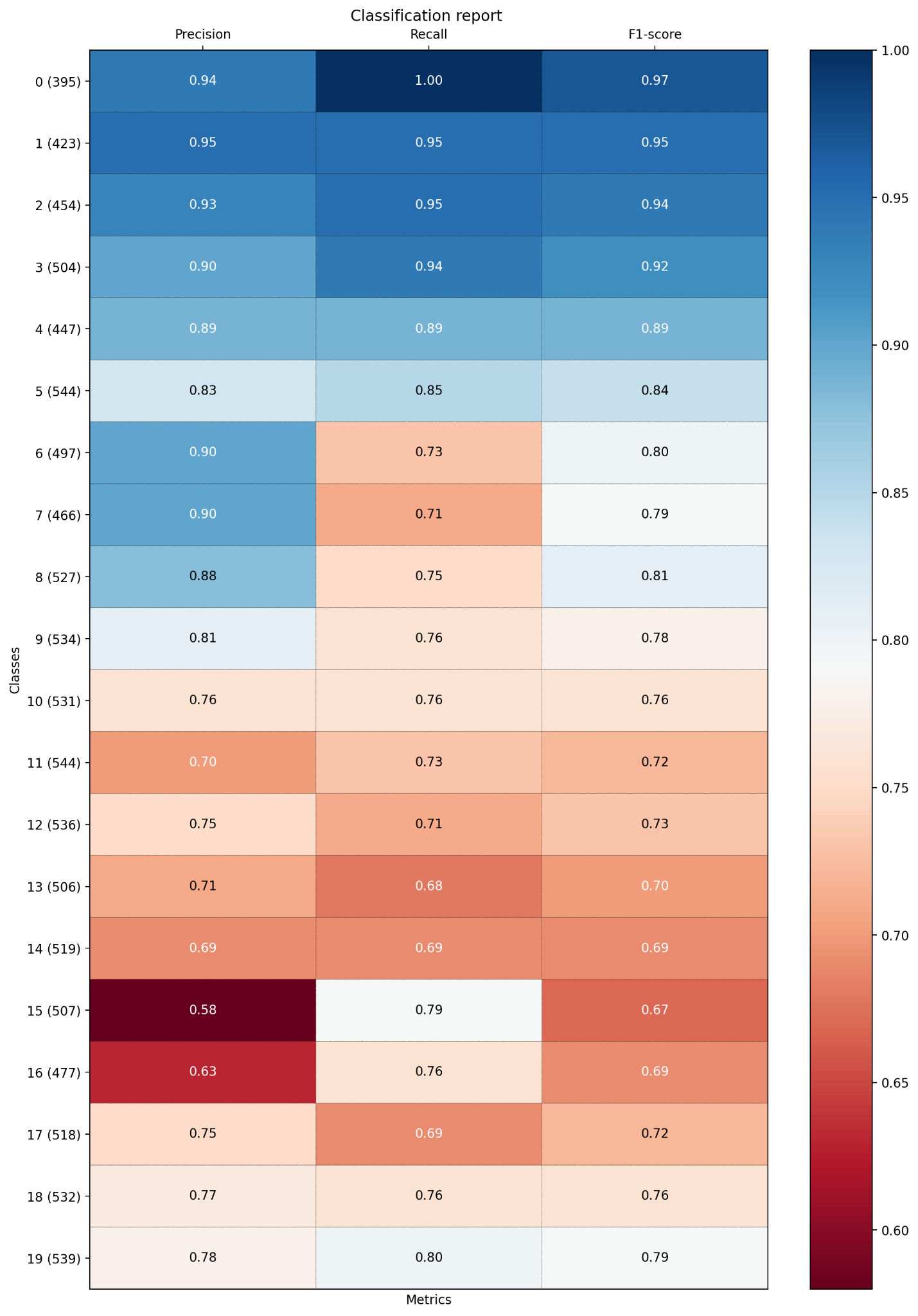


Figure 7: Part 2 Classification report

## Part 3: Convolutional Neural Network trained on AddNIST dataset

For training a model on the AddNIST dataset, I decided to take the same approach as in Part 1. This time, I used a much deeper convolutional neural network, with three convolution blocks, each with two convolutional layers with ReLU activation and max pooling. I addressed the overfitting problem of the previous model by adding a dropout layer to the second and third convolution blocks. To improve feature selection, I added two hidden fully connected layers with ReLU activation, before applying batch normalization and getting the final 20 class prediction with a final fully connected layer with softmax activation.

After experimenting with stochastic gradient descent learning rates, the combination of categorical cross-entropy and Adam optimizer still provided the highest accuracy and lowest loss scores. The relatively small size of the neural network allowed for me to use 20 epochs with a batch size of 64, yielding significantly better results.

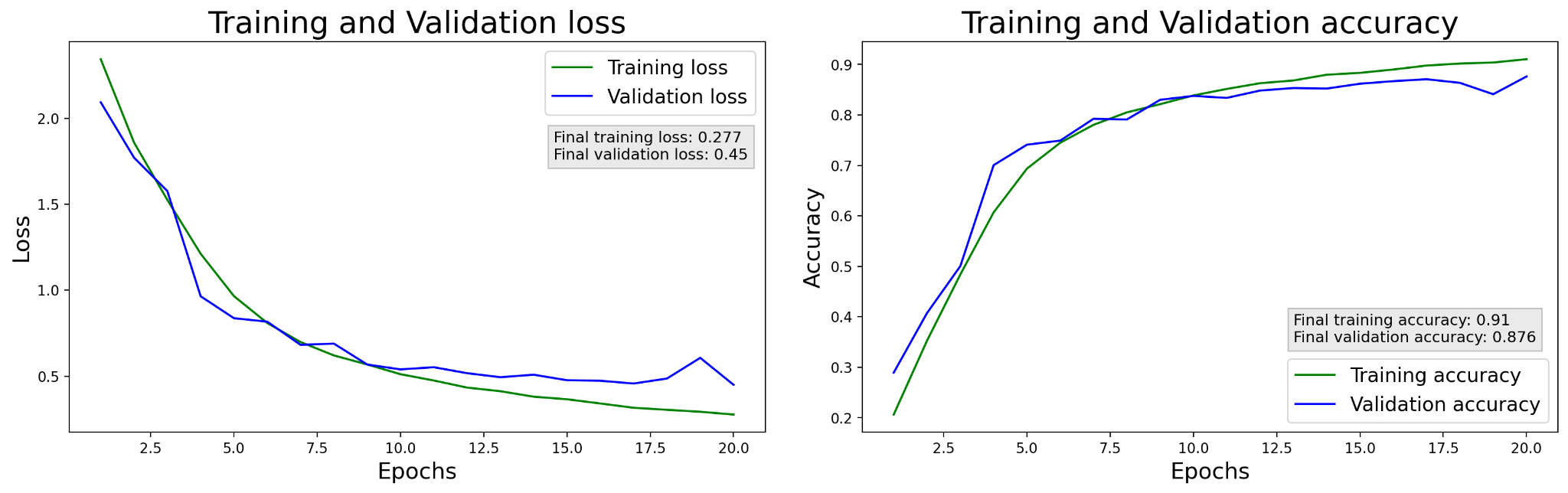


Figure 8: Part 3 Training and Validation accuracy and loss

The model performed slightly worse than DenseNet during training, with 91% accuracy, but dealt much better with overfitting, and maintained a high 87% accuracy on the validation dataset, and 89% accuracy on the testing dataset.



Figure 9: Part 3 Classification report