13. Deep Learning

- CNN(Convolutional Neural Network)
- Stride와 Pooling
- MNIST를 위한 CNN구현
- CNN의 사례연구(AlexNet, ResNet 등)

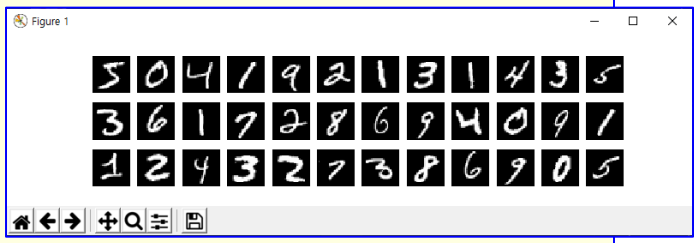
MNIST Dataset 98%이상 달성하기

- 구현방법
  - 대상 : MNIST Dataset
  - 레이어 구성
    - 4 layer
      - L1(784, 1024) + Dropout 0.3
      - L2(512, 1024) + Dropout 0.3
      - L3(512, 1024) + Dropout 0.3
      - L4(512, 10)
  - Activation Function : ReLU
  - Optimizer : Adam
  - Batch_size : 100, Epochs : 15

MNIST Dataset 98%이상 달성하기

- Dataset 정의

```
import tensorflow as tf
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# plt.figure(figsize=(8, 2))  # 8 x 2 inchs
# for i in range(36):
#     plt.subplot(3, 12, i+1)
#     plt.imshow(X_train[i], cmap="gray")
#     plt.axis("off")
# plt.show()

# print(X_train.shape, X_train.dtype)
# print(Y_train.shape, Y_train.dtype)
# print(X_test.shape, X_test.dtype)
# print(Y_test.shape, Y_test.dtype)
```




```
(60000, 28, 28) float64
(60000,) uint8
(10000, 28, 28) float64
(10000,) uint8
```

Model 구성

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              803840
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,432,970
Trainable params: 3,432,970
Non-trainable params: 0
```

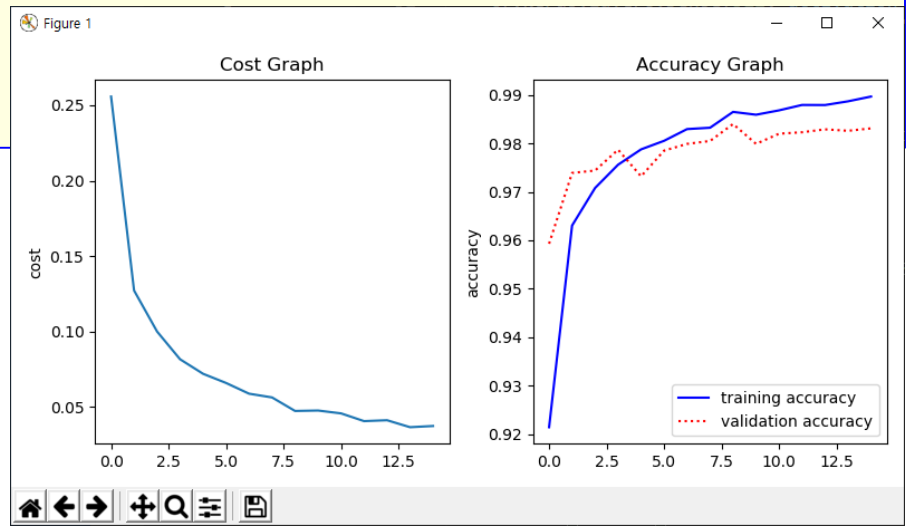
Model compile / fit / evaluate

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
model.summary()
hist = model.fit(X_train, Y_train,
                 validation_data=(X_test, Y_test),
                 verbose=2, batch_size=100, epochs=15,
                  use_multiprocessing=True)
model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 - 12s - loss: 0.2558 - accuracy: 0.9214 - val_loss: 0.1380 - val_accuracy: 0.9593
Epoch 2/15
60000/60000 - 11s - loss: 0.1272 - accuracy: 0.9631 - val_loss: 0.0897 - val_accuracy: 0.9739
...
Epoch 14/15
60000/60000 - 13s - loss: 0.0367 - accuracy: 0.9887 - val_loss: 0.0811 - val_accuracy: 0.9826
Epoch 15/15
60000/60000 - 13s - loss: 0.0375 - accuracy: 0.9897 - val_loss: 0.0796 - val_accuracy: 0.9831
10000/1 - 0s - loss: 0.0509 - accuracy: 0.9831
```

Cost / Accuracy

```
# Reporting......
plt.figure(figsize=(8, 4))  # 8 x 4 inchs
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'])
plt.title("Cost Graph")
plt.ylabel("cost")
plt.subplot(1, 2, 2)
plt.title("Accuracy Graph")
plt.ylabel("accuracy")
plt.plot(hist.history['accuracy'], 'b-', label="training accuracy")
plt.plot(hist.history['val_accuracy'], 'r:', label="validation accuracy")
plt.legend()
plt.tight_layout()
plt.show()
```



MNIST Dataset 98%이상 달성하기

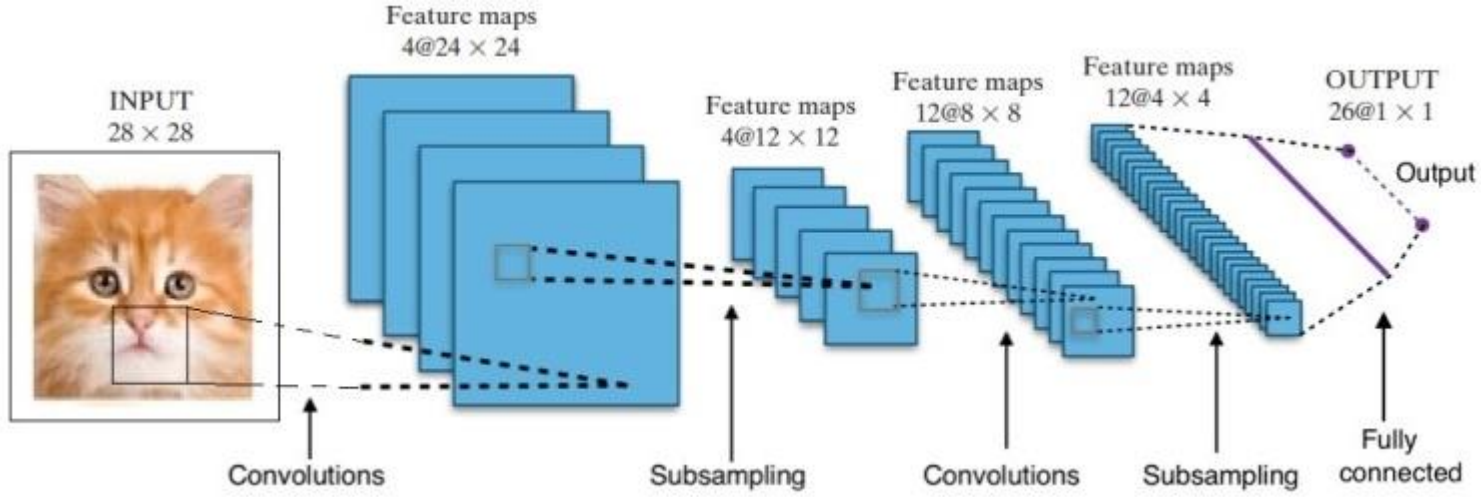
- 목표 달성!

