

# AI 프로그래밍

- 12주차



인하 공전 컴퓨터 정보과  
민 정혜

- 케라스 딥러닝 예제 복습
- Open CV 실습
- 컨볼루션 신경망 (CNN: Convolutional Neural Network)

▼ 그림 14-1 | 학습셋, 테스트셋, 검증셋



학습 셋 : training set  
검증 셋: validation set  
테스트 셋: test set

## ● 데이터의 확인과 검증셋

- 학습이 끝난 모델을 테스트해 보는 것이 테스트셋의 목적이라면, 최적의 학습 파라미터를 찾기 위해 학습 과정에서 사용하는 것이 검증셋
- 검증셋을 설정하면 검증셋에 테스트한 결과를 추적하면서 최적의 모델을 만들 수 있음
- 검증셋은 `model.fit()` 함수 안에 `validation_split`이라는 옵션을 주면 만들어짐
- 그림 14-1과 같이 전체의 80%를 학습셋으로 만들고 이 중 25%를 검증셋으로 하면 학습셋:검증셋:테스트셋의 비율이 60:20:20이 됨

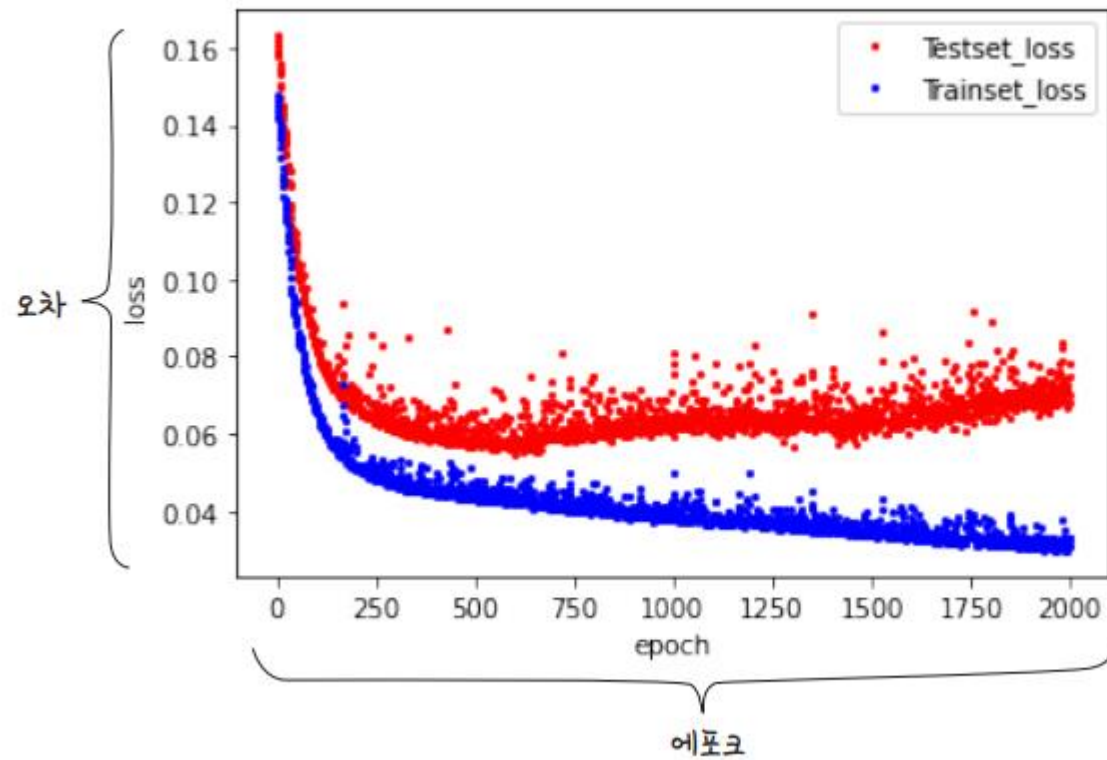
- 데이터의 확인과 검증셋

```
model.add(Dense(30, input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 모델을 실행합니다.
history = model.fit(X_train, y_train, epochs=50, batch_size=500,
validation_split=0.25) # 0.8 x 0.25 = 0.2
```

실행 결과



## ■ 그래프로 과적합 확인하기

- 그래프의 형태는 실행에 따라 조금씩 다를 수 있지만 대략 그림 14-2와 같은 그래프가 나옴
- 우리가 눈여겨보아야 할 부분은 학습이 오래 진행될수록 검증셋의 오차(파란색)는 줄어들이지만 테스트셋의 오차(빨간색)는 다시 커진다는 것
- 이는 과도한 학습으로 과적합이 발생했기 때문임
- 이러한 사실을 통해 알 수 있는 것은 검증셋 오차가 커지기 직 전까지 학습한 모델이 최적의 횟수로 학습한 모델이라는 것
- 이제 검증셋의 오차가 커지기 전에 학습을 자동으로 중단시키고, 그때의 모델을 저장하는 방법을 알아보자

- 학습의 자동 중단

- 텐서플로에 포함된 케라스 API는 EarlyStopping() 함수를 제공
- 학습이 진행되어도 테스트셋 오차가 줄어들지 않으면 학습을 자동으로 멈추게 하는 함수
- 이를 조금 전 배운 ModelCheckpoint() 함수와 함께 사용해 보면

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)
```



## ▪ 학습의 자동 중단

- monitor 옵션은 model.fit()의 실행 결과 중 어떤 것을 이용할지 정함
- 검증셋의 오차(val\_loss)로 지정
- patience 옵션은 지정된 값이 몇 번 이상 향상되지 않으면 학습을 종료시킬지 정함
- monitor='val\_loss', patience=20이라고 지정하면 검증셋의 오차가 20번 이상 낮아지지 않을 경우 학습을 종료하라는 의미

- 학습의 자동 중단

- 모델 저장에 관한 설정은 앞 절에서 사용한 내용을 그대로 따르겠음
- 다만 이번에는 최고의 모델 하나만 저장되게끔 해 보자

```
modelpath = "./data/model/Ch14-4-bestmodel.hdf5"

checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose = 0, save_best_only=True)
```

하

## 4 학습의 자동 중단

- 학습의 자동 중단
  - 모델을 실행
  - 자동으로 최적의 에포크를 찾아 멈출 예정이므로 epochs는 넉넉하게 설정

```
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,  
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,  
checkpointer])
```

## 4 학습의 자동 중단

- 학습의 자동 중단
  - 모델을 실행
  - 자동으로 최적의 에포크를 찾아 멈출 예정이므로 epochs는 넉넉하게 설정

```
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,  
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,  
checkpointer])
```

- 학습의 자동 중단

- 앞서 만든 기본 코드에 다음과 같이 새로운 코드를 불러와 덧붙

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# 학습이 언제 자동 중단될지 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)

# 최적화 모델이 저장될 폴더와 모델 이름을 정합니다.
modelpath = "./data/model/Ch14-4-bestmodel.hdf5"

# 최적화 모델을 업데이트하고 저장합니다.
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=0, save_best_only=True)
```

- 학습의 자동 중단

```
# 모델을 실행합니다.  
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,  
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,  
checkpointer])
```

```
1/8 [==> .....] - ETA: 0s - loss: 0.1462 - accuracy: 0.9540
Epoch 31: val_loss improved from 0.16765 to 0.16567, saving model to ./data/model/Ch14-4-bestmodel.hdf5
8/8 [=====] - 0s 9ms/step - loss: 0.1712 - accuracy: 0.9430 - val_loss: 0.1657 - val_accuracy: 0.9392
Epoch 32/2000
1/8 [==> .....] - ETA: 0s - loss: 0.2067 - accuracy: 0.9400
Epoch 32: val_loss did not improve from 0.16567
8/8 [=====] - 0s 7ms/step - loss: 0.1678 - accuracy: 0.9430 - val_loss: 0.1675 - val_accuracy: 0.9346
Epoch 33/2000
1/8 [==> .....] - ETA: 0s - loss: 0.1330 - accuracy: 0.9560
Epoch 33: val_loss did not improve from 0.16567
8/8 [=====] - 0s 6ms/step - loss: 0.1661 - accuracy: 0.9438 - val_loss: 0.1691 - val_accuracy: 0.9431
Epoch 34/2000
1/8 [==> .....] - ETA: 0s - loss: 0.1694 - accuracy: 0.9400
Epoch 34: val_loss improved from 0.16567 to 0.15903, saving model to ./data/model/Ch14-4-bestmodel.hdf5
8/8 [=====] - 0s 9ms/step - loss: 0.1694 - accuracy: 0.9418 - val_loss: 0.1590 - val_accuracy: 0.9362
Epoch 35/2000
1/8 [==> .....] - ETA: 0s - loss: 0.1949 - accuracy: 0.9380
Epoch 35: val_loss improved from 0.15903 to 0.15821, saving model to ./data/model/Ch14-4-bestmodel.hdf5
8/8 [=====] - 0s 9ms/step - loss: 0.1628 - accuracy: 0.9464 - val_loss: 0.1582 - val_accuracy: 0.9369
Epoch 36/2000
1/8 [==> .....] - ETA: 0s - loss: 0.1485 - accuracy: 0.9360
```

# 케라스 신경망 실습 - 주택 가격 예측

인하공전 컴퓨터 정보과

## 실행 결과

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...
...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...

PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000
...	...	...	...	...	...	...	...	...	...
0	NaN	NaN	NaN	0	8	2007	WD	Normal	175000
0	NaN	MnPrv	NaN	0	2	2010	WD	Normal	210000
0	NaN	GdPrv	Shed	2500	5	2010	WD	Normal	266500
0	NaN	NaN	NaN	0	4	2010	WD	Normal	142125
0	NaN	NaN	NaN	0	6	2008	WD	Normal	147500

1460 rows × 81 columns



- 데이터 파악하기

- 총 80개의 속성으로 이루어져 있고 마지막 열이 우리의 타깃인 집 값(SalePrice)
- 모두 1,460개의 샘플이 들어 있음

## ■ 주택 가격 예측 모델

- 모델의 구조와 실행 옵션을 설정
- 입력될 속성의 개수를 X\_train.shape[1]로 지정해 자동으로 세도록 했음

```
model = Sequential()  
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))  
model.add(Dense(30, activation='relu'))  
model.add(Dense(40, activation='relu'))  
model.add(Dense(1))  
model.summary()
```



activation

.

- 주택 가격 예측 모델

- 실행에서 달라진 점은 손실 함수
- 선형 회귀이므로 평균 제곱 오차(mean\_squared\_error)를 적음

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(2,)))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

입력 층 : 2  
은닉층 1 : 16 ( relu)  
은닉층 2 : 8 (relu)  
은닉층 3 : 8 (relu)  
출력 층 : 1

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
=====		
dense_6 (Dense)	(None, 16)	48
dense_7 (Dense)	(None, 8)	136
dense_8 (Dense)	(None, 8)	72
dense_9 (Dense)	(None, 1)	9

=====

Total params: 265  
Trainable params: 265  
Non-trainable params: 0

```

model.add(Dense(30, input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

```

입력 층 : 12  
 은닉층 1 : 30 ( relu)  
 은닉층 2 : 12 (relu)  
 은닉층 3 : 8 (relu)  
 출력 층 : 1 (sigmoid)

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 30)	390
dense_1 (Dense)	(None, 12)	372
dense_2 (Dense)	(None, 8)	104
dense_3 (Dense)	(None, 1)	9

```

model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()

```

입력 층 : 28 * 28
은닉층 1 : 128 ( relu)
출력 층 : 10

Layer (type)	Output Shape	Param #
=====		
=====		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290
=====		
=====		

```
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

입력 층 : 28 \* 28  
은닉층 1 : 32 ( relu)  
은닉층 2 : 16 ( relu)  
출력 층 : 10

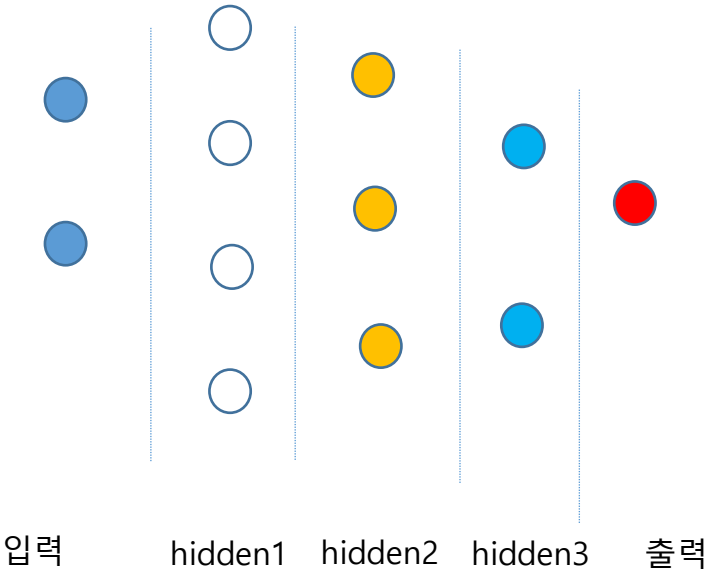
Layer (type)	Output Shape	Param #
=====		
flatten_1 (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 32)	25120
dense_3 (Dense)	(None, 16)	528
dense_4 (Dense)	(None, 10)	170

```
model = tf.keras.models.Sequential()  
model.add(tf.keras.layers.Dense(4, activation='relu', input_shape=(2,)))  
model.add(tf.keras.layers.Dense(3, activation='relu'))  
model.add(tf.keras.layers.Dense(2, activation='relu'))  
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

입력 층 : 2  
은닉층 1 : 4 ( relu)  
은닉층 2 : 3 ( relu)  
은닉층 3 : 2 ( relu)  
출력 층 : 1

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 4)	12
dense_5 (Dense)	(None, 3)	15
dense_6 (Dense)	(None, 2)	8
dense_7 (Dense)	(None, 1)	3

total params: 38  
Trainable params: 38  
Non-trainable params: 0

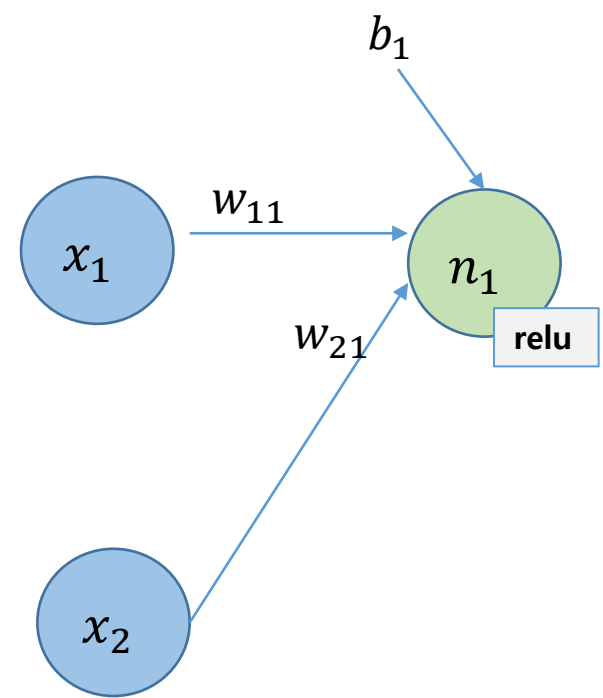
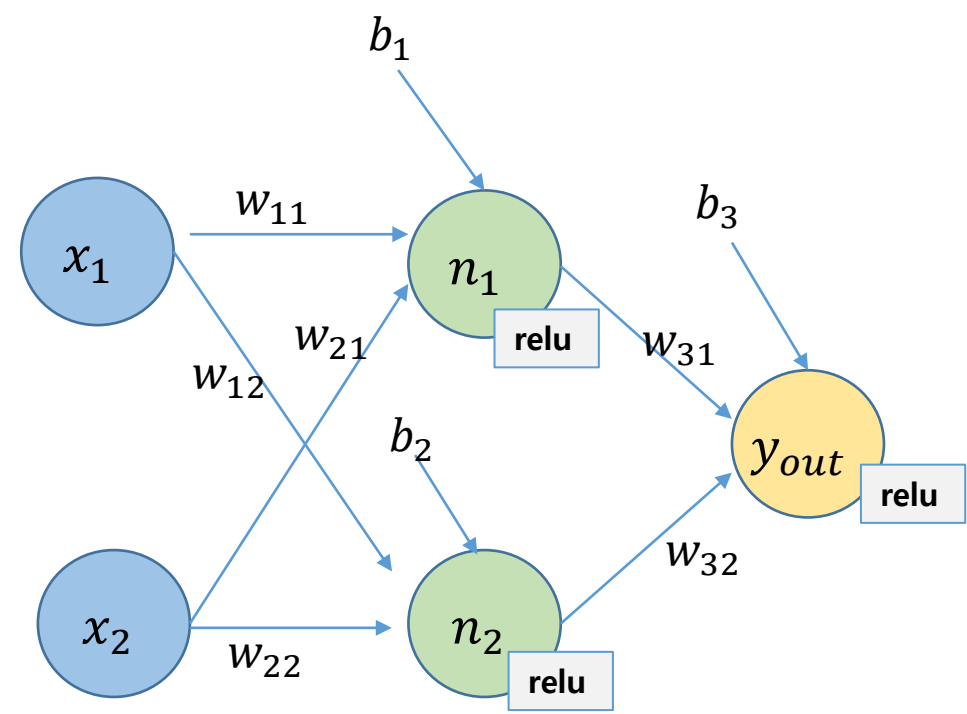


Parameter 수 = 앞 layer node수\* 현재 layer node수 + 현재 layer node 수



# 다중 퍼셉트론 (Multi Layer Perceptron: MLP)

입력 (1,0)



과제 (house.ipynb, house\_train.csv )

1) 아래와 같이 모델과 학습 방법을 변경 한 후 모델을 house1.hdf5로 저장 하시오.

