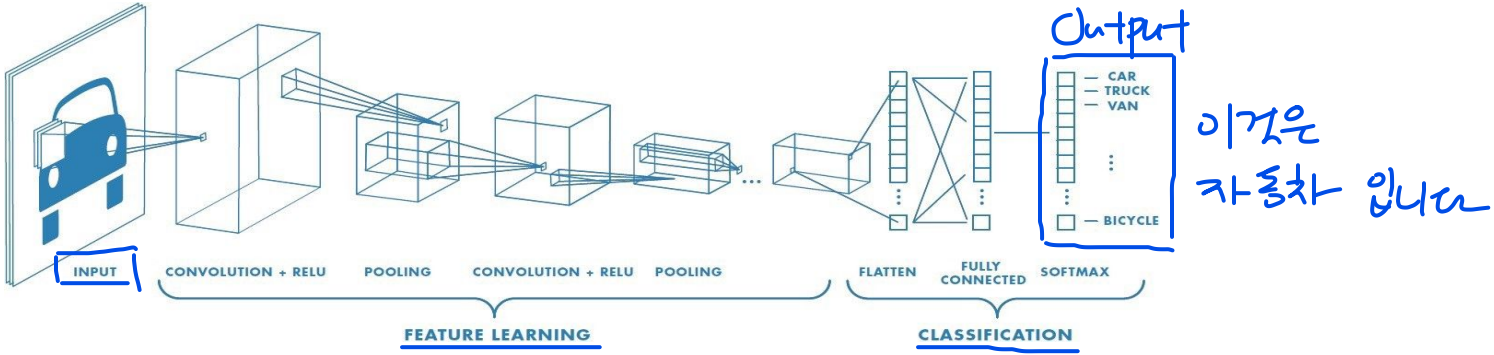
# Convolutional Neural Network (CNN / ConvNet)

합성곱 신경망

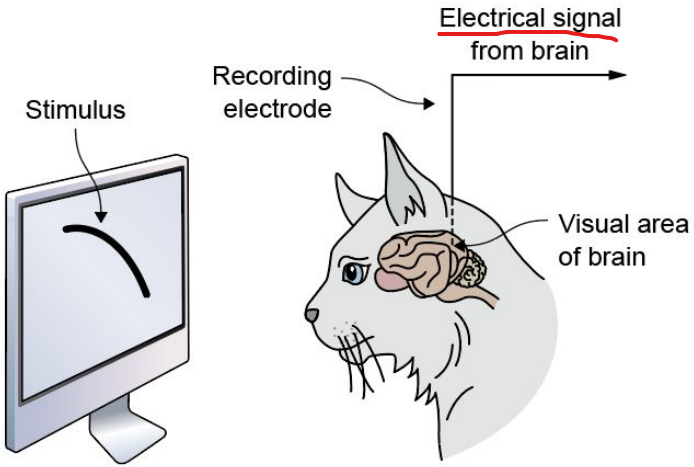
# 컨볼루션 신경망 (Convolutional Neural Network)



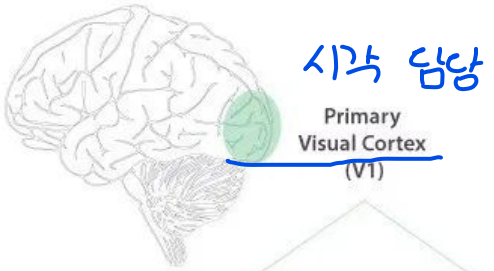

- **시각적 이미지 분석**에 많이 활용되는 **심층신경망(Deep NN)**의 일종으로 **합성곱(Convolution) 신경망**이라고도 함.
- 아키텍처는 **인간 두뇌의 뉴런 연결 패턴과 유사**하며 시각 피질조직 (Visual Cortex)의 구성에서 영감을 받았습니다.
- 네트워크가 **convolution**이라는 수학적 연산을 사용함을 나타냅니다.
  컨볼루션은 (디지털 필터링에 주로 사용되는) 특수한 종류의 선형 연산입니다.

벡터 x 행렬

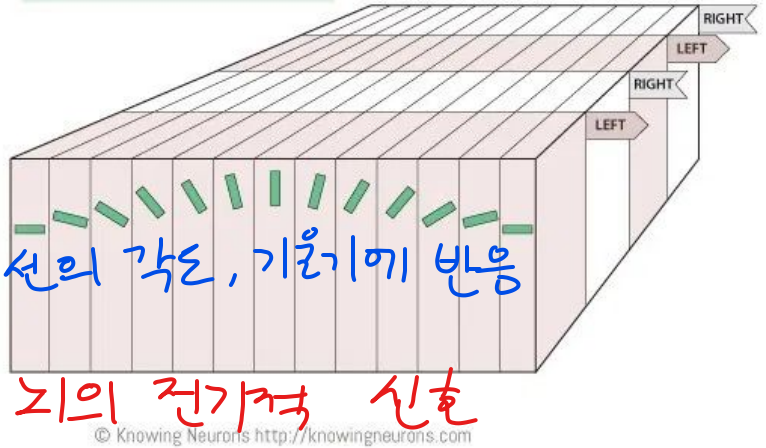
# 뇌의 동작으로부터 영감(Inspiration)



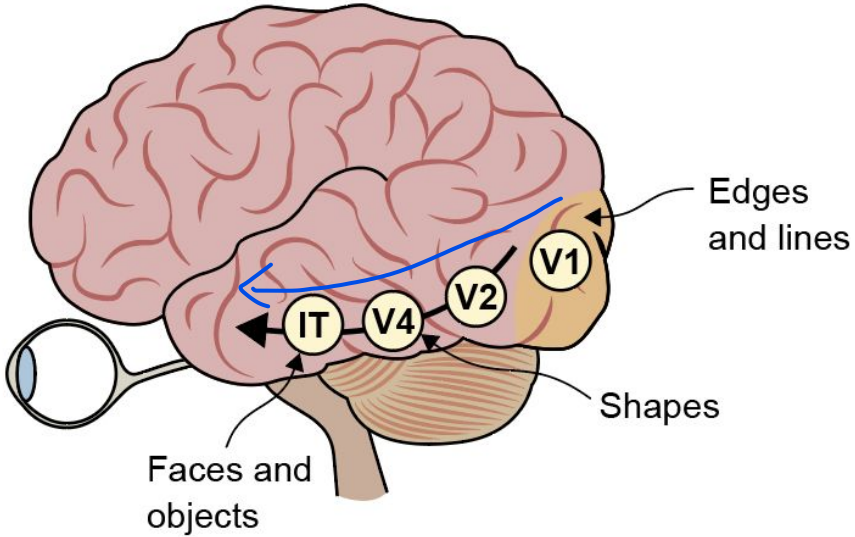

Hubel과 Wiesel (1959)의 실험 설정 - 움직이는 경계(Edge)가 고양이에게 보여질 때 반응하는 뉴런을 시각 피질에서 발견.

