Report1 CNNs for Handwritten Digit Classification

24011149X SONG Zhengfei

1. Introduction to CNN

Convolutional Neural Network (CNN) is a feedforward neural network inspired by the natural visual perception mechanism of organisms. The structure of CNN mainly includes convolutional layers, pooling layers and fully connected layers. After the image is input, a convolution operation is performed. Each convolutional filter can be seen as a feature extractor, generating a feature map in the next layer. Subsequently, the pooling operation, also known as down-sampling, is used to reduce the amount of data, and pooling can make the network reduce dependence on the position of the object in the image. After several layers of convolution and pooling operations, the obtained feature maps are expanded row by row, connected into vectors, and input into the fully connected network to make the final prediction.

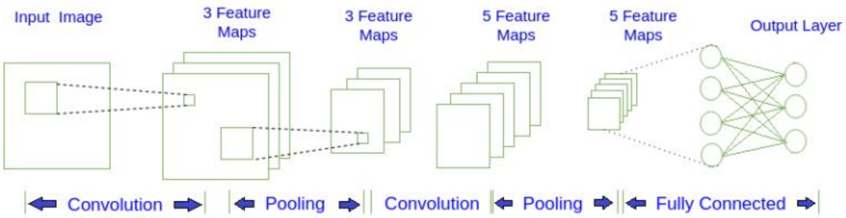


Figure 1 Schematic diagram of CNN structure

CNN performs well in many tasks, including image classification, object detection and semantic segmentation. Among them, handwritten digit recognition is one of the most classic applications of CNN. Lab1 helps us to explore the application of CNN in this area and explore the impact of network or setting changes on train and test results.

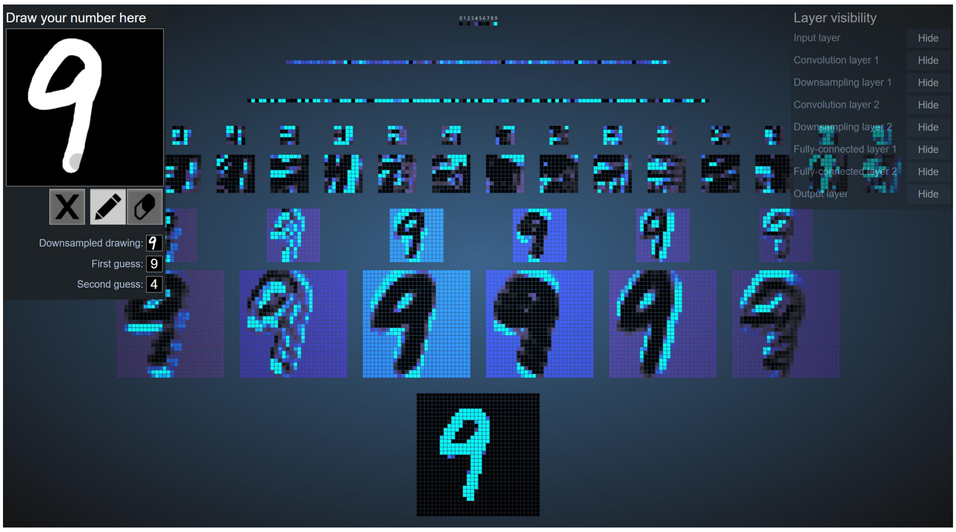


Figure 2 Handwritten Digit Classification[1]

2. Experiment on Handwritten Digit Classification

1) CNN implemented in TensorFlow (Keras)

TensorFlow is Google's open source machine learning framework based on data flow graphs. Keras is a deep learning library based on TensorFlow and Theano.

1 https://adamharley.com/nn_vis/

We explored the use of Keras to develop CNN handwritten digit classification at first.

I trained the model for 5 epochs and got the result shown in Figure 3. As the number of training rounds increases, the accuracy on the train set continues to improve and the loss continues to decrease. The model achieved a loss of 0.0849 and an accuracy of 97.35% on the unseen test set. However, as can be seen from Figure 3b, the accuracy and loss of the model have not converged, which means that the performance of the model can still be improved.

```
 Total params: 278,858 (1.06 MB)
 Trainable params: 277,834 (1.06 MB)
 Non-trainable params: 1,024 (4.00 KB)
Epoch 1/5
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1727958035.741825   24220 service.cc:146] XLA service 0x7bfae4002570 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1727958035.741872   24220 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1727958040.778135   24220 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 15ms/step - accuracy: 0.3941 - loss: 2.0414
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8041 - loss: 0.6105
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8765 - loss: 0.4082
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9030 - loss: 0.3244
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9213 - loss: 0.2719
Saving CNN to models/mnist_cnn.keras
```