입력 층 : 변동 없음  
은닉층 1 : 20 ( relu)  
은닉층 2 : 20 (relu)  
은닉층 3 : 40 (relu)  
출력 층 변동 없음.

17회 이상 결과가 향상 되지 않으면 자동으로 학습이 중단되게  
Batch size 16

2) 아래와 같이 모델과 학습 방법을 변경 한 후 모델을 house2.hdf5로 저장 하시오.

입력 층 : 변동 없음  
은닉층 1 : 10 ( relu)  
은닉층 2 : 20 (relu)  
은닉층 3 : 20 (relu)  
출력 층 변동 없음.



15회 이상 결과가 향상 되지 않으면 자동으로 학습이 중단되게  
Batch size 20

code (.ppt에 직접 붙여도 되고 .py 파일로 제출 가능) 와 model 제출 (house1.hdf5, hoise2.hdf5) 제출

- 저장된 모델로 확인
- 과제로 업로드한 house1.hdf5, house2.hdf5를 load 하여

test loss 확인

house\_model\_test.ipynb

 house1.hdf5	2022-05-14 오후 ...	HDF5 파일	69KB
 house2.hdf5	2022-05-14 오후 ...	HDF5 파일	61KB

```
X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_size=0.2, shuffle=False)
```

```
model=load_model ('/content/house1.hdf5')  
print('테스트 loss1', model.evaluate(X_test,y_test)/10000000)
```

```
model=load_model ('/content/house2.hdf5')  
print('테스트 loss2', model.evaluate(X_test,y_test)/10000000)
```

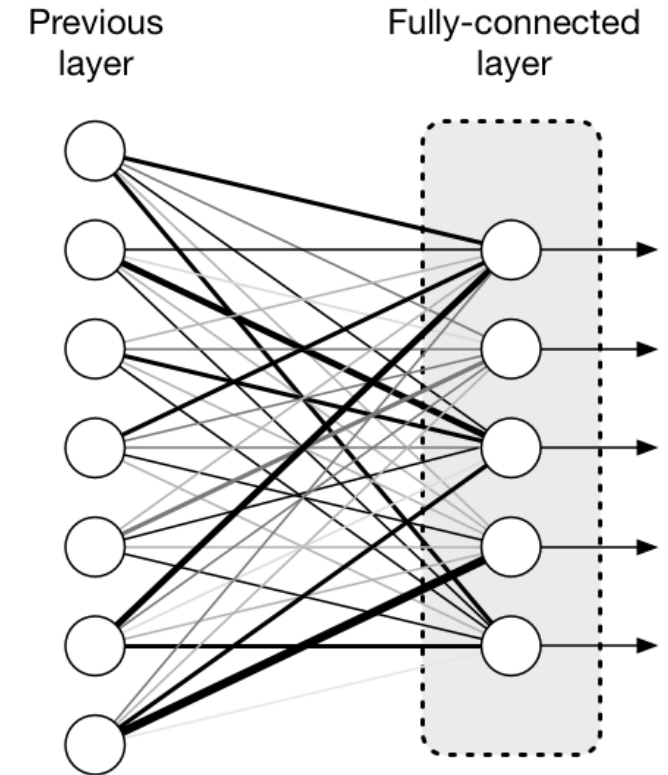
# 완전 연결 신경망 (Fully Connected Neural Network)

인하공전 컴퓨터 정보과

`tf.keras.layers.Dense`

완전 연결 계층 : (Fully Connected Layer, Densely connected layer)

한 층 (layer)의 모든 뉴런이 그 다음 층의 모든 뉴런과 연결된 상태

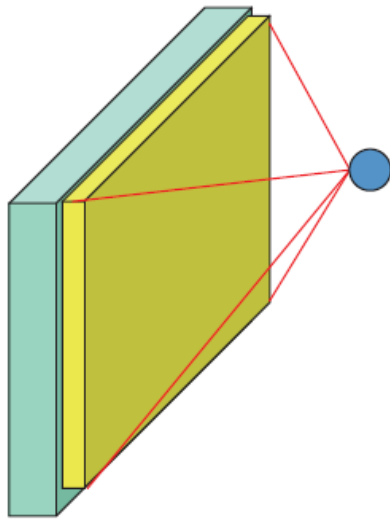


[https://keras.io/api/layers/core\\_layers/](https://keras.io/api/layers/core_layers/)

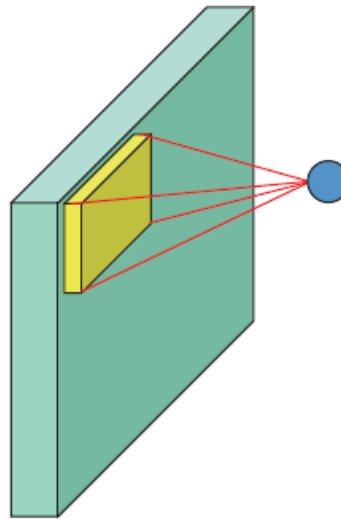
# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- 컨볼루션(Convolution Neural Network: CNN) 신경망에서는 하위 레이어의 노드들과 상위 레이어의 노드들이 부분적으로만 연결되어 있다.



(a) 완전연결 신경망

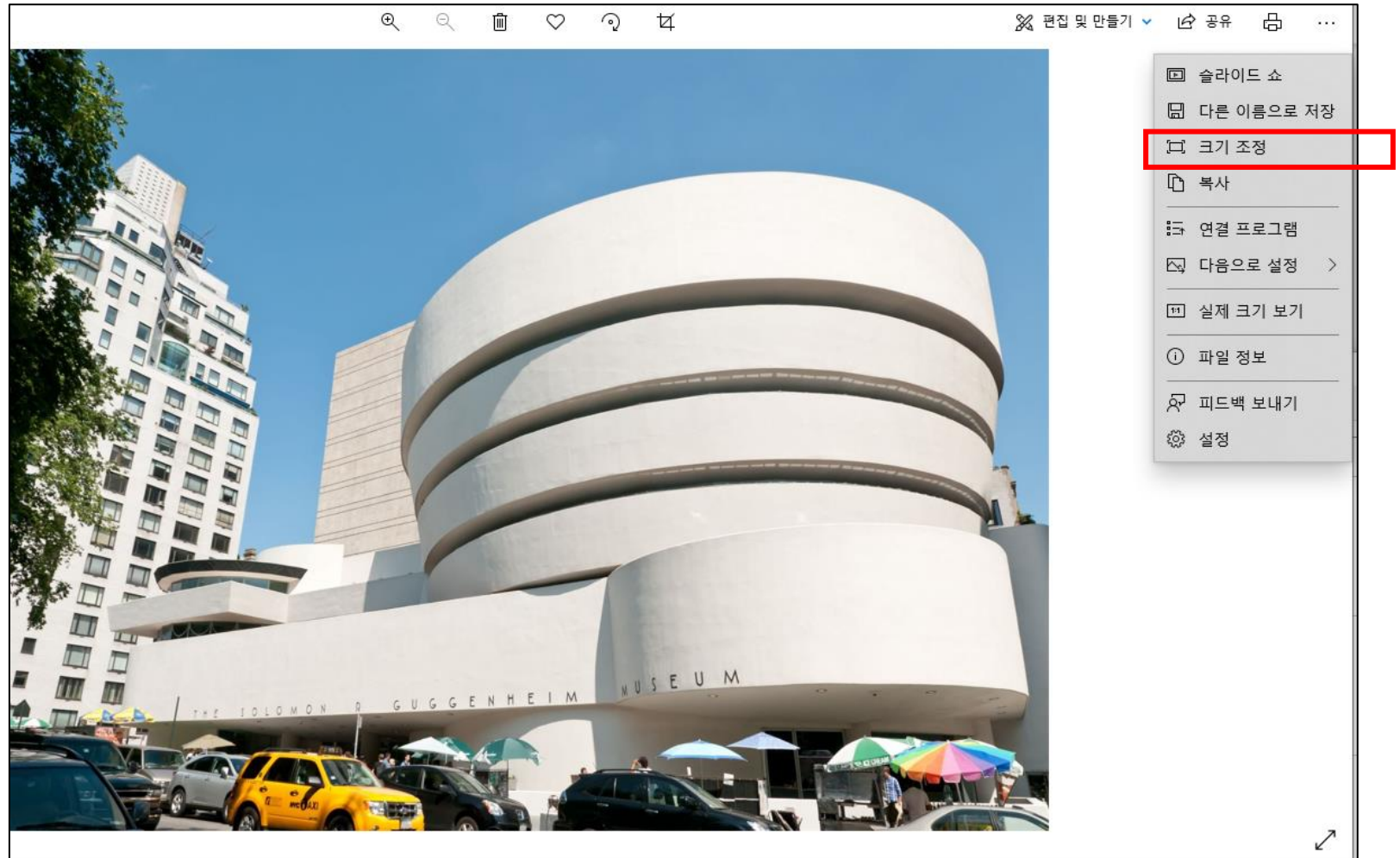


(b) 컨볼루션 신경망

# 영상 처리 기초 – Open CV 실습

본인이 찍어온 사진 or test1\_1920.jpg 사용

본인이 찍어온 사진 크기 조정 => 1920x1440 과 비슷한 크기로



# 영상 처리 기초 – Open CV 실습

Colab에서 실습

```
import cv2
from google.colab.patches import cv2_imshow
img=cv2.imread('/content/test1_1920.jpg')
print('img.ndim =',img.ndim)
print('img.shape =',img.shape)
print('img.dtype = ',img.dtype)
#cv2_imshow(img)
img=cv2.resize(img, (0,0),fx=0.5,fy=0.5)
cv2.imwrite('test1_half.jpg',img)
print(img.shape)
cv2_imshow(img)
```

jupyter

cv2.imshow(img)



# 영상 처리 기초 – Open CV 실습

Colab에서 실습

#colab

```
cv2_imshow(img)
```

#jupyter

```
cv2.imshow(img)
```

# 영상 처리 기초 – Open CV 실습

```
img=cv2.imread('/content/test1_half.jpg')
img[100:300,100:300,:]=255
img[100:300,300:500,:]=0
img[100:300,500:700,:]=127

img[300:500,100:300,0]=255
img[300:500,300:500,1]=255
img[300:500,500:700,2]=255

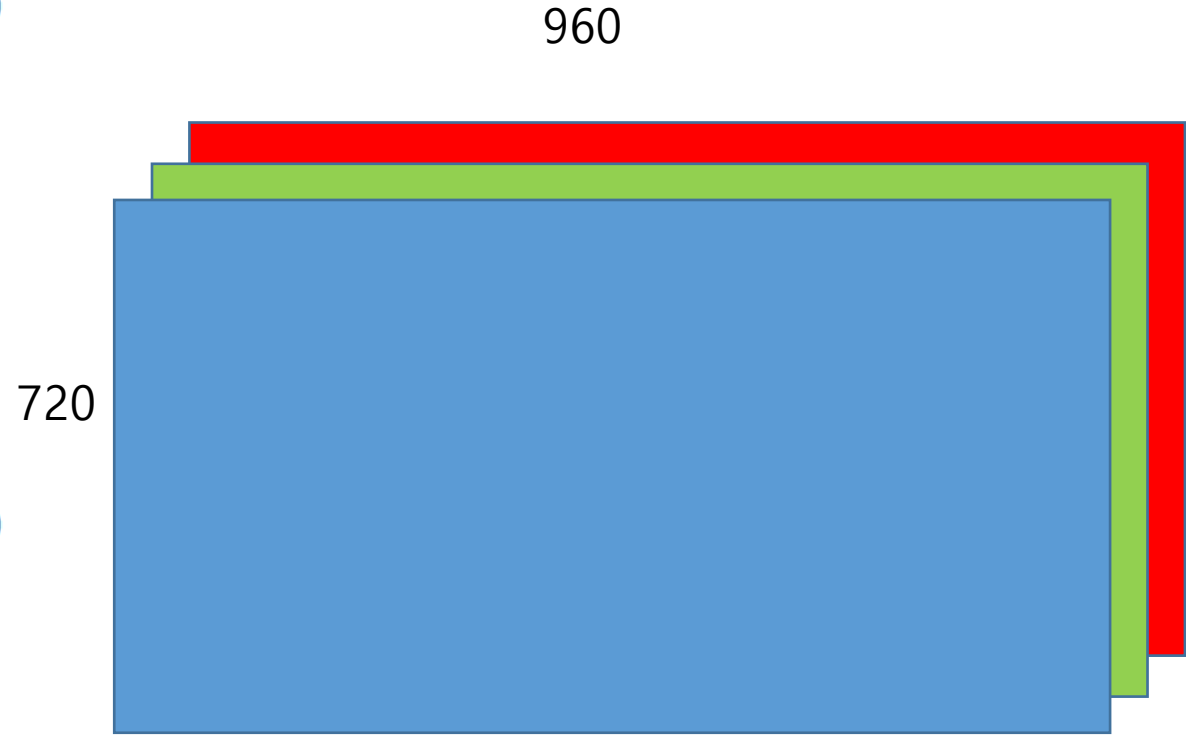
cv2.imwrite('test1_half_changed.jpg',img)

cv2.imshow(img)
```

White= (255,255,255)

Black = (0,0,0)

Gray =(127,127,127)



# 영상 처리 기초 – Open CV 실습

```
img=cv2.imread('/content/test1_half.jpg')
gimg=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

cv2.imwrite('test1_halfgray.jpg',gimg)
print(gimg.shape)
cv2_imshow(gimg)
```

# 영상 처리 기초 - Open CV 실습

Convolution(컨벌루션)은 주변 화소값들에 가중치를 곱해서 더한 후에 이것을 새로운 화소값으로 하는 연산이다

$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$

## 커널

$$\begin{aligned} & 1/9 * 3 + 1/9 * 6 + 1/9 * 6 + \\ & 1/9 * 3 + 1/9 * 4 + 1/9 * 3 + \\ & 1/9 * 5 + 1/9 * 7 + 1/9 * 7 = 4.88 \end{aligned}$$

