텐서플로로 시작하는 딥러닝

# 4. Convolutional Neural Networks


Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

2019.

HJYU@UOS.AC.KR

# CHAPTER 4 Convolution filter를 통한 image 특징 추출

# 4.1 Convolution Filter 의 기능
## 4.1.1 Convolution Filter 의 예

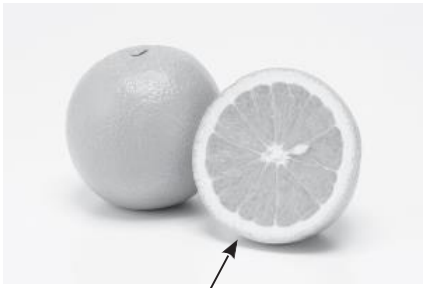
| | | |
|---|---|---|
| 0.11 | 0.11 | 0.11 |
| 0.11 | 0.12 | 0.12 |
| 0.11 | 0.11 | 0.11 |

| | | |
|---|---|---|
| 0.05 | 0.05 | 0.05 |
| 0.05 | 0.60 | 0.05 |
| 0.05 | 0.05 | 0.05 |

Fig 4-2 image 그러데이션 효과를 주는 filter의 예

- 각 pixel 의 값에 가중치를 곱해서 합산한 값을 중앙 pixel 의 값으로 치환
- 모든 가중치의 합계가 1

# Fig 4-3 Convolution Filter 의 적용 예



| | | |
|---|---|---|
| 91 | 46 | 35 |
| 141 | 140 | 135 |
| 156 | 153 | 153 |

×

| | | |
|---|---|---|
| 0.11 | 0.11 | 0.11 |
| 0.11 | 0.12 | 0.11 |
| 0.11 | 0.11 | 0.11 |

117

$$91 \times 0.11 + 46 \times 0.11 + 35 \times 0.11$$
$$+141 \times 0.11 + 140 \times 0.12 + 135 \times 0.11$$
$$+156 \times 0.11 + 153 \times 0.11 + 153 \times 0.11 = 116.9$$

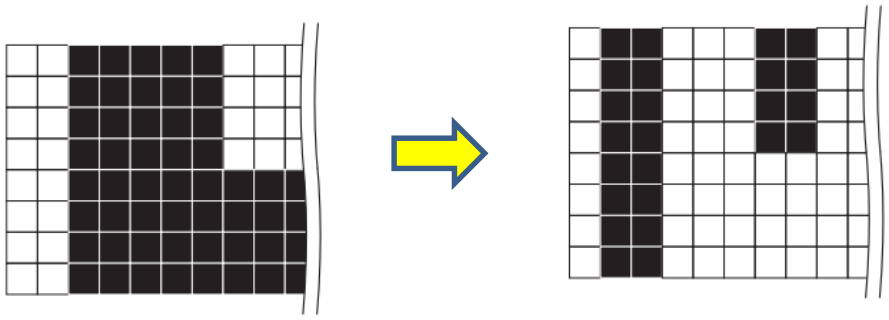
# Filters

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

평균을 추출하는 filter

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -3 | 0 | 3 |

Fig 4-4. 세로 edge를 추출하는 filter
가로로 같은 색이 이어지면 +-상쇄
- 결과가 음수이면 절대값



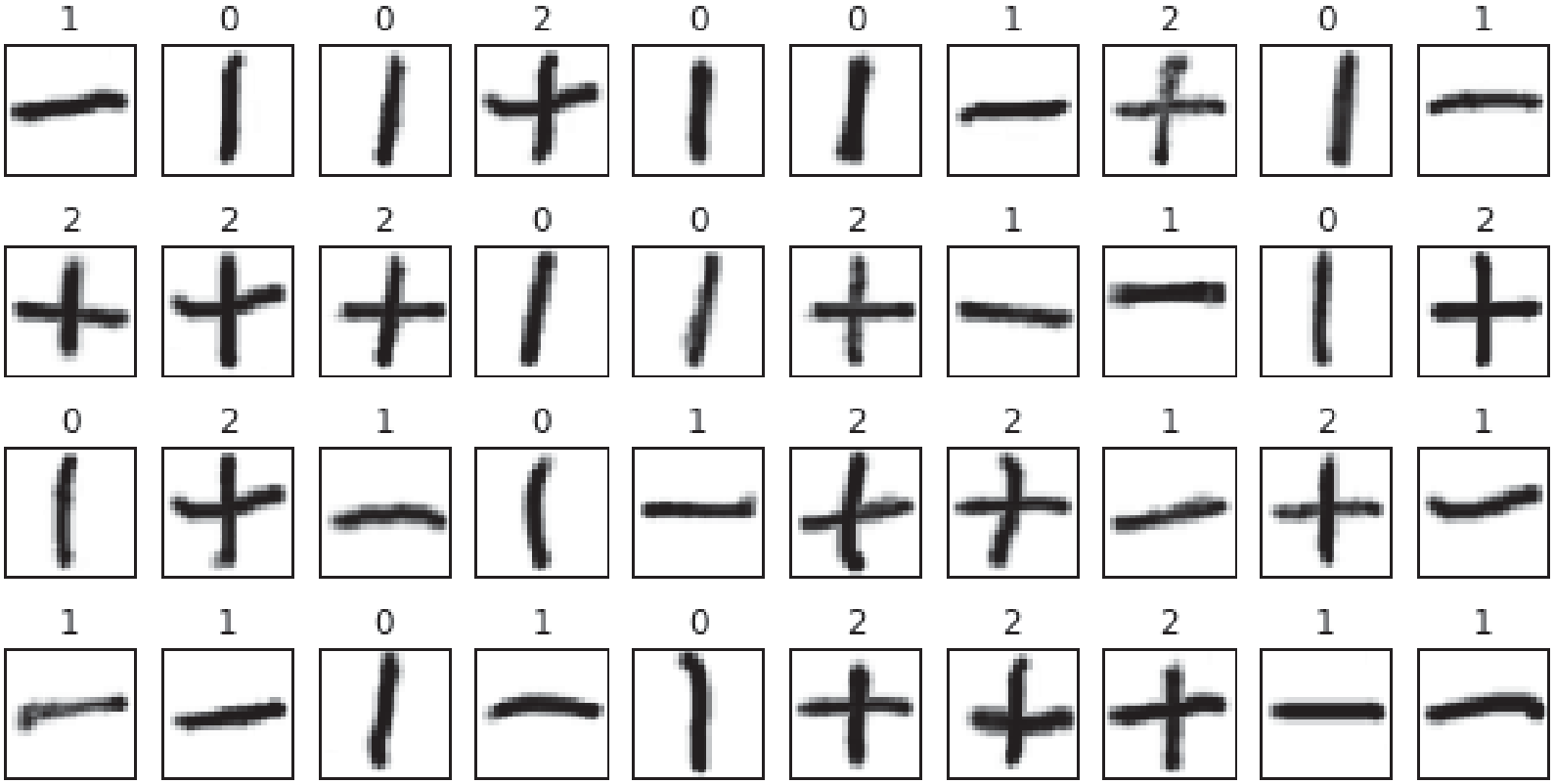
# Fig 4-5 세로와 가로 edge를 보다 두꺼운 폭으로 추출하는 filter

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 0 | -1 | -2 |
| 3 | 2 | 0 | -2 | -3 |
| 4 | 3 | 0 | -3 | -4 |
| 3 | 2 | 0 | -2 | -3 |
| 2 | 1 | 0 | -1 | -2 |

| | | | | |
|---|---|---|---|---|
| 2 | 3 | 4 | 3 | 2 |
| 1 | 2 | 3 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| -1 | -2 | -3 | -2 | -1 |
| -2 | -3 | -4 | -3 | -2 |

※실제로는 각 성분을 23.0으로 나눈 값을 사용한다

# 4.1.2 tensorflow를 이용한 convolution filter 적용




data 세트의 image data(일부)