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 - 12s - loss: 0.2558 - accuracy: 0.9214 - val_loss: 0.1380 - val_accuracy: 0.9593
Epoch 2/15
60000/60000 - 11s - loss: 0.1272 - accuracy: 0.9631 - val_loss: 0.0897 - val_accuracy: 0.9739
...
Epoch 14/15
60000/60000 - 13s - loss: 0.0367 - accuracy: 0.9887 - val_loss: 0.0811 - val_accuracy: 0.9826
Epoch 15/15
60000/60000 - 13s - loss: 0.0375 - accuracy: 0.9897 - val_loss: 0.0796 - val_accuracy: 0.9831
10000/1 - 0s - loss: 0.0509 - accuracy: 0.9831
```

- 더 좋은 방법은 없을까?
- 99%이상은 불가능할까?

CNN이란?

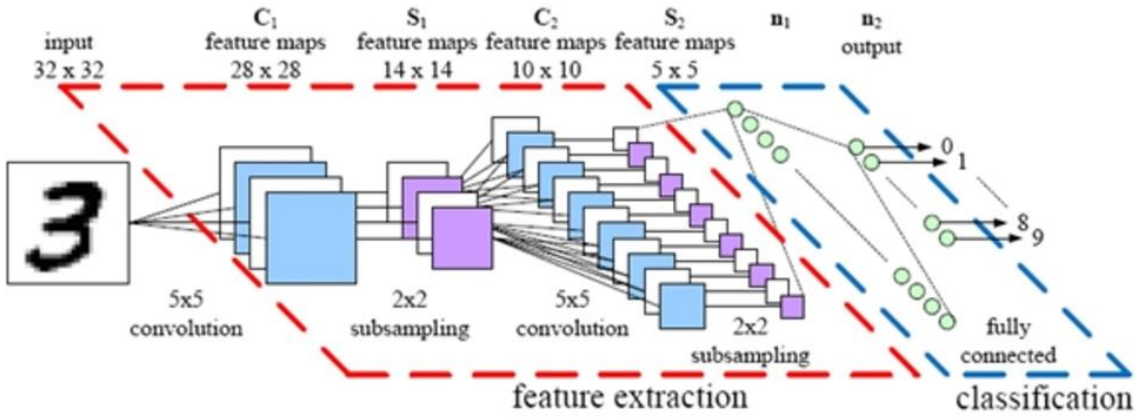
- CNN(Convolution Neural Network)
  - 입력데이터 : 이미지
  - 특징 추출과 분류를 동시에 학습하는 NN



  - 고양이 실험
    - 어떤 이미지를 보여주었을 때, 고양이의 모든 neuron들이 작동하는 것이 아니라 어떤 형태의 그림에 대해 부분적으로 작동(Hubel & Wiesel, 1959~1968)

Convolutional NN

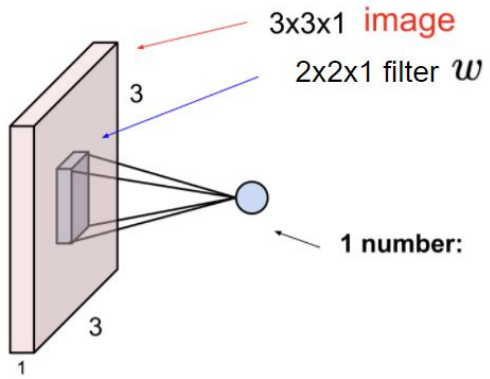
- CNN은 일반적으로 3종류의 Layer로 구성



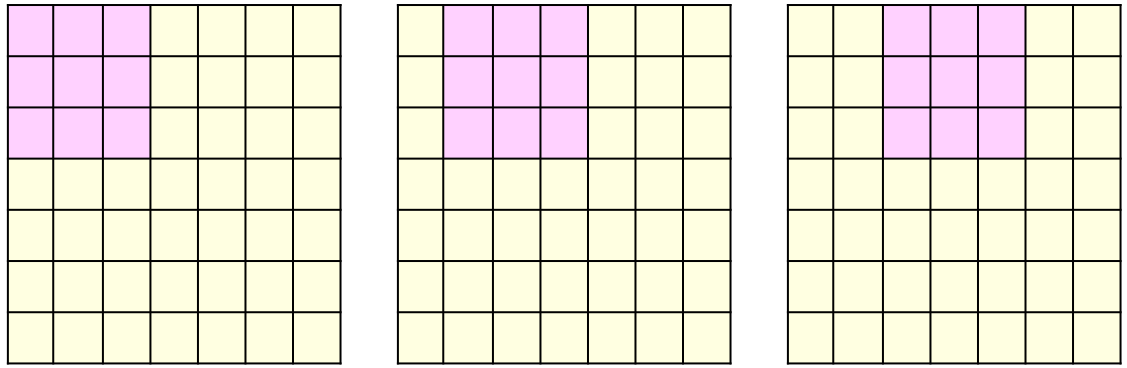
  - Convolution Layer
    - 의미있는 featur를 추출하는 layer
  - Pooling Layer
    - feature를 줄이기 위해 subsampling을 수행