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

# 영상 처리 기초 – Open CV 실습

$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$

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

입력

$$\begin{aligned} & 1/9 * 6 + 1/9 * 6 + 1/9 * 4 + \\ & 1/9 * 4 + 1/9 * 3 + 1/9 * 8 + \\ & 1/9 * 7 + 1/9 * 7 + 1/9 * 7 = 5.77 \end{aligned}$$

[illegible]

# 영상 처리 기초 – Open CV 실습

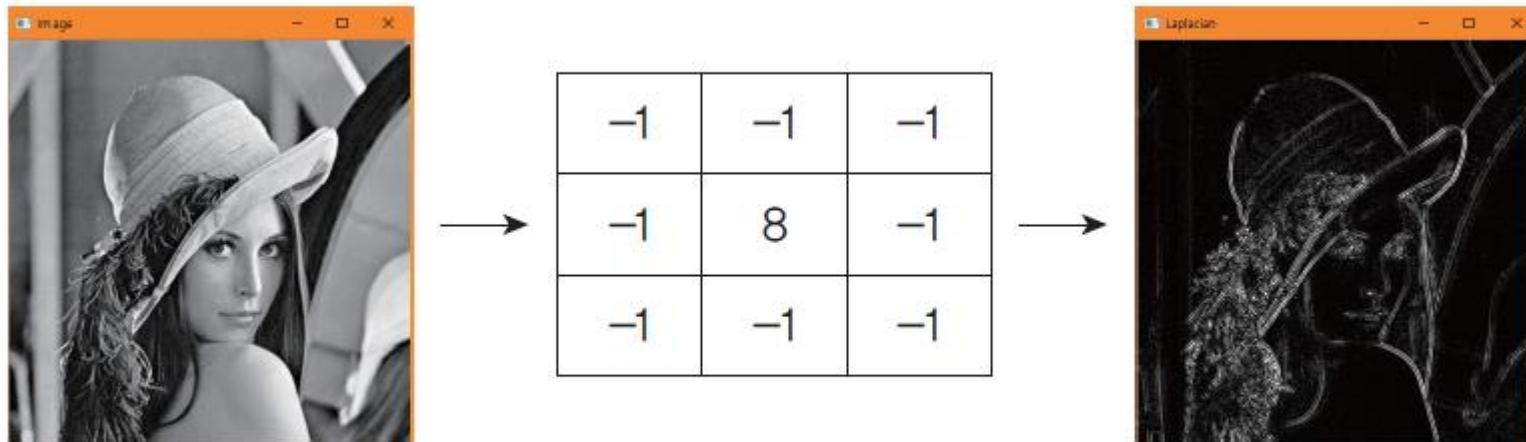


그림 9-9 | 영상 처리에서의 컨벌루션 연산

# 영상 처리 기초 – Open CV 실습

```
import numpy as np
gimg=cv2.imread('test1_halfgray.jpg')
kernel1=np.ones((3,3),dtype=np.float64)/9.
gimg1=cv2.filter2D(gimg,-1,kernel1)
kernel2=np.ones((5,5),dtype=np.float64)/25.
gimg2=cv2.filter2D(gimg,-1,kernel2)
kernel3=np.ones((7,7),dtype=np.float64)/49.
gimg3=cv2.filter2D(gimg,-1,kernel3)
cv2_imshow(gimg1)
cv2_imshow(gimg2)
cv2_imshow(gimg3)
cv2.imwrite('test_halfgray_filt1.jpg',gimg1)
cv2.imwrite('test_halfgray_filt2.jpg',gimg2)
cv2.imwrite('test_halfgray_filt3.jpg',gimg3)
```

# 영상 처리 기초 – Open CV 실습

```
kernel1 [[0.11 0.11 0.11]
[0.11 0.11 0.11]
[0.11 0.11 0.11]]
```

```
kernel2 [[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]]
```

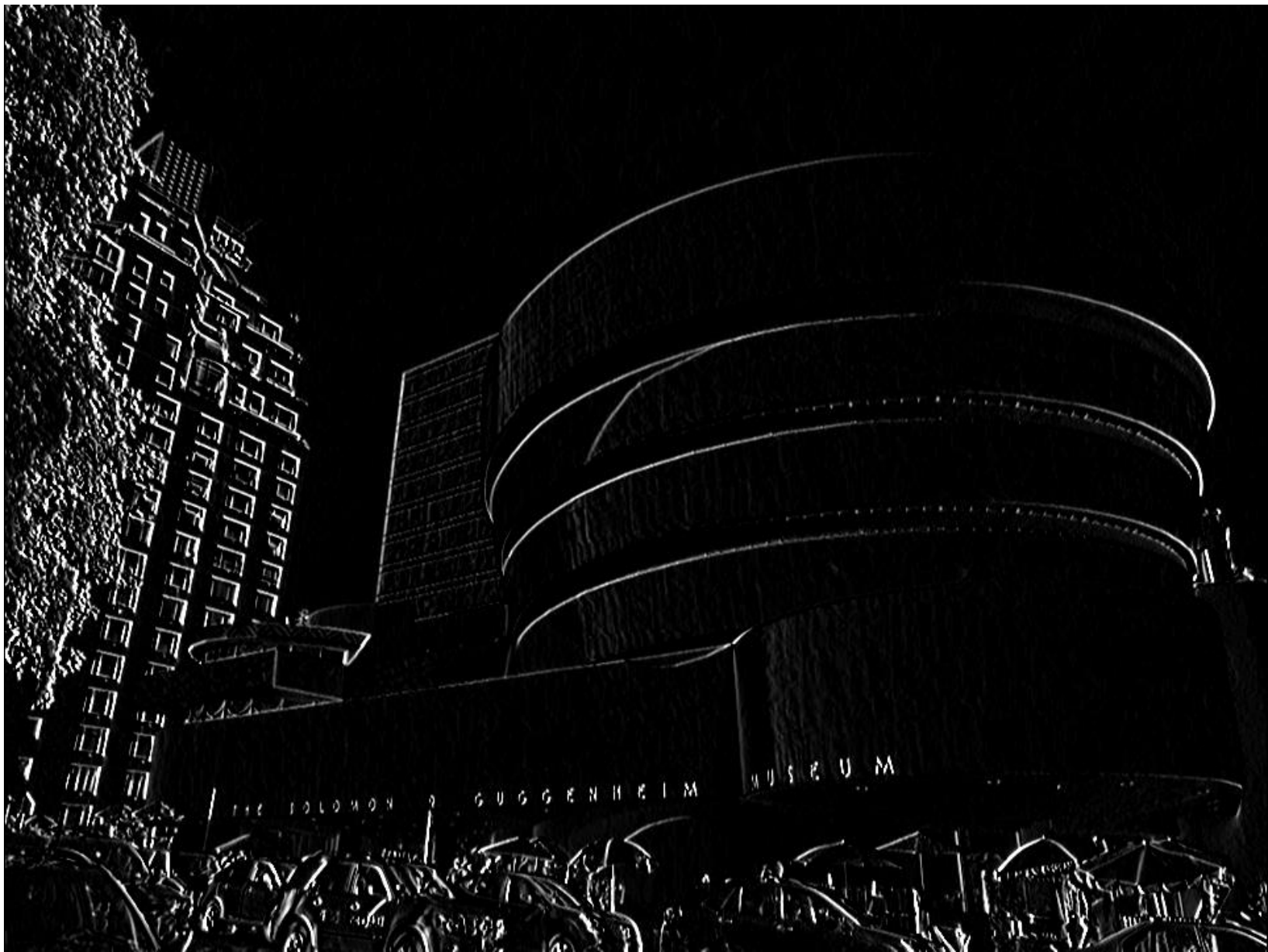
```
kernel3 [[0.02 0.02 0.02 0.02 0.02 0.02 0.02]
[0.02 0.02 0.02 0.02 0.02 0.02 0.02]
[0.02 0.02 0.02 0.02 0.02 0.02 0.02]
[0.02 0.02 0.02 0.02 0.02 0.02 0.02]
[0.02 0.02 0.02 0.02 0.02 0.02 0.02]
[0.02 0.02 0.02 0.02 0.02 0.02 0.02]]
```



# 영상 처리 기초 – Open CV 실습

```
import numpy as np
gimg=cv2.imread('test1_halfgray.jpg')
kernel_x = np.array([[ -1,0,1], [-2,0,2], [-1,0,1]])
gimge=cv2.filter2D(gimg,-1,kernel_x)
cv2_imshow(gimge)
cv2.imwrite('test_halfgray_filte.jpg',gimge)
```

# 영상 처리 기초 – Open CV 실습



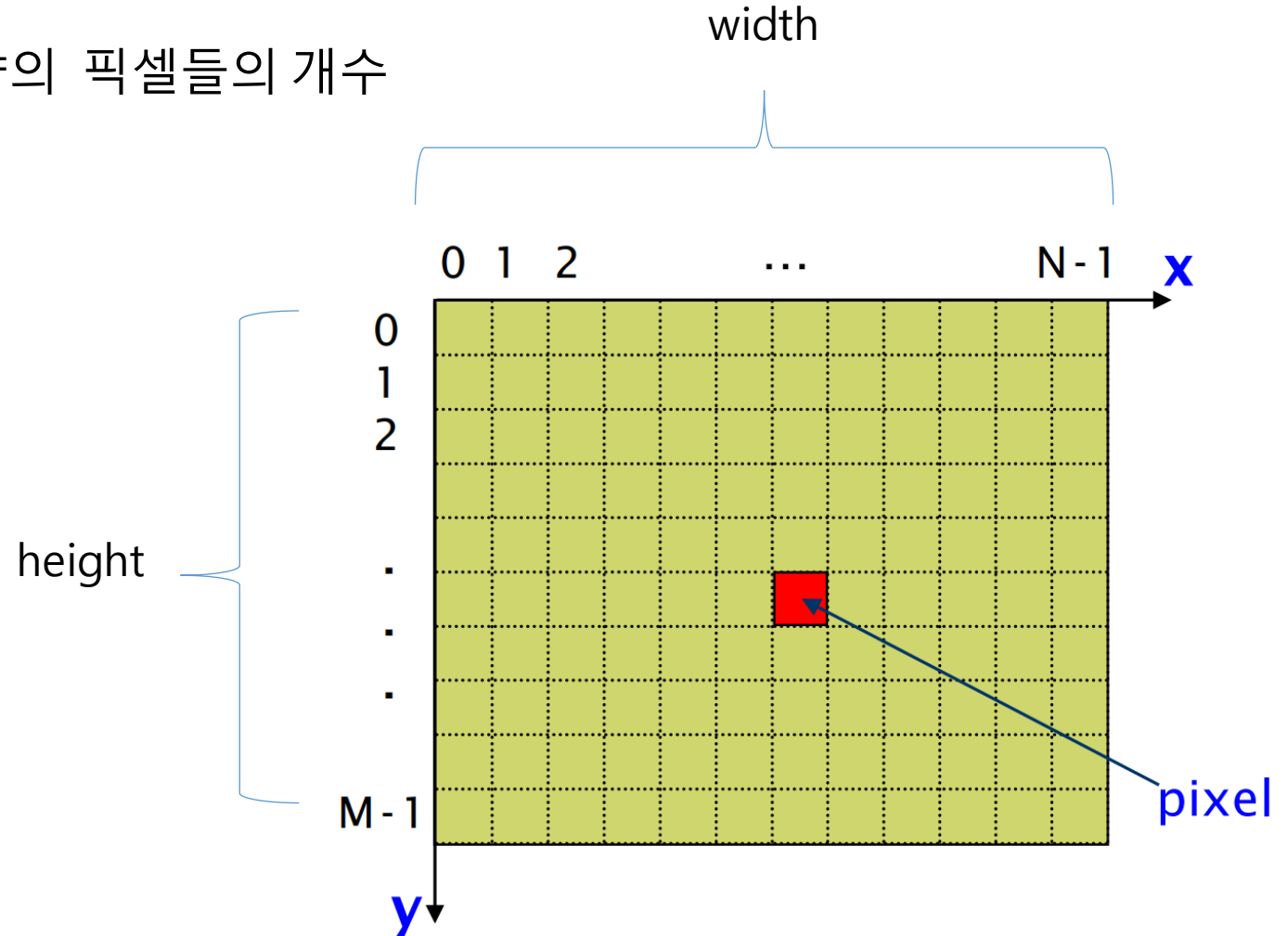
# 영상 처리 기초 – Open CV 실습

과제 : 아래 파일 upload

- 1) `test1_half.jpg`
- 2) `test1_half_changed.jpg`
- 3) `test1_halfgray.jpg`
- 4) `test_halfgray_filt1.jpg`
- 5) `test_halfgray_filt2.jpg`
- 6) `test_halfgray_filt3.jpg`
- 7) `test_halfgray_filte.jpg`

# 영상 처리 기초

- 영상은 픽셀로 이루어짐
- 해상도 (Resolution)
  - 2차원 공간 영역에서,  $x, y$  축 방향의 픽셀들의 개수



# 영상 처리 기초

- 영상은 픽셀로 이루어짐.

- RGB color space**

- Red, Green, Blue

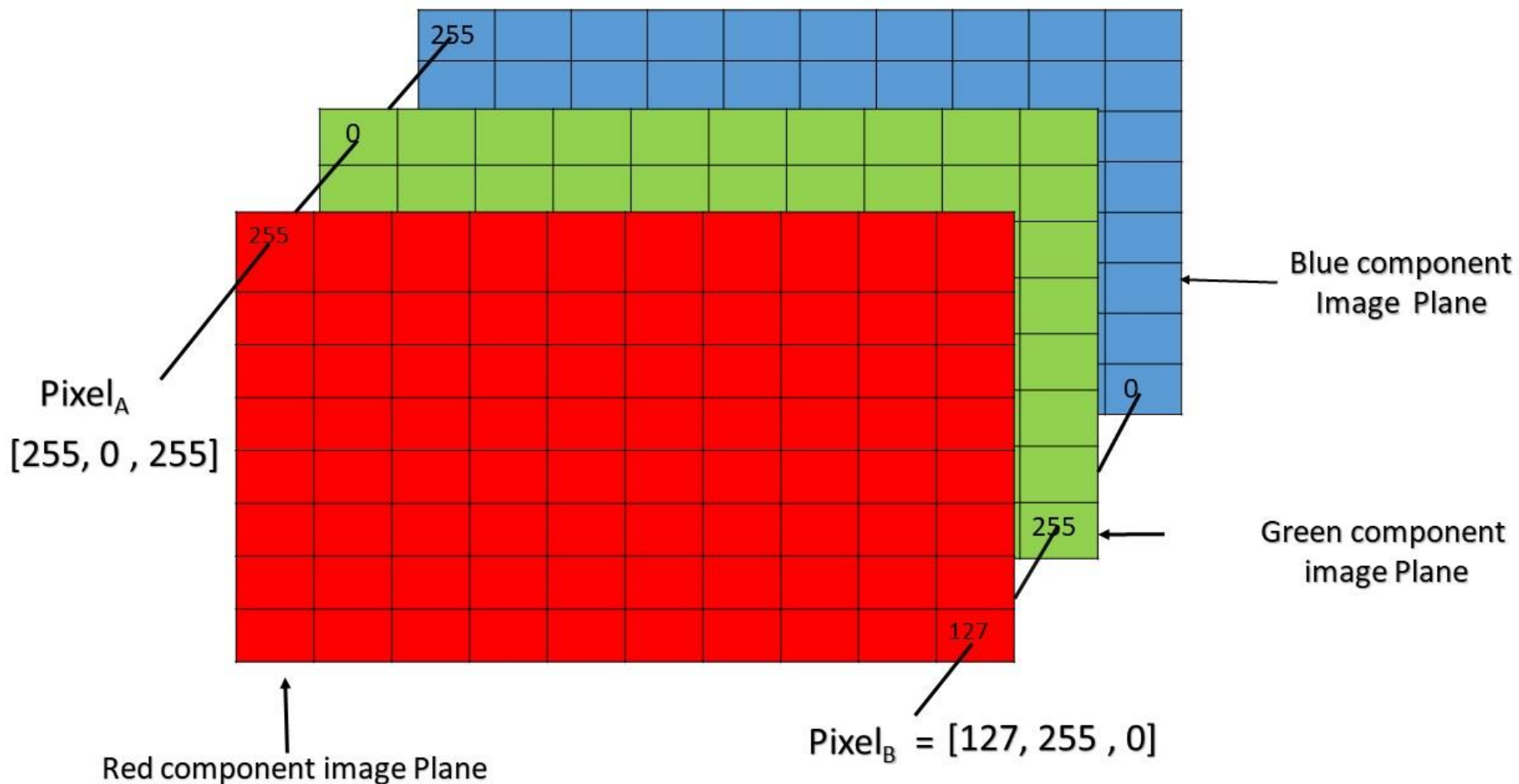
White = (255,255,255)

