



Chapter

# 1

## CNN 구현

1. CNN 기본
2. CNN with MNIST
3. Keras
4. Cat vs Dog Image classification

# 강의에 앞서서..

❖ 본 문서는 아래의 자료들을 활용하여 만들어 졌음을 알립니다

❖ **모두를 위한 딥러닝 강좌**

- 네이버 Search & Clova AI 부분 리더 김성훈 교수님
- [https://www.youtube.com/playlist?list=PLIMkM4tgfjnLSOjrEJN31gZATbcj\\_MpUm](https://www.youtube.com/playlist?list=PLIMkM4tgfjnLSOjrEJN31gZATbcj_MpUm)
- <https://www.edwith.org/boostcourse-dl-tensorflow/lecture/43739/>

❖ **스탠포드 대학 CNN 강좌**

- Fei-Fei Li & Andrej Karpathy & Justin Johnson
- <http://cs231n.stanford.edu/slides/2020/>

CS231n: Convolutional Neural Networks for Visual Recognition

- This course, Prof. Fei-Fei Li & Justin Johnson & Serena Yeung
- Focusing on applications of deep learning to computer vision

# 강의에 앞서서..

---

## ❖ A Beginner's Guide To Understanding Convolutional Neural Networks

- <https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>

## ❖ DeepLearning Getting Started with TensorFlow

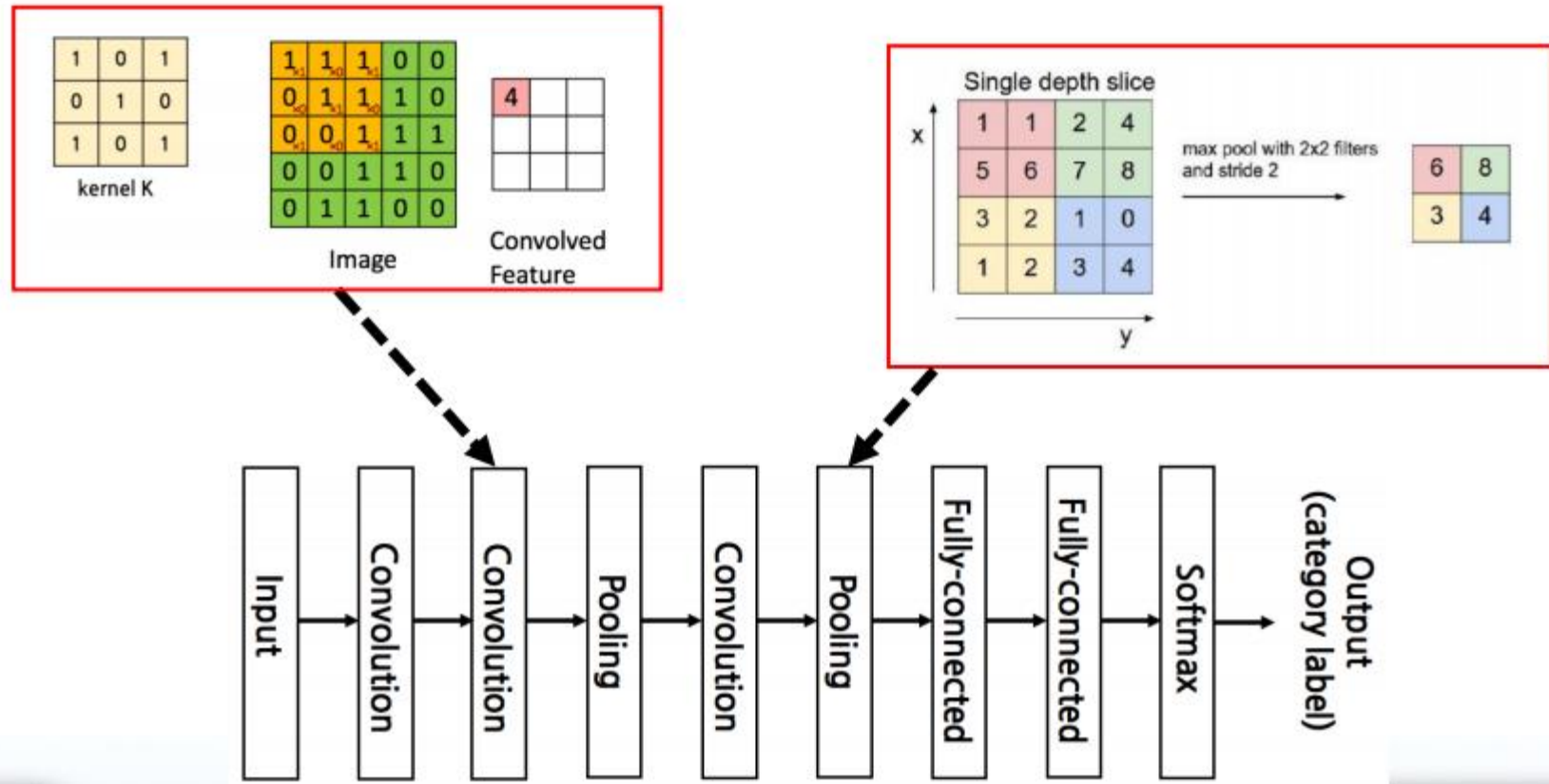
- <https://github.com/Jpub/TensorflowDeeplearning>



1

# 1. CNN 기본

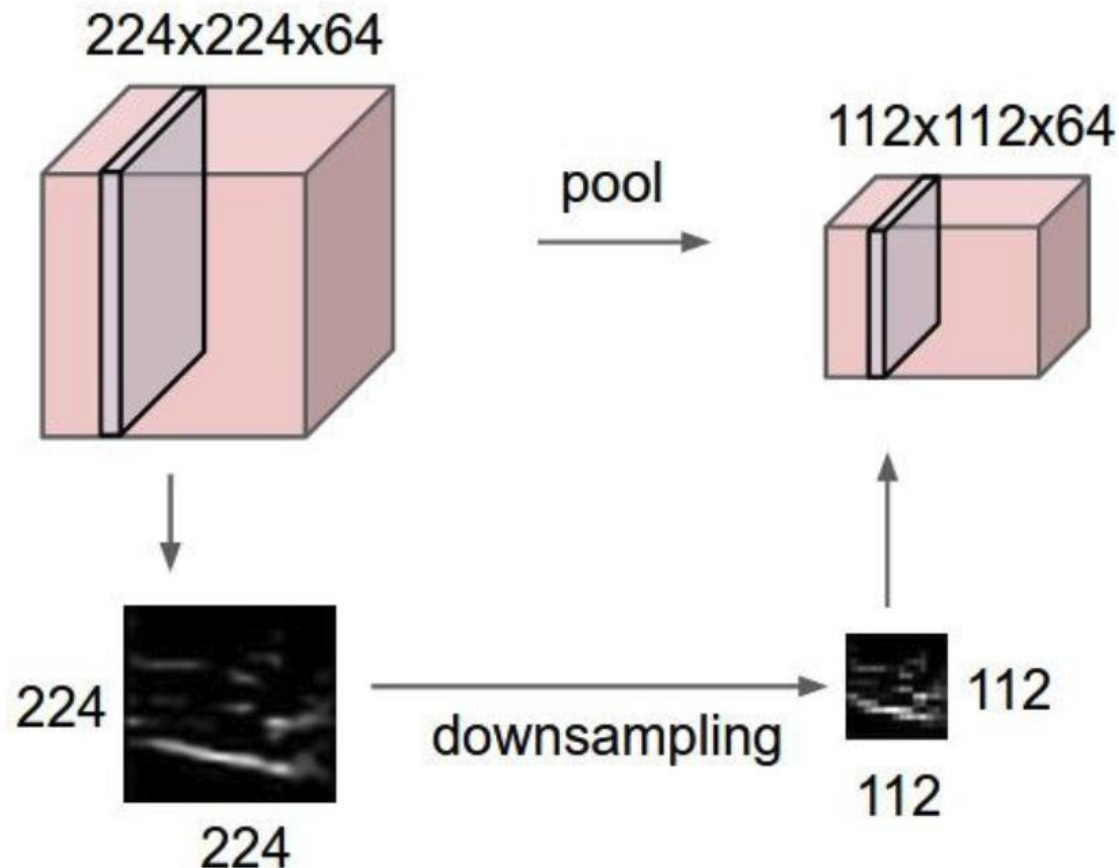
# CNN 구조



# Pooling Layer

## ❖ Pooling

- 데이터의 사이즈를 줄이거나 강조할 때 사용 (stride, filter, padding 개념 사용)



