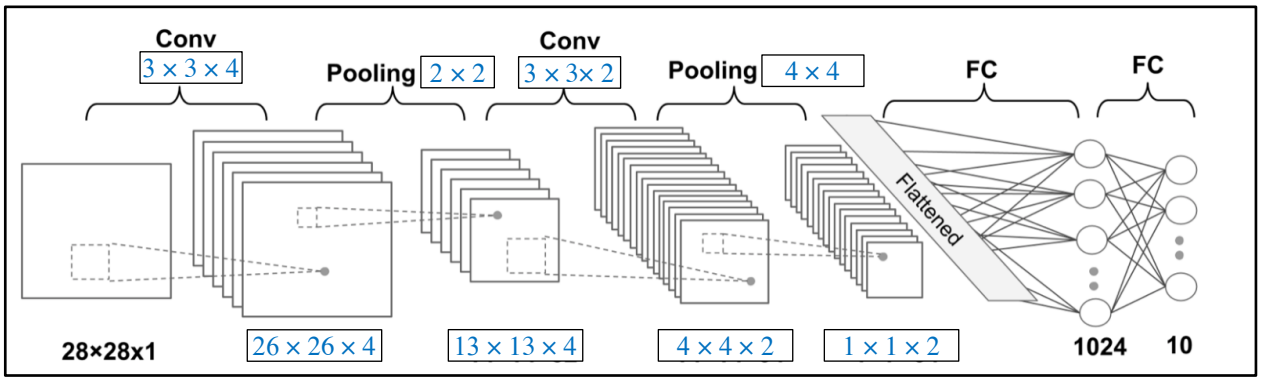
Report

- **Colab link:**

https://colab.research.google.com/drive/1MAbmRp6IS3oA1PMgPS79dk2Mq61DKfd_?usp=sharing

- **Task 1:**



1. Create the first convolutional layer using kernel size 3×3, strides 1×1, valid padding mode, and output channel size 4.
2. Create the first pooling layer using max pooling with pooling size 2×2 and strides 2×2.
3. Create the second convolutional layer using kernel size 3×3 , strides 3×3 , valid padding mode, and output channel size 2.
4. Create the second pooling layer using max pooling with pooling size 4×4 , and strides 4×4 .
5. Create a fully connected layer from the second pooling layer with output channel size 10.

- **Task 2:**

Layers	Output shape
Input	[100 × 28 × 28 × 1]
Conv_1	[100 × 26 × 26 × 4]
Pooling_1	[100 × 13 × 13 × 4]
Conv_2	[100 × 4 × 4 × 2]
Pooling_1	[100 × 1 × 1 × 2]
FC_1	[100 × 1024]
FC_2	[100 × 10]

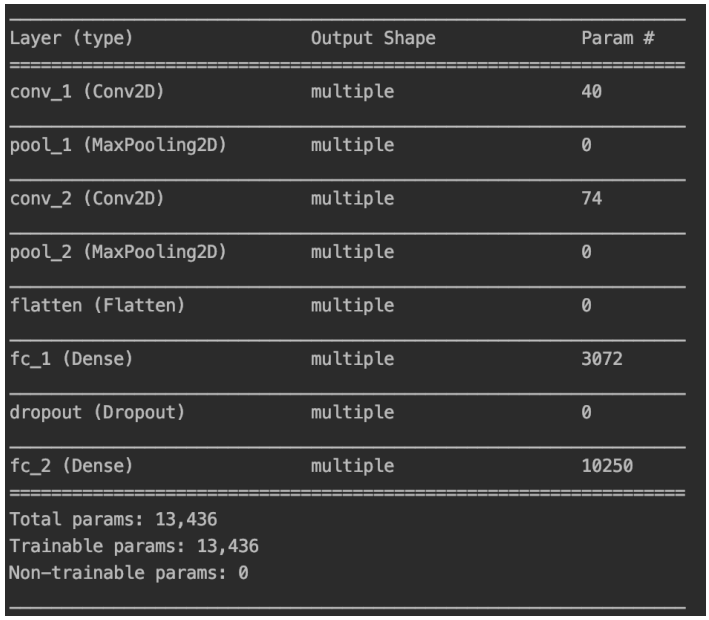
- **Task 3:**

```
import tensorflow as tf

model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=4, kernel_size=(3, 3), strides=(1, 1), padding='valid', data_format='channels_last', name='conv_1', activation='relu'))
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2), name='pool_1'))
model.add(tf.keras.layers.Conv2D(filters=2, kernel_size=(3, 3), strides=(3, 3), padding='valid', name='conv_2', activation='relu'))
model.add(tf.keras.layers.MaxPool2D(pool_size=(4, 4), strides=(4, 4), name='pool_2'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=1024, name='fc_1', activation='relu'))
model.add(tf.keras.layers.Dropout(rate=0.5))
model.add(tf.keras.layers.Dense(units=10, name='fc_2', activation='softmax'))
tf.random.set_seed(1)
model.build(input_shape=(None, 28, 28, 1))

print(model.summary())
```