  - Feedforward Layer
    - 분류를 위한 classification layer(Neural Network)

Convolutional Layer

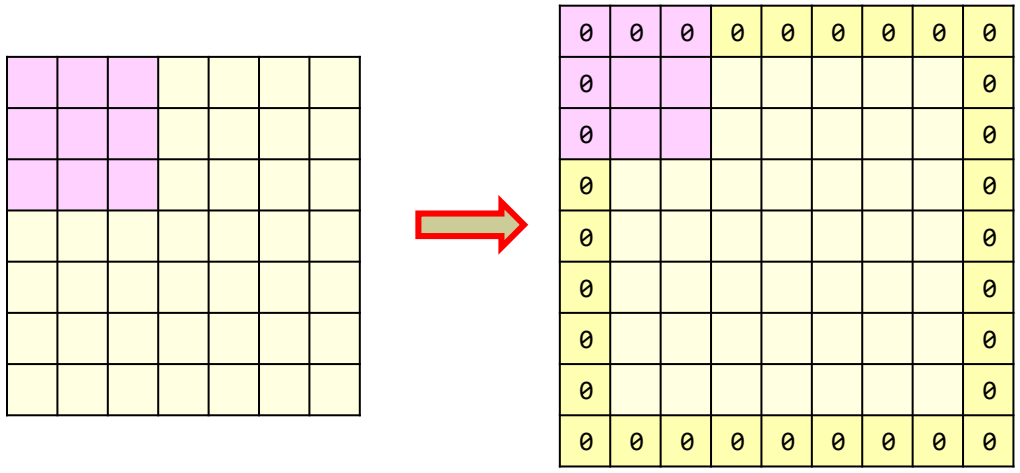
- 입력된 이미지의 일부분을 잘라서 계산된 값을 입력으로 받는 Layer
  - 대부분의 컬러이미지 width*height*depth로 구성
- Layer구성
  - filter를 적용하여 one number를 얻어낸다
  - 적용함수
    - $f(x) = Wx + b$ 또는 $f(x) = ReLU(Wx + b)$



- **Convolutional Layer의 크기는?**
  - **output size = (N-F)/stride + 1**
    - image size : N x N
    - filter size : F x F
    - stride : filter를 움직이는 간격
  - (예)
    - N=7, F=3, stride=1 output size=5
    - N=7, F=3, stride=2 output size=3



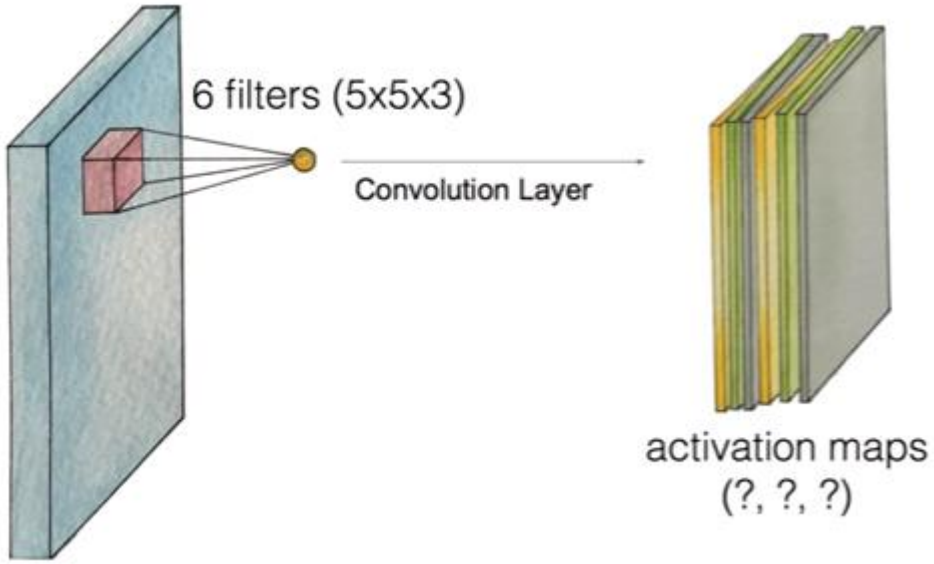
- output size를 크게 할 수 없을까?
  - padding : 이미지의 바깥경계에 0을 추가
  - (예) padding = 1인 경우,



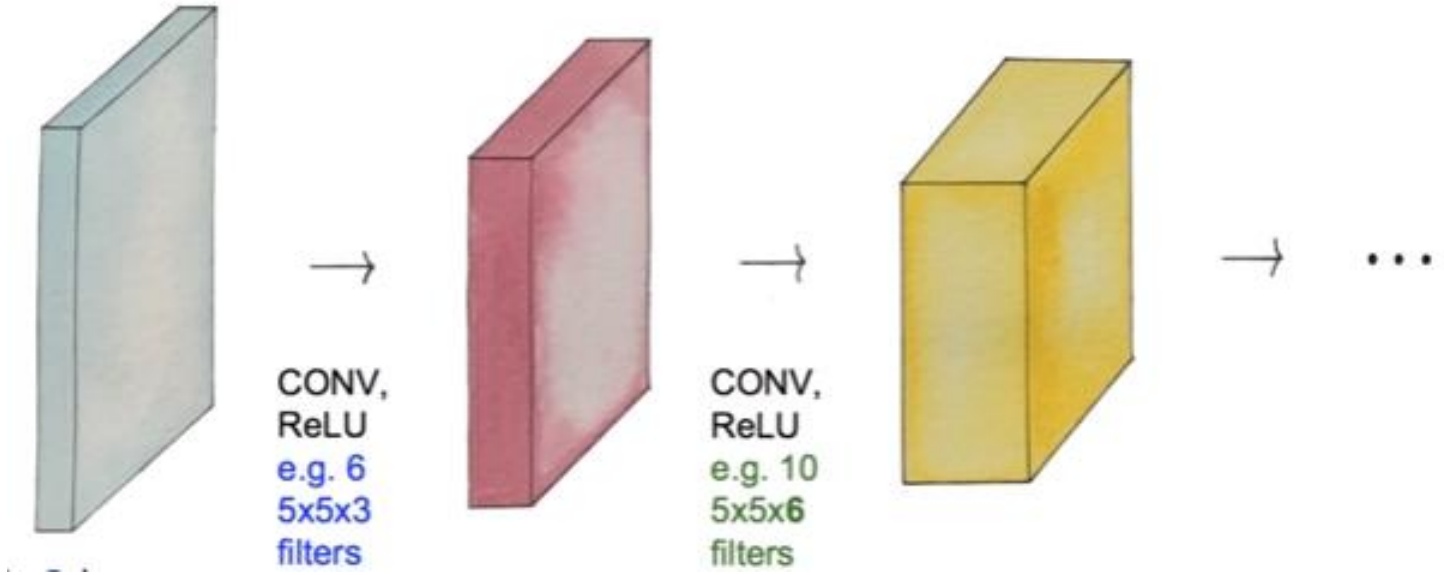
7x7 image　　9x9 image

  - 그러면, N=7, F=3, stride=1, padding=1
    - output size = (9-3)/1+1 = 7

- filer를 동시에 여러 개 적용하면?
  - 각 filter당 1개의 activation map이 생성
  - (예) (32,32,3)이미지에 (5,5,3)의 filter를 6개 적용하면 (stride=1, padding=1)
    - output size = (34-5)/1+1 = 30
    - activation map의 크기 : (30,30,6)



