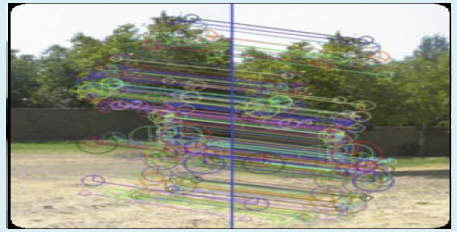
Deep Learning Basics
(#xx: Keras-based Convolutional Neural Network Practice-Part1)

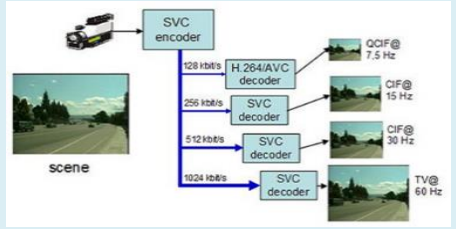
2023 Autumn

Prof. Byung-Gyu Kim
Intelligent Vision Processing Lab. (IVPL)
http://ivpl.sookmyung.ac.kr
Dept. of IT Engineering, Sookmyung Women's University
E-mail: bg.kim@ivpl.sookmyung.ac.kr

Gaol of this lecture

- Understand structure and how to develop my Convolutional Neural Network (CNN)
  - MNIST-based exercises: Digits Recognition



