人工智慧模型設計與應用 Lab2

NM6121030 余振揚

1. Hyper Parameters:

- Epochs: 10
- Batch Size: 128
- Validation Split: 20% of training set

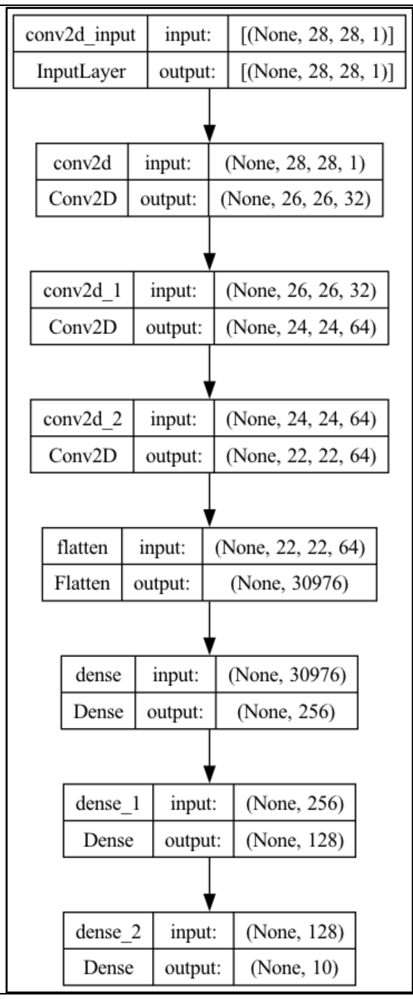
2. Base Model (3 CNN layers + 3 NN layers):

- Model Structure:

```
# CNN layers
# 1st
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))   # (28, 28, 1) -> (26, 26, 32)
# 2nd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (26, 26, 32) -> (24, 24, 64)
# 3rd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (24, 24, 64) -> (22, 22, 64)
# Flatten for NN layers
model.add(Flatten())    # (22, 22, 64) -> (30976,)
# NN layers
# 1st
model.add(Dense(256, activation='relu'))    # (30976,) -> (256,)
# 2nd
model.add(Dense(128, activation='relu'))    # (256,) -> (128,)
# 3rd
model.add(Dense(classes, activation='softmax')) # (128,) -> (10,)
```

- Model Summary and Plot:

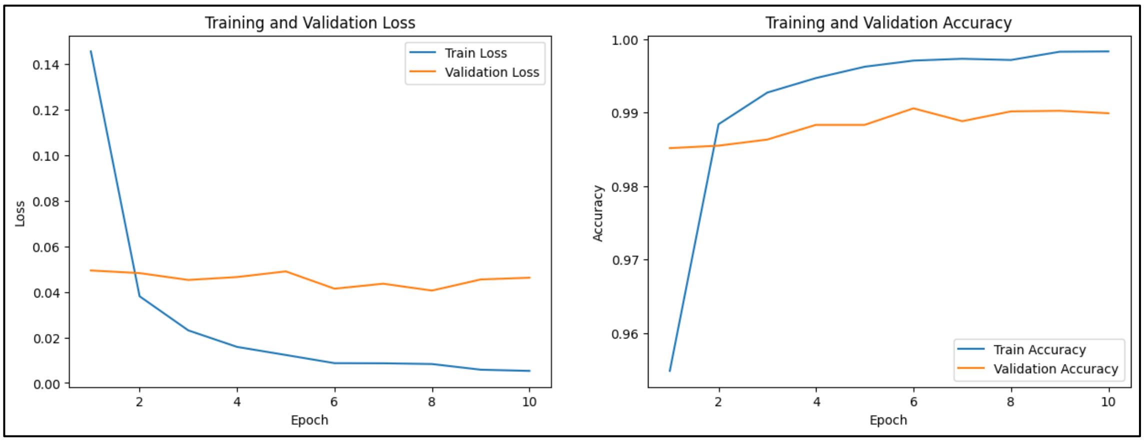
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 conv2d_2 (Conv2D)           (None, 22, 22, 64)        36928

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 256)               7930112

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 8020042 (30.59 MB)
Trainable params: 8020042 (30.59 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



- Accuracy and Loss (Final Epoch):

