Hao Sheng (Jack) Ning

April 05th, 2023

**Assignment #5 Report**

**Short Description:**

For this project, we had an opportunity to turn past weeks’ lessons into workable programs, enabling us to experiment the process of building a model (from scratch or loading) and using a large amount of data to thoroughly train it. We added different types of convolutional neural network layers to the model and adjust their settings to experiment their effectiveness. Adding layers also helped us to understand how images are changed throughout the process, including number of pixels, channels and their abstractness.

**Required Images:**

**Task 1A** (A plot of the first six example digits)

Graphical user interface, application

Description automatically generated

**Task 1C** (a diagram of my network)

Diagram

Description automatically generated

Description: Diagram of my network, together with the image sizes and number of weights at the linear layer stage.

**Task 1D** (Collect the accuracy scores and plot the training and testing accuracy in a graph. Include this plot)

Accuracy and average loss score at the end of each epoch:

Epoch 1: Accuracy: 68.9%, Avg loss: 1.826439

Epoch 2: Accuracy: 87.4%, Avg loss: 0.713746

Epoch 3: Accuracy: 90.2%, Avg loss: 0.532802

Epoch 4: Accuracy: 91.5%, Avg loss: 0.442450

Epoch 5: Accuracy: 92.4%, Avg loss: 0.395925

Epoch 6: Accuracy: 93.5%, Avg loss: 0.349644

Epoch 7: Accuracy: 93.8%, Avg loss: 0.318461

Epoch 8: Accuracy: 94.4%, Avg loss: 0.287923

Epoch 9: Accuracy: 95.0%, Avg loss: 0.265544

Epoch 10: Accuracy: 95.3%, Avg loss: 0.249120

Chart, histogram

Description automatically generated

Description: The overall accuracy and loss scores are lower after each epoch of training.

**Task 1F** (Include a table (or screen shot) of your printed values and the plot of the first 9 digits in your report)

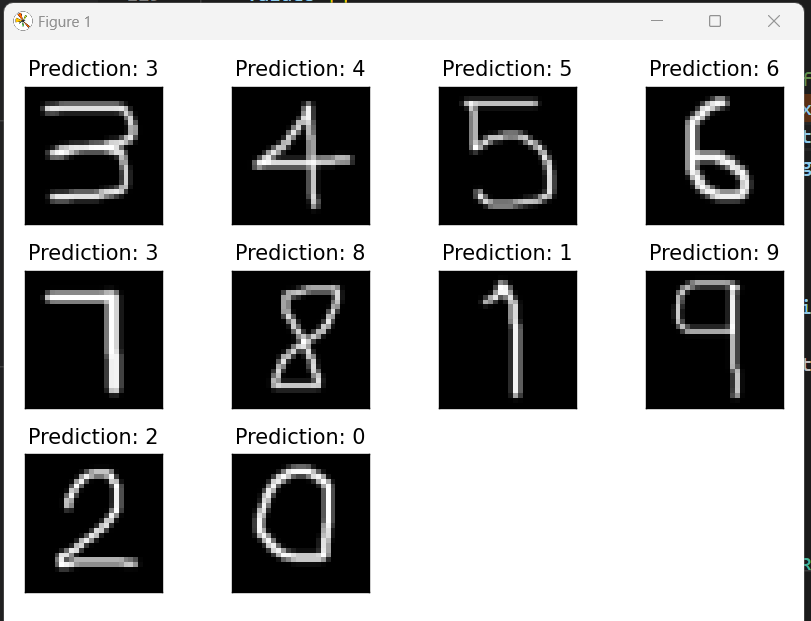
|  |  |  |  |
| --- | --- | --- | --- |
|  | 10 output values | index of the max output value | correct label of the digit |
| Example 1: | [ -8.9250,  -8.5618,  -8.0053,  -4.9688,  -2.8471,  -5.5932, -10.0255, -2.5603,  -5.2407,  -0.1648] | 9 | 9 |
| Example 2: | [-5.4570, -3.6299, -1.8536, -0.9732, -5.6756, -3.3779, -8.1233, -1.7375, -4.2598, -1.5763] | 3 | 3 |
| Example 3: | [-0.0586, -9.4175, -4.4945, -7.7929, -7.9652, -6.1000, -3.2582, -8.8236, -5.9440, -6.5717] | 0 | 0 |
| Example 4: | [-7.6126, -4.2080, -6.1460, -3.7689, -1.6982, -4.6247, -7.2595, -3.1894, -3.3760, -0.3703] | 9 | 9 |
| Example 5: | [-4.9375, -8.8029, -4.1713, -3.1061, -9.3136, -0.1054, -4.3559, -9.3898, -4.1560, -5.5696] | 5 | 5 |
| Example 6: | [-3.9328, -0.4299, -2.9716, -3.3344, -4.0018, -4.2801, -4.4056, -2.2363, -3.0632, -3.1002] | 1 | 1 |
| Example 7: | [-6.6344, -4.2752, -5.2044, -6.5391, -0.1183, -5.0388, -3.5821, -5.6046, -3.3293, -4.1613] | 4 | 4 |
| Example 8: | [-7.4358, -5.2494, -4.8368, -3.9306, -5.3627, -4.5087, -7.3168, -4.1380, -0.1181, -3.0857] | 8 | 8 |
| Example 9: | [ -0.0123, -11.5474,  -6.4549,  -8.1531, -12.1044,  -5.5671,  -6.6945, -8.3287,  -5.3759,  -7.7493] | 0 | 0 |
| Example 10: | [-3.1889, -0.7334, -2.7959, -4.3761, -2.2760, -6.5021, -4.0009, -3.8393, -1.4533, -3.6092] | 1 | 1 |