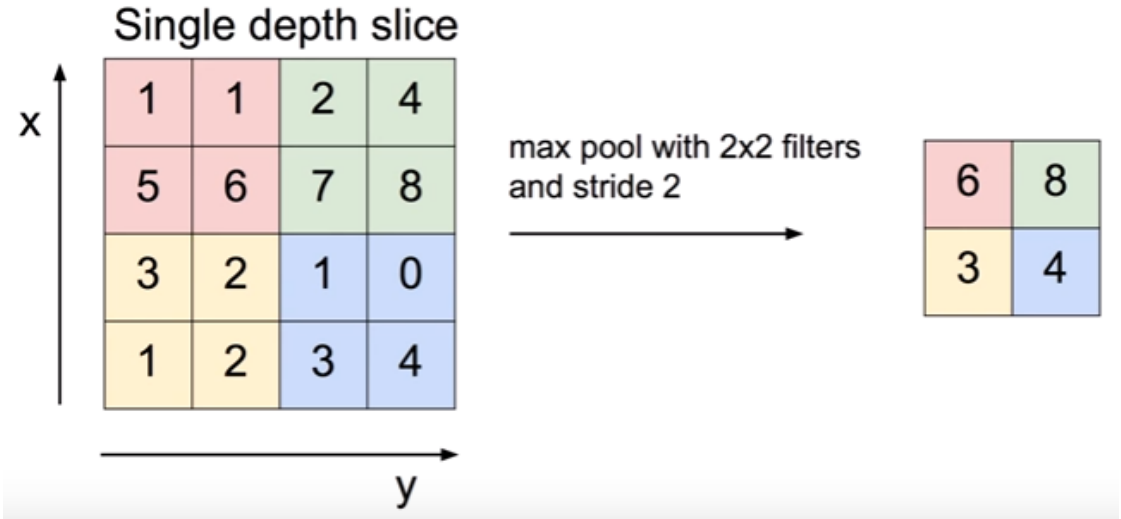
- 이러한 과정을 여러 번 반복할 수 있음



  - 그러면 매우 많은 weight들이 발생
  - 이것을 적절하게 subsampling할 필요가 있음
    - Pooling이라고 부름

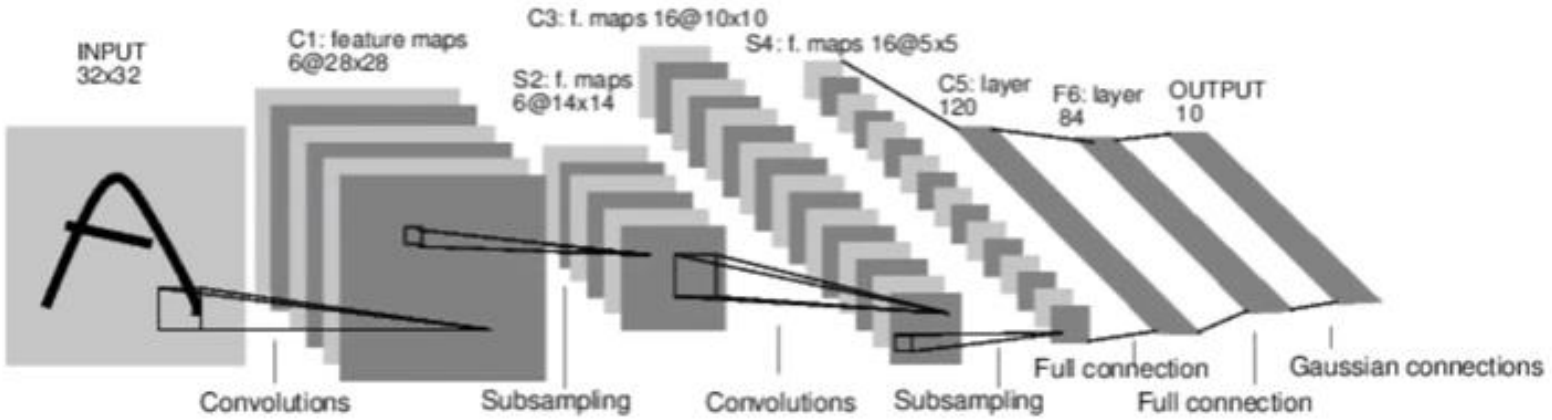
Pooling Layer

- **sampling된 layer**
  - Convolutional Layer의 크기를 resize
  - 주어진 layer를 적절하게 subsampling하여 크기를 줄여 줌(주로 max pooling을 사용)
- **max pooling**
  - filter안의 값들 중 최대값을 고르는 방법



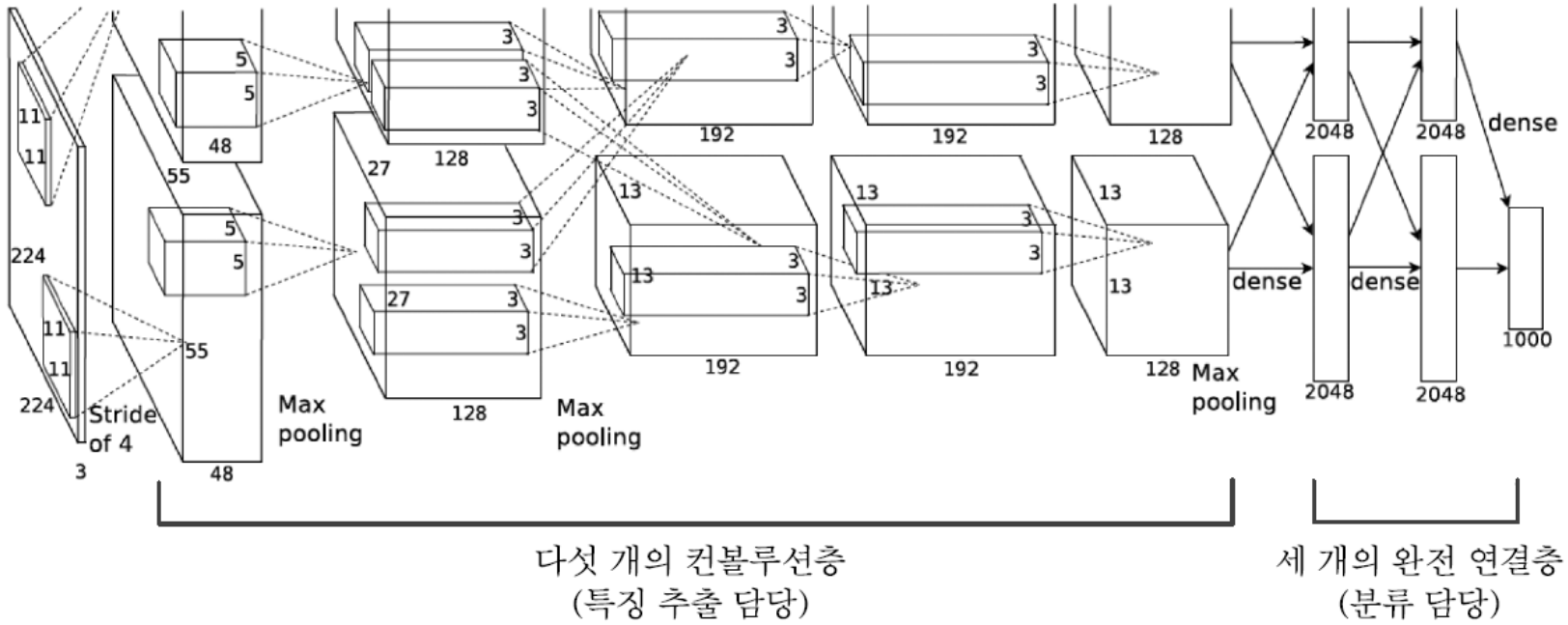
CNN Model : Case 1

[LeCun et al., 1998]

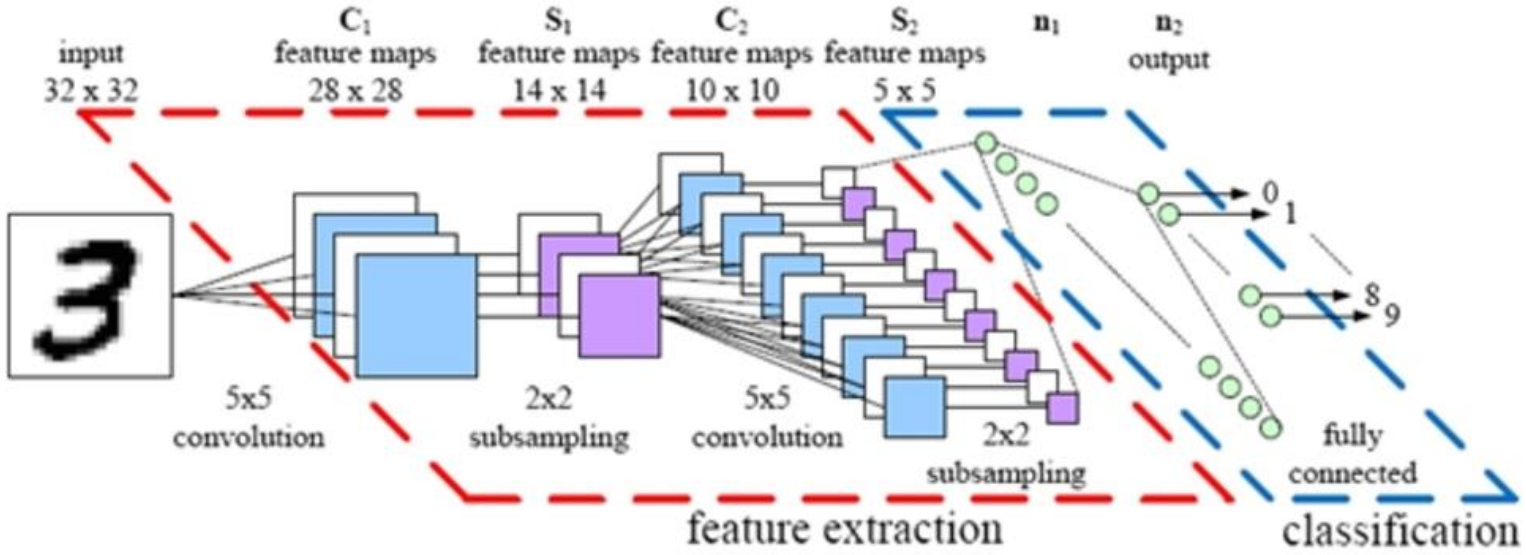


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

CNN Model : Case 2



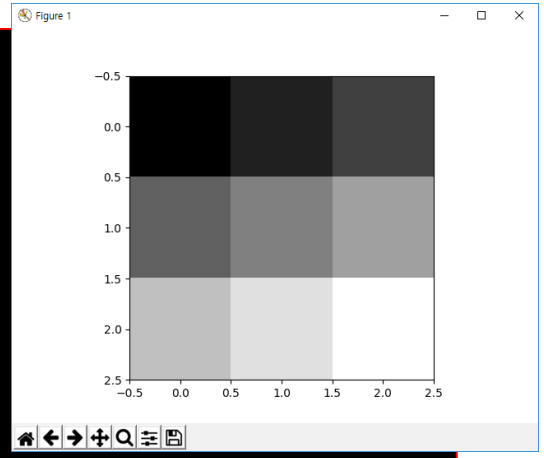
CNN Model : Case 3



Tensorflow로 간단한 CONV Layer만들기

- 간단한 이미지 만들기

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random

image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print('image.shape = ', image.shape) #(1,3,3,1) : (number, row, col, channel)
print('image.reshaped.shape = ', image.reshape(3,3).shape)
print('image.reshaped:\n', image.reshape(3,3), sep='')
plt.imshow(image.reshape(3,3), cmap='gray')
```