Total Training Time: 591s

	Accuracy	Loss
Train	0.9983	0.0053
Validation	0.9899	0.0462



- Test Accuracy: 0.9912

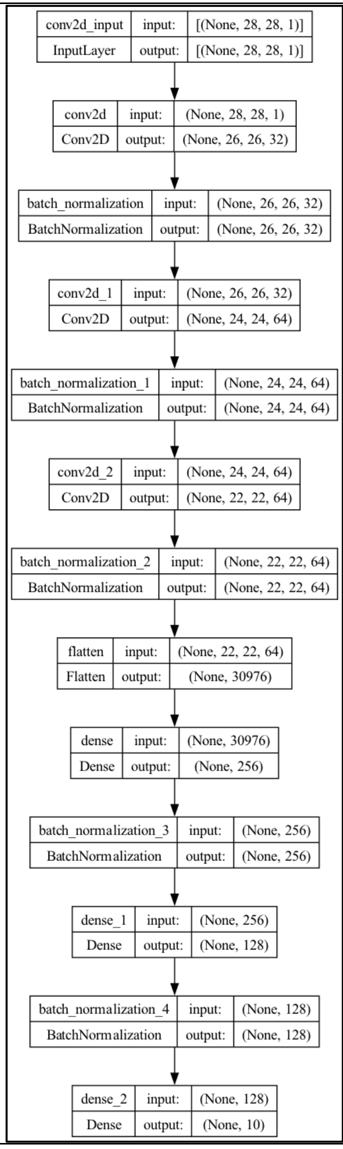
3. Base Model + BatchNormalization:

- Model Structure:

```
# CNN layers
# 1st
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))   # (28, 28, 1) -> (26, 26, 32)
model.add(BatchNormalization())
# 2nd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (26, 26, 32) -> (24, 24, 64)
model.add(BatchNormalization())
# 3rd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (24, 24, 64) -> (22, 22, 64)
model.add(BatchNormalization())
# Flatten for NN layers
model.add(Flatten())    # (22, 22, 64) -> (30976,)
# NN layers
# 1st
model.add(Dense(256, activation='relu'))    # (30976,) -> (256,)
model.add(BatchNormalization())
# 2nd
model.add(Dense(128, activation='relu'))    # (256,) -> (128,)
model.add(BatchNormalization())
# 3rd
model.add(Dense(classes, activation='softmax')) # (128,) -> (10,)
```

- Model Summary:

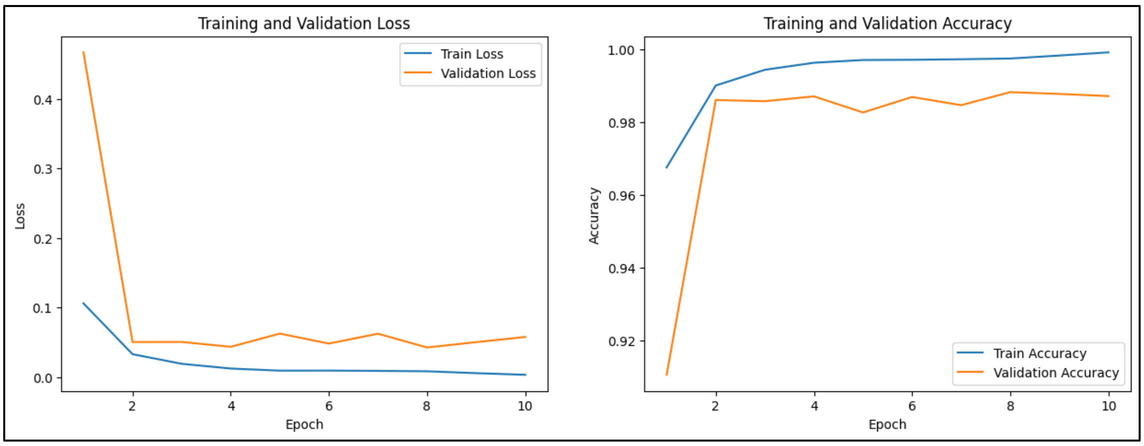
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (Batch  (None, 26, 26, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 batch_normalization_1 (Bat  (None, 24, 24, 64)        256
 chNormalization)

 conv2d_2 (Conv2D)           (None, 22, 22, 64)        36928

 batch_normalization_2 (Bat  (None, 22, 22, 64)        256
 chNormalization)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 256)               7930112

 batch_normalization_3 (Bat  (None, 256)               1024
 chNormalization)