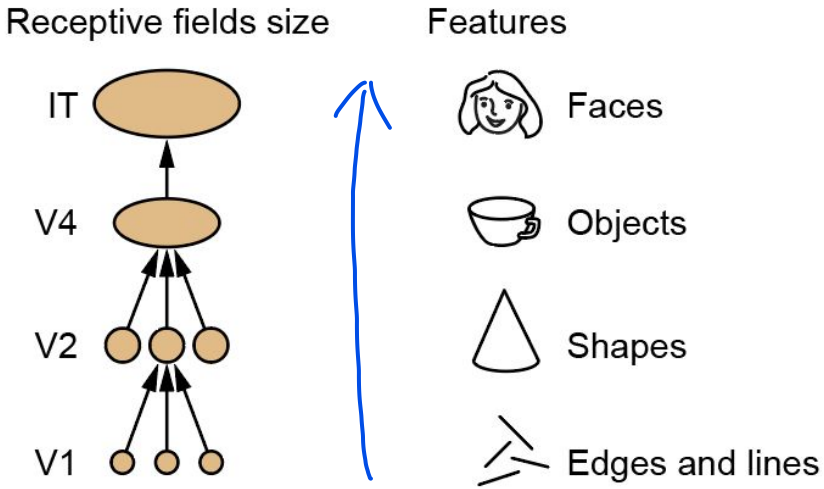



**개별 뉴런**은 수용 영역으로 알려진 **시야의 제한된 영역에서만 자극에 반응**합니다. 이러한 필드 모음은 전체 가시 영역을 덮기 위해 겹쳐집니다.



# 뇌의 동작으로부터 영감(Inspiration)

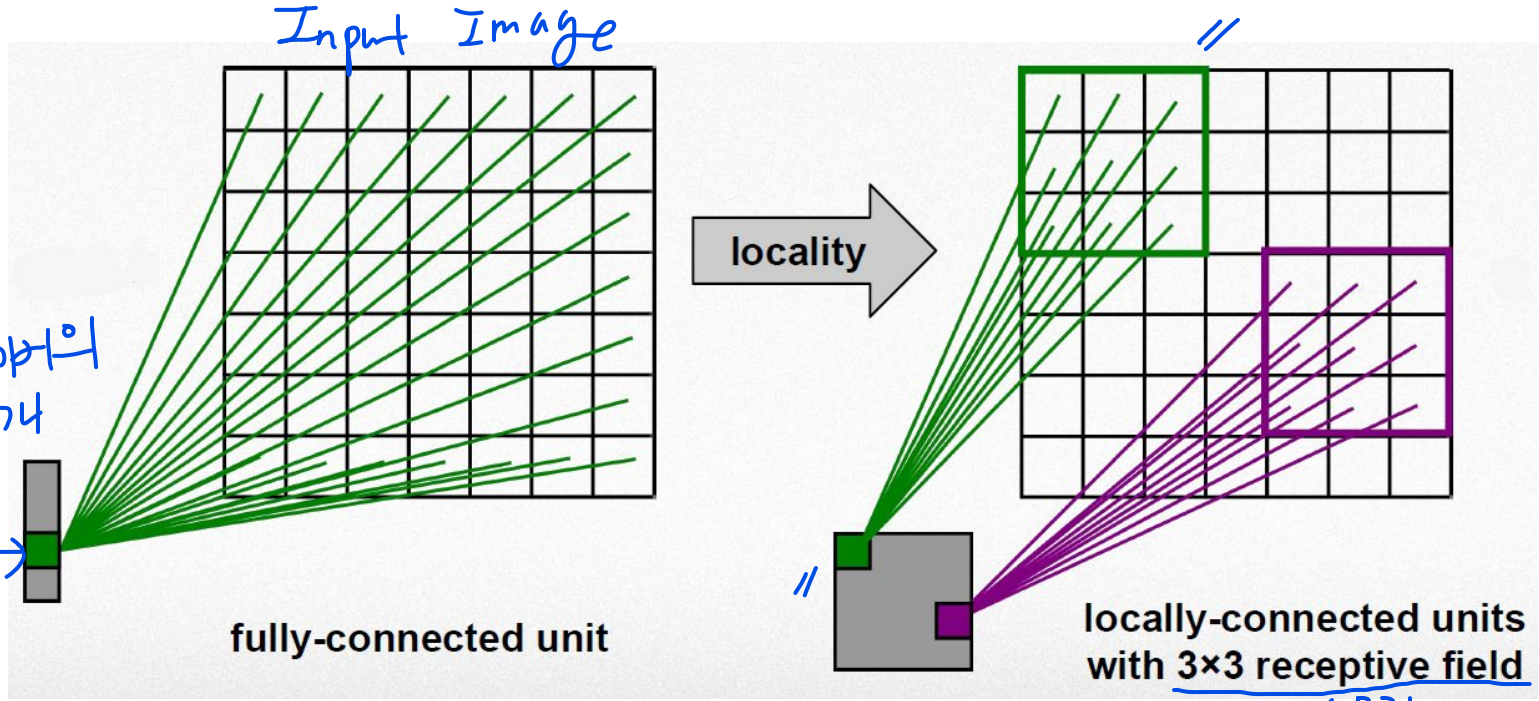



서로 다른 지역의 뉴런이 점점 더 큰 **수용장(Receptive fields)** 과 점점 더 복잡한 자극(Features)에 반응하는 뇌의 시각 피질 조직. Ref.: https://medium.com/@ManningBooks/deep-learning-for-image-like-data-5ff459329b11