Figure 3a Accuracy and loss on train set

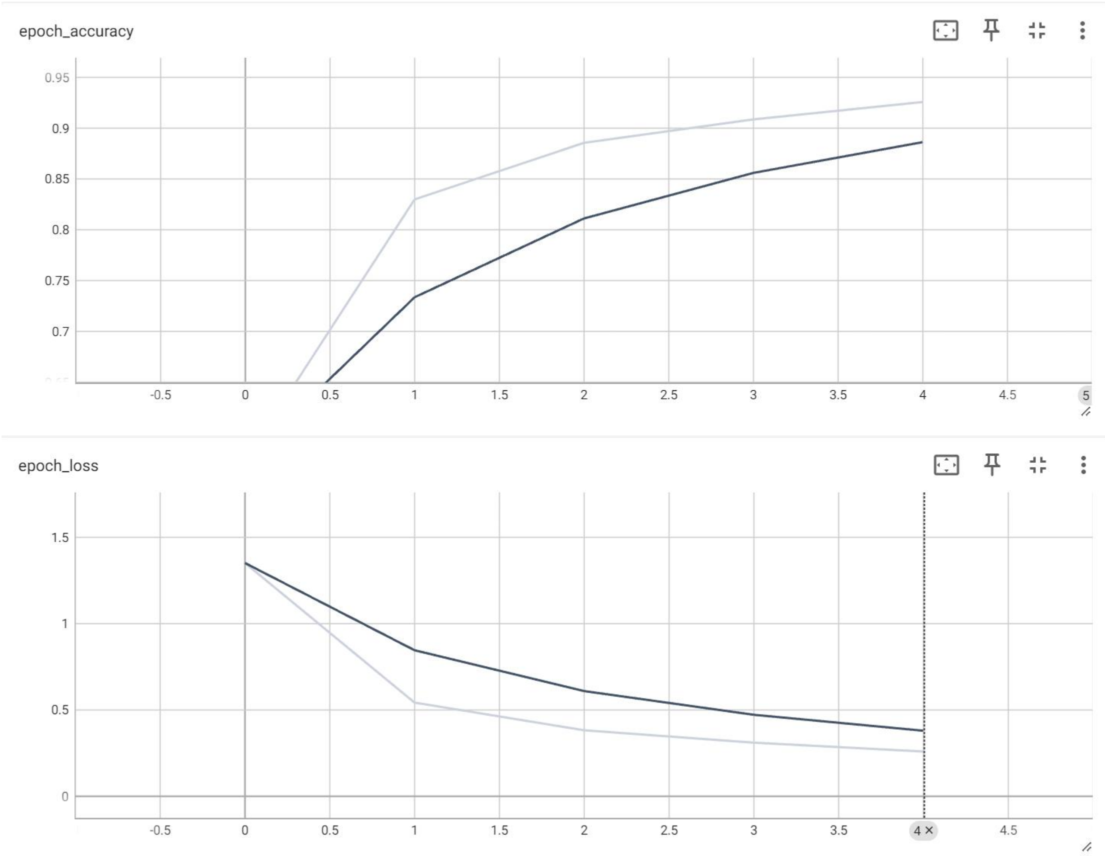


Figure 3b Accuracy and loss plots

```
 Total params: 278,858 (1.06 MB)
 Trainable params: 277,834 (1.06 MB)
 Non-trainable params: 1,024 (4.00 KB)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1727959181.679266   29459 service.cc:146] XLA service 0x7803480067e0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1727959181.679317   29459 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1727959182.646245   29459 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
Test loss: 0.08490477502346039
Test accuracy: 97.35%
```

Figure 3c Accuracy and loss on test set

Then, I increased the training epochs to 20 to observe the changes in model performance.

```
Epoch 1/20
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1727959509.062647   31029 service.cc:146] XLA service 0x7f8e1400e2c0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1727959509.062721   31029 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1727959513.399604   31029 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 17ms/step - accuracy: 0.4041 - loss: 1.9461
Epoch 2/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8173 - loss: 0.5815
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.8806 - loss: 0.3931
Epoch 4/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9086 - loss: 0.3159
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9238 - loss: 0.2640
Epoch 6/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9329 - loss: 0.2379
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9400 - loss: 0.2100
Epoch 8/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9425 - loss: 0.1969
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9502 - loss: 0.1808
Epoch 10/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9535 - loss: 0.1665
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9571 - loss: 0.1582
Epoch 12/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9590 - loss: 0.1458
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9612 - loss: 0.1414
Epoch 14/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9621 - loss: 0.1396
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9637 - loss: 0.1308
Epoch 16/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9636 - loss: 0.1319
Epoch 17/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9653 - loss: 0.1245
Epoch 18/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9669 - loss: 0.1229
Epoch 19/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9676 - loss: 0.1178
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9683 - loss: 0.1135
Saving CNN to models/mnist_cnn_epoch20.keras
```

Figure 4a Accuracy and loss on train set

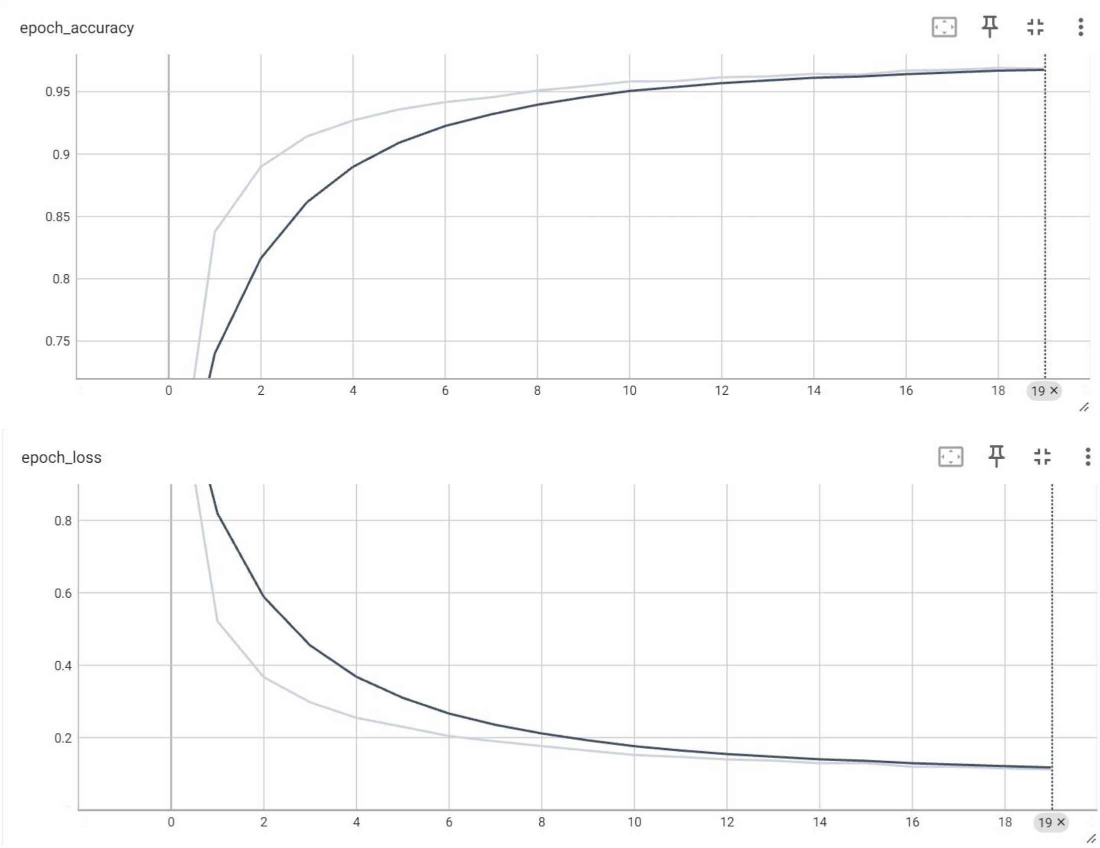


Figure 4b Accuracy and loss plots

```
 Total params: 278,858 (1.06 MB)
 Trainable params: 277,834 (1.06 MB)
 Non-trainable params: 1,024 (4.00 KB)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1727959909.405262   32916 service.cc:146] XLA service 0x78e8ac006d00 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1727959909.405304   32916 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1727959910.459621   32916 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
Test loss: 0.04038699343800545
Test accuracy: 98.69%
```