- shape(1, 3, 3, 1)
  - (개수, row, col, channel)

```
image.shape =  (1, 3, 3, 1)
image.reshaped.shape =  (3, 3)
image.reshaped:
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
```

tf.keras.layers.Conv2D

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), groups=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, **kwargs
)
```

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).
data_format	A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape **(batch_size, height, width, channels)** while channels_first corresponds to inputs with shape (batch_size, 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.

Filter 정의(1개)

```
weight = tf.constant([[[[1.0]],[[1.0]]],
                      [[[1.0]],[[1.0]]]])
print('weight.shape = ', weight.shape) #(2,2,1,1):(row,col,channel,number)
weight_init = tf.constant_initializer(weight.numpy())
weight_img = tf.reshape(weight, (2,2))
print('weight.reshaped.shape = ', weight_img.shape)
print('weight.reshaped:\n', weight_img.numpy(), sep='')
```

- shape(2, 2, 1, 1)
  - (row, col, channel, 개수)

```
weight.shape =  (2, 2, 1, 1)
weight.reshaped.shape =  (2, 2)
weight.reshaped:
[[ 1.  1.]
 [ 1.  1.]]
```

Filter 적용하기(padding='VALID')

```
print("padding='VALID'") # 패딩(바깥여백)을 적용안함
conv2d = tf.keras.layers.Conv2D(filters=1, kernel_size=2, padding='VALID',
                                kernel_initializer=weight_init)(image)
conv2d_img = conv2d.numpy()
print('conv2d_img.shape = ', conv2d_img.shape)
print('conv2d_img:\n', conv2d_img, sep='')
print('conv2d_img.reshaped.shape = ', conv2d_img.reshape(2,2).shape)
print('conv2d_img.reshaped:\n', conv2d_img.reshape(2,2), sep='')
```

- **tf.keras.layers.Conv2D()**
  - filter=1
  - kernel_size=2
  - padding
    - VALID : 0
  - kernel_initializer=weight_init

```
padding='VALID'
conv2d_img.shape =  (1, 2, 2, 1)
conv2d_img:
[[[[ 12.]
   [ 16.]]

  [[ 24.]
   [ 28.]]]]
conv2d_img.reshaped.shape =  (2, 2)
conv2d_img.reshaped:
[[ 12.  16.]
 [ 24.  28.]]
```

Filter 적용하기(padding='SAME')

```
print("padding='SAME'") # 출력이 입력과 같은 크기가 되도록 패딩을 적용함
conv2d = tf.keras.layers.Conv2D(filters=1, kernel_size=2, padding='SAME',
                                kernel_initializer=weight_init)(image)
conv2d_img = conv2d.numpy()
print('conv2d_img.shape = ', conv2d_img.shape)
print('conv2d_img:\n', conv2d_img, sep='')
print('conv2d_img.reshaped.shape = ', conv2d_img.reshape(3,3).shape)
print('conv2d_img.reshaped:\n', conv2d_img.reshape(3,3), sep='')
```

- **tf.keras.layers.Conv2D()**
  - filter=1
  - kernel_size=2
  - padding
    - SAME : 입력과 같은 크기가 되도록 함
  - kernel_initializer=weight_init

```
padding='SAME'
conv2d_img.shape =  (1, 3, 3, 1)
conv2d_img:
[[[[ 12.]
   [ 16.]
   [  9.]]

  [[ 24.]
   [ 28.]
   [ 15.]]

  [[ 15.]
   [ 17.]
   [  9.]]]]
conv2d_img.reshaped.shape =  (3, 3)
conv2d_img.reshaped:
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
```

여러 개의 Filter 적용하기

- Filter 정의(3개)

```
weight = tf.constant([[[[1.0, 10.0, -1.0]],[[1.0, 10.0, -1.0]]],
                      [[[1.0, 10.0, -1.0]],[[1.0, 10.0, -1.0]]]])
weight_img = weight.numpy()
print('weight.shape = ', weight.shape) # (2,2,1,3)=>(row,col,channel,number)
weight_init = tf.constant_initializer(weight.numpy())
weight_img = np.swapaxes(weight_img, 0, 3)
for i, one_img in enumerate(weight_img):
    print(one_img.reshape(2,2))
```

- shape(2, 2, 1, 3)
  - (row, col, depth, 개수)

axes 0과 3을 swap한 결과

```
weight.shape =  (2, 2, 1, 3)
[[ 1.  1.]
 [ 1.  1.]]
[[ 10.  10.]
 [ 10.  10.]]
[[-1. -1.]
 [-1. -1.]]
```

참고:numpy.swapaxes() 함수

- 두 축(axis)을 서로 바꾼다.
- (예)

```python
import numpy as np

x = np.array([[1,2,3]])
print('----------------------------------')
print('shape = ', x.shape)
print(x)
print('---- swap 0 and 1 -----------------')
print('shape = ', np.swapaxes(x, 0, 1).shape)
print(np.swapaxes(x, 0, 1))

y = np.array([[[0,1],[2,3],[4,5],[6,7]]])
print('----------------------------------')
print('shape = ', y.shape)
print(y)
print('---- swap 0 and 1 -----------------')
print('shape = ', np.swapaxes(y, 0, 1).shape)
print(np.swapaxes(y, 0, 1))
print('---- swap 0 and 2 -----------------')
print('shape = ', np.swapaxes(y, 0, 2).shape)
print(np.swapaxes(y, 0, 2))
print('---- swap 1 and 2 -----------------')
print('shape = ', np.swapaxes(y, 1, 2).shape)
print(np.swapaxes(y, 1, 2))
```

```
----------------------------------
shape =  (1, 3)
[[1 2 3]]
---- swap 0 and 1 ----------------
shape =  (3, 1)
[[1]
 [2]
 [3]]
----------------------------------
shape =  (1, 4, 2)
[[[0 1]
  [2 3]
  [4 5]
  [6 7]]]
---- swap 0 and 1 ----------------
shape =  (4, 1, 2)
[[[0 1]]

 [[2 3]]

 [[4 5]]

 [[6 7]]]
---- swap 0 and 2 ----------------
shape =  (2, 4, 1)
[[[0]
  [2]
  [4]
  [6]]

 [[1]
  [3]
  [5]
  [7]]]
---- swap 1 and 2 ----------------
shape =  (1, 2, 4)
[[[0 2 4 6]
  [1 3 5 7]]]
```

Filter 3개 적용하기(padding='SAME')

```
print("padding='SAME'")
conv2d = tf.keras.layers.Conv2D(filters=3, kernel_size=2, padding='SAME',
                                kernel_initializer=weight_init)(image)
conv2d_img = conv2d.numpy()
print('conv2d_img.shape = ', conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
```

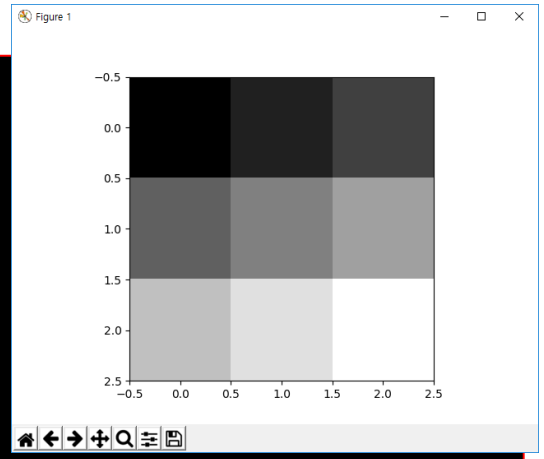
- **tf.keras.layers.Conv2D()**
  - filter=3
  - kernel_size=2
  - padding
    - SAME : 입력과 같은 크기가 되도록 함
  - kernel_initializer=weight_init

```
padding='SAME'
conv2d_img.shape =  (1, 3, 3, 3)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
[[ 120.  160.   90.]
 [ 240.  280.  150.]
 [ 150.  170.   90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```

Tensorflow로 Max Pooling 구현

- 간단한 이미지 만들기

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random

image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print('image.shape = ', image.shape)
print('image.reshaped.shape = ', image.reshape(3,3).shape)
print('image.reshaped:\n', image.reshape(3,3), sep='')
```



- shape(1, 3, 3, 1)
  - (개수, row, col, depth)

