HW10 Report

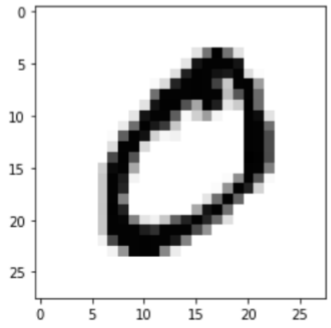
Computer Vision
Section 02
21800181 Kim Jisu

HW 10

- Result

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
from matplotlib import pyplot as plt

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
#x_train.shape
# type(x_train[0,0,0])
x_train = x_train.astype('float32') / 255.
n=1
plt.imshow(x_train[n], cmap='Greys', interpolation='nearest')
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```




```
[2] x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
    #x_train.shape
    x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
    input_shape = (28, 28, 1)
    #input_shape
    y_train[0:10]
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

```
[3] num_classes = 10
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)
    y_train[0:10]
```

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)
```

```
[4] import sys
    import tensorflow as tf
    import keras
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Flatten
    from keras.layers.convolutional import Conv2D, MaxPooling2D
    import numpy as np
    np.random.seed(7)
```

```
[5] model = Sequential()
    model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), padding='same',
                     activation='relu',
                     input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1000, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
```

```
[6] model = Sequential()
    model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), padding='same',
                     activation='relu',
                     input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

```
[7] model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

```
[8] model.add(Flatten())
    model.add(Dense(1000, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
```

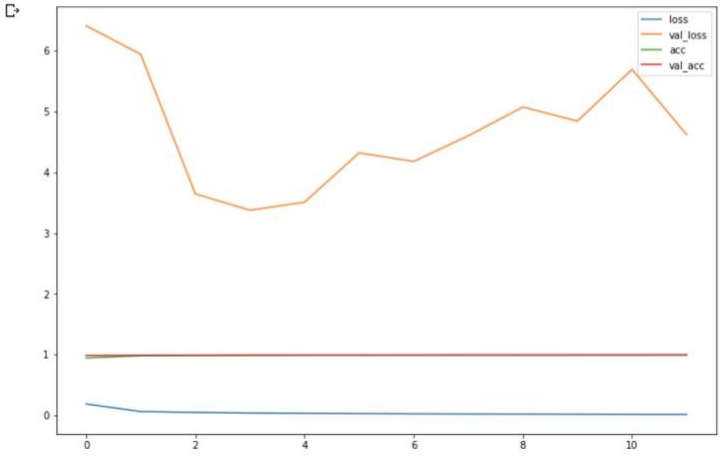
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 32)        832

max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)        0
2D)

conv2d_3 (Conv2D)            (None, 14, 14, 64)        8256

max_pooling2d_3 (MaxPooling  (None, 7, 7, 64)          0
2D)

dropout_2 (Dropout)          (None, 7, 7, 64)          0

flatten_1 (Flatten)          (None, 3136)              0

dense_2 (Dense)              (None, 1000)              3137000

dropout_3 (Dropout)          (None, 1000)              0

dense_3 (Dense)              (None, 10)                10010

=================================================================
Total params: 3,156,098
Trainable params: 3,156,098
Non-trainable params: 0
_________________________________________________________________
```

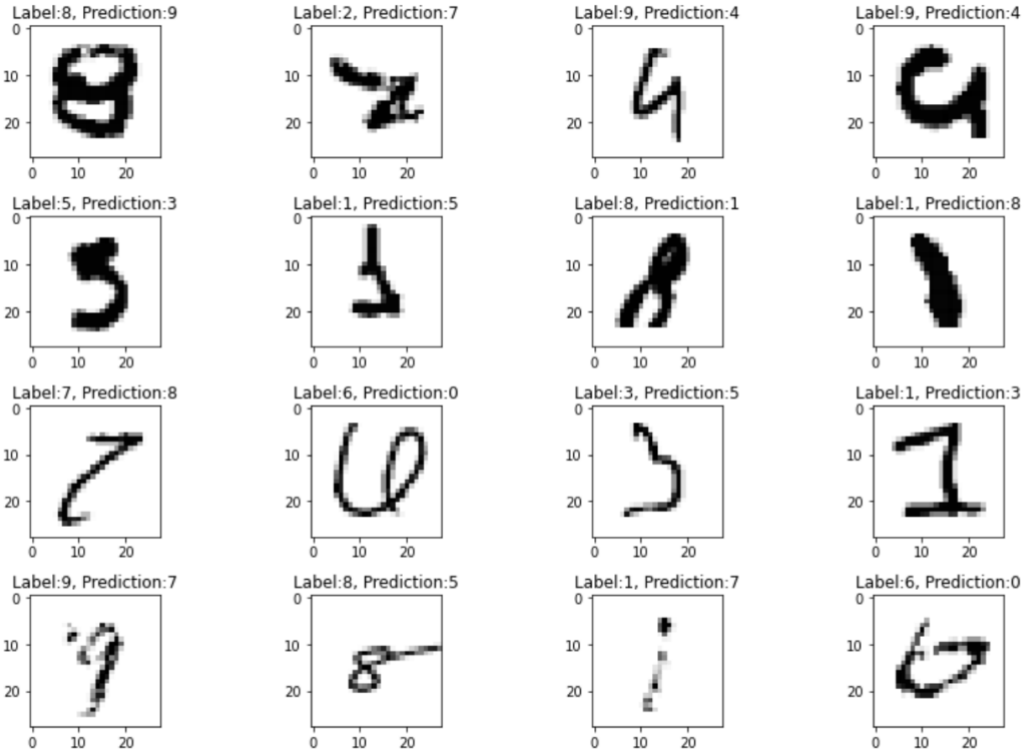
```
batch_size = 128
epochs = 12
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
hist = model.fit(x_train, y_train,
batch_size=batch_size, epochs=epochs,
verbose=1, validation_data=(x_test, y_test))
```