 dense_1 (Dense)             (None, 128)               32896

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 8022218 (30.60 MB)
Trainable params: 8021130 (30.60 MB)
Non-trainable params: 1088 (4.25 KB)
```



- Accuracy and Loss (Final Epoch):

Total Training Time: 710s

	Accuracy	Loss
Train	0.9992	0.0030
Validation	0.9872	0.0574



- Test Accuracy: 0.9876

4. **Base Model + Arbitrary Layer:**

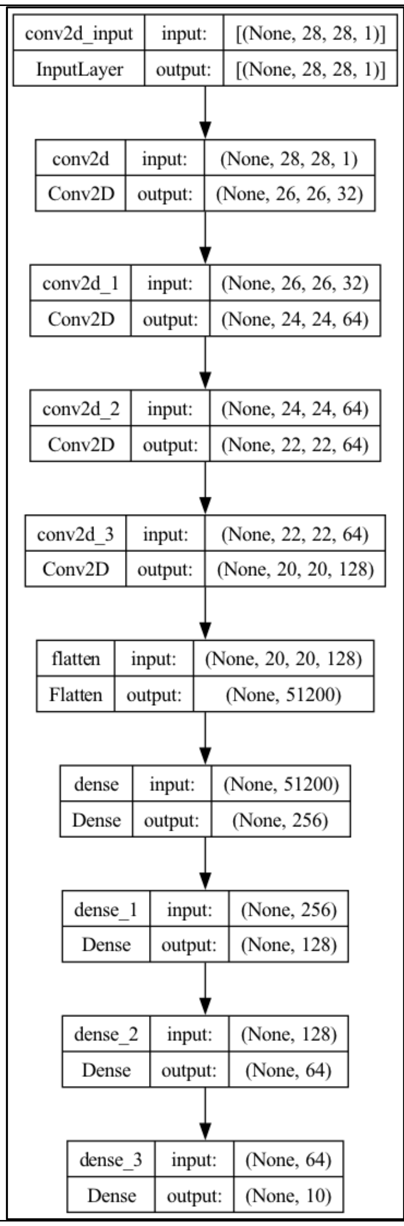
Add one more CNN layer and one more Dense Layer.

- Model Structure:

```
# CNN layers
# 1st
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))   # (28, 28, 1) -> (26, 26, 32)
# 2nd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (26, 26, 32) -> (24, 24, 64)
# 3rd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (24, 24, 64) -> (22, 22, 64)
# 4th
model.add(Conv2D(128, (3, 3), activation='relu'))    # (22, 22, 64) -> (20, 20, 64)
# Flatten for NN layers
model.add(Flatten())    # (20, 20, 64) -> (25600,)
# NN layers
# 1st
model.add(Dense(256, activation='relu'))    # (25600,) -> (256,)
# 2nd
model.add(Dense(128, activation='relu'))    # (256,) -> (128,)
# 3rd
model.add(Dense(64, activation='relu'))    # (128,) -> (64,)
# 4th
model.add(Dense(classes, activation='softmax')) # (64,) -> (10,)
```

- Model Summary and Plot:

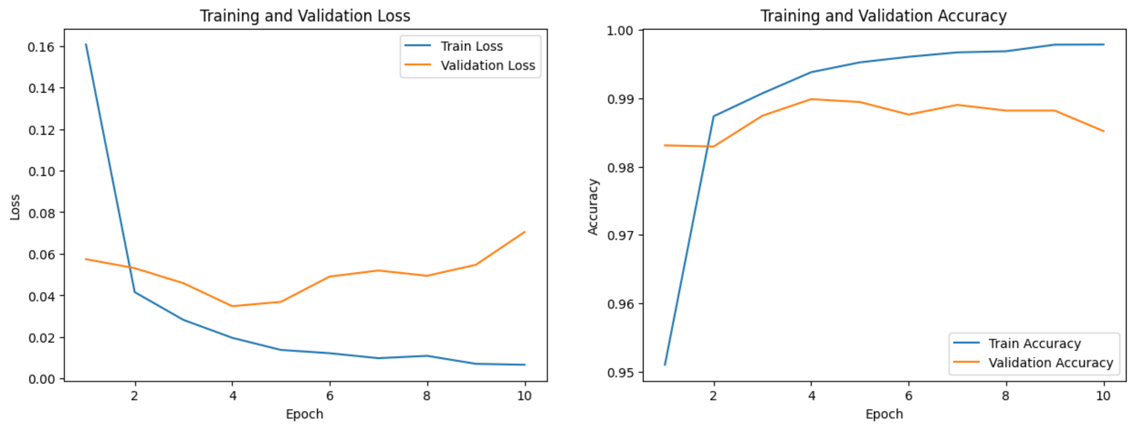
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 conv2d_2 (Conv2D)           (None, 22, 22, 64)        36928

 conv2d_3 (Conv2D)           (None, 20, 20, 128)       73856

 flatten (Flatten)           (None, 51200)             0

 dense (Dense)               (None, 256)               13107456

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 13278858 (50.65 MB)
Trainable params: 13278858 (50.65 MB)
Non-trainable params: 0 (0.00 Byte)
```