Black = (0,0,0)

Gray = (127,127,127)

Img.shape

=(height, width, rgb==3)



Pixel of an RGB image are formed from the corresponding pixel of the three component images

# 영상 처리 기초



컬러 이미지

`Img.shape`

`=(height, width, rgb==3)`



그레이 이미지

`Img.shape`

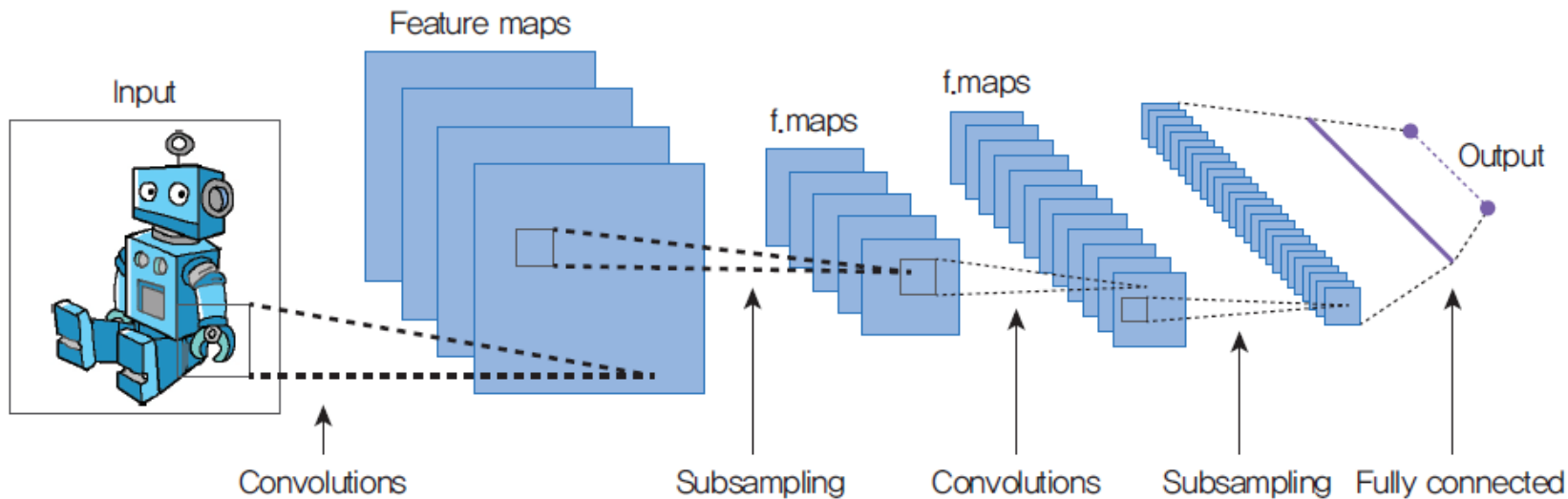
`=(height, width)`

$$\text{Gray} = 0.299 * R + 0.587 * G + 0.114 * B$$

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- 컨볼루션 신경망은 모든 신경망 구조 중에서 가장 강력한 성능을 보여주는 신경망 중의 하나이다.
- 컨볼루션 신경망은 2차원 형태의 입력을 처리하기 때문에, 이미지 처리에 특히 적합하다. 신경망의 각 레이어에서 일련의 필터가 이미지에 적용된다.





# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- 영상 인식과 처리에 사용되는 딥러닝 기술
- 합성곱 연산을 신경망에 적용한 영상 데이터에 최적화된 구조
- 신경망의 input이 영상데이터 임



CAT

CAT

[영상 인식]

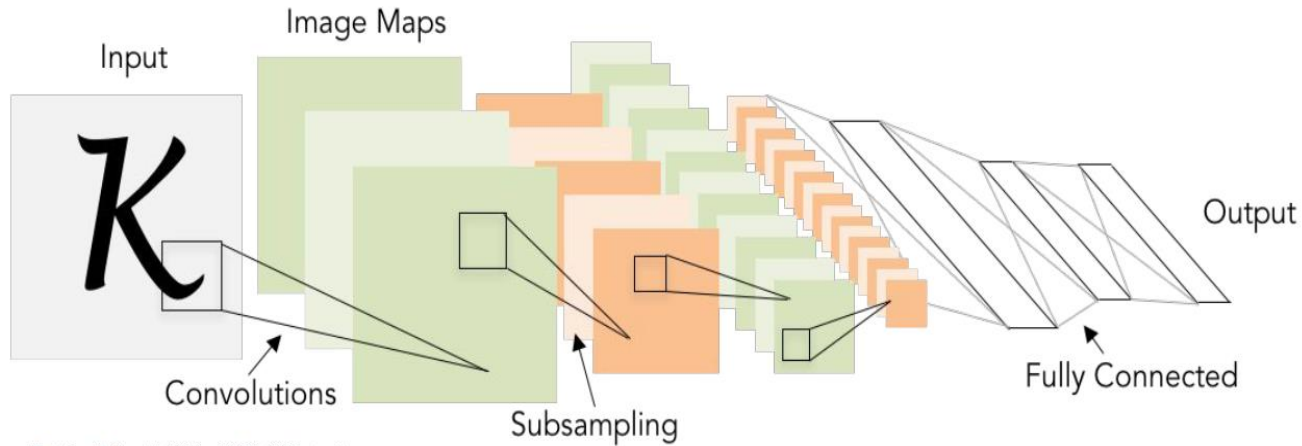


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

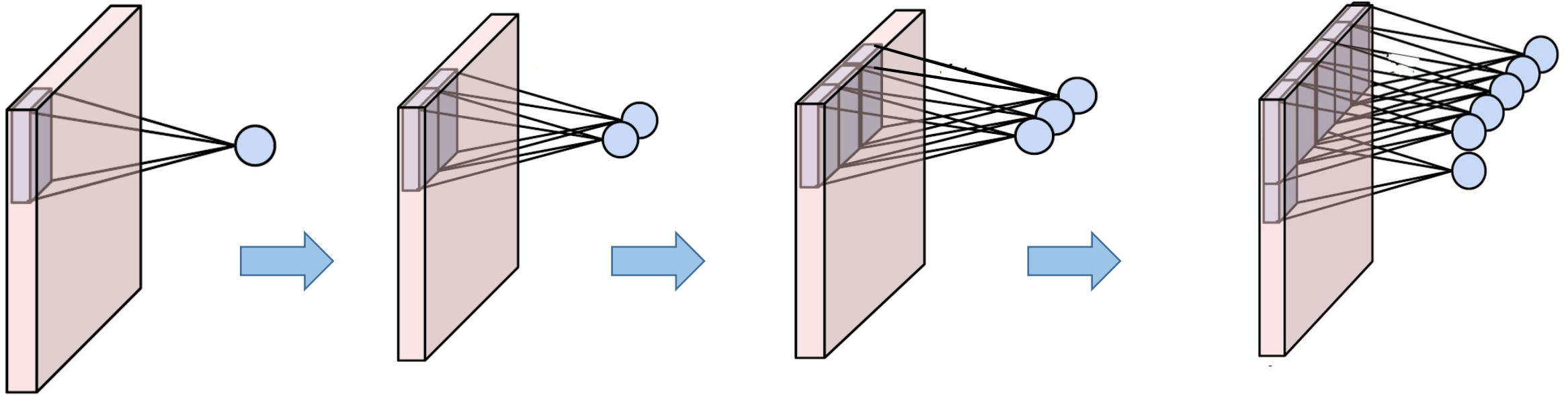
[합성곱 신경망 (Stanford cse231)]



[영상 변환]



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

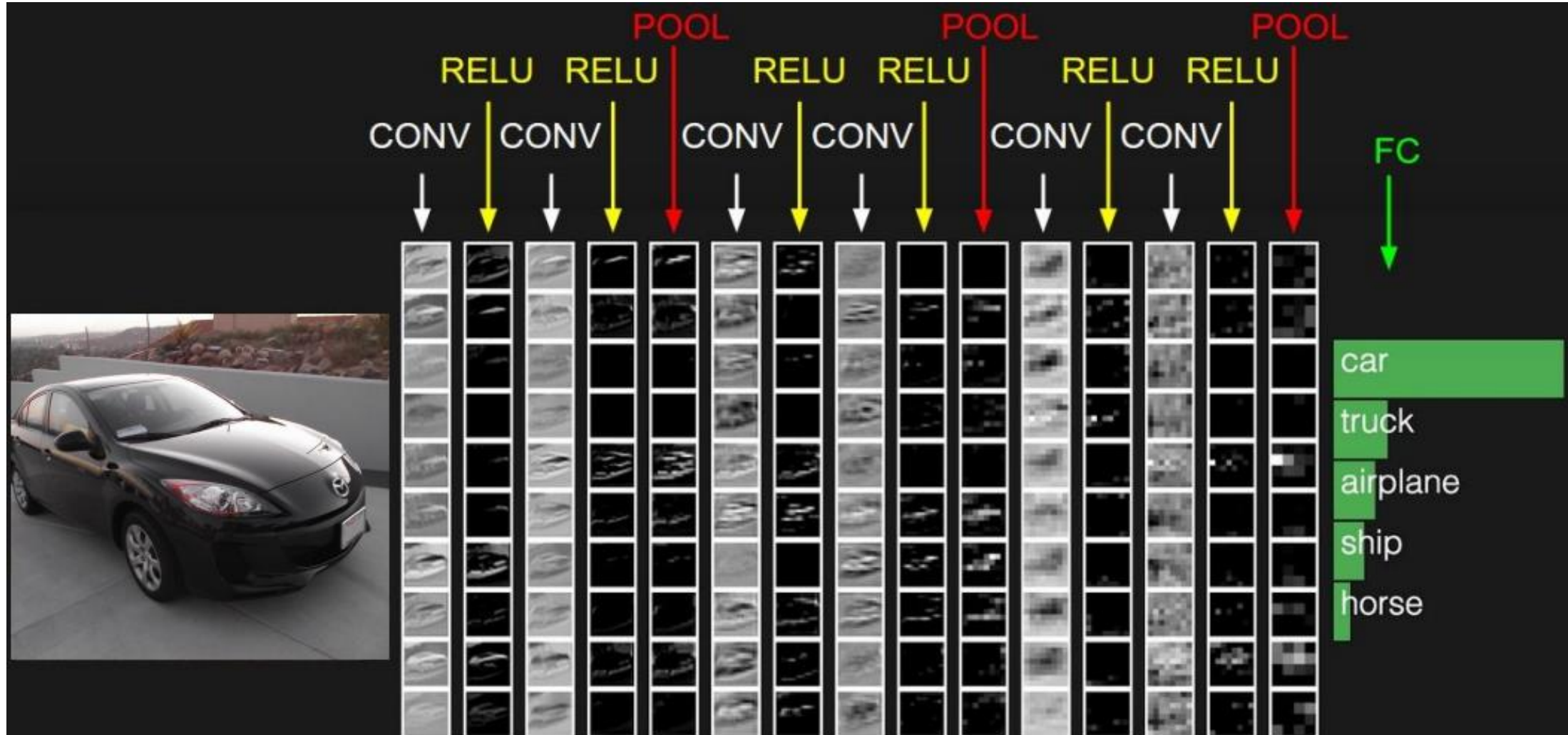


[ CNN (Stanford cse231) ]