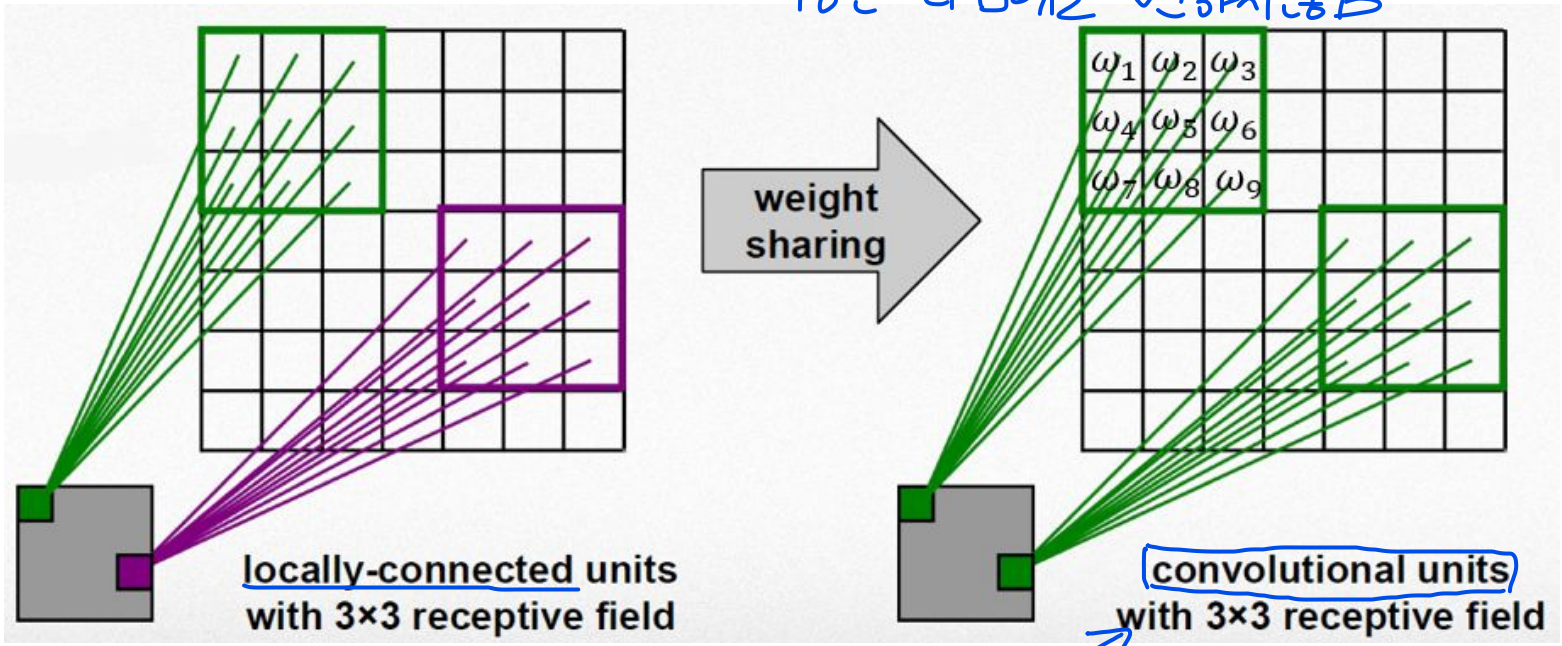
전부 연결되어있는 Network 부분적 (국지적)으로 연결 //

# Fully-Connected NN (fcNN) vs. Locally-Connected NN



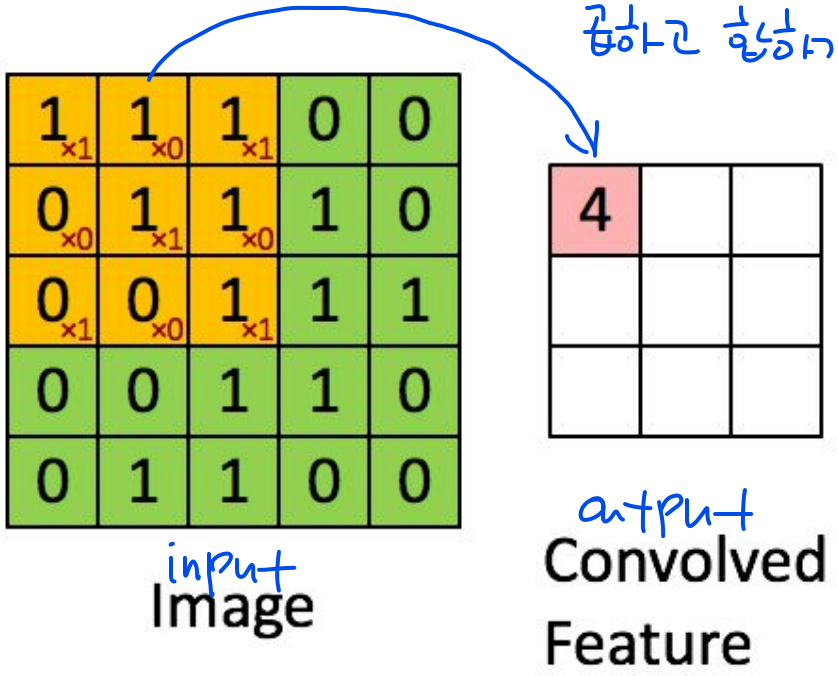

두 개의 레이어 사이에 모든 뉴런을 상호 연결(Fully-connected NN)하는 대신
인접한 픽셀의 작은 패치만 다음 레이어의 뉴런에 연결(Locally-connected NN)합니다

# Locally-Connected NN with **translation invariance**




객체의 존재가 위치에 무관하게 된다. 가중치 공유(weight sharing) - 서로 다른 곳에 연결된 노드들이 모두 같은 가중치를 가진다. 이 가중치는 주어지는 것이 아니라, 훈련자료를 학습함으로 얻어진다.

∴ CNN = 가중치를 공유하는 Convolution NN ⇐ translation invariant locally-connected NN




3x3x1 커널로 5x5x1 이미지를
건컨벌루션하여 3x3x1 결과를 가진다.

- 입력이미지: 녹색 섹션 (5x5x1)
- 커널/필터/가중치: 노란색 (3x3x1)

영상처리...?

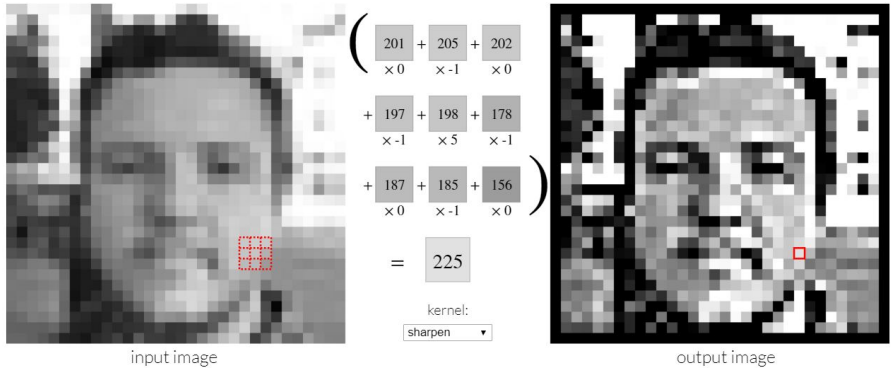
| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

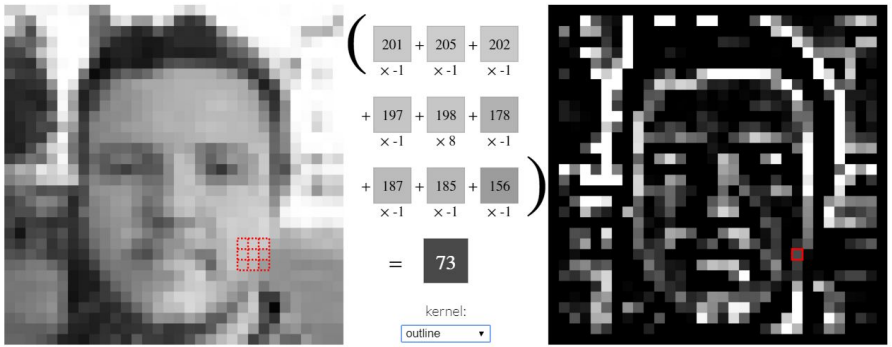
- 컨벌루션 결과 : 핑크색 (3x3x1)

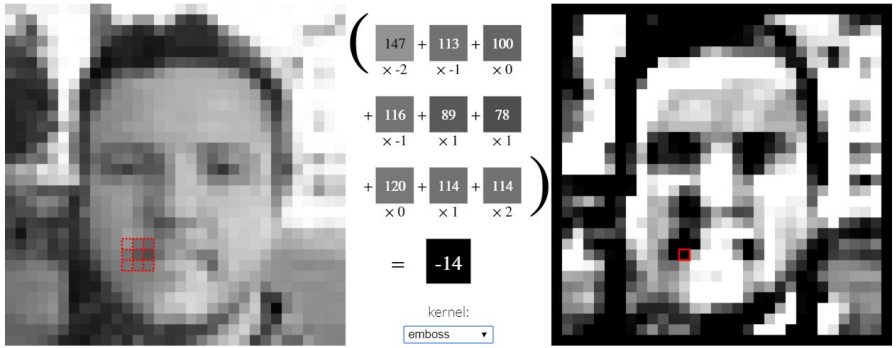
커널은 보폭길이(Stride Length)=1 로 인해 9번 이동합니다. 1칸씩 필터 이동
커널과 커널이 겹쳐지는 이미지의 부분 사이에서 내적 연산을 수행 한다.