# [OFE-01] 필요한 module을 import한다.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cPickle as pickle        # image data 를 읽기 위한 module, Python 2 만가능
```

# [OFE-02] data file 'ORENIST.data'에서 image와 label data를 읽어들인다.

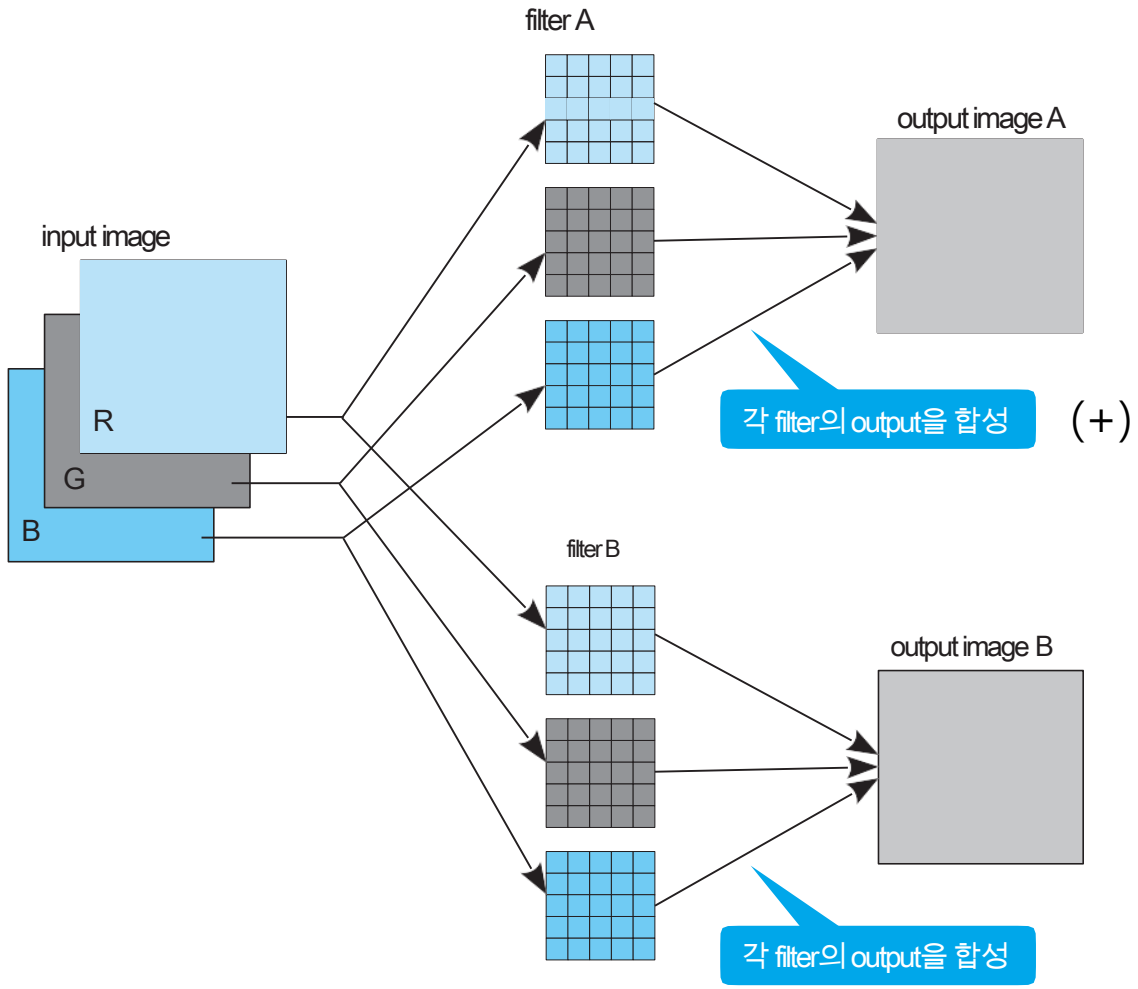
```
with open('ORENIST.data', 'rb') as file:
    images, labels = pickle.load(file)
```

```
# colab 에 file upload
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

# [OFE-03] Show image data samples

```
fig = plt.figure(figsize=(10,5))
for i in range(40):
    subplot = fig.add_subplot(4, 10, i+1)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' % np.argmax(labels[i]))
    subplot.imshow(images[i].reshape(28,28), vmin=0, vmax=1,
                   cmap=plt.cm.gray_r, interpolation='nearest')
```

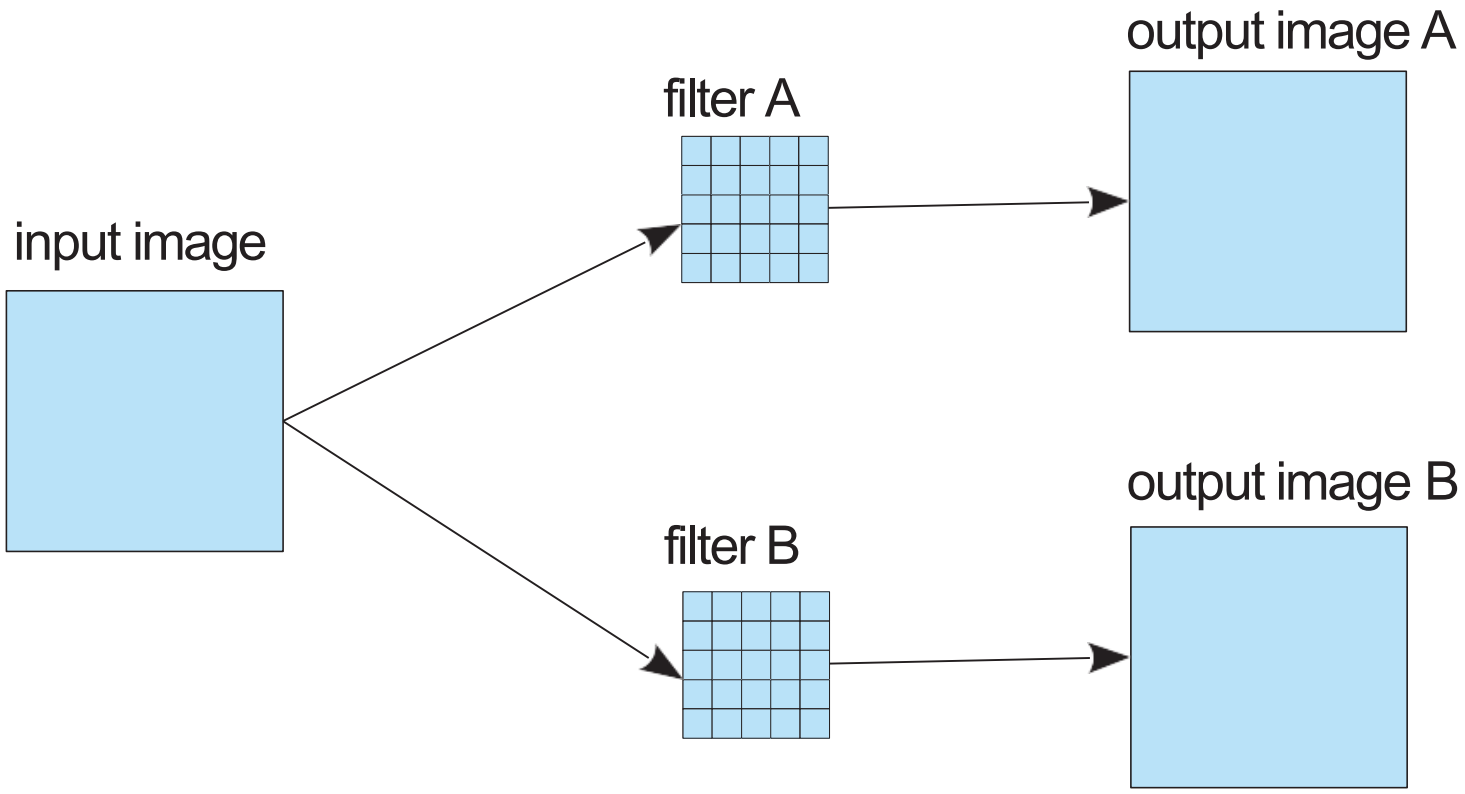
# Fig 4-7 Color image에 대한 convolution filter 적용