Description: Out of 10 output values, the highest is the one with highest confidence. And as seen above, all predicted classification matches with the actual label. (10/10)

**A picture containing logo

Description automatically generated**

**Task 1G** (Display how well the network performed on this new input in your report)



Description: Drew the digits using Microsoft Paint, and added to the folder, and loaded with datasets.ImageFolder function. All results match the actual label.

**Task 2B** (In your report, include the plot and note whether the results make sense given the filters)

**Graphical user interface, application

Description automatically generated**

Description: The result makes sense, because higher weight means filters have higher influence on the resulting images, and lower weights means filters have lower influence on the resulting images. As seen in the screenshot above, filters with higher weights (eg. filters with darker colors) tended to have bigger changes on the images, for example image applied by filter 9 looks very different from its original, and filters with lower weights (eg. filters with lighter color) didn’t do much change to the image, for example, image applied by Image 7 does not look too different from its original.

**Task 3B**

A plot of the training error:

Graphical user interface, chart, line chart

Description automatically generated

A printout of your modified network:

**Text

Description automatically generated**

The results on the additional data:

**A picture containing chart

Description automatically generated**

**Task 4** (Design your own experiment)

Description: Will experiment 61 different options from 5 different dimensions. Design and result shown below.

Original:

Text

Description automatically generated

**Dimension 1: Number of filters**

Develop a plan:

Convolutional Layer #1 currently contains 10 filters, and Convolutional Layer #2 currently contains 20 filters. My plan is to adjust the number of filters in each of convolutional layer by a scale factor in the range of [1/10, 1/5, 1/2, 1, 6/5, 3/2, 9/5, 2, 5/2, 3, 4, 5], there are in total 12 options. For example, a scale factor of 1/10 will change the number of filters in each layer to 1 (10\*1/10) and 2 (20\*1/10) respectively. The metrics I am looking after is accuracy, and average loss.

Predict result:

More filters a layer has means more patterns a layer is able to capture from the training sets, hence providing a more accurate prediction. Models with lower scale factors are expected to yield higher error (low accuracy), and those with higher scale factors will yield lower error (high accuracy).

Execution:

Text

Description automatically generated

Text

Description automatically generated

Conclusion of Dimension 1: The result completely complied with the hypothesis, more filters yields higher accuracy.

**Dimension 2: # of hidden nodes in the dense layer**

Develop a plan:

The plan is to adjust the number of hidden nodes in the range [10, 5, 4, 3, 2, 1, 1/2, 1/3, 1/4, 1/5, 0.15, 1/10], there are in total 12 options. The metrics I am looking after is accuracy, and average loss.

Predict result:

High number of hidden nodes can lead to large number of weights and in turns increases the risk of vanishing gradient, however low number of hidden nodes can also lead to problems, such as not enough weight applied to the model to thoroughly adjust the model. The optimal scale factor should be somewhere around 1 and 2.

Execution:

Text

Description automatically generated

Text

Description automatically generated

Conclusion of Dimension 2:

The result did not go exactly as my hypothesis. The result showed higher the number of hidden nodes produced higher accuracy and lower error. This trend might only be applicable to this specific scenario though.

**Dimension 3: # of batches**

Develop a plan:

Adjust the number of batch sizes in the range [1/8, 1/4, 1/2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. There are in total 13 options. The metrics I am looking after is accuracy, average loss and the time needed to complete the training.

Predict result:

Lower batch size means more frequent the model will do back propagation and calculate its error value to adjust the weighting on the edges connecting all the neural network nodes, hence lead to a more accurate result. However this will also lead to longer training time.