# Convolution result of different Kernels ( Link for experiment )

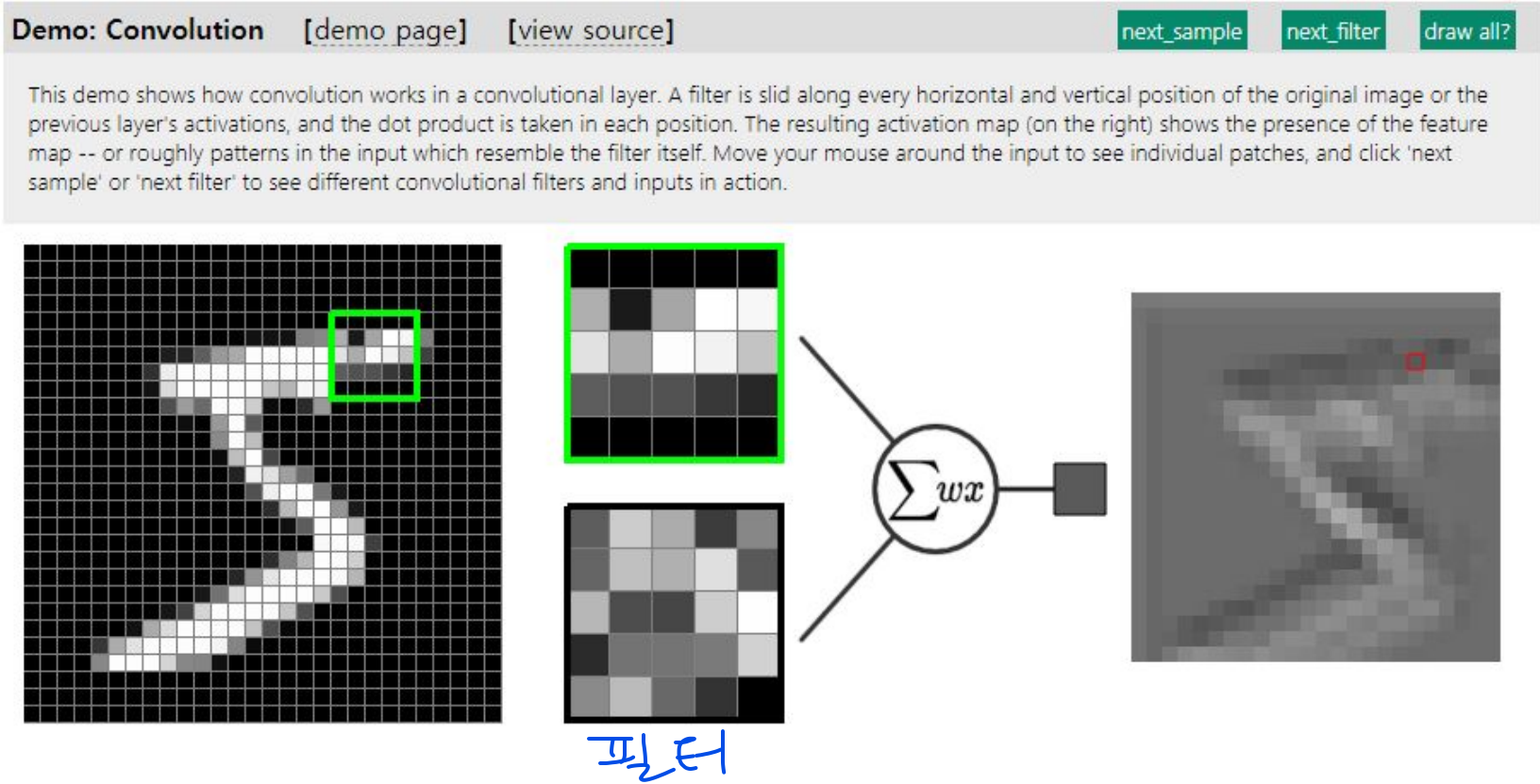
서로 다른 커널은 각각 다른 필터로 작동하여 다른 특징맵을 생성합니다. 예를 들어 위쪽 경계, 아래쪽 경계, 대각선 등을 추출합니다.



# Convolution result of different Kernels ( interactive example Link )



# Convolution NN for color image

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 156 | 155 | 156 | 158 | 158 | ... |
| 0 | 153 | 154 | 157 | 159 | 159 | ... |
| 0 | 149 | 151 | 155 | 158 | 159 | ... |
| 0 | 146 | 146 | 149 | 153 | 158 | ... |
| 0 | 145 | 143 | 143 | 148 | 158 | ... |
| ... | ... | ... | ... | ... | ... | ... |

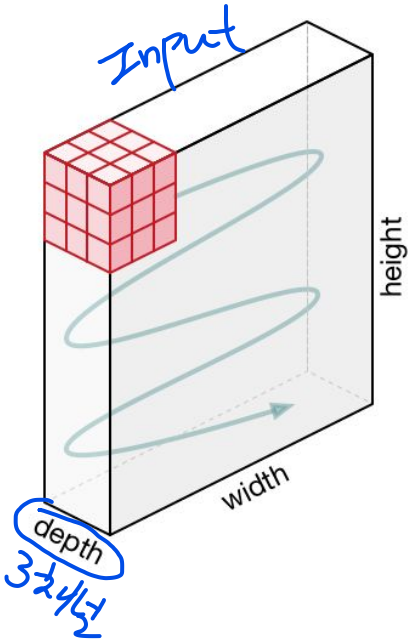
Input Channel #1 (Red)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 167 | 166 | 167 | 169 | 169 | ... |
| 0 | 164 | 165 | 168 | 170 | 170 | ... |
| 0 | 160 | 162 | 166 | 169 | 170 | ... |
| 0 | 156 | 156 | 159 | 163 | 168 | ... |
| 0 | 155 | 153 | 153 | 158 | 168 | ... |
| ... | ... | ... | ... | ... | ... | ... |

Input Channel #2 (Green)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 163 | 162 | 163 | 165 | 165 | ... |
| 0 | 160 | 161 | 164 | 166 | 166 | ... |
| 0 | 156 | 158 | 162 | 165 | 166 | ... |
| 0 | 155 | 155 | 158 | 162 | 167 | ... |
| 0 | 154 | 152 | 152 | 157 | 167 | ... |
| ... | ... | ... | ... | ... | ... | ... |

Input Channel #3 (Blue)