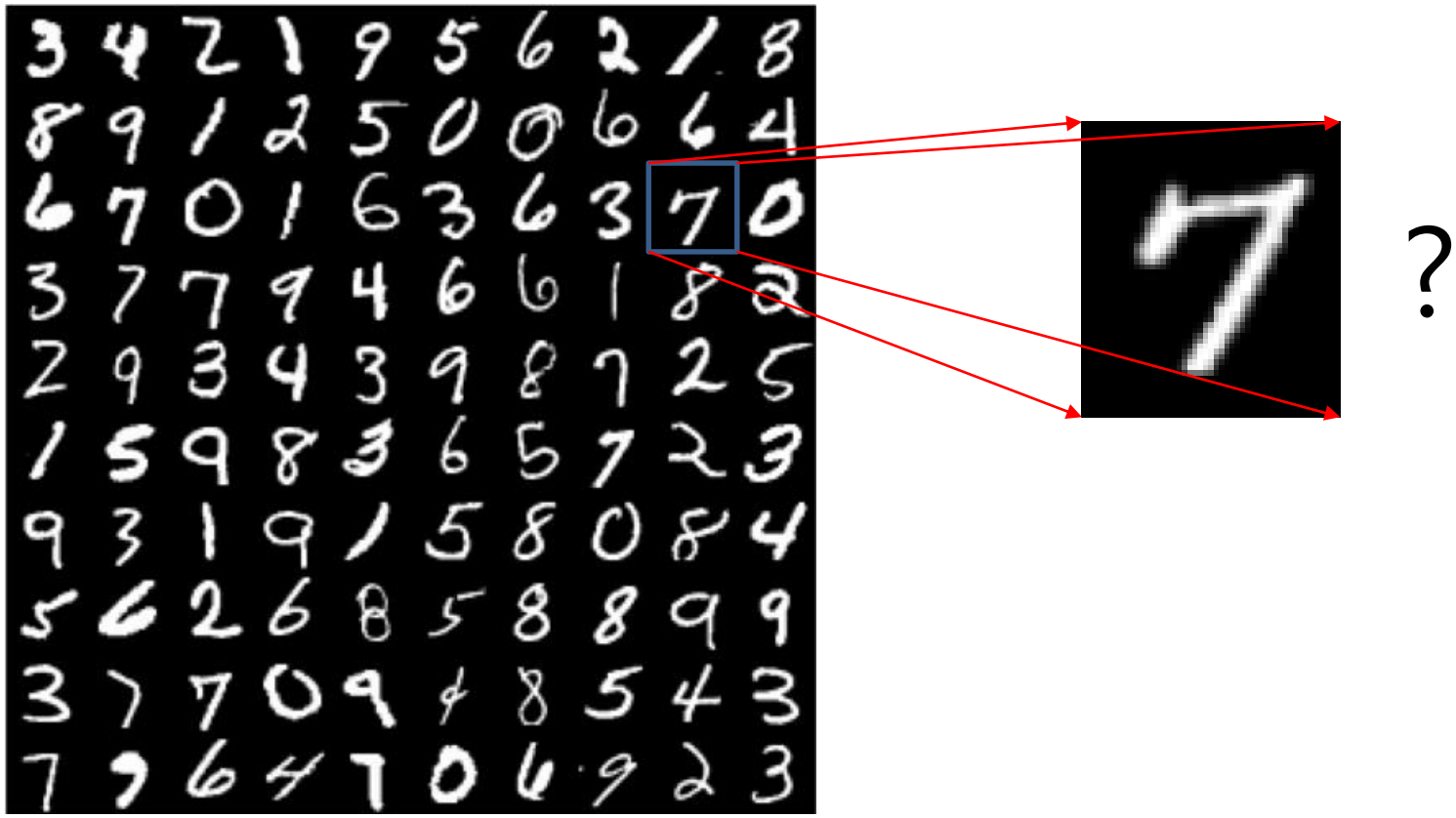


Contents

- **CNN implementation and its structure**

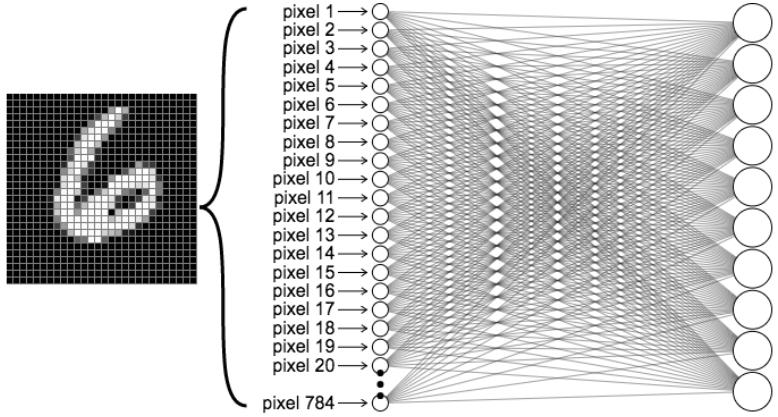
Keras Convolutional Neural Network Tutorial: MNIST (1)

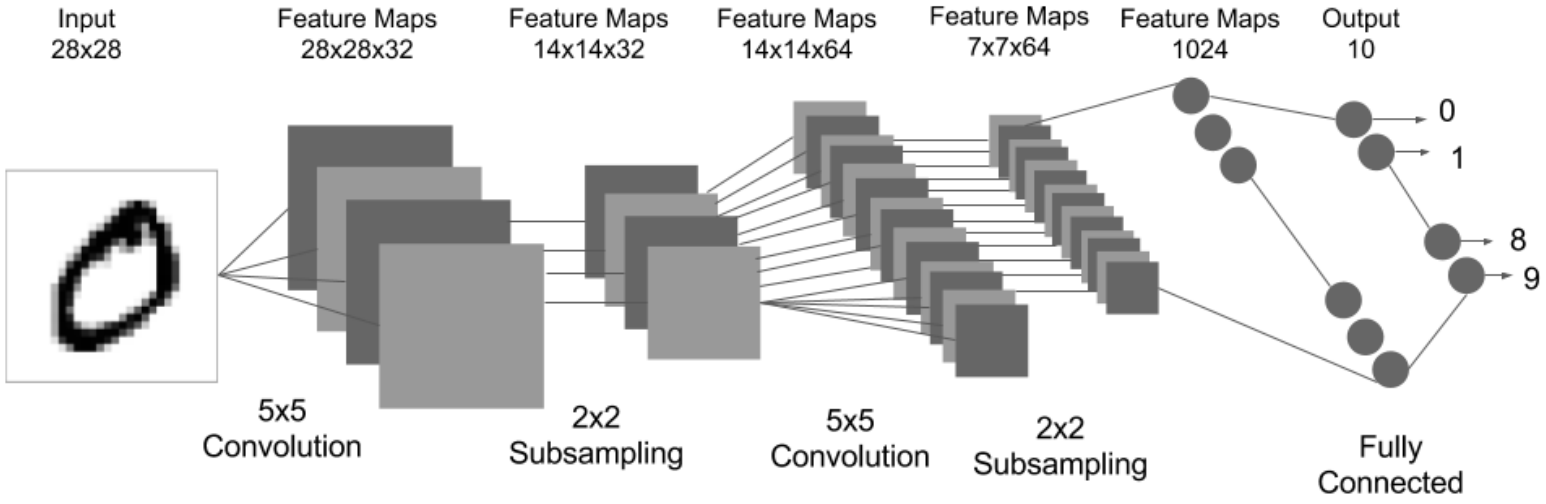
- Digits Recognition: "Just like programming has Hello World, machine learning has MNIST".
  - Goal: Handwritten Digit Prediction (Recognition) using Convolutional Neural Networks



Keras Convolutional Neural Network Tutorial: MNIST (2)

