Report

for

COMP 3314 Assignment 2

Prepared by Shunhua Guo, 3035447635

Convolutional Neural Networks

A convolutional neural network for the task of digit recognition using the PyTorch framework: its implementation,

May 9, 2021

1. Overview

This report contains a discussion on CNN design, implementation, and the analysis of parameters and the net structure. The ML problem is this 10-category classification problem which classifies digit images into 0-9.

2. Implementations

Library requirement

The algorithms are implemented in Python, using the PyTorch framework.
Torch and Torchvision packages are required. Install the libraries with

```
pip install torch==1.2.0 torchvision==0.4.0
```

File structure

The implementation is in 3 files located in the root directory:

1. model.py defines the net structure
2. main.py defines the parameters and the train-test procedure.
3. utils.py contains help functions, like loading the dataset and storing the loss and accuracy result.
4. result visualization (Jupiter notebook) provides visualization for loss and accuracy results on the trained model or training process.

NN Structure

As required, my CNN structure contains convolutional layers, fully connected layers, activation layers, and max-pooling layers.

I am using the following as my CNN structure. The reason I am choosing this one will be explained in the *Result discussion on Model* section.

Conv - Conv - ReLU - MaxPool
Conv - Conv - ReLU - MaxPool
FC - ReLU - Dropout - FC - ReLU - FC

The network depth is less than 20.

Train-test process

This part follows the standard training process of Neural Networks with additional functions embedded. The training will loop by the number of epochs, and for each loop, it will go through a forward and a backward process with the help from the *optimizer* and *scheduler* that we pre-define.

I added the in-training evaluation function on the test set for further over-fitting or under-fitting analysis of the epochs' number. Also, a storing option can be turned on to store the loss and accuracy result and the final model.

The detailed code guide can be found in the README.md file following the link.

Training Model Statistics

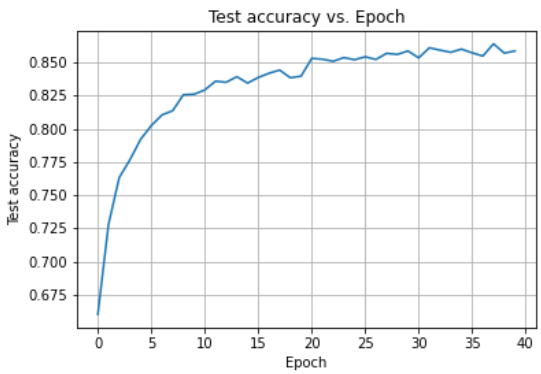
Computer Configure

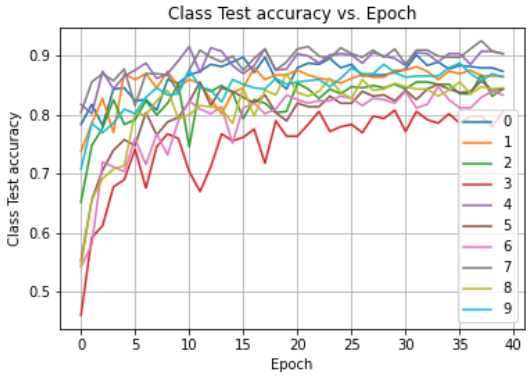
型号名称:	MacBook Pro
处理器名称:	4-core Intel Core i7
处理器速度:	2.3 GHz
处理器数目:	1
核总数:	4
L2缓存（每个核）:	512 KB
L3缓存:	8 MB
超线程技术:	已启用
内存:	16 GB
系统固件版本:	1554.80.3.0.0 (iBridge: 18.16.14347.0.0,0)

Running time of one training process

I timed the training with each epoch, it averaged around 92 seconds for each epoch, and my training process is designed to 25 epochs. Therefore, it needs around 38 minutes to finish one training process, with in-training evaluation turned off.

The best model trained reached around 86% test accuracy. To restore this model, run `model = restore_model('best')` in main.py.





An interesting side discovery is that class of number ‘3’ has the worst classification performance across nearly all models I trained.