```
image.shape =  (1, 3, 3, 1)
image.reshaped.shape =  (3, 3)
image.reshaped:
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
```

tf.keras.layers.MaxPool2D

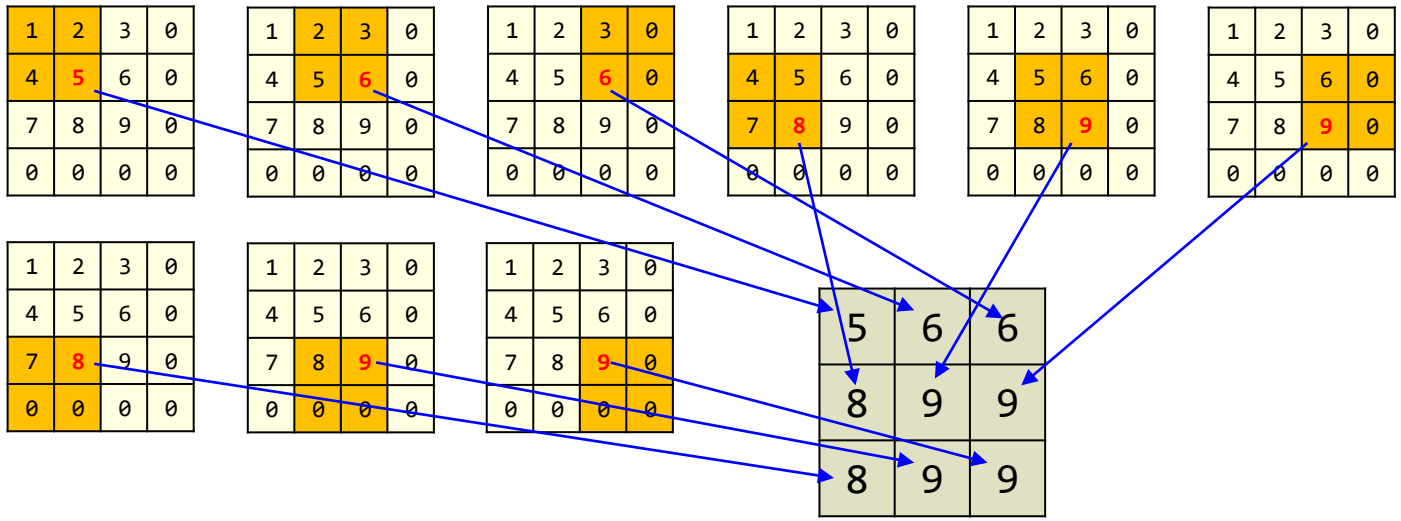
```
tf.keras.layers.MaxPool2D(
    pool_size=(2, 2), strides=None, padding='valid', data_format=None, **kwargs
)
```

pool_size	integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. 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. Specifies how far the pooling window moves for each pooling step. If None, it will default to pool_size.
padding	One of "valid" or "same" (case-insensitive). "valid" adds no zero padding. "same" adds padding such that if the stride is 1, the output shape is the same as input shape.
data_format	A string, one of channels_last (default) 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".

Max Pooling

```
padding='SAME'
pool_img.shape =  (1, 3, 3, 1)
[[ 5.  6.  6.]
 [ 8.  9.  9.]
 [ 8.  9.  9.]]
```

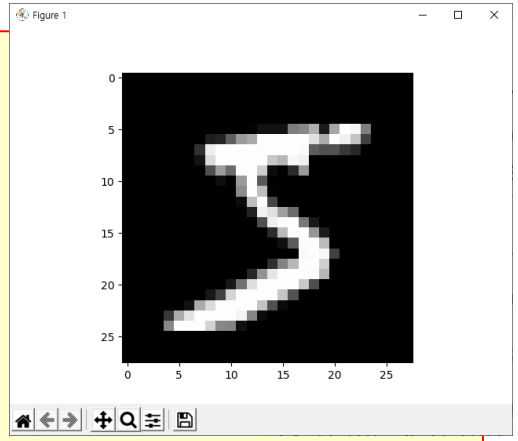
```
print("padding='SAME'")
pool = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides=1,
                                 padding='SAME')(image)
pool_img = pool.numpy()
print('pool_img.shape = ', pool_img.shape)
conv2d_img = np.swapaxes(pool_img, 0, 3)
for i, one_img in enumerate(pool_img):
    print(one_img.reshape(3,3))
```



MNIST 데이터를 이용한 Conv2D, MaxPool2D

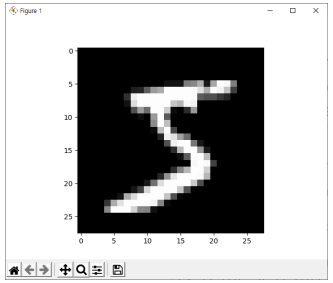
- **MINIST dataset loading & select an image**

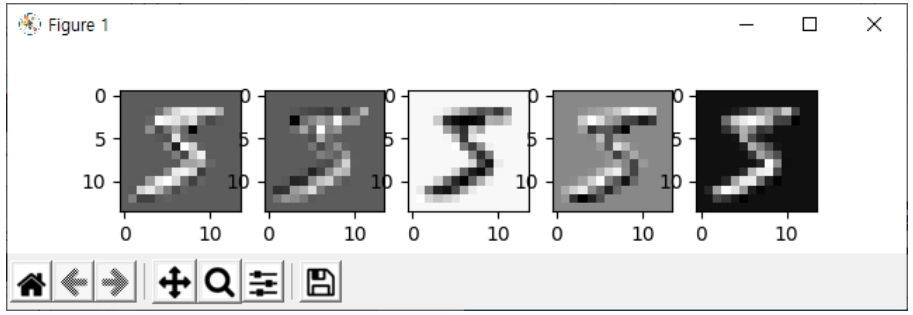
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.astype(np.float32) / 255.0
X_test = X_test.astype(np.float32) / 255.0

img = X_train[0]
plt.imshow(img, cmap='gray')
plt.show()
```



Convolution Layer

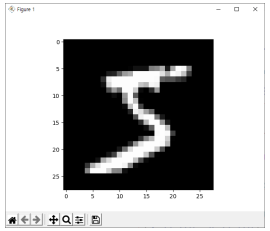
```
img = img.reshape(-1, 28, 28, 1)
img = tf.convert_to_tensor(img)
weight_init = tf.keras.initializers.RandomNormal(stddev=0.01)
conv2d = tf.keras.layers.Conv2D(filters=5, kernel_size=3, strides=(2,2),
                                padding='SAME',
                                kernel_initializer=weight_init)(img)
print(f"conv2d.shape = {conv2d.shape}")
images = np.swapaxes(conv2d, 0, 3)
for i, image in enumerate(images):
    plt.subplot(1, 5, i+1)
    plt.imshow(image.reshape(14,14), cmap='gray')
plt.show()
```