- Accuracy and Loss (Final Epoch):

  Total Training Time: 1063s (17mins)

	Accuracy	Loss
Train	0.9978	0.0066
Validation	0.9852	0.0705



- Test Accuracy: 0.9848

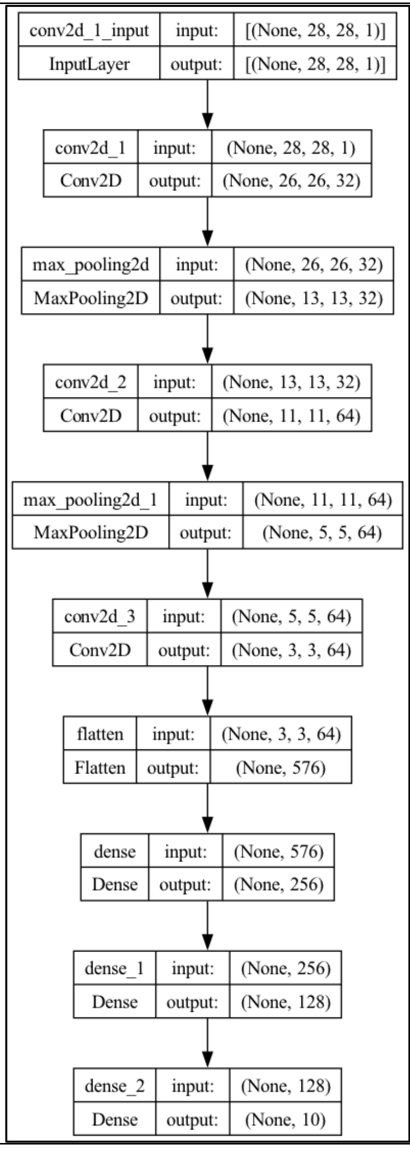
5. Base Model + MaxPooling: Final Model

- Model Structure:

```
# CNN layers
# 1st
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))   # (28, 28, 1) -> (26, 26, 32)
model.add(MaxPooling2D(2, 2))   # (26, 26, 32) -> (13, 13, 32)
# 2nd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (13, 13, 32) -> (11, 11, 64)
model.add(MaxPooling2D(2, 2))   # (11, 11, 64) -> (5, 5, 64)
# 3rd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (5, 5, 64) -> (3, 3, 64)
# Flatten for NN layers
model.add(Flatten())    # (3, 3, 64) -> (576,)
# 1st
model.add(Dense(256, activation='relu'))    # (576,) -> (256,)
# 2nd
model.add(Dense(128, activation='relu'))    # (256,) -> (128,)
# 3rd
model.add(Dense(classes, activation='softmax')) # (128,) -> (10,)
```

- Model Summary and Plot:

