Chapter07.

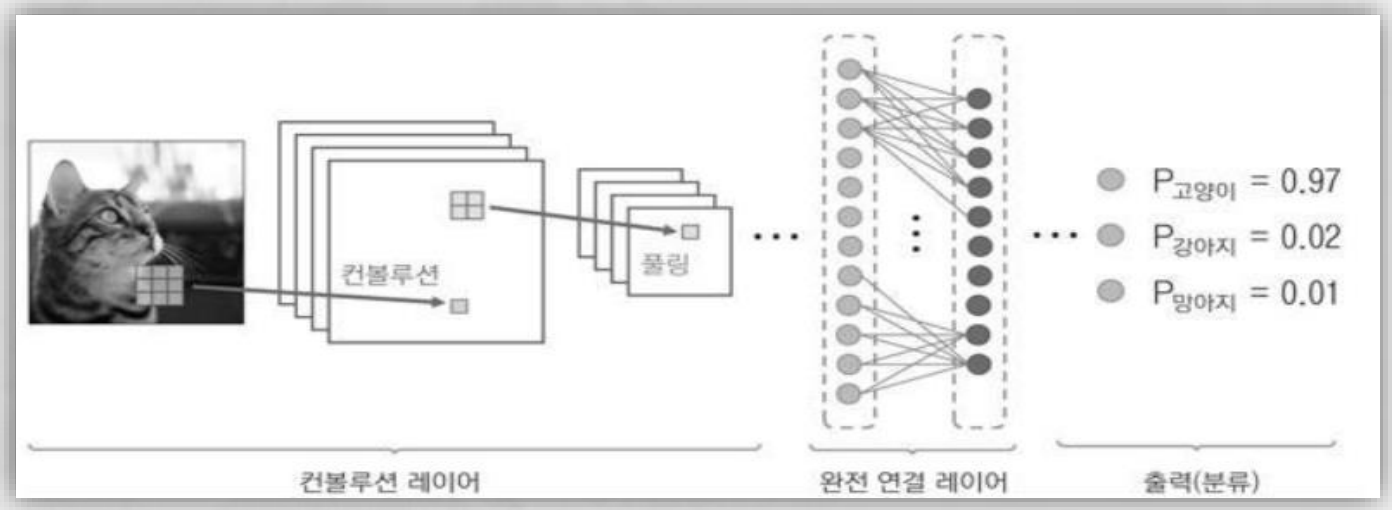
CNN

작성자 : 김진성

목차

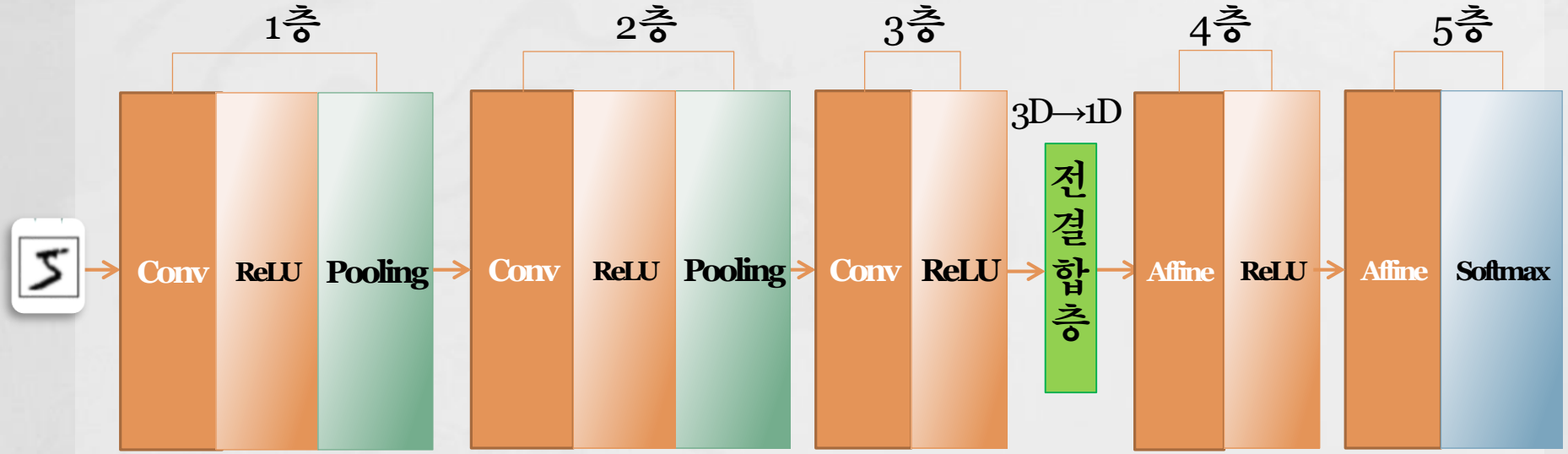
1. 합성곱(CNN) 개요

- 기존 영상 특징 추출 필터 기술 + DNN 기술 적용
- 영상, 음성 분야 모두에서 좋은 성능
- CNN(Convolution Neural Network) 구성
  - ✓ 컨볼루션(convolution) 레이어 : 2차원 영상에서 특징 추출 & 다운 샘플링(데이터 양 줄이고, 일부 특징 강조)
  - ✓ 완전 연결(FC, Fully Connected) 레이어 : 추출된 특징 분류기

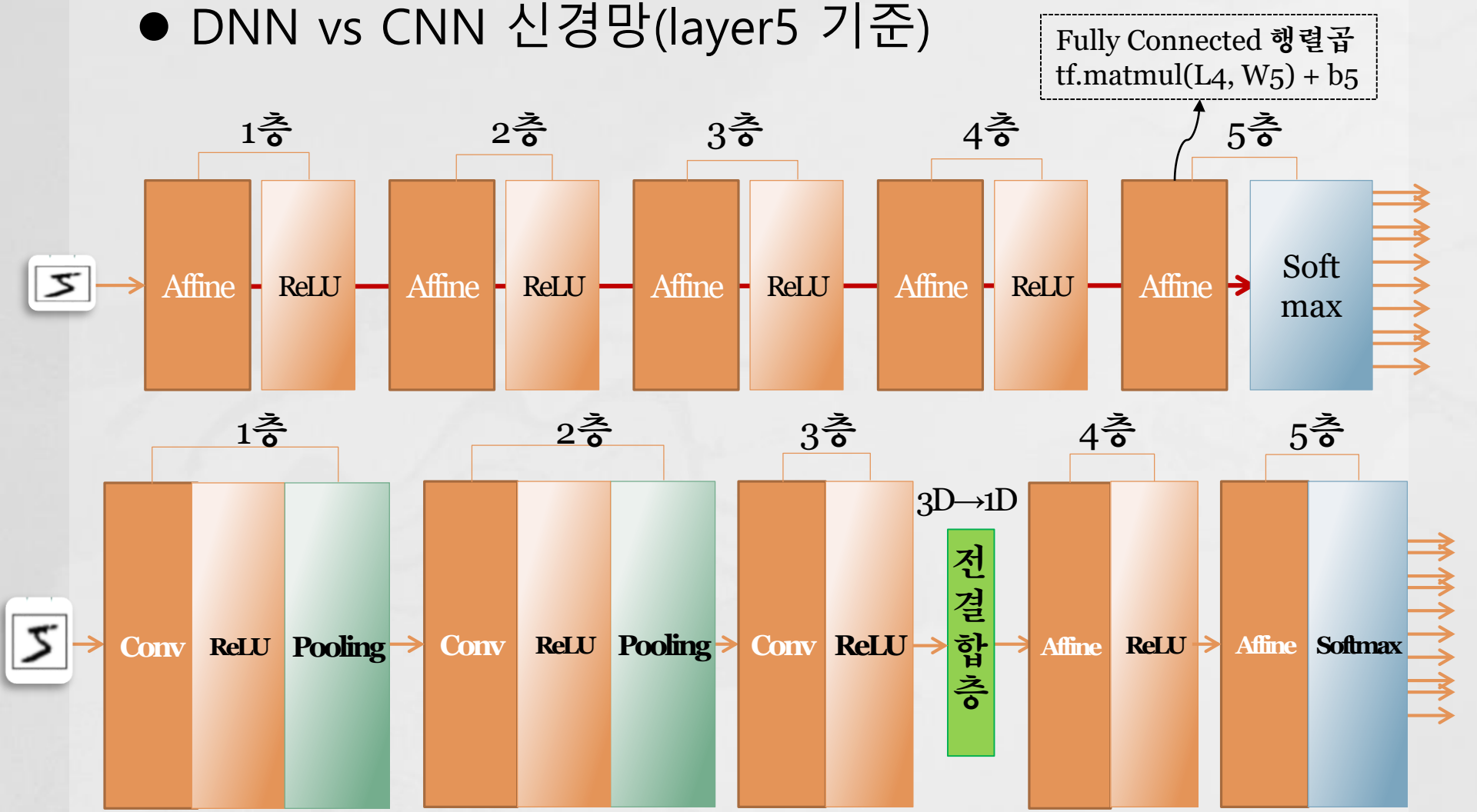


● CNN layer5 예

- ✓ 합성곱층(*Convolution* layer) = Conv+ReLU+Pooling 계층
- ✓ 전결합층(*Flatten layer*) = *Convolution* layer와 Affine layer 연결
- ✓ 완전연결층(*Fully Connected layer*) = Affine+Relu
- ✓ 출력층(*Output layer*) : Affine+Softmax or Sigmoid