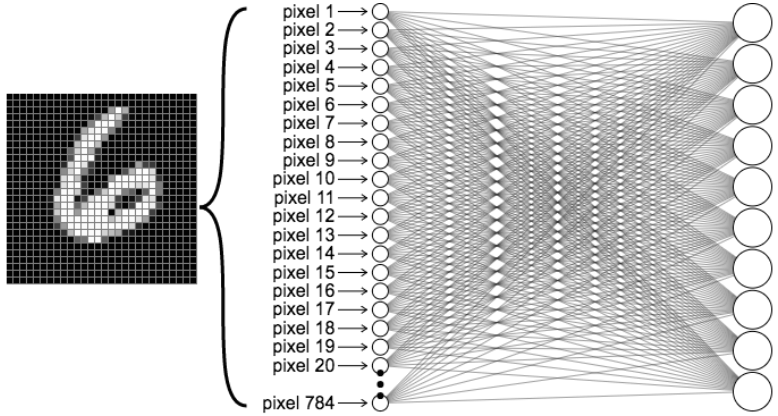
❖ Network Structure

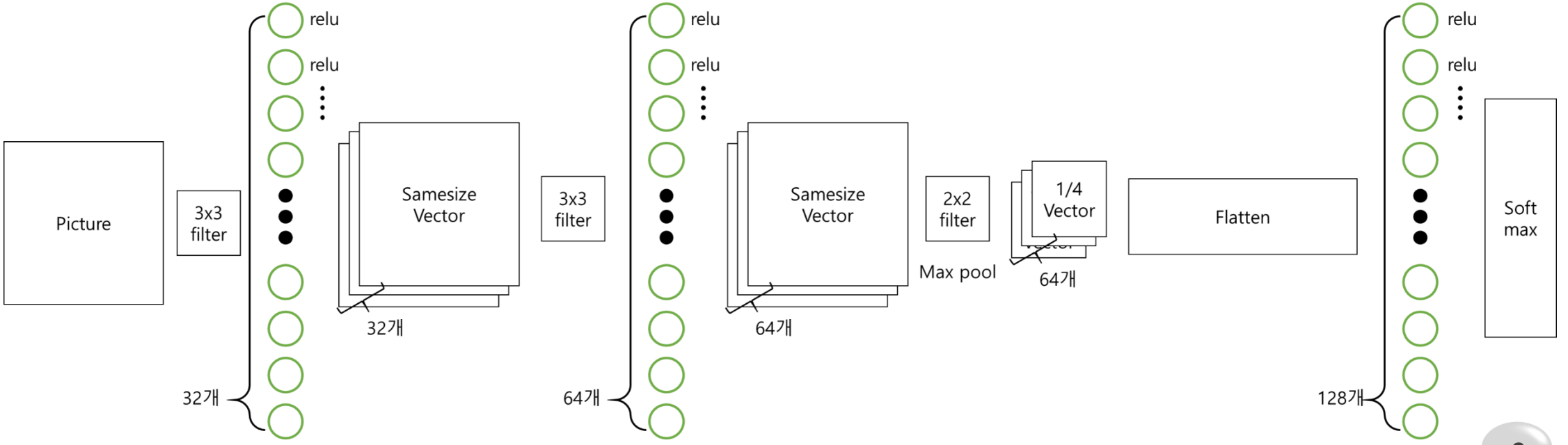




Keras Convolutional Neural Network Tutorial: MNIST (3)

❖ Network Structure





Keras Convolutional Neural Network Tutorial: MNIST (4)

❖ Install some packages

- **Pillow** package
  - Pillow is the friendly PIL fork (by Alex Clark and Contributors). PIL is **the Python Imaging Library** (by Fredrik Lundh and Contributors). https://pillow.readthedocs.io/en/stable/

(BGKim) C:\Users\vicl>pip3 install pillow

```
(BGKim) C:₩Users₩vicl>pip3 install pillow
Collecting pillow
  Downloading https://files.pythonhosted.org/packages/ae/96/6f83deebfcd20a5d4ad35e4e989814a16559d8715741457e670aae1a5a09/Pillow-6.1.0-cp3
7-cp37m-win_amd64.whl (2.0MB)
     |████████████████████████████████| 2.0MB 930kB/s
Installing collected packages: pillow
Successfully installed pillow-6.1.0

(BGKim) C:₩Users₩vicl>cd practices

(BGKim) C:₩Users₩vicl₩practices>cd cnn

(BGKim) C:₩Users₩vicl₩practices₩cnn>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 5417-ADDA

 C:₩Users₩vicl₩practices₩cnn 디렉터리

2019-07-26  오후 12:05    <DIR>          .
2019-07-26  오후 12:05    <DIR>          ..
2019-07-26  오후 12:05             2,676 mnist_cnn.py
               1개 파일               2,676 바이트
               2개 디렉터리  216,548,757,504 바이트 남음
```

Keras Convolutional Neural Network Tutorial: MNIST (5)

❖ CNN Implementation (editor: Visual Studio)

- Dataset Preparation
  - Directly download from server using API in source code

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Keras Convolutional Neural Network Tutorial: MNIST (6)

- CNN Implementation

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,
Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.utils import np_utils
from PIL import Image
import numpy as np
import os

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28
```

(continue)

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows,
img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols,
1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples' )
```

(continue)

Keras Convolutional Neural Network Tutorial: MNIST (7)

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train,
num_classes)
y_test = keras.utils.to_categorical(y_test,
num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation =
'softmax' ))

      (contiue)
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Keras Convolutional Neural Network Tutorial: MNIST (8)

