딥러닝

과제3

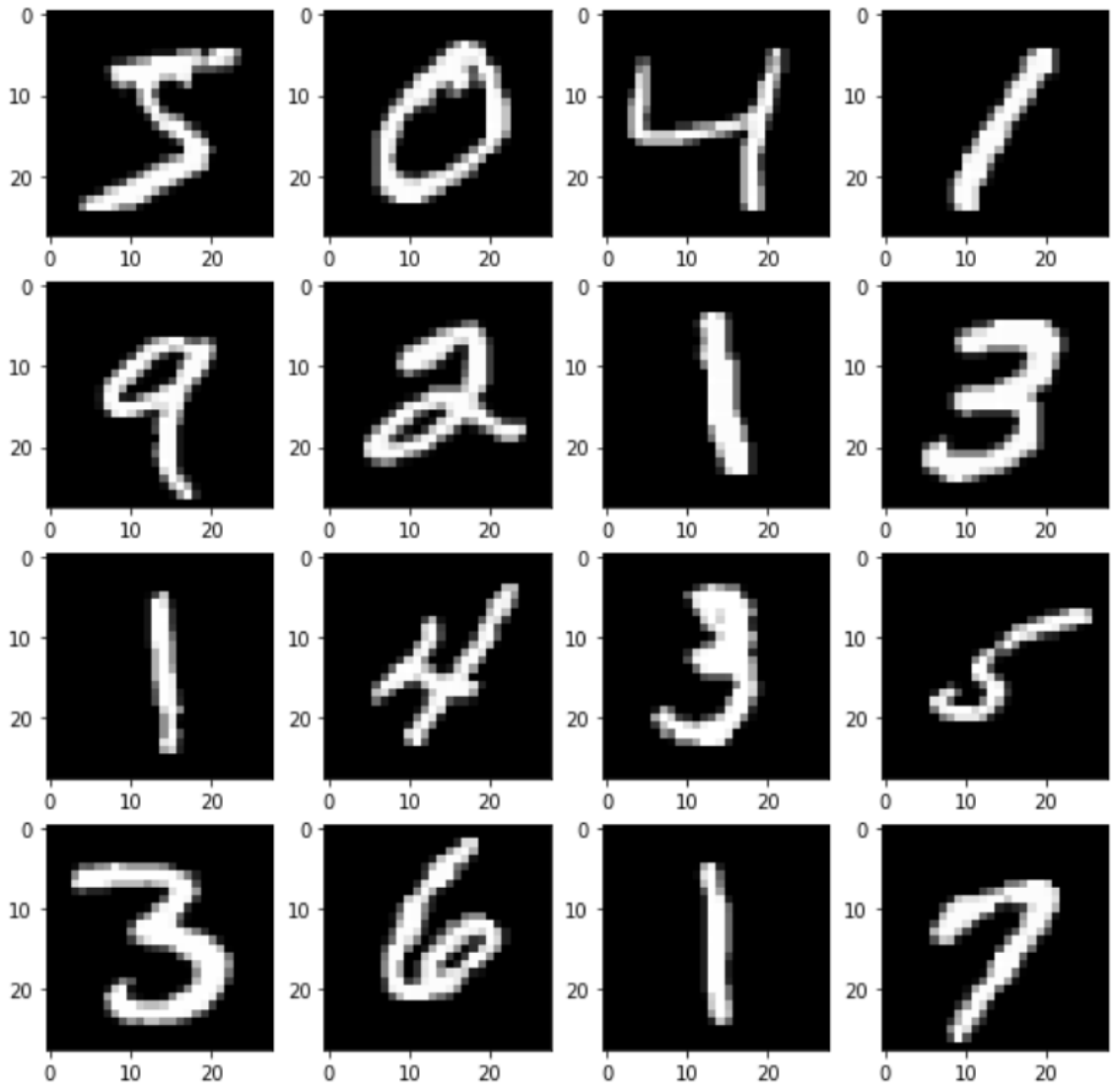
2019305059

이현수

문제 1> 필기체 MNIST에 관련한 내용이다.

①예제5.19를 이용하여 필기체 MNIST(tf.keras.datasets.mnist)의 학습데이터와 테스트 데이터를 불러온 후 0~1 사이로 정규화 하고 P152와 같이 4x4 16장 이미지를 plt.show() 한다. --> 코드와 이미지를 캡쳐하여 제출

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(train_X, train_Y), (test_X, test_Y) = mnist.load_data()