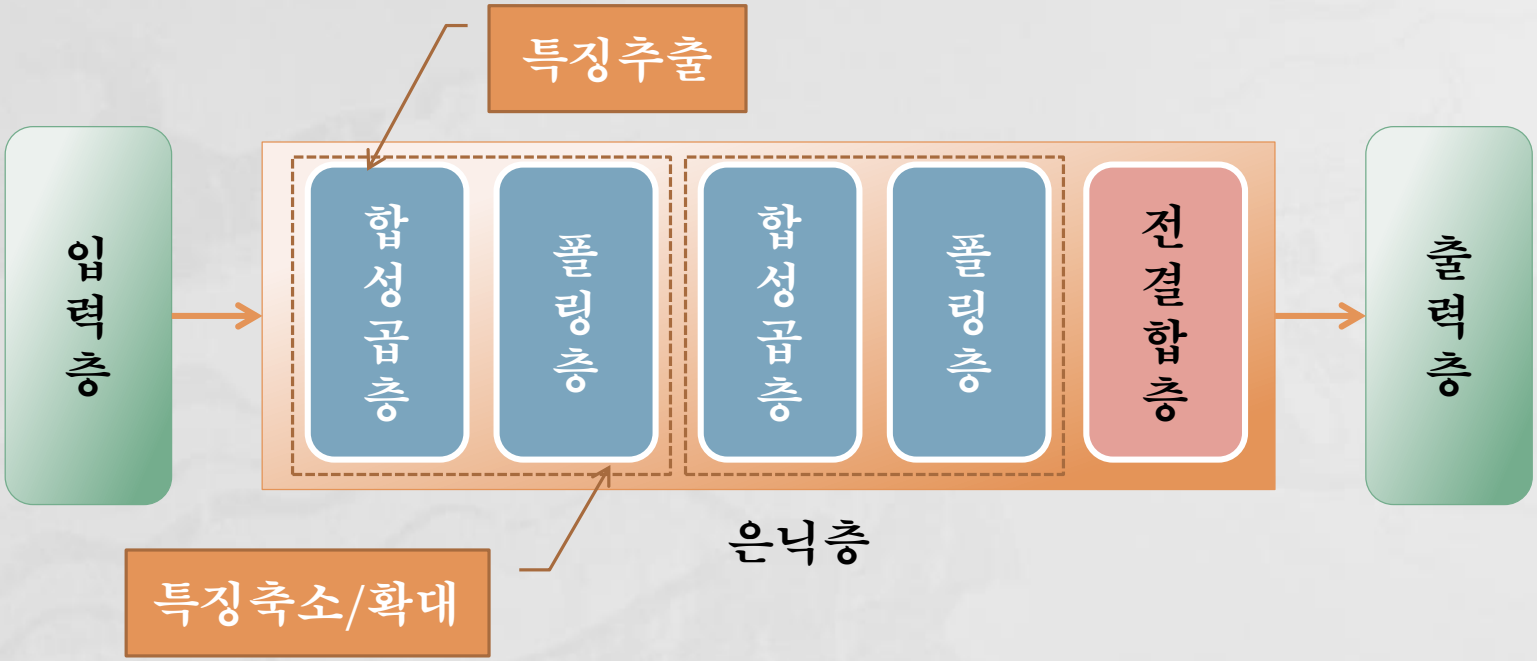


● DNN vs CNN 신경망(layer5 기준)



✓ Affine(fully connected)–ReLU 연결 → Conv-ReLU-Pooling 계층 변경
✓ 출력층과 가까운 층 : Affine-ReLU
✓ 마지막 출력층 : Affine-Softmax or Sigmoid

2. 합성곱 신경망(CNN) 구성



1) Convolution layer

- 합성곱 연산을 수행하는 계층
- 필터(hyper parameter)를 일정 간격으로 이동해가며 입력 이미지에 적용
- 입력과 필터에 대응하는 원소 끼리 곱한 후 총합 계산하여 합성곱 연산

예) 3*3 필터 형렬과 입력 image에 해당하는 원소들끼리 합성곱 연산
초록색 : 입력 이미지, 주황색 : 필터, 분홍색 : 합성곱 결과

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

입력 이미지
(Input Image)

*

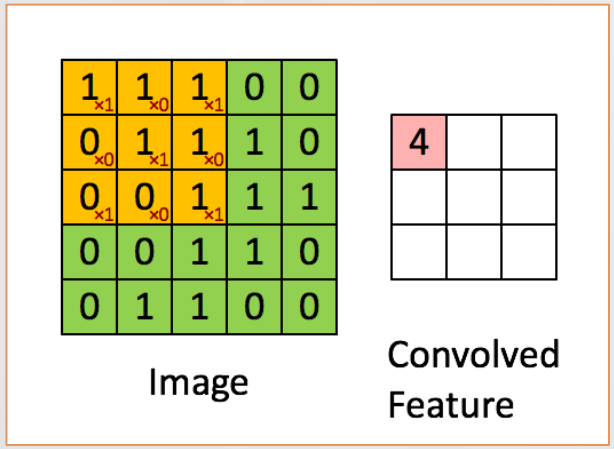
1	0	1
0	1	0
1	0	1

필터(Filter)

=

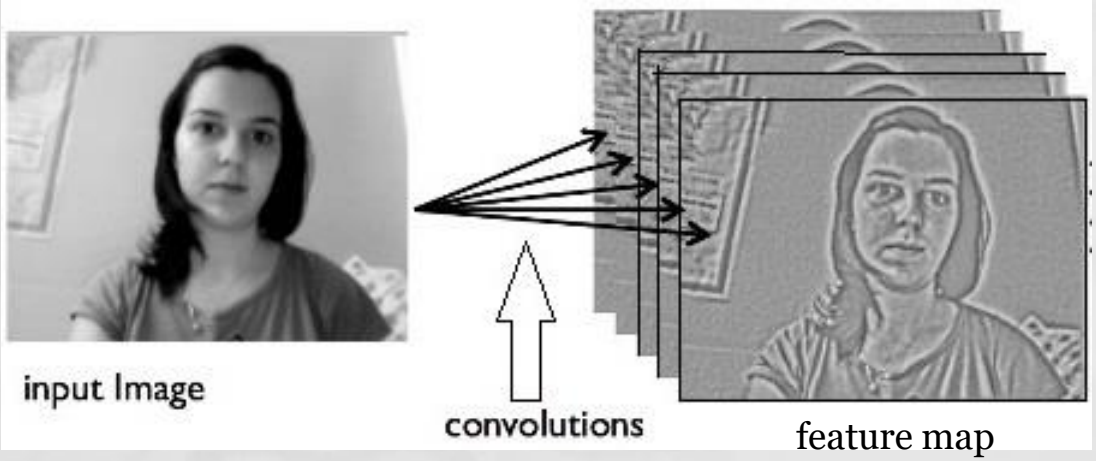
4	3	4
2	4	3
2	3	4

합성곱 결과
(Feature map)



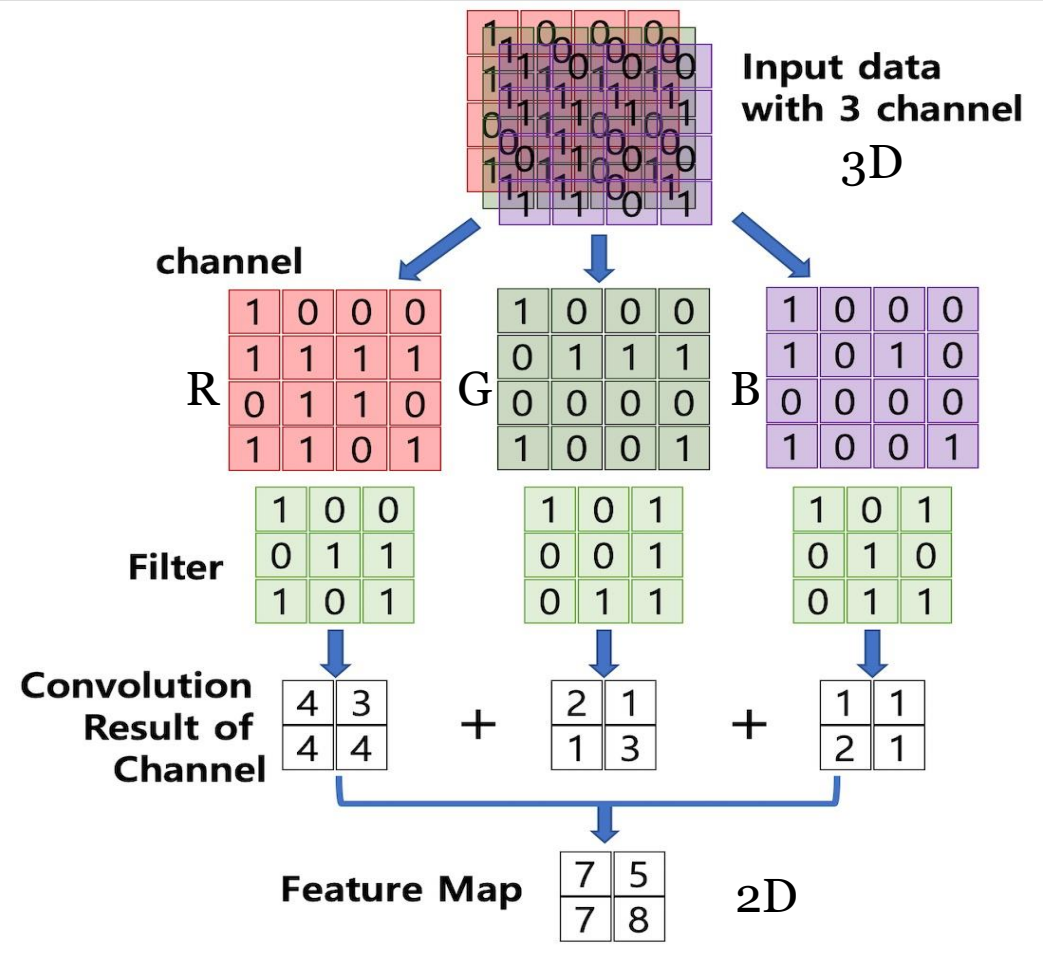
합성곱 animation

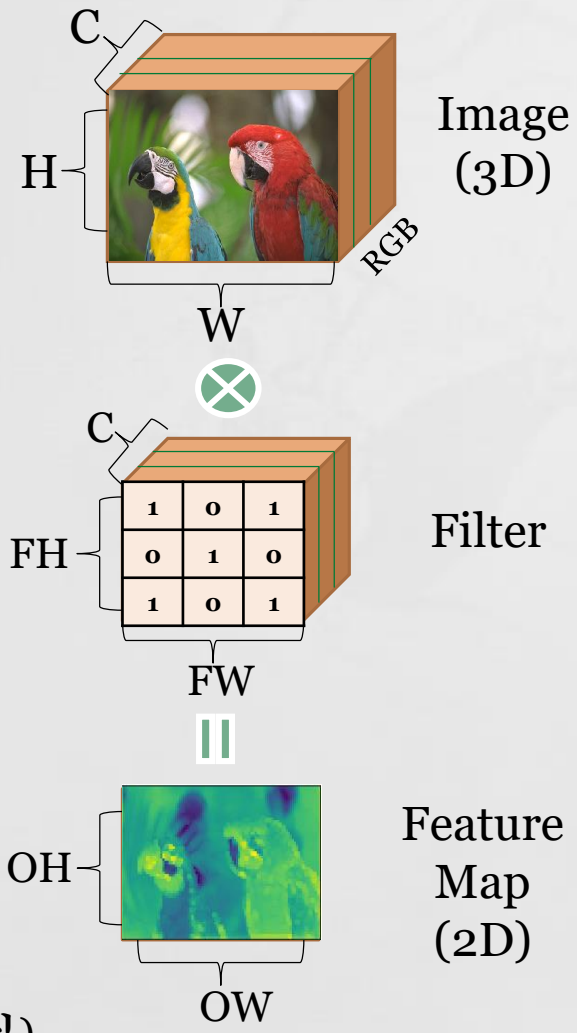
● 흑백 이미지 합성곱 연산(특징 맵) 예



❖ 합성곱 연산으로 이미지의 특징맵(5개) 추출 결과

● 칼럼(channel) 이미지 합성곱 연산





❖ 컬러 이미지 역시 Feature map에서 2D 변환(칼럼 정보 소실)