# Pooling Layer

## ❖ Max pooling

- stride 의 크기만큼 이동할 때 해당 filter 에서 가장 큰 값을 뽑아낸다
- 의미상 min 도 있지만 거의 안 쓰기 때문에 tensorflow 에서 미지원
  - Average 는 지원

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

원본  $4 \times 4 \times 1$

$2 \times 2 \times 1$  filter

Stride = 2

|    |    |
|----|----|
| 6  | 8  |
| 14 | 16 |

# Pooling Layer

## ❖ Pooling

**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 = tf.constant([[[[4],[3]],  
                     [[2],[1]]]], dtype=np.float32)  
pool = keras.layers.MaxPool2D(pool_size=(2,2), strides=1, padding='SAME')(image)  
print(pool.shape)  
print(pool.numpy())
```

```
(1, 2, 2, 1)
```

```
[[[4.]  
  [3.]
```

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



# Pooling Layer

## ❖ Pooling - padding 고급

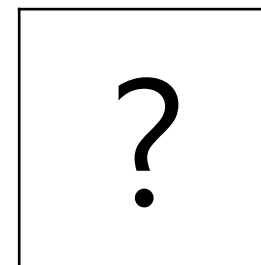
- padding = 'VALID' 일때와 'SAME' 일 때 계산해보기

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| 1  | 2  | 3  | 4  | 5  | 6  |
| 7  | 8  | 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |

원본 6 x 6 x 1

5 x 5 x 1 filter

Stride = 4

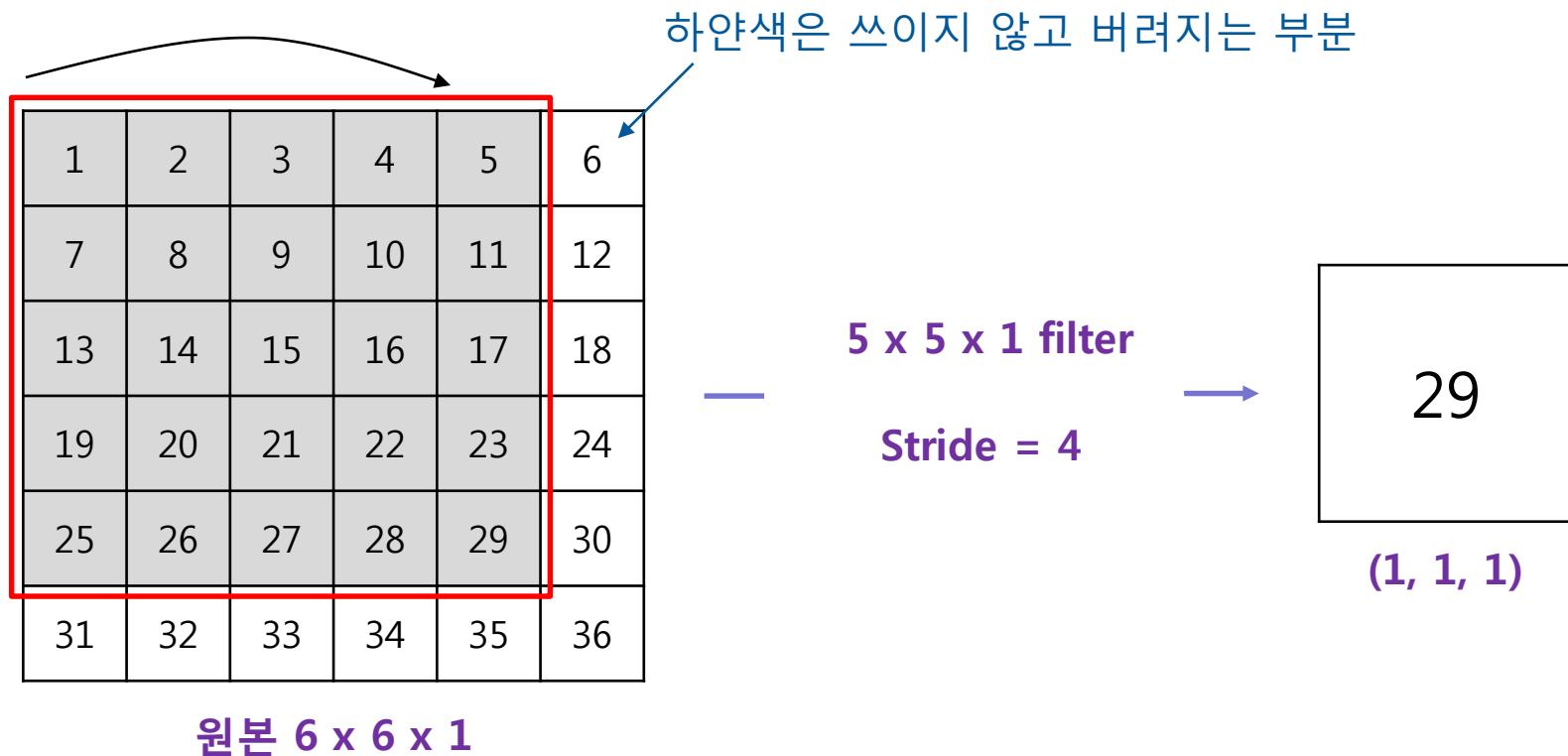


(?, ?, 1)

# Pooling Layer

## ❖ Pooling - padding 고급

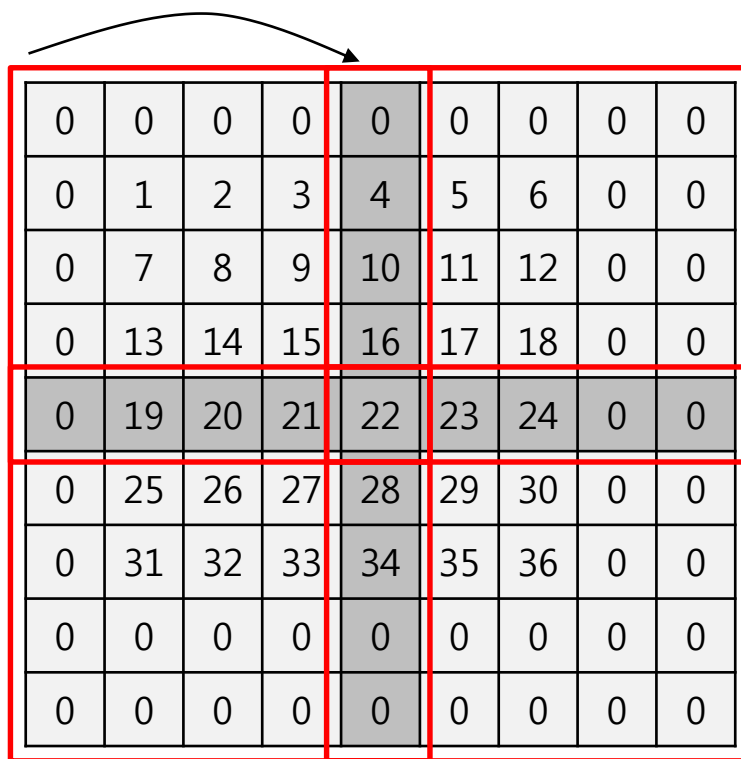
- padding = 'VALID'



# Pooling Layer

## ❖ Pooling - padding 고급

- padding = 'SAME' -> stride = 4, filter = 5 x 5 을 완수하려면 3칸이 더 필요



|   |    |    |    |    |    |    |   |   |
|---|----|----|----|----|----|----|---|---|
| 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 0 | 0 |
| 0 | 7  | 8  | 9  | 10 | 11 | 12 | 0 | 0 |