train_X = train_X / 255.0
test_X = test_X / 255.0

import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
for c in range(16):
    plt.subplot(4,4,c+1)
    plt.imshow(train_X[c].reshape(28,28), cmap='gray')
plt.show()
print(train_Y[:16])
```



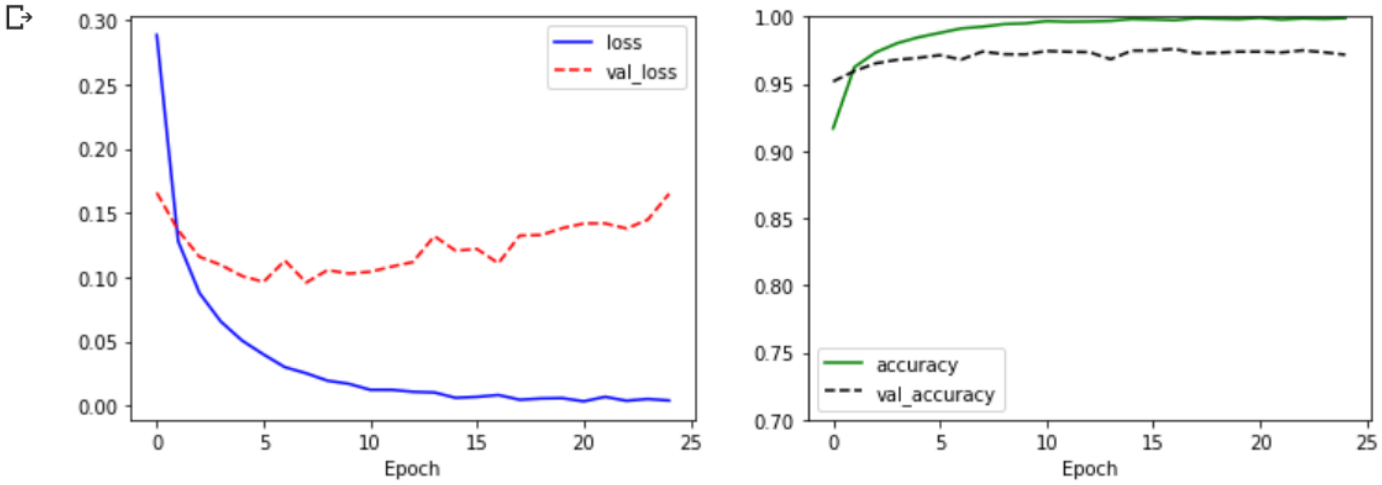
```
[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1 7]
```

②예제5.22의 신경망 모델을 사용하여 필기체 MNIST를 학습 한 후 P137와 같이 loss, val_loss, accuracy,

val_accuracy를 plt.show() 한다. --> 코드와 이미지를 캡쳐하여 제출

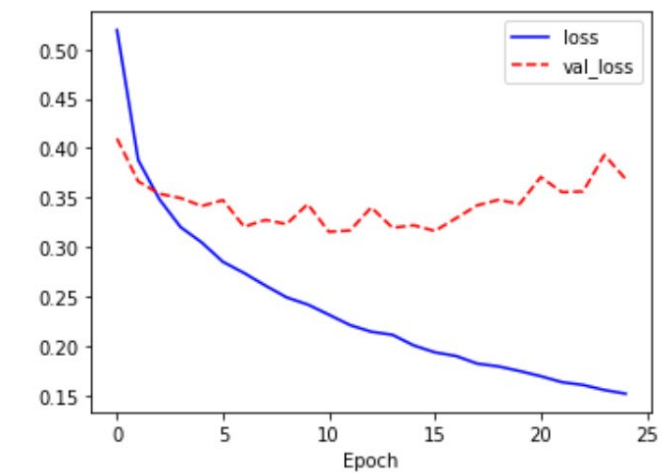
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

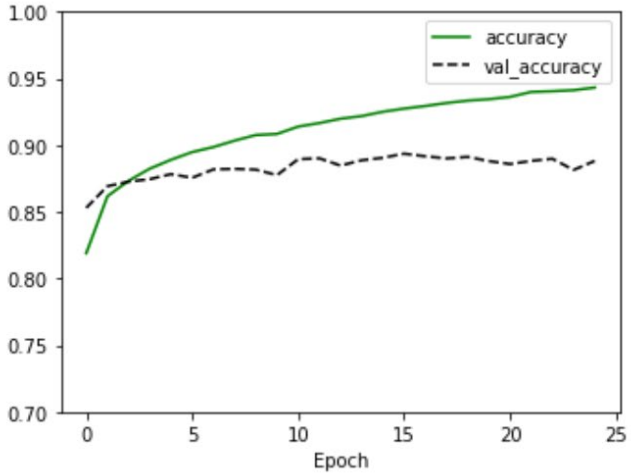
```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()
```



③예제5.22에서 fashion_MNIST의 loss와 accuracy와 ②번의 결과와 비교하고 차이가 나면 그이유를 설명하시오

Fashion_MNIST

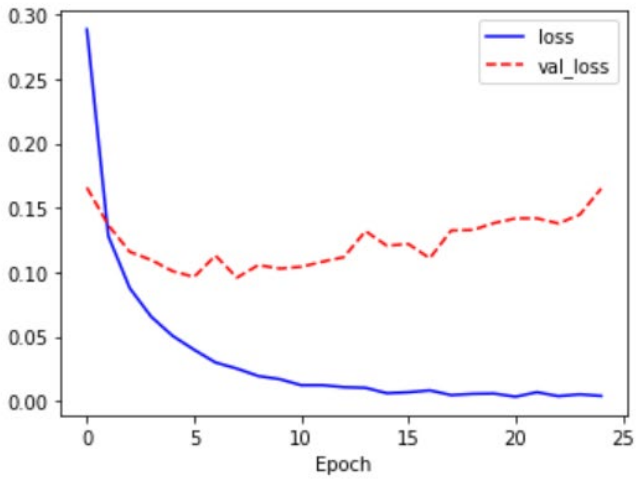


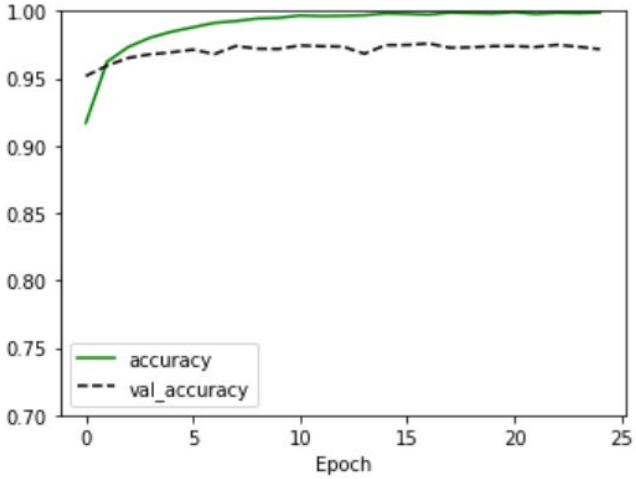