● *Feature map*

✓ Convolution에 의해서 생성된 이미지(이미지 특징)

✓ 입력 image(X), W : Filter(W)

✓Filter(W) 학습을 통해서 수정되는 대상

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

입력 Image(X)

*

1	0	1
0	1	0
1	0	1

Filter(W)

=

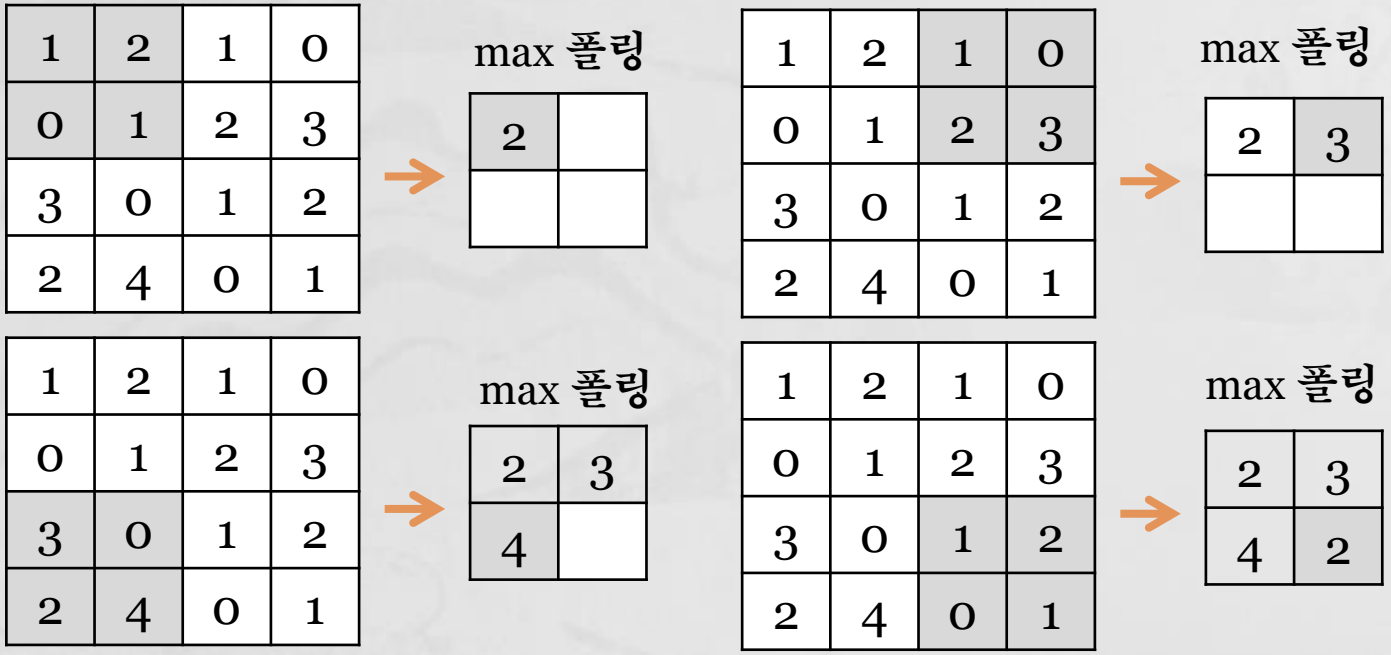
4	3	4
2	4	3
2	3	4

Output(Feature map)

❖ 특징맵은 Pooling을 통해서 한 번더 공간의 크기가 줄어든다.

2) Pooling layer

- ✓ Feature map 대상 가로, 세로 방향의 공간을 줄이는 연산(down sampling)
- ✓ 데이터 양을 줄이고, 일부 특징 강조
- ✓ 예) 2x2 영역을 하나의 원소로 공간 줄이기
  (폴링 윈도 : 2x2, 스트라이드 : 2)

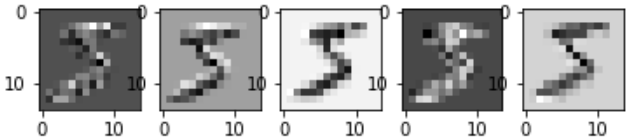


❖ max-pooling은 각 윈도우에서 가장 큰 값 선택(가장 많이 사용)

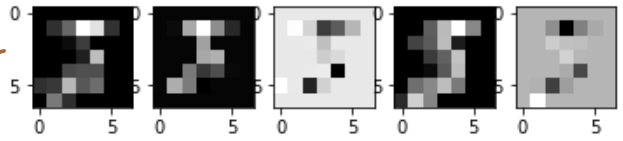
max 폴링예)

2x2 영역으로 공간의 크기를 줄여도 어미 사자와 아기 사자의 얼굴에서 주요 특징(눈, 코, 입)은 모두 추출 가능

합성곱 층 :특징맵(5,14,14,1)



폴링 층 : max polling(5,7,7,1)



다운 샘플링= 특징 강조

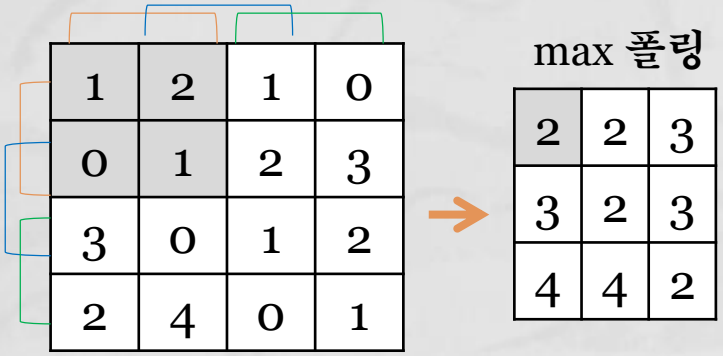
3) Stride

폴링 과정에서 가로, 세로 방향으로 이동하는 크기

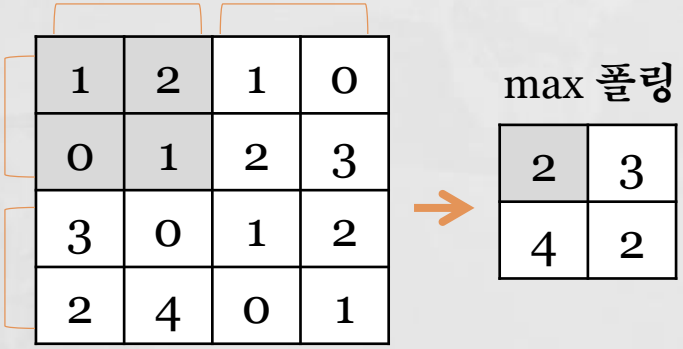
stride=2 일 때 기존 image pixel수를 전반으로 줄임

연산량을 줄이기 위해서 <u>입력단에서 가까운 쪽에서만 사용</u>

stride=1 일 때



stride=2 일 때



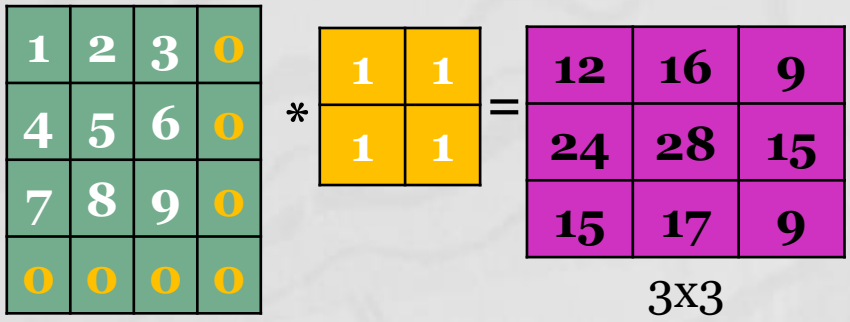
4) padding

합성곱 연산에 의해서 출력 이미지가 입력 이미지 보다 작아진다.

