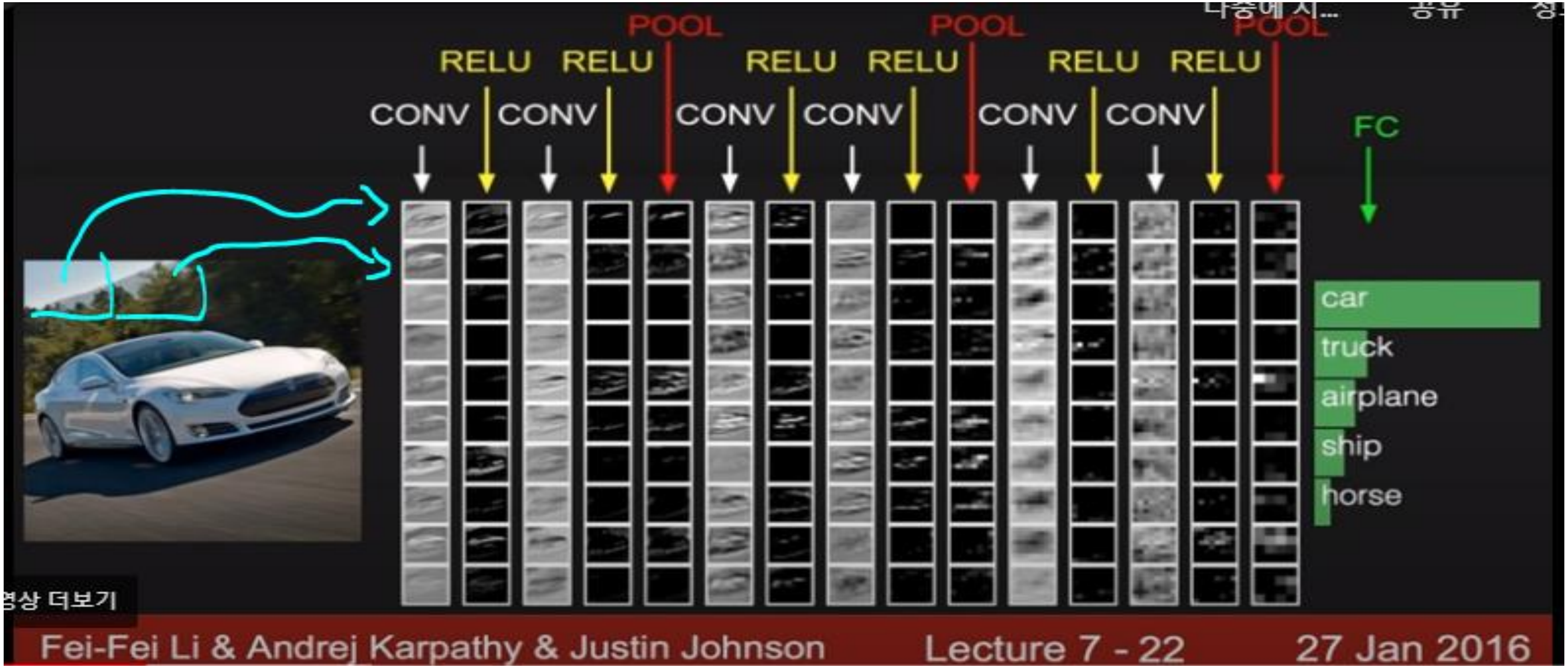
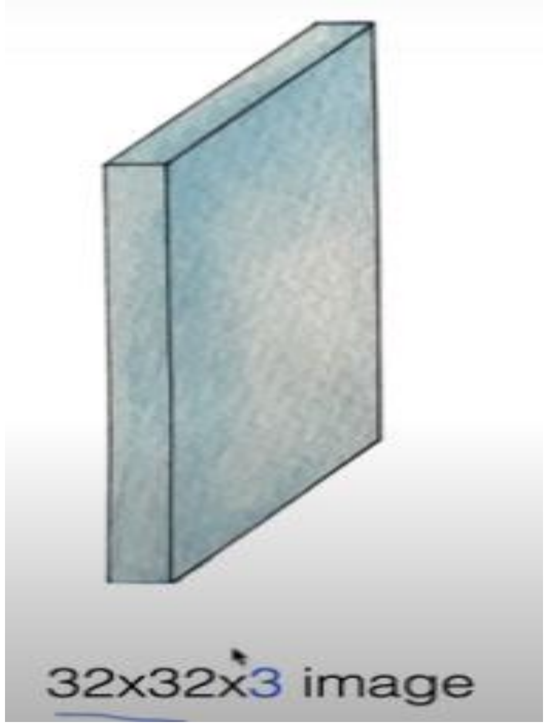
Convoluntional Neural Networks

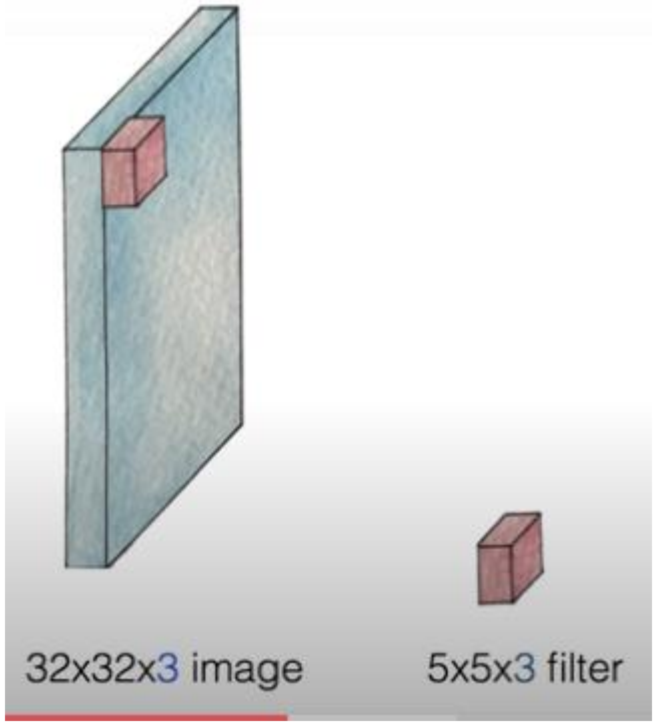
전체구조



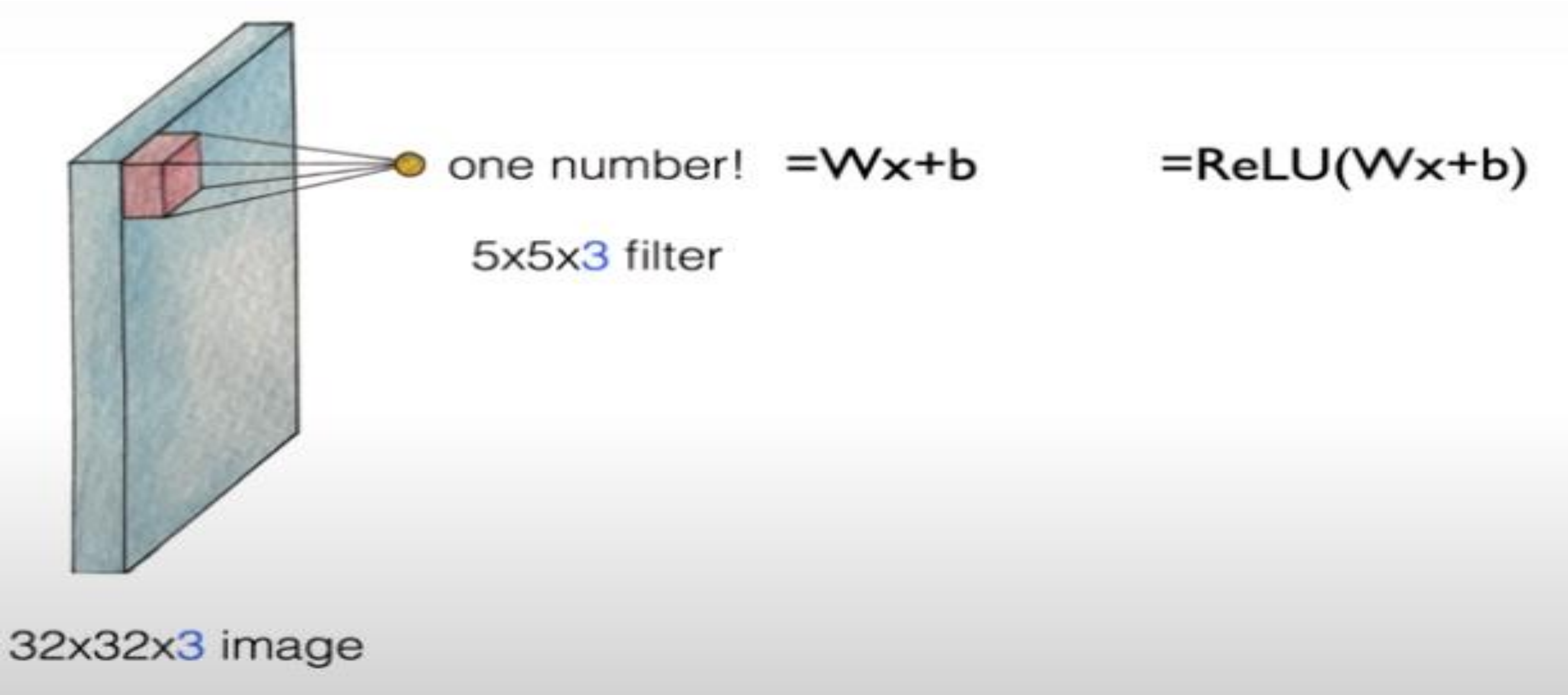
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 22 27 Jan 2016



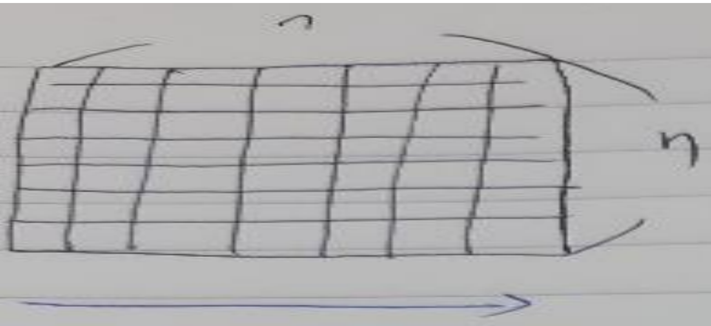
Input image (R,G,B)



Input image의 일부분 filter (R,G,B)
Filter 크기 조정 가능



끝까지 filter 단위로 one number들을 뽑아낸다.



filter : 3×3

∴ Output = 5×5.

Stride 은 1
(몇 칸씩 움직이는가를)

⇙

일반화

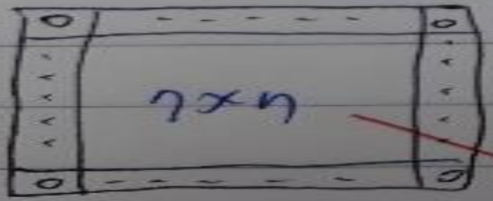
Input img : N×N , filter : F×F.

Output : (N−F) / Stride + 1 "

↳ 이미지가 계속 작아짐 & 모서리 알려주기.

를

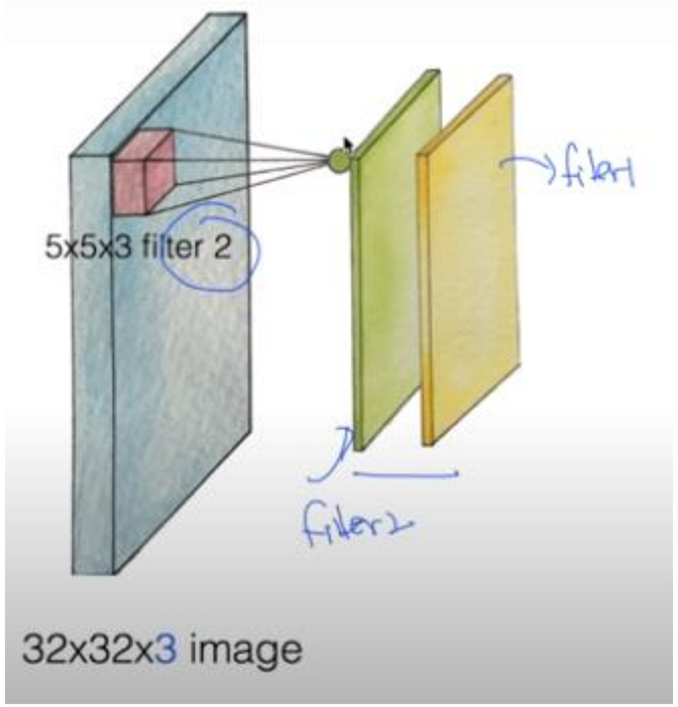
"Padding" ~> 겉에 0으로 덮어준다.

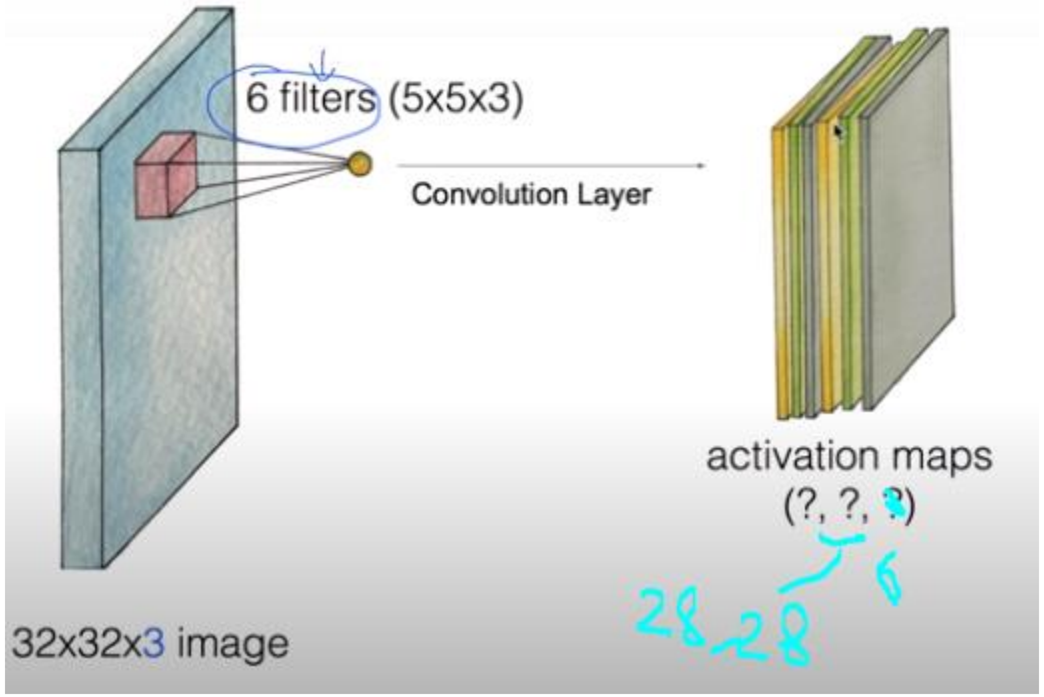


input → 9×9 filter → 3×3

Output → 7×7

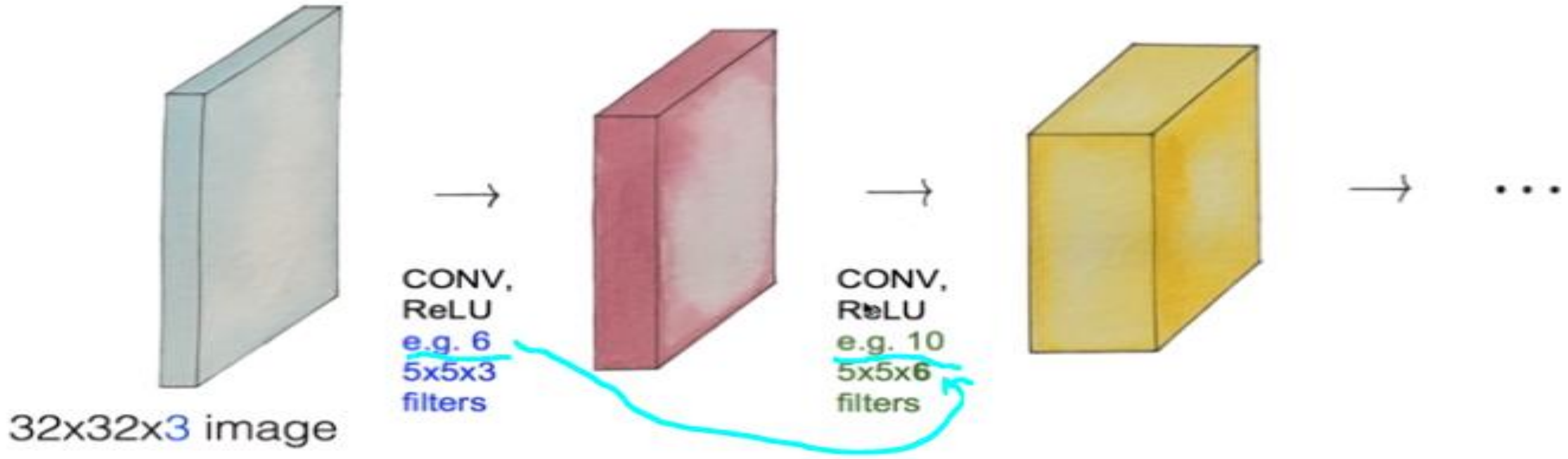
원래 이미지와 같은 Size.