신경망 안에서 합성곱(Convolution) 연산을 수행

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과



CONV: Convolution Layer

POOL : Pooling Layer

# 컨볼루션 (Convolution)

7x7 input image


\*

3x3 커널

w00	w01	w02
w10	w11	w21
w20	w21	w22

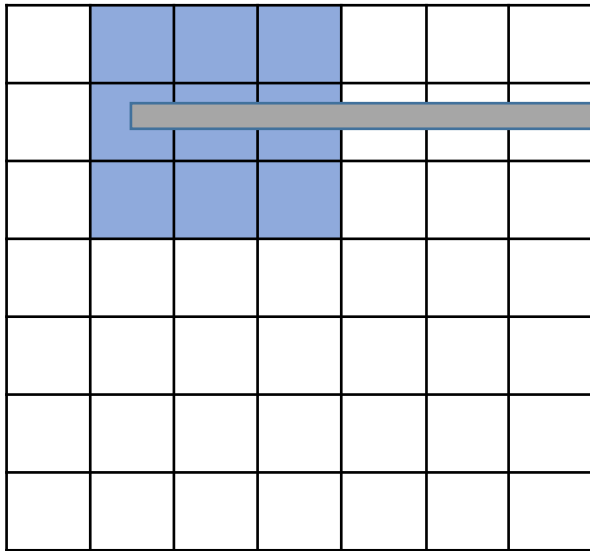
-1.2	2.0	3.1
0.1	1.5	-1.5
1.3	1.6	-0.7

output image




# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image

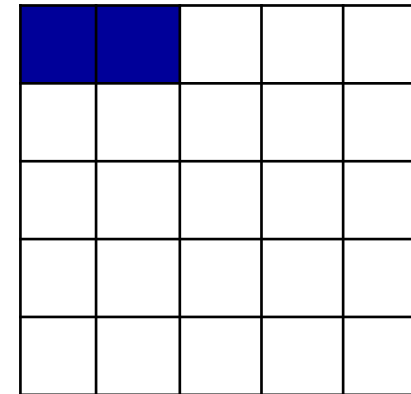


\*

3x3 kernel

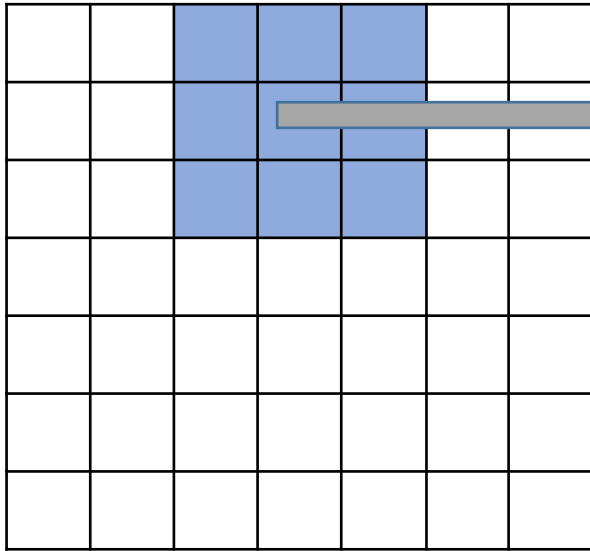
w00	w01	w02
w10	w11	w21
w20	w21	w22

output image



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image

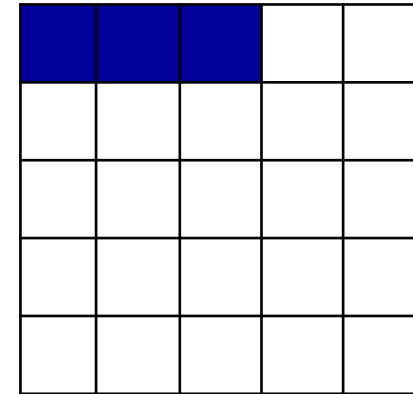


\*

3x3 kernel

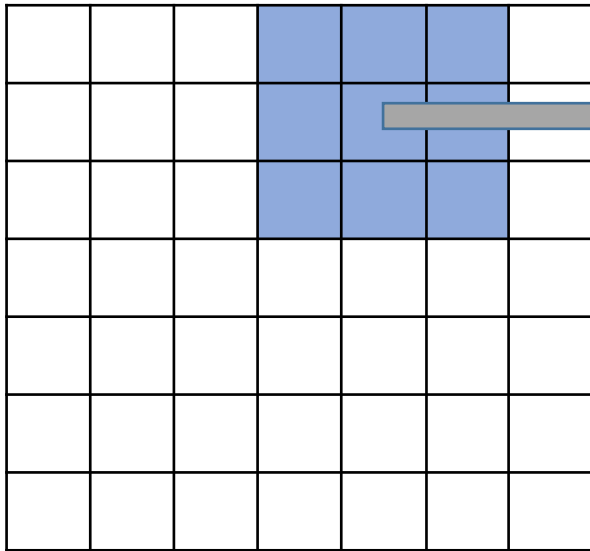
w00	w01	w02
w10	w11	w21
w20	w21	w22

output image



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image

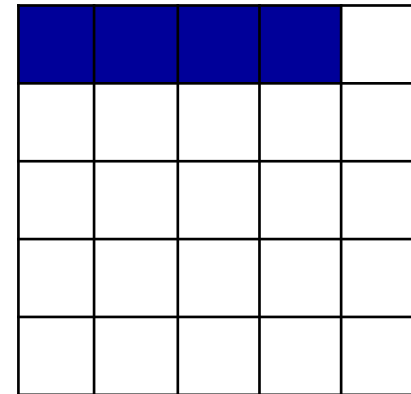


\*

3x3 kernel

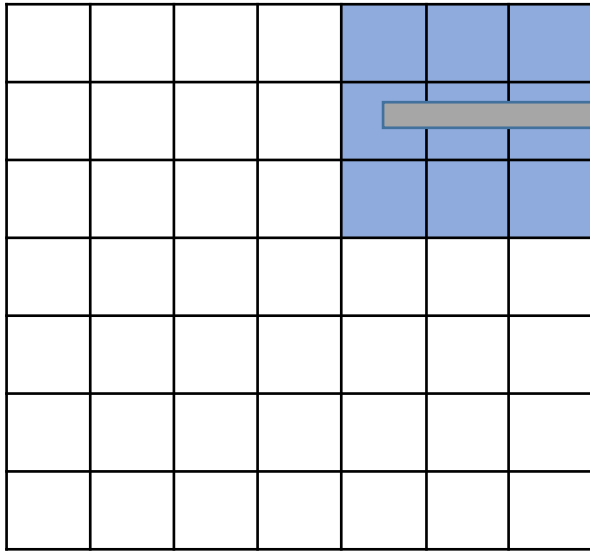
w00	w01	w02
w10	w11	w21
w20	w21	w22

output image



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image



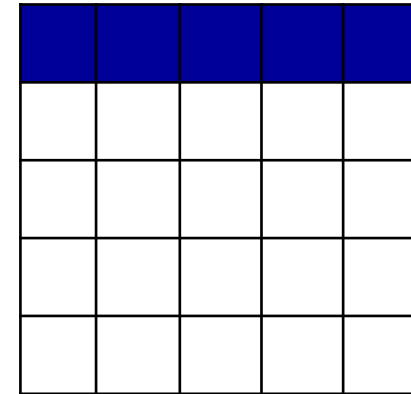
3x3 kernel

w00	w01	w02
w10	w11	w21
w20	w21	w22

\*

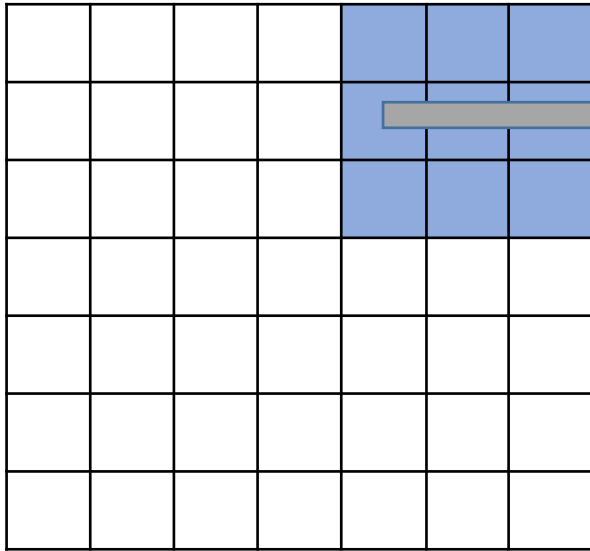


output image



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image



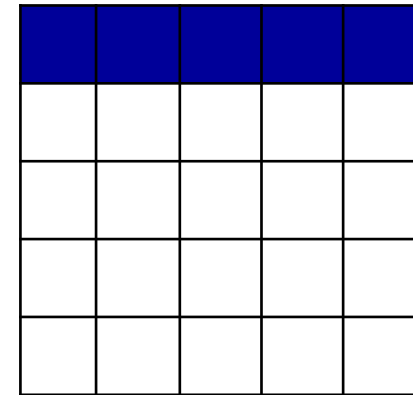
3x3 kernel

w00	w01	w02
w10	w11	w21
w20	w21	w22

\*



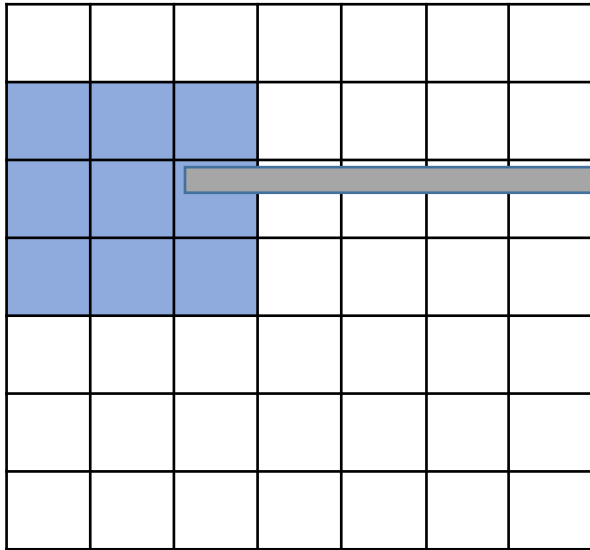
output image





# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image

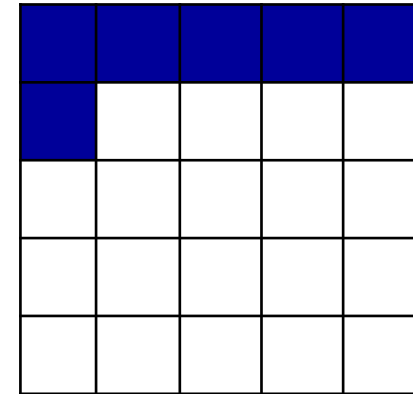


\*

3x3 kernel

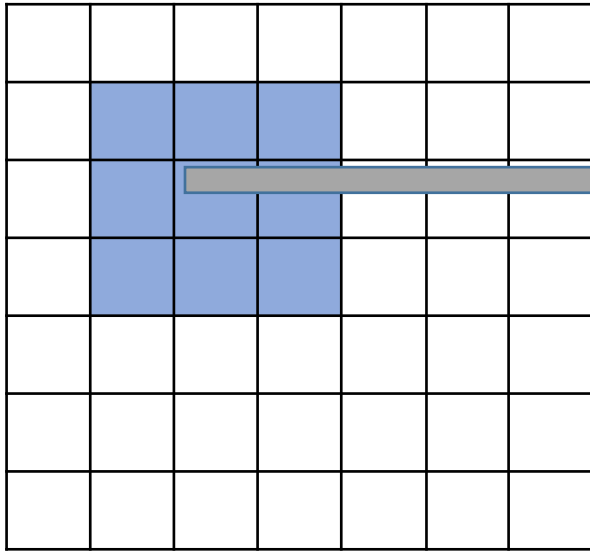
w00	w01	w02
w10	w11	w21
w20	w21	w22

output image



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image

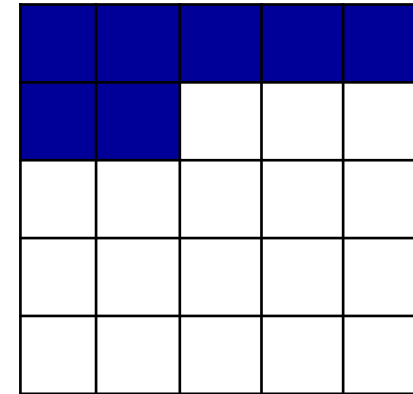


\*

3x3 kernel

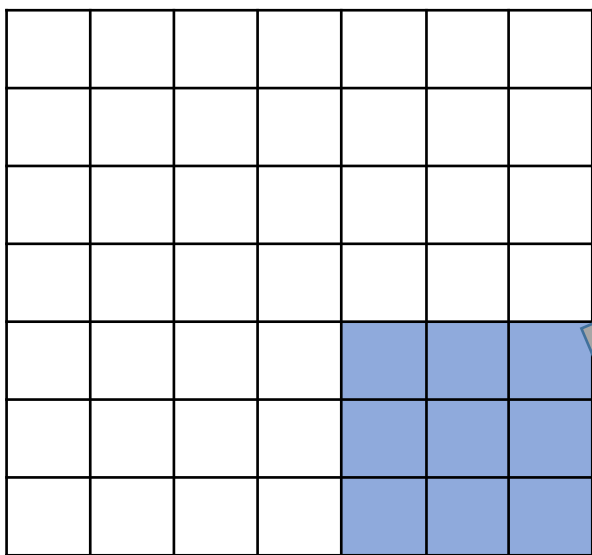
w00	w01	w02
w10	w11	w21
w20	w21	w22

output image



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

7x7 input image



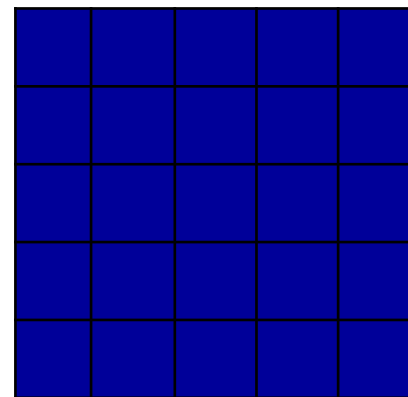
\*

3x3 kernel

w00	w01	w02
w10	w11	w21
w20	w21	w22

output image

5X5 image



출력 사이즈 = (입력 사이즈 - 커널 사이즈) + 1 = (7 - 3) + 1 = 5



# 컨볼루션 신경망 (CNN: Convolutional Neural Network) 인하공전 컴퓨터 정보과

- 컨볼루션 (Convolution)
- 컨볼루션은 주변 화소값들에 가중치를 곱해서 더한 후에 이것을 새로운 화소값으로 하는 연산이다.

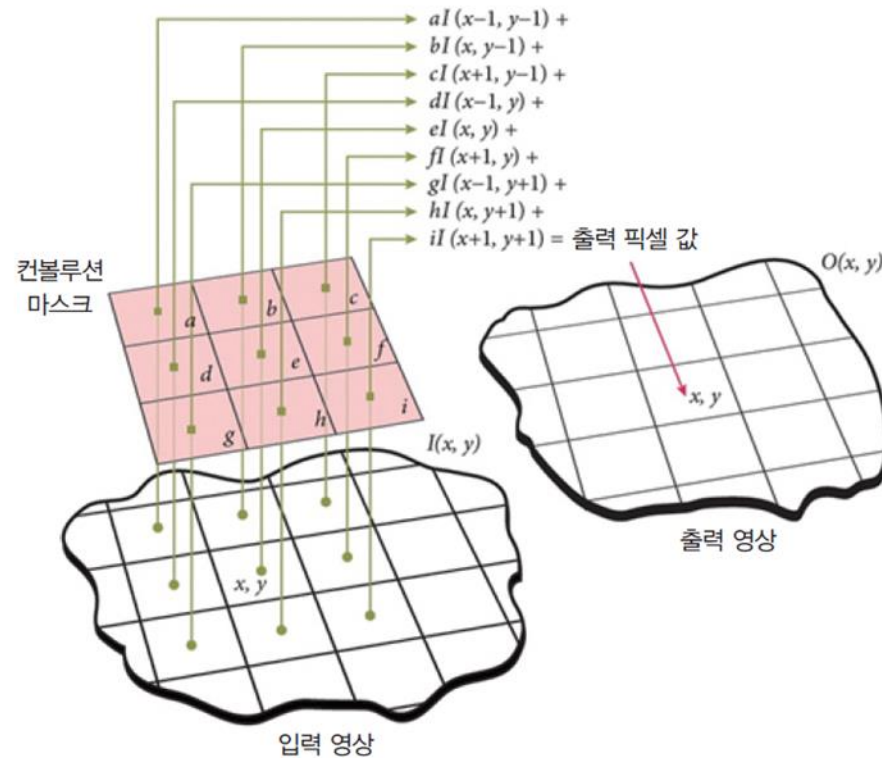


그림 9-6 영상 처리에서 컨볼루션 연산

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

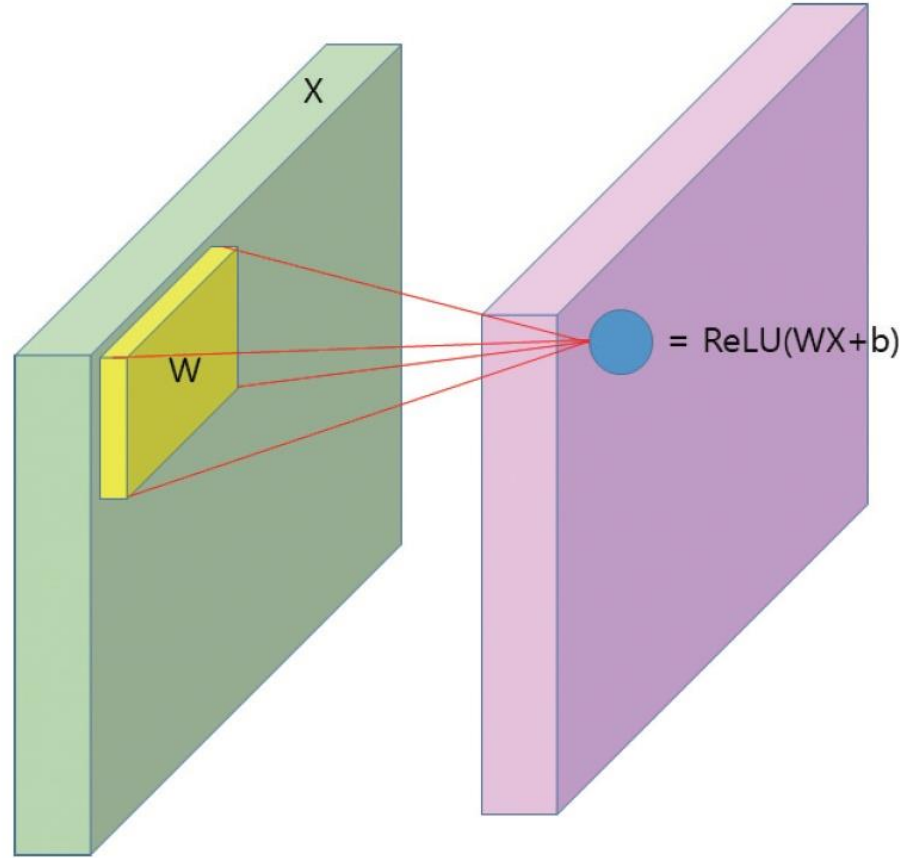


그림 15-15 신경망에서의 컨볼루션 연산

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- 컨볼루션을 수행한 결과는 특징맵(feature map)이라고 불리는 데, 그림 9-9가 그 이유를 보여준다.