Figure 4c Accuracy and loss on test set

After training for 20 epochs, the model achieves higher training accuracy and lower training loss. At the same time, it can be seen from the curves that with the increase of training epochs, the increase in accuracy and the decrease in loss are reduced, and the curves of both accuracy and loss tend to converge to a value. Also, the accuracy on the test set is 98.69%, higher than the previous one, which means the model performs better.

```python
# CNN structure definition
model = Sequential()
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
```

Figure 5 Layers change

I also added some layers to change the network structure, and then trained it for 20 epochs. As shown in Figure 6, the training accuracy is lower than the original model, and the testing accuracy achieves 98.23%, which indicates that the model can achieve good digit recognition and classification results, but is not as good as the original model. The decrease in accuracy after adding an additional convolutional layer may be due to several factors, including overfitting, as the model becomes more complex and learns noise rather than general features. Increased parameters can make training more difficult, especially with limited data. Additionally, the learning rate may need adjustment, and the placement or effectiveness of batch normalization might be affected. Lastly, the dropout rate could hinder learning if set too high.

```
Epoch 1/20
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1728616263.778758    6084 service.cc:146] XLA service 0x79d738010550 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1728616263.778815    6084 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1728616270.674358    6084 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 20ms/step - accuracy: 0.2568 - loss: 2.5356
Epoch 2/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.6428 - loss: 1.0640
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.7772 - loss: 0.6962
Epoch 4/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8447 - loss: 0.5193
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8741 - loss: 0.4267
Epoch 6/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8933 - loss: 0.3685
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9088 - loss: 0.3214
Epoch 8/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9160 - loss: 0.2928
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9245 - loss: 0.2666
Epoch 10/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9313 - loss: 0.2480
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9347 - loss: 0.2355
Epoch 12/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9394 - loss: 0.2235
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9426 - loss: 0.2110
Epoch 14/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9460 - loss: 0.2013
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9489 - loss: 0.1913
Epoch 16/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9508 - loss: 0.1812
Epoch 17/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9511 - loss: 0.1804
Epoch 18/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9508 - loss: 0.1769
Epoch 19/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9554 - loss: 0.1626
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9559 - loss: 0.1686
Saving CNN to models/mnist_cnn_epoch20.keras
```

Figure 6a Accuracy and loss on train set

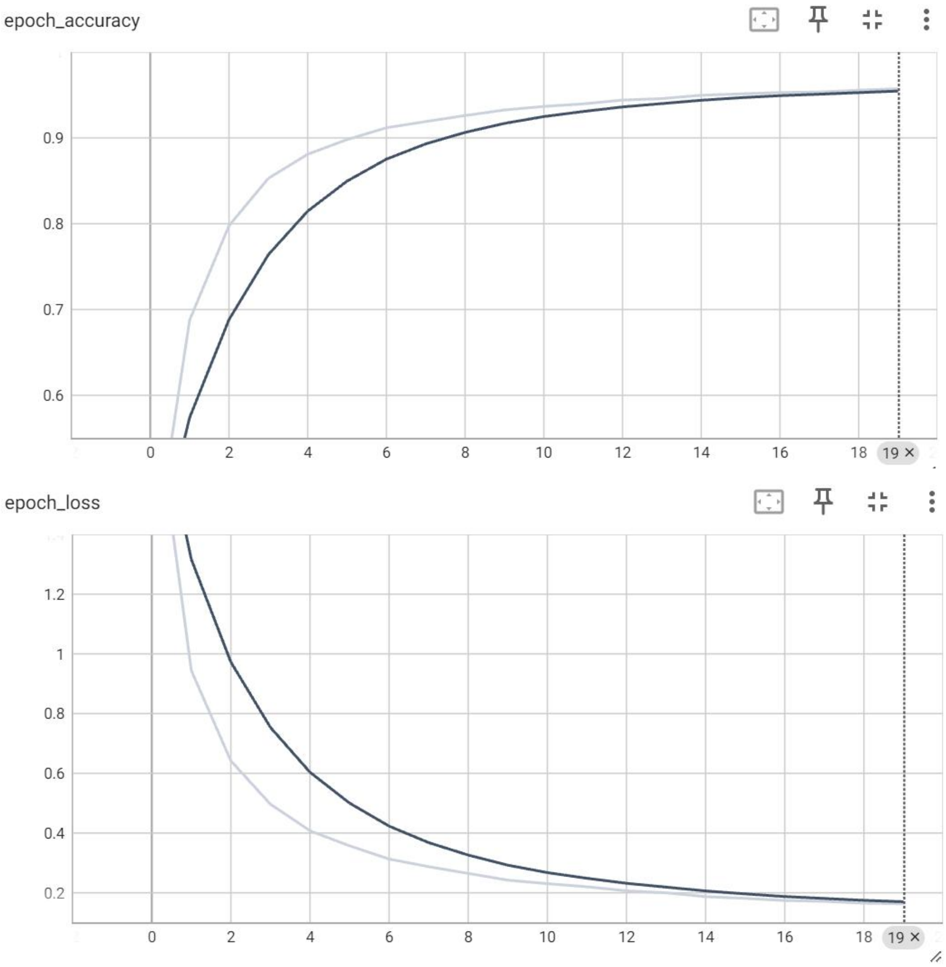


Figure 6b Accuracy and loss plots

```
 Total params: 119,434 (466.54 KB)
 Trainable params: 118,282 (462.04 KB)
 Non-trainable params: 1,152 (4.50 KB)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1728616630.832580    7894 service.cc:146] XLA service 0x7bf21c00d330 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1728616630.832645    7894 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1728616632.243729    7894 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
Test loss: 0.0646471455693245
Test accuracy: 98.23%
```

Figure 6c Accuracy and loss on test set

The activation function layer, also called the non-linearity mapping layer, is used to increase the non-linearity of the entire network. I changed the activation function from ReLU to Leaky ReLU based on the original model structure. The Leaky ReLU can address the ‘dying ReLU’ problem, where neurons can become inactive and stop learning. Leaky ReLU allows a small, non-zero gradient when the unit is not active, which can help with learning.