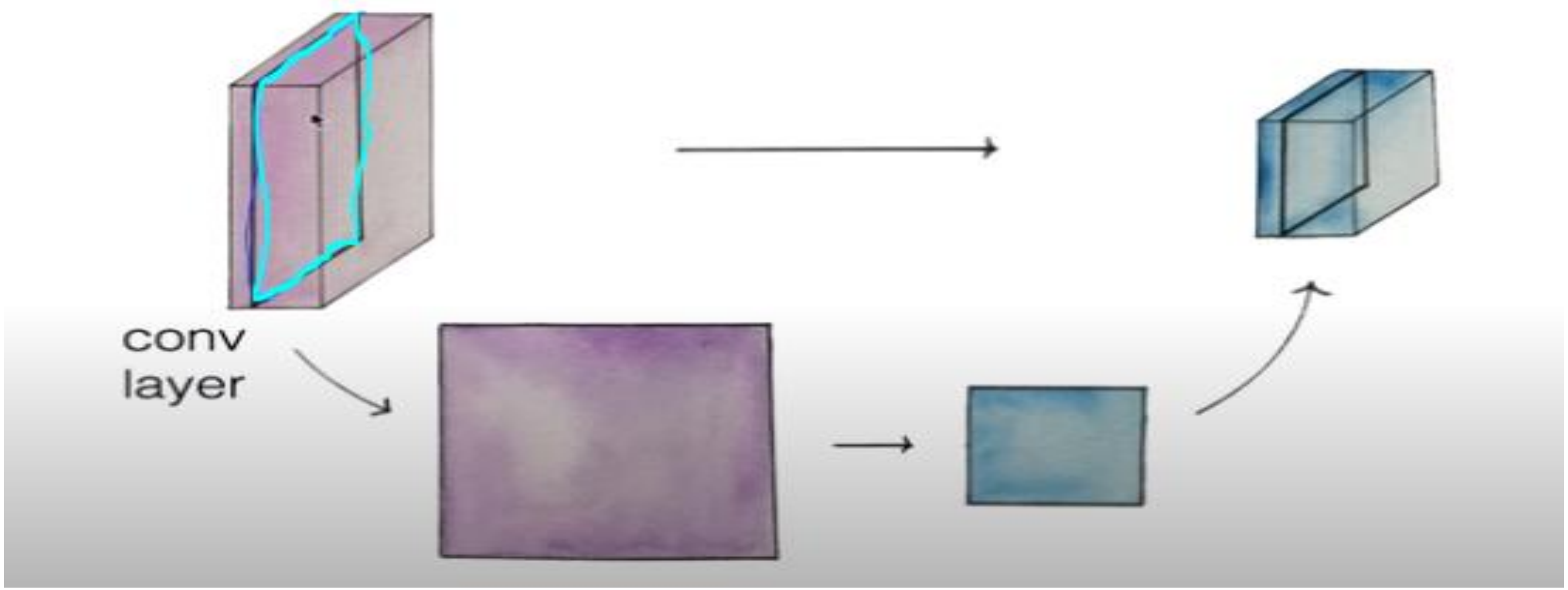
여러 개의 필터를 적용해 만든 activation maps
Input: 32x32, filter:5x5 이므로
Output : (32-5)/1+1=28

Convolution layers



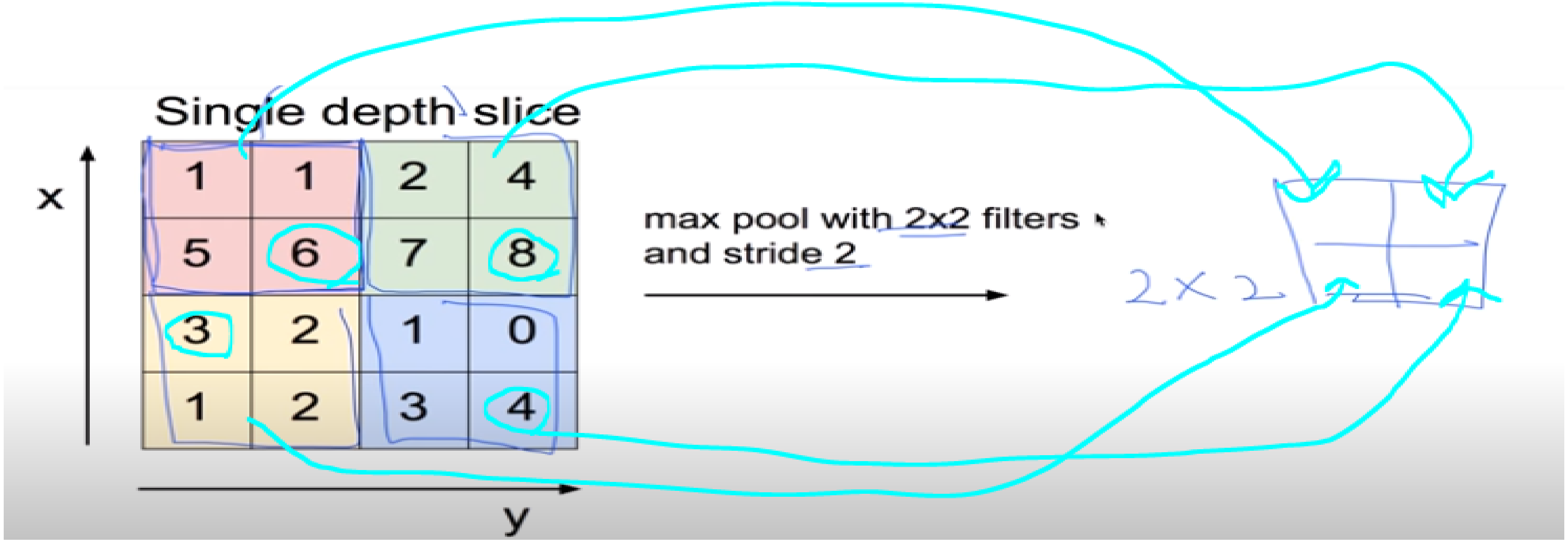
e.g. = filter의 개수

Pooling layer(sampling)



Conv layer에서 1개씩 뽑아내서 resize(sampling)

MAX POOLING



Single depth slice

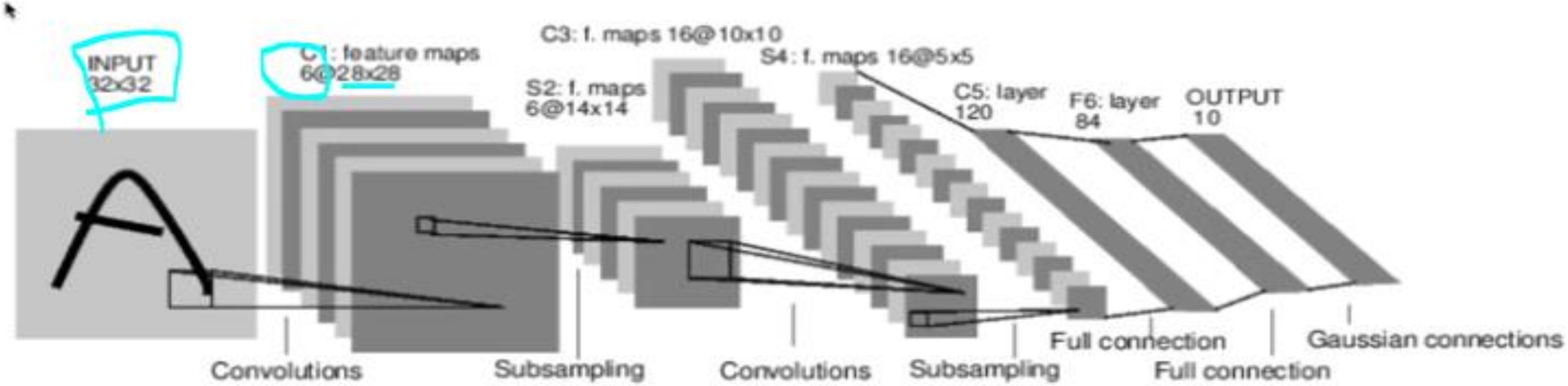
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

Filter에서 가장 큰 값만 뽑는다.

Case Study: LeNet-5

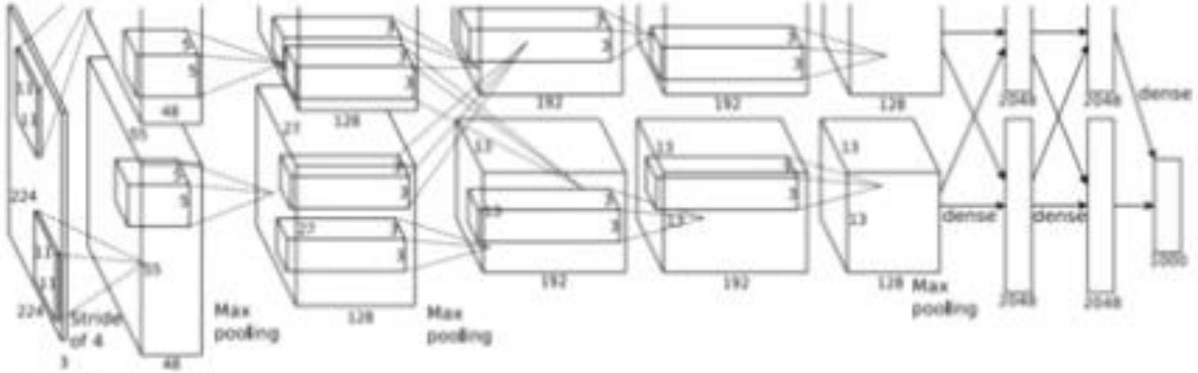
[LeCun et al., 1998]



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]

Case Study: AlexNet

[Krizhevsky et al. 2012]

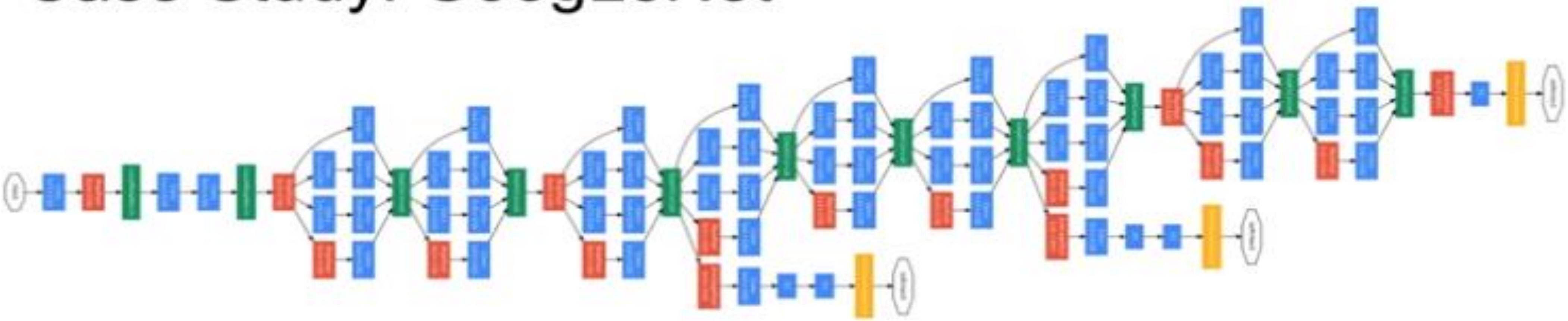


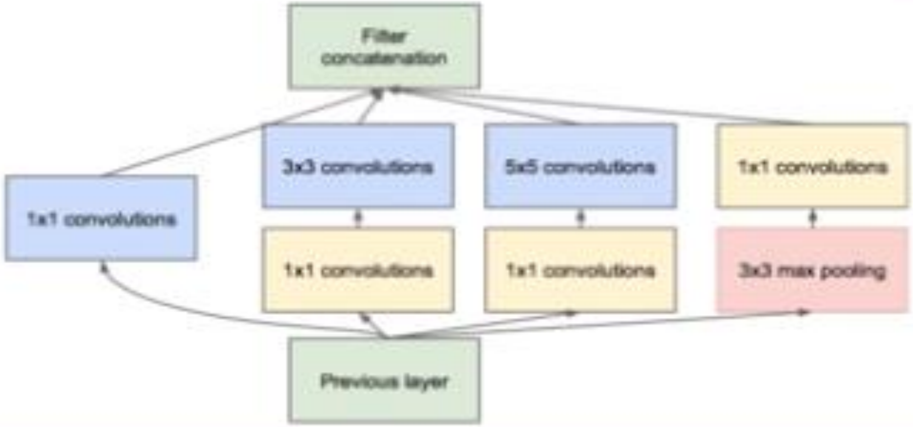
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

최근에는 normalization 안함

Case Study: GoogLeNet

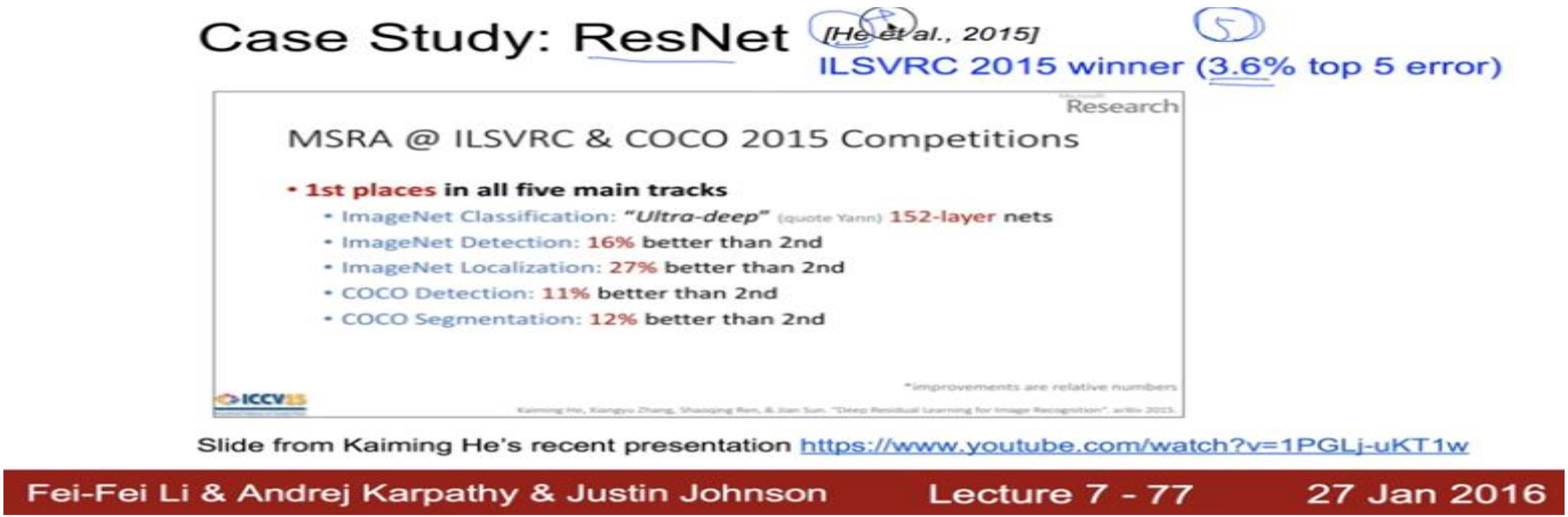
[Szegedy et al., 2014]





Inception module

ILSVRC 2014 winner (6.7% top 5 error)



Case Study: ResNet

[He et al., 2015]

5

ILSVRC 2015 winner (3.6% top 5 error)

Research

MSRA @ ILSVRC & COCO 2015 Competitions

- 1st places in all five main tracks
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

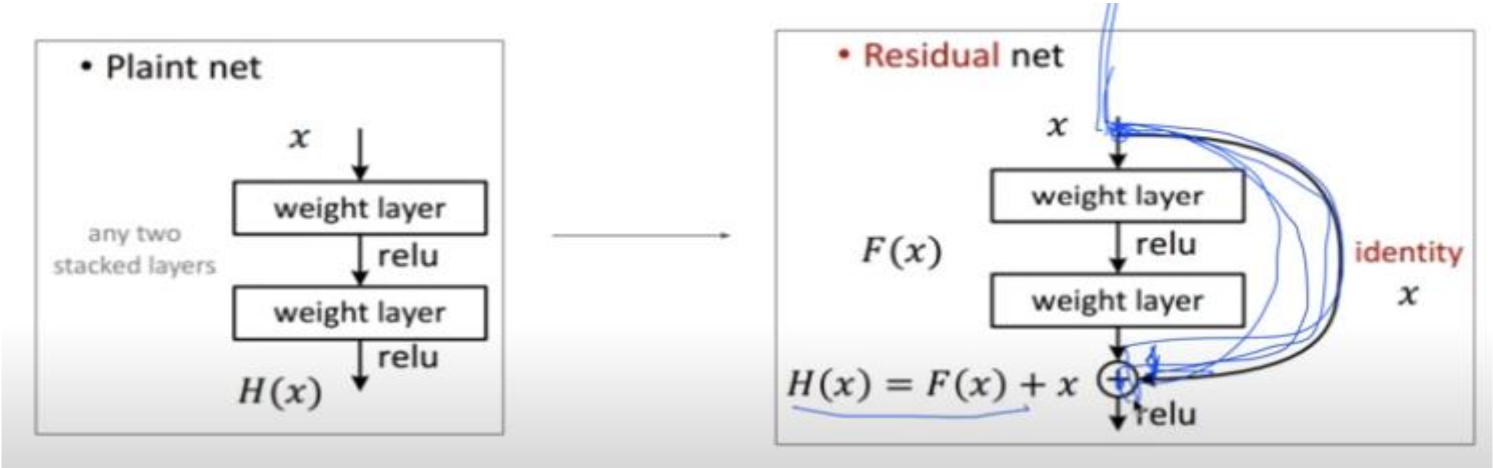
*improvements are relative numbers

ICCV15

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

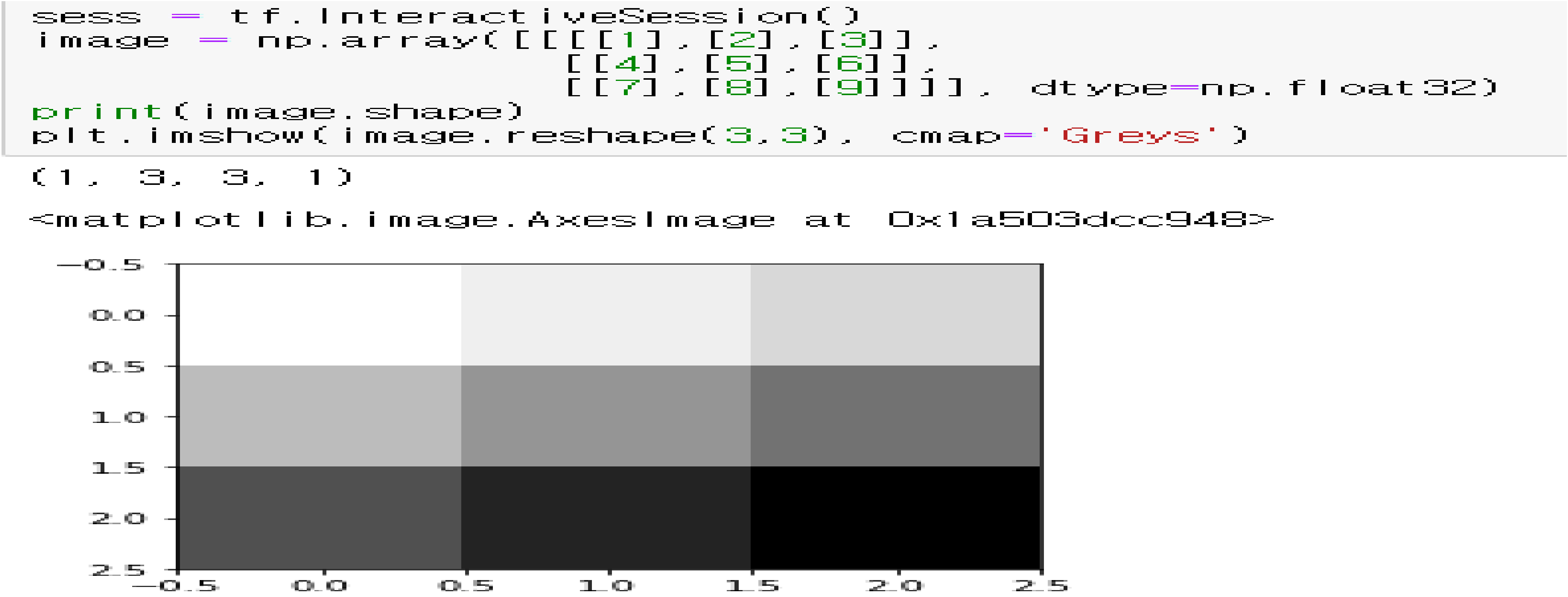
Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 77 27 Jan 2016