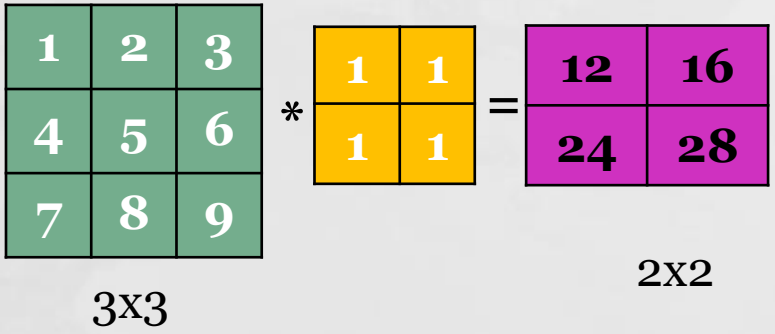
zero-padding : 합성곱 연산으로 가장 자리 정보 삭제 방지

padding = 'SAME' : 입력 크기와 출력 크기가 같다는 속성

stride=1, padding= *'SAME'* 일 때

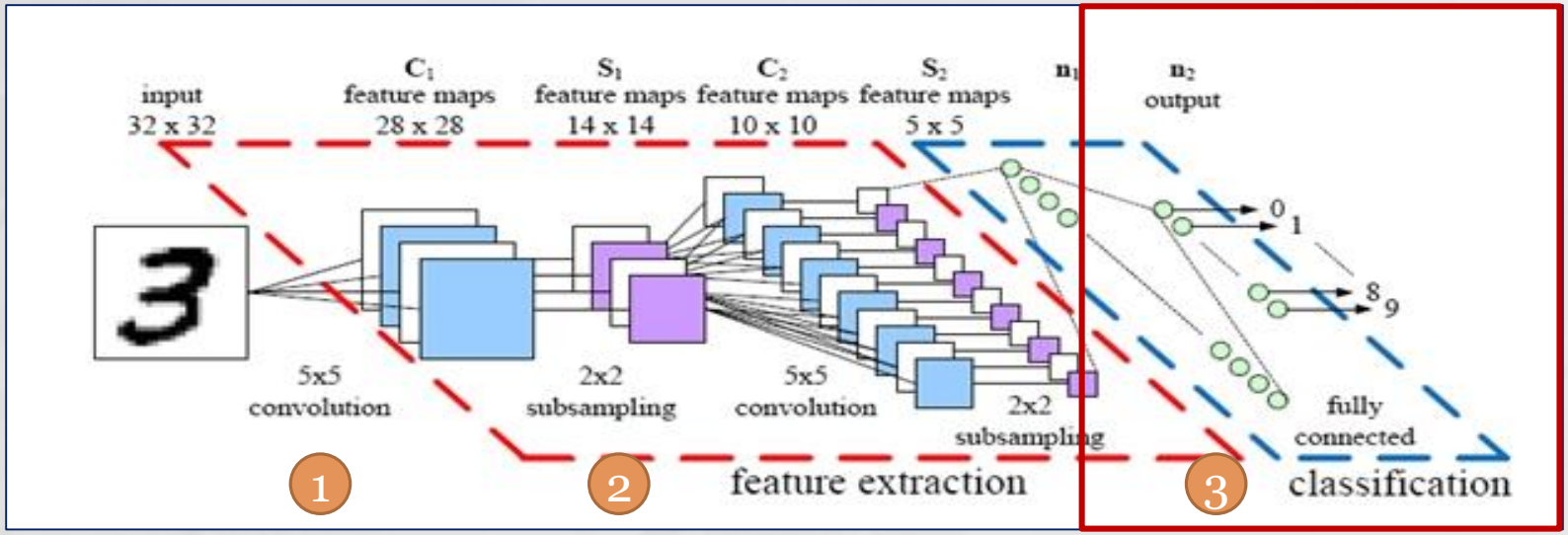


stride=1, padding=*'VALID' 일 때*



5) 전결합층(fully connected layer)

합성곱층의 이미지 차원을 폴링층에서 처리할 수 있도록 차원을 맞추는 역할



흑백 : 2D(?, 28, 28) ⇒ 1D(?, 784)

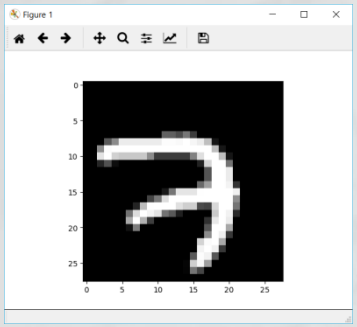
컬러 : 3D(?, 28, 28, 3) ⇒ 1D(?, 2352)

step01_mnist_cnn_basic.py(mnist data)

- **image 합성곱 함수**

tf.nn.conv2d(image, Filter, strides=[1,2,2,1], padding='*SAME*')

Image



(1,28,28,1)

*

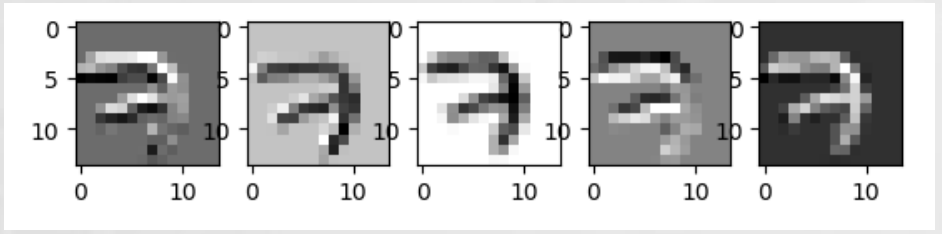
Filter

0.1	0.2	0.2
0.3	0.1	0.4
0.2	0.3	0.1

(3, 3, 1, 5)

=

Feature map

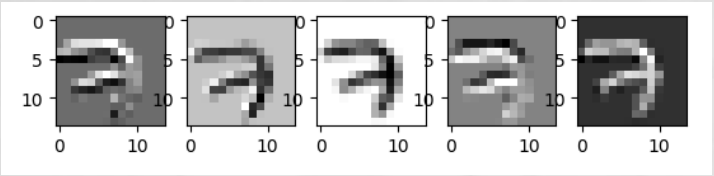


(1, 14, 14, 5)

- **image sub sampling 함수**

tf.nn.max_pool(conv2d, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

Feature map



(1, 14, 14, 5)

*

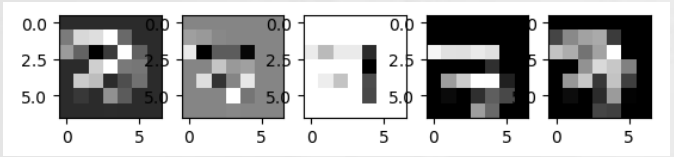
Filter

0.1	0.2
0.3	0.1

(1,2,2,1)

=

pool image

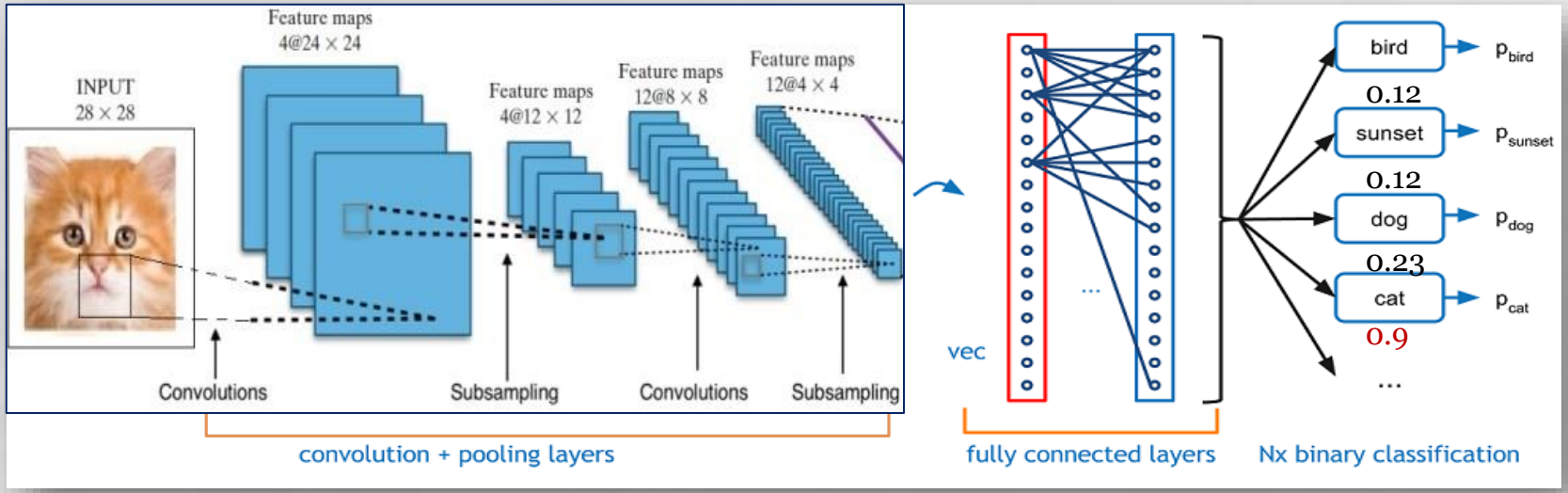


(1,7,7,1)