| | | |
|---|---|---|
| -1 | -1 | 1 |
| 0 | 1 | -1 |
| 0 | 1 | 1 |

Kernel Channel #1

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | -1 | -1 |
| 1 | 0 | -1 |

Kernel Channel #2

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | -1 | 1 |

Kernel Channel #3

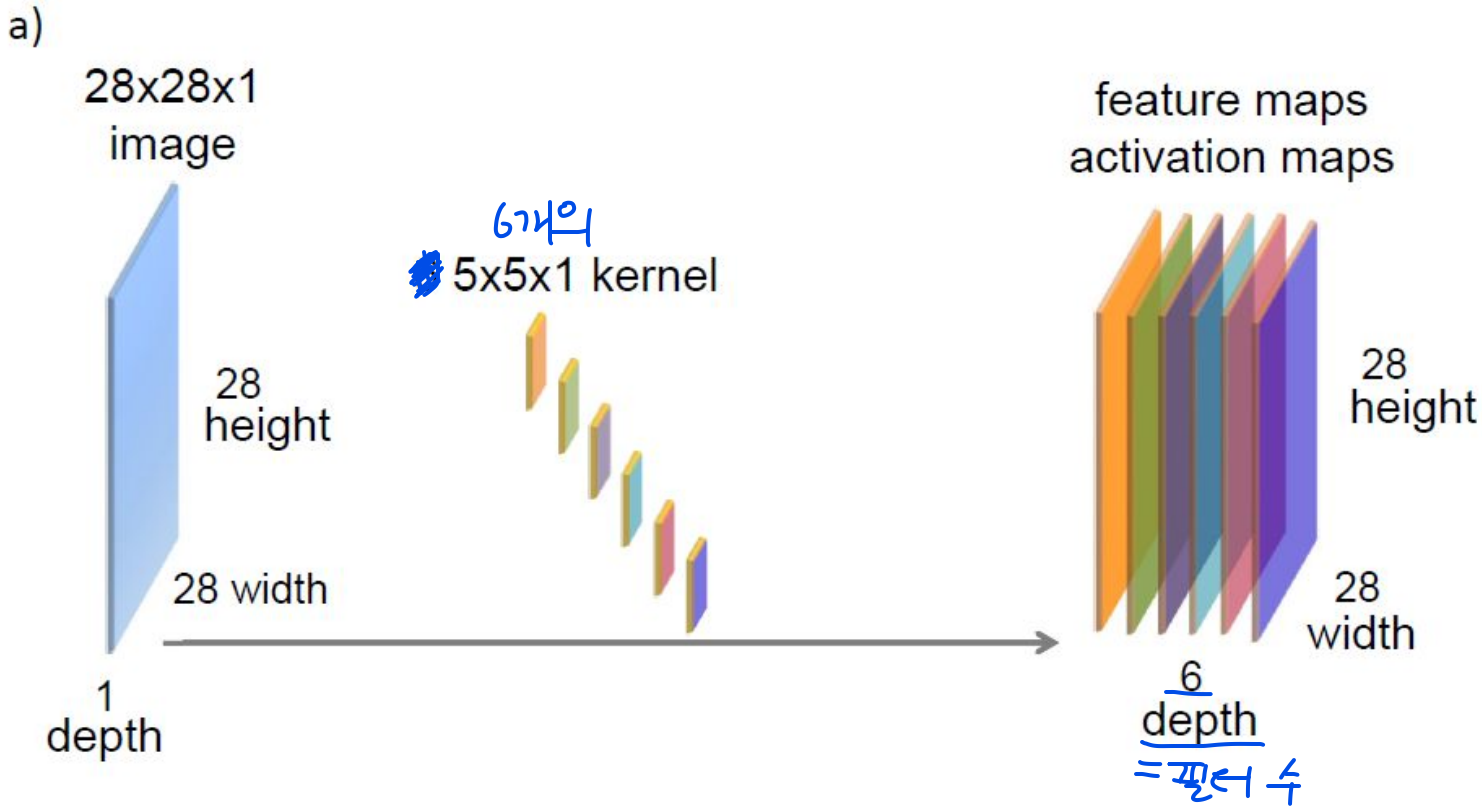
$308 + (-498) + 164 + 1 = -25$

Bias = 1

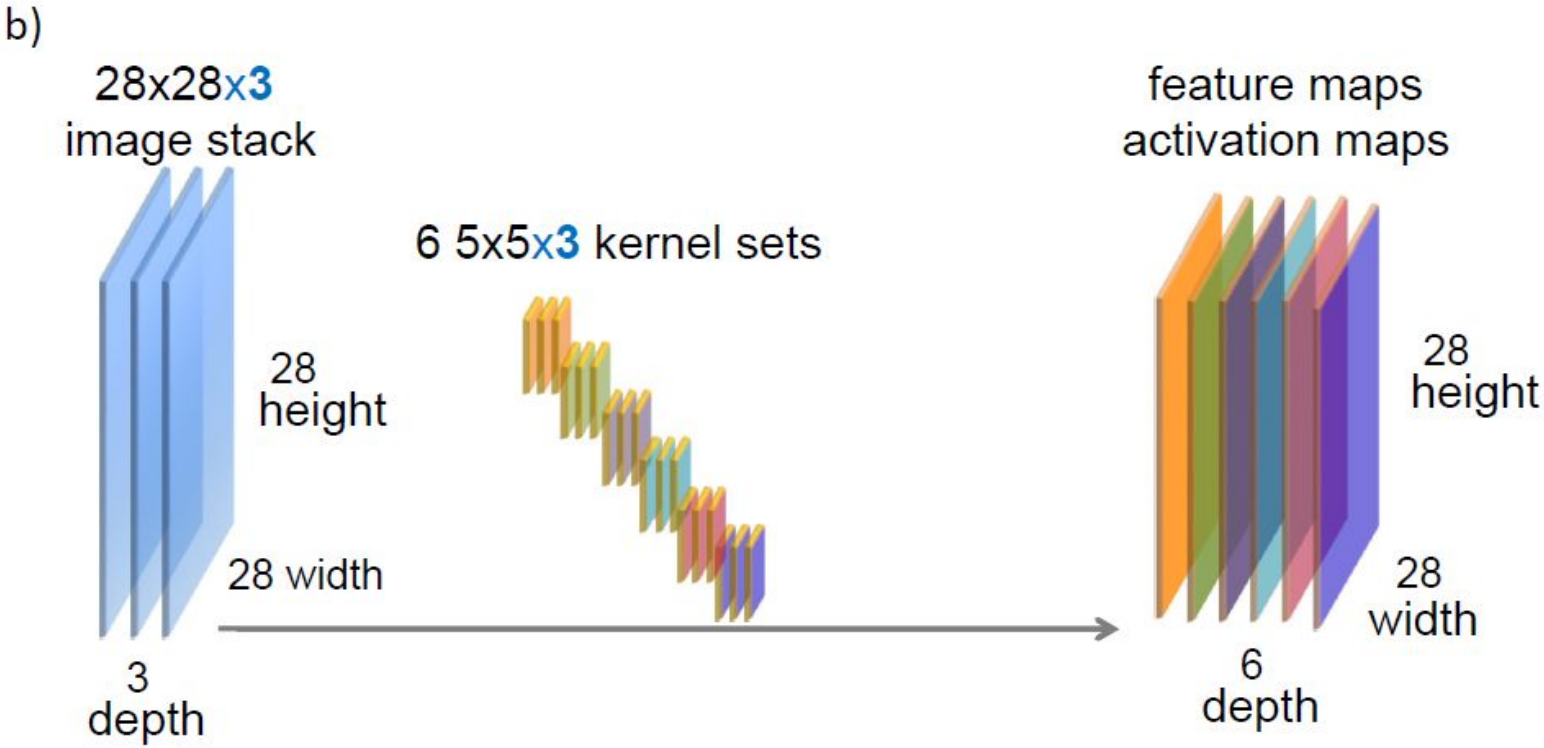
Output

| | | | | |
|---|---|---|---|---|
| -25 | | | | ... |
| | | | | ... |
| | | | | ... |
| | | | | ... |
| ... | ... | ... | ... | ... |

채널이 여러 개인 이미지(예: RGB)의 경우 커널은 입력 이미지와 동일한 깊이를 갖습니다. 필터는 3개

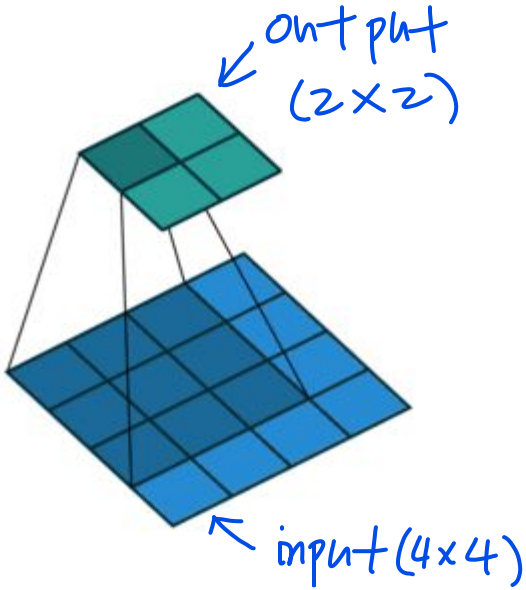



입력 이미지에 채널이 하나만 있는 경우, 6 개의 다른 커널로 컨볼루션하면 6 개의 활성화 맵이 생성됩니다.

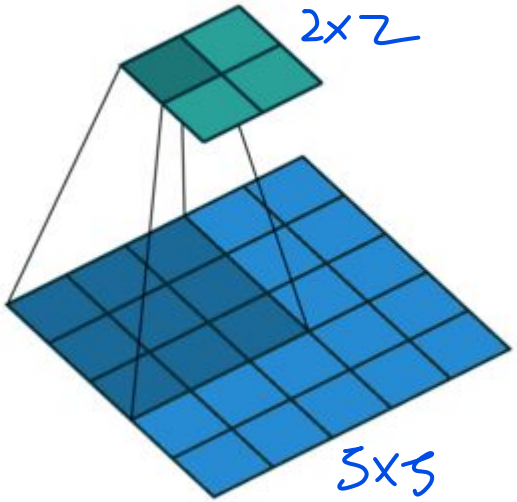



입력 이미지에 3 개의 채널이 있으면 각 커널/필터에도 3 개의 채널이 있습니다.
6 개의 다른 커널로 컨볼루션하면 6 개의 활성화 맵이 생성됩니다.