인간(에러율 5%)보다
좋은 성능

CNN 기본 - input



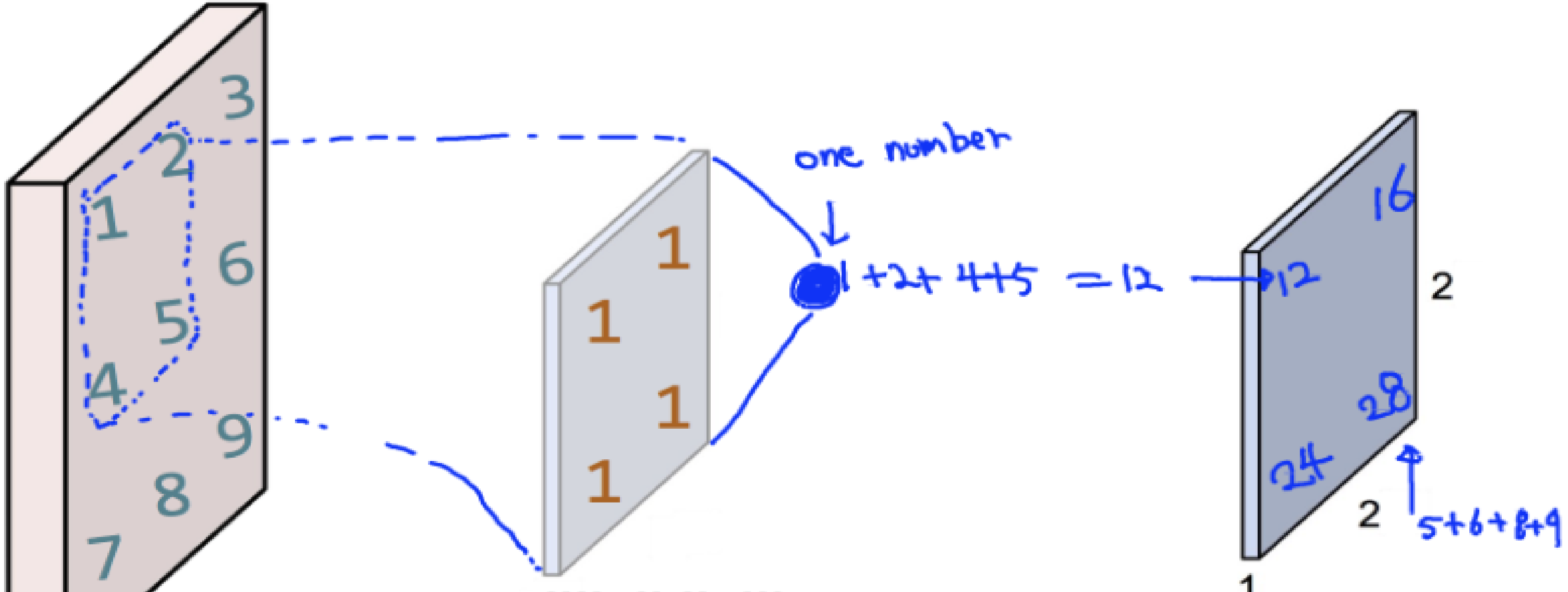
```
sess = tf.InteractiveSession()
image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print(image.shape)
plt.imshow(image.reshape(3,3), cmap='Greys')
```

```
(1, 3, 3, 1)

<matplotlib.image.AxesImage at 0x1a503dcc948>
```

Input : N개 이미지, 사이즈, 색상
(1,3,3,1) 1개 img, 3x3, 1(흑백)

Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID

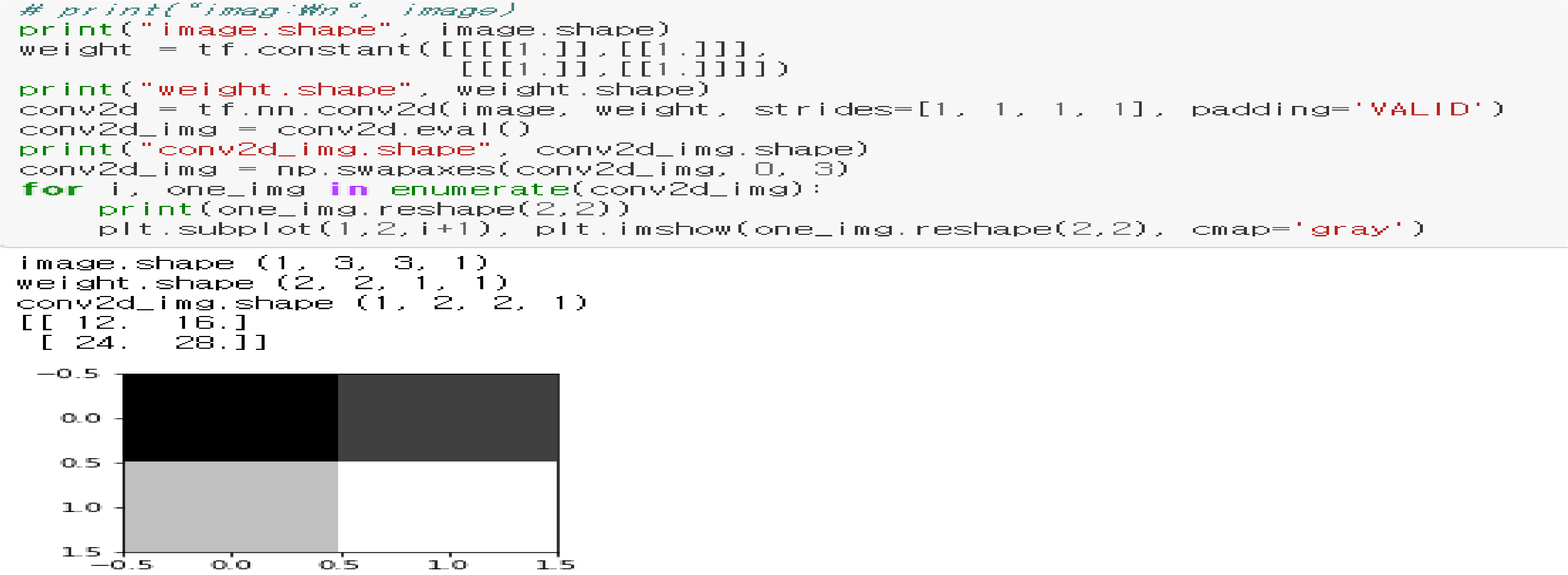


```
[[[[1],[2],[3]],
  [[4],[5],[6]],
  [[7],[8],[9]]]]
 shape=(1,3,3,1)
```

```
[[[[1.]],[[1.]]],
  [[[1.]],[[1.]]]]
shape=(2,2,1,1)
```

Weight(=filter)

CNN 기본 – filter , output



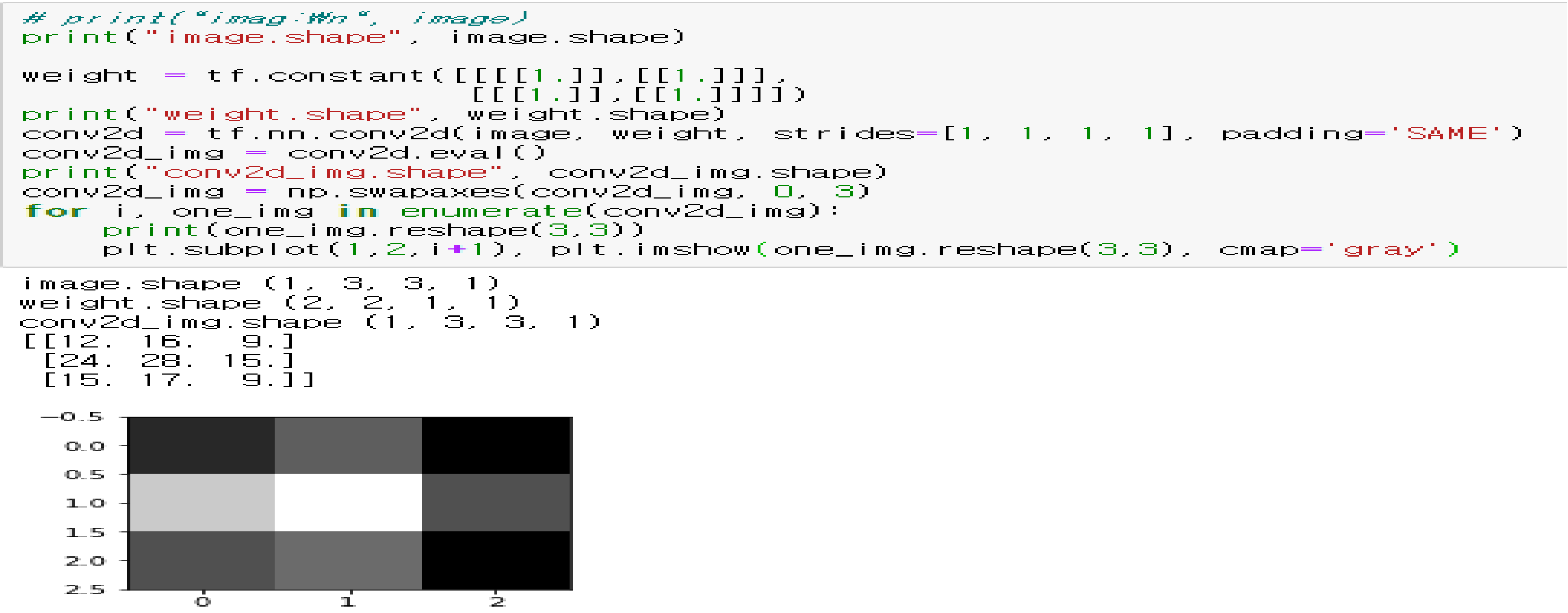
```
# print("imag:₩n", image)
print("image.shape", image.shape)
weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 2, 2, 1)
[[ 12.   16.]
 [ 24.   28.]]
```

Filter(=weight) : 사이즈, 색상, N개 이미지
(2,2,1,1) 2x2, 1(흑백), 1개 img

Output : N개 이미지, 사이즈, 색상
(1,2,2,1) 1개 img, 2x2, 1(흑백)

CNN 기본 – padding(output)

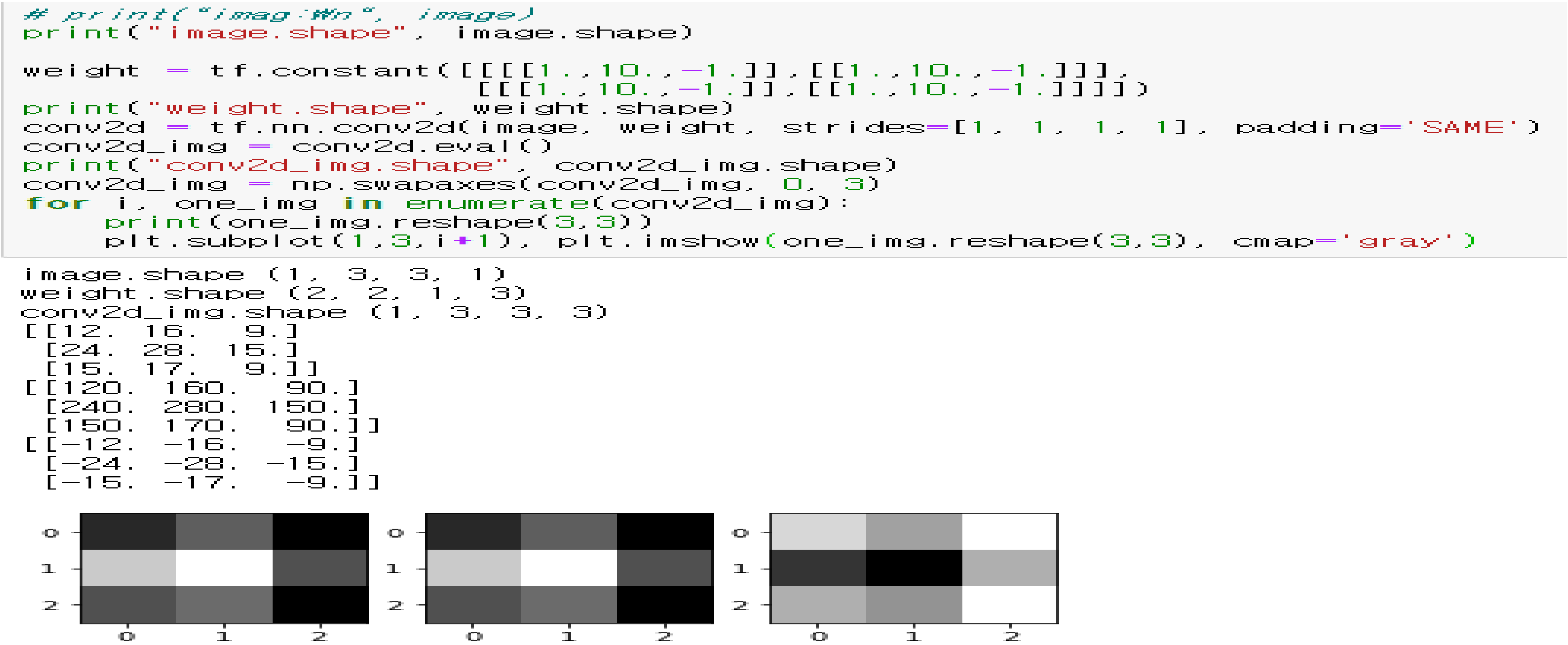


```
# print("imag:₩n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
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))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 3, 3, 1)
[[12. 16.  9.]
 [24. 28. 15.]
 [15. 17.  9.]]
```

padding = ‘SAME’ 을 통해 input이미지와 output 이미지 사이즈가 같게 된다.