Stride 속성과 서브샘플링 관계

Stride = 1,1 → 입력 size와 동일
Stride = 2,2 → 입력 size의 절반

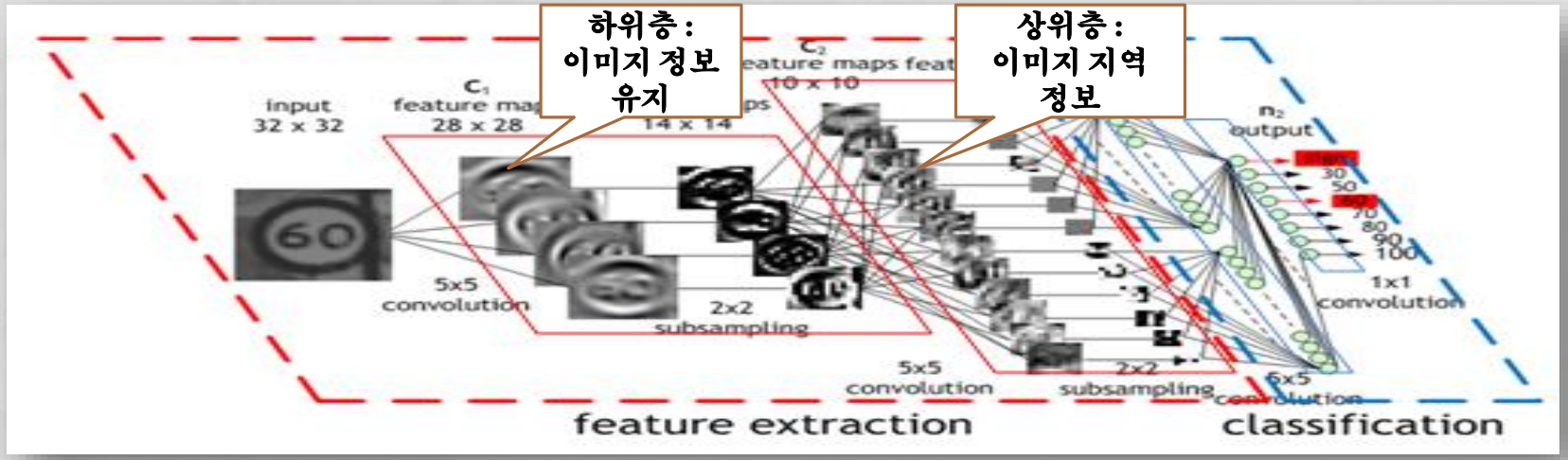
3. CNN(Convolution Neural Network) 예



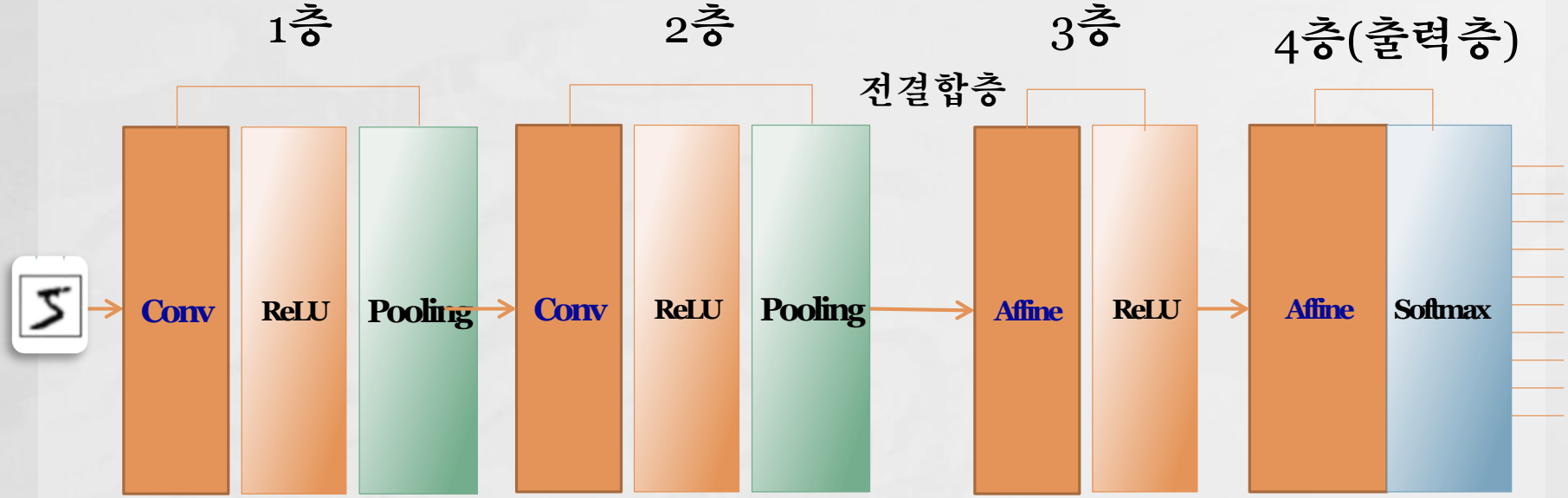
이미지 해상도 낮추고, 이미지 경계 강조

출력 준비

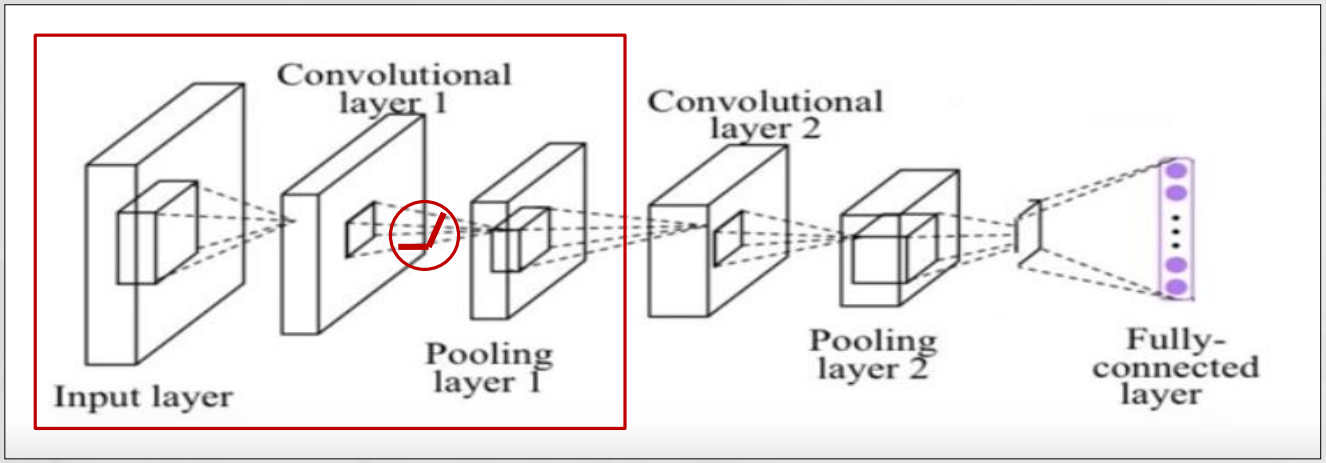
확률 예측



4. MNIST CNN model(4층 기준)

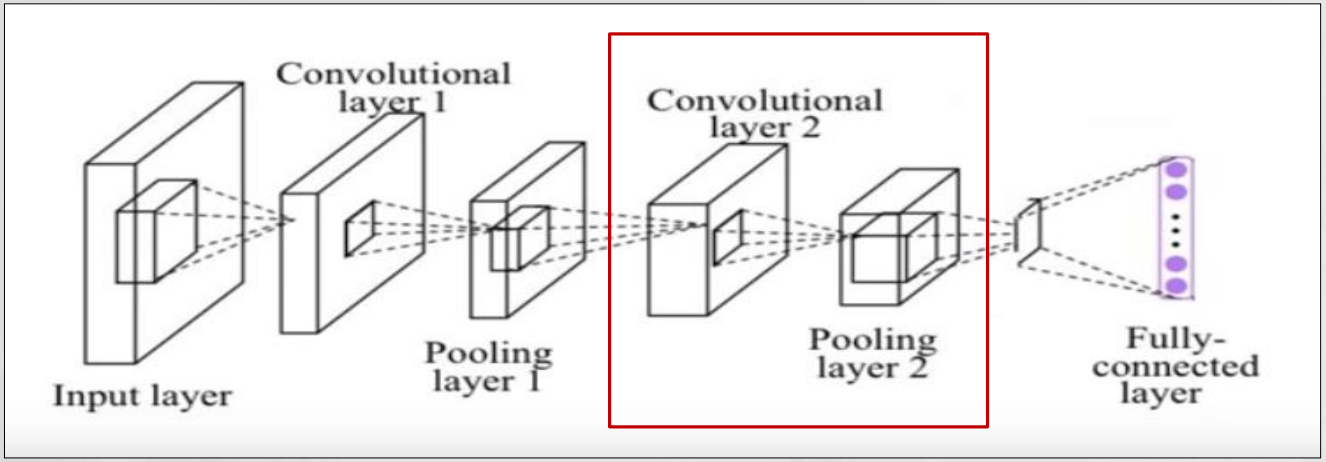


● *step03_mnist_cnn_layer.py(CNN Model 적용)*




```
# 1. Conv layer1(Conv->relu->pool)
X_img = tf.reshape(X, [-1, 28, 28, 1])
Filter1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
L1 = tf.nn.conv2d(X_img, Filter1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1_out = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],  strides=[1, 2, 2, 1], padding='SAME')
```

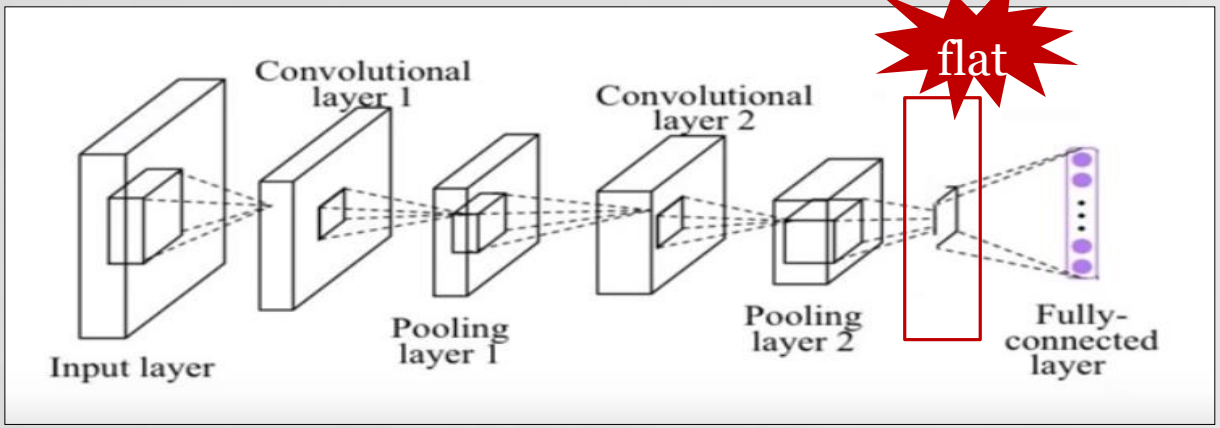
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32) : 합성곱1(입력size 동일)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32) : Relu 활성함수(변경 없음)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32) : 폴링(stride size 영향)



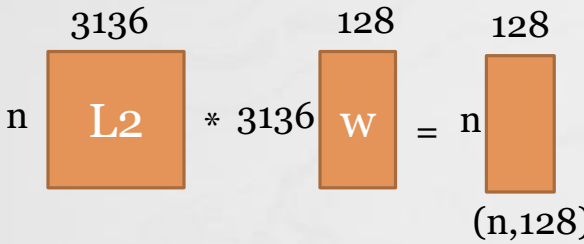
 # 2. Conv layer2(Conv->relu->pool)

```
Filter2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
L2 = tf.nn.conv2d(L1_out, Filter2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2_out = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],  strides=[1, 2, 2, 1], padding='SAME')
```

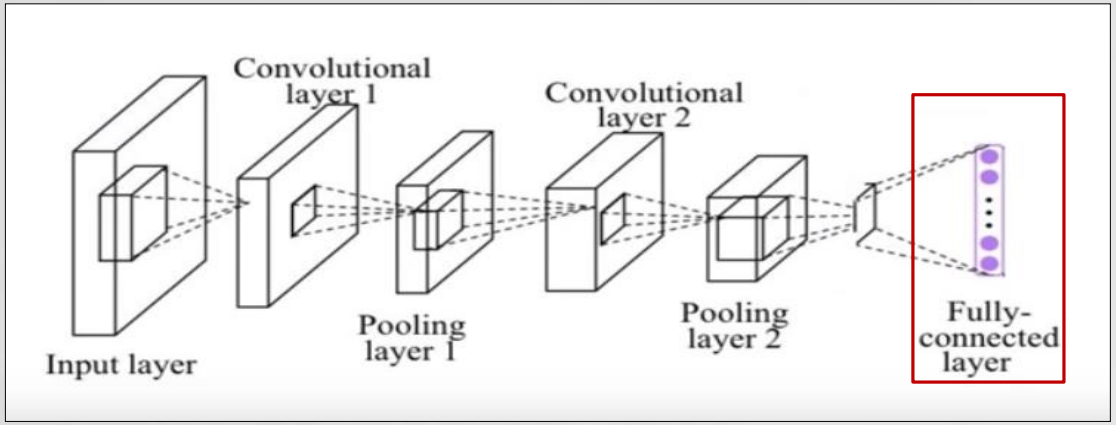
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32) : 합성곱2(입력 size 동일)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32) : Relu 활성함수(변경 없음)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32) : 폴링(stride size 영향)