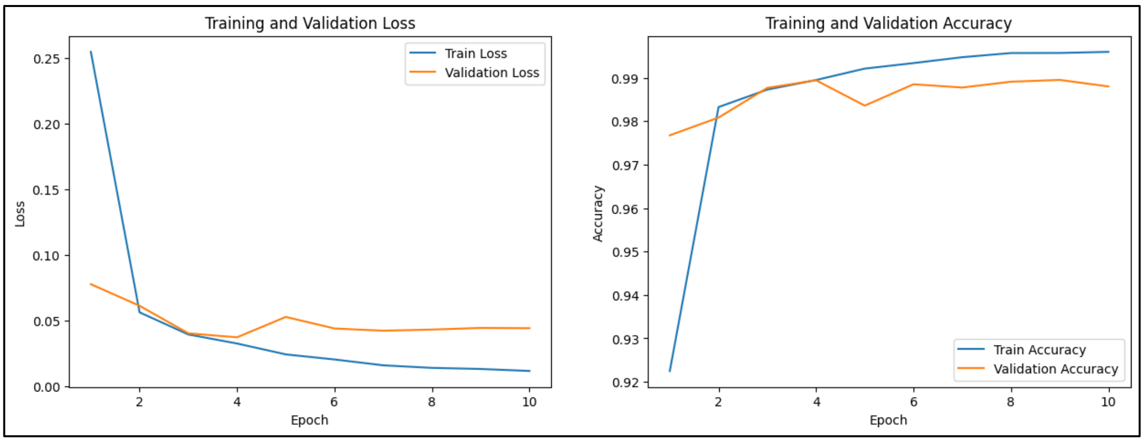
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_3 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 237642 (928.29 KB)
Trainable params: 237642 (928.29 KB)
Non-trainable params: 0 (0.00 Byte)
```



- Accuracy and Loss (Final Epoch):

Total Training Time: 70s

	Accuracy	Loss
Train	0.9960	0.0118
Validation	0.9880	0.0443



- Test Accuracy: 0.9893

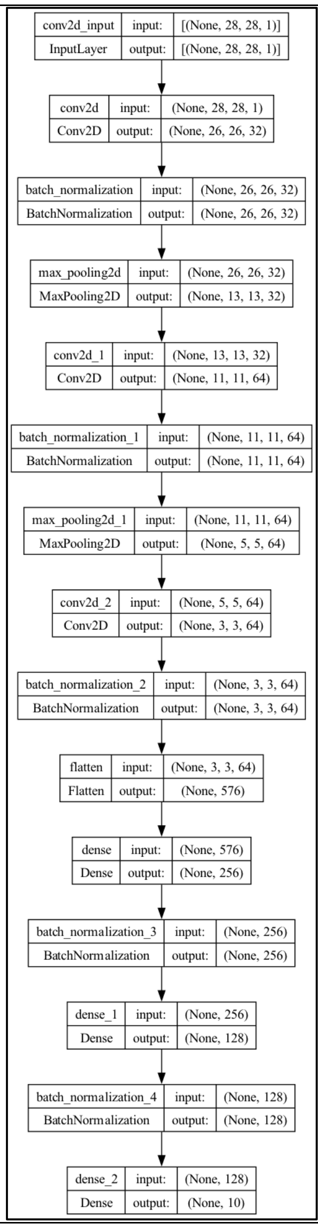
6. Base Model + MaxPooling + BatchNormalization:

- Model Structure:

```
# CNN layers
# 1st
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))   # (28, 28, 1) -> (26, 26, 32)
model.add(BatchNormalization()) # Batch Normalization
model.add(MaxPooling2D(2, 2))   # (26, 26, 32) -> (13, 13, 32)
# 2nd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (13, 13, 32) -> (11, 11, 64)
model.add(BatchNormalization()) # Batch Normalization
model.add(MaxPooling2D(2, 2))   # (11, 11, 64) -> (5, 5, 64)
# 3rd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (5, 5, 64) -> (3, 3, 64)
model.add(BatchNormalization()) # Batch Normalization
# Flatten for NN layers
model.add(Flatten())    # (3, 3, 64) -> (576,)
# 1st
model.add(Dense(256, activation='relu'))    # (576,) -> (256,)
model.add(BatchNormalization()) # Batch Normalization
# 2nd
model.add(Dense(128, activation='relu'))    # (256,) -> (128,)
model.add(BatchNormalization()) # Batch Normalization
# 3rd
model.add(Dense(classes, activation='softmax')) # (128,) -> (10,)
```

- Model Summary and Plot:

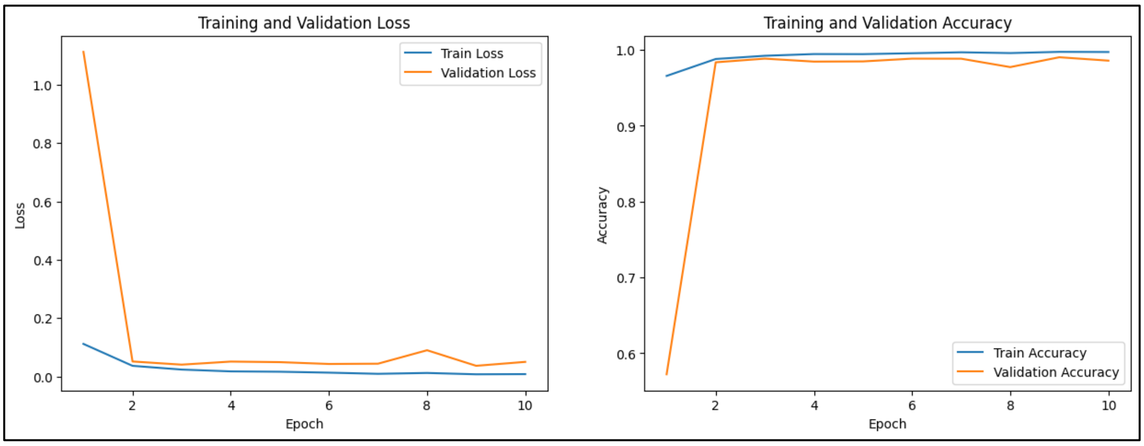
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (Batch  (None, 26, 26, 32)        128
Normalization)

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 batch_normalization_1 (Bat  (None, 11, 11, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 batch_normalization_2 (Bat  (None, 3, 3, 64)          256
 chNormalization)

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 batch_normalization_3 (Bat  (None, 256)               1024
 chNormalization)

 dense_1 (Dense)             (None, 128)               32896

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 239818 (936.79 KB)
Trainable params: 238730 (932.54 KB)
Non-trainable params: 1088 (4.25 KB)
```



