ECE421 Assignment 4

Part 1) Building a CNN

1) Rato of split: Train: 0.8 (80%); 48,000 examples   0.8×60,000

Validation: 0.1 (10%); 6,000 examples   0.1×60,000

Test Set: 0.1 (10%); 6,000 examples   0.1×60,000

There are 48,000 training examples. If in each step of gradient descent, we randomly select 32 examples from the training set, and no example appears in more than one batch, then going through the entire training set takes $\frac{48,000}{32} = 1,500$ iterations.

Since going through the entire dataset once is one epoch, then in 30 epochs, there is $30 \times 1500 = 45,000$ iterations.

2) See code and explanation in notebook

3) See code and explanation in notebook

4) The training set is used by the learning algorithm to find the parameters in our model such that the model gives the highest accuracy (makes least mistakes) overall when evaluated on the training examples. By reducing error when applying the model on the data it was trained with, we hope to capture the underlying relationships between input and target variables as they appear in the training set. A validation set is a set of labelled data that were not shown to the model during learning/training. It's purpose is to determine whether a model is able to generalize, and thus be used for inference on input data not see in the training set. Three important uses of validation set are hyperparameter tuning, modell selection and detecting overfit. Usually, we train multiple models varying in model architecture and hyperparameter values for learning process; and select the model that performs best on new data (validation set). Overfit can be determined by tracking accuracy on train + validation sets during training - overfitted models exhibit high accuracy on training set but lower accuracy when evaluated on validation set.

Part 2: Training and Tuning a CNN

1) See code in notebook

```
params.assign(params.value - lr * grad)
```

2) See code in notebook

```
batch = train_indices[it:it+BATCH]
batch = val_indices[it: it+ BATCH]
```

3) See code and plots in notebook

4) A hyperparameter is a parameter that either controls the learning process or model architecture; they are not derived during learning but rather influence values of other parameters that are learned.

It is important not to evaluate accuracy on test set until all hyperparameters have been tuned because we should not tune hyperparameters on test data. The purpose of test set is to determine how well our final model; trained on training set and optimized on validation set, performs on unseen data. So, we should only measure test performance once. Hyperparameter tuning on test set is a form of "cheating" because we are optimizing the model based on test performance, so test set is no longer "unseen data" and we could end up overfitting hyperparameters to test set, making it difficult to realistically evaluate/measure generalization.

5) The 4 hyperparameters we will consider are number of output channels in convolution layer 1, batch size, learning rate and activation function (use one fctn for both layers.). One of the hyperparameters; number of output channels in convolution layer 1 involves the CNN architecture, as required. Activation fctn is also part of model architecture.

So: $H = \{$number of output channels in convolution layer 1, batch size, learning rate, activation function$\}$

The 2 sets of hyperparameters we pick are:

$H_1 = \{24, 32, 0.005, \text{sigmoid}\}$

$H_2 = \{32, 64, 0.01, \tanh\}$

6) Best validation accuracy as well as the corresponding epoch for which this occurs for Base Model and my two additional models are listed below. Each model was trained for 30 epochs.

Base Model: 23.60% at epoch 28 ($H_0 = \{16, 32, 9e\text{-}4, \text{relu}\}$)

M_1: 18.75% at epoch 30 ($H_1 = \{24, 32, 0.005, \text{sigmoid}\}$)

M_2: 32.55% at epoch 29 ($H_2 = \{32, 64, 0.01, \tanh\}$)

M_2 performs best in terms of validation accuracy. This is because the validation accuracy for M_2 is highest amongst the 3 models at the end of training. So, of the 3 hyperparameter sets tested, H_2 is best.

See code!

7) M2 should be picked as our final model. This is because it has the highest validation accuracy at the end of training, as we expect it to generalize best amongst the 3 trained models. This is a valid claim because validation set performance reflects performance on unseen data.

The final test set accuracy is 32.88% (See code in notebook)

Part 3: Trying Out a new Dataset

1) See code in notebook

2) See code in notebook.

The base model architecture is same as base model architecture for part 2 (2 conv layers followed by linear layer)

See code → 3) The hyperparameters we choose to tune are the batchsize, learning rate, number of output channels in each convolution layer, and activation function of each convolution layer. These are similar to what we investigated in Part 2. Ideally, I would perform a grid search over these 6 hyperparameters and select the model that performs best.