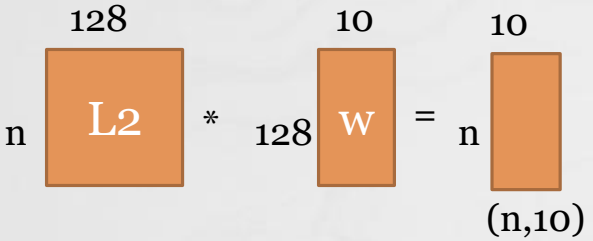
Flatten + Affine 행렬곱




```
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],  strides=[1, 2, 2, 1], padding='SAME')
#Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32) : 폴링(stride size 영향)

# 3. Flatten layer : 합성곱(3D) -> 행렬곱(1D)
n = 7 * 7 * 64 # n=3136
L2_flat = tf.reshape(L2_out, [-1, n]) # 3차원 -> 1차원
print(L2_flat) # Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)

# 4. Affine layer(Fully Connected + relu) : [3136, 128]
W1 = tf.Variable(tf.random_normal([n, 128]))
b1 = tf.Variable(tf.random_normal([128]))
affine_out = tf.nn.relu(tf.matmul(L2_flat, W1) + b1)
```



Affine + Output 행렬곱




```
# 5. Output layer(Fully Connected + softmax)
W2 = tf.Variable(tf.random_normal([128, 10]))
b2 = tf.Variable(tf.random_normal([10]))

# 1) model
model = tf.matmul(affine_out, W2) + b2
# 2) cost function : softmax + entropy
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
      labels=Y, logits=model))
# 3) optimizer
train = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

- ***Name scope + Tensorboard***

```
# 합성곱 계층 함수 정의
def conv2d_fun(Img, Fiter) :
    return tf.nn.conv2d(Img, Fiter, strides=[1,1,1,1], padding='SAME')

# 폴링 계층 함수 정의
def max_pool(X) :
    return tf.nn.max_pool(X, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

# 1. Conv layer1(Conv->relu->pool)
with tf.name_scope('Convolution1') as scope:
    Fiter1 = tf.Variable(tf.random_normal([3,3,1,32]))
    conv2d = conv2d_fun(X_img, Fiter1)
    L1 = tf.nn.relu(conv2d) #활성함수
    L1_out = max_pool(L1)
```

```
# 2. Conv layer2(Conv->relu->pool)
with tf.name_scope('Convolution2') as scope:
    Fiter2 = tf.Variable(tf.random_normal([3,3,32,64])) # [h,w,수일치,fmap]
    conv2d = conv2d_fun(L1_out, Fiter2)
    L2 = tf.nn.relu(conv2d) #활성함수
    L2_out = max_pool(L2)
```

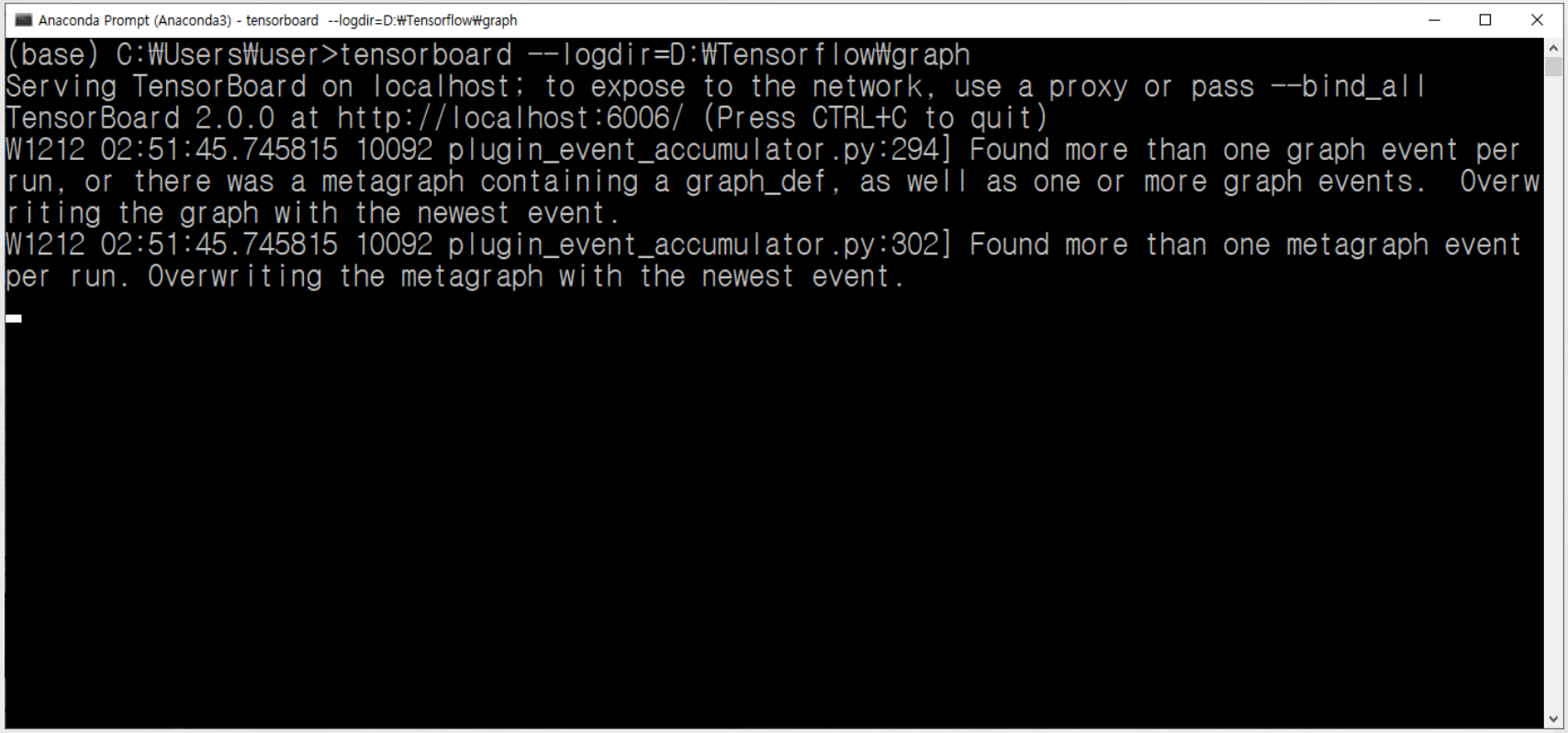
```
# 3. Flatten layer : 합성곱(3차원) -> 행렬곱(1차원)
with tf.name_scope('Flatten') as scope :
    n = 7 * 7 * 64
    L2_flat = tf.reshape(L2_out, [-1, n]) # 4차원 -> 2차원
```

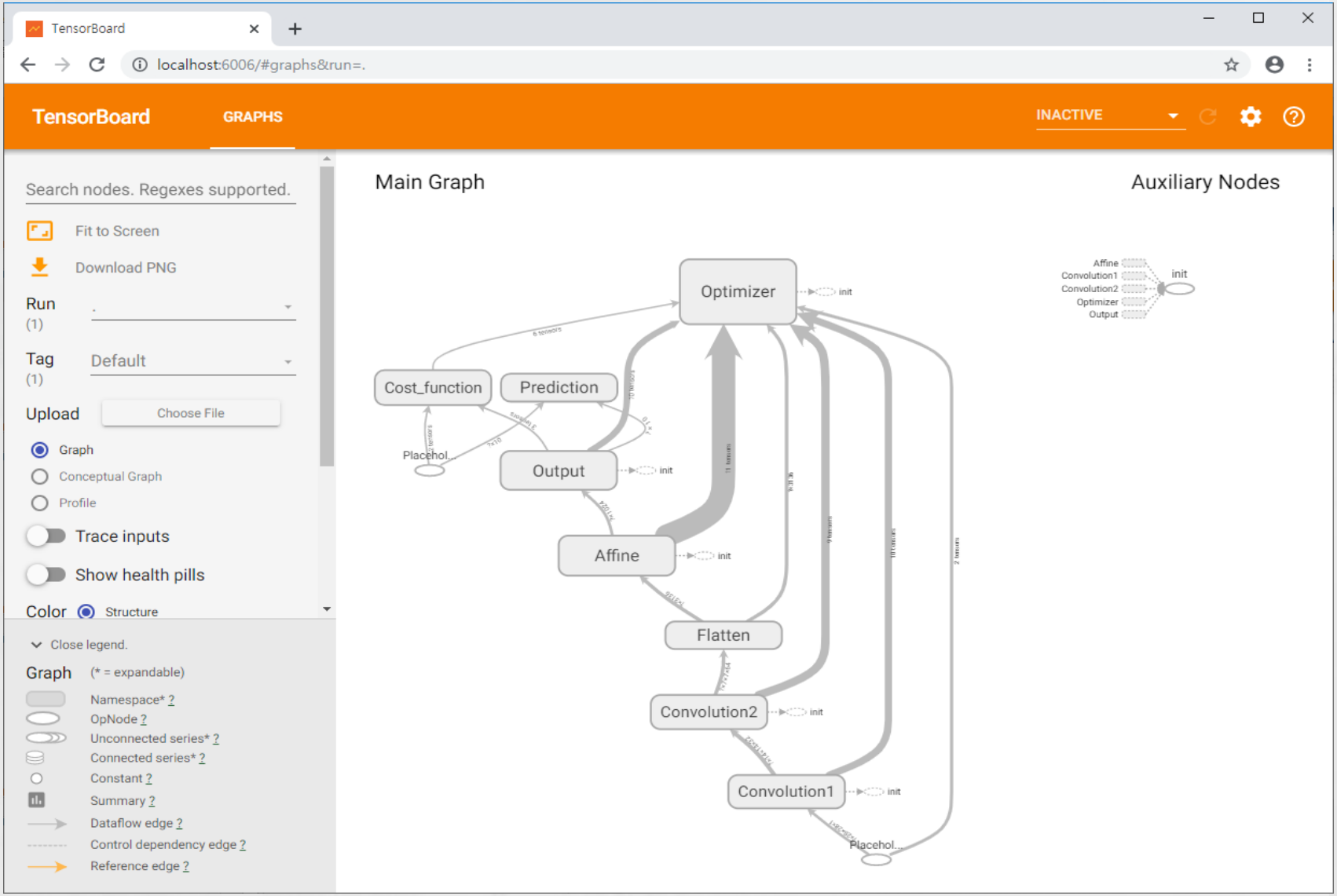
```
# 4. Affine layer(Fully Connected + relu)
with tf.name_scope('Affine') as scope :
    W1 = tf.Variable(tf.random_normal([n, 1024]), name="W1")# [input,
output]
    b1 = tf.Variable(tf.random_normal([1024]), name="b1")# 1024
    L3 = tf.matmul(L2_flat, W1) + b1
    L3_out = tf.nn.relu(L3)
```