| 0 | 13 | 14 | 15 | 16 | 17 | 18 | 0 | 0 |
| 0 | 19 | 20 | 21 | 22 | 23 | 24 | 0 | 0 |
| 0 | 25 | 26 | 27 | 28 | 29 | 30 | 0 | 0 |
| 0 | 31 | 32 | 33 | 34 | 35 | 36 | 0 | 0 |
| 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 |
| 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 |

원본 6 x 6 x 1

5 x 5 x 1 filter

Stride = 4

|    |    |
|----|----|
| 22 | 24 |
| 34 | 36 |

(2, 2, 1)



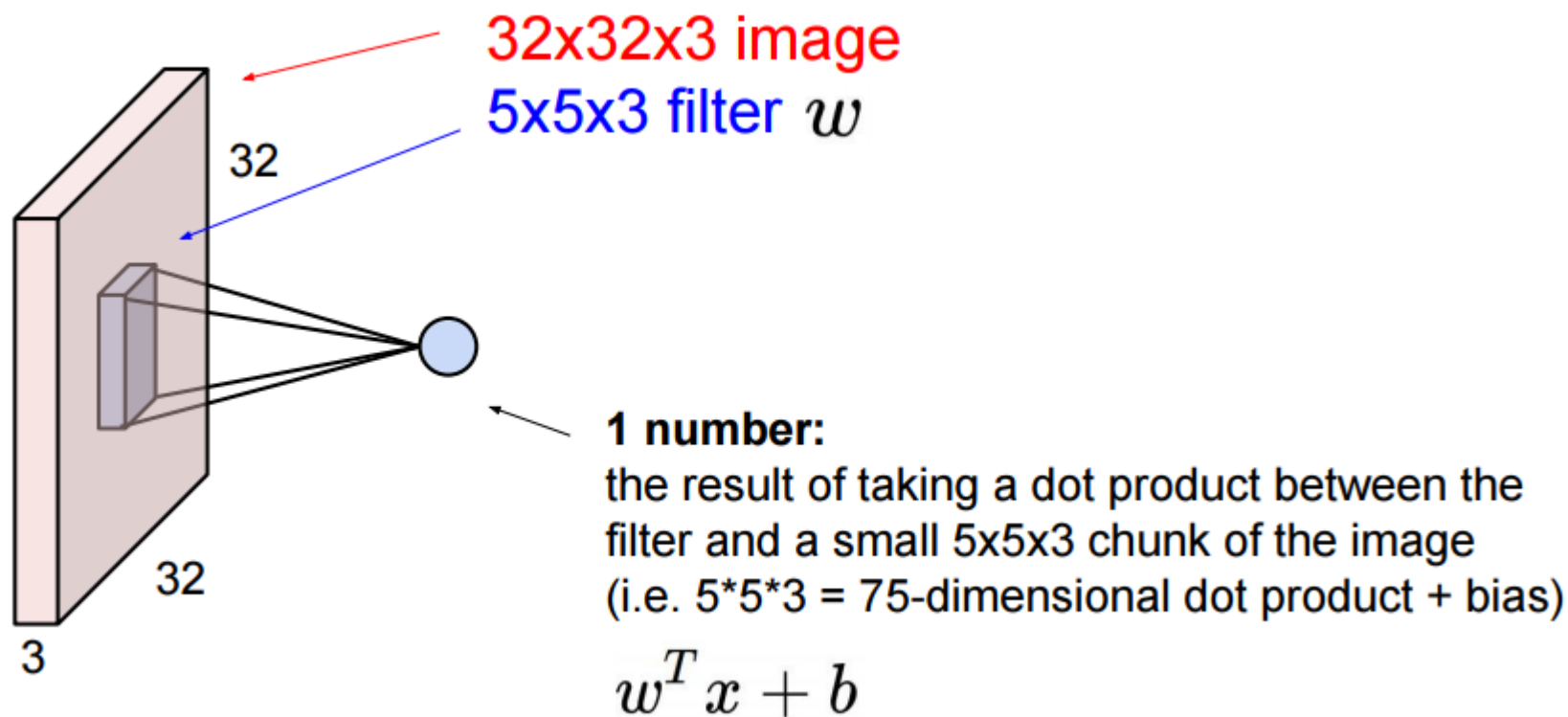
1

## 2. CNN 기본 2

# Filter

❖ Filter == Kernel mask == weight

❖ 특징을 추출하는 마스크

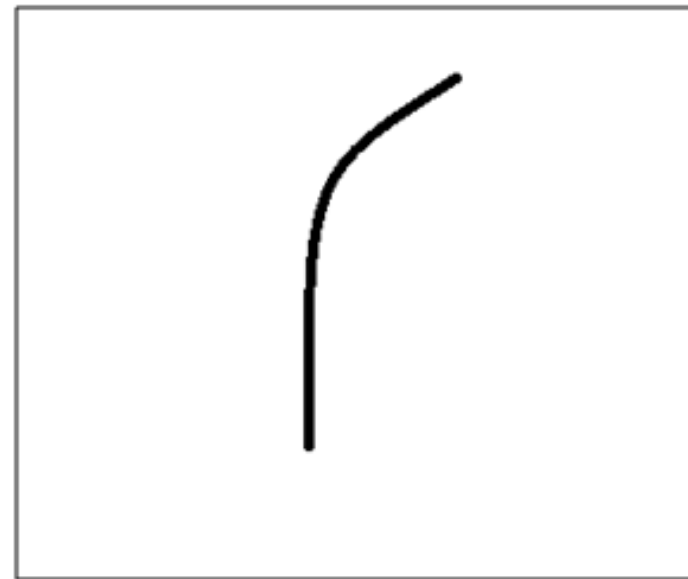


# Filter

- ❖ 특징을 검출 마스크
- ❖ 단순한 특징 : 직선, 모서리, 단순한 색상, 곡선
- ❖ 곡선 검출 필터 예

|   |   |   |    |    |    |   |
|---|---|---|----|----|----|---|
| 0 | 0 | 0 | 0  | 0  | 30 | 0 |
| 0 | 0 | 0 | 0  | 30 | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 0  | 0  | 0  | 0 |

Pixel representation of filter

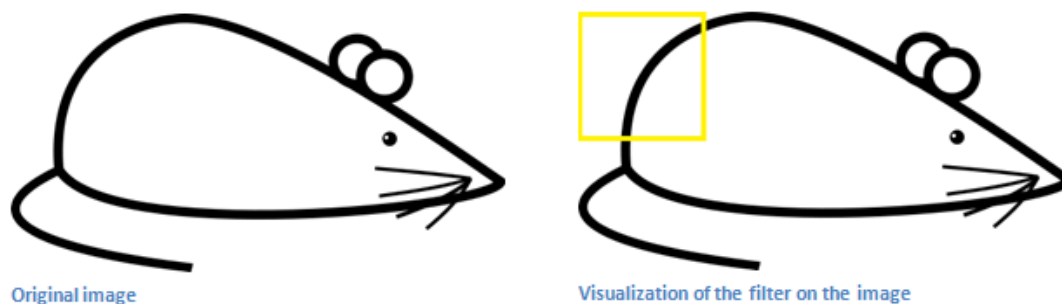


Visualization of a curve detector filter

<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>

# Filter

## ❖ 곡선 검출 필터 예



Visualization of the receptive field

|   |   |   |    |    |    |    |
|---|---|---|----|----|----|----|
| 0 | 0 | 0 | 0  | 0  | 0  | 30 |
| 0 | 0 | 0 | 0  | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0  | 0  |
| 0 | 0 | 0 | 50 | 50 | 0  | 0  |
| 0 | 0 | 0 | 50 | 50 | 0  | 0  |
| 0 | 0 | 0 | 50 | 50 | 0  | 0  |
| 0 | 0 | 0 | 50 | 50 | 0  | 0  |

Pixel representation of the receptive field

\*

|   |   |   |    |    |    |   |
|---|---|---|----|----|----|---|
| 0 | 0 | 0 | 0  | 0  | 30 | 0 |