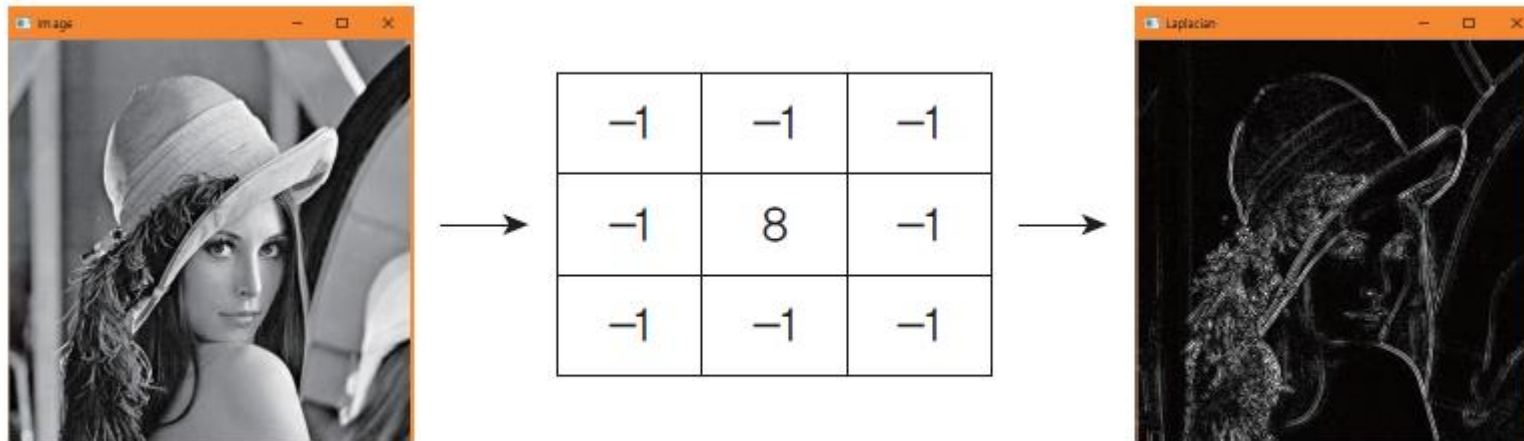


그림 9-9 영상 처리에서의 컨볼루션 연산

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

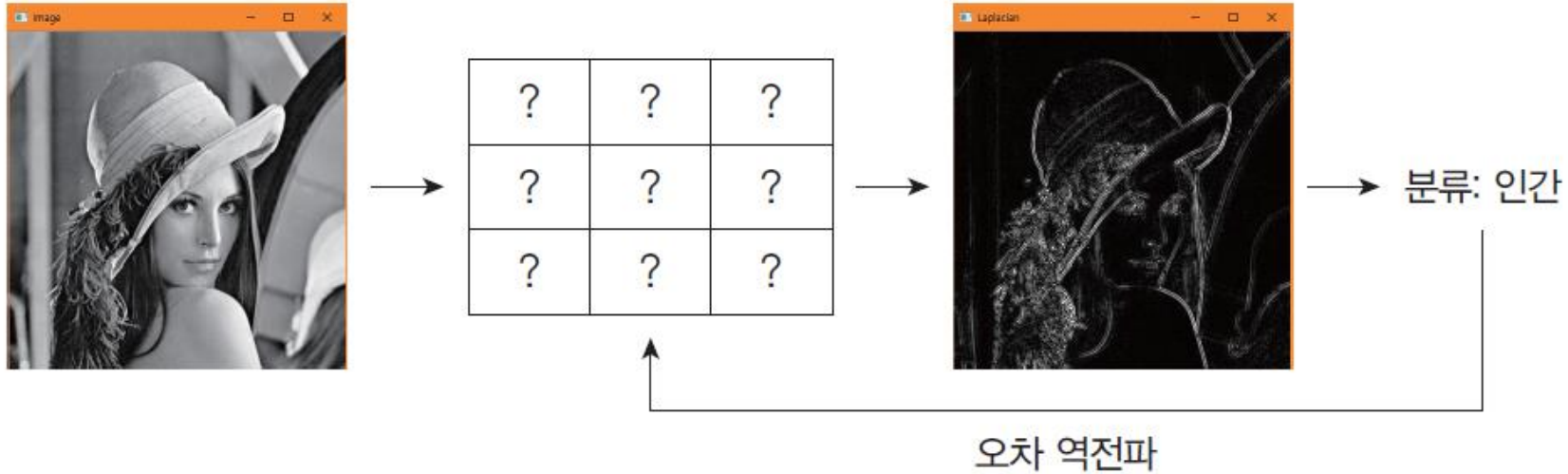


그림 9-10 컨볼루션 신경망에서는 커널의 가중치들이 학습된다.

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- 컨볼루션 신경망에서도 커널이 입력층의 각 화소를 중심으로 덮여 씌워진다. 앞 레이어의 값  $x$ 는 각 커널  $w$ 와 곱해져서 더해져서  $wX+b$ 가 된다.
- 이 계산값은  $\text{ReLU}()$ 와 같은 활성화 함수를 통과해서, 다음 레이어의 동일한 위치에  $\text{ReLU}(WX+b)$ 로 저장된다.

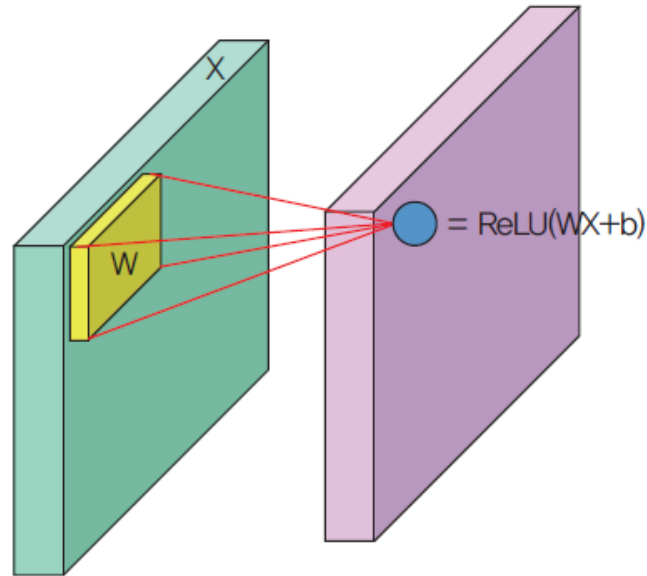


그림 9-11 신경망에서의 컨볼루션 연산



- 여러 개의 필터를 이용할 수 있다.
- *필터의 값은 미리 정해진 것이 아니고 학습된다.*

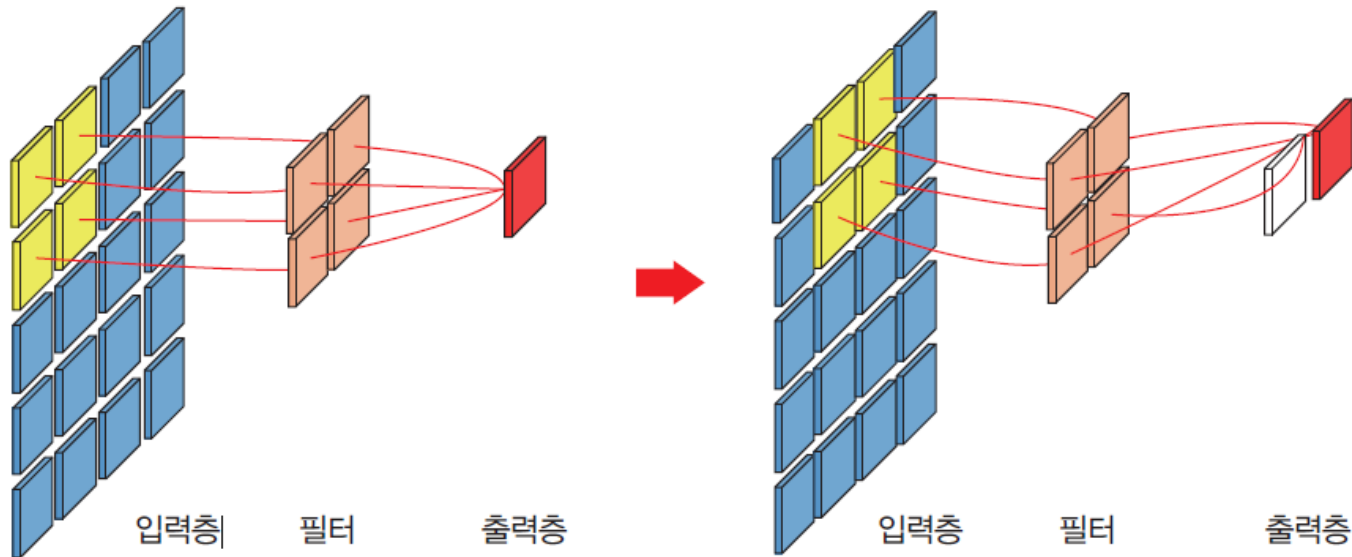
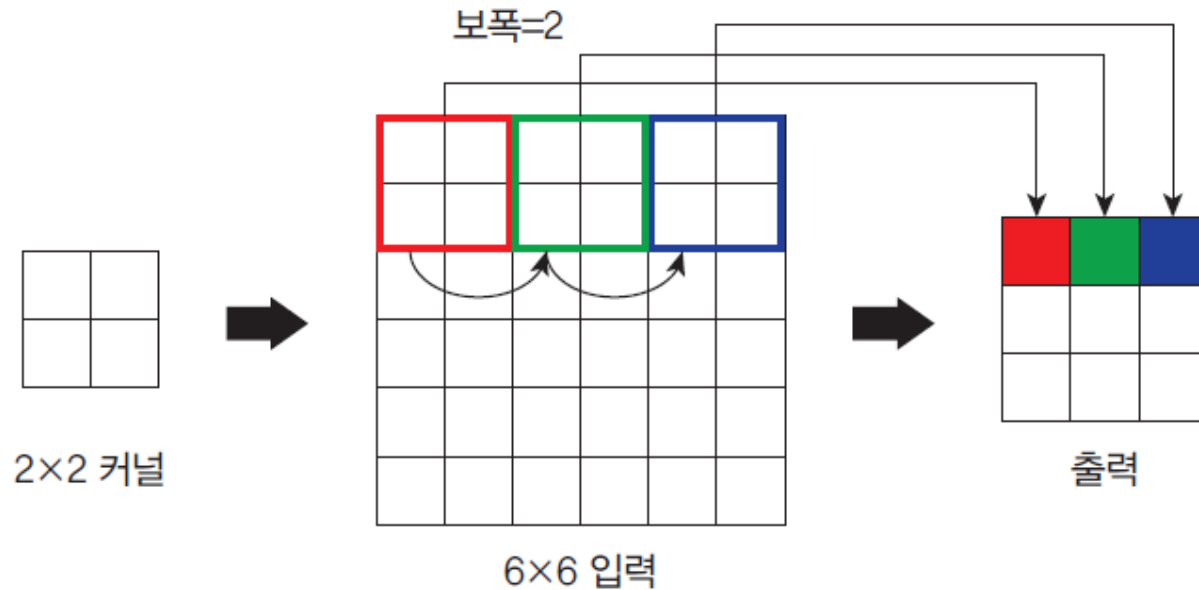


그림 9-12 컨볼루션 연산

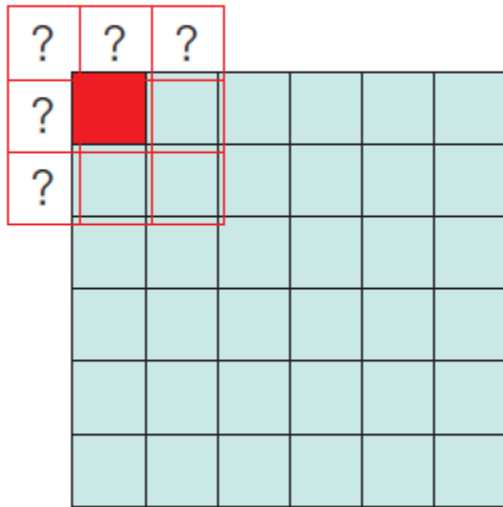
# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- 보폭(stride) 은 커널을 적용하는 거리이다. 보폭이 1이면 커널을 한 번에 1픽셀씩 이동하면서 커널을 적용하는 것이다.
- 보폭이 2라는 것은 하나씩 건너뛰면서 픽셀에 커널을 적용한다



- 패딩(padding)은 이미지의 가장자리를 처리하기 위한 기법 이



이미지의 가장 자리에 커널을  
적용하려니, 커널 아래에 픽셀이  
없네요. 어떻게 해야 할까요?

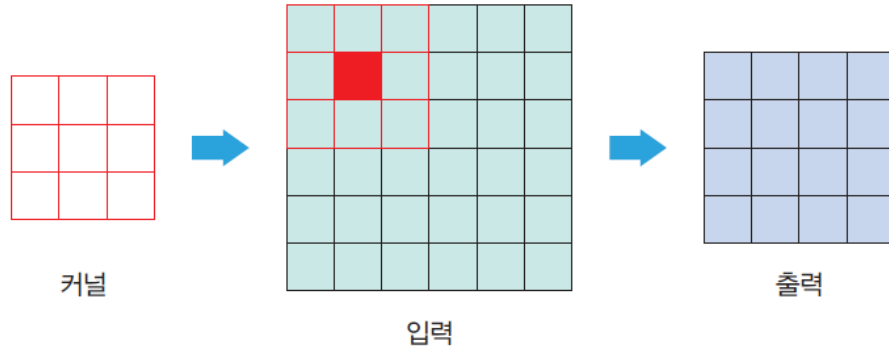


그림 9-13 패딩이 필요한 이유

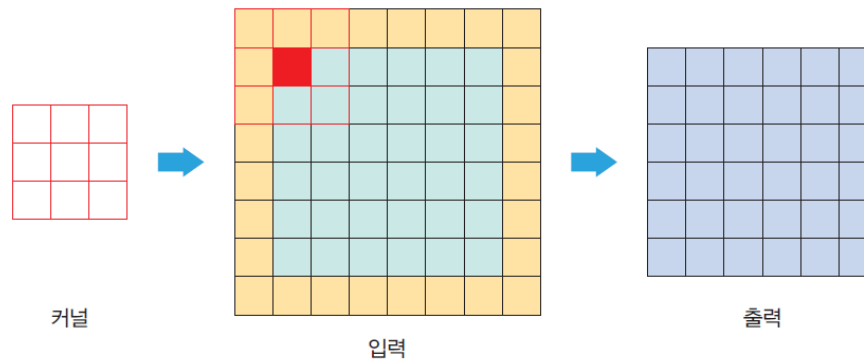
# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- **Valid:** 커널을 입력 이미지 안에서만 움직인다.