```
Epoch 1/12
469/469 [==============================] - 90s 191ms/step - loss: 0.1865 - accuracy: 0.9425 - val_loss: 6.4107 - val_accuracy: 0.9854
Epoch 2/12
469/469 [==============================] - 89s 191ms/step - loss: 0.0616 - accuracy: 0.9808 - val_loss: 5.9421 - val_accuracy: 0.9873
Epoch 3/12
469/469 [==============================] - 89s 190ms/step - loss: 0.0472 - accuracy: 0.9854 - val_loss: 3.6443 - val_accuracy: 0.9905
Epoch 4/12
469/469 [==============================] - 89s 190ms/step - loss: 0.0374 - accuracy: 0.9880 - val_loss: 3.3751 - val_accuracy: 0.9926
Epoch 5/12
469/469 [==============================] - 89s 189ms/step - loss: 0.0322 - accuracy: 0.9897 - val_loss: 3.5066 - val_accuracy: 0.9931
Epoch 6/12
469/469 [==============================] - 89s 189ms/step - loss: 0.0269 - accuracy: 0.9914 - val_loss: 4.3183 - val_accuracy: 0.9922
Epoch 7/12
469/469 [==============================] - 89s 190ms/step - loss: 0.0237 - accuracy: 0.9922 - val_loss: 4.1772 - val_accuracy: 0.9925
Epoch 8/12
469/469 [==============================] - 88s 188ms/step - loss: 0.0210 - accuracy: 0.9933 - val_loss: 4.5999 - val_accuracy: 0.9929
Epoch 9/12
469/469 [==============================] - 87s 186ms/step - loss: 0.0201 - accuracy: 0.9936 - val_loss: 5.0729 - val_accuracy: 0.9924
Epoch 10/12
469/469 [==============================] - 88s 188ms/step - loss: 0.0184 - accuracy: 0.9938 - val_loss: 4.8420 - val_accuracy: 0.9927
Epoch 11/12
469/469 [==============================] - 89s 189ms/step - loss: 0.0148 - accuracy: 0.9953 - val_loss: 5.6930 - val_accuracy: 0.9920
Epoch 12/12
469/469 [==============================] - 88s 188ms/step - loss: 0.0145 - accuracy: 0.9954 - val_loss: 4.6198 - val_accuracy: 0.9932
```

```
plt.figure(figsize=(12,8))
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.legend(['loss','val_loss', 'acc','val_acc'])
plt.show()
```




```
import random
predicted_result = model.predict(x_test)
predicted_labels = np.argmax(predicted_result, axis=1)
test_labels = np.argmax(y_test, axis=1)
wrong_result = []
for n in range(0, len(test_labels)):
  if predicted_labels[n] != test_labels[n]:
    wrong_result.append(n)
samples = random.choices(population=wrong_result, k=16)
count = 0
nrows = ncols = 4
plt.figure(figsize=(12,8))
for n in samples:
  count += 1
  plt.subplot(nrows, ncols, count)
  plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
  tmp = "Label:" + str(test_labels[n]) + ", Prediction:" + str(predicted_labels[n])
  plt.title(tmp)
plt.tight_layout()
plt.show()
```


Label:8, Prediction:9
Label:2, Prediction:7
Label:9, Prediction:4
Label:9, Prediction:4
Label:5, Prediction:3
Label:1, Prediction:5
Label:8, Prediction:1
Label:1, Prediction:8
Label:7, Prediction:8
Label:6, Prediction:0
Label:3, Prediction:5
Label:1, Prediction:3
Label:9, Prediction:7
Label:8, Prediction:5
Label:1, Prediction:7
Label:6, Prediction:0