[5,5,3,2]

Filter size (세로x가로), num of input layers, num of filters

# Fig 4-8 Gray scale image 에 대한 convolution filter 적용




[5,5,1,2]

Filter size (세로x가로), num of input layers, num of filters

# # [OFE-04] filter 정보를 저장한 다차원 list를 만드는 함수를 준비

```
def edge_filter( ):
    filter0 = np.array(
            [[ 2, 1, 0,-1,-2],
             [ 3, 2, 0,-2,-3],
             [ 4, 3, 0,-3,-4],
             [ 3, 2, 0,-2,-3],
             [ 2, 1, 0,-1,-2]]) / 23.0
    filter1 = np.array(
            [[ 2, 3, 4, 3, 2],
             [ 1, 2, 3, 2, 1],
             [ 0, 0, 0, 0, 0],
             [-1,-2,-3,-2,-1],
             [-2,-3,-4,-3,-2]]) / 23.0
```

```
    filter_array = np.zeros([5,5,1,2])
    filter_array[:,:,0,0] = filter0
    filter_array[:,:,0,1] = filter1

    return tf.constant(filter_array, dtype=tf.float32)
```

Filter size (세로x가로), num of input layers, num of output layers

Session 내에서 사용하는 값은 모두 tensorflow object로 준비

# [OFE-05] Image data에 filter를 적용하는 계산식을 준비

```
x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])

W_conv = edge_filter()
h_conv = tf.abs(tf.nn.conv2d(x_image, W_conv,
            strides=[1,1,1,1], padding='SAME'))
h_conv_cutoff = tf.nn.relu(h_conv-0.2)

h_pool =tf.nn.max_pool(h_conv_cutoff, ksize=[1,2,2,1],
                strides=[1,2,2,1], padding='SAME')

# [OFE-06] 세션을 준비하고 Variable을 초기화

sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
```

input

# of images x size(row x col) x # of layers
-1 : undefined

abs: 결과가 음수이면 절대값

[1, dy, dx, 1]
[1,1,1,1] : 모든 값 계산

SAME
VALID → reduced

Stride [2,2]

Cutoff : 0.2보다 작은 값을 0으로

# [OFE-07] 최초 9개의 image data에 대해 convolution filter를 적용한 결과를 계산

```
filter_vals, conv_vals = sess.run([W_conv, h_conv_cutoff],
                                  feed_dict={x:images[:9]})
```

Image 중 첫 9개

# of images x size(row x col) x # of output layers

```
# [OFE-08] 얻어진 결과를 image로 output

fig = plt.figure(figsize=(10,3))

for i in range(2):
    subplot = fig.add_subplot(3, 10, 10*(i+1)+1)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.imshow(filter_vals[:,:,0,i], cmap=plt.cm.gray_r, interpolation='nearest')
```

```
v_max = np.max(conv_vals)
```

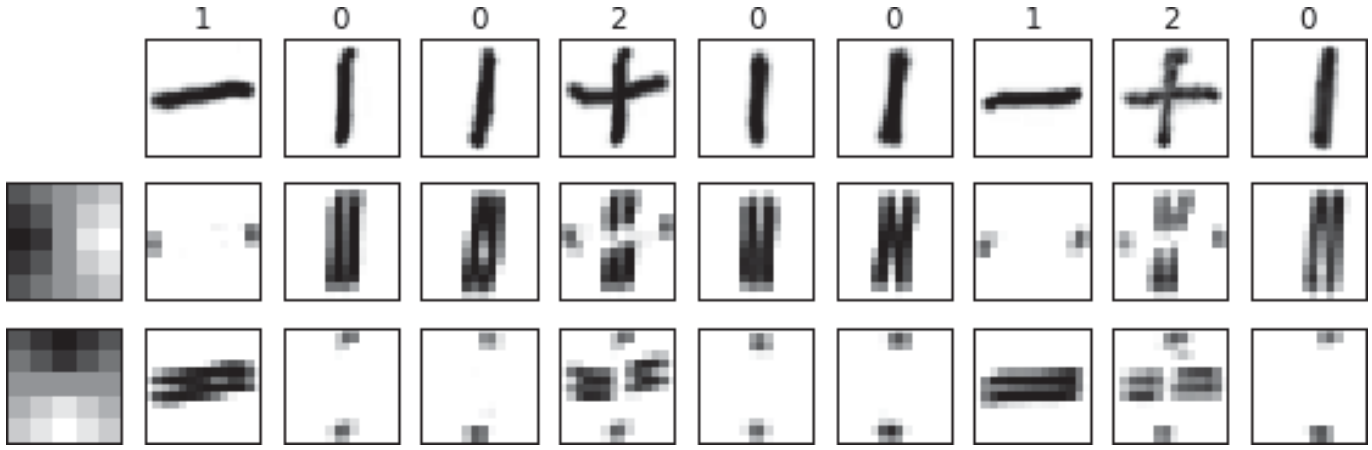
filter 적용 후 Image 값이 1보다 커질 가능성이 있으므로

```
for i in range(9):
    subplot = fig.add_subplot(3, 10, i+2)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' % np.argmax(labels[i]))
    subplot.imshow(images[i].reshape((28,28)), vmin=0, vmax=1,
                   cmap=plt.cm.gray_r, interpolation='nearest')

    subplot = fig.add_subplot(3, 10, 10+i+2)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.imshow(conv_vals[i,:,:,0], vmin=0, vmax=v_max,
                   cmap=plt.cm.gray_r, interpolation='nearest')

    subplot = fig.add_subplot(3, 10, 20+i+2)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.imshow(conv_vals[i,:,:,1], vmin=0, vmax=v_max,
                   cmap=plt.cm.gray_r, interpolation='nearest')
```