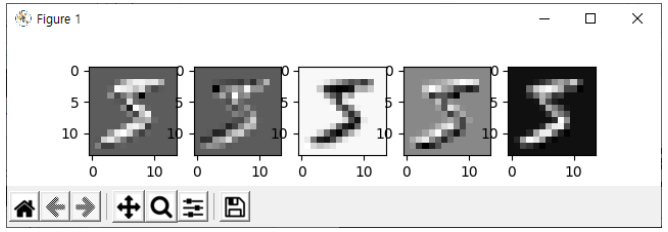


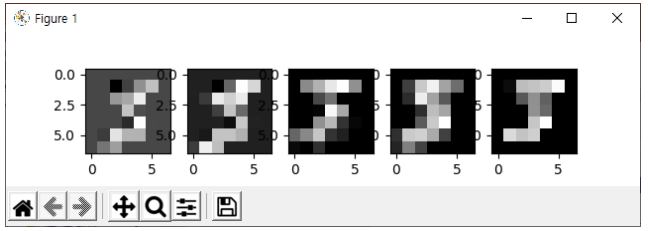


Pooling Layer

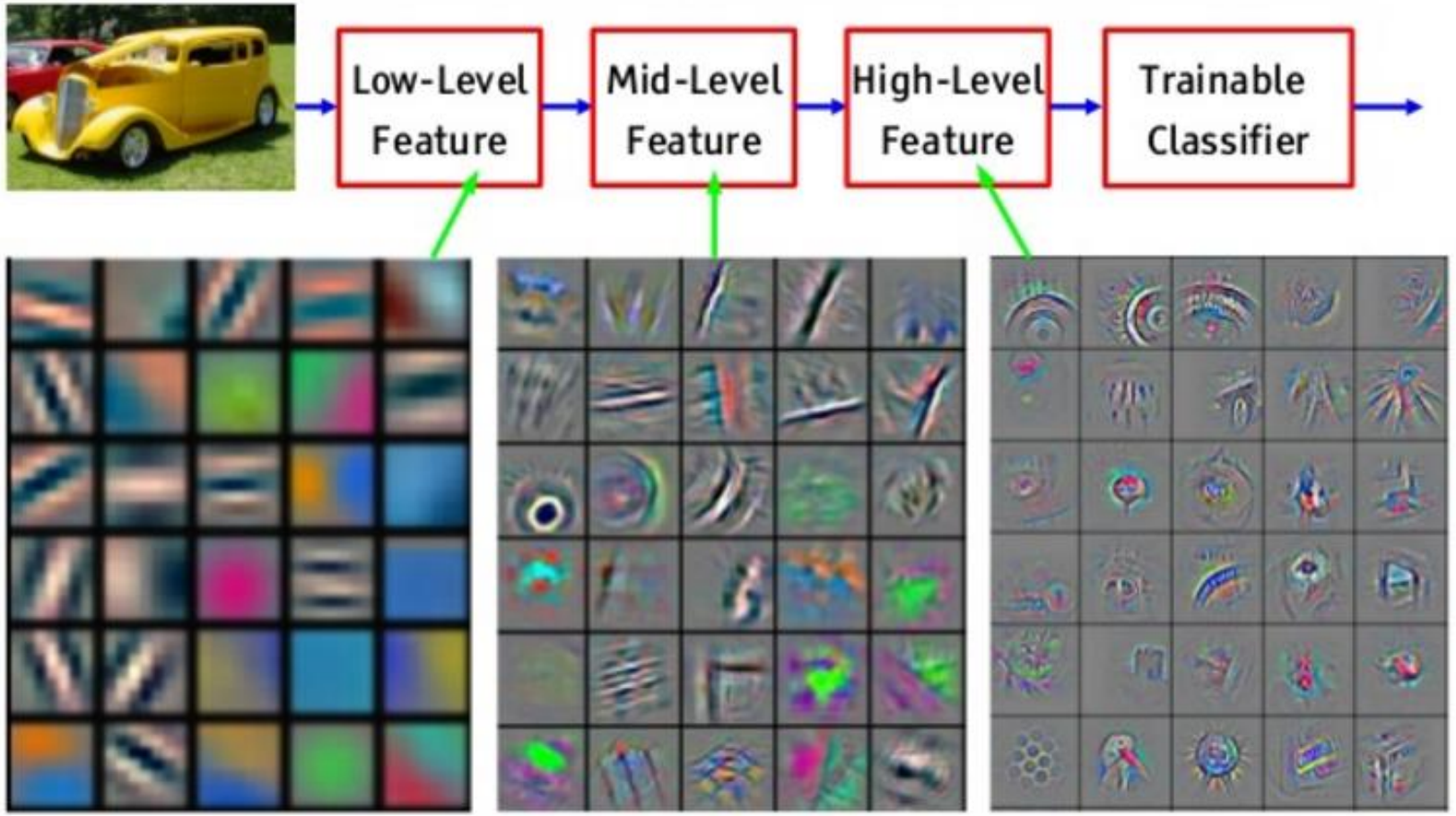
```
pool = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2),
                                 padding='SAME')(conv2d)
print(f"pool.shape = {pool.shape}")
images = np.swapaxes(pool, 0, 3)
for i, image in enumerate(images):
    plt.subplot(1, 5, i+1)
    plt.imshow(image.reshape(7,7), cmap='gray')

plt.show()
```







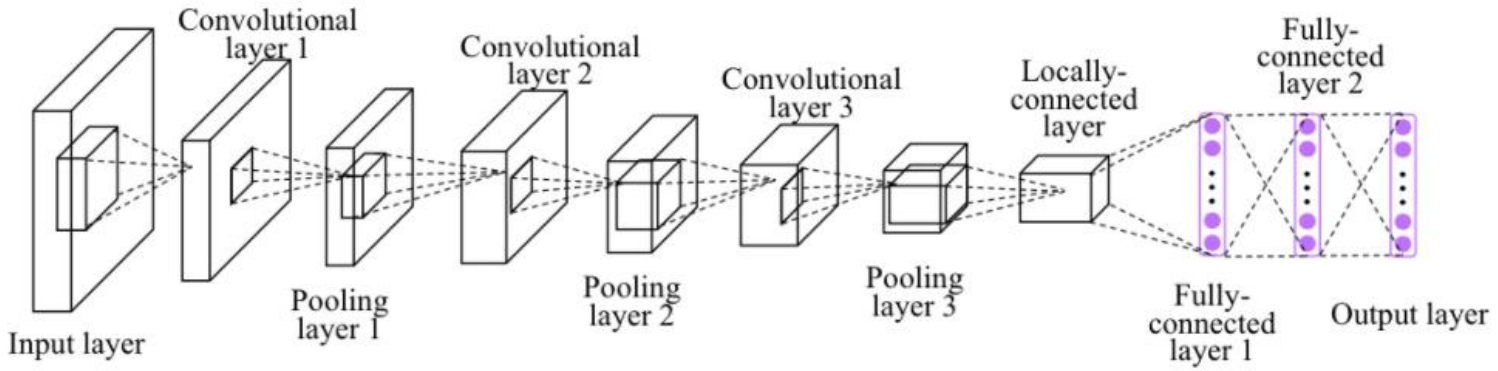
CNN 각 layer의 상태변화



https://informatics.sydney.edu.au/news/tf_on_linux/

MNIST Dataset을 CNN으로 깊은학습

- CNN의 구성
  - Input Layer : 1개
  - Convolutional layer : 3개
    - 각 layer마다 Pooling layer추가
    - Dropout : 0.3
  - Fully-connected Layer : 2개
  - Output Layer : 1개



CNN for MNIST Dataset

- **Dataset 정의**

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import tensorflow as tf
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)
X_train, X_test = X_train / 255.0, X_test / 255.0

# print(X_train.shape, X_train.dtype)
# print(Y_train.shape, Y_train.dtype)
# print(X_test.shape, X_test.dtype)
# print(Y_test.shape, Y_test.dtype)
```

```
(60000, 28, 28) float64
(60000,) uint8
(10000, 28, 28) float64
(10000,) uint8
```

Model 구성

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), padding='same’,
                activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1),
                padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(128, (3, 3), padding='same’,
                activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 356,234
Trainable params: 356,234
Non-trainable params: 0
```

Model compile / fit / evaluate

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
model.summary()
hist = model.fit(X_train, Y_train,
      validation_data=(X_test, Y_test),
      verbose=2, batch_size=100, epochs=5, use_multiprocessing=True)
model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 - 28s - loss: 0.2291 - accuracy: 0.9272 - val_loss: 0.0756 - val_accuracy: 0.9744
Epoch 2/5
60000/60000 - 27s - loss: 0.0656 - accuracy: 0.9788 - val_loss: 0.0383 - val_accuracy: 0.9877
Epoch 3/5
60000/60000 - 28s - loss: 0.0492 - accuracy: 0.9849 - val_loss: 0.0249 - val_accuracy: 0.9921
Epoch 4/5
60000/60000 - 27s - loss: 0.0389 - accuracy: 0.9874 - val_loss: 0.0253 - val_accuracy: 0.9915
Epoch 5/5
60000/60000 - 27s - loss: 0.0336 - accuracy: 0.9890 - val_loss: 0.0209 - val_accuracy: 0.9932
10000/1 - 1s - loss: 0.0128 - accuracy: 0.9932
```

전체코드

```
import tensorflow as tf
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)
X_train, X_test = X_train / 255.0, X_test / 255.0

# print(X_train.shape, X_train.dtype)
# print(Y_train.shape, Y_train.dtype)
# print(X_test.shape, X_test.dtype)
# print(Y_test.shape, Y_test.dtype)

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu’,
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1), padding='same’,
                           activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

continue...