```
# 5. Output layer(Fully Connected + softmax)
with tf.name_scope('Output') as scope :
    W2 = tf.Variable(tf.random_normal([1024, 10]), name="W2")
    b2 = tf.Variable(tf.random_normal([10]), name='b2')

    # 1) model
    model = tf.matmul(L3_out, W2) + b2
```

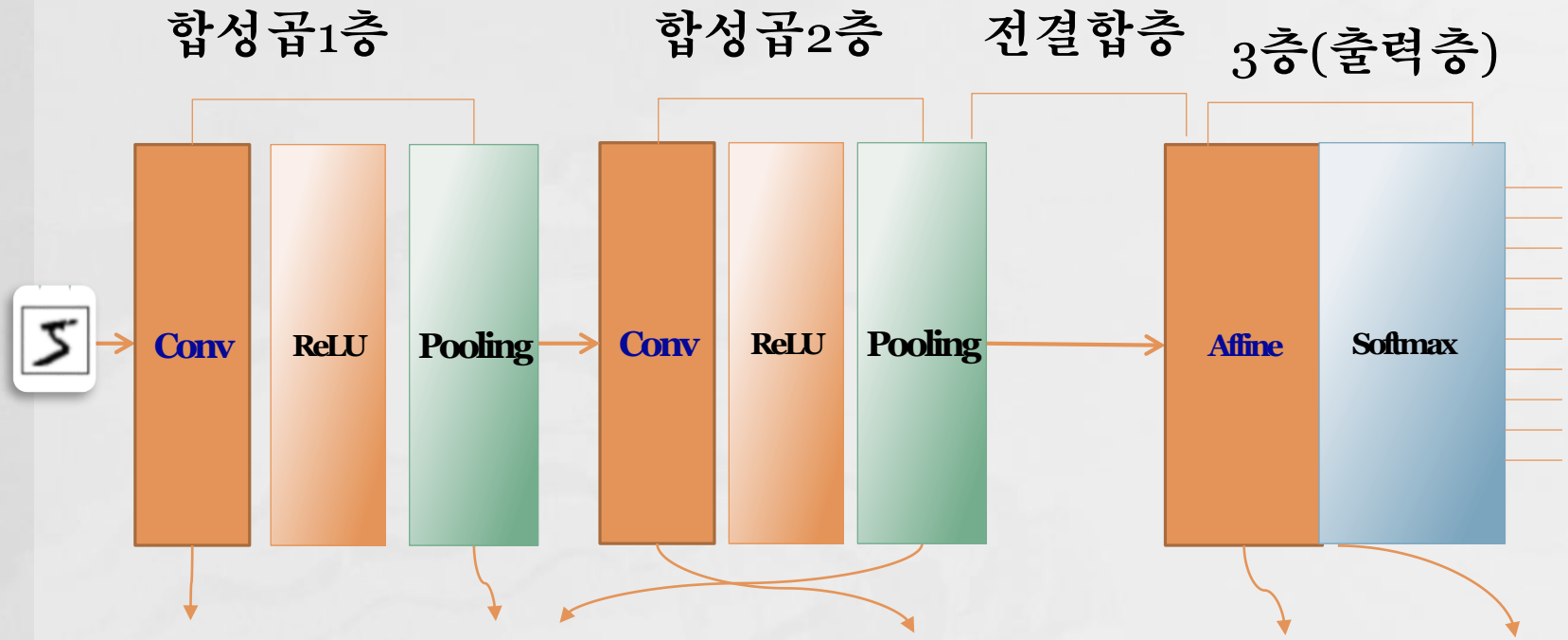
```
# 2) cost function : softmax + entropy
with tf.name_scope('Cost_function') as scope :
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
            labels=Y, logits=model))

# 3) optimizer
with tf.name_scope('Optimizer') as scope :
    train = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# 4) model 평가
with tf.name_scope('Prediction') as scope:
    pred = tf.argmax(model, 1)
    label = tf.argmax(Y, 1)
```

- ***Tensorboard 확인***




TensorBoard
localhost:6006/#graphs&run=.
TensorBoard
GRAPHS
INACTIVE
Search nodes. Regexes supported.
Fit to Screen
Download PNG
Run
(1)
.
Tag
(1)
Default
Upload
Choose File
Graph
Conceptual Graph
Profile
Trace inputs
Show health pills
Color
Structure
Close legend.
Graph
(* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
Main Graph
Auxiliary Nodes
Optimizer
init
Cost_function
Prediction
Placehol...
Output
Affine
Flatten
Convolution2
Convolution1
Placehol...
Affine
Convolution1
Convolution2
Optimizer
Output
init

exam_mnist_cnn_layer(MNIST CNN model 3층 기준)



Filter[5,5,1,32]
strides=[1, 1, 1, 1]
padding='SAME'

ksize[1,2,2,1]
strides=[1,2,2,1]
padding='SAME'

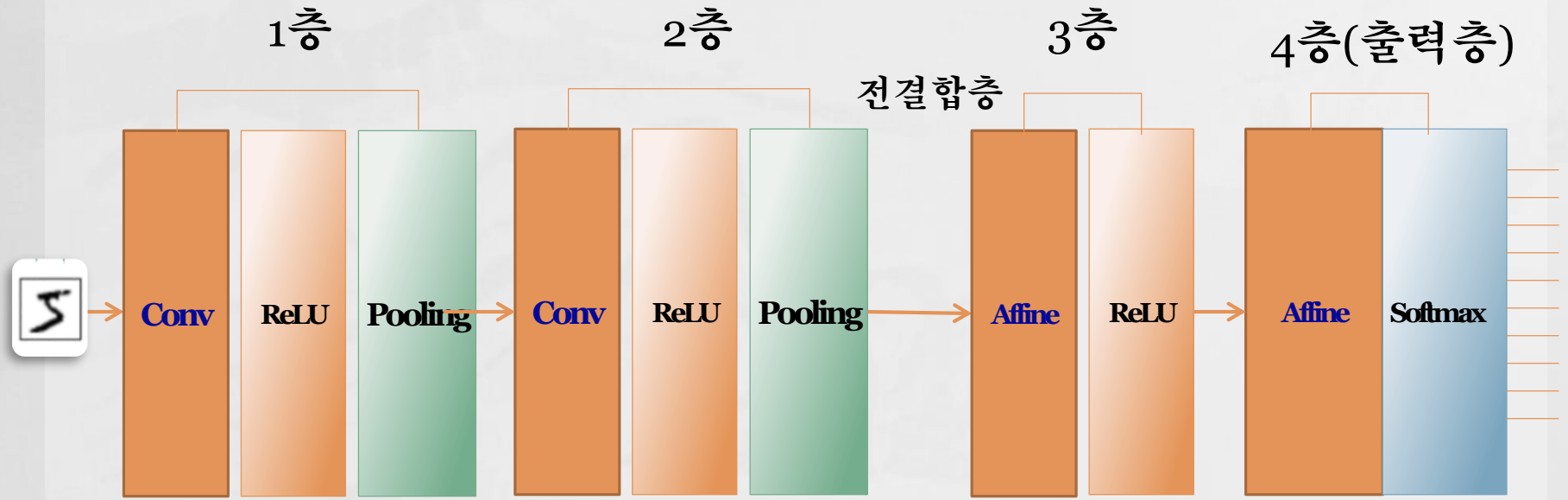
Filter[5,5,32,64]
strides=[1, 1, 1, 1]
padding='SAME'

[-1, 7x7x64]

[7x7x64,10]

5. Keras CNN model

- MNIST data set 적용



MNIST Keras CNN model 작성/실행

```
# 합성곱 model 생성
model = Sequential()
# 합성곱 1층 - [3, 3, 1, 32]
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
# 합성곱 2층 - [3, 3, 32, 64]
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
# Flatten 층
model.add(Flatten())

# Affine-ReLU 3층
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

# Affine-Softmax 4층(출력층)
model.add(Dense(10, activation='softmax'))
```

Tensorflow_1.x

```
# 1. Conv layer1(Conv->relu->pool)
Fiter1=tf.Variable(tf.truncated_normal([3,3,1,32]))
conv1=tf.nn.conv2d(X_img, Fiter1, strides=[1,1,1,1], padding='SAME')
L1 = tf.nn.relu(conv1) #활성함수
L1_out = tf.nn.max_pool(L1, ksize=[1,2,2,1], strides=[1,2,2,1],
padding='SAME')

# 2. Conv layer2(Conv->relu->pool)
Fiter2=tf.Variable(tf.truncated_normal([3,3,32,64]))
conv2=tf.nn.conv2d(L1_out, Fiter2, strides=[1,1,1,1],padding='SAME')
L2 = tf.nn.relu(conv2) #활성함수
L2_out = tf.nn.max_pool(L2, ksize=[1,2,2,1], strides=[1,2,2,1],
padding='SAME')

# 3. Flatten layer : 합성곱(3차원) -> 행렬곱(1차원)
n = 8 * 8 * 64
flat_out = tf.reshape(L2_out, [-1, n]) # 3차원 -> 1차원

# 4. Affine layer(Fully connected + relu) : [n, 128]
W1 = tf.Variable(tf.random_normal([n, 128]))
b1 = tf.Variable(tf.random_normal([128]))
affine_out = tf.nn.relu(tf.matmul(L2_flat, W1) + b1)