# Fig 4-11 convolution filter와 pooling layer를 적용한 결과



# 4.1.3 Pooling layer

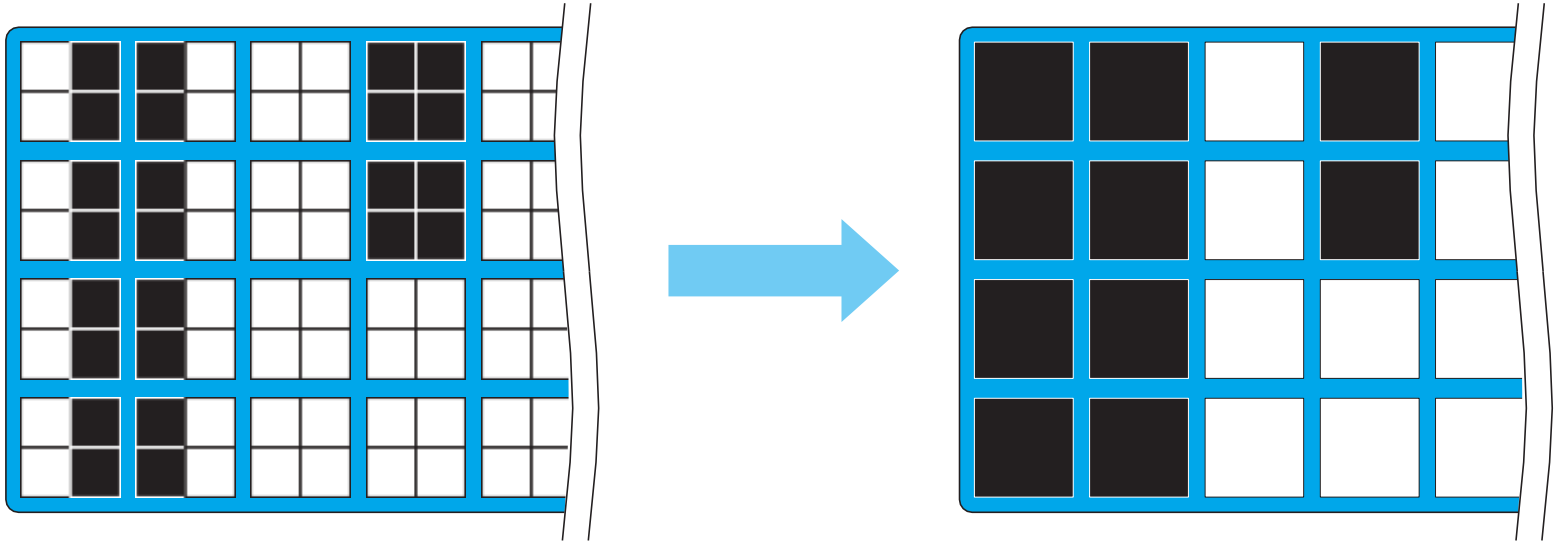


Fig 4-10 pooling layer

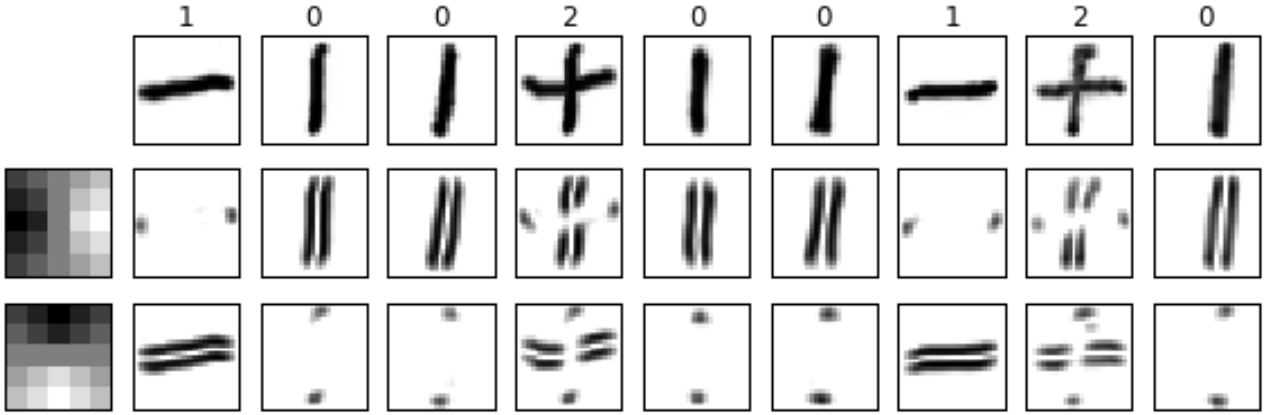
- Image 를 2x2 block 으로 분해
- 각각의 block을 하나의 pixel로 치환 (maximum)
- 28x28 image → 14x14 image

tf.nn.max_pool(h_conv_cutoff, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

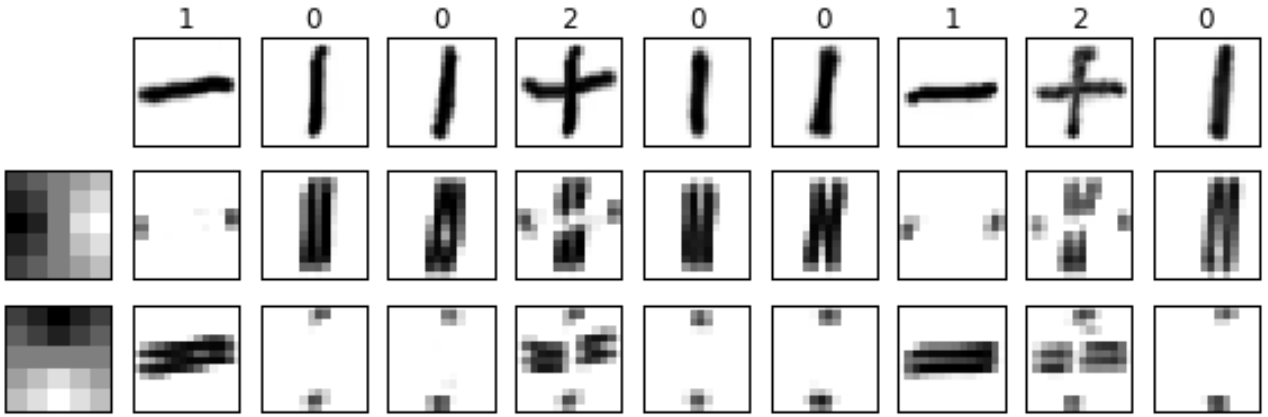
tf.nn.avg_pool : average

# Pooling layer를 적용한 결과

- 28x28 → 14x14 : 대략적인 특징만 유지 → 특징을 이용하여 image 분류




Convolution result




Pooling layer result

# 4.2 Image classification using convolution filter

## 4.2.1 Image classification using feature vectors

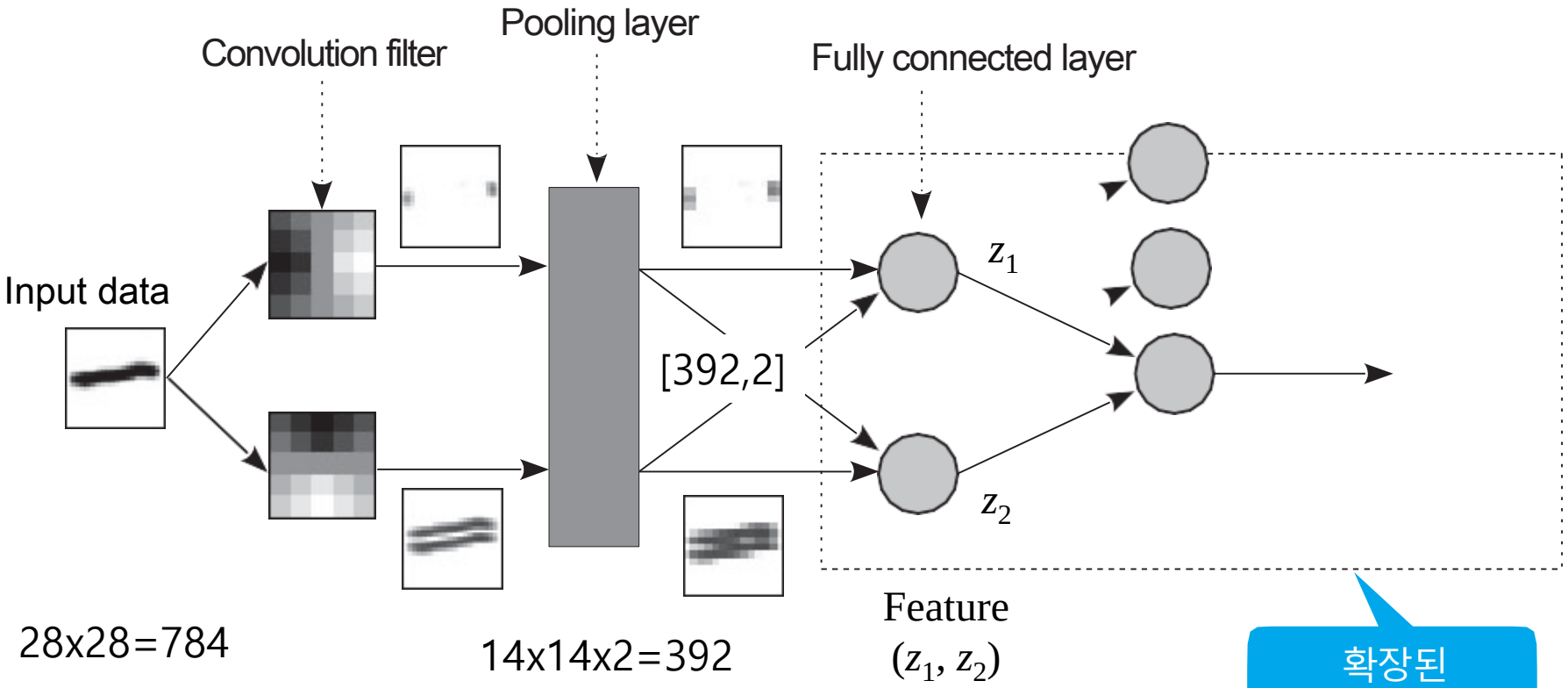



그림 4-12 이미지 데이터를 특징 변수로 변환하는 신경망

# Image classification using convolution filter

```
# [OCE-05] pooling layer의 output을 fully connected layer와 softmax function으로 된 '확장된 output layer'으로 input하는 계산식을 준비한다.
h_pool_flat = tf.reshape(h_pool, [-1, 392])
num_units1 = 392
num_units2 = 2
w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.zeros([num_units2]))
hidden2 = tf.nn.tanh(tf.matmul(h_pool_flat, w2) + b2)
w0 = tf.Variable(tf.zeros([num_units2, 3]))
b0 = tf.Variable(tf.zeros([3]))
p = tf.nn.softmax(tf.matmul(hidden2, w0) + b0)

# 나머지는 전과 동일
```