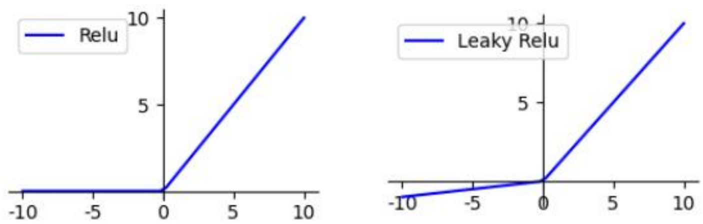


Figure 7 ReLU and Leaky ReLU

```
Epoch 1/5
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1728617673.655205   12718 service.cc:146] XLA service 0x7a198400cd90 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1728617673.655256   12718 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1728617677.917454   12718 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 16ms/step - accuracy: 0.4282 - loss: 1.8772
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.8302 - loss: 0.5413
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8855 - loss: 0.3853
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9095 - loss: 0.3051
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9235 - loss: 0.2609
Saving CNN to models/mnist_cnn_leakyReLU_epoch5.keras
```

Figure 8a Accuracy and loss on train set

```
 Total params: 278,858 (1.06 MB)
 Trainable params: 277,834 (1.06 MB)
 Non-trainable params: 1,024 (4.00 KB)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1728617847.581156   13572 service.cc:146] XLA service 0x7ea5c4005b30 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1728617847.581208   13572 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1728617848.609421   13572 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
Test loss: 0.08441925793886185
Test accuracy: 97.36%
```

Figure 8b Accuracy and loss on test set

After 5-epoch training, the model's testing accuracy (shown in Figure 8) achieves 97.36%, higher than the original one, which indicates that Leaky ReLU improved the model's training and test performance. But when it is trained for 20 epochs, its testing accuracy is 98.64% (shown in Figure 9), which is a little bit worse than the original setting.

Leaky ReLU can improve gradient flow, especially in the early stages of training, leading to faster convergence initially. With more epochs, the model might start overfitting to the training data. Leaky ReLU can sometimes exacerbate this if it allows the model to fit noise. Also, Leaky ReLU might increase the effective complexity of the model, which can be beneficial initially but harmful if not regularized properly. As a result, in the early epochs, its improvement effect is more obvious, but as training epochs increases, it is not as good as the original activation function ReLU.

```
Epoch 1/20
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1728617368.988184   11145 service.cc:146] XLA service 0x7b5ee000d8d0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1728617368.988237   11145 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1728617373.367054   11145 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 16ms/step - accuracy: 0.4245 - loss: 1.9439
Epoch 2/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8245 - loss: 0.5552
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8848 - loss: 0.3811
Epoch 4/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9110 - loss: 0.3026
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9238 - loss: 0.2575
Epoch 6/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9344 - loss: 0.2279
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9389 - loss: 0.2103
Epoch 8/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9450 - loss: 0.1909
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9483 - loss: 0.1789
Epoch 10/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9513 - loss: 0.1664
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 8ms/step - accuracy: 0.9557 - loss: 0.1561
Epoch 12/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9564 - loss: 0.1516
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9587 - loss: 0.1477
Epoch 14/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9601 - loss: 0.1403
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9625 - loss: 0.1271
Epoch 16/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9635 - loss: 0.1261
Epoch 17/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9652 - loss: 0.1233
Epoch 18/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9631 - loss: 0.1247
Epoch 19/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9675 - loss: 0.1153
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9672 - loss: 0.1155
Saving CNN to models/mnist_cnn_leakyReLU_epoch20.keras
```

Figure 9a Accuracy and loss on train set

```
 Total params: 278,858 (1.06 MB)
 Trainable params: 277,834 (1.06 MB)
 Non-trainable params: 1,024 (4.00 KB)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1728617525.397676   12004 service.cc:146] XLA service 0x7b6db8006e20 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1728617525.397729   12004 service.cc:154]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1728617526.360102   12004 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
Test loss: 0.04109155759215355
Test accuracy: 98.64%
```

Figure 9b Accuracy and loss on test set

2) CNN implemented in PyTorch

Then, we used pytorch to develop CNN for handwritten digit classification. Pytorch is the python version of torch. It is an open source neural network framework developed by Facebook and is specifically designed for GPU-accelerated deep neural network (DNN) programming.

Like for the Keras model, I first trained the model for 5 epochs. Its epoch_accuracy increases and epoch_loss decreases gradually with the increasing training epochs. It achieves 96.53% accuracy and 0.1062 loss on the test set. From the curve in Figure 10, we can predict that the model has not reached the optimal state, which means that the model can still be improved as the number of training epochs increases.