Layer (type)	Output Shape	Param #
conv_1 (Conv2D)	multiple	40
pool_1 (MaxPooling2D)	multiple	0
conv_2 (Conv2D)	multiple	74
pool_2 (MaxPooling2D)	multiple	0
flatten (Flatten)	multiple	0
fc_1 (Dense)	multiple	3072
dropout (Dropout)	multiple	0
fc_2 (Dense)	multiple	10250

Total params: 13,436
Trainable params: 13,436
Non-trainable params: 0

Figure 1. CNN implementation

- **Task 4:**

The WHOLE MNIST dataset is successfully read and properly preprocessed.

Datasets:

1. mnist_train (50000)
2. mnist_valid (10000)
3. mnist_test (10000)

- **Task 5:**

Cases	Output Shape	Restults
Case 1: (Q1) Conv_1-(kernel_size:3×3, output_size: 4) Pooling_1-(MaxPool2D) Conv_2-(kernel_size:3×3, output_size: 2) Pooling_2-(MaxPool2D)	Input: (100, 28, 28, 1) Conv_1: (100, 26, 26, 4) Pooling_1: (100, 13, 13, 4) Conv_2: (100, 4, 4, 2) Pooling_2: (100, 1, 1, 2) FC_1: (100, 1024) FC_2: (100, 10)	Running time: 29.2 s Test Acc. 47.43%
Case 2: Conv_1-(kernel_size:3×3, output_size: 16) Pooling_1-(MaxPool2D) Conv_2-(kernel_size:3×3, output_size: 32) Pooling_2-(MaxPool2D)	Input: (100, 28, 28, 1) Conv_1: (100, 26, 26, 16) Pooling_1: (100, 13, 13, 16) Conv_2: (100, 4, 4, 32) Pooling_2: (100, 1, 1, 32) FC_1: (100, 1024) FC_2: (100, 10)	Running time: 27.6 s Test Acc. 95.84%
Case 3: Conv_1-(kernel_size:3×3, output_size: 4) Pooling_1-(AveragePool) Conv_2-(kernel_size:3×3, output_size: 2) Pooling_2-(AveragePool)	Input: (100, 28, 28, 1) Conv_1: (100, 26, 26, 4) Pooling_1: (100, 13, 13, 4) Conv_2: (100, 4, 4, 2) Pooling_2: (100, 1, 1, 2) FC_1: (100, 1024) FC_2: (100, 10)	Running time: 28.5 s Test Acc. 55.88%
Case 4: Conv_1-(kernel_size:5×5, output_size: 4) Pooling_1-(MaxPool2D) Conv_2-(kernel_size:3×3, output_size: 2) Pooling_2-(MaxPool2D)	Input: (100, 28, 28, 1) Conv_1: (100, 24, 24, 4) Pooling_1: (100, 12, 12, 4) Conv_2: (100, 4, 4, 2) Pooling_2: (100, 1, 1, 2) FC_1: (100, 1024) FC_2: (100, 10)	Running time: 27.1 s Test Acc. 52.26%