- Accuracy and Loss (Final Epoch):

  Total Training Time: 82s

	Accuracy	Loss
Train	0.9973	0.0086
Validation	0.9859	0.0505



- Test Accuracy: 0.9870

7. **Base Model + MaxPooling + CrossEntropy (Label as OneHotEncoding):**

At previous model, I use sparse_categorical_crossentropy as my loss function.

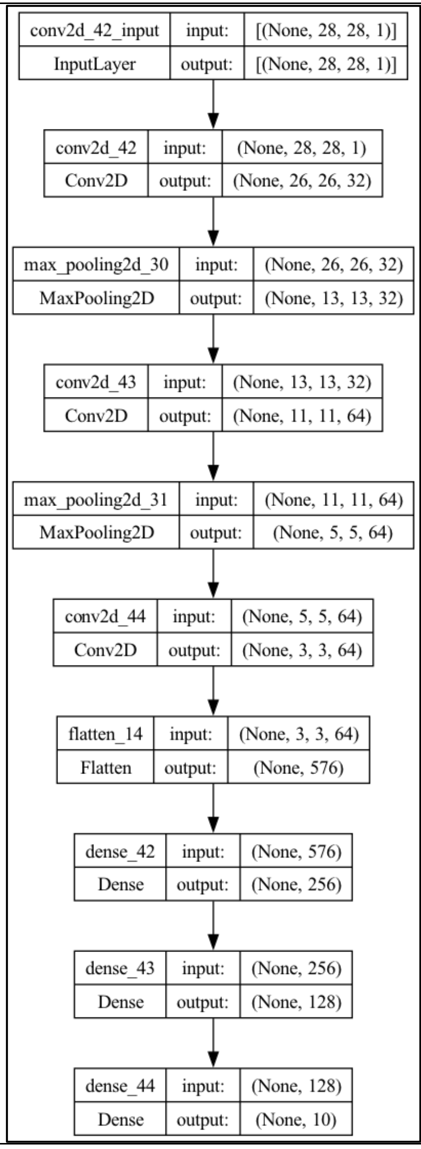
Now I convert the train_y into one hot encoding format, making it to fit the format of using categorical_crossentropy loss function.

- Model Structure:

```
# CNN layers
# 1st
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))   # (28, 28, 1) -> (26, 26, 32)
model.add(MaxPooling2D((2, 2))) # (26, 26, 32) -> (13, 13, 32)
# 2nd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (13, 13, 32) -> (11, 11, 64)
model.add(MaxPooling2D((2, 2))) # (11, 11, 64) -> (5, 5, 64)
# 3rd
model.add(Conv2D(64, (3, 3), activation='relu'))    # (5, 5, 64) -> (3, 3, 64)
# The choice of where to place MaxPooling layers depends on the network architecture and the trade-off bet
# Flatten for NN layers
model.add(Flatten())    # (3, 3, 64) -> (576,)
# NN layers
# 1st
model.add(Dense(256, activation='relu'))    # (576,) -> (256,)
# 2nd
model.add(Dense(128, activation='relu'))    # (256,) -> (128,)
# 3rd
model.add(Dense(classes, activation='softmax')) # (128,) -> (10,)
```