- Run this CNN code…!!! (epoch = 12) → Accuracy is almost 99.1%.

```
2019-07-26 12:23:59.923344: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1640] Found device 0 with properties:
name: GeForce GTX 1070 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683
pciBusID: 0000:01:00.0
2019-07-26 12:23:59.930003: I tensorflow/stream_executor/platform/default/dlopen_checker_stub.cc:25] GPU libraries are statically linked,
 skip dlopen check.
2019-07-26 12:23:59.937206: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1763] Adding visible gpu devices: 0
2019-07-26 12:24:00.510224: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1181] Device interconnect StreamExecutor with strength 1 e
dge matrix:
2019-07-26 12:24:00.514384: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1187]      0
2019-07-26 12:24:00.516691: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1200] 0:   N
2019-07-26 12:24:00.523133: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1326] Created TensorFlow device (/job:localhost/replica:0/
task:0/device:GPU:0 with 6357 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1070 Ti, pci bus id: 0000:01:00.0, compute capabil
ity: 6.1)
60000/60000 [==============================] - 7s 118us/step - loss: 0.2673 - acc: 0.9177 - val_loss: 0.0574 - val_acc: 0.9808
Epoch 2/12
60000/60000 [==============================] - 5s 80us/step - loss: 0.0886 - acc: 0.9735 - val_loss: 0.0400 - val_acc: 0.9857
Epoch 3/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0652 - acc: 0.9806 - val_loss: 0.0327 - val_acc: 0.9887
Epoch 4/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0553 - acc: 0.9831 - val_loss: 0.0325 - val_acc: 0.9890
Epoch 5/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0478 - acc: 0.9848 - val_loss: 0.0302 - val_acc: 0.9895
Epoch 6/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0404 - acc: 0.9877 - val_loss: 0.0267 - val_acc: 0.9910
Epoch 7/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0378 - acc: 0.9885 - val_loss: 0.0265 - val_acc: 0.9913
Epoch 8/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0339 - acc: 0.9900 - val_loss: 0.0267 - val_acc: 0.9909
Epoch 9/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0323 - acc: 0.9902 - val_loss: 0.0287 - val_acc: 0.9908
Epoch 10/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0297 - acc: 0.9908 - val_loss: 0.0294 - val_acc: 0.9905
Epoch 11/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0260 - acc: 0.9920 - val_loss: 0.0278 - val_acc: 0.9916
Epoch 12/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0262 - acc: 0.9916 - val_loss: 0.0279 - val_acc: 0.9910
Test loss: 0.027891269745134558
Test accuracy: 0.991
```

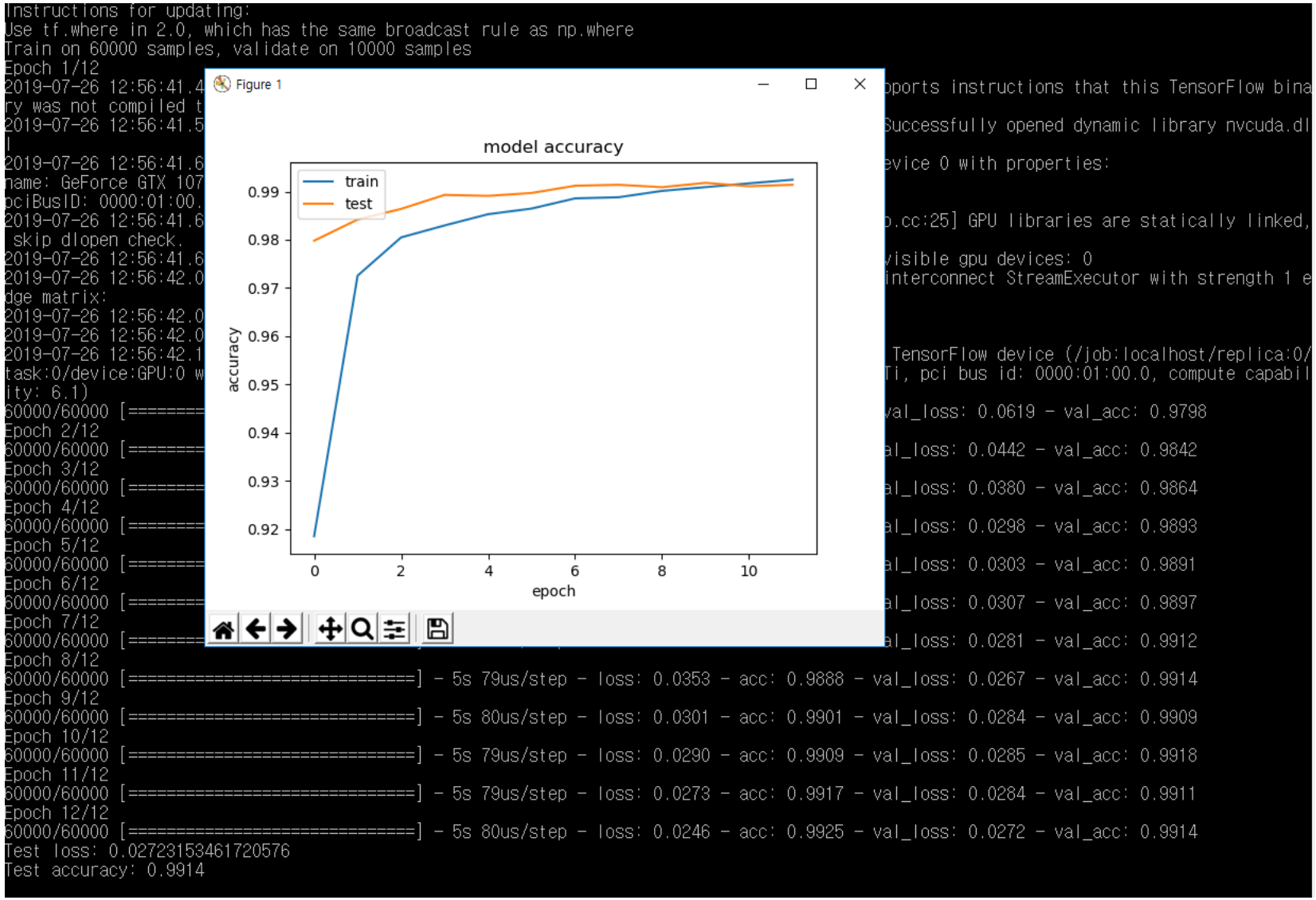
Keras Convolutional Neural Network Tutorial: MNIST (9)

❖ Before studying APIs,

If you want to see the trained result as figure, how to change the code?

