Image and Video Recognition
Exercise 2

Shen Ruoyue 20M38216

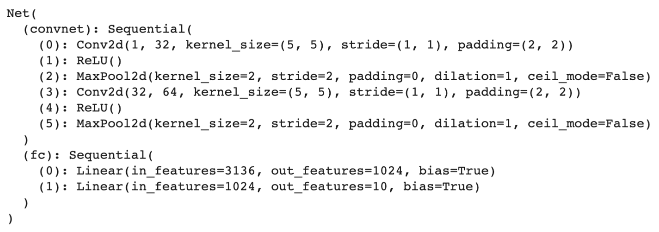
Lecture Date: Dec 7th

1 Exercise 2-1 & 2-2

1.1 Settings

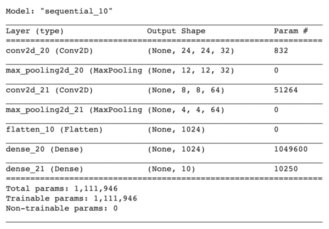
In order to compare the performance using PyTorch and TensorFlow, I use the same network structure and same hyper parameters.

As for the network structure, I build a simple Convolutional Neural Network, which has 2 convolution layers, 2 Max-pooling layers and 2 Full Connected Layers. The detailed structure and network hyper parameters are in Fig. 1.



```
Net(
  (convnet): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=3136, out_features=1024, bias=True)
    (1): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

(a) Pytorch



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_20 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_21 (Dense)             (None, 10)                10250
=================================================================
Total params: 1,111,946
Trainable params: 1,111,946
Non-trainable params: 0
_________________________________________________________________
```

(b) TensorFlow

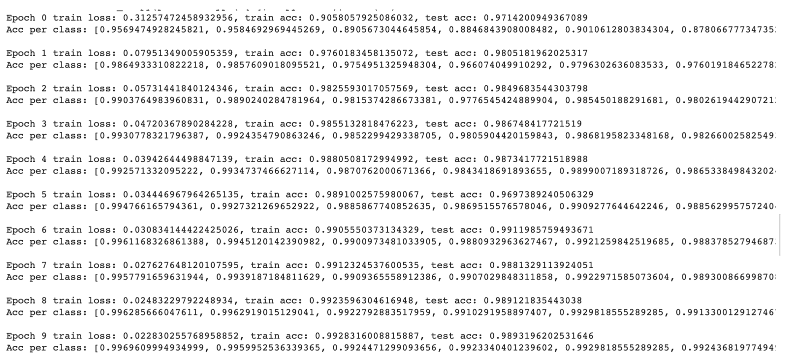
Figure 1: Network Structure

For training, I used the same loss function(Cross Entropy Loss), optimizer(SGD) and hyper parameters(batch size = 128, epochs = 10, learning rate=0.1). The main purpose is to compare the two frameworks, so I didn't tune the hyper parameters to get the best performance, but just use a uniform setting.

Because it's a simple task and network, I only used training set and testing set, which have 60000 train samples and 10000 test samples, rather than split and generate another validation set.

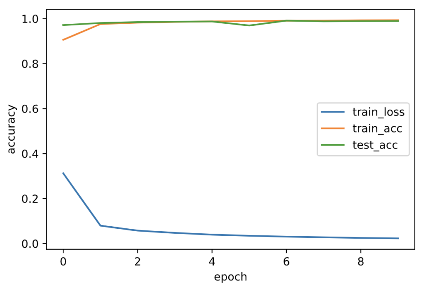
1.2 PyTorch

The loss and accuracy variation during the training process are in Fig. 2. For a clearer view, I plotted it into a change curve as shown in Fig. 3(a). The variation of the accuracy of each class over time is shown in Fig. 4.