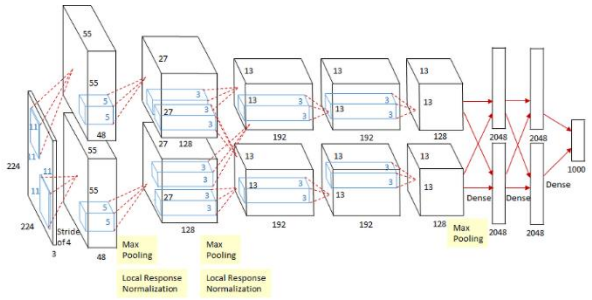
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
model.summary()
hist = model.fit(X_train, Y_train,
                 validation_data=(X_test, Y_test),
                 verbose=2, batch_size=100, epochs=5, use_multiprocessing=True)
model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)

# Reporting......
plt.figure(figsize=(8, 4))  # 8 x 4 inchs
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'])
plt.title("Cost Graph")
plt.ylabel("cost")
plt.subplot(1, 2, 2)
plt.title("Accuracy Graph")
plt.ylabel("accuracy")
plt.plot(hist.history['accuracy'], 'b-', label="training accuracy")
plt.plot(hist.history['val_accuracy'], 'r:', label="validation accuracy")
plt.legend()
plt.tight_layout()
plt.show()
```

MNIST Dataset Classification

- 지금까지 각 메소드의 결과 요약
  - Softmax Classification
    - 약 92.78%
  - Neural Network(NN) Classification
    - 약 98.31%
  - Convolutional-NN Classification
    - 약 99.32%

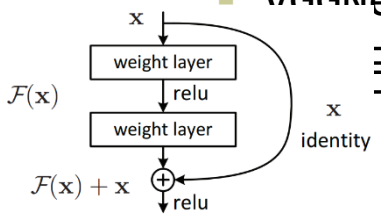
AlexNet

- ILSVRC(ImageNet Large-Scale Visual Recognition Challenge)의 2012년 대회에서 1위
  - ILSVRC : **컴퓨터 비전 분야의 '올림픽'이라 할 수 있는 대회**
  - **제프리 힌튼 교수팀**
    - top 5 test error 기준 15.4%를 기록해 2위(26.2%)를 큰 폭으로 따돌리고 1위를 차지
      - top 5 test error : **예측한 최상위 5개 범주 가운데 정답이 없는 경우의 오류율**
- **주요특징**
  - conv layer, max-pooling layer, dropout layer 5개
  - fully connected layer 3개
  - nonlinearity function : ReLU
  - batch stochastic gradient descent



ResNet

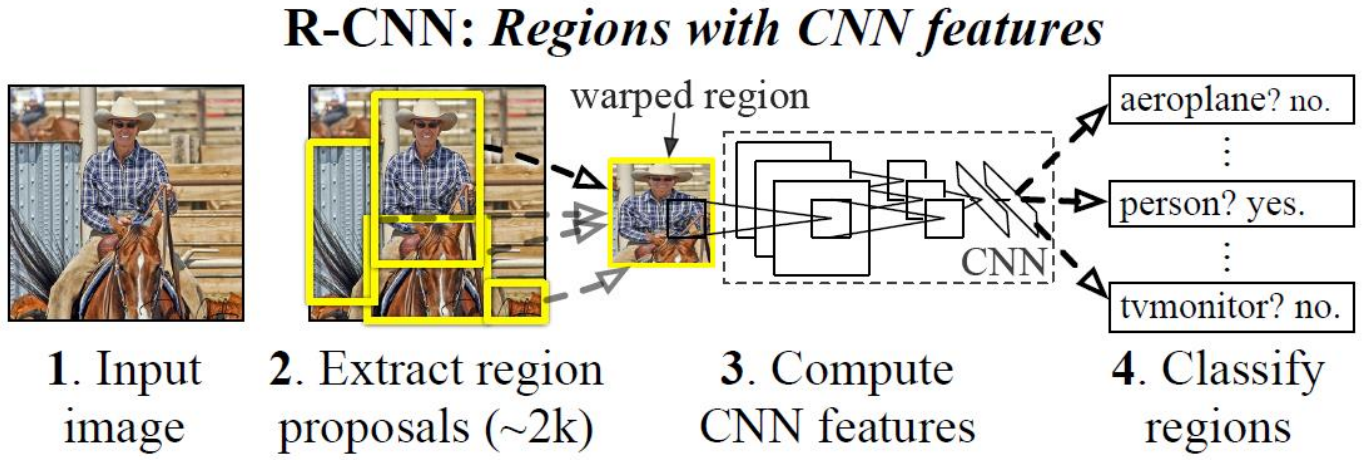
- ILSVRC(ImageNet Large-Scale Visual Recognition Challenge)의 2015년 대회에서 1위(오류율 3.6%)
  - ILSVRC : 컴퓨터 비전 분야의 '올림픽'이라 할 수 있는 대회
  - Microsoft Kaiming He 등
    - top 5 test error 기준 10.6%를 기록
      - top 5 test error : 예측한 최상위 5개 범주 가운데 정답이 없는 경우의 오류율
- **주요특징**
  - Residual block
    - 층이 깊어질수록 gradient vanishing 문제가 발생
      - VGGNet(19개 레이어), GoogleNet(22개 레이어)



문제를 해결하기 위해 이전 step의 gradient를 잘 흐르도록 함

Region based CNNs

- R-CNN(2013), Fast R-CNN(2015), Faster R-CNN(2015)
  - object detection 문제를 풀기 위해 제안된 모델들
  - 물체의 경계를 찾거나, 물체를 분류하는 문제
- **R-CNN**
  - UC Berkeley Ross Girshick 등
  - 참고 : http://www.cs.berkeley.edu/~rbg/rcnn



- **R-CNN 과정**
  - 1. region proposal
    - selective search를 통해 먼저 region proposal 단계를 수행하여, 물체가 있을만한 영역 탐색
  - 2. CNN
    - feature vector를 자동으로 생성
    - SVM로 classification을 수행
  - 3. bounding box regression
    - localization error를 줄이기 위해 CNN feature를 이용하여 bounding box regression model을 수정

- **Fast R-CNN**
  - R-CNN의 느린 학습시간 등 문제를 해결하기 위한 모델
    - ROI-Pooling 사용
- **Faster R-CNN**
  - Region Proposal을 CNN으로 수행하는 모델

앞으로 더 공부해야 할 것들

- **Machine Learning의 주요 문제**
  - Classification
  - Object Detection
  - Image Segmentation
  - Natural Language Processing
- **Deep Learning Methods**
  - R-CNN(Region based CNN)
    - 영역기반 물체인식
  - RNN(Recurrent Neural Network)
    - LSTM(Long Short-term Memory)
    - 번역기, 음성인식 등
  - **강화학습**(Deep Reinforcement Learning)
    - 로봇 제어, 엘리베이터 스케줄링, 통신망, 게임 등
  - GAN(Generative Adversarial Networks)
    - 데이터 생성
- **적용사례**
  - **딥러닝의 30가지 적용사례**(https://brunch.co.kr/@itschloe1/23)
  - **딥러닝 프로젝트**(http://cs230.stanford.edu/proj-spring-2018.html)

참고문헌

- 밑바닥부터 시작하는 딥러닝(Deep Learning from Scratch), 싸이토 고키, 한빛미디어
- 모두를 위한 머신러닝/딥러닝 강의, Sung Kim, (https://www.youtube.com/playlist?list=PLlMkM4tgfjnLSOjrEJN31gZATbcj_MpUm)
- 텐서플로우 기초 이해하기(R0.12), 문용준, https://www.slideshare.net/dahlmoon/20160623-63318427
- Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets.html

Thank you!