CNN 기본 – 필터 3장



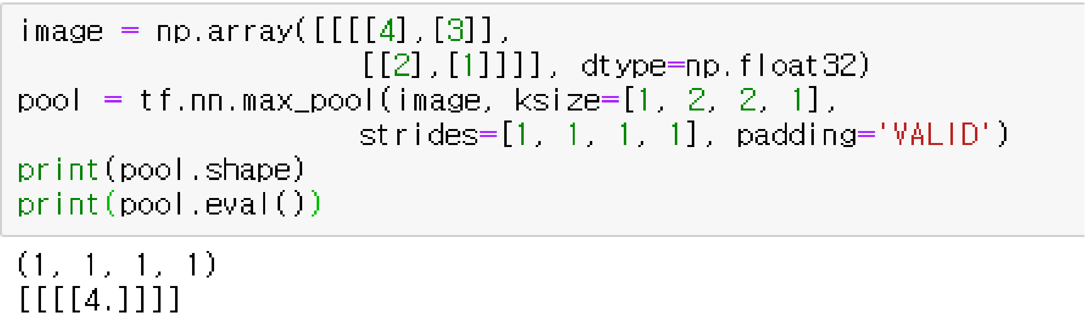
```
# print("imag:₩n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.,10.,-1.]],[[1.,10.,-1.]]],
                      [[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
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))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
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.]]
```

Filter(=weight) : 사이즈, 색상, N개 이미지
(2,2,1,3) 2x2, 1(흑백), 3개 img

CNN 기본 – Max Pooling

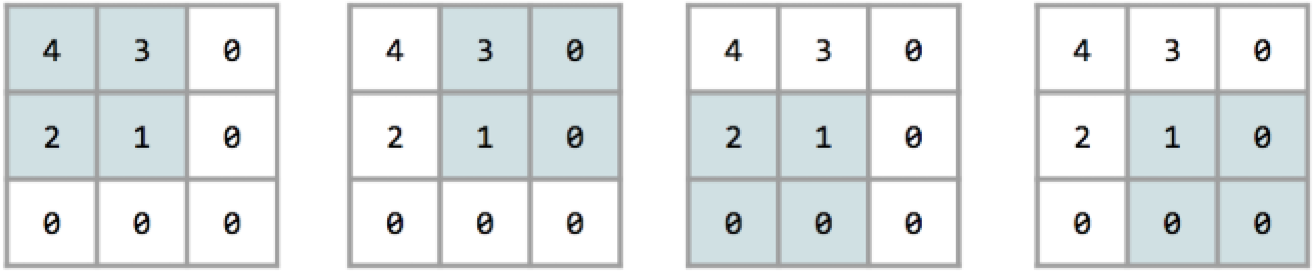
4	3
2	1



```
image = np.array([[[[4],[3]],
                   [[2],[1]]]], dtype=np.float32)
pool  = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
                       strides=[1, 1, 1, 1], padding='VALID')
print(pool.shape)
print(pool.eval())
```

```
(1, 1, 1, 1)
[[[[4.]]]]
```

SAME: Zero paddings

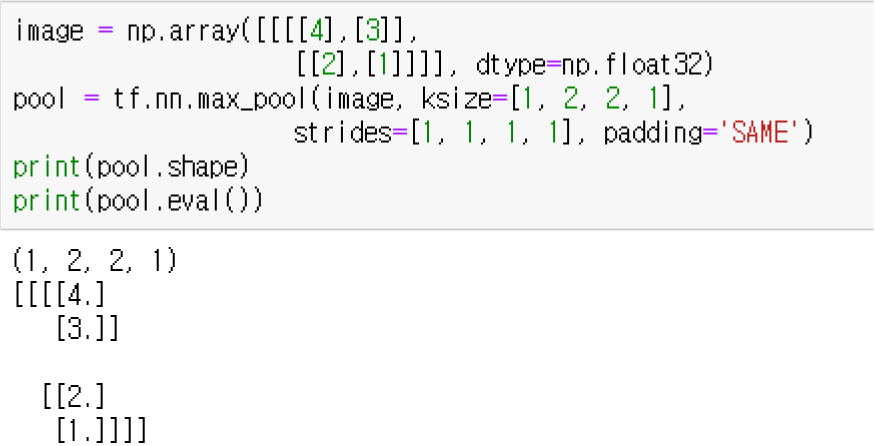


4	3	0
2	1	0
0	0	0

4	3	0
2	1	0
0	0	0

4	3	0
2	1	0
0	0	0

4	3	0
2	1	0
0	0	0



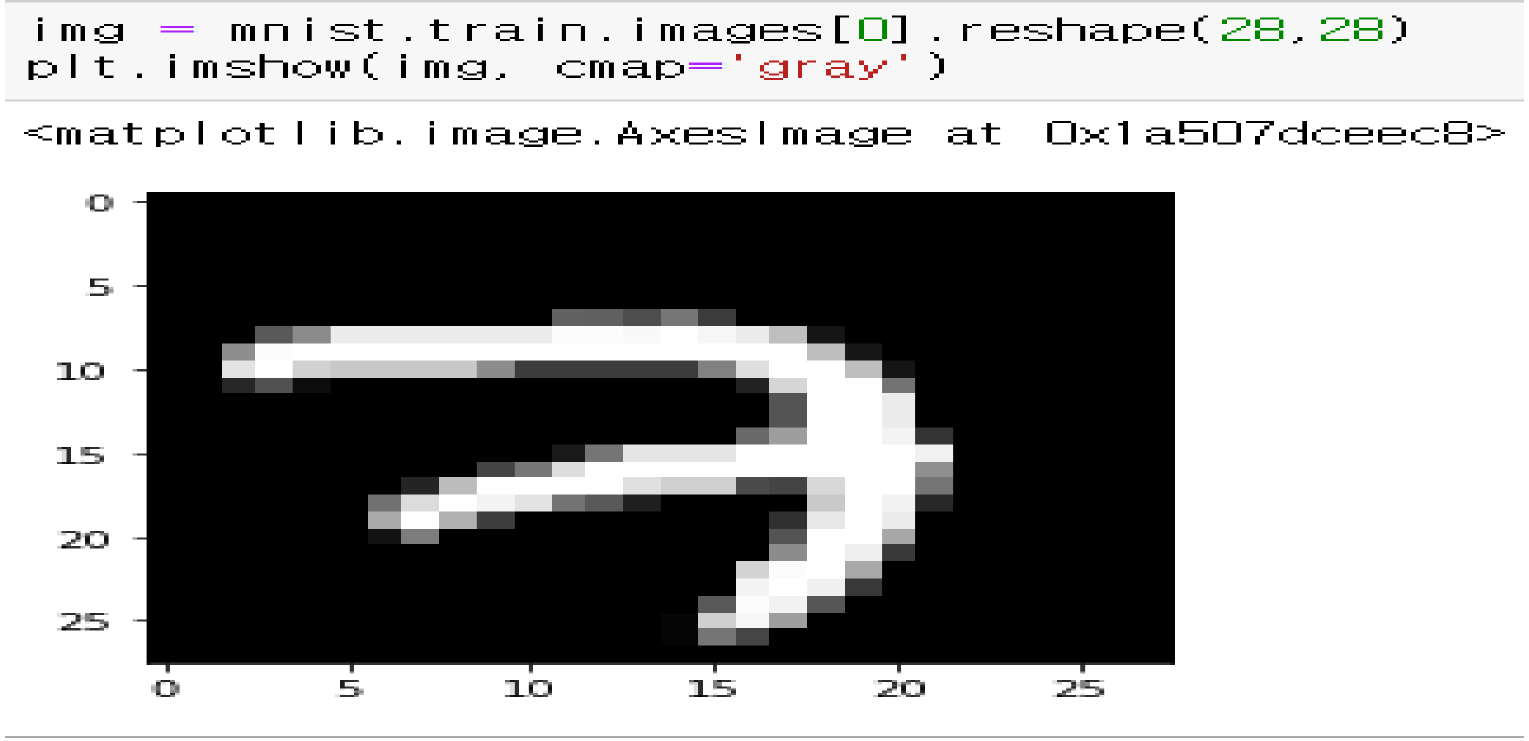
```
image = np.array([[[[4],[3]],
                   [[2],[1]]]], dtype=np.float32)
pool  = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
                       strides=[1, 1, 1, 1], padding='SAME')
print(pool.shape)
print(pool.eval())
```

```
(1, 2, 2, 1)
[[[[4.]
   [3.]]

  [[2.]
   [1.]]]]
```

Output(=ksize) : img에 패딩한 것을 2x2사이즈로 stride 만큼 보면서
max_pool(최대값)반환

padding = ‘SAME’하면
패딩 적용한 결과

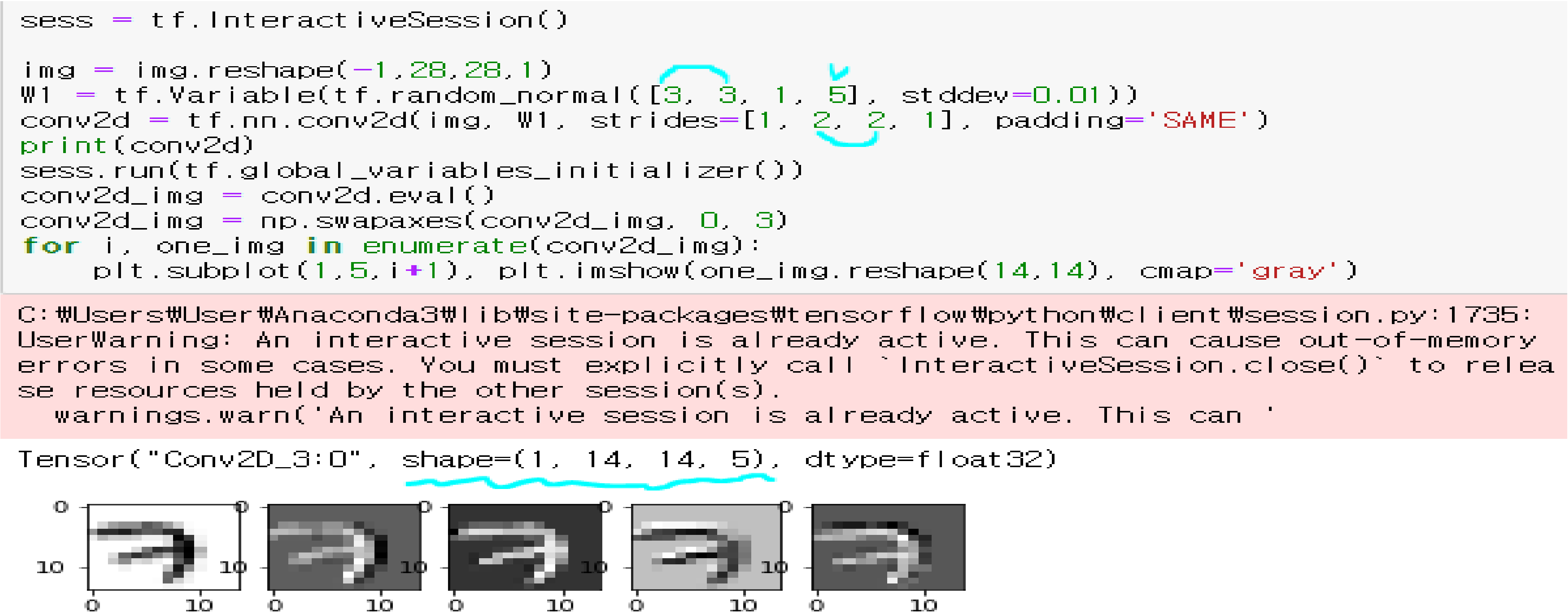
MNIST



```
img = mnist.train.images[0].reshape(28,28)
plt.imshow(img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1a507dceec8>
```

MNIST Convolution layer



```
sess = tf.InteractiveSession()

img = img.reshape(-1,28,28,1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 5], stddev=0.01))
conv2d = tf.nn.conv2d(img, W1, strides=[1, 2, 2, 1], padding='SAME')
print(conv2d)
sess.run(tf.global_variables_initializer())
conv2d_img = conv2d.eval()
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(14,14), cmap='gray')
```

```
C:₩Users₩User₩Anaconda3₩lib₩site-packages₩tensorflow₩python₩client₩session.py:1735:
UserWarning: An interactive session is already active. This can cause out-of-memory
errors in some cases. You must explicitly call `InteractiveSession.close()` to relea
se resources held by the other session(s).
  warnings.warn('An interactive session is already active. This can '
```

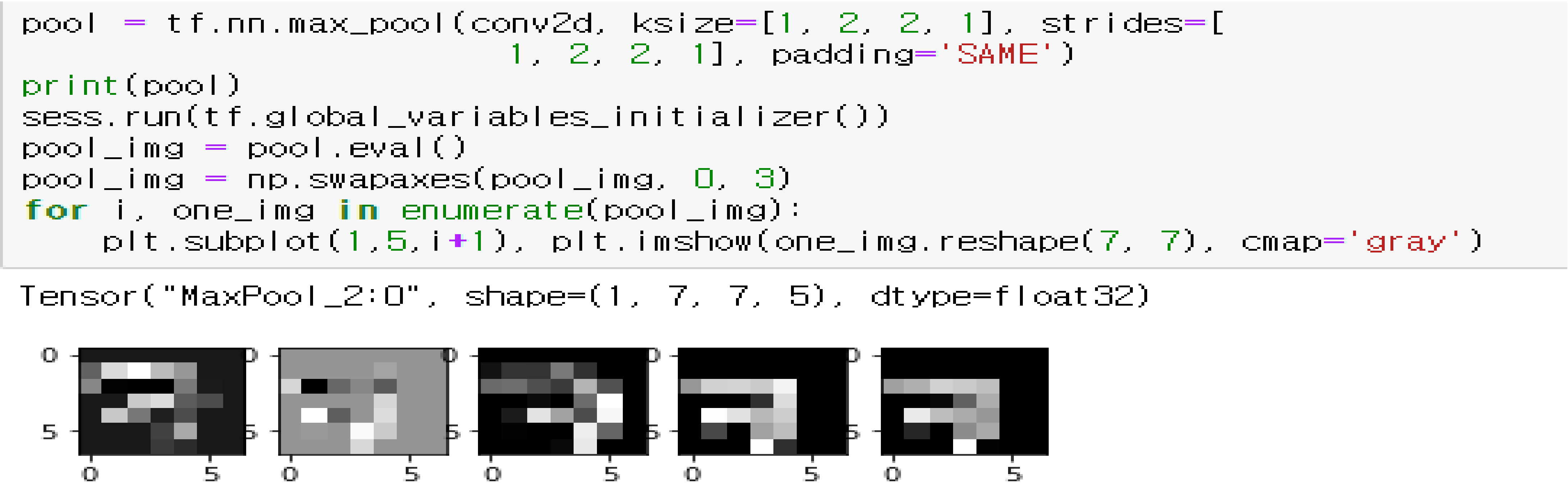
```
Tensor("Conv2D_3:0", shape=(1, 14, 14, 5), dtype=float32)
```

Filter(=weight) : 사이즈, 색상, N개 이미지
(3,3,1,5) 3x3, 1(흑백), 5개 img

Stride = 2x2

(28-3)/2+1=14

MNIST Max Pooling

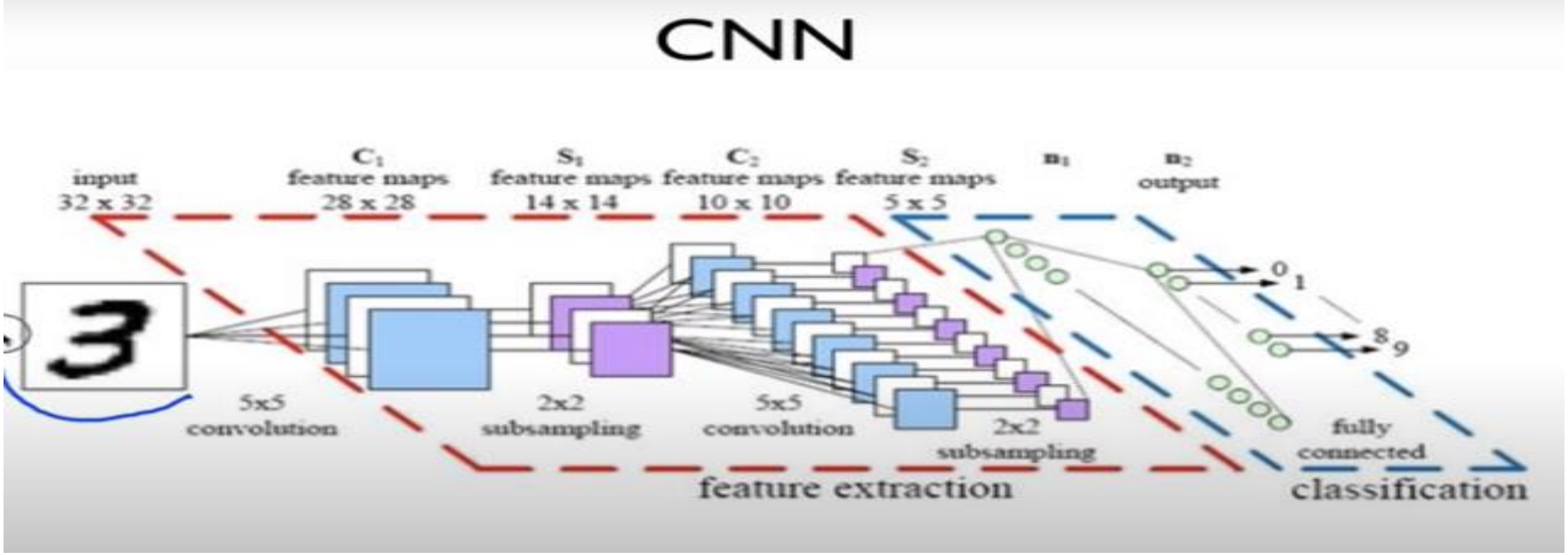


```
pool = tf.nn.max_pool(conv2d, ksize=[1, 2, 2, 1], strides=[
                        1, 2, 2, 1], padding='SAME')
print(pool)
sess.run(tf.global_variables_initializer())
pool_img = pool.eval()
pool_img = np.swapaxes(pool_img, 0, 3)
for i, one_img in enumerate(pool_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(7, 7), cmap='gray')
```

```
Tensor("MaxPool_2:0", shape=(1, 7, 7, 5), dtype=float32)
```

Output(=ksize) : img에 패딩한 것을 2x2사이즈로 stride 만큼 보면서 max_pool(최대값)반환

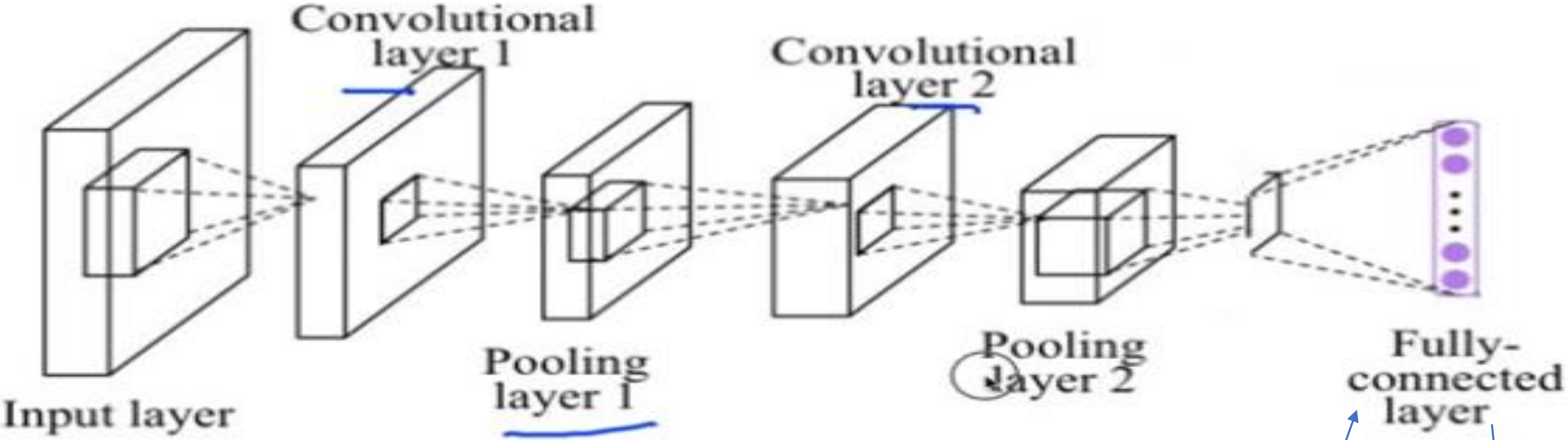
CNN MNIST: 99%



이미지를 받아 컨벌루션 통과 후 subsampling 반복 후 fully connected에 연결한다.