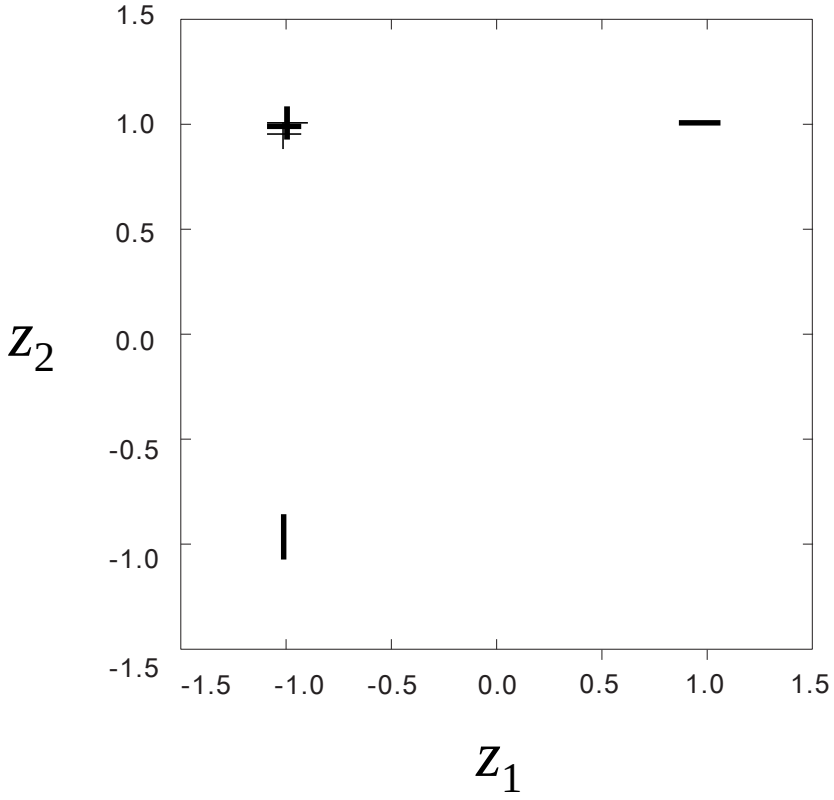
14x14x2

→ 정답율 100%

# 특징변수 (z1, z2) 산포도

```
# [OCE-09] 각각의 data의 특징변수 (z1, z2)를 산포도로 나타낸다.
hidden2_vals = sess.run(hidden2, feed_dict={x:images})
z1_vals = [[],[],[]]   # label 0,1,2
z2_vals = [[],[],[]]
for hidden2_val, label in zip(hidden2_vals, labels):
        label_num = np.argmax(label)
        z1_vals[label_num].append(hidden2_val[0])
        z2_vals[label_num].append(hidden2_val[1])
fig = plt.figure(figsize=(5,5))
subplot = fig.add_subplot(1,1,1)
subplot.scatter(z1_vals[0], z2_vals[0], s=200, marker='|')
subplot.scatter(z1_vals[1], z2_vals[1], s=200, marker='_')
subplot.scatter(z1_vals[2], z2_vals[2], s=200, marker='+')
```