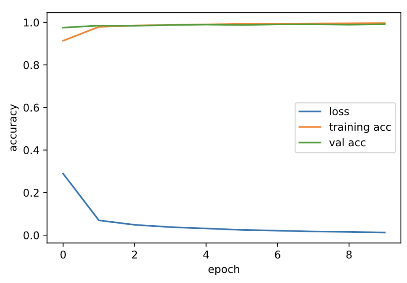


```
Epoch 0 train loss: 0.31257472458932956, train acc: 0.9058057925086032, test acc: 0.9714200949367089
Acc per class: [0.9569474928245821, 0.9584692969445269, 0.8905673044645854, 0.8846843908008482, 0.9010612803834304, 0.87806677734735

Epoch 1 train loss: 0.07951349005905359, train acc: 0.9760183458135072, test acc: 0.9805181962025317
Acc per class: [0.9864933310822218, 0.9857609018095521, 0.9754951325948304, 0.966074049910292, 0.9796302636083533, 0.97601918465227

Epoch 2 train loss: 0.05731441840124346, train acc: 0.9825593017057569, test acc: 0.9849683544303798
Acc per class: [0.9903764983960831, 0.9890240284781964, 0.9815374286673381, 0.9776545424889904, 0.985450188291681, 0.98026194429072

Epoch 3 train loss: 0.04720367890284228, train acc: 0.9855132818476223, test acc: 0.986748417721519
Acc per class: [0.9930778321796387, 0.9924354790863246, 0.9852299429338705, 0.9805904420159843, 0.9868195823348168, 0.98266002582549

Epoch 4 train loss: 0.03942644498847139, train acc: 0.9880508172994992, test acc: 0.9873417721518988
Acc per class: [0.992571332095222, 0.9934737466627114, 0.9870762000671366, 0.9843418691893655, 0.9899007189318726, 0.98653384984320

Epoch 5 train loss: 0.034446967964265135, train acc: 0.9891002575980067, test acc: 0.9697389240506329
Acc per class: [0.994766165794361, 0.9927321269652922, 0.9885867740852635, 0.9869515576578046, 0.9909277644642246, 0.98856299575724

Epoch 6 train loss: 0.030834144422425026, train acc: 0.9905550373134329, test acc: 0.9911985759493671
Acc per class: [0.9961168326861388, 0.9945120142390982, 0.9900973481033905, 0.9880932963627467, 0.9921259842519685, 0.98837852794687

Epoch 7 train loss: 0.027627648120107595, train acc: 0.9912324537600535, test acc: 0.9881329113924051
Acc per class: [0.9957791659631944, 0.9939187184811629, 0.9909365558912386, 0.9907029848311858, 0.9922971585073604, 0.98930086699870

Epoch 8 train loss: 0.02483229792248934, train acc: 0.9923596304616948, test acc: 0.989121835443038
Acc per class: [0.996285666047611, 0.9962919015129041, 0.9922792883517959, 0.9910291958897407, 0.9929818555289285, 0.99133001291274

Epoch 9 train loss: 0.022830255768958852, train acc: 0.9928316008815887, test acc: 0.9893196202531646
Acc per class: [0.9969609994934999, 0.9959952536339365, 0.9924471299093656, 0.9923340401239602, 0.9929818555289285, 0.99243681977494
```

Figure 2: Change curve of Pytorch



(a) Pytorch



(b) TensorFlow

Figure 3: Change curve of loss and accuracy

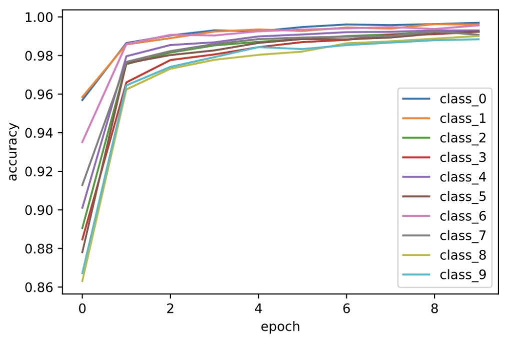
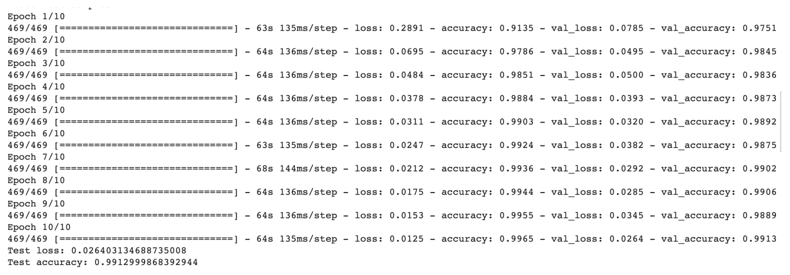


Figure 4: Change curve of accuracy of each class

From figures above, after the 10th epoch, the train accuracy is 0.9928316008815887, test accuracy is 0.9893196202531646, the accuracy for each class is [0.9969609994934999, 0.9959952536339365, 0.9924471299093656, 0.9923340401239602, 0.9929818555289285, 0.9924368197749492, 0.995606623859412, 0.9907422186751795, 0.99008716458725, 0.9884014120020171]. And the final train loss is 0.022830255768958852.