- **Same :** 입력 이미지의 주변을 특정값(예를 들면 0, 또는 이웃 픽셀값)으로 채우는 것



# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

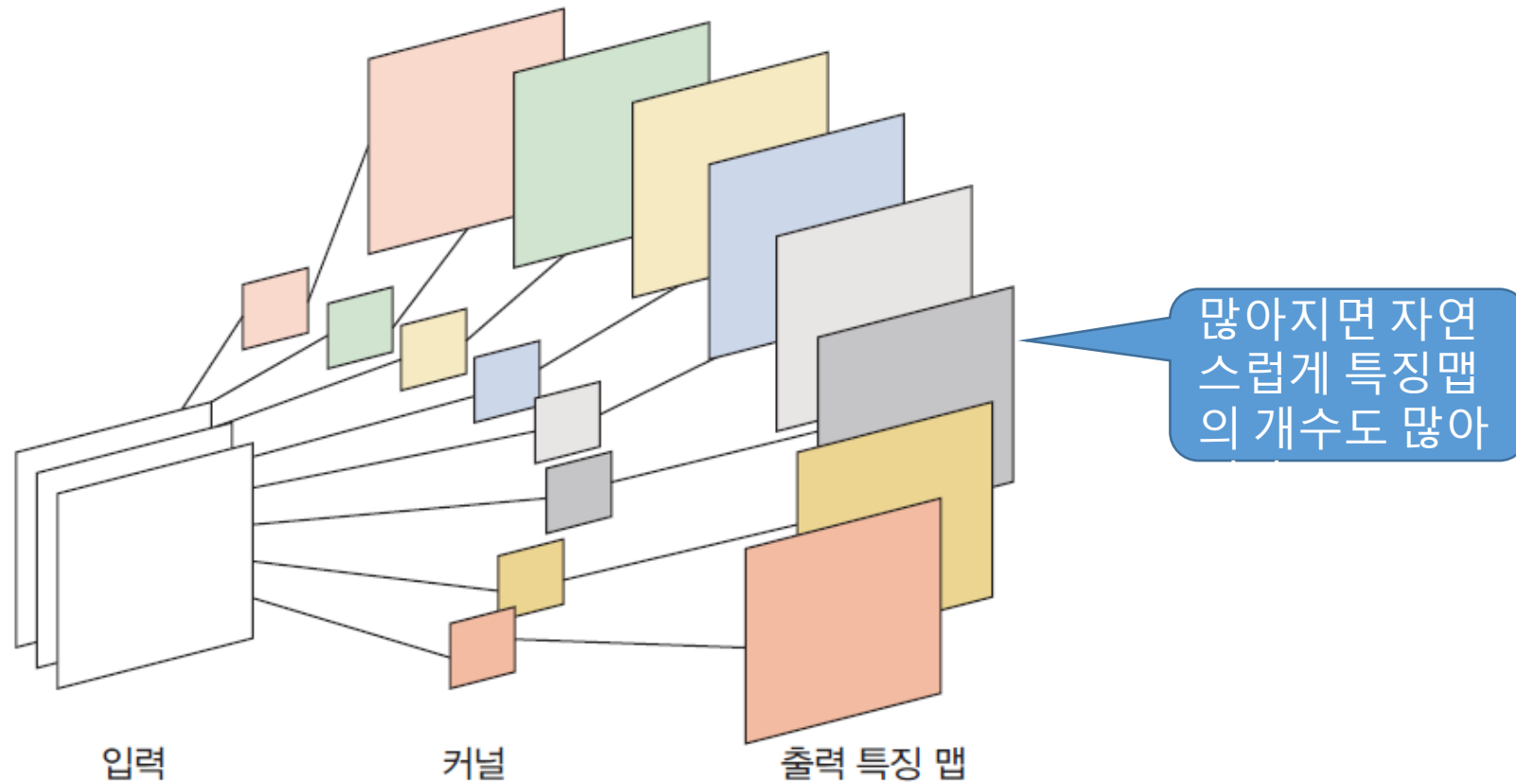
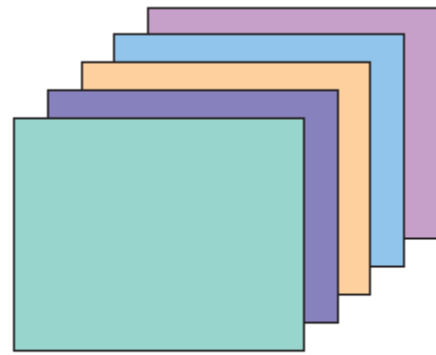


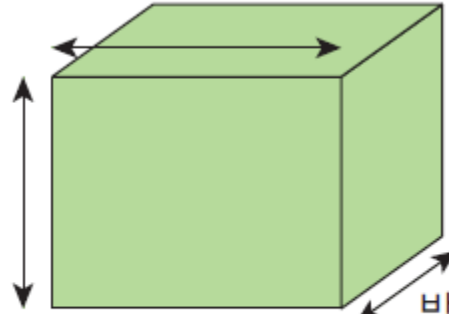
그림 9-14 필터가 여러 개일 때의 컨볼루션 레이어

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과



특징 맵들



특징 맵들을 박스  
형태로도 표시한다.

박스의 깊이가  
커널의 개수이다.

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

- `tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), activation=None, input_shape, padding='valid')`
  - filters: 필터의 개수이다.
  - kernel\_size: 필터의 크기이다.
  - strides: 보폭이다.
  - activation: 유닛의 활성화 함수이다.
  - input\_shape: 입력 배열의 형상
  - padding: 패딩 방법을 선택한다. 디폴트는 "valid"이다.

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

```
shape = (4, 28, 28, 3)
x = tf.random.normal(shape)
y = tf.keras.layers.Conv2D(2, 3, activation='relu', input_shape=shape[1:])(x)
print(y.shape)
```

(4, 26, 26, 2)



- 풀링(Pooling)이란 서브 샘플링이라고도 하는 것으로 입력 데이터의 크기를 줄이는 것이다.

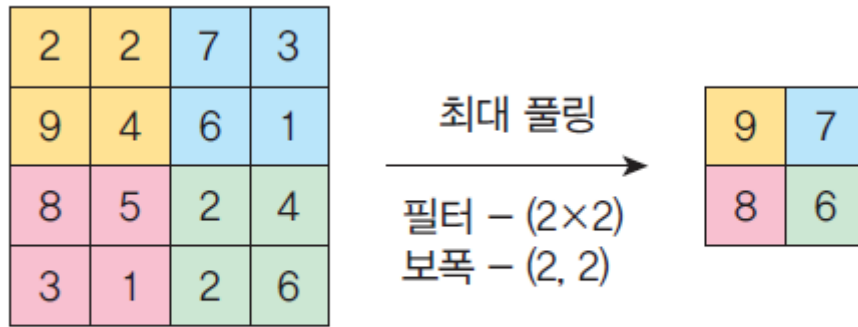


그림 9-17 풀링 레이어

- 컨벌루션처럼 윈도우를 움직여서 윈도우 안에 있는 숫자 중에서 가장 큰 값만 출력하는 연산이다.

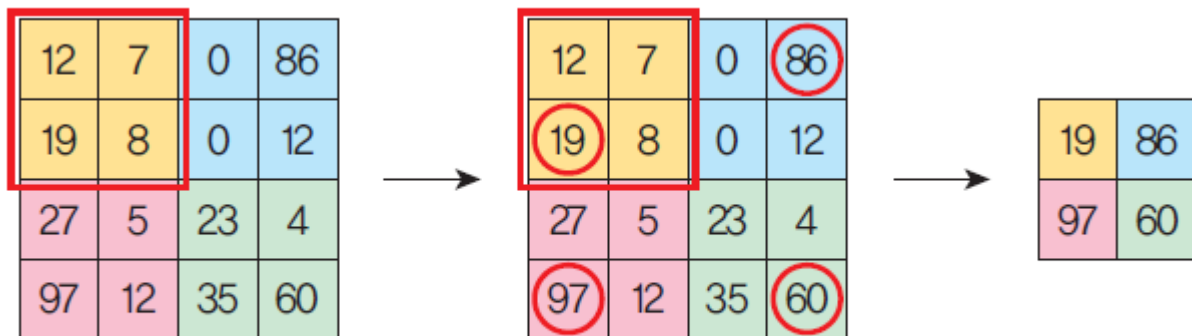


그림 9-18 풀링 연산

- 레이어의 크기가 작아지므로 계산이 빨라진다.
- 레이어의 크기가 작아진다는 것은 신경망의 매개변수가 작아진다는 것을 의미한다. 따라서 과적합이 나올 가능성이 줄어든다.
- 공간에서 물체의 이동이 있어도 결과는 변하지 않는다. 즉 물체의 공간이동에 대하여 둔감해지게 된다.

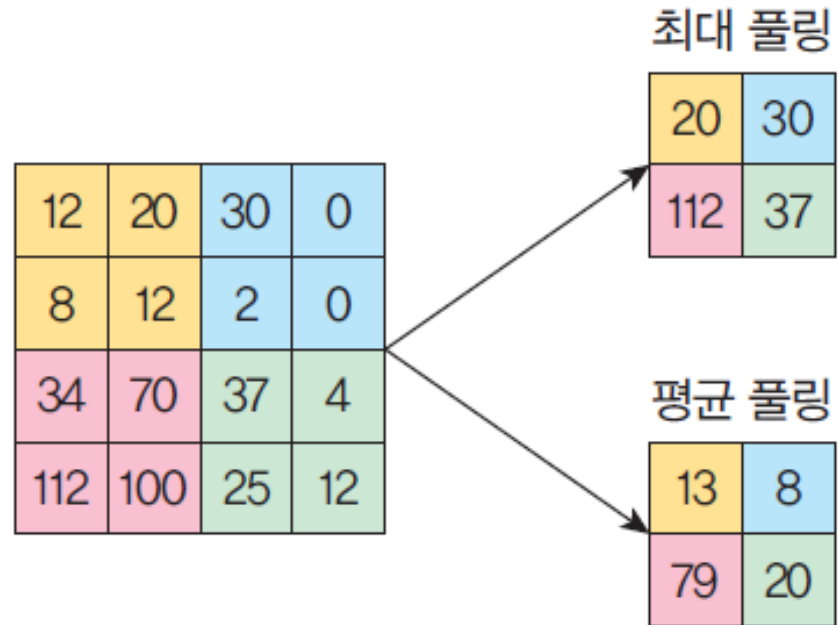
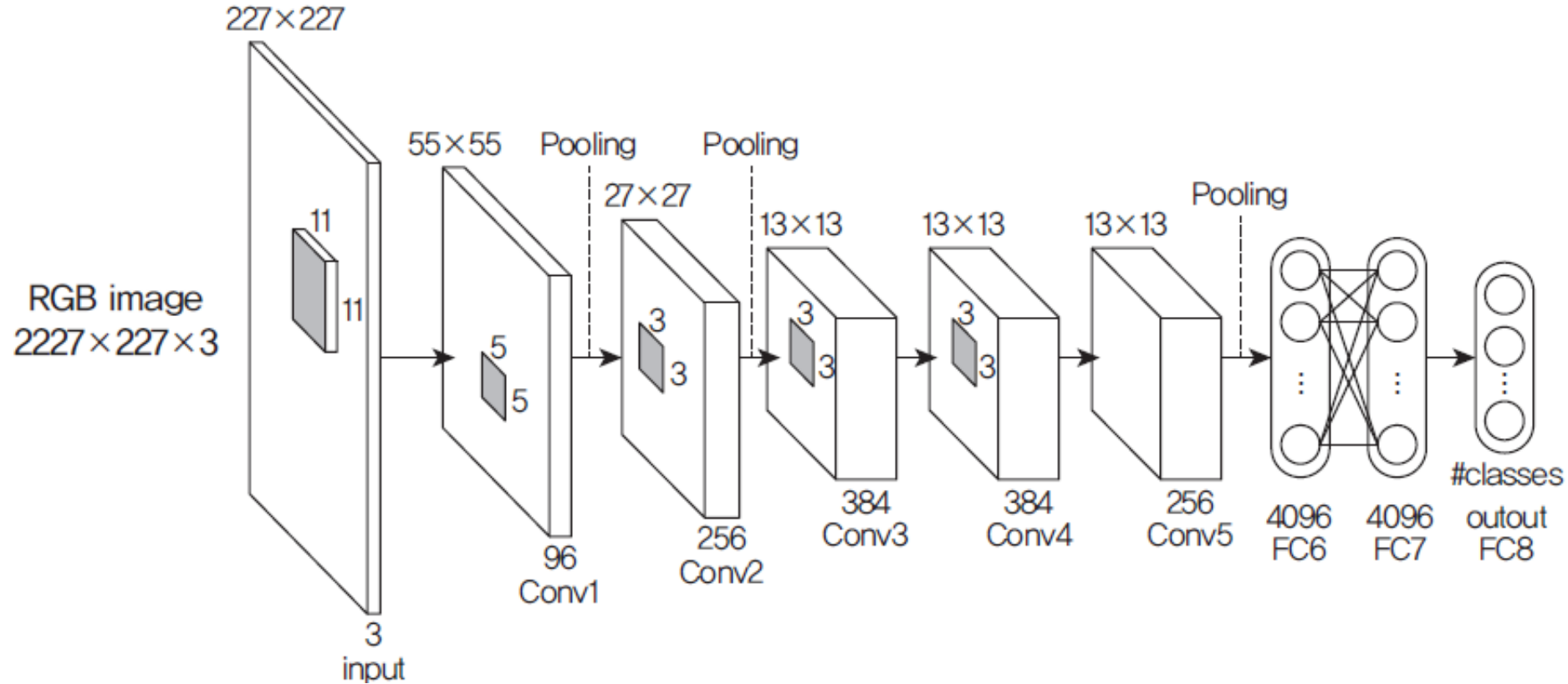


그림 9-19 풀링의 종류

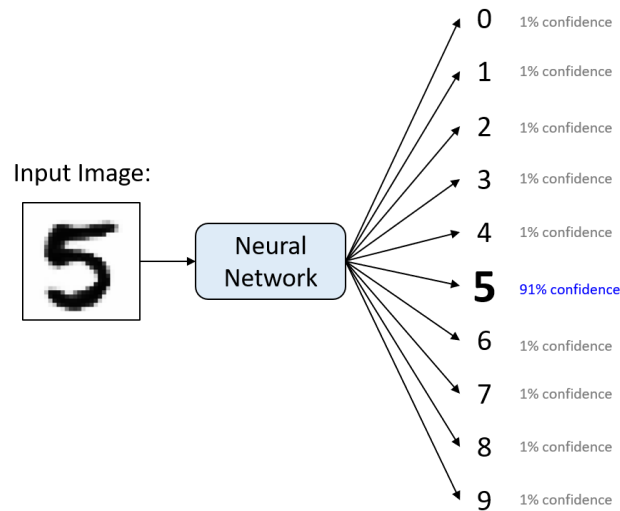
# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

인하공전 컴퓨터 정보과

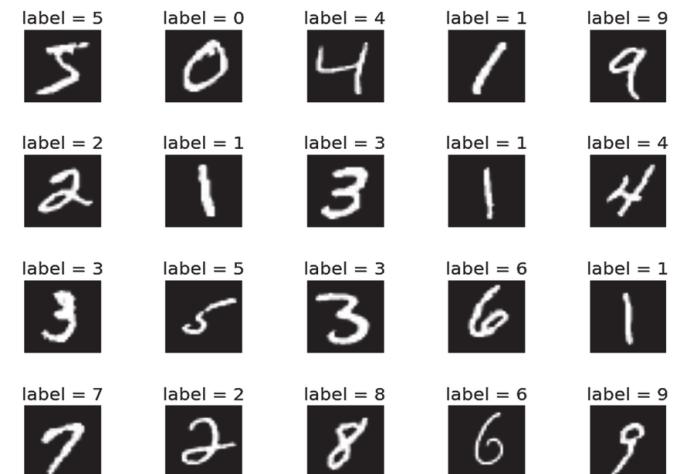


# MNIST 손 글씨 이미지 분류

- 필기체 이미지를 입력해 숫자를 인식 함
- MNIST dataset
  - Image size : 28x28
  - Image와 label (category)가 같이 저장되어 있음.
  - Training image 60000 장, test image 10000 장



[MNIST 손 글씨 이미지 분류]



[MNIST dataset]

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# 픽셀 값을 0~1 사이로 정규화한다.
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```



```
model.summary()
```

```
Model: "sequential_1"
```

---

Layer (type)	Output Shape	Param #
--------------	--------------	---------

---

conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
-------------------	--------------------	-----

---

max_pooling2d_2 (MaxPooling2)	(None, 13, 13, 32)	0
-------------------------------	--------------------	---

---

conv2d_4 (Conv2D)	(None, 11, 11, 64)	18496
-------------------	--------------------	-------