Simple CNN

98% 정확도



입력 : nx784

nx28x28x32

nx14x14x32

nx14x14x64

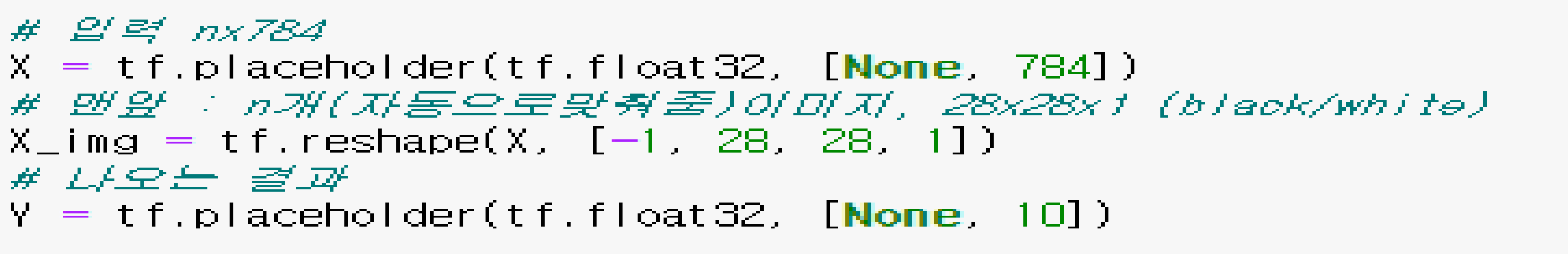
nx7x7x64

nx(7x7x64)

7x7x64,10(0~9 중 1개)

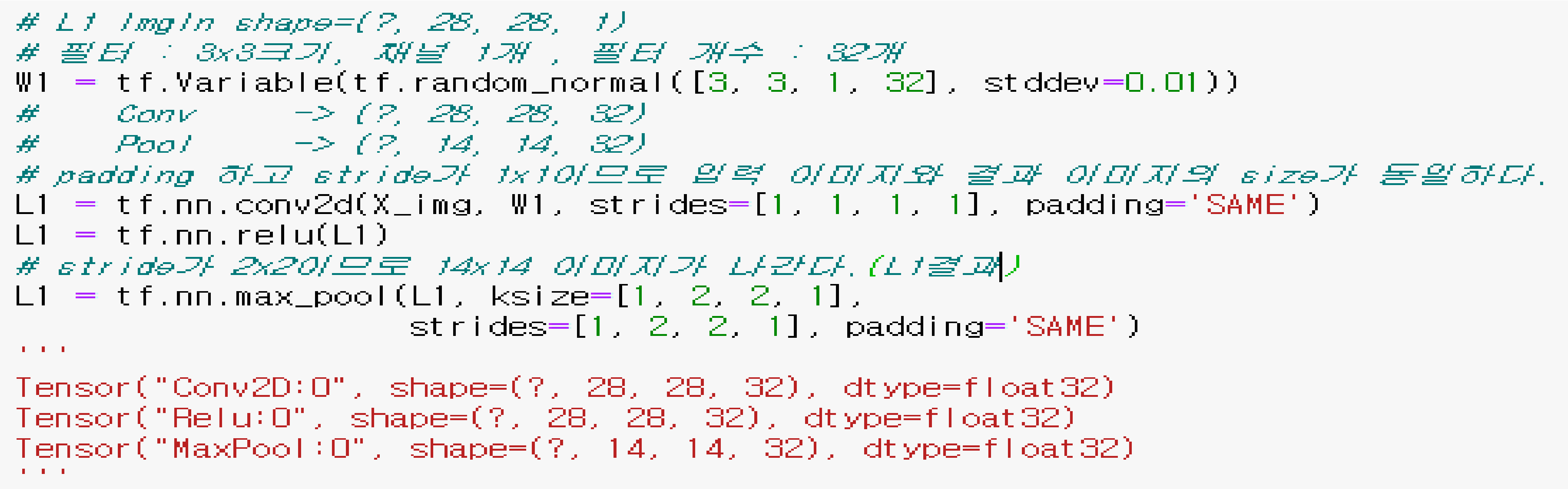
이미지개수, size, 필터 수

이미지개수, size, 필터 수



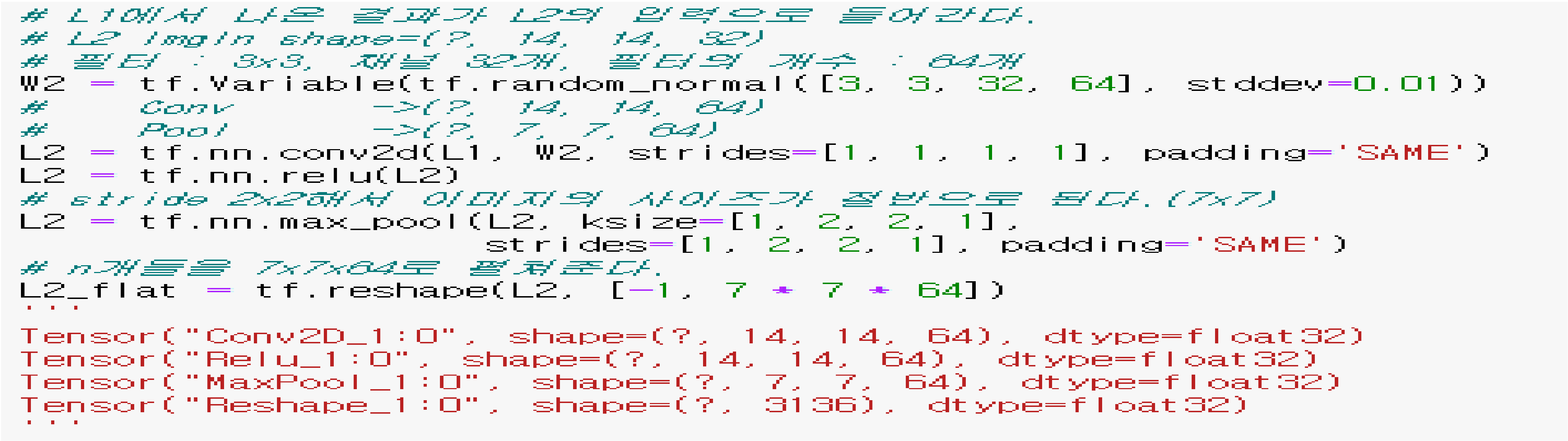
```python
# 입력 nx784
X = tf.placeholder(tf.float32, [None, 784])
# 맨앞 : n개(자동으로맞춰줌)이미지, 28x28x1 (black/white)
X_img = tf.reshape(X, [-1, 28, 28, 1])
# 나오는 결과
Y = tf.placeholder(tf.float32, [None, 10])
```

입력 이미지 포맷 : n개, size(28x28), channel(1,grayscale)



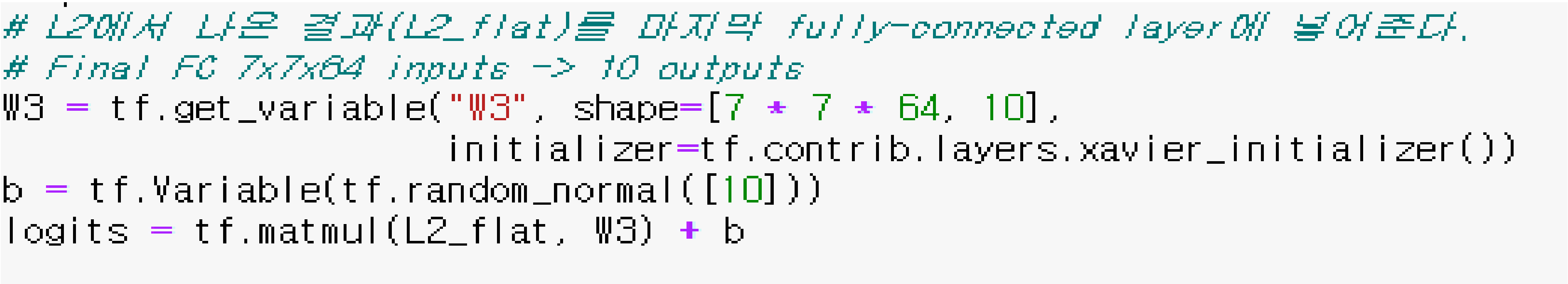
```python
# L1 ImgIn shape=(?, 28, 28, 1)
# 필터 : 3x3크기, 채널 1개 , 필터 개수 : 32개
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv     -> (?, 28, 28, 32)
#    Pool     -> (?, 14, 14, 32)
# padding 하고 stride가 1x1이므로 입력 이미지와 결과 이미지의 size가 동일하다.
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
# stride가 2x2이므로 14x14 이미지가 나간다.(L1결과)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''
```

L1 : nxsize(28x28)x32 → L1 : nxsize(14x14)x32



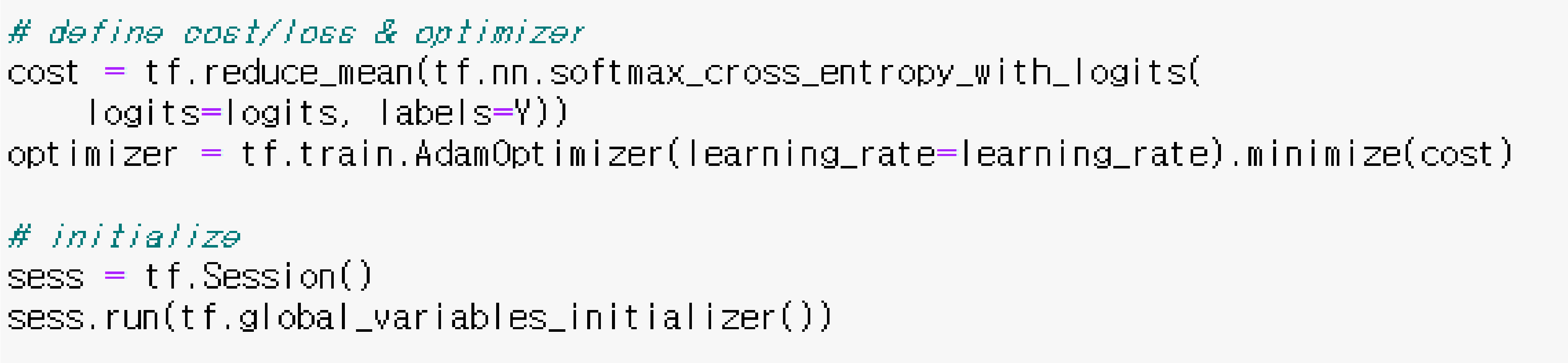
```
# L1에서 나온 결과가 L2의 입력으로 들어간다.
# L2 ImgIn shape=(?, 14, 14, 32)
# 필터 : 3x3, 채널 32개, 필터의 개수 : 64개
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
# stride 2x2해서 이미지의 사이즈가 절반으로 된다.(7x7)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
# n개들을 7x7x64로 펼쳐준다.
L2_flat = tf.reshape(L2, [-1, 7 * 7 * 64])
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
'''
```

L2 : nxsize(28x28)x64 → L2 : nxsize(14x14)x64



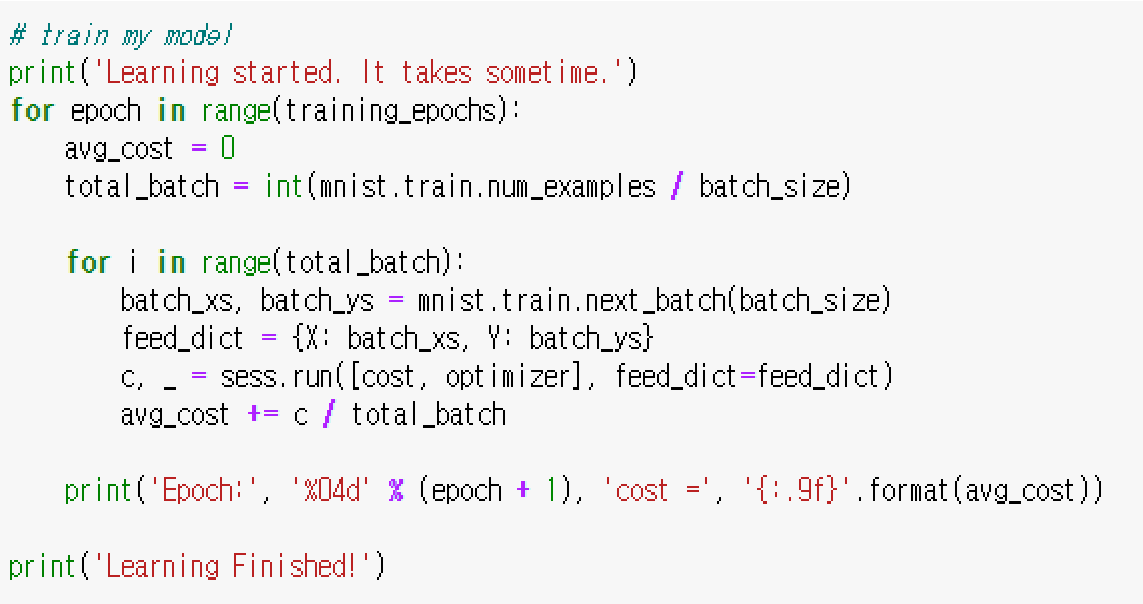
```
# L2에서 나온 결과(L2_flat)를 마지막 fully-connected layer에 넣어준다.
# Final FC 7x7x64 inputs -> 10 outputs
W3 = tf.get_variable("W3", shape=[7 * 7 * 64, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
logits = tf.matmul(L2_flat, W3) + b
```