However, the problem only asks us to tune until we reach a 5-10%. So, given that grid search would take a lot of time and GPU usage is limited, I will take another approach. Specifically, I will focus on each hyperparameter separately; first optimizing over batch size, then learning rate, then # of output channels in convolution layer 1 and finally activation function of convolution layer 1.

Starting with base model, optimizing over batch_size demonstrates that batch_size should be set to 8, since it results in a trained model with highest validation accuracy. I experimented with batch_size = 8, 16, 32, 64

Setting batch_size = 8 and optimizing over learning rate demonstrates that learning rate should be 0.1, since it results in a trained model with highest validation accuracy. I experimented with learning_rate = 0.005, 0.01, 0.05 and 0.1.

At this point, our trained model has a validation accuracy of 99.93%, so there is no need to continue tuning over the other 4 hyperparameters because we are well past the 5-10% improvement. If we were performing poorly, we would just repeat this process for other hyperparameters.

See code → 4) We select the model with the highest validation accuracy (93.89%), because we expect it to be able to generalize best.

The test accuracy with this new model is 93.49%.

The hyperparameter settings of our final model are:

batch_size = 8

learning_rate = 0.1

output channels in layer 1 = 16

output channels in layer 2 = 32

activation fctn of layer 1 : relu

activation fctn of layer 2 : relu

kernel size in layer 1 : 2×2

kernel size in layer 2 : 2×2

pooling dimensions : 2×2

Part 4 (Discussion here, code in notebook)

1) 1 pick Question 1.

We will build our experiments on top of Part 3

Our best model has a validation accuracy of ≈ 95%, so we are not confident in achieving 5-10% improvements

So, our base model will be a model that performs worse on validation set (76.76%), specifically

base model → $\boxed{H_0 = \{8, 0.005, 16, 32, \text{'relu'}, \text{'relu'}, 2, 2, 2\}},$ where in general

H = { batch size, learning-rate, # output channels in layer 1, # output channels in layer2, layer1 activation fctn, layer2 activation fctn, layer 1 kernel size, layer 2 kernel size, pooling-dim }

Even though we did not need to tune # output channels in each layer, or activation function, we still included them in our design process, so we will omit their inclusion here.

Instead, the hyperparameters we will tune will be the size of our filter in each layer, and the pooling dimension.

We first optimize over (layer1 kernel size, layer2 kernel size) by picking the model that maximizes validation accuracy from a set of models with different settings for these hyperparameters. We try (2,3), (3,2), (3,3) and (4,4) to vary settings while keeping # weights small.

We then optimize over pooling dimension. Since the input image is 28×28, we try pooling dimensions (2,4), (4,2) and (4,4), in order to keep output dimensions integers without reducing input dimension too much.

Combining the 2 optimization procedures, a total of 7 models were trained. Their performance in both training and validation accuracy are summarized below: The final model was chosen to be one that gave highest validation accuracy.

Note: # epochs = 30, models listing in order they were trained

Model	layer 1 kernel size	layer 2 kernel size	pooling dimension	train accuracy	validation accuracy
model1	3	2	(2,2)	90.75%	85.84%
model2	2	3	(2,2)	95.23%	90.33%
model3	3	3	(2,2)	96.40%	92.84%
model4	4	4	(2,2)	98.25%	96.19%
model5	4	4	(2,4)	98.58%	95.94%
model6	4	4	(4,2)	98.80%	96.89%
model7	4	4	(4,4)	98.84%	96.59%

At this point, observe → model4 performs best on validation set, so optimize with (layer1 size, layer2 size) = (4,4)

highest validation accuracy

2) The model we select is model6, since it has highest validation accuracy and is thus expected to generalize best. The validation accuracy is well over 5-10% more than the base model, satisfying problem requirements.

3) The test set accuracy with the tuned model is 96.91%, which is much better than test set accuracy with the base model (76.47%). This could possibly be credited to a more complex layer1 that allows us to learn patterns better through more weights; as well as a pooling layer that is better at finding abstract features by pooling a larger region to get from input to output.