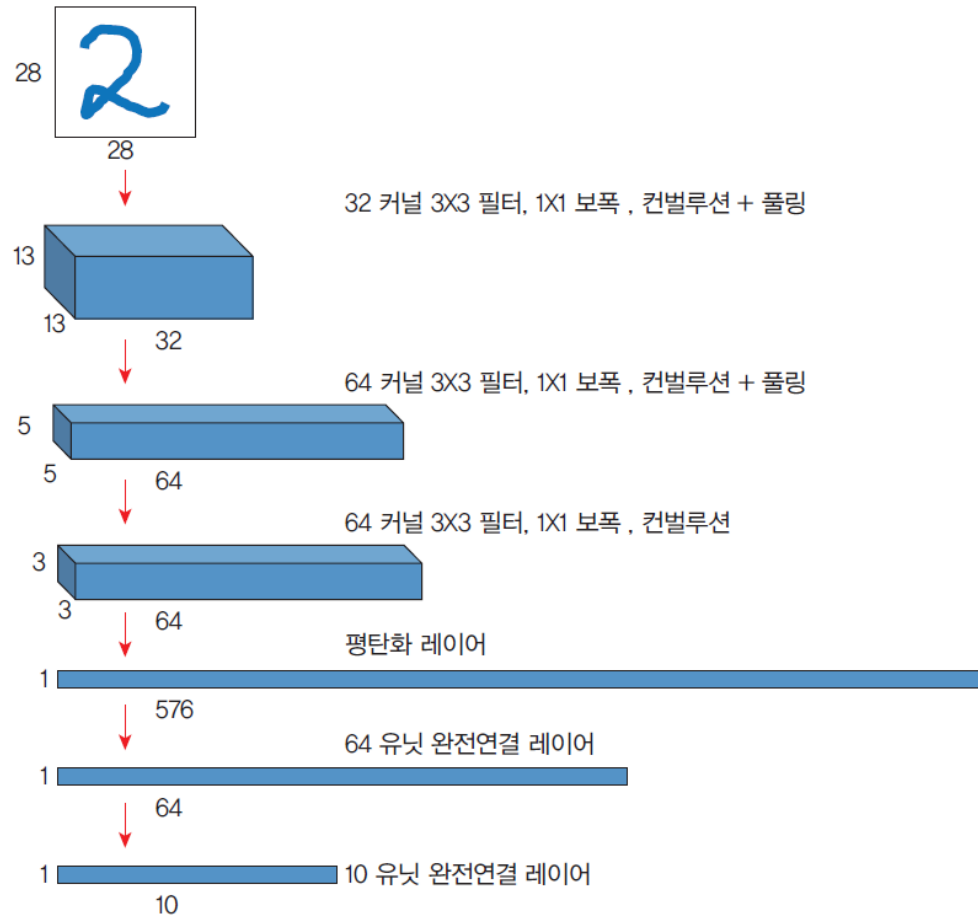
---

max_pooling2d_3 (MaxPooling2)	(None, 5, 5, 64)	0
-------------------------------	------------------	---

---

conv2d_5 (Conv2D)	(None, 3, 3, 64)	36928
-------------------	------------------	-------

---



```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])  
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5  
1875/1875 [=====] - 14s 7ms/step - loss: 0.1414 -  
accuracy: 0.9560  
...  
Epoch 5/5  
1875/1875 [=====] - 14s 7ms/step - loss: 0.0194 -  
accuracy: 0.9940
```

# 컨볼루션 신경망 (CNN: Convolutional Neural Network)

## 커널 (kernel)

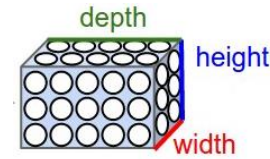
- 합성 곱 (Convolution)과정에서 사용되는 행렬

w00	w01	w02
w10	w11	w11
w20	w21	w22



## 채널 (channel)

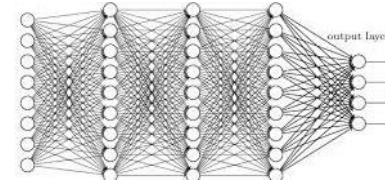
- 커널의 수
- 채널의 수가 output의 depth가 됨



w00	w01	w02	w00	w01	w02
w10	w11	w11	w00	w01	w02
w20	w21	w22	w00	w01	w02

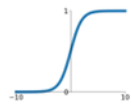
## 레이어 (layer)

- 신경망 안에서 연산을 수행하는 각 단계
- 레이어의 종류
  - . **Convolution Layer** : 합성곱 연산을 수행 하는 layer
  - . **Activation Layer** : 신경망 학습의 효율을 높이기 위하여 데이터 분포를 변형 해주는 layer
  - . **Pooling Layer** : 정보의 크기를 줄여주는 Layer. Ex) Average Pooling, Max Pooling

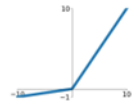


## Activation Functions

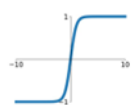
**Sigmoid**  
 $\sigma(x) = \frac{1}{1+e^{-x}}$



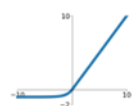
**Leaky ReLU**  
 $\max(0.1x, x)$



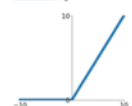
**tanh**  
 $\tanh(x)$



**Maxout**  
 $\max(w_1^T x + b_1, w_2^T x + b_2)$

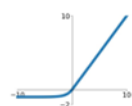


**ReLU**  
 $\max(0, x)$



**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Max

3	1	1	3
2	5	0	2
1	4	2	1
4	7	2	4

=

5	3
7	4

[max pooling]

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

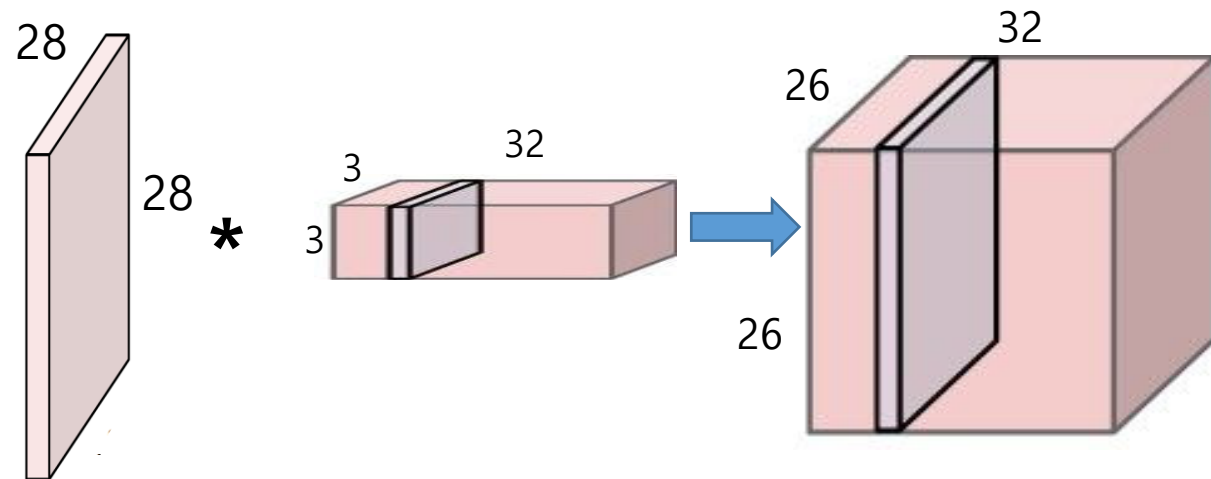
```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

Number of parameters  
= input depth \* kernel size \* kernel size \* channel + channel  
=>  $1 * 3 * 3 * 32 + 32 = 320$

→ Input=28x28x1, kernel=3x3, channel=32  
=> output size=26x26x32



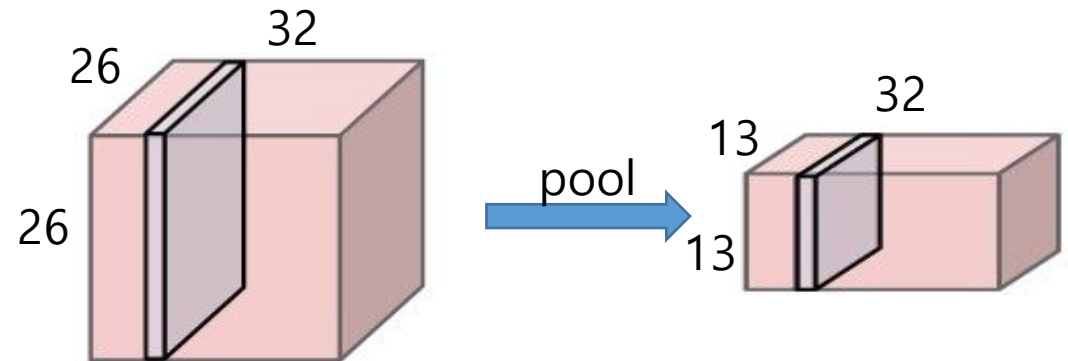
# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

→ Input=26x26x32, MaxPool =(2,2)  
=> output size=13x13x32



# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

Number of parameters

=input depth\*kernel size\*kernel size\*channel+channel  
=>  $32 \times 3 \times 3 \times 64 + 64 = 18496$



Input=13x13x32, kernel=3x3, channel=64  
=> output size=11x11x64



# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

→ Input=11x11x64, MaxPool =(2,2)  
=> output size=5x5x64

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

➡ Input=5x5x64, => output size=1600

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras  
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

➡ Input=1600, => output size=1600

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Number of parameters  
= input size \* number of class + number of class  
=>  $1600 * 10 + 10 = 16010$

➡ Input=1600, => output size=10

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
input_shape = (28, 28, 1)
```

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====Model: "sequential"		

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 26, 26, 32)	320
-----		
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
-----		
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
-----		
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
-----		
flatten (Flatten)	(None, 1600)	0
-----		
dropout (Dropout)	(None, 1600)	0
-----		
dense (Dense)	(None, 10)	16010

=====

Total params: 34,826

Trainable params: 34,826

Non-trainable params: 0

# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
batch_size = 128  
epochs = 15
```

```
model.compile(loss="categorical_crossentropy", optimizer="adam",  
metrics=["accuracy"])
```

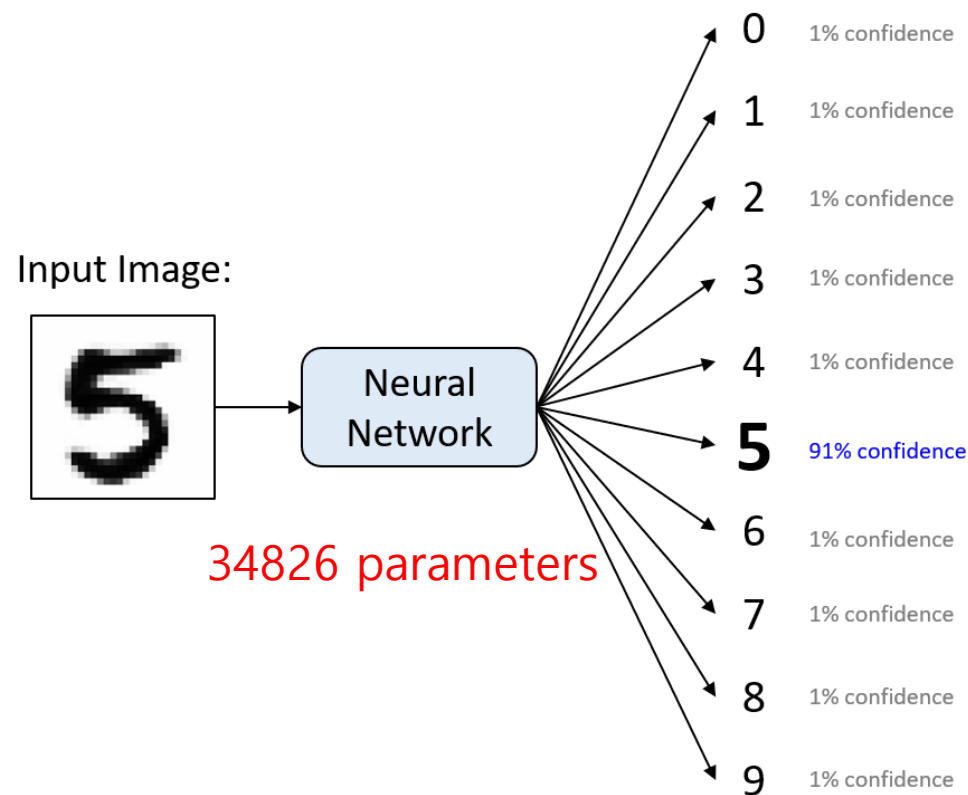
```
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,  
validation_split=0.1)
```

Compile : Model을 기계가 이해할 수 번역

Fit : 학습시킴 (training)

```
score = model.evaluate(x_test, y_test, verbose=0)  
print("Test loss:", score[0])  
print("Test accuracy:", score[1])
```

학습된 model의 성능 테스트



# MNIST 손 글씨 이미지 분류- CNN 케라스 구현

```
batch_size = 128  
epochs = 15
```

```
model.compile(loss="categorical_crossentropy", optimizer="adam",  
metrics=["accuracy"])
```

```
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,  
validation_split=0.1)
```

```
score = model.evaluate(x_test, y_test, verbose=0)  
print("Test loss:", score[0])  
print("Test accuracy:", score[1])
```

```
Epoch 1/15  
422/422 [=====] - 13s 29ms/step - loss: 0.7840 - accuracy:  
0.7643 - val_loss: 0.0780 - val_accuracy: 0.9780  
Epoch 2/15  
422/422 [=====] - 13s 31ms/step - loss: 0.1199 - accuracy:  
0.9639 - val_loss: 0.0559 - val_accuracy: 0.9843  
Epoch 3/15  
422/422 [=====] - 14s 33ms/step - loss: 0.0845 - accuracy:  
0.9737 - val_loss: 0.0469 - val_accuracy: 0.9877  
Epoch 4/15  
422/422 [=====] - 14s 33ms/step - loss: 0.0762 - accuracy:  
0.9756 - val_loss: 0.0398 - val_accuracy: 0.9895  
Epoch 5/15  
422/422 [=====] - 15s 35ms/step - loss: 0.0621 - accuracy:  
0.9812 - val_loss: 0.0378 - val_accuracy: 0.9890  
Epoch 6/15  
422/422 [=====] - 17s 40ms/step - loss: 0.0547 - accuracy:  
0.9825 - val_loss: 0.0360 - val_accuracy: 0.9910  
Epoch 7/15  
422/422 [=====] - 17s 41ms/step - loss: 0.0497 - accuracy:  
0.9840 - val_loss: 0.0311 - val_accuracy: 0.9920  
Epoch 8/15  
422/422 [=====] - 16s 39ms/step - loss: 0.0443 - accuracy:  
0.9862 - val_loss: 0.0346 - val_accuracy: 0.9910  
Epoch 9/15  
422/422 [=====] - 17s 39ms/step - loss: 0.0436 - accuracy:  
0.9860 - val_loss: 0.0325 - val_accuracy: 0.9915  
Epoch 10/15  
422/422 [=====] - 16s 38ms/step - loss: 0.0407 - accuracy:  
0.9865 - val_loss: 0.0301 - val_accuracy: 0.9920  
Epoch 11/15  
422/422 [=====] - 16s 37ms/step - loss: 0.0406 - accuracy:  
0.9874 - val_loss: 0.0303 - val_accuracy: 0.9920  
Epoch 12/15  
237/422 [=====>.....] - ETA: 7s - loss: 0.0398 - accuracy: 0.9877
```

```
Test loss: 0.023950600996613503  
Test accuracy: 0.9922000169754028
```

99.2% 정답률

[CS 230 - Convolutional Neural Networks Cheatsheet  
\(stanford.edu\)](#)



# 수고 하셨습니다



[jhmin@inhatec.ac.kr](mailto:jhmin@inhatec.ac.kr)