| 0 | 0 | 0 | 0  | 30 | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 0  | 0  | 0  | 0 |

Pixel representation of filter

Multiplication and Summation =  $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$  (A large number!)

곡선 검출 필터의 weighted sum 이 큰 값으로 계산, 곡선일 가능성이 높다

# Filter

## ❖ 곡선 검출 필터 예



Visualization of the filter on the image

|    |    |    |    |   |   |   |
|----|----|----|----|---|---|---|
| 0  | 0  | 0  | 0  | 0 | 0 | 0 |
| 0  | 40 | 0  | 0  | 0 | 0 | 0 |
| 40 | 0  | 40 | 0  | 0 | 0 | 0 |
| 40 | 20 | 0  | 0  | 0 | 0 | 0 |
| 0  | 50 | 0  | 0  | 0 | 0 | 0 |
| 0  | 0  | 50 | 0  | 0 | 0 | 0 |
| 25 | 25 | 0  | 50 | 0 | 0 | 0 |

Pixel representation of receptive field

\*

|   |   |   |    |    |    |   |
|---|---|---|----|----|----|---|
| 0 | 0 | 0 | 0  | 0  | 30 | 0 |
| 0 | 0 | 0 | 0  | 30 | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 30 | 0  | 0  | 0 |
| 0 | 0 | 0 | 0  | 0  | 0  | 0 |

Pixel representation of filter

Multiplication and Summation = 0

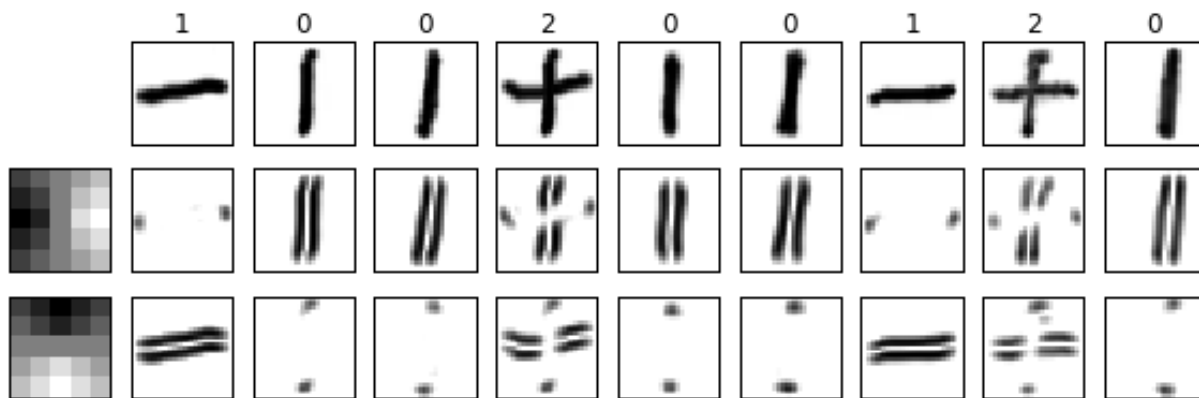
곡선 검출 필터의 weighted sum 이 0 값으로 계산, 곡선일 가능성이 없다



# Filter

## ❖ 에지검출필터

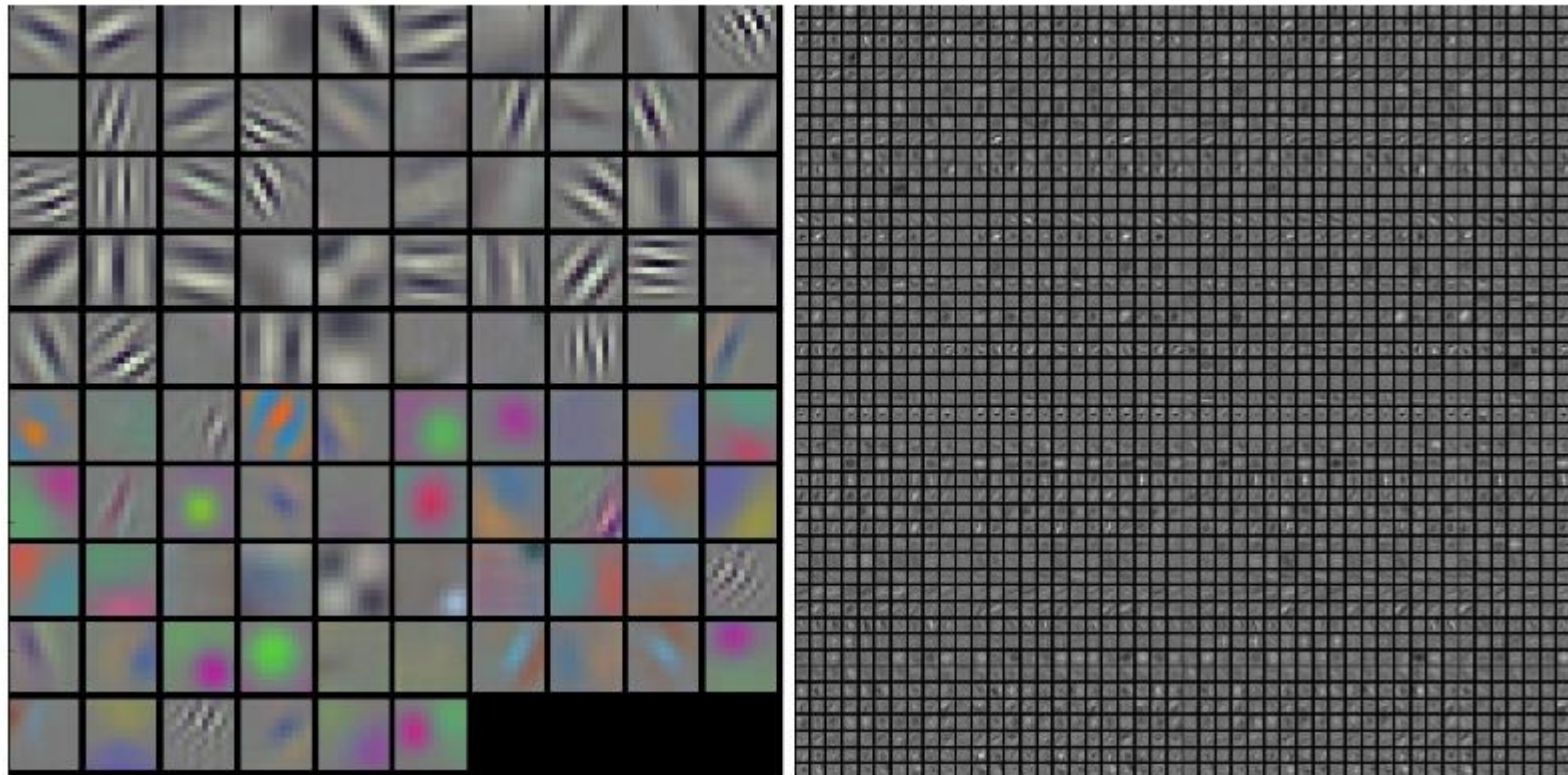
```
def edge_filter():  
    filter0 = np.array(  
        [[ 2, 1, 0, -1, -2],  
         [ 3, 2, 0, -2, -3],  
         [ 4, 3, 0, -3, -4],  
         [ 3, 2, 0, -2, -3],  
         [ 2, 1, 0, -1, -2]]) / 23.0  
    filter1 = np.array(  
        [[ 2, 3, 4, 3, 2],  
         [ 1, 2, 3, 2, 1],  
  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