Execution:

Text

Description automatically generated

Text

Description automatically generated

Conclusion:

The result is exactly what my hypothesis expected, smaller batch size led to higher accuracy and lower average loss and longer training time.

**Dimension 4: Dropout rate**

Develop a plan:

Adjust the dropout rate in the range [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]. There are in total 11 options. The metrics I am looking after is accuracy, and average loss.

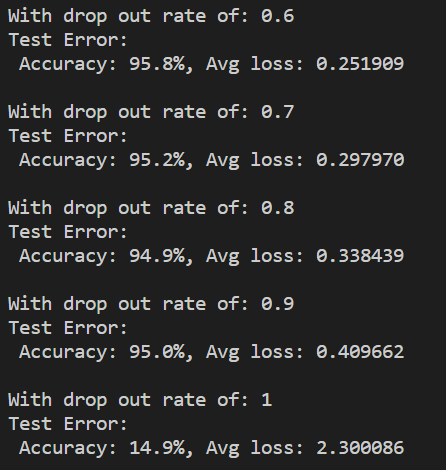
Predict result:

Optimal dropout rate depends on the model and the problem we are looking to solve, hence I predict the optimal dropout rate sits somewhere between 0.5 and 0.6.

Execution:

Text

Description automatically generated



Conclusion of Dimension 5:

My hypothesis is correct, the result shows the optimal dropout rate is around 0.6.

**Dimension 5: Activation functions**

Develop a plan:

Adjust the activation function in the options: [“Relu”,"ELU", "Hardshrink","Hardsigmoid", "Hardtanh", "Hardswish", "LeakyReLU", "LogSigmoid", "MultiheadAttention","PReLU", "ReLU6","RReLU"]. There are in total 12 options. The metrics I am looking after is accuracy, and average loss.

Predict result: The optimal activation function depends on the problem and the model. Based on some google searches, “ELU” should have the highest accuracy and lowest error rate

Execution:

Text

Description automatically generated

Text

Description automatically generated

Conclusion of Dimension 5:

My hypothesis is correct, ELU has the highest accuracy and lowest average loss.

**Most optimized**

Plan:

5 times number of filters in convolutional layers, 5 times number of hidden nodes, 60% dropout rate, batch size of 8, and activation function of ELU.

The metrics I am looking after is accuracy, and average loss.

Hypothesis: should produce a result of highest accuracy and lowest average loss.

Execution and result:

A screenshot of a computer

Description automatically generated with low confidence

Hypothesis is correct, the result did yield highest accuracy and lowest average loss.

**Extensions:**

1. Tried more Greek letters than alpha, beta, and gamma (Extention\_executeGreek.py)..

* Added more training data to the original data set, added **theta, lambda,** and **mu** inside greek\_train-6.zip.
* Added more testing data on these 3 extra Greek letters inside testDataGreek-6.zip

1. Evaluated more dimensions on task 3 (Extention\_executeGreek.py).

* Initially the model from Task 2 didn’t produce a good result (only 3/6 letters predicted correctly), hence changed few things:
  + Changed the second max pool layer’s size to 4.
  + Changed the number hidden nodes on Linear Layer 1 to 20
  + Up the number of epochs from 150 to 250
* Final result turned out to be 6/6 correct:
* Graphical user interface, application

  Description automatically generated

1. There are many pre-trained networks available in the PyTorch package. Try loading one and evaluate its first couple of convolutional layers as in task 2 (Extension\_analyze.py).

* Imported resnet50, resnet101, resnet152, resnet18, resnet34, alexnet model from torchvision.models
* Evaluated their conv1 layer similar to Task 2, following is the filter analysis from resnet 50:

A screenshot of a computer

Description automatically generated with low confidence

**Reflection:**

Through this project, I learned a lot about how Convolutional Neural Network (CNN) works behind the scenes. At first, I was struggling to understand the specific image size and channel changes throughout the process. Eventually after watching some lecture videos and online PyTorch tutorials, I understood the basic mathematical operations behind the implementation. I learned how to create the CNN from scratch, adding each layers to the neural network as a whole. Through this project, I also got to experiment how different parameters can enhance the overall accuracy of the model. With changing few parameters, I was able to train the model to accurately predict more complicated test cases. In the extension part of project, I also took a peek at other pretrained models (namely Resnet and AlexNet) and how their layers work. I was immensely fascinated by the sophisticated complicated layers they possess. Overall, this project is vey useful in helping me understand the steps behind the model training and possible ways to optimize training results without having to do extra training epochs.

**Acknowledgements of any material and people consulted:**

Did not work with a partner, 100% individual.

Lecture notes

Consulted with TA Guanang and David.