```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.4029 - accuracy: 0.8824
[0.40286707878112793, 0.8823999762535095]
```

MNIST






```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 0s 991us/step - loss: 0.1447 - accuracy: 0.9737
[0.1447293907403946, 0.9736999869346619]
```

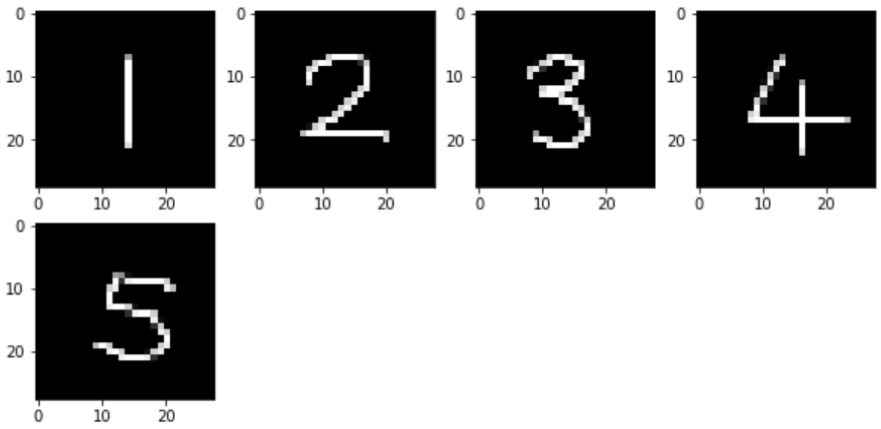
Fashion_MNIST 보다 MNIST의 오차가 더 적고, 정확도는 더 높다. 그 이유는 Fashion_MNIST의 학습데이터 옷, 신발, 가방의 이미지이다. 그래서 손으로 쓴 숫자 글씨 인 MNIST 학습데이터보다 Fashion_MNIST 학습데이터가 더 복잡하고 어렵다. 그래서 학습하기 더 어렵다. 그래서 같은 5.22예제 신경망 모델로 학습시켰을 때 MNIST가 Fashion_MNIST 보다 학습 결과가 더 좋아 오차는 적고 정확도가 높다.

④테스트 이미지 5장을 PC 그림판에서 아래와 같이 만들고 model.predict(X) 실행하여 5장의 accuracy를 출력한다. --> 코드와 테스트 이미지, accuracy를 제출한다.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
from PIL import Image
import numpy as np
img1=Image.open('/content/drive/MyDrive/test/test1.png').convert("L")
img2=Image.open('/content/drive/MyDrive/test/test2.png').convert("L")
img3=Image.open('/content/drive/MyDrive/test/test3.png').convert("L")
img4=Image.open('/content/drive/MyDrive/test/test4.png').convert("L")
img5=Image.open('/content/drive/MyDrive/test/test5.png').convert("L")
img1=np.array(img1)
img2=np.array(img2)
img3=np.array(img3)
img4=np.array(img4)
img5=np.array(img5)

X = np.array([img1,img2,img3,img4,img5])
Y=np.array([1,2,3,4,5])

X=X/255.0

plt.figure(figsize=(10,10))
for c in range(5):
    plt.subplot(4,4,c+1)
    plt.imshow(X[c].reshape(28,28), cmap='gray')
plt.show()
print(Y[:])
```



```
[1 2 3 4 5]
```

```
model.predict(X)
```

```
array([[1.5830171e-12, 9.9979728e-01, 7.0397900e-06, 1.2430070e-04,
        3.3800227e-06, 2.1294402e-07, 2.7784938e-06, 2.6670348e-05,
        1.9503663e-05, 1.8801638e-05],
       [1.1683363e-08, 4.0525670e-06, 9.9762863e-01, 1.4377261e-03,
        1.3730339e-15, 1.6575084e-09, 2.3881977e-08, 1.9986251e-04,
        7.2959077e-04, 8.8807148e-08],
       [4.9805732e-11, 8.3671712e-06, 4.1612231e-05, 9.9974614e-01,
        3.0036102e-08, 1.9525453e-04, 1.2268357e-09, 8.2149648e-10,
        7.4308668e-06, 1.1063046e-06],
       [4.5767625e-09, 1.3319900e-09, 1.6738979e-04, 6.8541695e-10,
        9.9977142e-01, 2.0152859e-06, 5.9092981e-05, 6.9570959e-08,
        5.4577010e-11, 3.5477263e-10],
       [7.9187512e-06, 1.0495744e-05, 8.0871029e-04, 7.5062513e-02,
        5.0368352e-04, 9.2285734e-01, 8.3925843e-06, 2.1366267e-04,
        1.7754218e-04, 3.4977455e-04]], dtype=float32)
```

```
model.evaluate(X,Y)
```

```
1/1 [==============================] - 0s 2ms/step - loss: 0.0167 - accuracy: 1.0000
[0.016668017953634262, 1.0]
```