원본 이미지와 두 종류의 필터를 적용한 결과  
왼쪽은 적용한 필터를 이미지화한 것

# Filter

- ❖ 학습된 AlexNet의 첫 번째 CONV 레이어, 두 번째 CONV 레이어의 일반적인 필터



<https://cs231n.github.io/understanding-cnn/>

# Filter

## ❖ filter = weight

- 필터의 값은 Neural Network 에서 Weight 에 해당되는 개념
- 학습을 통해 값을 적절히 변경

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

**3 x 3 x 1 image**  
(1, 3, 3, 1)

|    |    |
|----|----|
| w1 | w2 |
| w3 | w4 |

**2 x 2 x 1 filter**  
(2, 2, 1, 1)

**Stride = 1**  
[1, 1, 1, 1]

**padding = 'VALID'**

|                         |                         |
|-------------------------|-------------------------|
| $w1 + 2w2 + 4w3 + 5w4$  | $2w1 + 3w2 + 5w3 + 6w4$ |
| $4w1 + 5w2 + 7w3 + 8w4$ | $5w1 + 6w2 + 8w3 + 9w4$ |

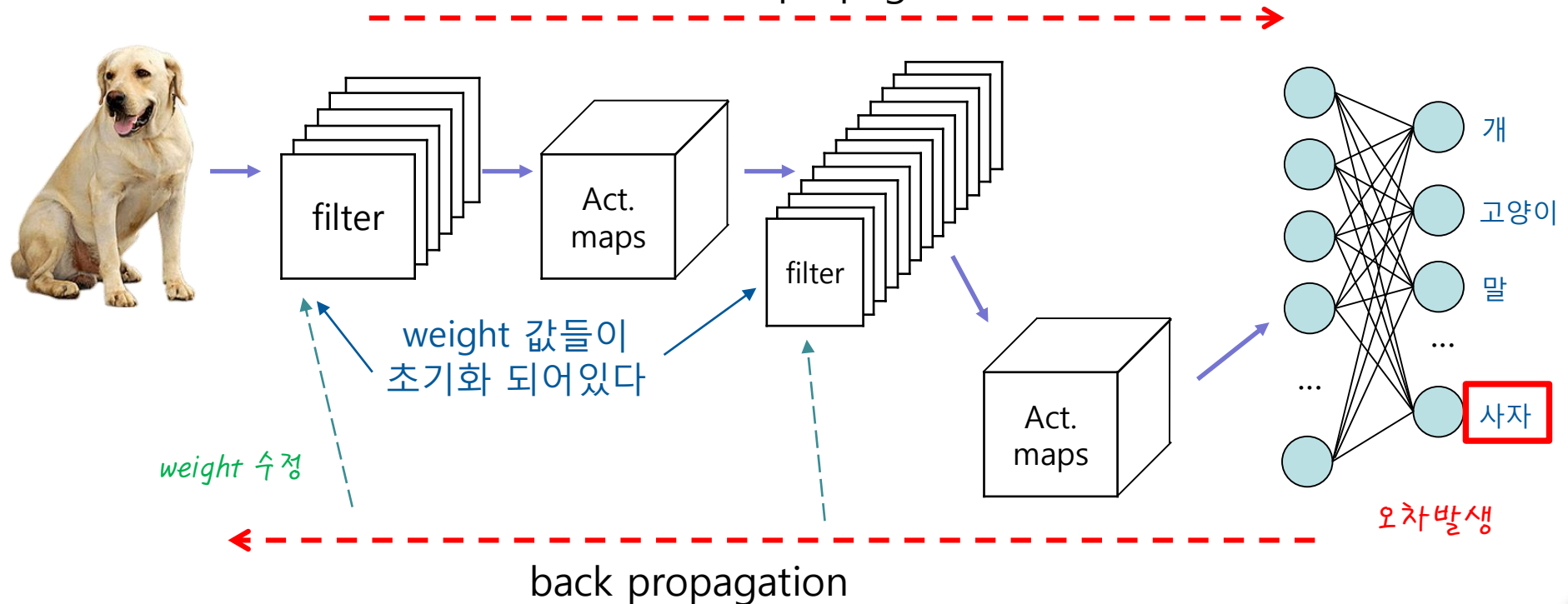
**Activation map**  
(1, 2, 2, 1)

# Filter

## ❖ Forward & Back propagation

- Filter 의 weight 를 학습!!

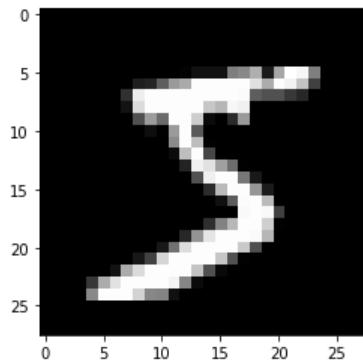
input 값을 넣었을 때 output 으로 나오는 과정  
forward propagation



오차가 발생했을 경우 뒤로 전달하면서 weight 와 bias 를 적당히 수정

# Filter

## ❖ MNIST Feature map



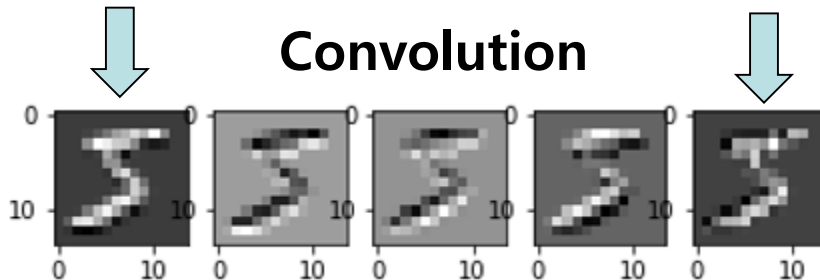
**X**

weight1

...

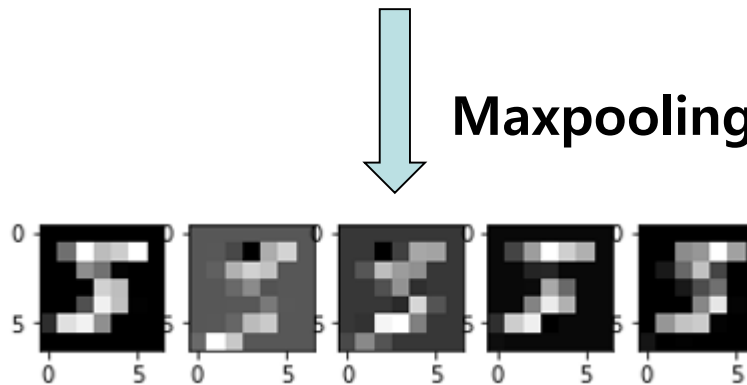
weight5

**Convolution**



Feature map

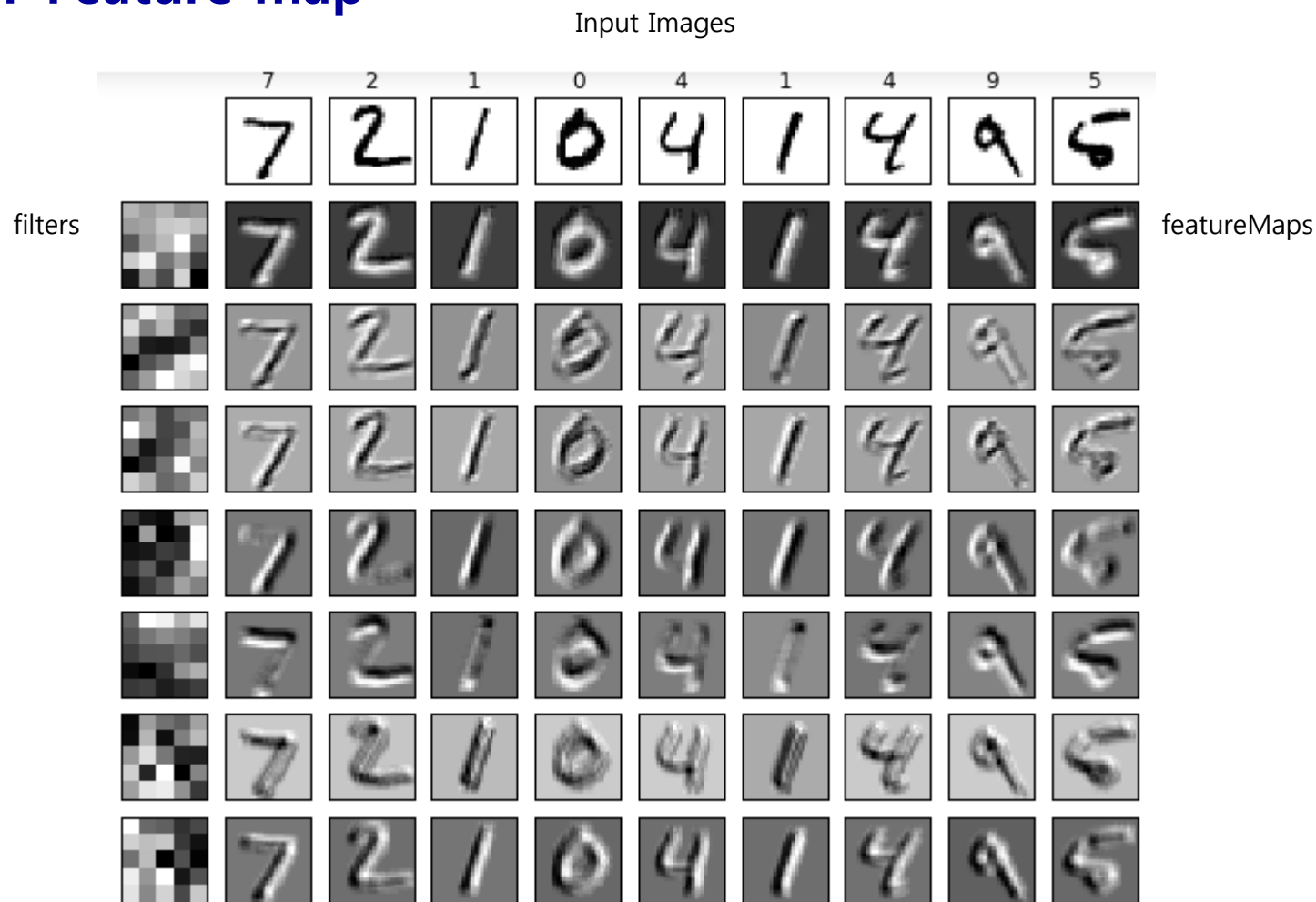
**Maxpooling**



Feature map

# Filter

## ❖ MNIST Feature map



# Filter

- [ConvNetJS](https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html) CIFAR-10 demo
- <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>