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.338448
Train Epoch: 1 [12800/60000 (21%)]	Loss: 1.858618
Train Epoch: 1 [25600/60000 (43%)]	Loss: 1.067451
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.755073
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.637283
Train Epoch: 2 [0/60000 (0%)]	Loss: 0.623550
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.406553
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.489201
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.503142
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.509695
Train Epoch: 3 [0/60000 (0%)]	Loss: 0.297666
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.378906
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.318437
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.464996
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.461571
Train Epoch: 4 [0/60000 (0%)]	Loss: 0.445092
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.381855
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.300320
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.205480
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.345541
Train Epoch: 5 [0/60000 (0%)]	Loss: 0.302741
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.290227
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.293388
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.513905
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.338972
Saving CNN to models/mnist_cnn_epoch5.pth
```

Figure 10a Loss on train set

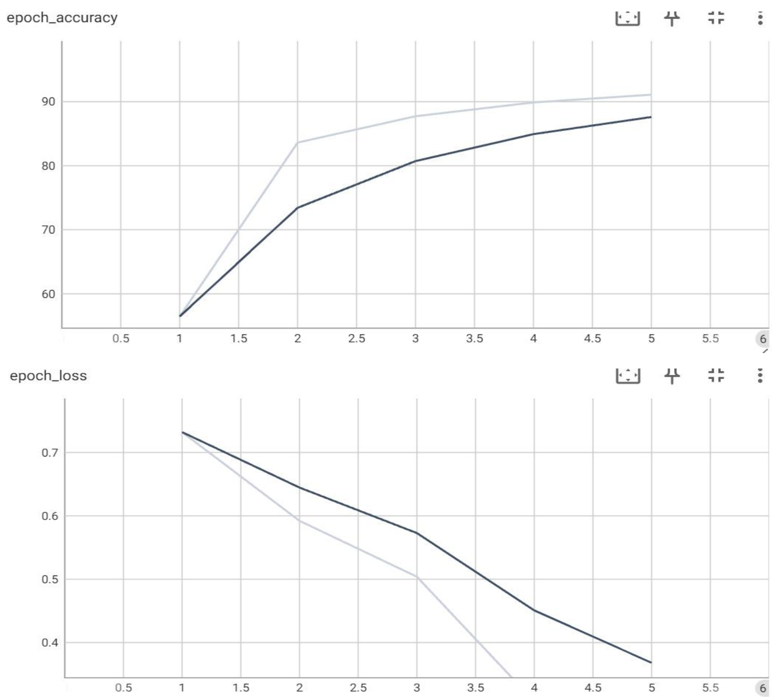


Figure 10b Accuracy and loss plots

```
!python mnist_cnn_pytorch.py --mode 'test' --model_path 'models/mnist_cnn_epoch5.pth'

Test set: Average loss: 0.1062, Accuracy: 9653/10000 (97%)
```

Figure 10c Accuracy and loss on test set

I increased the training epochs to 20. As can be seen from Figure 11, as the number of training epochs increases, the accuracy and loss of the model on the train set tend to converge. And the model achieved 98.42% accuracy and 0.0509 loss value on the train set. As the number of training epochs increases, the model can learn more from the input images in the train set, thus continuously improving its performance both on the train set and on unseen test sets.

```
Train Epoch: 14 [0/60000 (0%)]  Loss: 0.134377
Train Epoch: 14 [12800/60000 (21%)]     Loss: 0.168794
Train Epoch: 14 [25600/60000 (43%)]     Loss: 0.213798
Train Epoch: 14 [38400/60000 (64%)]     Loss: 0.119662
Train Epoch: 14 [51200/60000 (85%)]     Loss: 0.069178
Train Epoch: 15 [0/60000 (0%)]  Loss: 0.238926
Train Epoch: 15 [12800/60000 (21%)]     Loss: 0.236162
Train Epoch: 15 [25600/60000 (43%)]     Loss: 0.260716
Train Epoch: 15 [38400/60000 (64%)]     Loss: 0.222964
Train Epoch: 15 [51200/60000 (85%)]     Loss: 0.143555
Train Epoch: 16 [0/60000 (0%)]  Loss: 0.137080
Train Epoch: 16 [12800/60000 (21%)]     Loss: 0.192966
Train Epoch: 16 [25600/60000 (43%)]     Loss: 0.164472
Train Epoch: 16 [38400/60000 (64%)]     Loss: 0.163804
Train Epoch: 16 [51200/60000 (85%)]     Loss: 0.108393
Train Epoch: 17 [0/60000 (0%)]  Loss: 0.222496
Train Epoch: 17 [12800/60000 (21%)]     Loss: 0.099980
Train Epoch: 17 [25600/60000 (43%)]     Loss: 0.205496
Train Epoch: 17 [38400/60000 (64%)]     Loss: 0.138827
Train Epoch: 17 [51200/60000 (85%)]     Loss: 0.107947
Train Epoch: 18 [0/60000 (0%)]  Loss: 0.086404
Train Epoch: 18 [12800/60000 (21%)]     Loss: 0.107281
Train Epoch: 18 [25600/60000 (43%)]     Loss: 0.216492
Train Epoch: 18 [38400/60000 (64%)]     Loss: 0.186921
Train Epoch: 18 [51200/60000 (85%)]     Loss: 0.238535
Train Epoch: 19 [0/60000 (0%)]  Loss: 0.087038
Train Epoch: 19 [12800/60000 (21%)]     Loss: 0.152965
Train Epoch: 19 [25600/60000 (43%)]     Loss: 0.140084
Train Epoch: 19 [38400/60000 (64%)]     Loss: 0.111310
Train Epoch: 19 [51200/60000 (85%)]     Loss: 0.269570
Train Epoch: 20 [0/60000 (0%)]  Loss: 0.130322
Train Epoch: 20 [12800/60000 (21%)]     Loss: 0.134785
Train Epoch: 20 [25600/60000 (43%)]     Loss: 0.171895
Train Epoch: 20 [38400/60000 (64%)]     Loss: 0.144523
Train Epoch: 20 [51200/60000 (85%)]     Loss: 0.104554
Saving CNN to models/mnist_cnn_epoch20.pth
```

Figure 11a Loss on train set

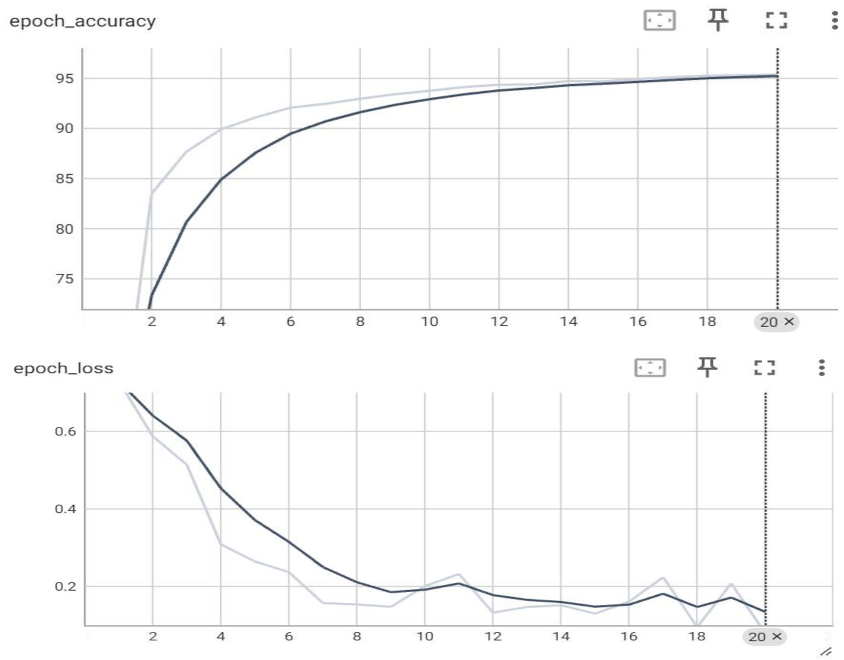


Figure 11b Accuracy and loss plots