L2_flat : nx(7*7*64) → F: (7*7*64)x10



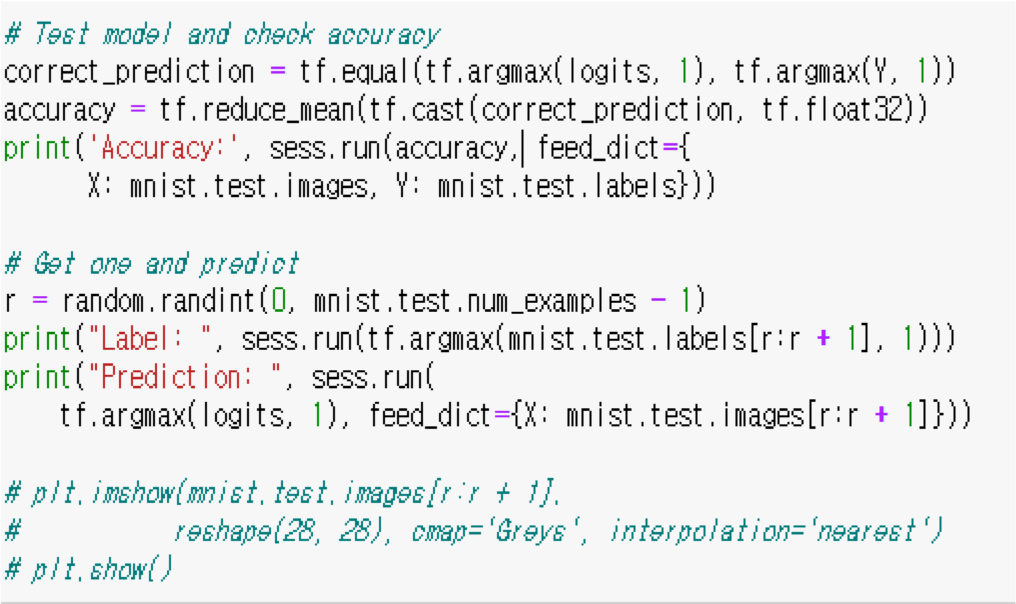
```python
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

Cost function 정의,
session 열기



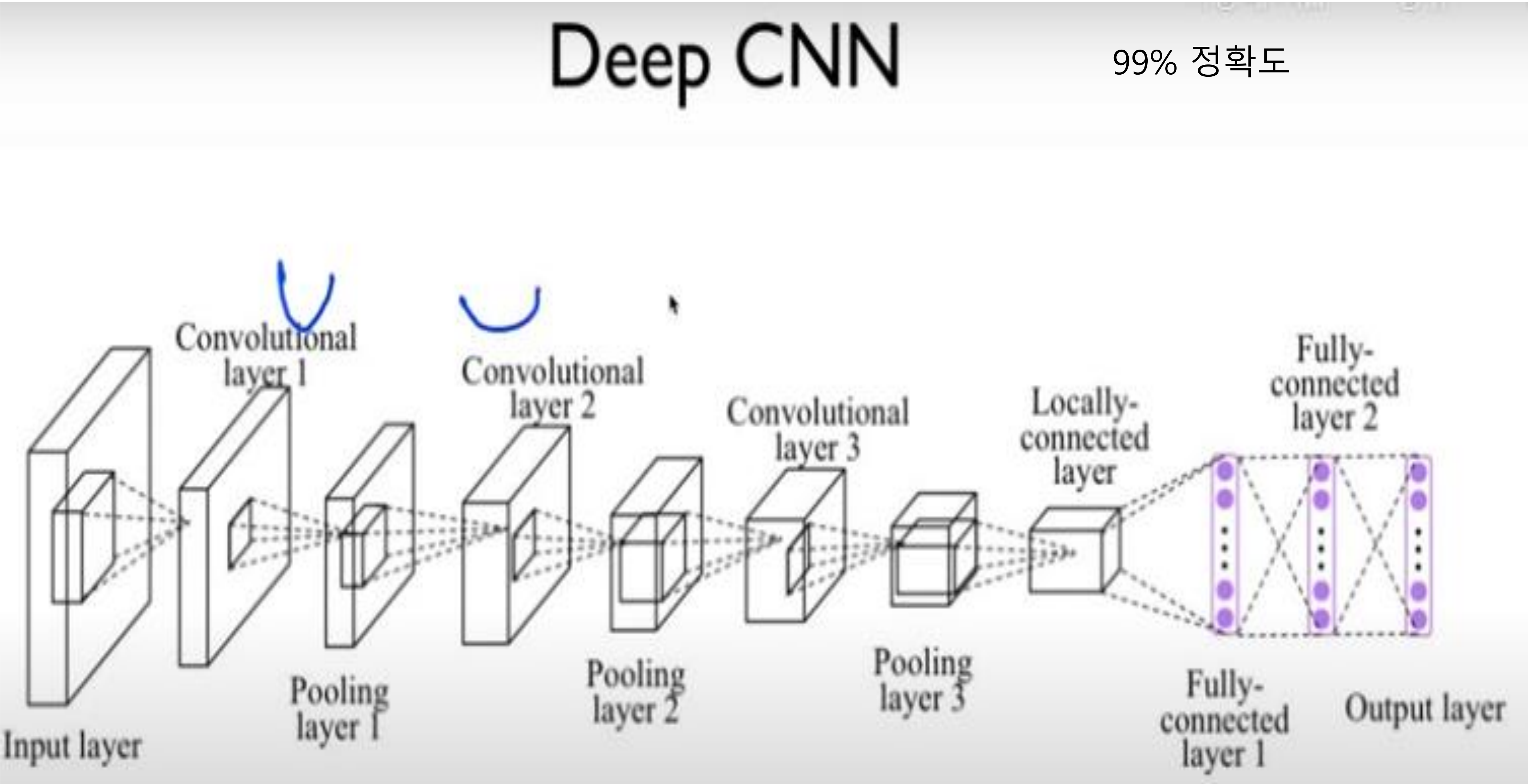
```python
# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')
```

모델 학습시키기



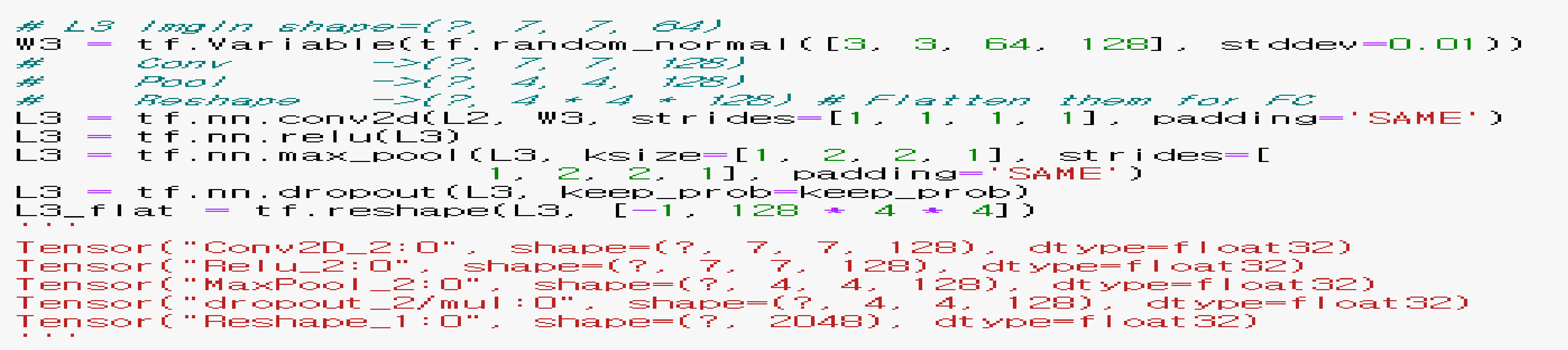
```python
# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

# plt.imshow(mnist.test.images[r:r + 1].
#           reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()
```

Test(정확도) 측정,
Prediction



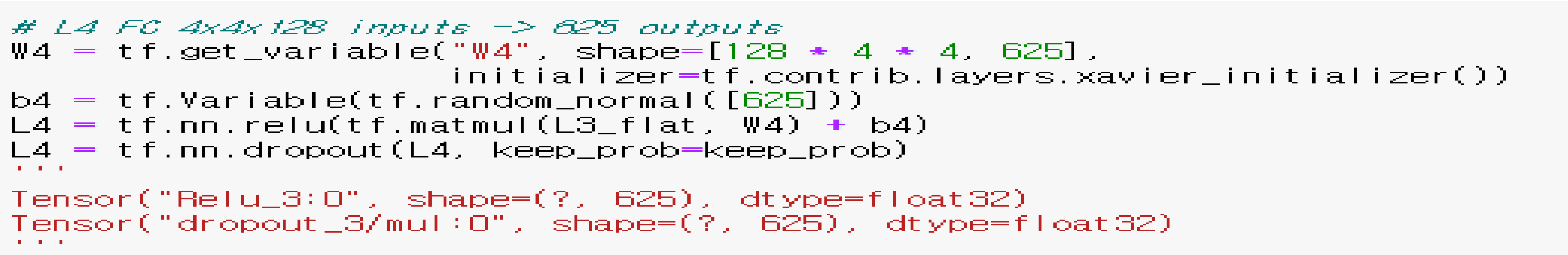
Deep CNN

99% 정확도



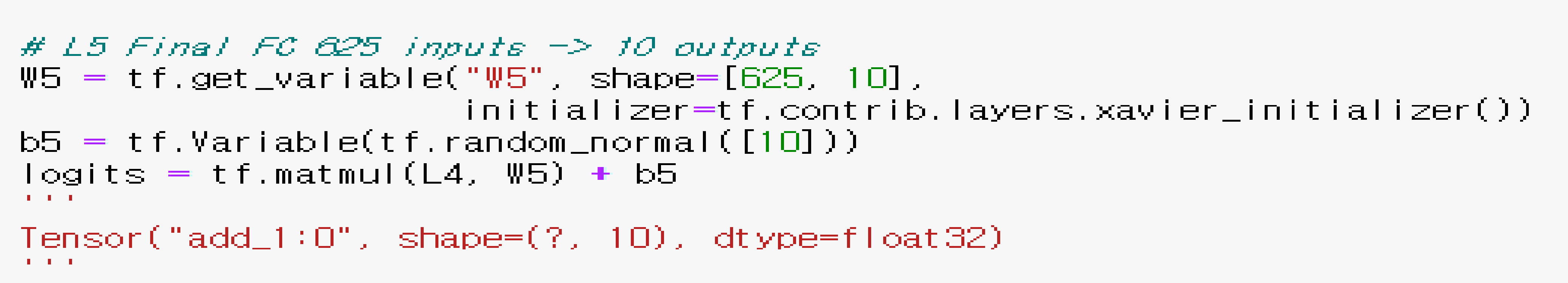
```
# L3 ImgIn shape=(?, 7, 7, 64)
W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
#    Conv      ->(?, 7, 7, 128)
#    Pool      ->(?, 4, 4, 128)
#    Reshape   ->(?, 4 * 4 * 128) # Flatten them for FC
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
L3 = tf.nn.relu(L3)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
                    1, 2, 2, 1], padding='SAME')
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
L3_flat = tf.reshape(L3, [-1, 128 * 4 * 4])
'''
Tensor("Conv2D_2:0", shape=(?, 7, 7, 128), dtype=float32)
Tensor("Relu_2:0", shape=(?, 7, 7, 128), dtype=float32)
Tensor("MaxPool_2:0", shape=(?, 4, 4, 128), dtype=float32)
Tensor("dropout_2/mul:0", shape=(?, 4, 4, 128), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 2048), dtype=float32)
'''
```

L3 : nxsize(7x7)x128 → L3 : nxsize(4x4)x128



```
# L4 FC 4x4x128 inputs -> 625 outputs
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([625]))
L4 = tf.nn.relu(tf.matmul(L3_flat, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
'''
Tensor("Relu_3:0", shape=(?, 625), dtype=float32)
Tensor("dropout_3/mul:0", shape=(?, 625), dtype=float32)
'''
```

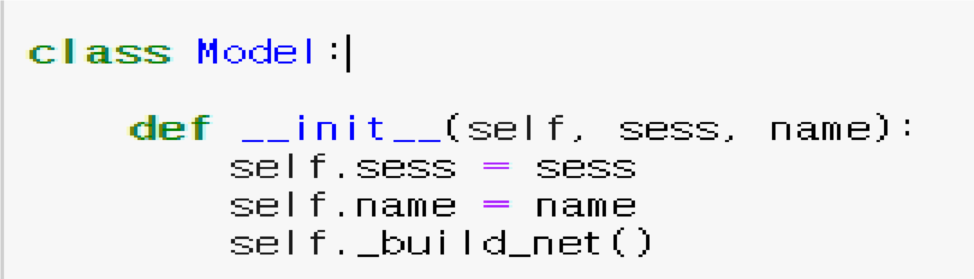
L3_flat : nx(4*4*128) → F1: (4*4*128)x625



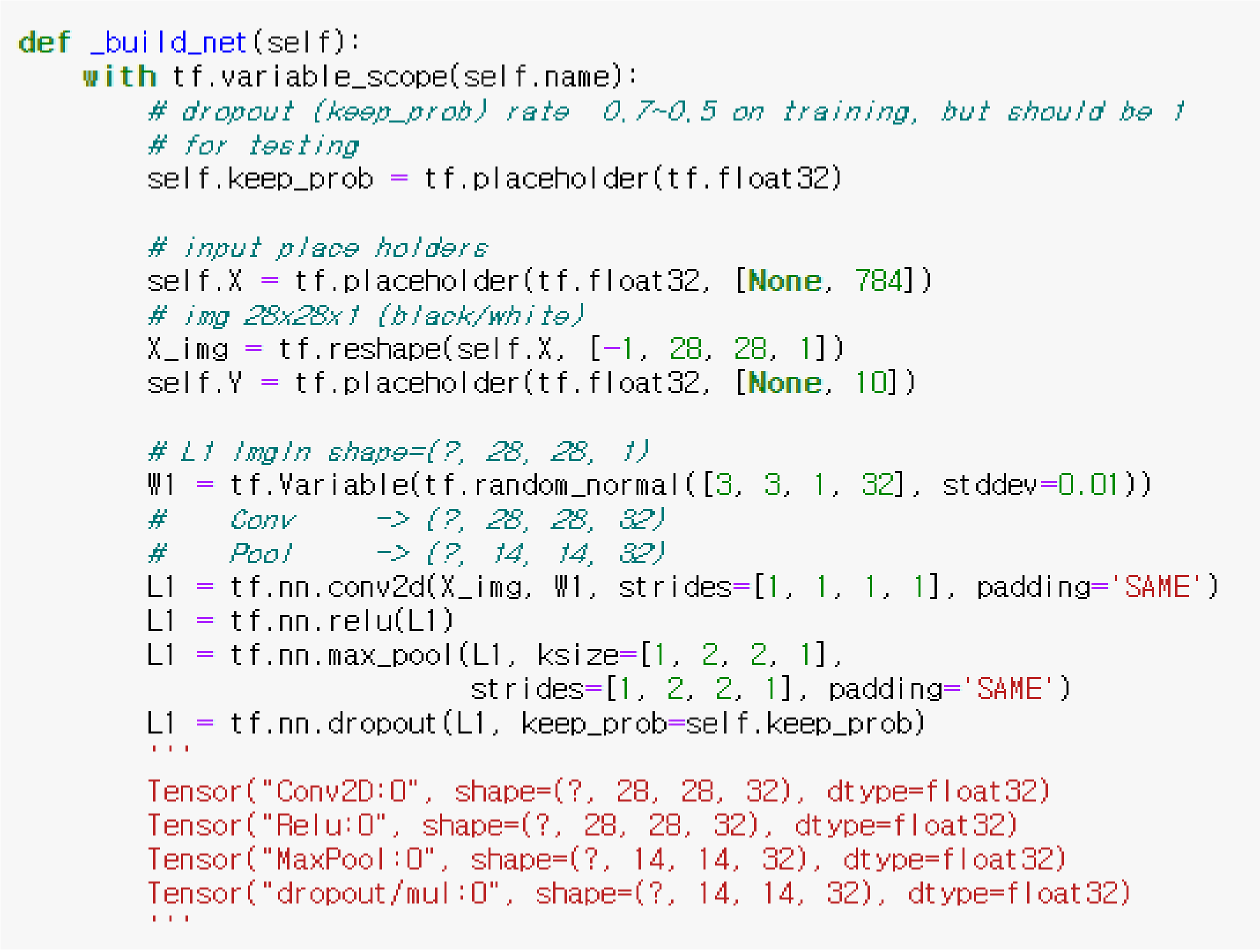
```
# L5 Final FC 625 inputs -> 10 outputs
W5 = tf.get_variable("W5", shape=[625, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
logits = tf.matmul(L4, W5) + b5
'''
Tensor("add_1:0", shape=(?, 10), dtype=float32)
'''
```

F1 : nx625 → F2: 625x10

CNN_class

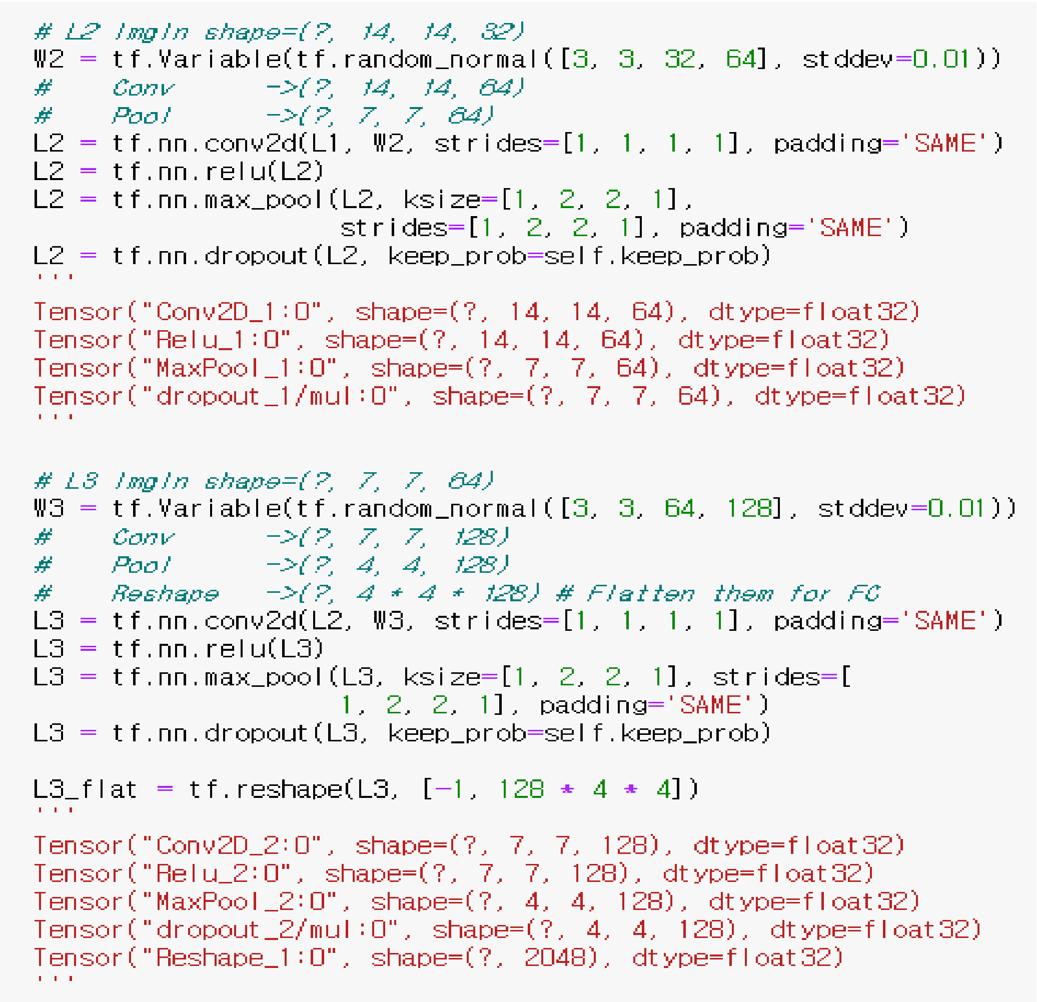