# 5. Output layer(Fully connected + softmax)
W2 = tf.Variable(tf.random_normal([n, 10]) )
b2 = tf.Variable(tf.random_normal([10]) )
model = tf.matmul(flat_out, W2) + b2
softmax = tf.nn.softmax(model)
```

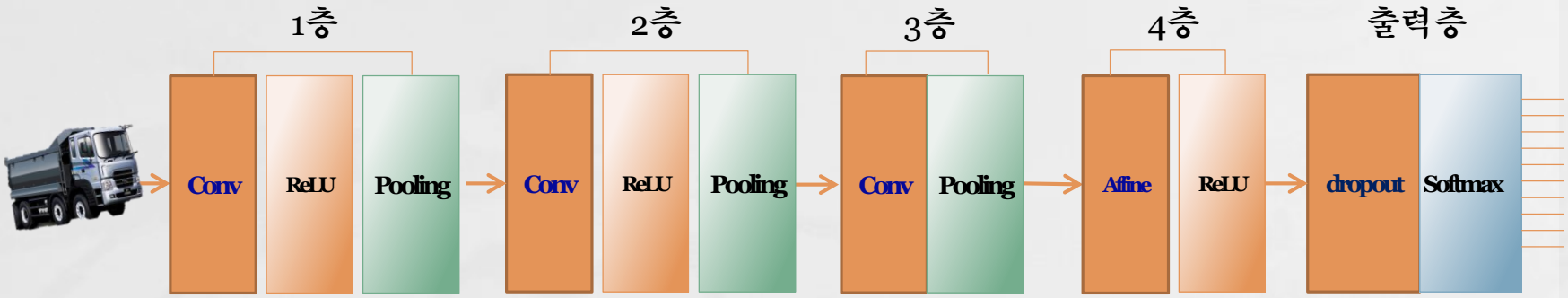
Tensorflow_2.x

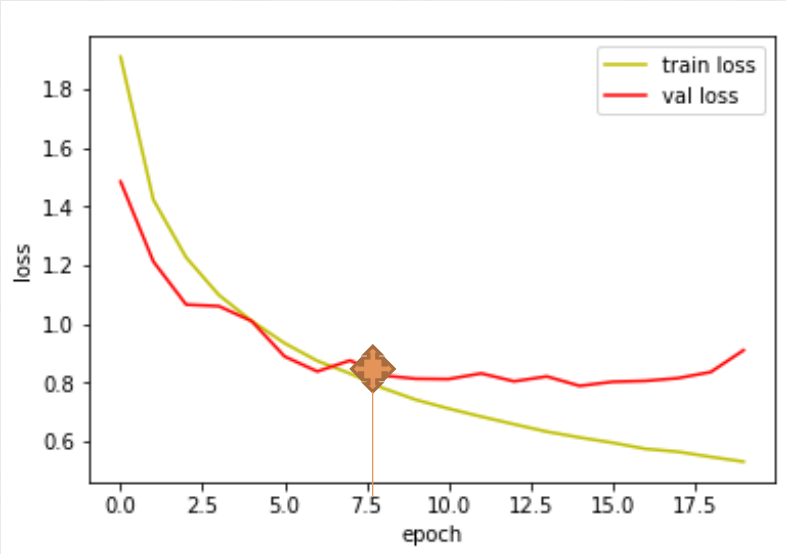
```
#1. Convolution1 : [3,3,1,32]
model.add(Conv2D(32, kernel_size=(3, 3),
padding='same',activation='relu',  input_shape =
input_shape))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2,
2), padding='same'))

#2. Convolution2 : [3,3,32,64]
model.add(Conv2D(64, kernel_size=(3, 3),
padding='same', activation='relu')))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2,
2), padding='same'))

#3. Flatten layer :3d -> 1d
model.add(Flatten())

#4. Affine layer(Fully connected + relu) : [n, 128]
model.add(Dense(128, activation = 'relu'))

#5. Output layer(Fully connected + softmax) : [128, 10]
model.add(Dense(10, activation = 'softmax'))
```

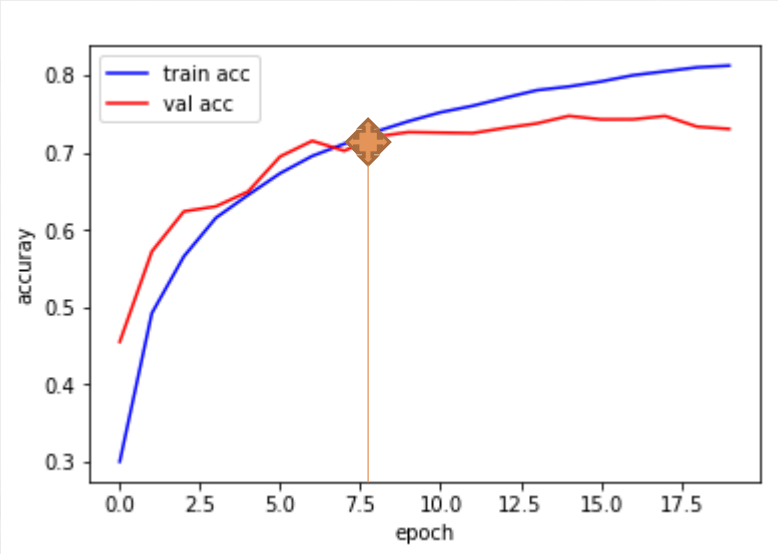
```
# 모델 학습
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=100,
          epochs=1,
          verbose=1,
          validation_data=(x_test, y_test))

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

```
58500/60000 [==============================>.] - ETA: 3s - loss: 0.4434 - acc: 0.8603
58600/60000 [==============================>.] - ETA: 3s - loss: 0.4430 - acc: 0.8604
58700/60000 [==============================>.] - ETA: 3s - loss: 0.4425 - acc: 0.8606
58800/60000 [==============================>.] - ETA: 3s - loss: 0.4423 - acc: 0.8607
58900/60000 [==============================>.] - ETA: 2s - loss: 0.4419 - acc: 0.8608
59000/60000 [==============================>.] - ETA: 2s - loss: 0.4416 - acc: 0.8609
59100/60000 [==============================>.] - ETA: 2s - loss: 0.4412 - acc: 0.8610
59200/60000 [==============================>.] - ETA: 2s - loss: 0.4408 - acc: 0.8611
59300/60000 [==============================>.] - ETA: 1s - loss: 0.4404 - acc: 0.8612
59400/60000 [==============================>.] - ETA: 1s - loss: 0.4398 - acc: 0.8614
59500/60000 [==============================>.] - ETA: 1s - loss: 0.4395 - acc: 0.8615
59600/60000 [==============================>.] - ETA: 1s - loss: 0.4391 - acc: 0.8617
59700/60000 [==============================>.] - ETA: 0s - loss: 0.4389 - acc: 0.8618
59800/60000 [==============================>.] - ETA: 0s - loss: 0.4385 - acc: 0.8619
59900/60000 [==============================>.] - ETA: 0s - loss: 0.4382 - acc: 0.8620
60000/60000 [==============================] - 162s - loss: 0.4377 - acc: 0.8622
Test loss: 0.103321682511
Test accuracy: 0.9681
```

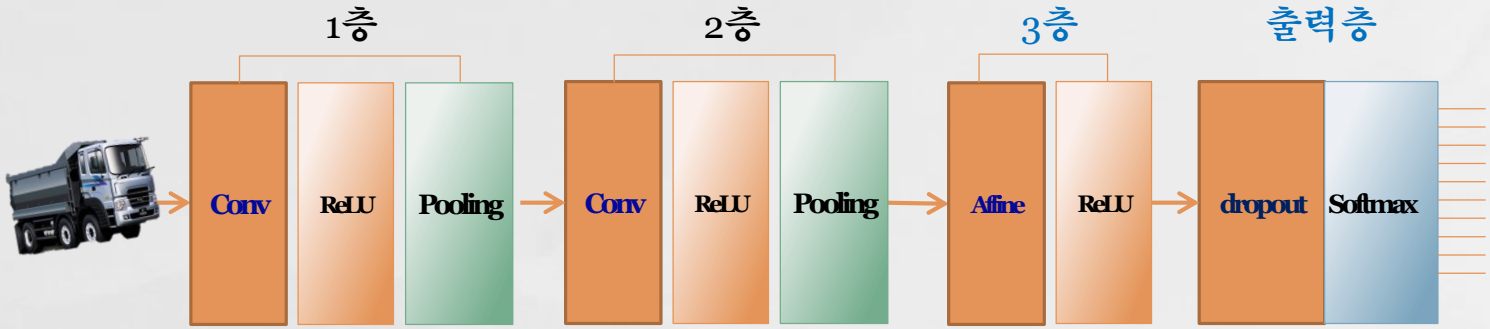
- cifar10 data set 적용





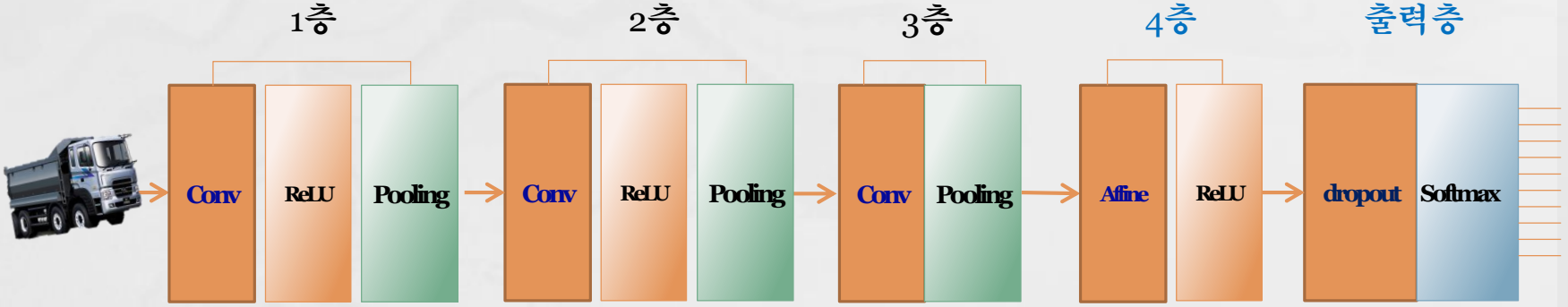


Convolution layer(2) + Affine(2)



Adam - Testing loss: 0.7945309375762939, acc: 0.7244
RMSprop - Testing loss: 0.8798973457336425, acc: 0.7173

Convolution layer(3) + Affine(2)



Adam - Testing loss: 0.745222040271759, acc: 0.7458
RMSprop - Testing loss: 0.9104682565689087, acc: 0.7302