```
!python mnist_cnn_pytorch.py --mode 'test' --model_path 'models/mnist_cnn_epoch20.pth'

Test set: Average loss: 0.0509, Accuracy: 9842/10000 (98%)
```

Figure 11c Accuracy and loss on test set

Later, I changed the activation function from ReLU to Leaky ReLU based on the original model structure. When training epoch number was 5, the model achieved average loss of 0.1065 and accuracy of 96.60% (shown in Figure 12) on the test set. When training epoch number was 20, the model achieved average loss of 0.0522 and accuracy of 98.36% (shown in Figure 13)on the test set.

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.338432
Train Epoch: 1 [12800/60000 (21%)]	Loss: 1.757658
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.966909
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.713497
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.619005
Train Epoch: 2 [0/60000 (0%)]	Loss: 0.569856
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.372910
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.472952
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.516967
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.479859
Train Epoch: 3 [0/60000 (0%)]	Loss: 0.297335
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.368275
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.315770
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.408523
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.419574
Train Epoch: 4 [0/60000 (0%)]	Loss: 0.326508
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.320240
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.337772
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.179650
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.287390
Train Epoch: 5 [0/60000 (0%)]	Loss: 0.298539
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.258681
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.240962
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.437587
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.317715
Saving CNN to models/mnist_activation_leakyrelu_epoch5.pth
```

Figure 12a Loss on train set

```
!python mnist_cnn_pytorch.py --mode 'test' --model_path 'models/mnist_activation_leakyrelu_epoch5.pth'

Test set: Average loss: 0.1065, Accuracy: 9660/10000 (97%)
```

Figure 12b Accuracy and loss on test set

```
Train Epoch: 14 [0/60000 (0%)]	Loss: 0.138369
Train Epoch: 14 [12800/60000 (21%)]	Loss: 0.137405
Train Epoch: 14 [25600/60000 (43%)]	Loss: 0.217095
Train Epoch: 14 [38400/60000 (64%)]	Loss: 0.095038
Train Epoch: 14 [51200/60000 (85%)]	Loss: 0.053778
Train Epoch: 15 [0/60000 (0%)]	Loss: 0.188373
Train Epoch: 15 [12800/60000 (21%)]	Loss: 0.209345
Train Epoch: 15 [25600/60000 (43%)]	Loss: 0.229429
Train Epoch: 15 [38400/60000 (64%)]	Loss: 0.156578
Train Epoch: 15 [51200/60000 (85%)]	Loss: 0.141650
Train Epoch: 16 [0/60000 (0%)]	Loss: 0.155571
Train Epoch: 16 [12800/60000 (21%)]	Loss: 0.125229
Train Epoch: 16 [25600/60000 (43%)]	Loss: 0.145554
Train Epoch: 16 [38400/60000 (64%)]	Loss: 0.097808
Train Epoch: 16 [51200/60000 (85%)]	Loss: 0.098421
Train Epoch: 17 [0/60000 (0%)]	Loss: 0.245400
Train Epoch: 17 [12800/60000 (21%)]	Loss: 0.108508
Train Epoch: 17 [25600/60000 (43%)]	Loss: 0.176686
Train Epoch: 17 [38400/60000 (64%)]	Loss: 0.118330
Train Epoch: 17 [51200/60000 (85%)]	Loss: 0.119917
Train Epoch: 18 [0/60000 (0%)]	Loss: 0.083063
Train Epoch: 18 [12800/60000 (21%)]	Loss: 0.080352
Train Epoch: 18 [25600/60000 (43%)]	Loss: 0.187465
Train Epoch: 18 [38400/60000 (64%)]	Loss: 0.160845
Train Epoch: 18 [51200/60000 (85%)]	Loss: 0.193339
Train Epoch: 19 [0/60000 (0%)]	Loss: 0.103118
Train Epoch: 19 [12800/60000 (21%)]	Loss: 0.150730
Train Epoch: 19 [25600/60000 (43%)]	Loss: 0.122162
Train Epoch: 19 [38400/60000 (64%)]	Loss: 0.117230
Train Epoch: 19 [51200/60000 (85%)]	Loss: 0.241442
Train Epoch: 20 [0/60000 (0%)]	Loss: 0.083570
Train Epoch: 20 [12800/60000 (21%)]	Loss: 0.122653
Train Epoch: 20 [25600/60000 (43%)]	Loss: 0.094877
Train Epoch: 20 [38400/60000 (64%)]	Loss: 0.136092
Train Epoch: 20 [51200/60000 (85%)]	Loss: 0.102913
Saving CNN to models/mnist_activation_leakyrelu_epoch20.pth
```

Figure 13a Loss on train set

```
!python mnist_cnn_pytorch.py --mode 'test' --model_path 'models/mnist_activation_leakyrelu_epoch20.pth'

Test set: Average loss: 0.0522, Accuracy: 9836/10000 (98%)
```

Figure 13b Accuracy and loss on test set

As in the case of the Keras model, when the training epoch is 5, Leaky ReLU can improve the performance of the model. However, when the epoch number increases to 20, the model with the activation function changed to Leaky ReLU is slightly worse than the model with the original activation function as ReLU.

I also changed the network structure to explore the changes of model performance.