Keras Convolutional Neural Network Tutorial: MNIST (10)

- Accuracy is almost 99.1% **as output graph**...!!!



Keras Convolutional Neural Network Tutorial: MNIST (11)

❖ APIs

- Data.reshape(args)
  - Change the shape and appearance of the given data

```
x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
```

Keras Convolutional Neural Network Tutorial: MNIST (11-1)

- Change the shape and appearance of the given data

```
x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
```

```
# as first layer in a Sequential model
model = Sequential()
model.add(Reshape((3, 4), input_shape=(12,)))
# now: model.output_shape == (None, 3, 4)
# note: `None` is the batch dimension # as intermediate layer in a Sequential model

model.add(Reshape((6, 2)))
# now: model.output_shape == (None, 6, 2)

# also supports  automatic shape inference using `-1` as dimension
model.add(Reshape((-1, 2, 2)))
# now: model.output_shape == (None, 3, 2, 2)
```

Keras Convolutional Neural Network Tutorial: MNIST (11-2)

- Data.astype(args)
  - Change the data type to the specified one.

```
x_train = x_train.astype('float32')
```

- keras.utils.to_categorical(args)
  - Converts a class vector (integers) to binary class matrix.

<Arguments>

- **y**: class vector to be converted into a matrix (integers from 0 to num_classes).
- **num_classes**: total number of classes.
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)

Keras Convolutional Neural Network Tutorial: MNIST (11-3)

```
# Consider an array of 5 labels out of a set of 3 classes {0, 1, 2}:
> labels array([0, 2, 1, 2, 0])
# `to_categorical` converts this into a matrix with as many
# columns as there are classes. The number of rows
# stays the same.
> to_categorical(labels)
array([[ 1., 0., 0.],
       [ 0., 0., 1.],
       [ 0., 1., 0.],
       [ 0., 0., 1.],
       [ 1., 0., 0.]], dtype=float32)
```

- Layers.Conv2D(args)
  - 2D convolution layer (e.g. spatial convolution over images).

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,     dilation_rate=(1, 1),
activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
bias_constraint=None)
```

Keras Convolutional Neural Network Tutorial: MNIST (11-4)

- Layers.Conv2D(args)

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=**None**, dilation_rate=(1, 1), activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

This layer creates a convolution kernel that is convolved **with the layer input to produce a tensor of outputs.**
If use_bias is True, a bias vector is created and added to the outputs.
Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument input_shape(tuple of integers, does not include the batch axis), e.g. input_shape=(128, 128, 3) for 128x128 RGB pictures in data_format="channels_last".

<Arguments>

- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding**: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with strides != 1, as described here

Keras Convolutional Neural Network Tutorial: MNIST (11-5)

• **kernel_regularizer**: Regularizer function applied to the kernel weights matrix (see regularizer).
• **data_format**: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
• **dilation_rate**: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
• **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
• **use_bias**: Boolean, whether the layer uses a bias vector.
• **kernel_initializer**: Initializer for the kernel weights matrix (see initializers).
• **bias_initializer**: Initializer for the bias vector (see initializers).
• **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
• **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
• **kernel_constraint**: Constraint function applied to the kernel matrix (see constraints).
• **bias_constraint**: Constraint function applied to the bias vector (see constraints).

https://keras.io/layers/convolutional/#conv2d

Keras Convolutional Neural Network Tutorial: MNIST (11-6)

- Keras.layers.**MaxPooling2D(agrs)**
  - Max pooling operation for spatial data (pick the max value as subsampling).

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

<Arguments>

- **pool_size**: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
- **strides**: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.
- **padding**: One of "valid" or "same" (case-insensitive).
- **data_format**: A string, one of channels_last (default) or channels_first.

```
Ex)) model.add(MaxPooling2D(pool_size=(2, 2)))
```

Keras Convolutional Neural Network Tutorial: MNIST (11-7)

- Keras.layers.**Dropout(agrs)**
  - Applies Dropout to the input.
  - Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps **prevent overfitting**.

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

<Arguments>

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise_shape=(batch_size, 1, features).