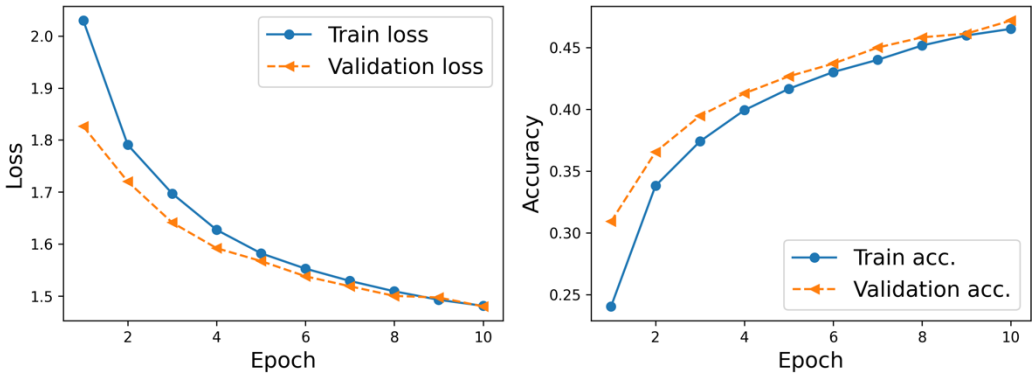


Figure 2 Case 1

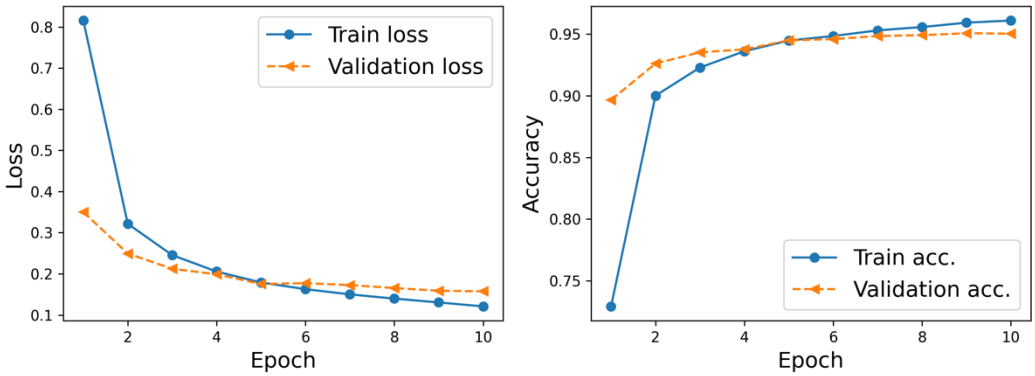


Figure 3 Case 2

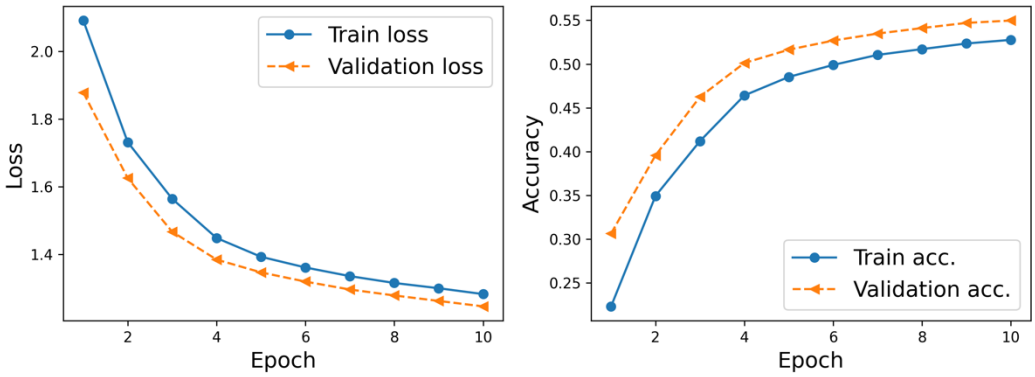


Figure 4 Case 3

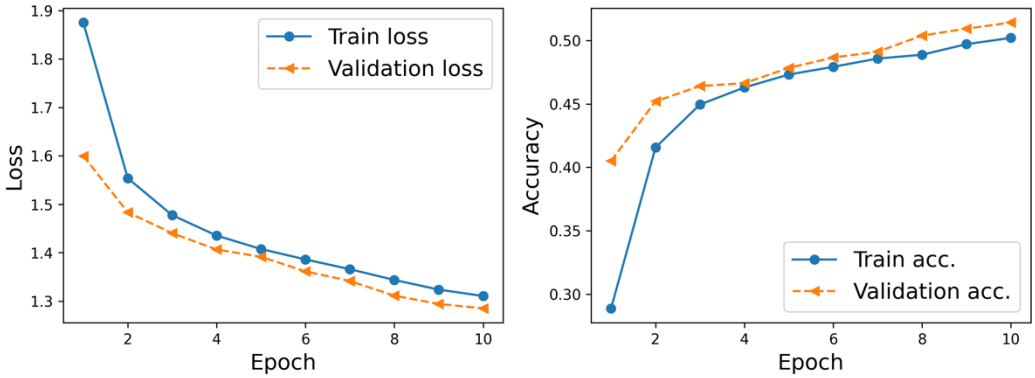


Figure 5 Case 4

Running cases:

- Case 1: test case for Q1.
- Case 2: only change filters in Conv_1 and Conv_2 layers. (compared with Case 1)
- Case 3: only change pooling method to AvgPol2D in Pooling_1 and Pooling_2 layers (compared with Case 1)
- Case 4: only change kernel_size in Conv_1 layer. (compared with Case 1)

Analysis:

- Compare Case 1 and 2, increasing the number of filters in Convolution layers can significantly increase the accuracy. (testing accuracy increase from 47.43% to 95.84%)
- Compare Case 1 and 3, changing the pooling method from MaxPool2D to AvgPool2D lead to an accuracy increase in testing accuracy (from 47.43% to 55.88%).
- Compare Case 1 and 3, increasing the kernel_size in Conv_1 layer from 3×3 to 5×5 result in the testing accuracy from 47.43% to 52.26%.
- Overall running time: the running time for these four cases do not vary too much (27.1~29.2s) with the help of GPU acceleration in Google Colab.
- Training and validation loss: the loss curves in Figures 2 to 5 experience a continuous decrease as the increase of epochs, which means good parameter setting.
- Training and validation accuracy: only Figure 3 shows a slight overfitting. Whereas, Figures 2, 4, 5 show the trend of underfitting since the validation accuracy is higher than training accuracy during the training process.