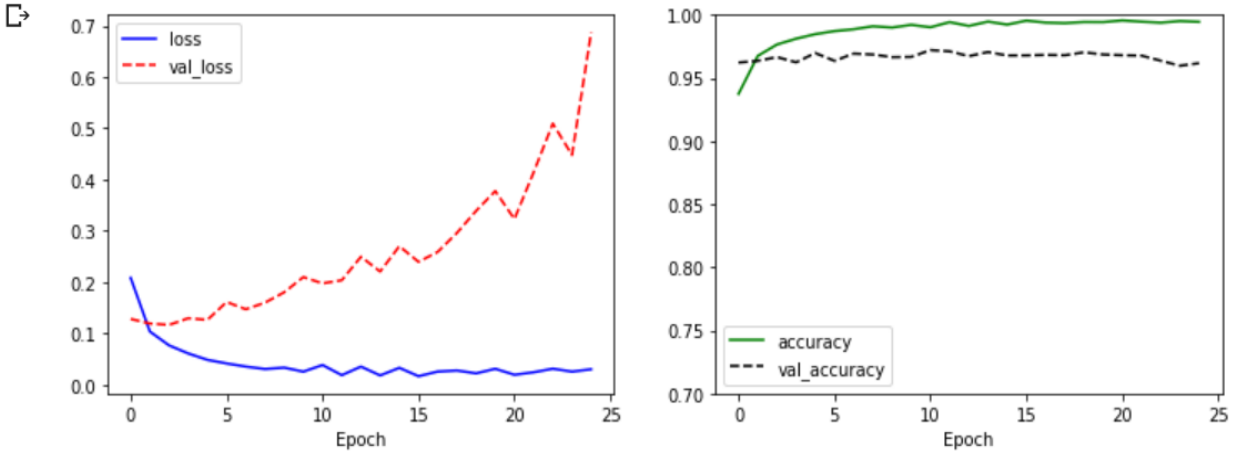
문제 2> MNIST을 CNN에 적용한 내용이다.

①문제1>의 MNIST 학습데이터를 이용하여 예제 6.7의 CNN에 적용하여 학습시키고 그 결과를 문제1>의 ②와 비교한다. -->비교설명과 코드와 이미지를 캡쳐하여 제출

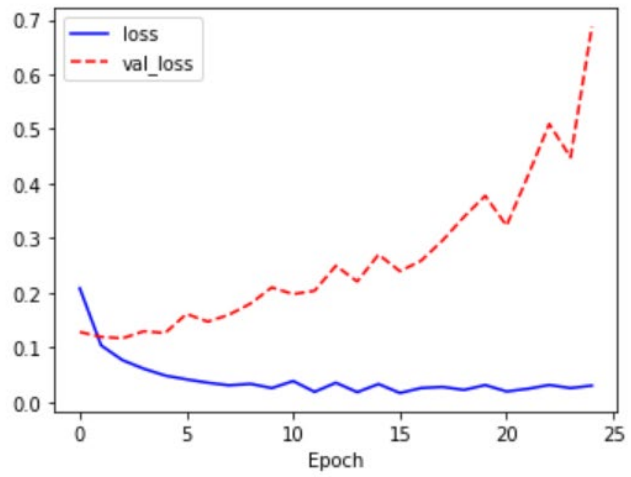
```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(train_X, train_Y), (test_X, test_Y) = mnist.load_data()

train_X = train_X / 255.0
test_X = test_X / 255.0

train_X = train_X.reshape(-1,28,28,1)
test_X = test_X.reshape(-1,28,28,1)
```

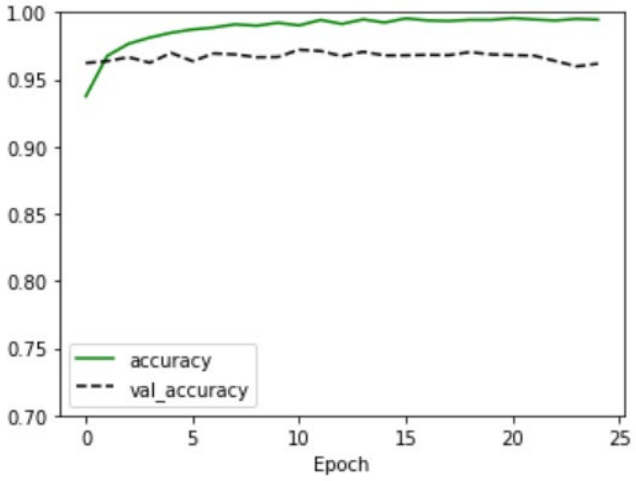
```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=16),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history=model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()
```