```
class Model:

    def __init__(self, sess, name):
        self.sess = sess
        self.name = name
        self._build_net()
```

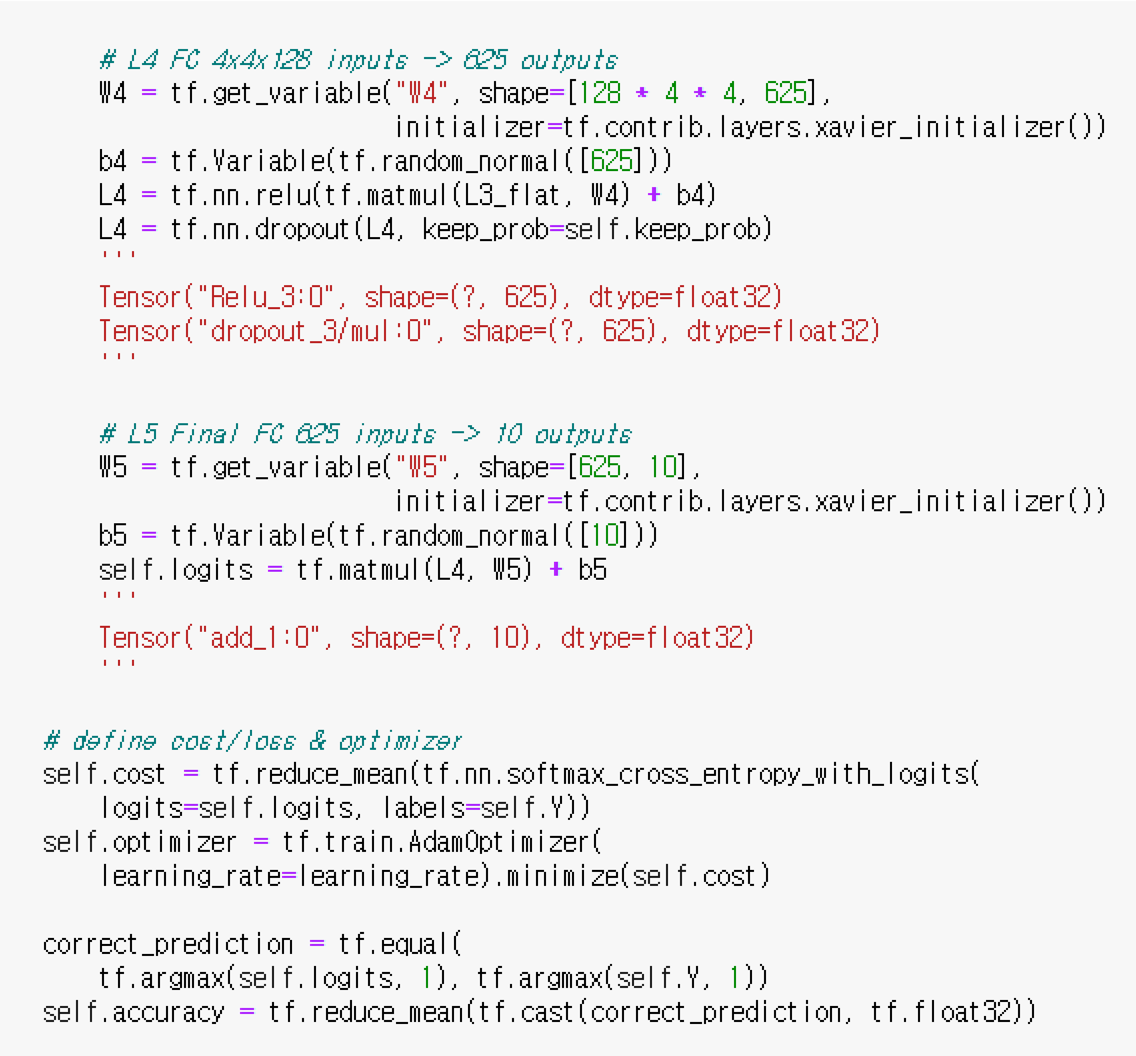


```
def _build_net(self):
    with tf.variable_scope(self.name):
        # dropout (keep_prob) rate  0.7~0.5 on training, but should be 1
        # for testing
        self.keep_prob = tf.placeholder(tf.float32)

        # input place holders
        self.X = tf.placeholder(tf.float32, [None, 784])
        # img 28x28x1 (black/white)
        X_img = tf.reshape(self.X, [-1, 28, 28, 1])
        self.Y = tf.placeholder(tf.float32, [None, 10])

        # L1 ImgIn shape=(?, 28, 28, 1)
        W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
        L1 = tf.nn.relu(L1)
        L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1], padding='SAME')
        L1 = tf.nn.dropout(L1, keep_prob=self.keep_prob)
        '''
        Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
        Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
        Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
        Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)
        '''
```

CNN_class

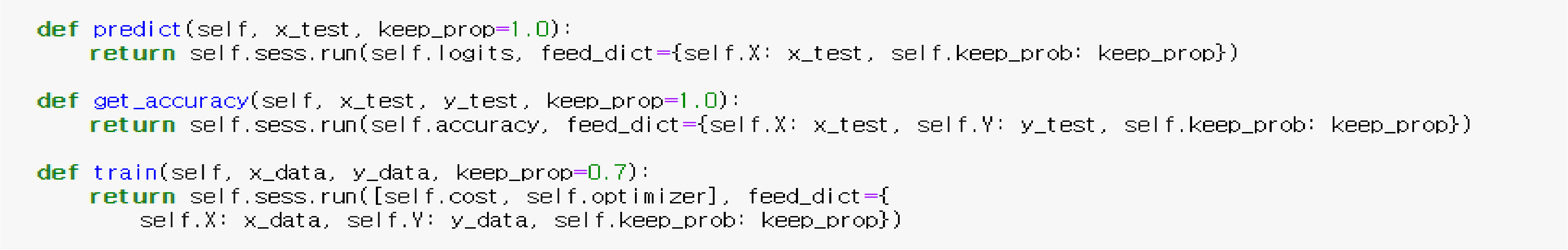


```
# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.nn.dropout(L2, keep_prob=self.keep_prob)
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("dropout_1/mul:0", shape=(?, 7, 7, 64), dtype=float32)
'''

# L3 ImgIn shape=(?, 7, 7, 64)
W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
#    Conv      ->(?, 7, 7, 128)
#    Pool      ->(?, 4, 4, 128)
#    Reshape   ->(?, 4 * 4 * 128) # Flatten them for FC
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
L3 = tf.nn.relu(L3)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
                    1, 2, 2, 1], padding='SAME')
L3 = tf.nn.dropout(L3, keep_prob=self.keep_prob)

L3_flat = tf.reshape(L3, [-1, 128 * 4 * 4])
'''
Tensor("Conv2D_2:0", shape=(?, 7, 7, 128), dtype=float32)
Tensor("Relu_2:0", shape=(?, 7, 7, 128), dtype=float32)
Tensor("MaxPool_2:0", shape=(?, 4, 4, 128), dtype=float32)
Tensor("dropout_2/mul:0", shape=(?, 4, 4, 128), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 2048), dtype=float32)
'''
```

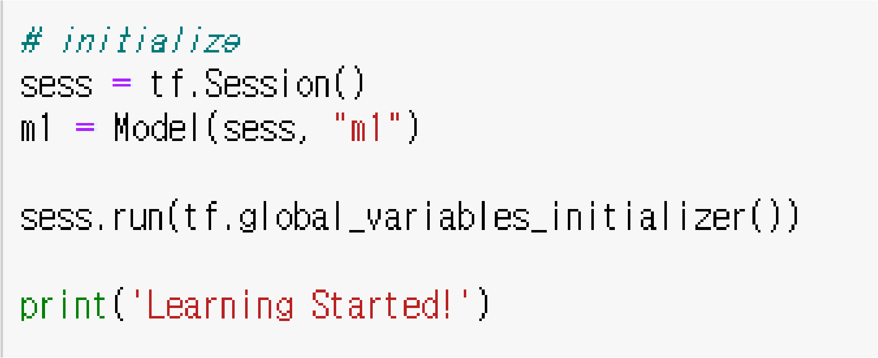


```
    # L4 FC 4x4x128 inputs -> 625 outputs
    W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
                         initializer=tf.contrib.layers.xavier_initializer())
    b4 = tf.Variable(tf.random_normal([625]))
    L4 = tf.nn.relu(tf.matmul(L3_flat, W4) + b4)
    L4 = tf.nn.dropout(L4, keep_prob=self.keep_prob)
    '''
    Tensor("Relu_3:0", shape=(?, 625), dtype=float32)
    Tensor("dropout_3/mul:0", shape=(?, 625), dtype=float32)
    '''

    # L5 Final FC 625 inputs -> 10 outputs
    W5 = tf.get_variable("W5", shape=[625, 10],
                         initializer=tf.contrib.layers.xavier_initializer())
    b5 = tf.Variable(tf.random_normal([10]))
    self.logits = tf.matmul(L4, W5) + b5
    '''
    Tensor("add_1:0", shape=(?, 10), dtype=float32)
    '''

# define cost/loss & optimizer
self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=self.logits, labels=self.Y))
self.optimizer = tf.train.AdamOptimizer(
    learning_rate=learning_rate).minimize(self.cost)

correct_prediction = tf.equal(
    tf.argmax(self.logits, 1), tf.argmax(self.Y, 1))
self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

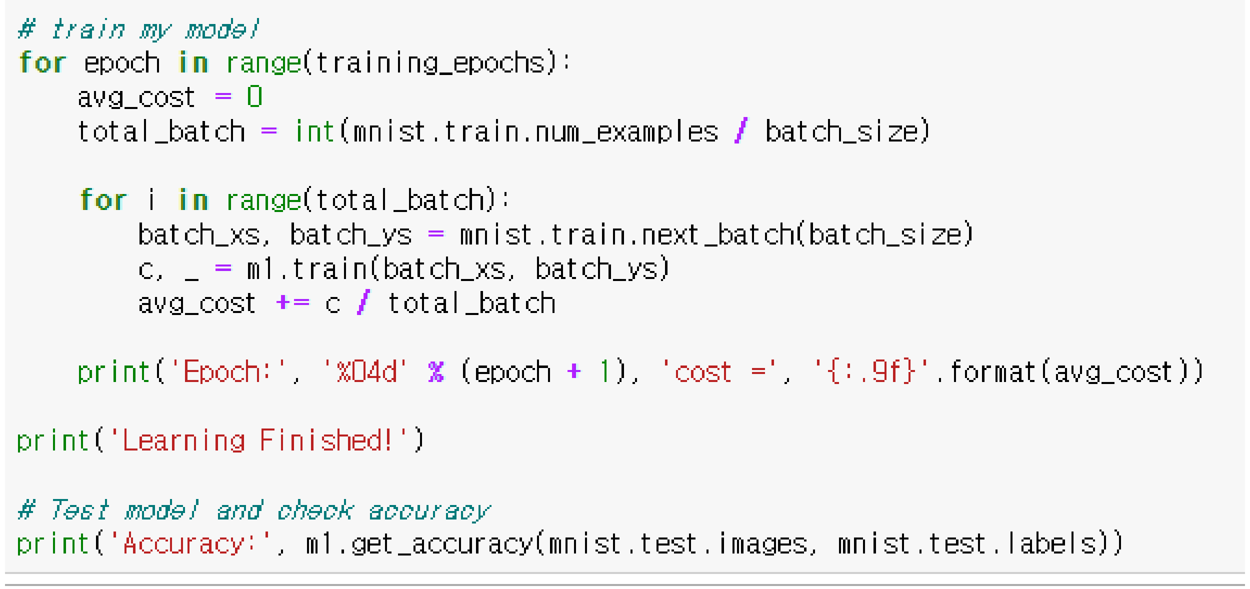
CNN_class



```
def predict(self, x_test, keep_prop=1.0):
    return self.sess.run(self.logits, feed_dict={self.X: x_test, self.keep_prob: keep_prop})

def get_accuracy(self, x_test, y_test, keep_prop=1.0):
    return self.sess.run(self.accuracy, feed_dict={self.X: x_test, self.Y: y_test, self.keep_prob: keep_prop})

def train(self, x_data, y_data, keep_prop=0.7):
    return self.sess.run([self.cost, self.optimizer], feed_dict={
        self.X: x_data, self.Y: y_data, self.keep_prob: keep_prop})
```

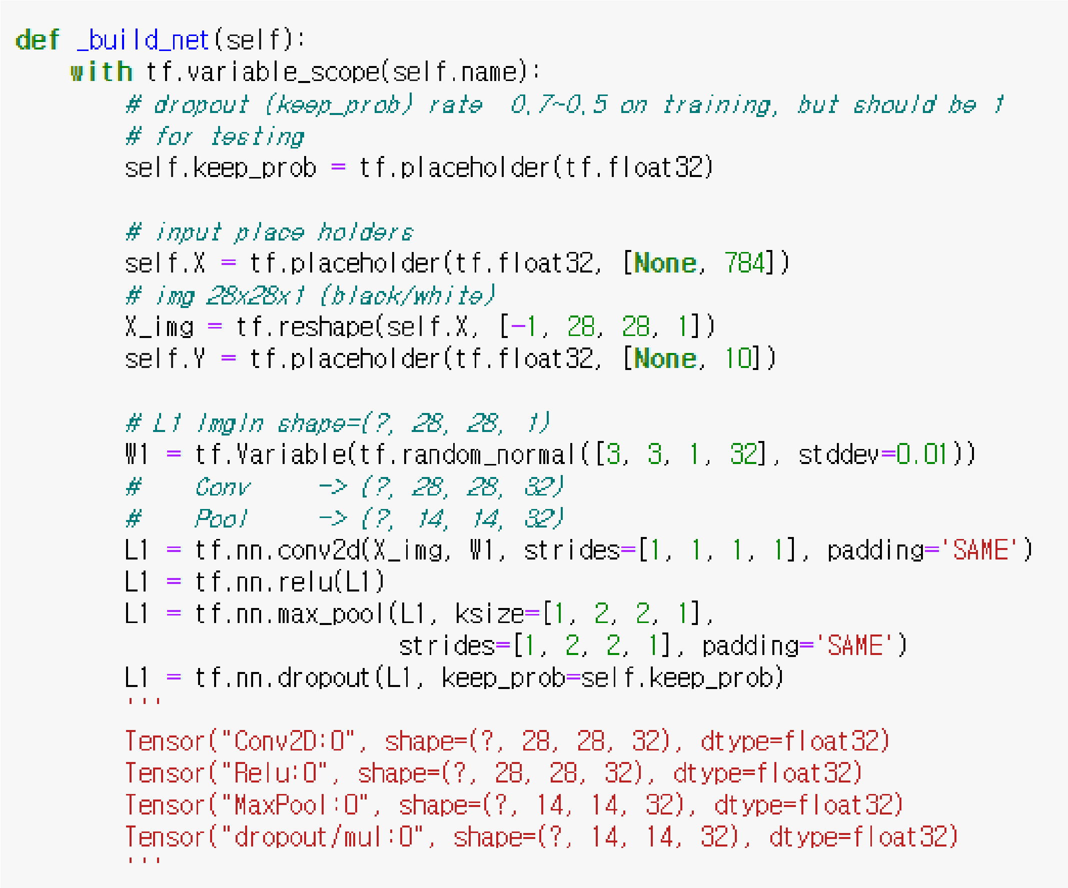


```
# initialize
sess = tf.Session()
m1 = Model(sess, "m1")

sess.run(tf.global_variables_initializer())

print('Learning Started!')
```

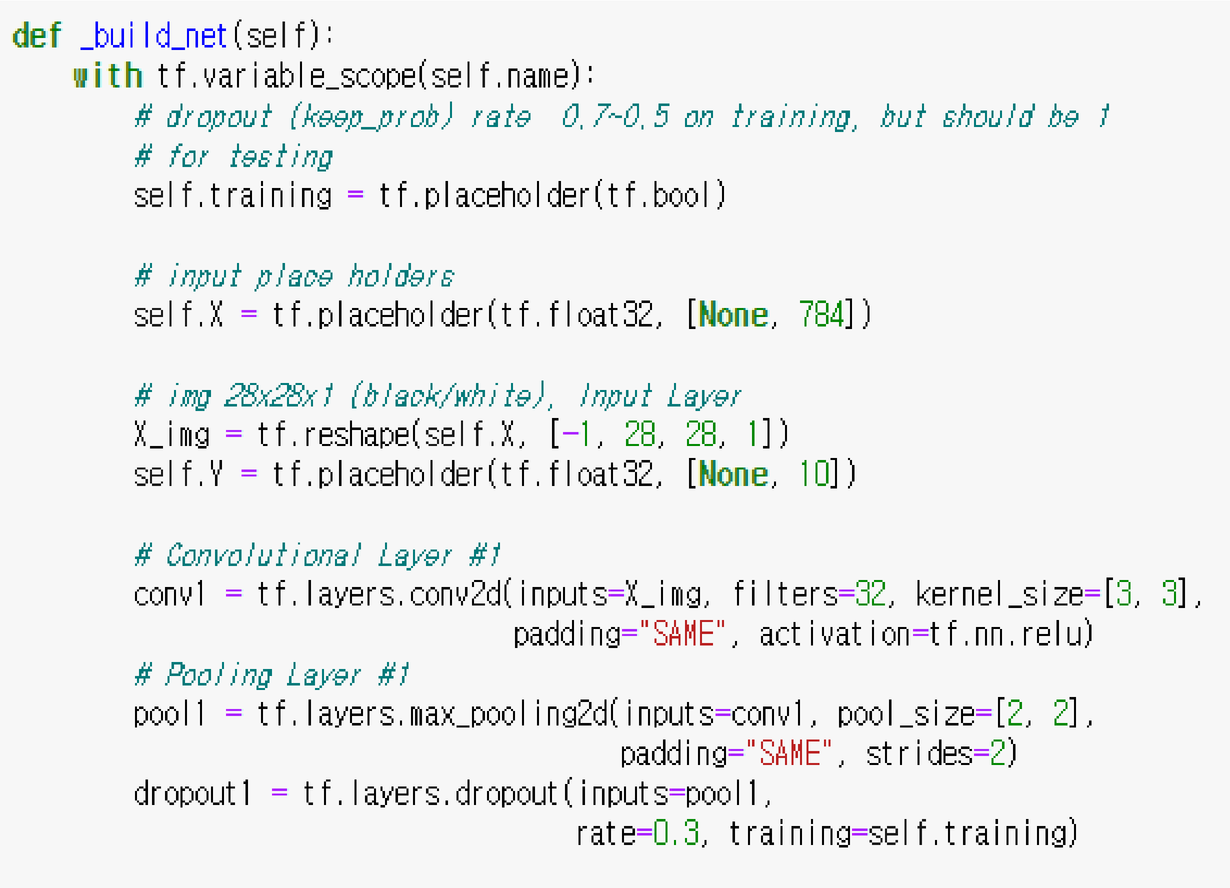


```
# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        c, _ = m1.train(batch_xs, batch_ys)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
print('Accuracy:', m1.get_accuracy(mnist.test.images, mnist.test.labels))
```