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)  # Output channels: 32
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)  # Output channels: 64
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)  # New layer: Output channels: 128

        self.batch_norm1 = nn.BatchNorm2d(32)
        self.batch_norm2 = nn.BatchNorm2d(64)
        self.batch_norm3 = nn.BatchNorm2d(128)

        self.conv2_drop = nn.Dropout2d(p=0.25)

        self.fc1 = nn.Linear(128 * 3 * 3, 256)  # Adjusted input size and increased output neurons
        self.fc2 = nn.Linear(256, 128)  # New fully connected layer
        self.fc3 = nn.Linear(128, 10)  # Final layer for 10 classes

        self.leaky_relu = nn.LeakyReLU(negative_slope=0.1)

    def forward(self, x):
        x = self.leaky_relu(self.batch_norm1(F.max_pool2d(self.conv1(x), 2)))
        # Output size: [32, 14, 14]

        x = self.leaky_relu(self.batch_norm2(F.max_pool2d(self.conv2(x), 2)))
        # Output size: [64, 7, 7]

        x = self.leaky_relu(self.batch_norm3(F.max_pool2d(self.conv3(x), 2)))
        x = self.conv2_drop(x)
        # Output size: [128, 3, 3]

        x = x.view(-1, 128 * 3 * 3)
        # Flatten: [128 * 3 * 3]

        x = self.leaky_relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        # Output size: [256]

        x = self.leaky_relu(self.fc2(x))
        x = F.dropout(x, training=self.training)
        # Output size: [128]

        x = self.fc3(x)
        # Output size: [10]

        return F.log_softmax(x, -1)
```

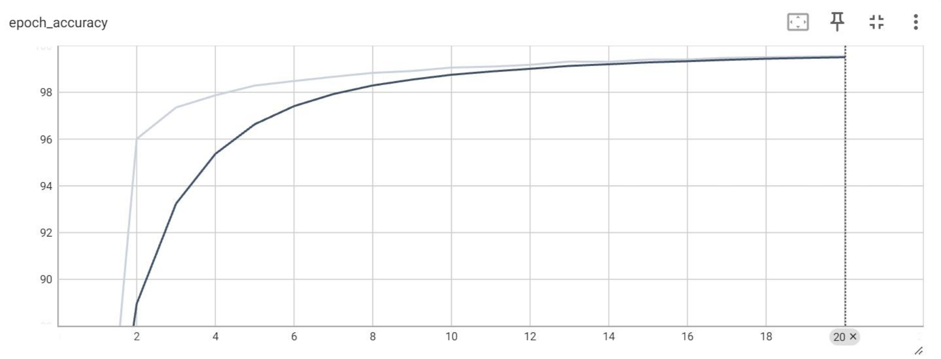
Figure 14 Network structure change

For convolution layers, I increased the output channels of conv1 from 10 to 32, increased the output channels of conv2 from 20 to 64 and added conv3 with 128 output channels to increase the depth of the network. I added batch normalization after each convolution layer. This normalized the output of the layers, improving convergence speed and stability during training. I also increased complexity in Fully Connected Layers, modifying fc1 to have more neurons and adding a new fully connected layer fc2.

As shown in Figure 15, the model achieved average loss of 0.0217 and average accuracy of 99.26% on test set, which outperformed than the initial model with original setting.

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.317928
Train Epoch: 1 [12800/60000 (21%)]	Loss: 1.428763
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.497697
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.423407
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.217909
Train Epoch: 2 [0/60000 (0%)]	Loss: 0.206332
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.301353
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.153258
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.145448
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.158609
Train Epoch: 3 [0/60000 (0%)]	Loss: 0.081703
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.133894
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.102026
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.081126
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.079195
Train Epoch: 4 [0/60000 (0%)]	Loss: 0.052822
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.075174
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.103186
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.098724
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.081792
Train Epoch: 5 [0/60000 (0%)]	Loss: 0.026657
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.040706
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.091907
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.034933
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.020930
Train Epoch: 6 [0/60000 (0%)]	Loss: 0.058314
Train Epoch: 6 [12800/60000 (21%)]	Loss: 0.052500
Train Epoch: 6 [25600/60000 (43%)]	Loss: 0.057745
Train Epoch: 6 [38400/60000 (64%)]	Loss: 0.083018
Train Epoch: 6 [51200/60000 (85%)]	Loss: 0.068276
Train Epoch: 7 [0/60000 (0%)]	Loss: 0.032775
Train Epoch: 7 [12800/60000 (21%)]	Loss: 0.051469
Train Epoch: 7 [25600/60000 (43%)]	Loss: 0.042848
Train Epoch: 7 [38400/60000 (64%)]	Loss: 0.041638
Train Epoch: 7 [51200/60000 (85%)]	Loss: 0.126986
Train Epoch: 8 [0/60000 (0%)]	Loss: 0.024478
Train Epoch: 8 [12800/60000 (21%)]	Loss: 0.035913
Train Epoch: 8 [25600/60000 (43%)]	Loss: 0.046754
Train Epoch: 8 [38400/60000 (64%)]	Loss: 0.061540
Train Epoch: 8 [51200/60000 (85%)]	Loss: 0.090749
Train Epoch: 9 [0/60000 (0%)]	Loss: 0.024209
Train Epoch: 9 [12800/60000 (21%)]	Loss: 0.038254
Train Epoch: 9 [25600/60000 (43%)]	Loss: 0.150601
Train Epoch: 9 [38400/60000 (64%)]	Loss: 0.081489
Train Epoch: 9 [51200/60000 (85%)]	Loss: 0.049549
Train Epoch: 10 [0/60000 (0%)]	Loss: 0.041645
Train Epoch: 10 [12800/60000 (21%)]	Loss: 0.020176
Train Epoch: 10 [25600/60000 (43%)]	Loss: 0.013933
Train Epoch: 10 [38400/60000 (64%)]	Loss: 0.086620
Train Epoch: 10 [51200/60000 (85%)]	Loss: 0.005981
Train Epoch: 11 [0/60000 (0%)]	Loss: 0.044194
Train Epoch: 11 [12800/60000 (21%)]	Loss: 0.017278
Train Epoch: 11 [25600/60000 (43%)]	Loss: 0.014232
Train Epoch: 11 [38400/60000 (64%)]	Loss: 0.038509
Train Epoch: 11 [51200/60000 (85%)]	Loss: 0.017319
Train Epoch: 12 [0/60000 (0%)]	Loss: 0.019007
Train Epoch: 12 [12800/60000 (21%)]	Loss: 0.062559
Train Epoch: 12 [25600/60000 (43%)]	Loss: 0.024487
Train Epoch: 12 [38400/60000 (64%)]	Loss: 0.009158
Train Epoch: 12 [51200/60000 (85%)]	Loss: 0.050040
Train Epoch: 13 [0/60000 (0%)]	Loss: 0.030887
Train Epoch: 13 [12800/60000 (21%)]	Loss: 0.015050
Train Epoch: 13 [25600/60000 (43%)]	Loss: 0.029468
Train Epoch: 13 [38400/60000 (64%)]	Loss: 0.010826
Train Epoch: 13 [51200/60000 (85%)]	Loss: 0.007459
Train Epoch: 14 [0/60000 (0%)]	Loss: 0.025902
Train Epoch: 14 [12800/60000 (21%)]	Loss: 0.006942
Train Epoch: 14 [25600/60000 (43%)]	Loss: 0.007084
Train Epoch: 14 [38400/60000 (64%)]	Loss: 0.035104
Train Epoch: 14 [51200/60000 (85%)]	Loss: 0.021043
Train Epoch: 15 [0/60000 (0%)]	Loss: 0.012573
Train Epoch: 15 [12800/60000 (21%)]	Loss: 0.012417
Train Epoch: 15 [25600/60000 (43%)]	Loss: 0.058063
Train Epoch: 15 [38400/60000 (64%)]	Loss: 0.010134
Train Epoch: 15 [51200/60000 (85%)]	Loss: 0.037344
Train Epoch: 16 [0/60000 (0%)]	Loss: 0.008779
Train Epoch: 16 [12800/60000 (21%)]	Loss: 0.007901
Train Epoch: 16 [25600/60000 (43%)]	Loss: 0.024021
Train Epoch: 16 [38400/60000 (64%)]	Loss: 0.028614
Train Epoch: 16 [51200/60000 (85%)]	Loss: 0.018528
Train Epoch: 17 [0/60000 (0%)]	Loss: 0.019057
Train Epoch: 17 [12800/60000 (21%)]	Loss: 0.029382
Train Epoch: 17 [25600/60000 (43%)]	Loss: 0.004835
Train Epoch: 17 [38400/60000 (64%)]	Loss: 0.010800
Train Epoch: 17 [51200/60000 (85%)]	Loss: 0.026977
Train Epoch: 18 [0/60000 (0%)]	Loss: 0.019179
Train Epoch: 18 [12800/60000 (21%)]	Loss: 0.035081
Train Epoch: 18 [25600/60000 (43%)]	Loss: 0.056589
Train Epoch: 18 [38400/60000 (64%)]	Loss: 0.005937
Train Epoch: 18 [51200/60000 (85%)]	Loss: 0.008548
Train Epoch: 19 [0/60000 (0%)]	Loss: 0.019751
Train Epoch: 19 [12800/60000 (21%)]	Loss: 0.011249
Train Epoch: 19 [25600/60000 (43%)]	Loss: 0.009067
Train Epoch: 19 [38400/60000 (64%)]	Loss: 0.004700
Train Epoch: 19 [51200/60000 (85%)]	Loss: 0.004256
Train Epoch: 20 [0/60000 (0%)]	Loss: 0.007066
Train Epoch: 20 [12800/60000 (21%)]	Loss: 0.002187
Train Epoch: 20 [25600/60000 (43%)]	Loss: 0.003748
Train Epoch: 20 [38400/60000 (64%)]	Loss: 0.032284
Train Epoch: 20 [51200/60000 (85%)]	Loss: 0.022556
Saving CNN to models/mnist_addlayer_epoch20.pth
```