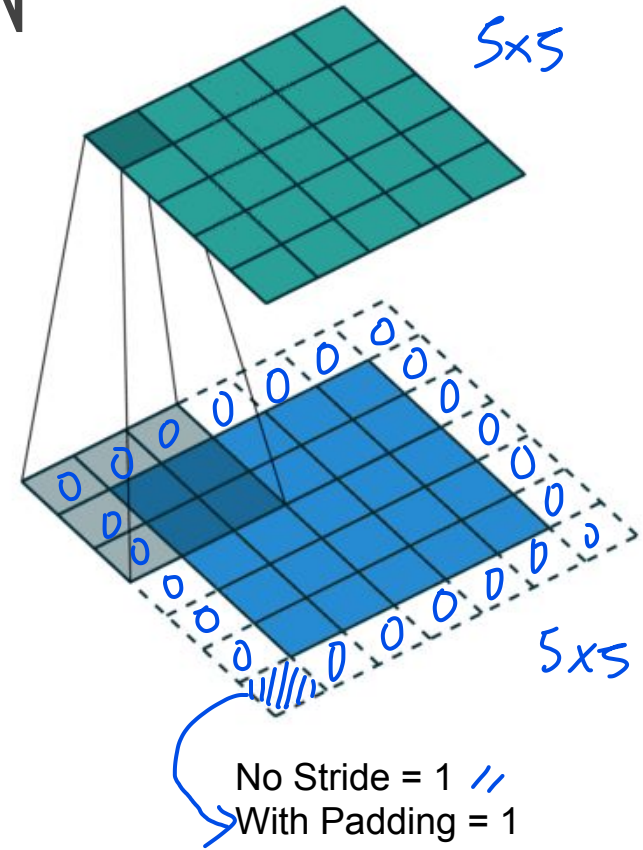
# 보폭(Stride) and 패딩(Padding) in CNN




No Stride(보폭) = 1 칸씩 이동
No Padding = 0

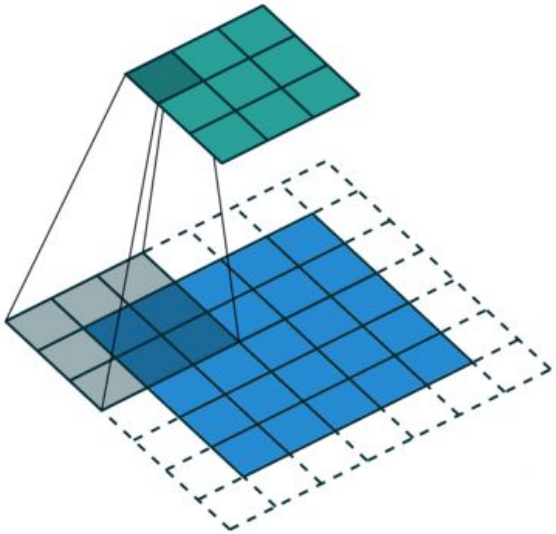



With Stride = 2 //
No Padding = 0

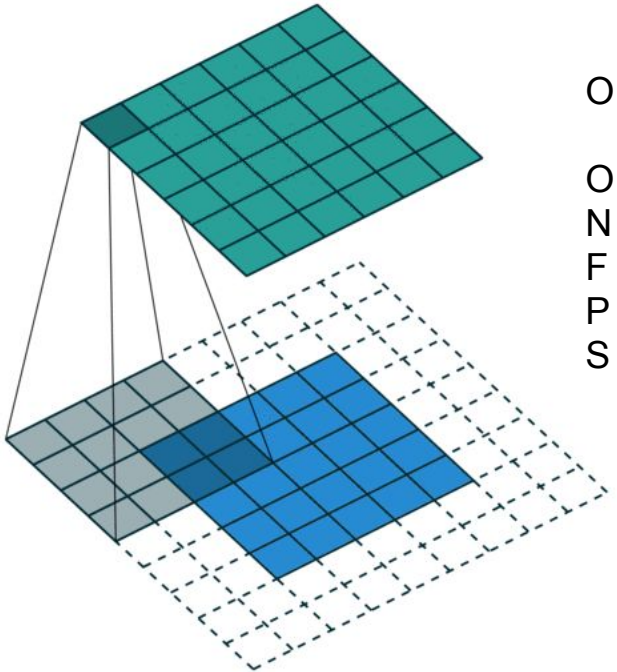



No Stride = 1 //
With Padding = 1

# 보폭(Stride) and 패딩(Padding) in CNN



With Padding = 1, 보폭(Stride Length) = 2 인 경우 5x5 이미지에 대한 콘벌루션 결과가 3x3 이미지로 된다.



With Padding = 2, 보폭(Stride Length) = 1 인 경우 5x5 이미지에 대한 콘벌루션 결과가 6x6 이미지가 된다.

$$O = (N - F + 2P) / S + 1$$

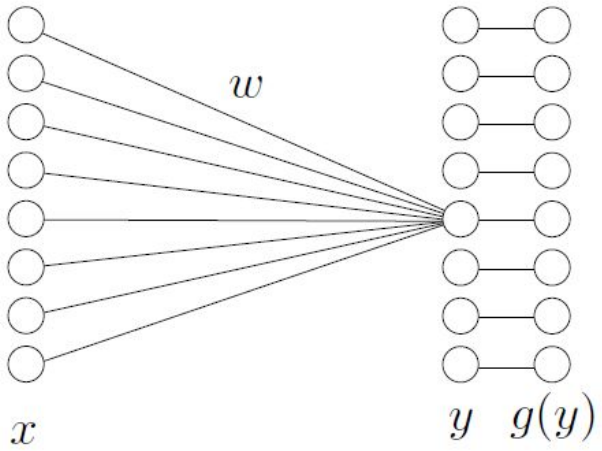
O : Size of Output
N : Size of Input
F : Size of Filter
P : Size of Padding
S : Size of Stride

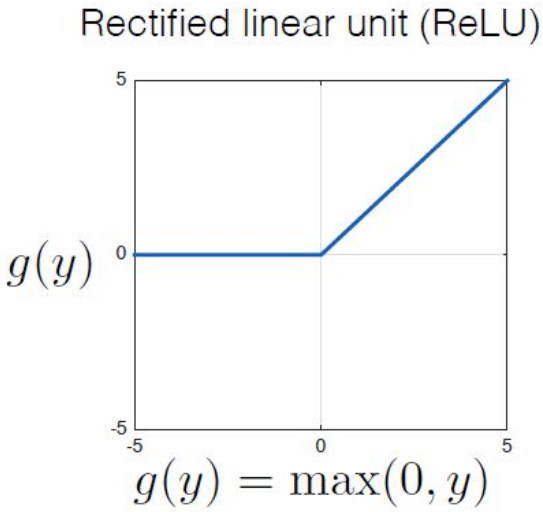
# 활성함수 : ReLU(Rectified Linear Unit) in CNN

일종의 심층신경망에서 ReLU로 사라지는 미분치 문제 극복

CNN은 모델에 비선형성을 주기 위해 컨볼루션 작업이 끝날 때마다 합성곱된 특성에 Rectified Linear Unit (ReLU) 변형을 활성함수로 적용합니다.