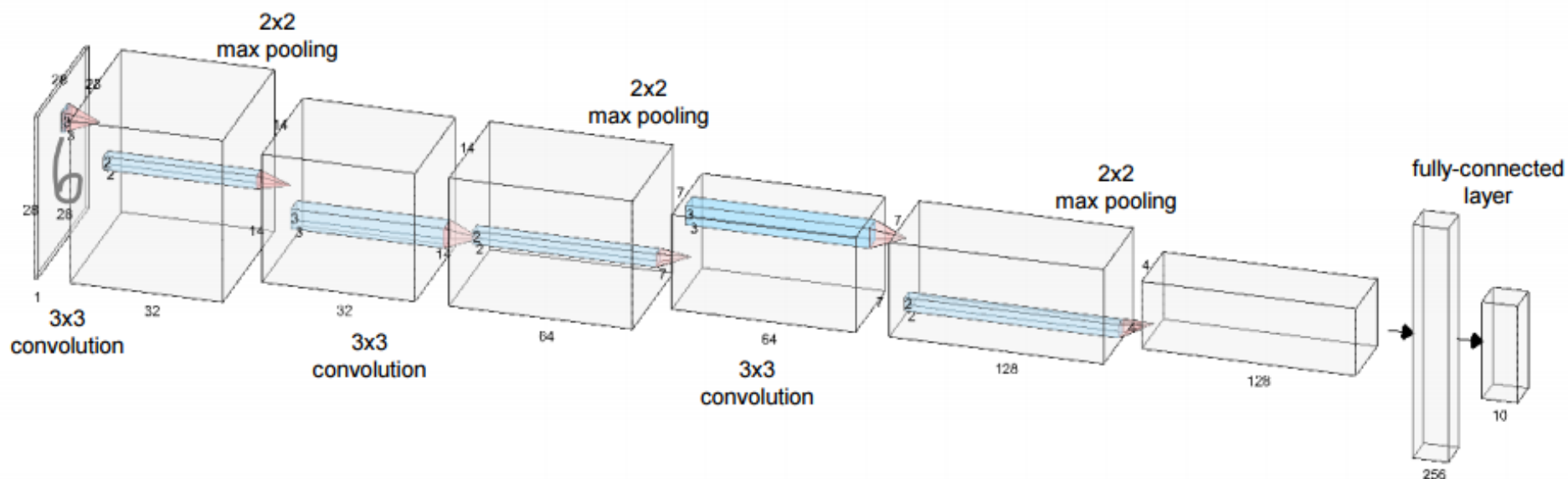


1

## 4. MNIST with CNN



# MNIST with CNN



# MNIST with CNN

## ❖ lab-11-1-mnist-cnn-keras-sequential-eager.ipynb

```
[ ] def create_model():  
    model = keras.Sequential()  
    model.add(keras.layers.Conv2D(filters=32, kernel_size=3, activation=tf.nn.relu, padding='SAME',  
                                   input_shape=(28, 28, 1)))  
    model.add(keras.layers.MaxPool2D(padding='SAME'))  
    model.add(keras.layers.Conv2D(filters=64, kernel_size=3, activation=tf.nn.relu, padding='SAME'))  
    model.add(keras.layers.MaxPool2D(padding='SAME'))  
    model.add(keras.layers.Conv2D(filters=128, kernel_size=3, activation=tf.nn.relu, padding='SAME'))  
    model.add(keras.layers.MaxPool2D(padding='SAME'))  
    model.add(keras.layers.Flatten())  
    model.add(keras.layers.Dense(256, activation=tf.nn.relu))  
    model.add(keras.layers.Dropout(0.4))  
    model.add(keras.layers.Dense(10))  
    return model
```

# MNIST with CNN

## ❖ lab-11-1-mnist-cnn-keras-sequential-eager.ipynb

```
model = create_model()  
model.summary()
```

Model: "sequential\_2"

| Layer (type)                   | Output Shape       | Param # |
|--------------------------------|--------------------|---------|
| conv2d_6 (Conv2D)              | (None, 28, 28, 32) | 320     |
| max_pooling2d_6 (MaxPooling2D) | (None, 14, 14, 32) | 0       |
| conv2d_7 (Conv2D)              | (None, 14, 14, 64) | 18496   |
| max_pooling2d_7 (MaxPooling2D) | (None, 7, 7, 64)   | 0       |
| conv2d_8 (Conv2D)              | (None, 7, 7, 128)  | 73856   |
| max_pooling2d_8 (MaxPooling2D) | (None, 4, 4, 128)  | 0       |
| flatten_2 (Flatten)            | (None, 2048)       | 0       |
| dense_4 (Dense)                | (None, 256)        | 524544  |
| dropout_2 (Dropout)            | (None, 256)        | 0       |
| dense_5 (Dense)                | (None, 10)         | 2570    |