```
Ex)) model.add(Dropout(0.25))
```

Keras Convolutional Neural Network Tutorial: MNIST (11-8)

- Keras.layers.**Flatten(agrs)**
  - Flattens the input. Does not affect the batch size.

keras.layers.Flatten(data_format=None)

<Arguments>

- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. The purpose of this argument is to preserve weight ordering when switching a model from one data format to another. channels_last corresponds to inputs with shape (batch, ..., channels) while channels_first corresponds to inputs with shape (batch, channels, ...). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Ex)) model.add(Flatten())

```
model = Sequential()
model.add(Conv2D(64, (3, 3),
                 input_shape=(3, 32, 32), padding='same',))
# now: model.output_shape == (None, 64, 32, 32)

model.add(Flatten())
# now: model.output_shape == (None, 65536)
```

Keras Convolutional Neural Network Tutorial: MNIST (11-9)

- Keras.model.**evaluate(agrs)**
  - **Returns the loss value & metrics values for the model** in test mode (evaluation for performance).

keras.model.evaluate(x=**None**, y=**None**, batch_size=**None**, verbose=1, sample_weight=**None**, steps=**None**, callbacks=**None**)

<Arguments>

- **x**: Numpy array of test data (if the model has a single input), or list of Numpy arrays (if the model has multiple inputs). If input layers in the model are named, you can also pass a dictionary mapping input names to Numpy arrays. x can be None (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).
- **y**: Numpy array of target (label) data (if the model has a single output), or list of Numpy arrays (if the model has multiple outputs). If output layers in the model are named, you can also pass a dictionary mapping output names to Numpy arrays. y can be None (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).
- **batch_size**: Integer or None. Number of samples per evaluation step. If unspecified, batch_sizewill default to 32.
- **verbose**: 0 or 1. Verbosity mode. 0 = silent, 1 = progress bar.
- **sample_weight**: Optional Numpy array of weights for the test samples, used for weighting the loss function. You can either pass a flat (1D) Numpy array with the same length as the input samples (1:1 mapping between weights and samples), or in the case of temporal data, you can pass a 2D array with shape (samples, sequence_length), to apply a different weight to every timestep of every sample. In this case you should make sure to specifysample_weight_mode="temporal" in compile().
- **steps**: Integer or None. Total number of steps (batches of samples) before declaring the evaluation round finished. Ignored with the default value of None.
- **callbacks**: List of keras.callbacks.Callback instances. List of callbacks to apply during evaluation. See callbacks.

<Returns>

Scalar test loss (if the model has a single output and no metrics) or list of scalars (if the model has multiple outputs and/or metrics). The attribute model.metrics_names will give you the display labels for the scalar outputs.

Keras Convolutional Neural Network Tutorial: MNIST (11-10)

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

- Keras.model.**predict**(args)
  - Give output predictions for the input samples.

```
predict(x, batch_size=None, verbose=0, steps=None, callbacks=None)
```

<Arguments>

- **x**: The input data, as a Numpy array (or list of Numpy arrays if the model has multiple inputs).
- **batch_size**: Integer. If unspecified, it will default to 32.
- **verbose**: Verbosity mode, 0 or 1.
- **steps**: Total number of steps (batches of samples) before declaring the prediction round finished. Ignored with the default value of None.
- **callbacks**: List of keras.callbacks.Callback instances. List of callbacks to apply during prediction. See callbacks.

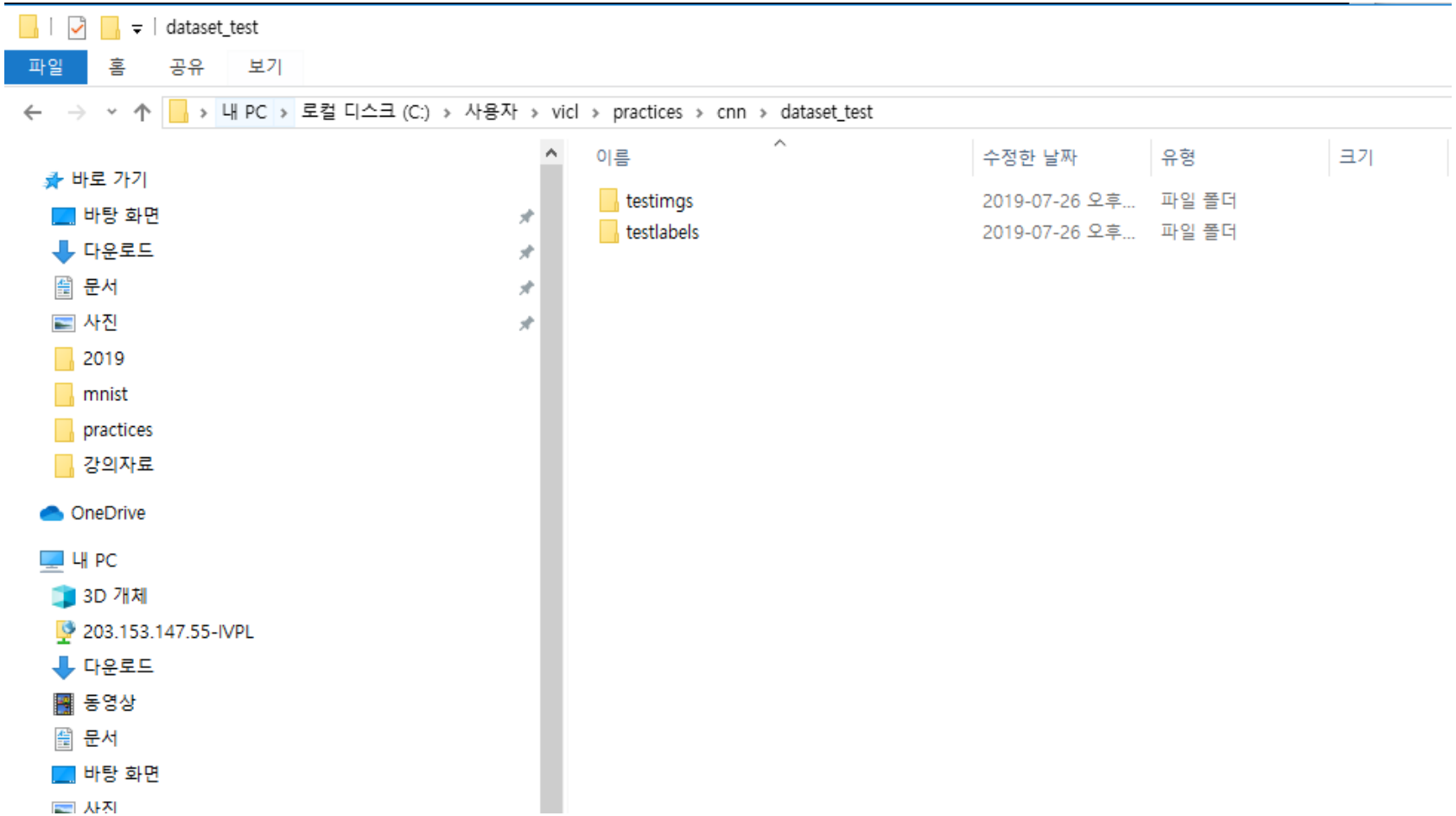
<Returns>

Numpy array(s) of predictions.

Keras Convolutional Neural Network Tutorial: MNIST (12)

❖ Test for trained model

- Prepare your test dataset…!!!! (from MNIST)

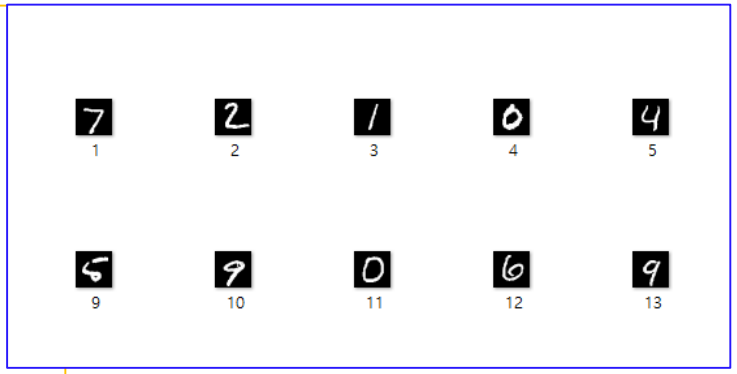


Keras Convolutional Neural Network Tutorial: MNIST (13)

- Add your code for testing the recognition as:

