Zachary Wilkerson
U96598098

**EEL 6814**
**Homework 4 - Convolutional Neural Network (CNN)**

# Problem #1:

The MNIST Database of Handwritten Digits [1] is a collection of 60,000 training and 10,000 testing handwritten digit samples, each of size 28x28 pixels. Our goal is to classify the digit sample into one of the ten classes using a convolutional neural network (CNN).

For this problem, we coded our CNN in Python with the help of the Keras API [2]. We design the structure of our model based on a combination of two previous models: a CNN model built for image classification of the MNIST dataset [3] and a CNN model built for image classification of the CIFAR10 dataset [4]. Specifically, our final model design starts by passing input images into a convolution layer using a RELU activation function with a kernel size of 3x3, then into a max pooling layer with a pool size of 2x2, then a dropout layer whose rate will be tuned. This process is then repeated one more, and the output is flattened, sent through a dense layer whose activation function and number of units will be tuned, then through a dropout layer whose rate will be tuned, and finally through a dense layer that classifies the image into one of the ten digit classes using a softmax activation function. We compile this model using the Adam optimizer, whose learning rate will be tuned, with the sparse categorical cross-entropy loss function. We also record the accuracy of our model at each epoch.

The hyperparameters of our model were selected with the help of the Keras Tuner [5]. Specifically, the Hyperband algorithm [6] was used. This algorithm performs similar to a March Madness bracket, where we start out with a large number of models that are trained for a small amount of epochs and only the top performing half of the models move on and the process gets repeated until there is a designated winner. The benefit to this algorithm is that it is designed for adaptive resource allocation, along with early-stopping to decrease time to convergence. We set our Hyperband algorithm to maximize validation accuracy, using a maximum of 40 epochs with a maximum of 20 trials (number of hyperparameter combinations) and 2 executions (number of models to be built) per trial. These values selected are consistent with those found in [4]. To maintain reproducibility when using our Hyperband tuner, we set the seed to 55.

For all convolution layers, we set the model to tune using 16 and 64 (default) filters. For all dropout layers, we set the minimum rate to be 0.0, maximum rate to be 0.5, default rate to be 0.25, and a step size of 0.05 to tune with. For the second to last dense layer, we set the minimum number of units to be 32, maximum number of units to be 512, default number of units to be 128, and a step size of 32 to tune with. In that same layer, we also tune with three different activation functions: RELU (default), tanh, and sigmoid. We tune the learning rate of the Adam optimization algorithm used as well, setting the minimum value to be $1x10^{-4}$ and the maximum value to be $1x10^{-2}$ (default being $1x10^{-3}$), using log sampling. Again, these values selected are consistent with those found in [4].

We perform our search for the best hyperparameters of our model, taking 10% of our training data set to be used as a validation set. After 10 hours and 18 minutes and 90 trials, we

found our best performance on the validation set to be 98.89% accuracy. The top 3 models found, and their corresponding tuned parameters, are displayed in Table 1. We then fit the models to the training data using the optimized epoch numbers (based on early-stopping) and then evaluated them on the test data, with their test accuracy being added to Table 1 and their confusion matrices being shown in Figure 1. Note that our third best-fitting model on the validation set proved to be the best-fitting model on the test set.

| | Model #1 | Model #2 | Model #3 |
|---|---|---|---|
| Number of Filters in First Convolutional Layer | 16 | 16 | 64 |
| Rate in First Dropout Layer | 0.35 | 0.15 | 0.2 |
| Number of Filters in Second Convolutional Layer | 64 | 64 | 64 |
| Rate in Second Dropout Layer | 0.15 | 0.4 | 0.25 |
| Number of Units in First Dense Layer | 160 | 160 | 288 |
| Activation Function of First Dense Layer | Sigmoid | RELU | Sigmoid |
| Rate in Third Dropout Layer | 0.2 | 0.0 | 0.05 |
| Learning Rate | 0.0009 | 0.0007 | 0.0002 |
| Best Epochs to Train For | 59 | 79 | 72 |
| Validation Accuracy | 98.89% | 98.87% | 98.86% |
| Test Accuracy | 98.73% | 99.01% | 99.02% |