Total params: 619,786

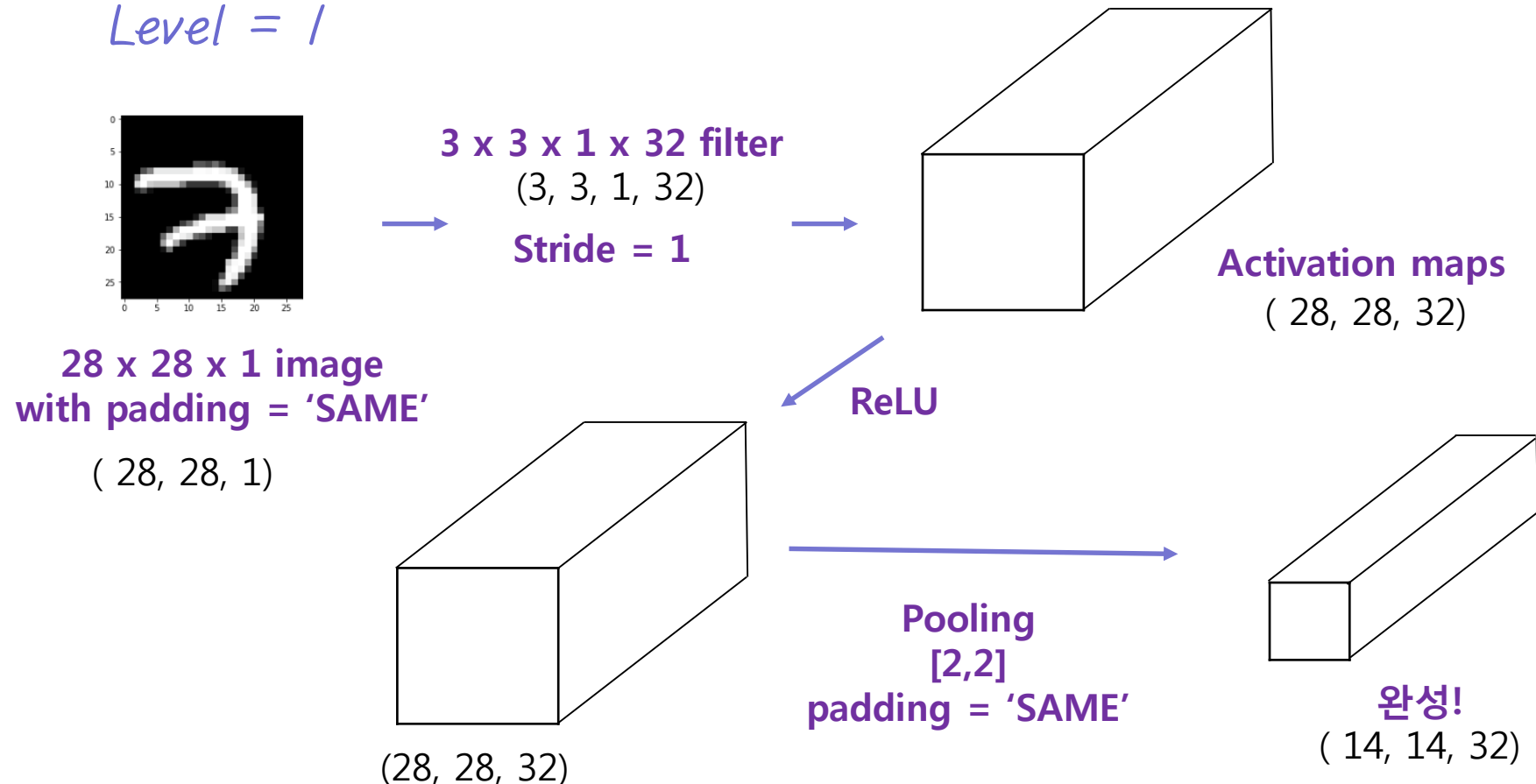
Trainable params: 619,786

Non-trainable params: 0

# MNIST with CNN

## ❖ 설계 (Convolution Layer)

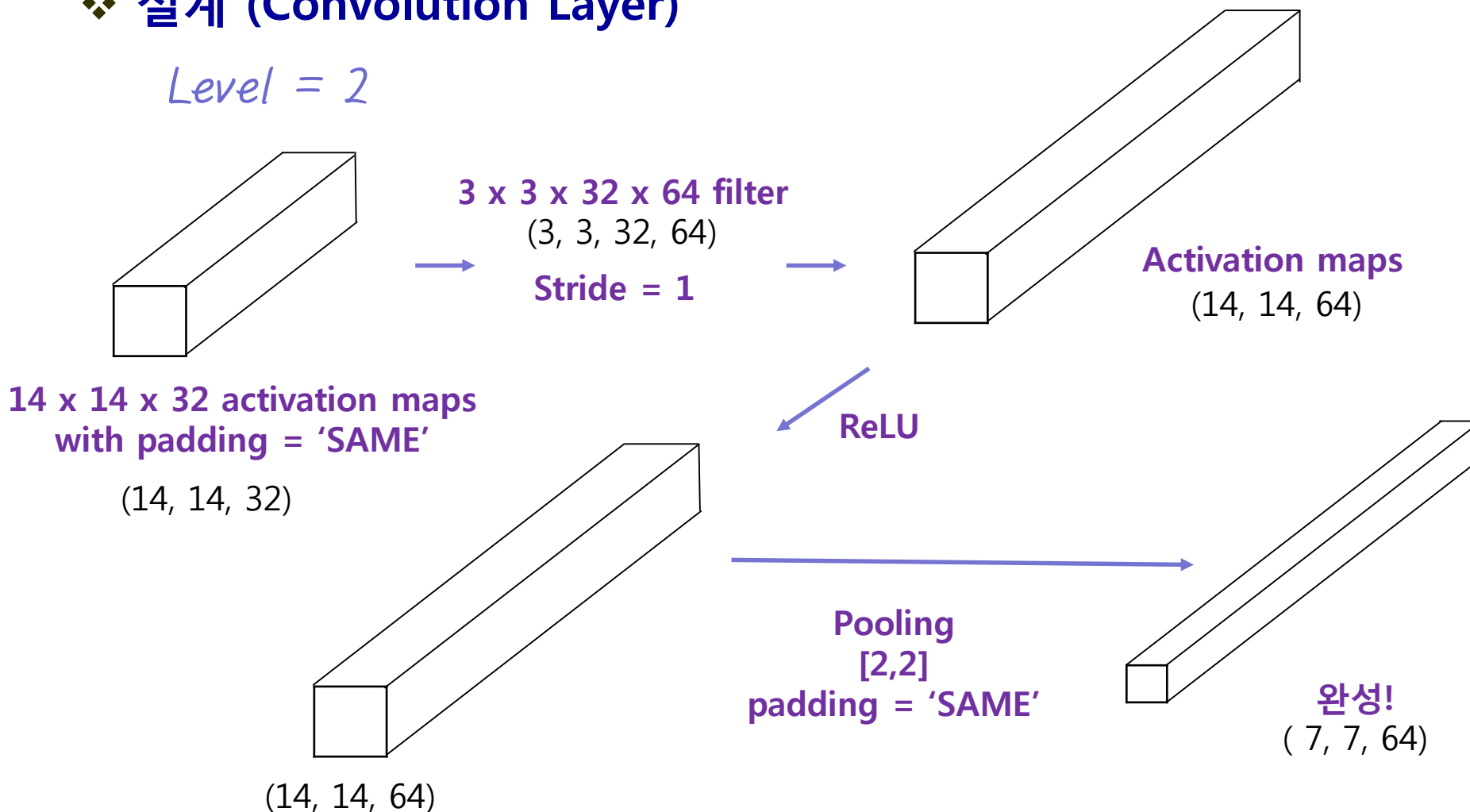
*Level = 1*



# MNIST with CNN

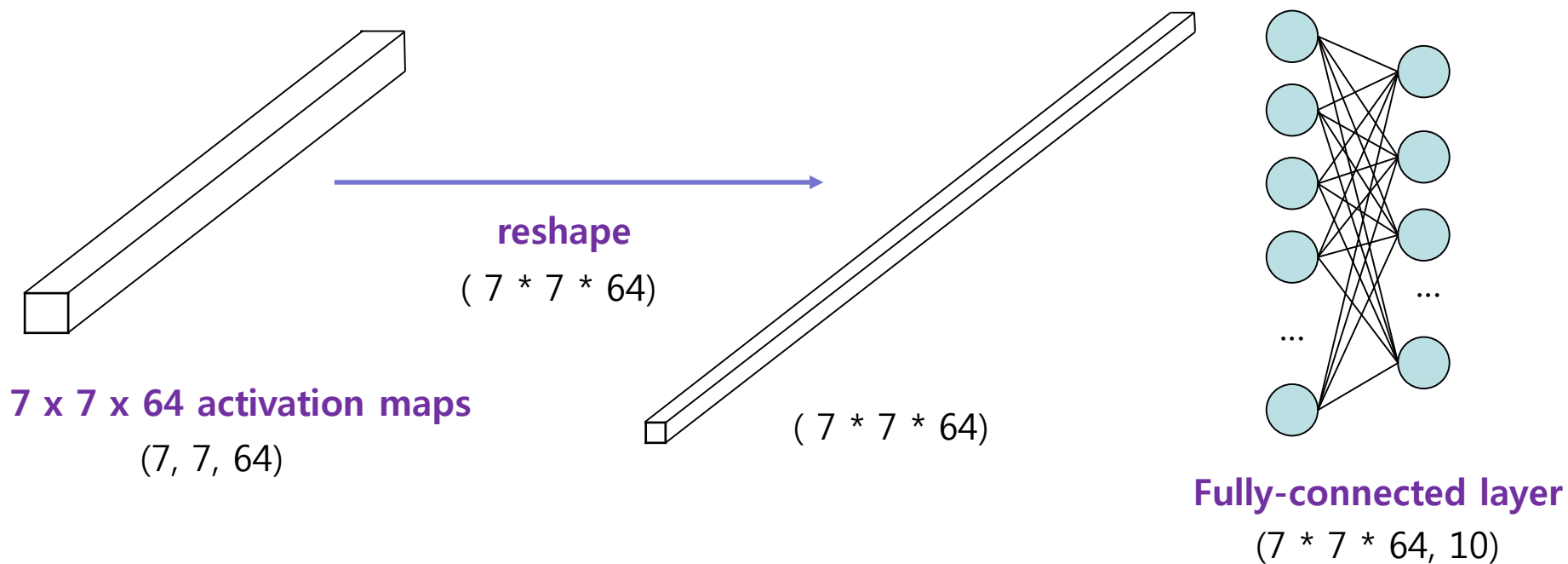
## ❖ 설계 (Convolution Layer)

*Level = 2*



# Selected Layer) MNIST with CNN

Level = 3



# MNIST with CNN

---

## ❖ Hyper Parameter

- Learning rate = 0.01
- epoch = 15
- batch size = 100

## ❖ Cost

- Cross entropy
  - $\text{logits} = \text{L2}(\text{fully-connected}) * W3 + b, \text{label} = Y$

## ❖ Optimizer

- Adam

# MNIST with CNN

---

## ❖ Hyper Parameter

- Learning rate = 0.01
- epoch = 15
- batch size = 100

## ❖ Cost

- Cross entropy
  - $\text{logits} = \text{L2}(\text{fully-connected}) * W3 + b, \text{label} = Y$

## ❖ Optimizer

- Adam



# MNIST with CNN

## Loss Function

```
[ ] @tf.function
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.keras.losses.categorical_crossentropy(
        y_pred=logits, y_true=labels, from_logits=True))
    return loss
```

## Calculating Gradient

```
[ ] @tf.function
def grad(model, images, labels):
    with tf.GradientTape() as tape:
        loss = loss_fn(model, images, labels)
    return tape.gradient(loss, model.variables)
```

# MNIST with CNN

---

## Optimizer

```
[ ]
```

```
[ ] optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```

## Training

```
[ ] @tf.function  
    def train(model, images, labels):  
        grads = grad(model, images, labels)  
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