CNN적용

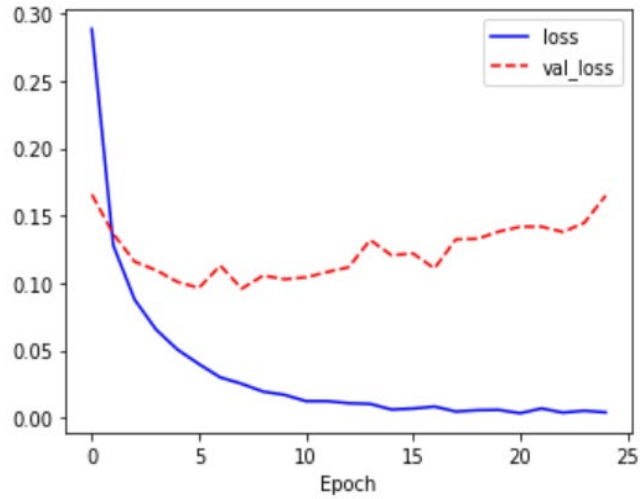


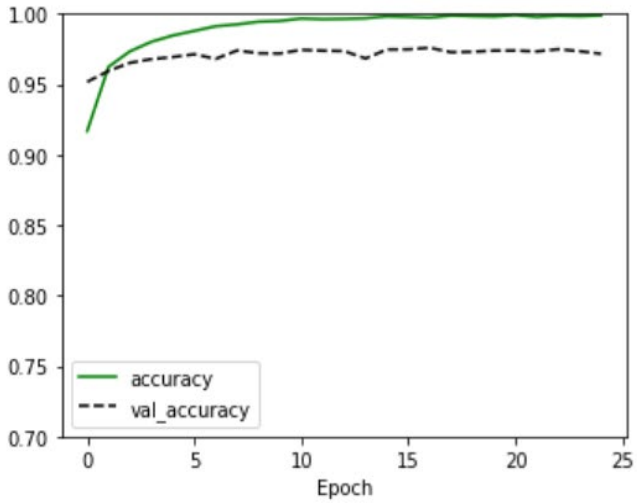



```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.6283 - accuracy: 0.9640
[0.6283432841300964, 0.9639999866485596]
```

문제1>의②






```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 0s 991us/step - loss: 0.1447 - accuracy: 0.9737
[0.1447293907403946, 0.9736999869346619]
```

CNN을 적용했지만 그래프와 오차, 정확도를 보게되면 학습결과가 더 나쁘다. 오차는 더 커졌고 정확도 역시 줄어들었다.

CNN적용의 왼쪽 그래프를 보면 loss는 감소하고 val_loss는 증가하는 전형적인 과적합의 형태를 나타낸다.

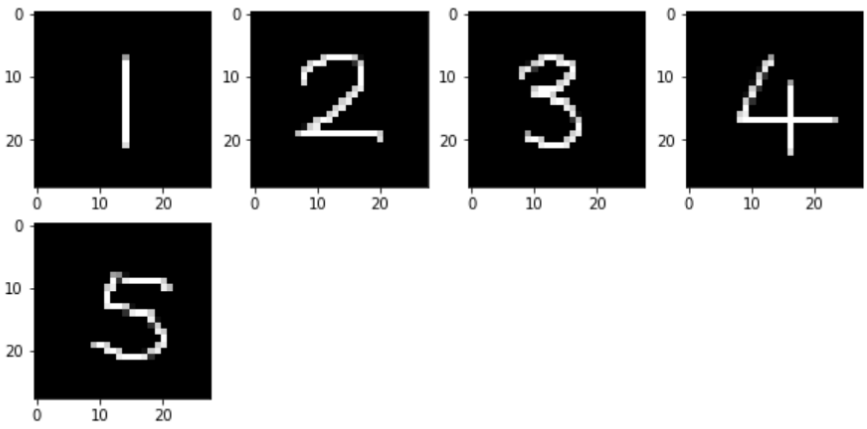
과적합이 일어나서 CNN(컨볼루션 신경망)이 문제1>의 ②신경망보다 더 학습결과가 나쁘게 나왔다.

②위 ①번에 문제1> ④번의 테스트 이미지 5장을 사용하여 accuracy를 구하여 제출

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from PIL import Image
import numpy as np
img1=Image.open('/content/drive/MyDrive/test/test1.png').convert("L")
img2=Image.open('/content/drive/MyDrive/test/test2.png').convert("L")
img3=Image.open('/content/drive/MyDrive/test/test3.png').convert("L")
img4=Image.open('/content/drive/MyDrive/test/test4.png').convert("L")
img5=Image.open('/content/drive/MyDrive/test/test5.png').convert("L")
img1=np.array(img1)
img2=np.array(img2)
img3=np.array(img3)
img4=np.array(img4)
img5=np.array(img5)

X = np.array([img1,img2,img3,img4,img5])
Y=np.array([1,2,3,4,5])

X=X/255.0
X = X.reshape(-1,28,28,1)

plt.figure(figsize=(10,10))
for c in range(5):
    plt.subplot(4,4,c+1)
    plt.imshow(X[c].reshape(28,28), cmap='gray')
plt.show()
print(Y[:])
```