# Fig 4-13 Feature variables ($z_1$, $z_2$)

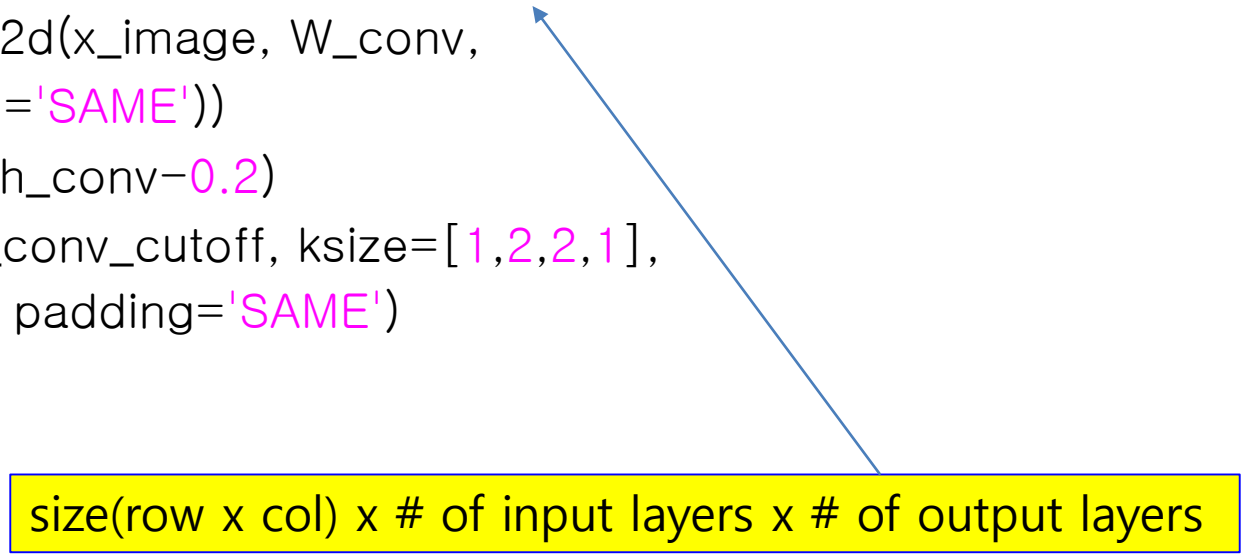


# 4.2.2 Convolution filter의 동적인 학습

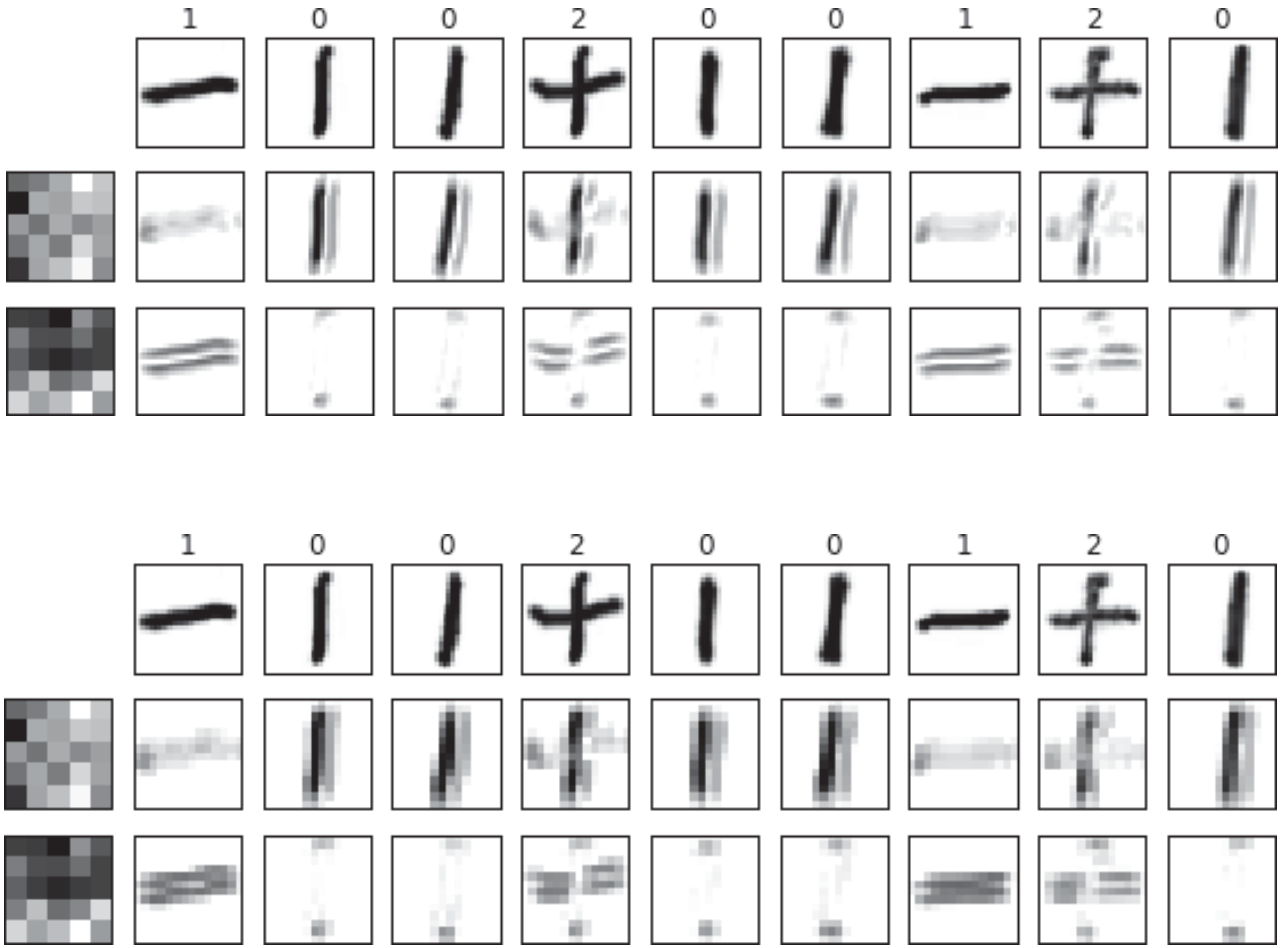
- 특징 추출
- 세로막대, 가로막대
  → 사선, 원형, 곡선, 점 … ?
  → 자동 학습
  → Filter 를 weights와 같이 학습
  → W_conv = constant → variable

# ORENIST dynamic filter example.ipynb

```
# [ODE-03] image data에 filter와 pooling layer을 적용하는 계산식을 준비한다.
x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])
W_conv = tf.Variable(tf.truncated_normal([5,5,1,2], stddev=0.1))
h_conv = tf.abs(tf.nn.conv2d(x_image, W_conv,
strides=[1,1,1,1], padding='SAME'))
h_conv_cutoff = tf.nn.relu(h_conv-0.2)
h_pool =tf.nn.max_pool(h_conv_cutoff, ksize=[1,2,2,1],
            strides=[1,2,2,1], padding='SAME')
```



# Fig 4-14 convolution filter를 동적으로 학습한 결과




Convolution result

Pooling layer result

# 4.3 convolution filter를 이용한 필기 문자 분류

4.3.1 세션 정보의 저장 기능
4.3.2 단층 CNN을 이용한 필기 문자 분류
4.3.3 동적으로 학습된 filter 확인

# 4.3.1 세션 정보의 저장 기능

- 트레이닝을 실행하는 도중에 세션 상태를 저장
- 트레이닝을 중단하더라도 나중에 다시 재개 가능
- saver = tf.train.Saver()

# MNIST dynamic filter classification.jpynb

- saver = tf.train.Saver()
- saver.save(sess, '/tmp/mdc_session', global_step=i)

- '/tmp/mdc_session' 디렉터리에 'mdc_session-<처리 횟수〉 및 mdc_session-<처리 횟수>.meta'라는 파일이 생성
- 이전 5회까지의 파일만 저장되고, 그보다 오래된 파일은 자동으로 삭제

- saver.restore(sess, '/tmp/mdc_session-4000')
- 복원
  - 먼저 각종 계산식을 원래 세션과 동일하게 정의
  - 세션을 준비하고 Variable을 초기화
  - tf(train.Saver) 오브젝트를 준비하고 restore 메소드를 호출

# Parameter optimization with saver

```
i = 0
for _ in range(4000):
        i += 1
        batch_xs, batch_ts = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, t: batch_ts})
        if i % 100 == 0:
                loss_val, acc_val = sess.run([loss, accuracy],
                feed_dict={x:mnist.test.images, t:mnist.test.labels})
                print ('Step: %d, Loss: %f, Accuracy: %f'
                        % (i, loss_val, acc_val))
                saver.save(sess, '/tmp/mdc_session', global_step=i)
```