```
##-- Model Test using Test datasets
print()
print("----Actual test for digits----")
img = Image.open('dataset_test/testimgs/1.png').convert("L")
img = np.resize(img, (28,28,1))
im2arr = np.array(img)
im2arr = im2arr.reshape(1,28,28,1)
y_pred = model.predict_classes(im2arr)
print(y_pred)

img = Image.open('dataset_test/testimgs/5.png').convert("L")
img = np.resize(img, (28,28,1))
im2arr = np.array(img)
im2arr = im2arr.reshape(1,28,28,1)
y_pred = model.predict_classes(im2arr)
print(y_pred)
```



Keras Convolutional Neural Network Tutorial: MNIST (14)

- Run the changed “mnist_cnn.py”

```
60000/60000 [==============================] - 7s 118us/step - loss: 0.2677 - acc: 0.9169 - val_loss: 0.0569 - val_acc: 0.9803
Epoch 2/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0879 - acc: 0.9741 - val_loss: 0.0411 - val_acc: 0.9854
Epoch 3/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0669 - acc: 0.9795 - val_loss: 0.0333 - val_acc: 0.9892
Epoch 4/12
60000/60000 [==============================] - 5s 80us/step - loss: 0.0566 - acc: 0.9833 - val_loss: 0.0315 - val_acc: 0.9896
Epoch 5/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0477 - acc: 0.9857 - val_loss: 0.0280 - val_acc: 0.9897
Epoch 6/12
60000/60000 [==============================] - 5s 80us/step - loss: 0.0411 - acc: 0.9878 - val_loss: 0.0287 - val_acc: 0.9905
Epoch 7/12
60000/60000 [==============================] - 5s 77us/step - loss: 0.0383 - acc: 0.9889 - val_loss: 0.0276 - val_acc: 0.9906
Epoch 8/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0350 - acc: 0.9888 - val_loss: 0.0298 - val_acc: 0.9908
Epoch 9/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0314 - acc: 0.9904 - val_loss: 0.0272 - val_acc: 0.9910
Epoch 10/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0295 - acc: 0.9908 - val_loss: 0.0238 - val_acc: 0.9919
Epoch 11/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0273 - acc: 0.9916 - val_loss: 0.0265 - val_acc: 0.9913
Epoch 12/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0272 - acc: 0.9916 - val_loss: 0.0300 - val_acc: 0.9909
Test loss: 0.029996880742231O4
Test accuracy: 0.9909

----Actual test for digits----
[7]
[4]