```
[1 2 3 4 5]
```

```
model.predict(X)
```

```
array([[0.0000000e+00, 1.0000000e+00, 6.7231734e-21, 6.6441329e-22,
        1.0321223e-28, 1.9246241e-32, 1.6242982e-32, 1.6828670e-25,
        7.3893303e-32, 3.5118035e-31],
       [0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 3.3279653e-32,
        0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.7189119e-36,
        0.0000000e+00, 0.0000000e+00],
       [1.0125227e-37, 1.8738077e-06, 5.0258910e-14, 9.9894124e-01,
        6.1554196e-14, 1.0568369e-03, 0.0000000e+00, 3.0629898e-19,
        9.8306376e-09, 1.8770009e-13],
       [1.6821302e-31, 0.0000000e+00, 1.3149245e-18, 0.0000000e+00,
        1.0000000e+00, 5.4925100e-28, 5.1042579e-27, 2.4983566e-32,
        0.0000000e+00, 2.0557689e-22],
       [1.6974658e-30, 5.1224202e-18, 2.0515924e-15, 2.3811742e-07,
        6.0471929e-31, 9.9999976e-01, 1.3504103e-20, 2.4093161e-11,
        1.5848093e-08, 4.8414933e-25]], dtype=float32)
```

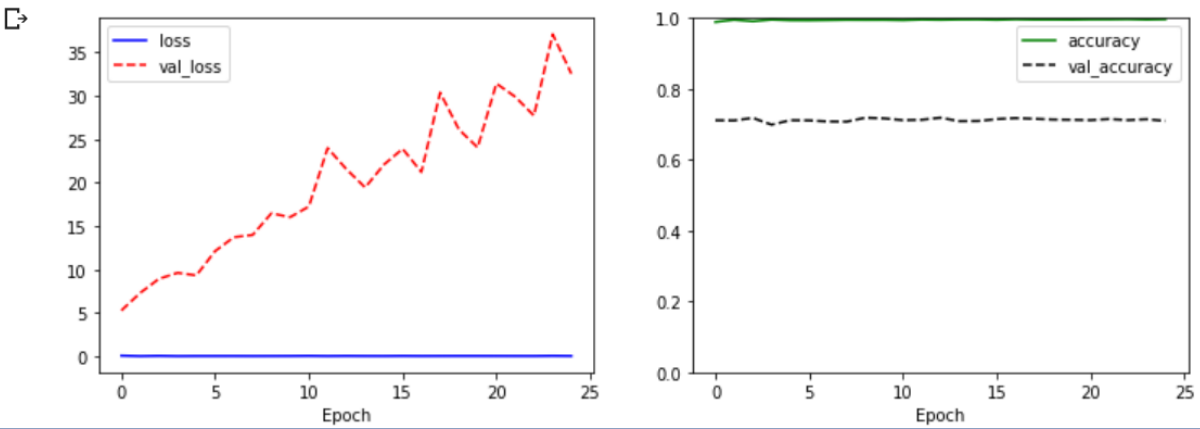
```
model.evaluate(X,Y)
```

```
1/1 [==============================] - 0s 2ms/step - loss: 2.1191e-04 - accuracy: 1.0000
[0.0002119134005624801, 1.0]
```

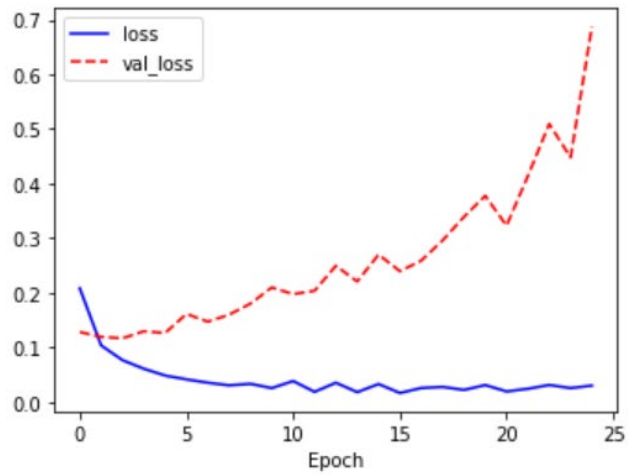
③위 ①번에 예제6.15 augment_size = 10000 으로 하여 학습이미지를 70000장으로 늘리고 image_generator를 적용한 학습을 시킨다. 그 결과를 ①번과 비교한다. -->비교설명과 코드와 이미지를 캡쳐하여 제출

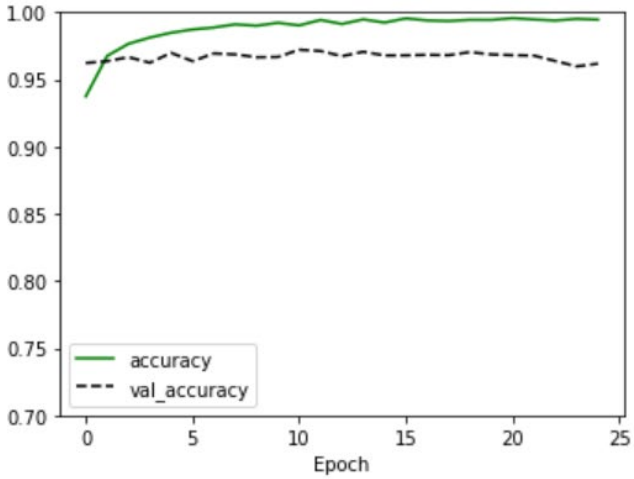
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

image_generator=ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.10,
    shear_range=0.5,
    width_shift_range=0.10,
    height_shift_range=0.10,
    horizontal_flip=True,
    vertical_flip=False)

augment_size=10000

randidx = np.random.randint(train_X.shape[0], size=augment_size)
x_augmented = train_X[randidx].copy()
y_augmented = train_Y[randidx].copy()
x_augmented = image_generator.flow(x_augmented, np.zeros(augment_size),
                                   batch_size=augment_size, shuffle=False).next()[0]
train_X = np.concatenate((train_X, x_augmented))
train_Y = np.concatenate((train_Y, y_augmented))

history=model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0, 1)
plt.legend()

plt.show()
```