- Model Summary and Plot:

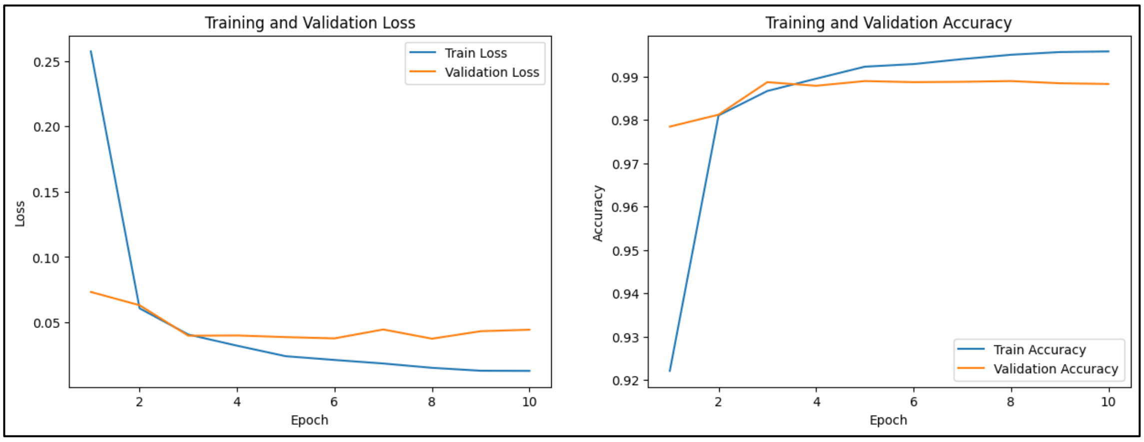
```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_42 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_30 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 conv2d_43 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_31 (MaxPooli  (None, 5, 5, 64)          0
 ng2D)

 conv2d_44 (Conv2D)          (None, 3, 3, 64)          36928

 flatten_14 (Flatten)        (None, 576)               0

 dense_42 (Dense)            (None, 256)               147712

 dense_43 (Dense)            (None, 128)               32896

 dense_44 (Dense)            (None, 10)                1290

=================================================================
Total params: 237642 (928.29 KB)
Trainable params: 237642 (928.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



- Accuracy and Loss:

Total Training Time: 70s

	Accuracy	Loss
Train	0.9958	0.0129
Validation	0.9883	0.0444



- Test Accuracy: 0.9889

8. **Model Comparisons:**

Training time depends on device, for my case, I use MacBook Air M2 for training.

	Base Model	Batch Normalization	Arbitrary Layer	MaxPooling (Final Model)	MaxPooling + BatchNormalization	MaxPooling + CrossEntropy
Test Accuracy	0.9912	0.9876	0.9848	0.9893	0.9870	0.9889
Validation Accuracy	0.9899	0.9872	0.9852	0.9880	0.9859	0.9883
Validation Loss	0.0462	0.0574	0.0705	0.0443	0.0505	0.0444
Training Time	591s	710s	1063s	70s	82s	70s

9. **Final Model and Discussion:**

After consideration for this classification task and taking into account the required training time, I have decided to use the base model with MaxPooling and the Sparse Categorical CrossEntropy loss function. This decision is based on several key factors:

1. **Label Format**: Sparse Categorical CrossEntropy is suitable when target labels are provided as integers, where each integer represents the class index. In our case, such integer-based labels align perfectly with the nature of our problem, where classes are distinct and have no inherent order. Using integer labels simplifies our data representation.
2. **Memory Efficiency**: Sparse Categorical CrossEntropy significantly reduces memory requirements compared to Categorical CrossEntropy with one-hot encoded labels. It saves memory by representing labels as integers rather than binary vectors. This memory efficiency becomes especially important when dealing with a large number of classes or a sizable dataset, as it minimizes storage needs and improves overall training efficiency.
3. **Task Alignment**: Our task involves predicting labels that do not exhibit strong correlations with each other. Sparse Categorical CrossEntropy suits this scenario by working directly with integer labels, avoiding the complexity of one-hot encoding. This choice aligns the loss function with the problem's characteristics.

10. **Final Model Prediction:**