CNN_layers



```
def _build_net(self):
    with tf.variable_scope(self.name):
        # dropout (keep_prob) rate  0.7~0.5 on training, but should be 1
        # for testing
        self.keep_prob = tf.placeholder(tf.float32)

        # input place holders
        self.X = tf.placeholder(tf.float32, [None, 784])
        # img 28x28x1 (black/white)
        X_img = tf.reshape(self.X, [-1, 28, 28, 1])
        self.Y = tf.placeholder(tf.float32, [None, 10])

        # L1 ImgIn shape=(?, 28, 28, 1)
        W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
        L1 = tf.nn.relu(L1)
        L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1], padding='SAME')
        L1 = tf.nn.dropout(L1, keep_prob=self.keep_prob)
        '''
        Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
        Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
        Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
        Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)
        '''
```

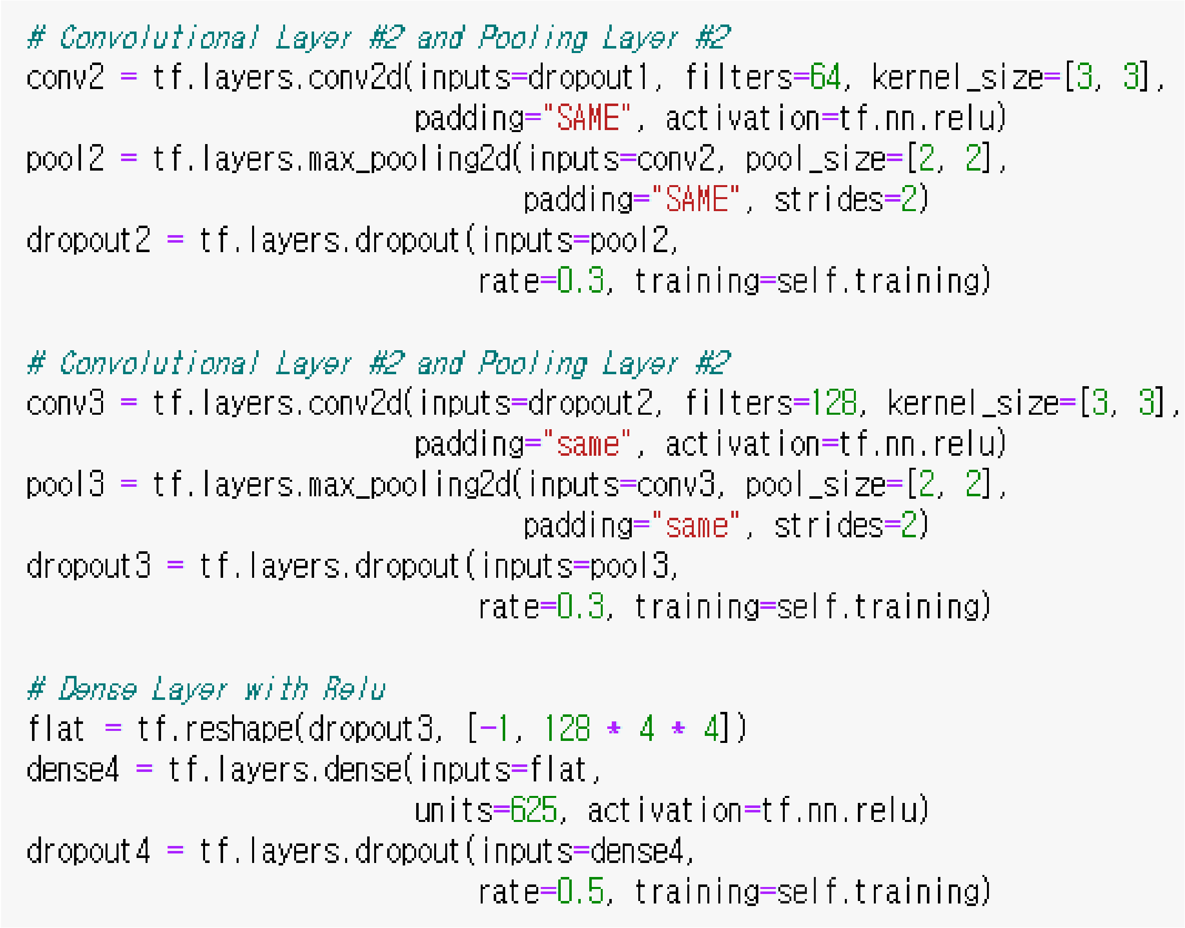


```
def _build_net(self):
    with tf.variable_scope(self.name):
        # dropout (keep_prob) rate  0.7~0.5 on training, but should be 1
        # for testing
        self.training = tf.placeholder(tf.bool)

        # input place holders
        self.X = tf.placeholder(tf.float32, [None, 784])

        # img 28x28x1 (black/white), Input Layer
        X_img = tf.reshape(self.X, [-1, 28, 28, 1])
        self.Y = tf.placeholder(tf.float32, [None, 10])

        # Convolutional Layer #1
        conv1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3, 3],
                                 padding="SAME", activation=tf.nn.relu)
        # Pooling Layer #1
        pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
                                        padding="SAME", strides=2)
        dropout1 = tf.layers.dropout(inputs=pool1,
                                     rate=0.3, training=self.training)
```

차이점 : conv2d를 쓰면 input을 조건(filter, kernel_size 등)에 적용시킨 결과를 도출한다.

Dropout : 학습인지, Test인지 (rate=1) 구별

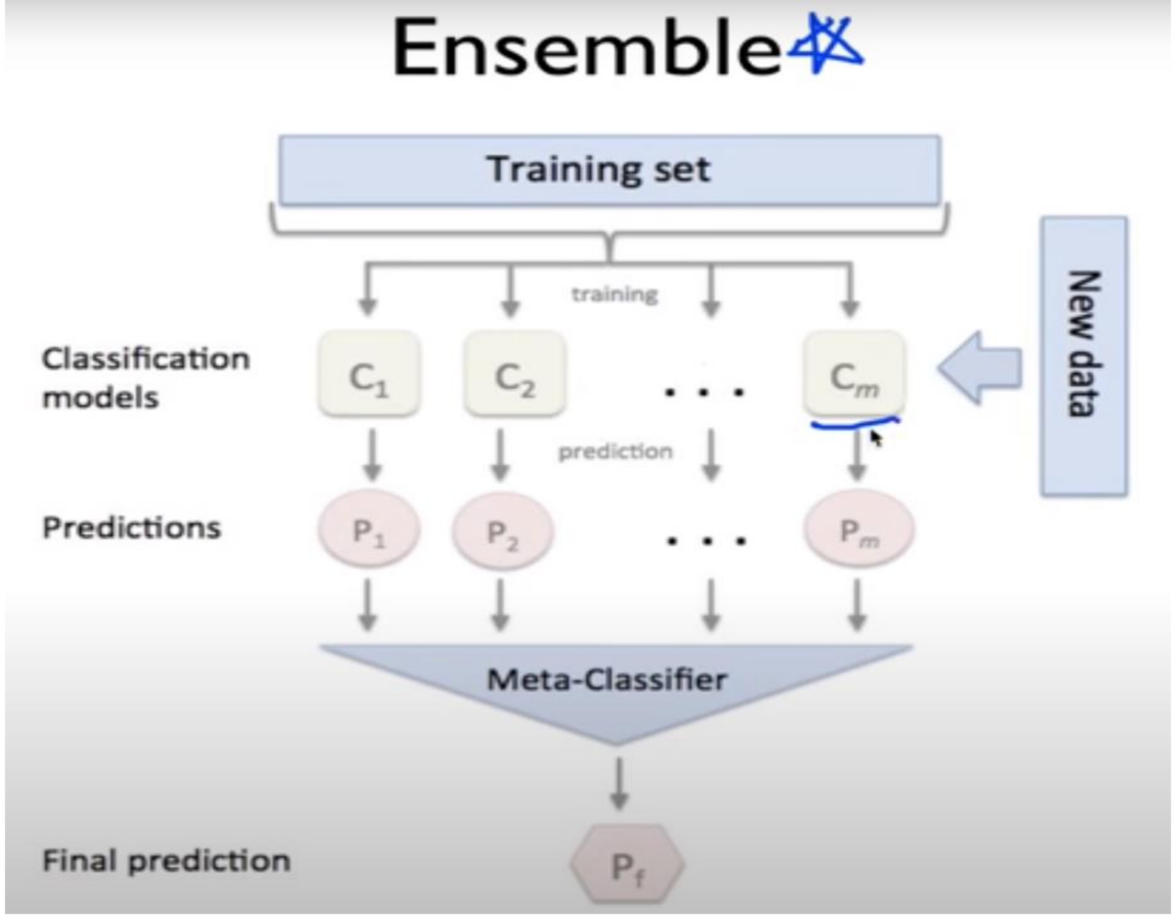
CNN_layers



```
# Convolutional Layer #2 and Pooling Layer #2
conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3],
                         padding="SAME", activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
                                padding="SAME", strides=2)
dropout2 = tf.layers.dropout(inputs=pool2,
                             rate=0.3, training=self.training)

# Convolutional Layer #2 and Pooling Layer #2
conv3 = tf.layers.conv2d(inputs=dropout2, filters=128, kernel_size=[3, 3],
                         padding="same", activation=tf.nn.relu)
pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2],
                                padding="same", strides=2)
dropout3 = tf.layers.dropout(inputs=pool3,
                             rate=0.3, training=self.training)

# Dense Layer with Relu
flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
dense4 = tf.layers.dense(inputs=flat,
                         units=625, activation=tf.nn.relu)
dropout4 = tf.layers.dropout(inputs=dense4,
                             rate=0.5, training=self.training)
```

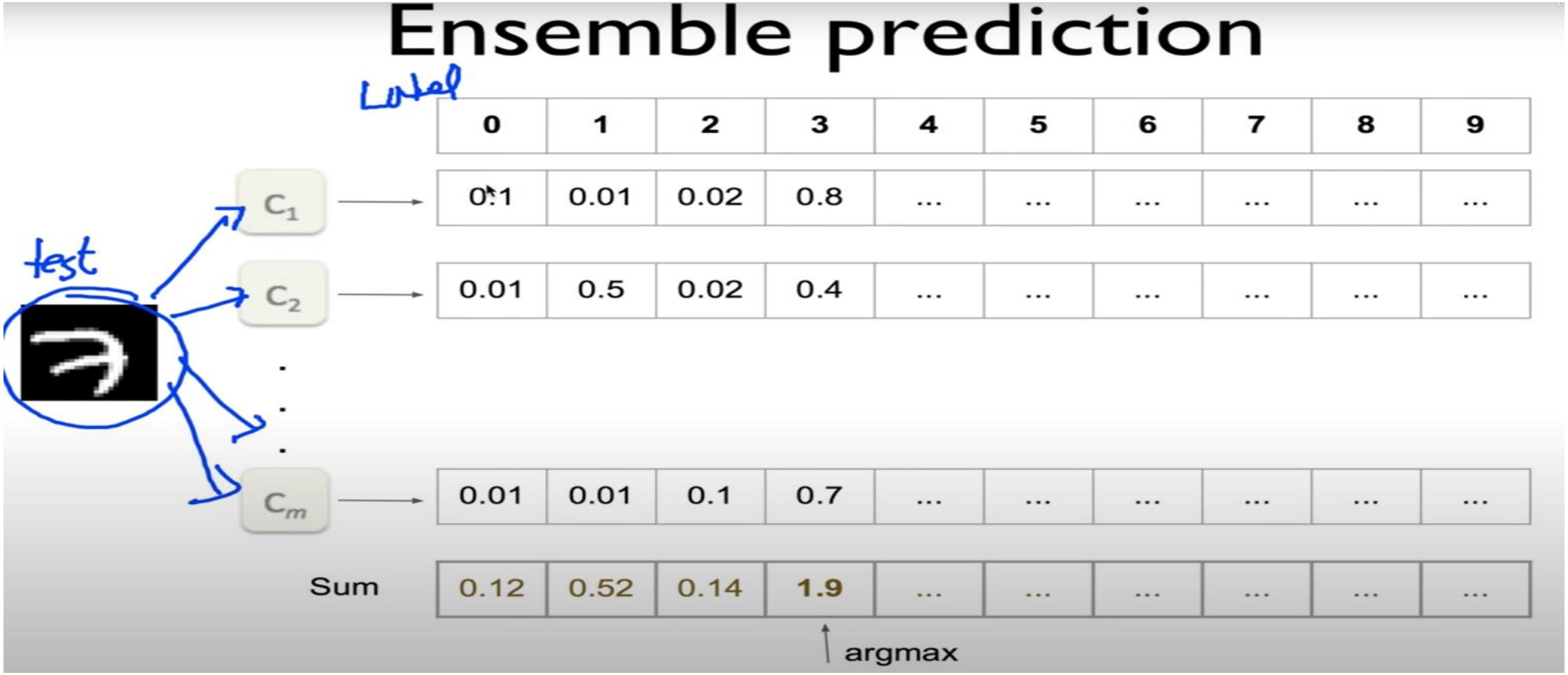
Dense: units(출력을 몇 개로 할 것인지), activation을 어떤 것으로 할 것인지만 결정해주면 자동으로 input에 대해 계산해줌 (Fully-connected-layer)

CNN_Ensemble



여러 개의 모델을 training 시키고
새로운 데이터(testing할 데이터)가 들어왔을 때
각 모델들을 예측 시키고
조합한 뒤 최종 결과를 낸다.

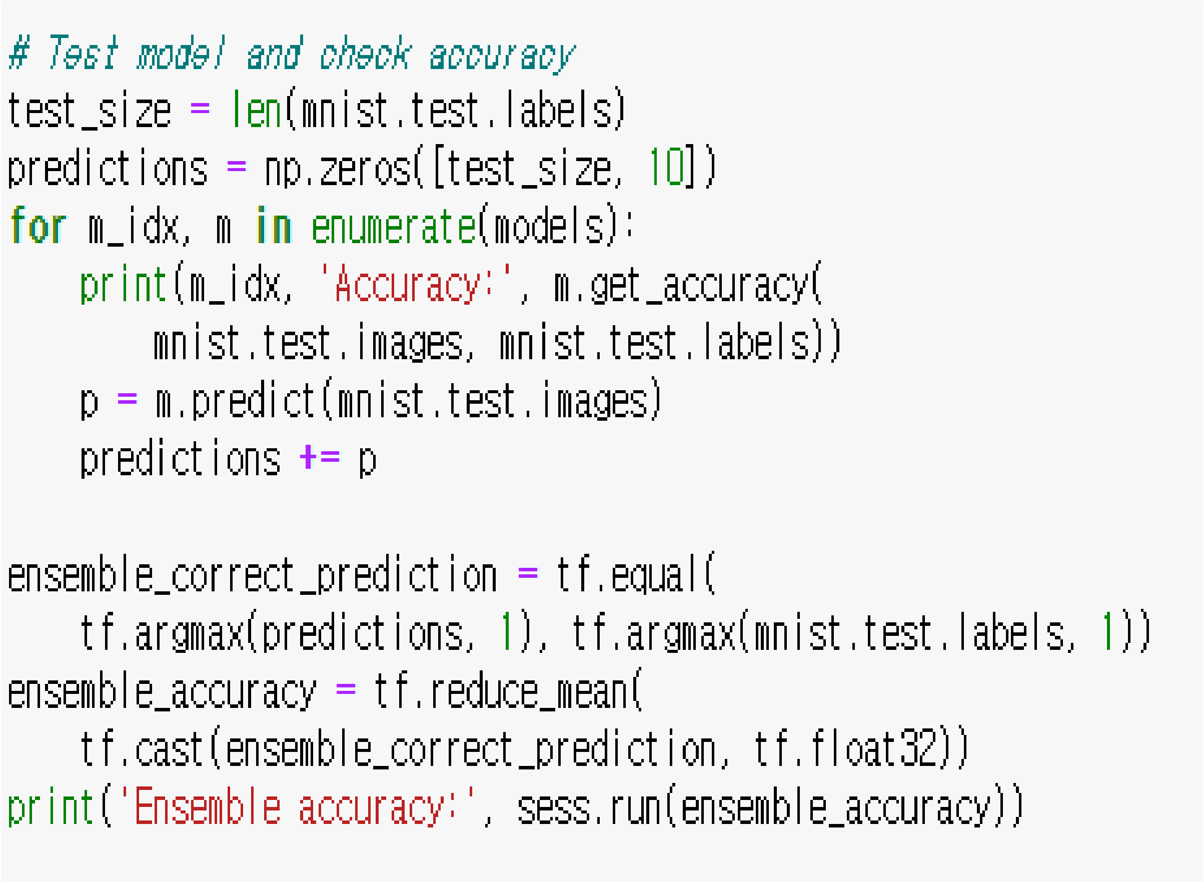
CNN_Ensemble



Ensemble prediction

	0	1	2	3	4	5	6	7	8	9
C_1	0.1	0.01	0.02	0.8	...	...	...	...	...	...
C_2	0.01	0.5	0.02	0.4	...	...	...	...	...	...
C_m	0.01	0.01	0.1	0.7	...	...	...	...	...	...
Sum	0.12	0.52	0.14	1.9	...	...	...	...	...	...

CNN_Ensemble



```
# Test model and check accuracy
test_size = len(mnist.test.labels)
predictions = np.zeros([test_size, 10])
for m_idx, m in enumerate(models):
    print(m_idx, 'Accuracy:', m.get_accuracy(
        mnist.test.images, mnist.test.labels))
    p = m.predict(mnist.test.images)
    predictions += p

ensemble_correct_prediction = tf.equal(
    tf.argmax(predictions, 1), tf.argmax(mnist.test.labels, 1))
ensemble_accuracy = tf.reduce_mean(
    tf.cast(ensemble_correct_prediction, tf.float32))
print('Ensemble accuracy:', sess.run(ensemble_accuracy))
```

여러 개의 모델을 training 시키고
새로운 데이터(testing할 데이터)가 들어왔을 때
각 모델들을 예측 시키고
조합한 뒤 최종 결과를 낸다.
⇒ 각 모델마다 label에 대해 예측한 값들을 다 더한다.
⇒ 최종적으로 다 더한 예측 값들 중 가장 큰(argmax) 값을 선택한다.