CNN적용 - ①번

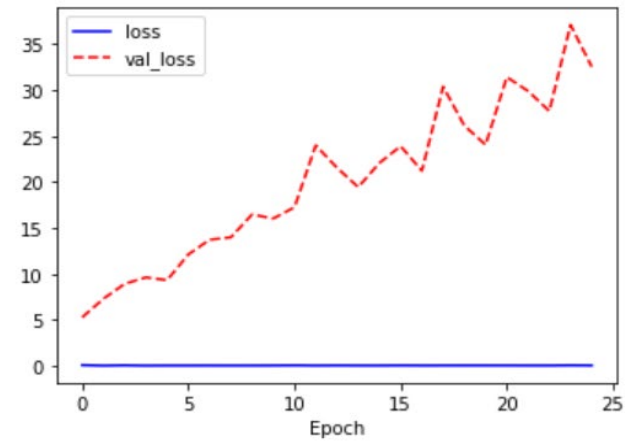


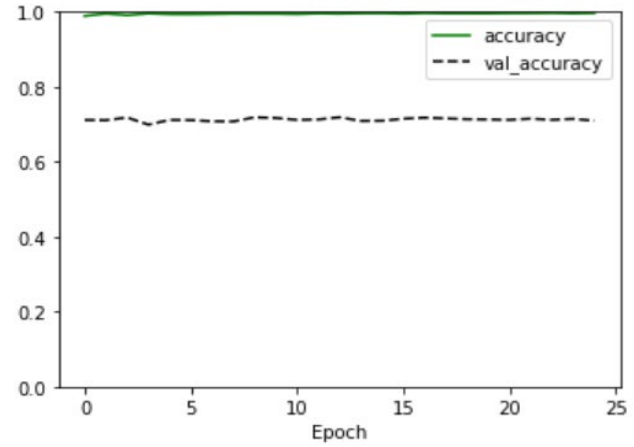



```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.6283 - accuracy: 0.9640
[0.6283432841300964, 0.9639999866485596]
```

CNN적용 - image_generator를 적용한 학습






```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.8767 - accuracy: 0.9729
[0.8766946196556091, 0.9728999733924866]
```

image_generator를 적용한 학습이 결과가 더 않좋다. 우선 왼쪽그래프를 보게되면 빨간색 val_loss그래프 모양이 비슷해보이지만 y축 숫자 단위 차이가 매우 크다. 이런 결과가 나온 이유는 이미 1번에서 과적합 현상이 나타났다. 근데 풀링 레이어나 드롭아웃 레이어를 사용해 과적합 현상을 개선하지도 않고, image_generator를 통해서 학습데이터량을 6만장에서 1만장 추가한 7만장으로 늘리는 등 과적합 현상을 더 악화시키게 했기 때문이다.
그래서 오차는 더 늘어났다.

학습을 개선하기위해서는 과적합을 줄이기위해 풀링레이어, 드롭아웃레이어를 사용하고 더 많은 레이어를 쌓는 것이다.

④위 ③번에 문제1> ④번의 테스트 이미지 5장을 사용하여 accuracy를 구하여 제출

```
model.predict(X)
```

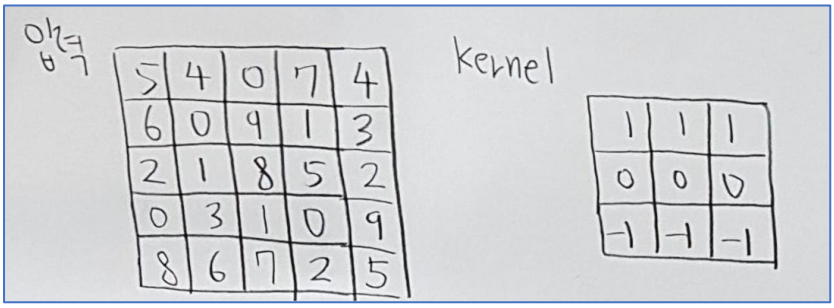
```
array([[0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [0.0000000e+00, 4.8580356e-10, 5.9290233e-38, 9.9910992e-01,
        2.9328409e-38, 8.9009438e-04, 0.0000000e+00, 0.0000000e+00,
        3.6362105e-10, 1.2250500e-16],
       [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [0.0000000e+00, 5.1253164e-32, 9.1270244e-28, 1.8674404e-15,
        0.0000000e+00, 9.9999988e-01, 0.0000000e+00, 8.2698715e-27,
        9.7631542e-08, 0.0000000e+00]], dtype=float32)
```

```
model.evaluate(X,Y)
```