3. Result Discussion on Model

Dataset – class imbalance

As instruction stated, the training set contains 3,000 samples for each class, and the testing set contains 500 samples for each class. Each input image has three channels, and each channel contains 32*32 pixels. Since the training and test dataset is distributed equally over each class, there will not be a class imbalance problem.

With and without ReLU activation layer comparison

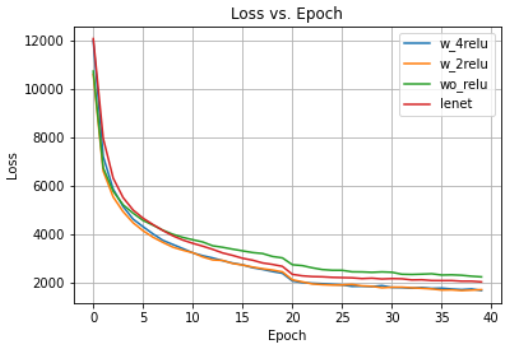
The construction inspiration of this CNN model is from the lecture. A combination of convolutional layers, activation layers, and pooling layer, as well as the fully connected layers, consist of my net structure. Also, I added a Dropout layer between fully connected layers to avoid the overfitting problem.

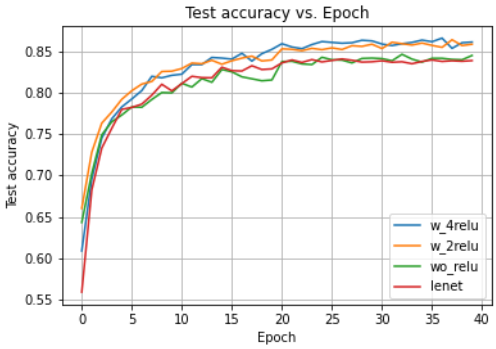
As previously mentioned, the final structure is as follows. But, before I finalized this model, I tried several net structures and compared their performance.

Conv - Conv - ReLU - MaxPool
Conv - Conv - ReLU - MaxPool
FC - ReLU - Dropout - FC - ReLU - FC

The comparison I would like to highlight is among this one and the other models, which are:

1. without any ReLU layer between Convolutional layers, called *wo_relu.*
2. With 4 ReLU layers between Convolutional layers, called *w_4relu.*
3. with 2 ReLU layers between Convolutional layers, called *w_2relu.* (the one I pick)
4. (additional) LeNet structure, called *lenet*.





Plotting details can refer to this Jupyter Notebook.

The loss curve is generated by recording training loss from each epoch. From the loss curve, we can see, there is not much difference between the model with 4 ReLU layers and the model with 2 ReLU layers. And, compared to the model without the ReLU layer, other models converge faster and to a lower loss level.

The test accuracy curve is generated by evaluating on test dataset after each epoch. We can access the overfitting problem in this graph. We can see that model with ReLU layers outperform the model without the ReLU layer and LeNet. The convergence speed is similar among those 4. Additionally, those 4 nets have no overfitting problem so far, as the test accuracy does not decrease again.

Since the model with 2 ReLU layers have similarly outstanding performance and has a simpler structure, therefore I chose this one for my model.

Moreover, as we can see from the graph, the test accuracy of those 4 models converges around the 20th epoch, then I used 25 as my number of training epochs parameter in the following discussion.

4. Parameter Analysis

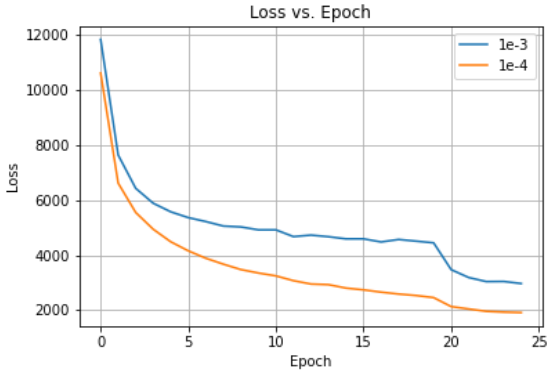
The analysis will be conducted on adjusting hyper-parameters such as the initial learning rate, decay strategy (step size and gamma), and the number of training epochs.