Figure 15a Loss on train set



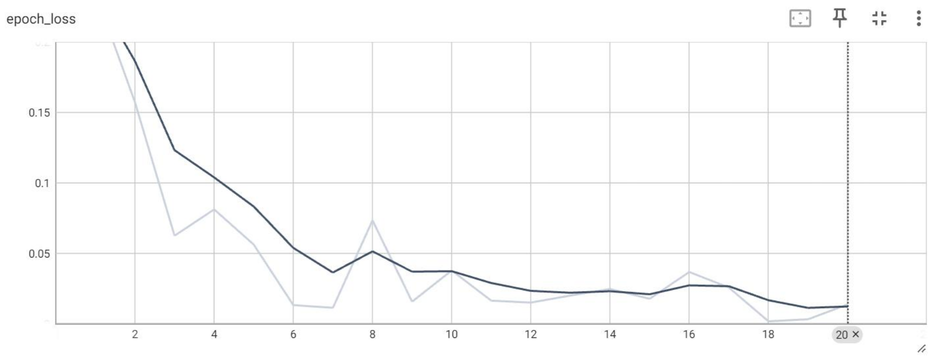


Figure 15b Accuracy and loss plots

```
!python mnist_cnn_pytorch.py --mode 'test' --model_path 'models/mnist_addlayer_epoch20.pth'

Test set: Average loss: 0.0217, Accuracy: 9926/10000 (99%)
```

Figure 15c Accuracy and loss on test set

The detection effect of this model is the best in the experimental attempts.There are several reasons. More channels of convolution layers allow the network to learn a greater number of feature maps, which helps in capturing more detailed patterns from the input data. Adding convolution layer increases the depth of the network, which helps in learning more complex features by adding more non-linear transformations. The increased complexity in FCN improves the capacity of the network to learn more complex relationships. These changes collectively provide the model with more capacity to learn complex patterns while also incorporating mechanisms to prevent overfitting, leading to better performance on the task.

3. Summary

In Lab 1, I gained a deeper understanding of the structure of Convolutional Neural Networks and their application in handwritten digit classification based on Keras and PyTorch. By adjusting the number of training epochs, experimenting with different activation functions, and modifying the model architecture, I was able to enhance the performance of the model.