# 4.3.2 단층 CNN을 이용한 필기 문자 분류

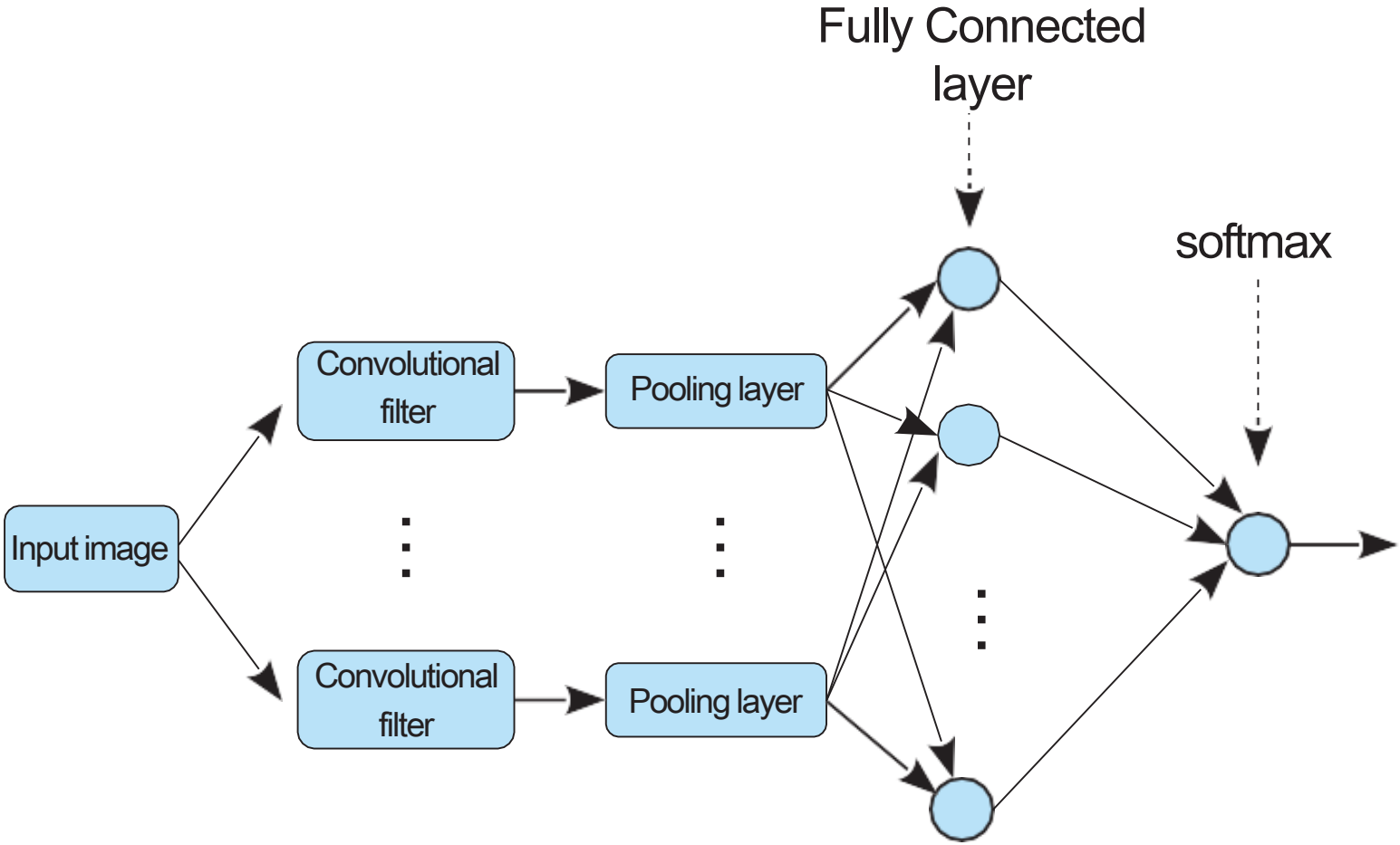



Fig 4-15 convolution filter와 pooling layer을 추가한 신경망

# MNIST dynamic filter classification.ipynb

- Filter 의 개수 : 임의로 지정 → 16

```
# [MDC-03] filter에 해당하는 Variable을 준비하고 input data에 filter와 pooling layer을 적용하는 계산식을 정의
num_filters = 16
x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])
W_conv = tf.Variable(tf.truncated_normal([5,5,1,num_filters],
        stddev=0.1))
h_conv = tf.nn.conv2d(x_image, W_conv,
        strides=[1,1,1,1], padding='SAME')      # 특징추출 (음수 가능)
h_pool =tf.nn.max_pool(h_conv, ksize=[1,2,2,1],
        strides=[1,2,2,1], padding='SAME')
```

```
# [MDC-04] pooling layer의 output을 전 결합층을 경유해서 소프트맥스 함수로 input하는 계산식을 정의한다.
h_pool_flat = tf.reshape(h_pool, [-1, 14*14*num_filters])

num_units1 = 14*14*num_filters
num_units2 = 1024

w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.zeros([num_units2]))
hidden2 = tf.nn.relu(tf.matmul(h_pool_flat, w2) + b2)

w0 = tf.Variable(tf.zeros([num_units2, 10]))
b0 = tf.Variable(tf.zeros([10]))
p = tf.nn.softmax(tf.matmul(hidden2, w0) + b0)
```

# [MDC-05] 오차 함수 loss, 트레이닝 알고리즘 train_step, 정답률 accuracy를 정의한다.

```
t = tf.placeholder(tf.float32, [None, 10])
loss = -tf.reduce_sum(t * tf.log(p))
train_step = tf.train.AdamOptimizer(0.0005).minimize(loss)
correct_prediction = tf.equal(tf.argmax(p, 1), tf.argmax(t, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

default = 0.001

# [MDC-06] 세션을 준비하고 Variable을 초기화한다.

```
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
saver = tf.train.Saver()
```

```
# [MDC-07] 파라미터 최적화를 4000회 반복한다
# 최종적으로 테스트 세트에 대해 약 98%의 정답률을 얻을 수 있다.
i = 0
for _ in range(4000):
        i += 1
        batch_xs, batch_ts = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, t: batch_ts})
        if i % 100 == 0:
        loss_val, acc_val = sess.run([loss, accuracy],
                feed_dict={x:mnist.test.images, t:mnist.test.labels})
        print ('Step: %d, Loss: %f, Accuracy: %f'
                % (i, loss_val, acc_val))
        saver.save(sess, '/tmp/mdc_session', global_step=i)
```

# 4.3.3 동적으로 학습된 filter 확인

```
# [MDR-06] 세션을 준비하고 Variable을 초기화한 후, 최적화 처리를 마친 세션을 복원한다.
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
saver = tf.train.Saver()
saver.restore(sess, '/tmp/mdc_session-4000')
# [MDR-07] convolution filter의 값과 최초 9개의 image data에 대해 convolution filter와 pooling layer을 적용한 결과를 얻는다.

filter_vals, conv_vals, pool_vals = sess.run(
      [W_conv, h_conv, h_pool],
      feed_dict={x:mnist.test.images[:9]})
```

```
# [MDR-08] convolution filter를 적용한 결과를 image로 output한다.
# convolution filter를 적용한 후에는 픽셀값이 음의 값을 갖는 경우도 있으므로 배경(픽셀값 0)부분이 흰색이 되지 않는다는 점에 주의

fig = plt.figure(figsize=(10,num_filters+1))

for i in range(num_filters):
      subplot = fig.add_subplot(num_filters+1, 10,
            10*(i+1)+1)
      subplot.set_xticks([])
      subplot.set_yticks([])
      subplot.imshow(filter_vals[:,:,0,i],
            cmap=plt.cm.gray_r, interpolation='nearest')
```

```
for i in range(9):
    subplot = fig.add_subplot(num_filters+1, 10, i+2)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' %
        np.argmax(mnist.test.labels[i]))
    subplot.imshow(mnist.test.images[i].reshape((28,28)),
        vmin=0, vmax=1,
        cmap=plt.cm.gray_r, interpolation='nearest')

for f in range(num_filters):
    subplot = fig.add_subplot(num_filters+1, 10,
        10*(f+1)+i+2)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.imshow(conv_vals[i,:,:,f],
        cmap=plt.cm.gray_r, interpolation='nearest')
```

# [MDR-09] 마찬가지로 convolution filter와 pooling layer을 적용한 결과를 image로 output한다.

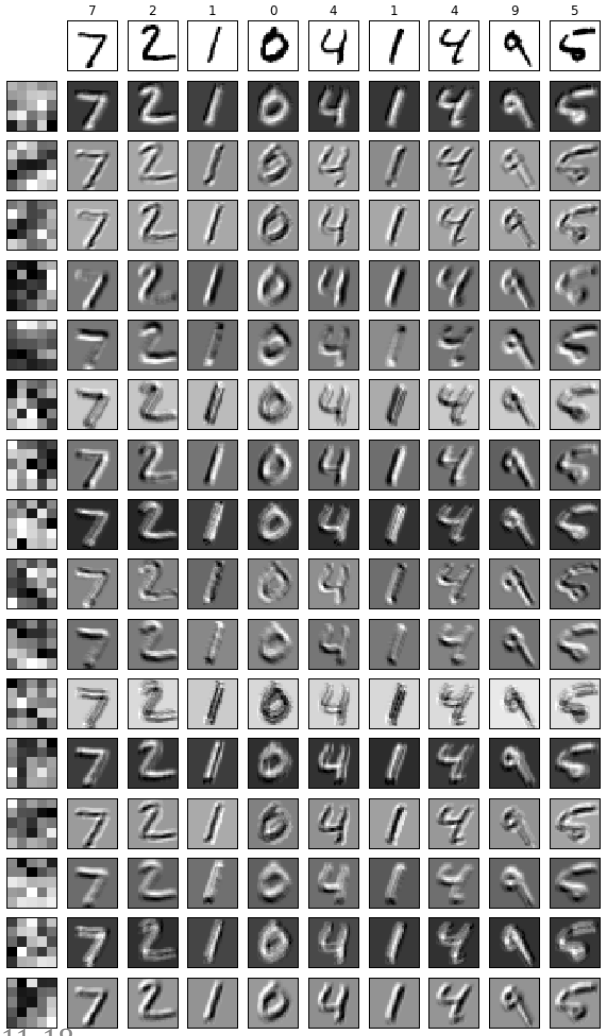
```
fig = plt.figure(figsize=(10,num_filters+1))
for i in range(num_filters):
      subplot = fig.add_subplot(num_filters+1, 10,
              10*(i+1)+1)
      subplot.set_xticks([])
      subplot.set_yticks([])
      subplot.imshow(filter_vals[:,:,0,i],
            cmap=plt.cm.gray_r, interpolation='nearest')
```

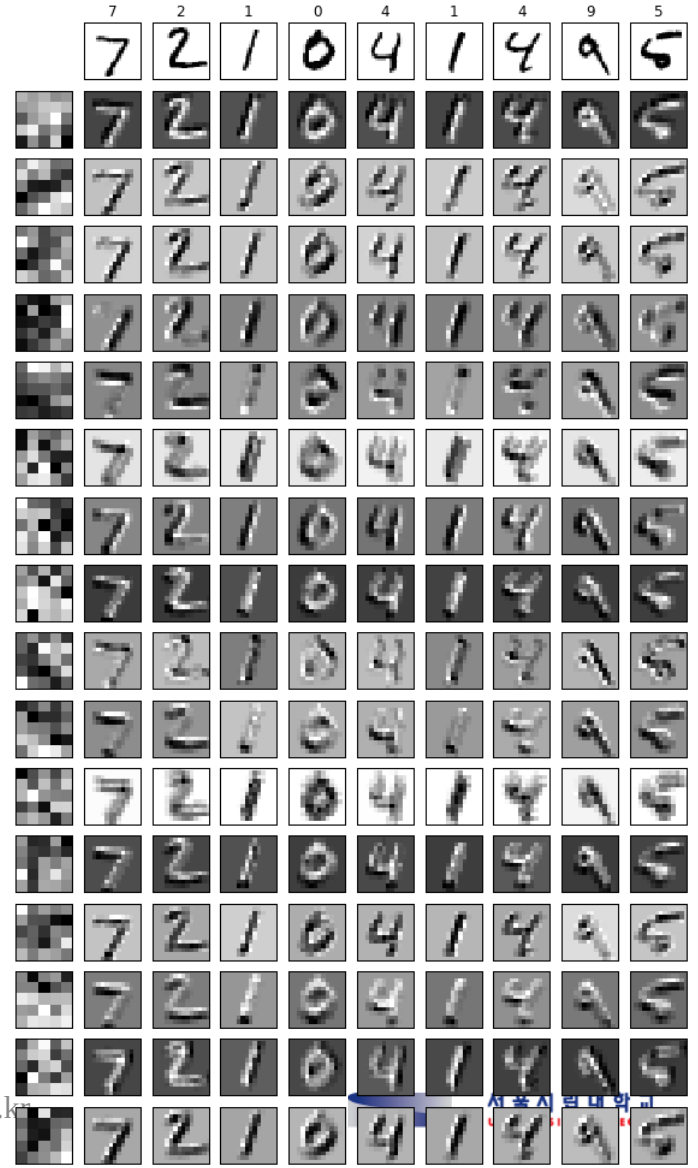
```
for i in range(9):
      subplot = fig.add_subplot(num_filters+1, 10, i+2)
      subplot.set_xticks([])
      subplot.set_yticks([])
      subplot.set_title('%d' %
            np.argmax(mnist.test.labels[i]))
     subplot.imshow(mnist.test.images[i].reshape((28,28)),
            vmin=0, vmax=1,
            cmap=plt.cm.gray_r, interpolation='nearest')