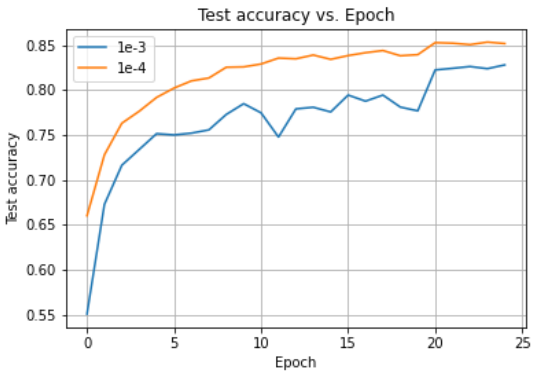
Number of training epochs

The discussion is covered in the previous section that the model converges after around 20-25 epochs.

The initial learning rates

I experimented with the learning rate equal to 0.0001 and 0.001. Obviously, the model with the learning rate equal to 0.0001 performs better.

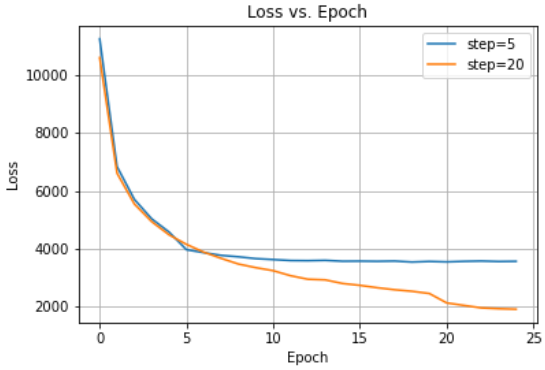


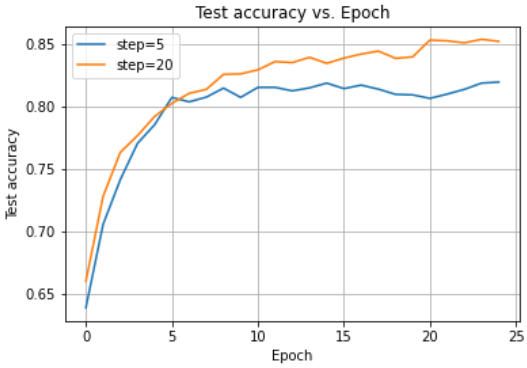


Decay strategy (schedule, step size and gamma)

Step size (20 vs. 5)

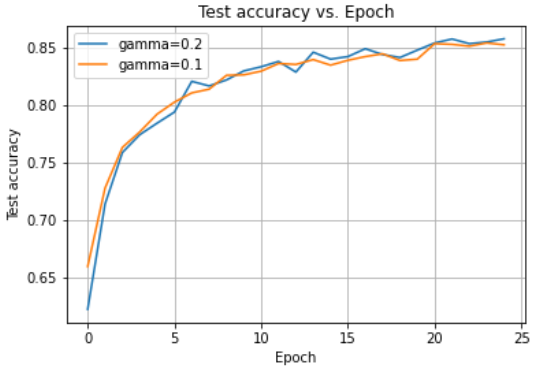
I experimented with the step size equal to 20 and 5. Obviously, the test accuracy and loss curve performs better with step size equal to 20.

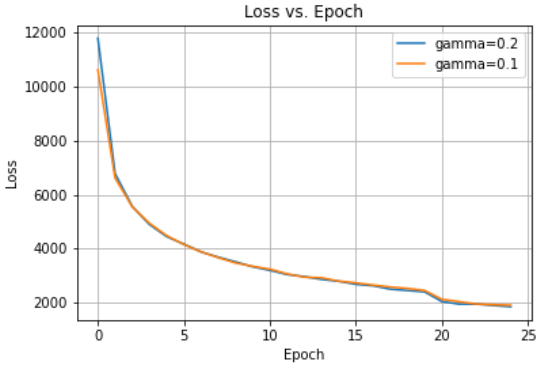




Gamma (0.1 vs. 0.2)

I experimented with the gamma equal to 0.1 and 0.2, the resulted plotting is as follows.





The gamma change seems to have no effect on the existing model with the given parameters.