# MNIST with CNN




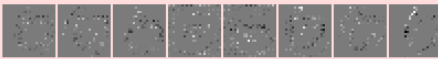

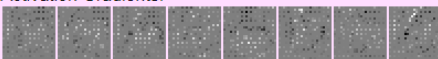




```
[ ] # train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_loss = 0.
    avg_train_acc = 0.
    avg_test_acc = 0.
    train_step = 0
    test_step = 0

    for images, labels in train_dataset:
        train(model, images, labels)
        #grads = grad(model, images, labels)
        #optimizer.apply_gradients(zip(grads, model.variables))
        loss = loss_fn(model, images, labels)
        acc = evaluate(model, images, labels)
        avg_loss = avg_loss + loss
        avg_train_acc = avg_train_acc + acc
        train_step += 1
    avg_loss = avg_loss / train_step
    avg_train_acc = avg_train_acc / train_step
```

# MNIST with CNN

## ConvNetJS MNIST demo

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>

| Network Visualization  |   |
|--|---|
| <p>input (24x24x1)<br/>max activation: 1, min: 0<br/>max gradient: 0.00036, min: -0.00041</p>  | <p>Activations:<br/></p> <p>Activation Gradients:<br/></p>  |
| <p>conv (24x24x8)<br/>filter size 5x5x1, stride 1<br/>max activation: 3.34765, min: -3.62583<br/>max gradient: 0.00013, min: -0.00014<br/>parameters: <math>8 \times 5 \times 5 \times 1 + 8 = 208</math></p>      | <p>Activations:<br/></p> <p>Activation Gradients:<br/></p> <p>Weights:<br/>(7)(8)(9)(0)(1)(2)(3)(4)(5)(6)</p> <p>Weight Gradients:<br/>(7)(8)(9)(0)(1)(2)(3)(4)(5)(6)</p> |
| <p>relu (24x24x8)<br/>max activation: 3.34765, min: 0<br/>max gradient: 0.00013, min: -0.00014</p>   | <p>Activations:<br/></p> <p>Activation Gradients:<br/></p>   |
| <p>pool (12x12x8)<br/>pooling size 2x2, stride 2<br/>max activation: 3.34765, min: 0<br/>max gradient: 0.00013, min: -0.00014</p>  | <p>Activations:<br/></p> <p>Activation Gradients:<br/></p>  |
| <p>conv (12x12x16)<br/>filter size 5x5x8, stride 1<br/>max activation: 7.95619, min: -13.09022<br/>max gradient: 0.00014, min: -0.00013<br/>parameters: <math>16 \times 5 \times 5 \times 8 + 16 = 3216</math></p> | <p>Activations:<br/></p> <p>Activation Gradients:<br/></p>  |



**1**

## **5. Keras**

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 1. tf.keras를 임포트하여 텐서플로 프로그램을 시작

```
import tensorflow as tf

from tensorflow import keras
```

### 2. 간단한 모델 만들기 (Sequential 모델)

```
from tensorflow.keras import layers

model = tf.keras.Sequential()
# 64개의 유닛을 가진 완전 연결 층을 모델에 추가합니다:
model.add(layers.Dense(64, activation='relu'))
# 또 하나를 추가합니다:
model.add(layers.Dense(64, activation='relu'))
# 10개의 출력 유닛을 가진 소프트맥스 층을 추가합니다:
model.add(layers.Dense(10, activation='softmax'))
```

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 3. 층(Layer) 설정

```
# 시그모이드 활성화 층을 만듭니다:
layers.Dense(64, activation='sigmoid')
# 또는 다음도 가능합니다:
layers.Dense(64, activation=tf.keras.activations.sigmoid)

# 커널 행렬에 L1 규제가 적용된 선형 활성화 층. 하이퍼파라미터 0.01은 규제의 양을 조절합니다:
layers.Dense(64, kernel_regularizer=tf.keras.regularizers.l1(0.01))

# 절편 벡터에 L2 규제가 적용된 선형 활성화 층. 하이퍼파라미터 0.01은 규제의 양을 조절합니다:
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# 커널을 랜덤한 직교 행렬로 초기화한 선형 활성화 층:
layers.Dense(64, kernel_initializer='orthogonal')

# 절편 벡터를 상수 2.0으로 설정한 선형 활성화 층:
layers.Dense(64, bias_initializer=tf.keras.initializers.Constant(2.0))
```

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 4. 학습과 평가

#### 1) 학습준비

(1)신경망 모델을 구성한 후 **compile** 메서드를 호출하여 학습 과정을 설정

```
model = tf.keras.Sequential([  
    # 64개의 유닛을 가진 완전 연결 층을 모델에 추가합니다:  
    layers.Dense(64, activation='relu', input_shape=(32,)),  
    # 또 하나를 추가합니다:  
    layers.Dense(64, activation='relu'),  
    # 10개의 출력 유닛을 가진 소프트맥스 층을 추가합니다:  
    layers.Dense(10, activation='softmax')])  
  
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```



# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 4. 학습과 평가

#### 1) 학습준비

(2)회귀모델, 분류모델을 구성한 후 compile 메서드를 호출하여 학습 과정을 설정

```
# 평균 제곱 오차로 회귀 모델을 설정합니다.
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='mse',          # 평균 제곱 오차
              metrics=['mae'])     # 평균 절댓값 오차

# 크로스엔트로피 손실 함수로 분류 모델을 설정합니다.
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.01),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 4. 학습과 평가

#### 2) 학습

##### (1)numpy를 사용한 학습

```
import numpy as np

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.fit(data, labels, epochs=10, batch_size=32)
```

##### tf.keras.Model.fit 매개변수

-epochs: 학습은 에포크(epoch)로 구성, 한 에포크는 학습 입력 데이터를 한번 순회(작은 배치로 나누어 수행).

-batch\_size: 학습데이터를 작은 배치로 나누고 학습 과정에서 이 배치를 순회

-validation\_data: 모델의 프로토타입 (prototype)을 만들 때 검증 데이터 (validation data)에서 간편하게 성능을 모니터

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 4. 학습과 평가

#### 2) 학습

##### (2) tf.data 데이터셋을 사용한 훈련

```
# 예제 `Dataset` 객체를 만듭니다:  
dataset = tf.data.Dataset.from_tensor_slices((data, labels))  
dataset = dataset.batch(32)  
  
# Dataset에서 `fit` 메서드를 호출할 때 `steps_per_epoch` 설정을 잊지 마세요.  
model.fit(dataset, epochs=10, steps_per_epoch=30)
```

```
dataset = tf.data.Dataset.from_tensor_slices((data, labels))  
dataset = dataset.batch(32)  
  
val_dataset = tf.data.Dataset.from_tensor_slices((val_data, val_labels))  
val_dataset = val_dataset.batch(32)  
  
model.fit(dataset, epochs=10,  
          validation_data=val_dataset)
```

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 4. 학습과 평가

#### 3) 평가와 예측

- tf.keras.Model.evaluate와 tf.keras.Model.predict 메서드 사용
- 넘파이 배열이나 tf.data.Dataset을 사용

```
data = np.random.random((1000, 32))  
labels = np.random.random((1000, 10))  
  
model.evaluate(data, labels, batch_size=32)  
  
model.evaluate(dataset, steps=30)
```

```
result = model.predict(data, batch_size=32)  
print(result.shape)
```

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 5. 학습모델 저장과 로드

tf.keras.Model.save\_weights를 사용하여 모델의 가중치를 저장하고 복원

```
model = tf.keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(32,)),
    layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
# 가중치를 텐서플로의 체크포인트 파일로 저장합니다.
model.save_weights('./weights/my_model')

# 모델의 상태를 복원합니다.
# 모델의 구조가 동일해야 합니다.
model.load_weights('./weights/my_model')
```

# Keras

## ❖ Keras 개요 (<https://www.tensorflow.org>)

### 5. 학습모델 저장과 로드

tf.keras.Model.save\_weights를 사용하여 모델의 가중치를 저장하고 복원

```
# 간단한 모델을 만듭니다.
model = tf.keras.Sequential([
    layers.Dense(10, activation='softmax', input_shape=(32,)),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels, batch_size=32, epochs=5)

# 전체 모델을 HDF5 파일로 저장합니다.
model.save('my_model.h5')

# 가중치와 옵티마이저를 포함하여 정확히 같은 모델을 다시 만듭니다.
model = tf.keras.models.load_model('my_model.h5')
```

# Keras를 사용한 예

---

## ❖ 자동차 연비 예측하기(regression)

- <https://www.tensorflow.org/tutorials/keras/regression>

## ❖ 패션 MNIST 이미지 분류 (neural network)

- <https://www.tensorflow.org/tutorials/keras/classification>

## ❖ 개, 고양이 이미지 분류 (convolution neural network)

- <https://www.tensorflow.org/tutorials/images/classification>