*Table 1: Hyperparameters tuned using Hyeprband Keras Tuner for top three performing CNN models (based on validation accuracy).*

Zachary Wilkerson
U96598098

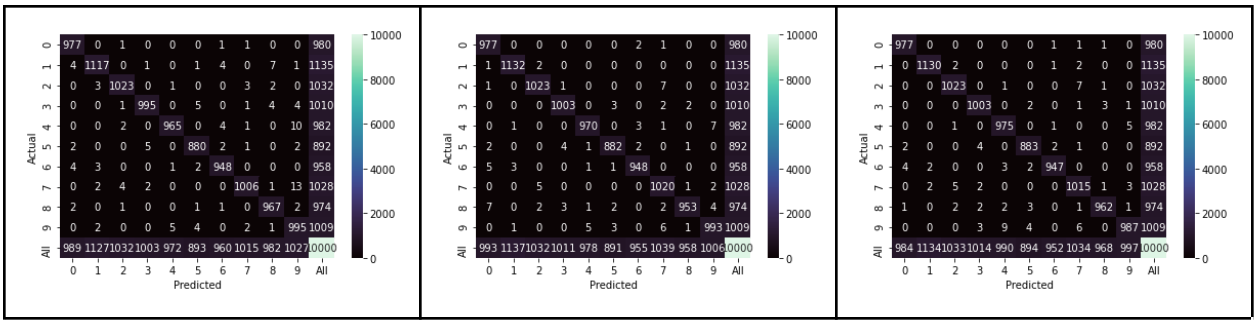


*Figure 1: Confusion matrices for top three performing CNN models (model #1 on the left, model #2 in the middle, model #3 on the right).*

Based on these results, we determine the best CNN model we came up with to be model #2, as it is able to achieve a comparable test accuracy to model #3 with fewer parameters to be learned (as the first dense layer has less units).

To improve our model, a few modifications are proposed based on past literature. To start, our model could be provided with more data via data augmentation to help learning. Models like EnsNet [7], which obtained a 99.84% accuracy on the MNIST dataset, utilized as many as 5 different augmentation techniques. The Branching/Merging CNN + Homogeneous Vector Capsules model [8] was able to obtain 99.87% accuracy utilizing augmentation as well.

On top of this, we could also modify the structure of our model, such as removing some dropout layers, adding more convolutional layers, including more dynamic ranges for the number of filters for each convolutional layer, using a different optimization algorithm, and other approaches. [7] showed that varying the location of dropout layers affected the overall accuracy of the model. [8] showed that using Homogeneous Vector Capsules (HVCs) allowed the model to score higher accuracy with fewer parameters and less computation expense. While HVCs might be a bit out of the scope of this course, it is interesting to note new modifications to model structures to be able to extract more with less.

It would also be interesting to ensemble the top 3 CNN's together to make our final prediction. We anticipate this could improve our systems accuracy, especially based on the results of our confusion matrices -- some models perform better for some numbers compared to others, such as model #2 with the number 1. Both [7] and [8] made use of ensemble learning in their work and currently have the best performing models on the MNIST dataset, placing second and first place respectively.

All of these approaches could prove to help our model in terms of performance.

References:

[1] http://yann.lecun.com/exdb/mnist/
[2] https://keras.io/about/
[3] https://keras.io/examples/vision/mnist_convnet/
[4] https://www.sicara.ai/blog/hyperparameter-tuning-keras-tuner
[5] https://keras.io/api/keras_tuner/

[6] https://arxiv.org/pdf/1603.06560.pdf
[7] https://arxiv.org/pdf/2001.08856v1.pdf
[8] https://arxiv.org/pdf/2003.08562v3.pdf
[9] https://arxiv.org/pdf/2001.09136v6.pdf