(BGKim) C:₩Users₩vicl₩practices₩cnn>
```

Keras Convolutional Neural Network Tutorial: MNIST (15)

❖ How to test all test dataset or number of data that I want, for accuracy ratio?

- I can use "for" or "while"

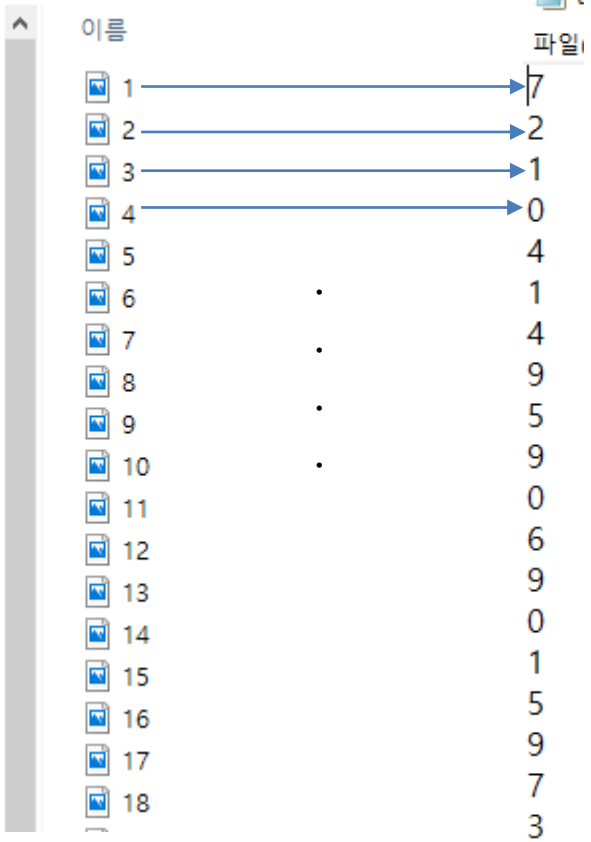
```
for index in range(10):
    img = Image.open('data/' + str(index) + '.png').convert("L")
    img = img.resize((28,28))
    im2arr = np.array(img)
    im2arr = im2arr.reshape(1,28,28,1)

    # Predicting the Test set results
    y_pred = model.predict(im2arr)
    print(y_pred)
```

Keras Convolutional Neural Network Tutorial: MNIST (16)

- I can use "for" or "while". **Here we have to load the label data too...!!!!**

Image data



Label data

Compare the prediction and its label data..!!

Keras Convolutional Neural Network Tutorial: MNIST (17)

```
##-- Model Test using Test datasets
print()
print("----Actual test for digits----")

mnist_label_file_path =
"dataset_test/testlabels/t_labels.txt"
mnist_label = open(mnist_label_file_path, "r")
cnt_correct = 0
for index in range(10):
    #-- read a label
    label = mnist_label.readline()
    print(label)
    #-- predict the test image (digit)
    img = Image.open('dataset_test/testimgs/' +
    str(index+1) + '.png').convert("L")
    img = img.resize((28,28))
    im2arr = np.array(img)
    im2arr = im2arr.reshape(1,28,28,1)

     (continue)
```

```
    # Predicting the Test set results
    #y_pred = model.predict(im2arr)#<-- [0,0 .....1. 0.]
    y_pred = model.predict_classes(im2arr)#<-- 7 or 4
    print(y_pred)

    #-- compute the accuracy of the preditcion
    if int(label)==y_pred:
        cnt_correct += 1

#-- Final accuracy
Final_acc = cnt_correct/10
print()
print("Final test accuray: %f" %Final_acc)
```

Keras Convolutional Neural Network Tutorial: MNIST (18)

- Run the modified cnn code….!!! You can see the recognition accuracy in the last line.

```
60000/60000 [==============================] - 5s 78us/step - loss: 0.0263 - acc: 0.9920 - val_loss: 0.0274 - val_acc: 0.9918
Epoch 12/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0241 - acc: 0.9923 - val_loss: 0.0273 - val_acc: 0.9921
Test loss: 0.027300588260562017
Test accuracy: 0.9921

----Actual test for digits----
7

[7]
2

[2]
1

[1]
0

[0]
4

[4]
1

[1]
4

[4]
9

[9]
5

[5]
9

[9]
Final test accuray: 1.000000

(BGKim) C:₩Users₩vicl₩practices₩cnn>
```

Keras Convolutional Neural Network Tutorial: MNIST (19)-Save/Load Model

❖ How to **save the trained model**?

- Usually, after fitting the model, you can save your trained model.

```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200)

# Save the model
model.save('models/mnistCNN.h5')
```

❖ How to **load to memory from your saved model**?

```
from keras.models import load_model
model = load_model('models/mnistCNN.h5')
from PIL import Image
import numpy as np
for index in range(10):
    img = Image.open('data/' + str(index) + '.png').convert("L")
    img = img.resize((28,28))
    im2arr = np.array(img)
    im2arr = im2arr.reshape(1,28,28,1)
    # Predicting the Test set results
    y_pred = model.predict(im2arr)
    print(y_pred)
```

Thank you for your attention.!!!
QnA

http://ivpl.sookmyung.ac.kr