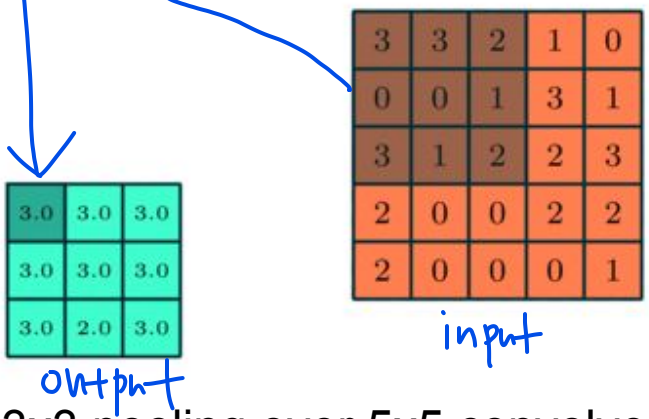
g(y) = max(0,y) 로 정의되는 ReLU 함수는 y > 0인 경우에 대해 y 를, y ≤ 0인 경우에 대해 0 을 반환합니다



컨볼루션의 결과에 적용하는 것

# Pooling in CNN

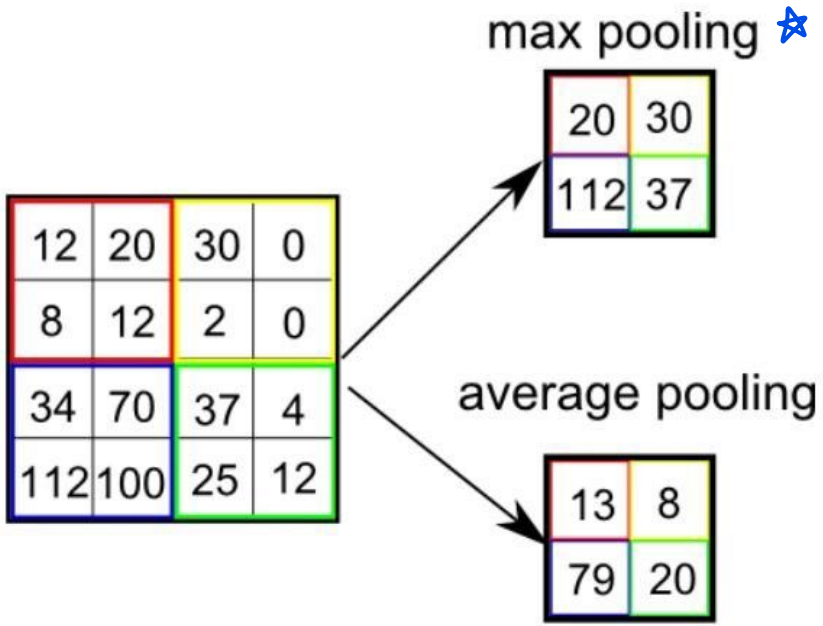



3x3 pooling over 5x5 convolved feature

- Pooling 레이어는 컨벌루션된 특징 (Convolved Feature)의 **공간 크기를 줄이는 역할**을 합니다.
- 차원 축소를 통해 데이터를 처리하는 데 필요한 **계산량을 감소**시키기 위함.
- 회전 및 위치 불변인 **지배적 특징을 추출**하여 모델의 효과적인 훈련 과정을 유지하는 데 유용합니다.

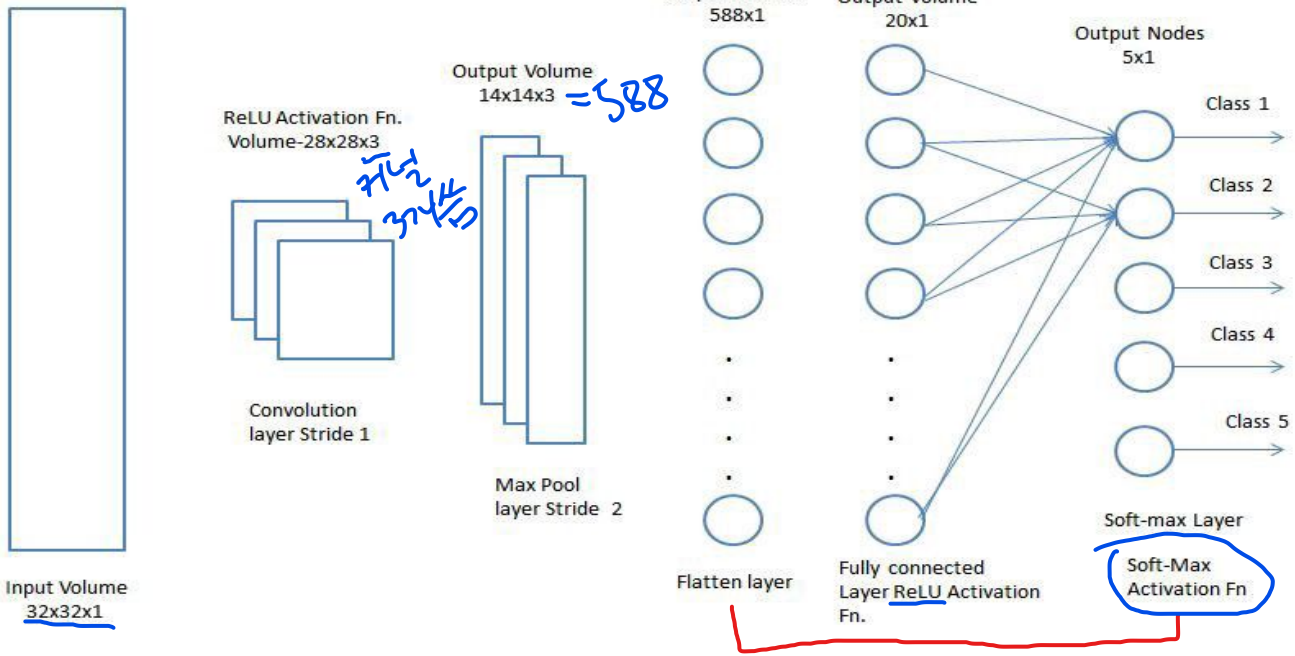
- **최대(Max) 풀링**과 **평균(Average) 풀링**의 두 가지 유형이 있습니다.
  최대 풀링은 커널에서 다루는 이미지 부분에서 최대 값을 반환합니다.
  평균 풀링은 커널에서 다루는 이미지 부분에서 모든 값의 평균을 반환합니다.