for f in range(num_filters):
      subplot = fig.add_subplot(num_filters+1, 10,
             10*(f+1)+i+2)
      subplot.set_xticks([])
      subplot.set_yticks([])
      subplot.imshow(pool_vals[i,:,:,f],
            cmap=plt.cm.gray_r, interpolation='nearest')
```

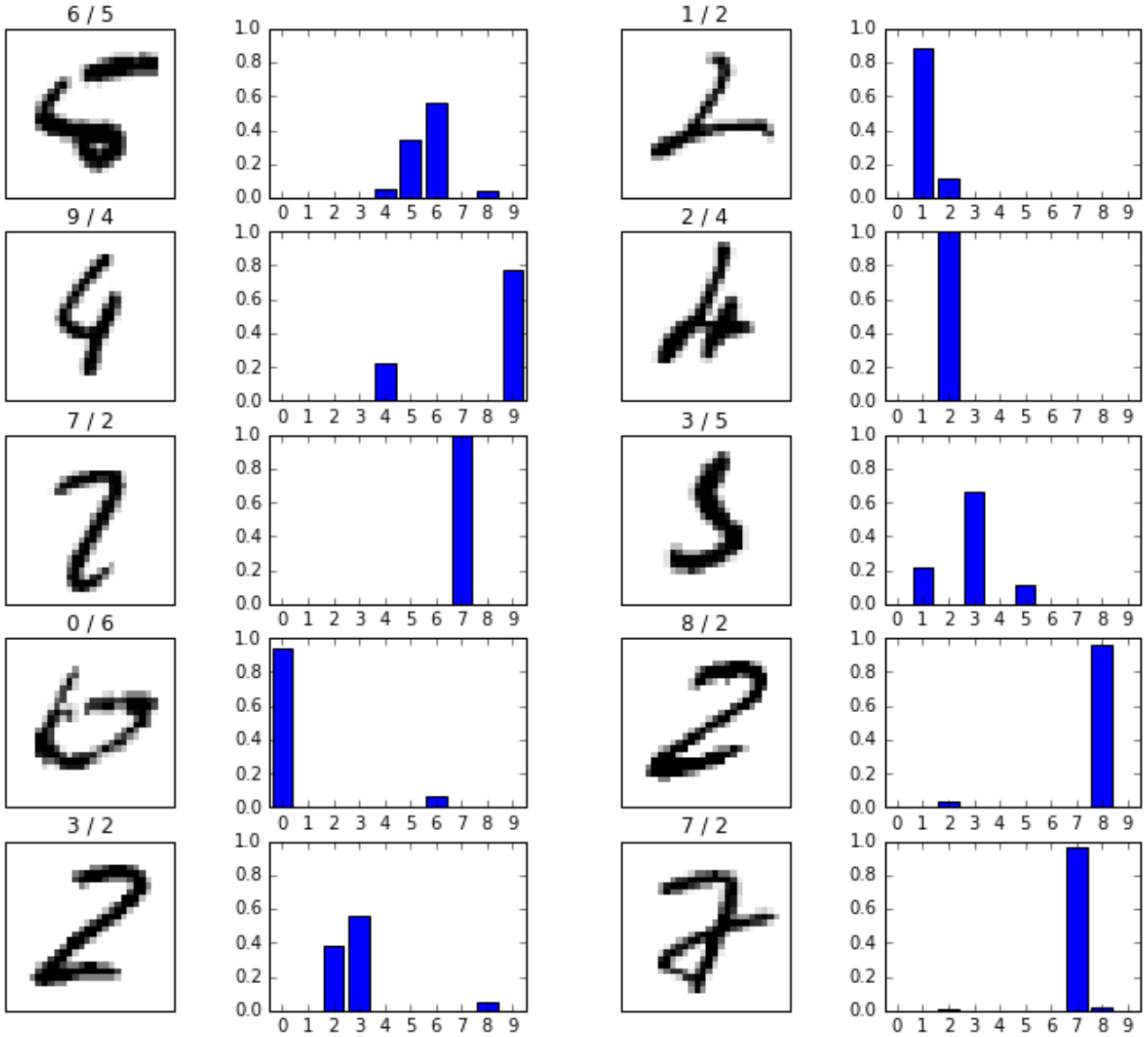
# Filters

- 배경색이 하얗지 않은 이유 : 음수 픽셀값



```
# [MDR-10] 올바르게 분류할 수 없었던 몇몇 data에 대해 각각의 문자일 확률을 확인
fig = plt.figure(figsize=(12,10))
c=0
for (image, label) in zip(mnist.test.images,mnist.test.labels):
        p_val = sess.run(p, feed_dict={x:[image]})
        pred = p_val[0]
        prediction, actual = np.argmax(pred), np.argmax(label)
        if prediction == actual:
                continue
        subplot = fig.add_subplot(5,4,c*2+1)
        subplot.set_xticks([])
        subplot.set_yticks([])
        subplot.set_title('%d / %d' % (prediction, actual))
        subplot.imshow(image.reshape((28,28)), vmin=0, vmax=1,
                cmap=plt.cm.gray_r, interpolation="nearest")
        subplot = fig.add_subplot(5,4,c*2+2)
        subplot.set_xticks(range(10))
        subplot.set_xlim(-0.5,9.5)
        subplot.set_ylim(0,1)
        subplot.bar(range(10), pred, align='center')
        c += 1
        if c == 10:
                break
```


6 / 5
1 / 2
9 / 4
2 / 4
7 / 2
3 / 5
0 / 6
8 / 2
3 / 2
7 / 2