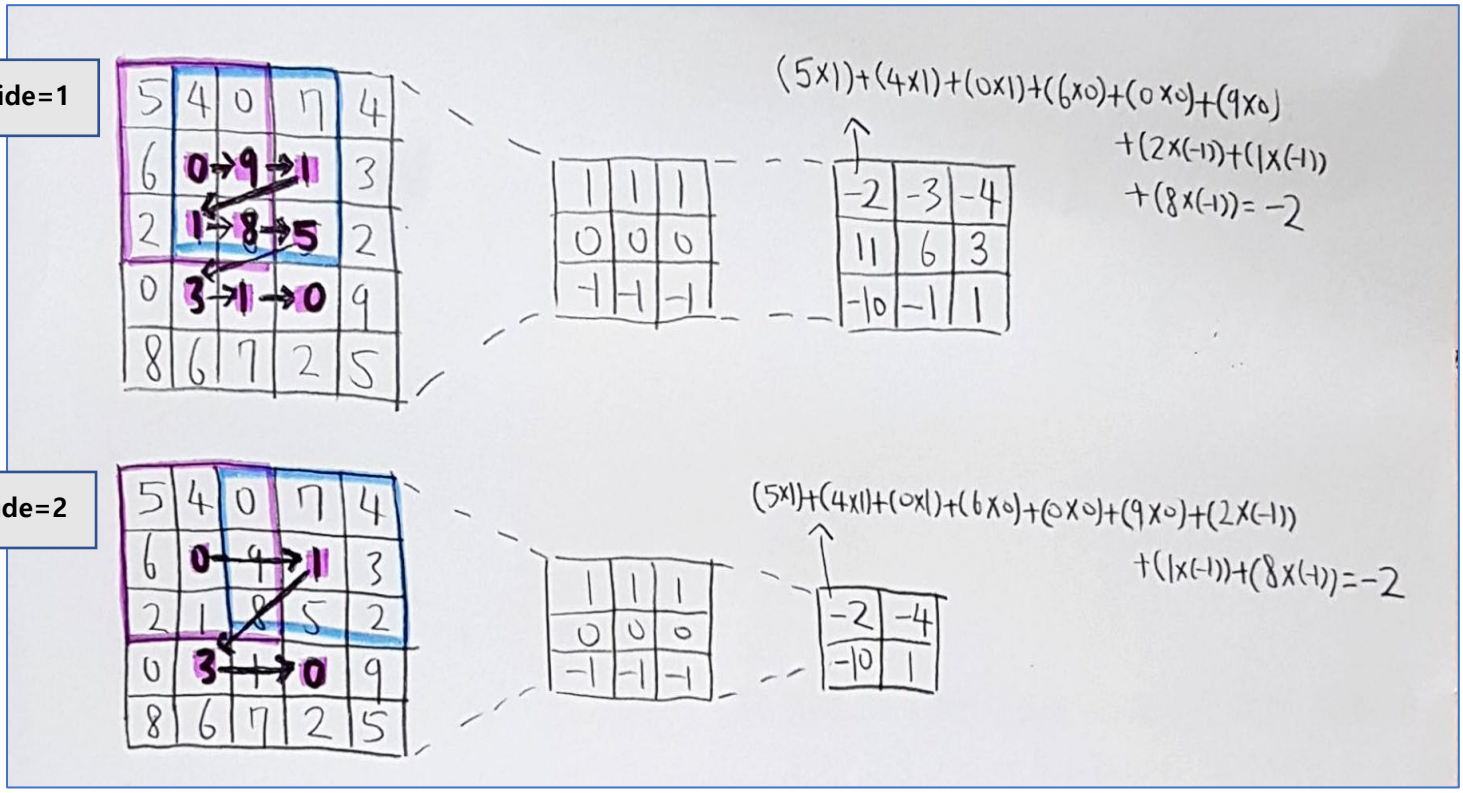
```
1/1 [==============================] - 0s 2ms/step - loss: 1.7811e-04 - accuracy: 1.0000
[0.00017811472935136408, 1.0]
```

문제 3> CNN구조를 이해하기 위한 내용이다.

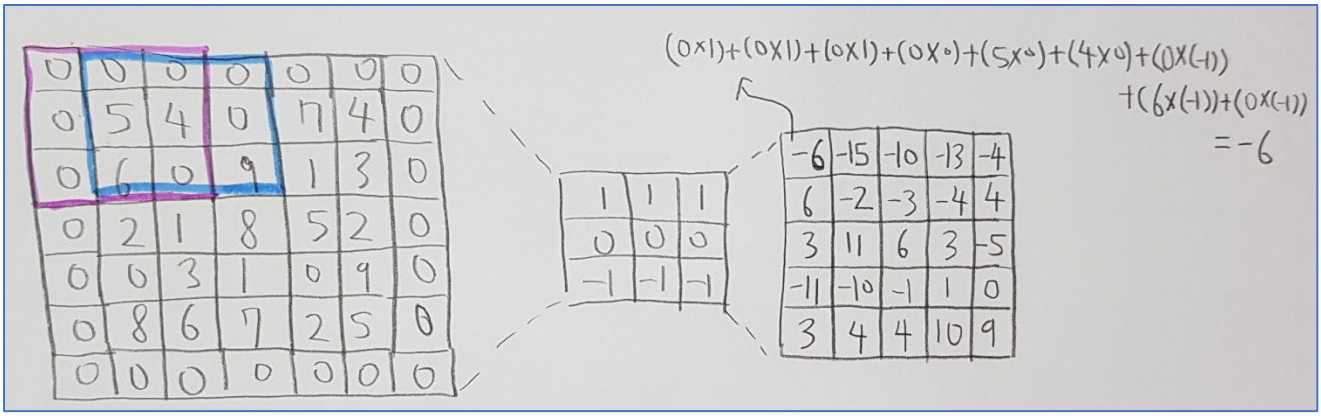
입력 크기 = 5 x 5, kernel 크기 = 3 x 3 으로 하고 행렬의 값은 임의의 정수로 정한다.(교재의 그림6.10의 값과 다르게 사용해야 함)



①stride=1, 2 두가지 경우에 대하여 convolution 연산을 하고 그 과정을 그림 6.10처럼 그려서 제출한다. (계산 과정은 첫 번째 연산만 그림처럼 보인다)
이때, padding은 하지 않는다.



②zero padding을 하여 convolution 연산을 하고(stride=1) 그림6.11처럼 그려서 제출한다. (계산 과정은 첫 번째 연산만 그림처럼 보인다)



③위 ②의 결과를 max pooling한 결과를 그림6.12처럼 그려서 제출한다. (계산 과정은 첫 번째 연산만 그림처럼 보인다)