1.3 TensorFlow

The loss and accuracy variation during the training process are in Fig. 5. For a clearer view, I plotted it into a change curve as shown in Fig. 3(b).



```
Epoch 1/10
469/469 [==============================] - 63s 135ms/step - loss: 0.2891 - accuracy: 0.9135 - val_loss: 0.0785 - val_accuracy: 0.9751
Epoch 2/10
469/469 [==============================] - 64s 136ms/step - loss: 0.0695 - accuracy: 0.9786 - val_loss: 0.0495 - val_accuracy: 0.9845
Epoch 3/10
469/469 [==============================] - 64s 136ms/step - loss: 0.0484 - accuracy: 0.9851 - val_loss: 0.0500 - val_accuracy: 0.9836
Epoch 4/10
469/469 [==============================] - 64s 136ms/step - loss: 0.0378 - accuracy: 0.9884 - val_loss: 0.0393 - val_accuracy: 0.9873
Epoch 5/10
469/469 [==============================] - 64s 136ms/step - loss: 0.0311 - accuracy: 0.9903 - val_loss: 0.0320 - val_accuracy: 0.9892
Epoch 6/10
469/469 [==============================] - 63s 135ms/step - loss: 0.0247 - accuracy: 0.9924 - val_loss: 0.0382 - val_accuracy: 0.9875
Epoch 7/10
469/469 [==============================] - 68s 144ms/step - loss: 0.0212 - accuracy: 0.9936 - val_loss: 0.0292 - val_accuracy: 0.9902
Epoch 8/10
469/469 [==============================] - 64s 136ms/step - loss: 0.0175 - accuracy: 0.9944 - val_loss: 0.0285 - val_accuracy: 0.9906
Epoch 9/10
469/469 [==============================] - 64s 136ms/step - loss: 0.0153 - accuracy: 0.9955 - val_loss: 0.0345 - val_accuracy: 0.9889
Epoch 10/10
469/469 [==============================] - 64s 135ms/step - loss: 0.0125 - accuracy: 0.9965 - val_loss: 0.0264 - val_accuracy: 0.9913
Test loss: 0.026403134688735008
Test accuracy: 0.9912999868392944
```

Figure 5: Change curve of TensorFlow

From the figures, we can get that after the 10th epoch, the training loss is 0.0125, training accuracy is 0.9965, and test accuracy is 0.9913. Because I'm not familiar with TensorFlow, I didn't record the variation of every classes through time.

2 Exercise 2-3

In my opinion, the main difference between PyTorch and TensorFlow is the type of computational graphs. TensorFlow uses a static graph, while PyTorch utilizes a dynamic graph.

For TensorFlow, the static computational graph is defined before we running our model, using session to execute the graph.

On the other hand, PyTorch, we can define our graph as we running the model, it's easy to define and change. This is very helpful while using variable length inputs for Recurrent Neural Networks(RNNs).

Besides, according to my experience, I think PyTorch is more clear and easy to use. Its API is intuitive and easier to learn, while TensorFlow is a bit complex even if it integrated Keras in 2.0 version.

3 Exercise 2-4

As I mentioned in section 1, I used basically the same parameters, optimizers, etc. Though both of them get a high accuracy, the results of TensorFlow is a little bit better than that of PyTorch as shown in Table 1.

Table 1: Results

	Train Loss	Train Acc	Test Acc
PyTorch	0.0228	0.9928	0.9893
TensorFlow	0.0125	0.9965	0.9913

I think that might because of the following reasons:

- Different default initialization method. Rather than manually trying to unify the initialization process of the two frameworks, I just directly used the built-in initialization methods, which may cause differences.
- Similarly, I did not use the same random seed, which may also lead to different performance.
- Different implementation of mathematical formulas. In terms of the definition of mathematics, both of the two frameworks are correct, but because of the limitation of the precision of the float number of the computer, differences can arise during iteration process of a multilayer network.

4 Exercise 2-5

According to the definition of trace, the matrix product is exchangeable when calculating the trace as follows.

$$tr(AB) = \sum_{k=1}^{n} \sum_{i=1}^{n} x_{ki} y_{ik} = \sum_{i=1}^{n} \sum_{k=1}^{n} y_{ik} x_{ki} = tr(BA)$$

So for $tr(WAW^T)$ and $(WW^T = I)$, we have $tr(WAW^T) = tr((AW)W^T) = tr(A)$. So $argmax_W \ tr(WAW^T) = tr(A)$.

In other words, W can be regarded as a similarity transformation, and the similarity transformation does not change the trace of the matrix. Therefore, no matter what W is, it will not affect the trace of A.