# Pooling in CNN



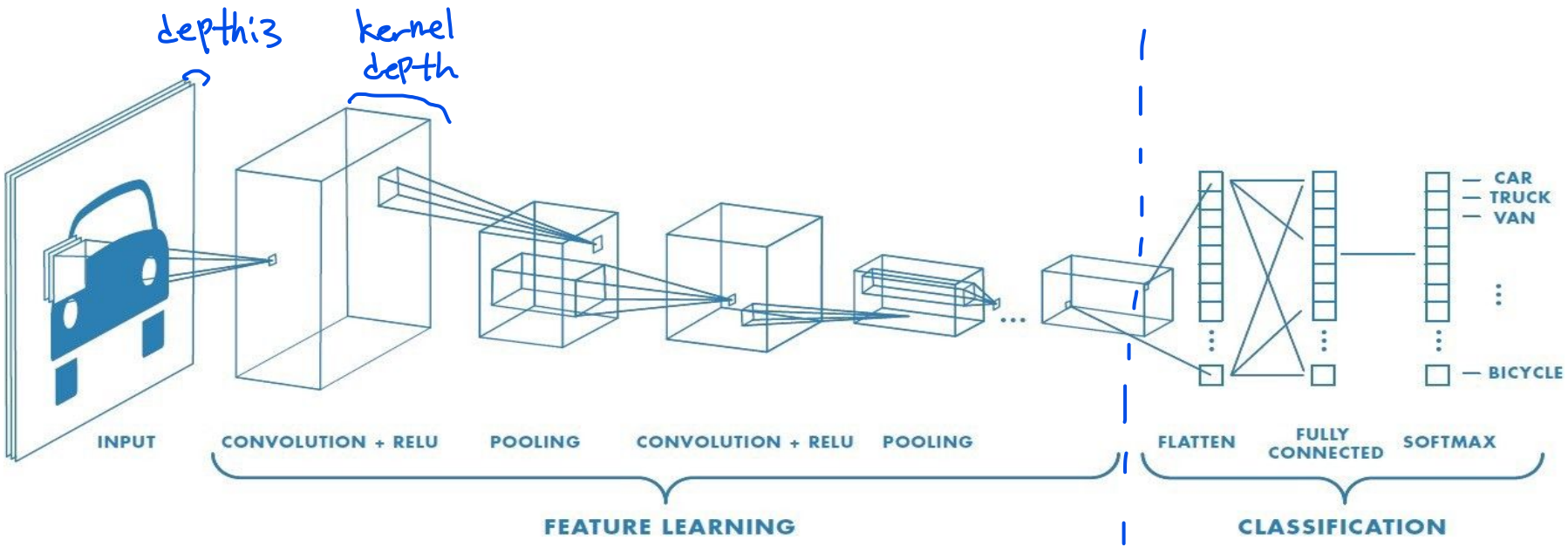

- **Max Pooling**은 **노이즈 억제 기능**도 합니다. **차원 축소**와 함께 블록에서 낮은 활성화를 버리고 최대 활성화를 취함으로써 **가장 중요한 특징 추출**을 수행합니다.
- 한편, 평균 풀링은 **단순히 노이즈 억제** 메커니즘으로 **차원 축소**를 수행합니다.
- 따라서 Max Pooling은 Average Pooling보다 훨씬 뛰어납니다.

# Classification : Fully-connected NN

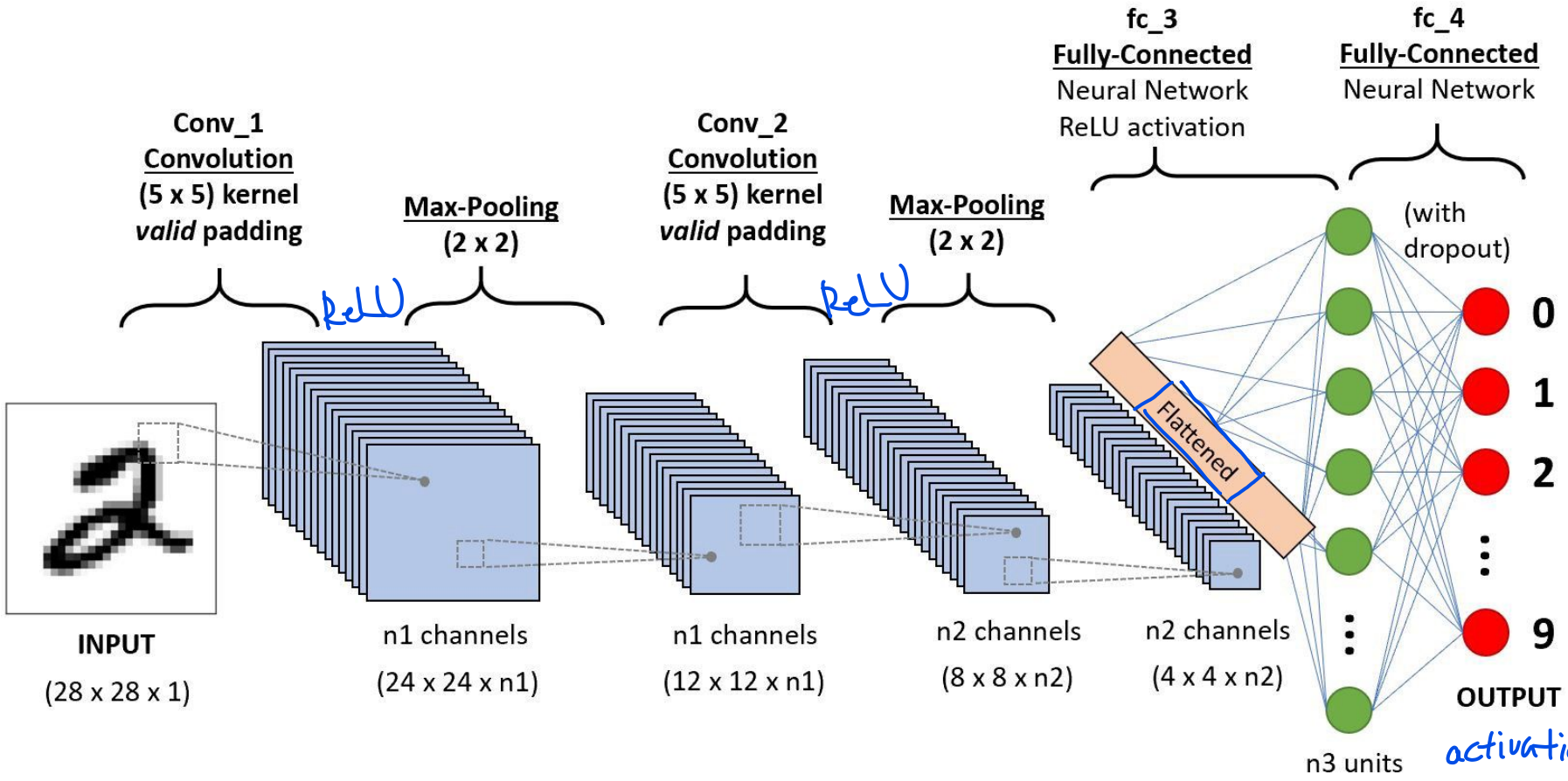



컨벌루션과 Poolling 과정을 거친 후, 모델이 특징들을 이해하도록 했으며,
계속해서 최종 출력을 평평하게 하고 분류 목적으로 일반 신경망에 공급합니다.

# Typical CNN Architecture




CNN Visualization by Otavio Good https://youtu.be/f0t-OCG79-U




A CNN sequence to classify handwritten digits

# Handwritten digit recognition with CNNs (Link to Example)

이 튜토리얼은 MNIST 숫자를 분류하기 위해 간단한 합성곱 신경망(Convolutional Neural Network, CNN)을 훈련합니다. 간단한 이 네트워크는 MNIST 테스트 세트에서 99% 정확도를 달성할 것입니다. 이 튜토리얼은 케라스 Sequential API를 사용하기 때문에 몇 줄의 코드만으로 모델을 만들고 훈련할 수 있습니다.

노트: GPU를 사용하여 CNN의 훈련 속도를 높일 수 있습니다. 코랩에서 이 노트북을 실행한다면 * 수정 -> 노트 설정 -> 하드웨어 가속기* 에서 GPU를 선택하세요.

### 1. 텐서플로 임포트하기

```
%tensorflow_version 2.x
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
print(tf.__version__)
```

TensorFlow 2.x selected.

2.1.0

## 2. MNIST 데이터셋 다운로드하고 준비하기

```
(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 정규화합니다.
train_images, test_images = train_images /  255.0, test_images / 255.0
```

28X28X1 60000개

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

11493376/11490434 [==============================] - 0s 0us/step

### 3. 합성곱 층 만들기 컨볼루션 레이어 만들기

아래 6줄의 코드에서 Conv2D와 MaxPooling2D 층을 쌓는 일반적인 패턴으로 합성곱 층을 정의합니다.

CNN은 배치(batch) 크기를 제외하고 (이미지 높이, 이미지 너비, 컬러 채널) 크기의 텐서(tensor)를 입력으로 받습니다. MNIST 데이터는 (흑백 이미지이기 때문에) 컬러 채널(channel)이 하나지만 컬러 이미지는 (R,G,B) 세 개의 채널을 가집니다. 이 예에서는 MNIST 이미지 포맷인 (28, 28, 1) 크기의 입력을 처리하는 CNN을 정의하겠습니다. 이 값을 첫 번째 층의 input_shape 매개변수로 전달합니다.

컨볼루션 레이어 1개

```
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'))
```

커널수 커널크기

Fully Connected 층

지금까지 모델의 구조를 출력해 보죠.

```
model.summary()
```

```
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 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

옆 에서 Conv2D와 MaxPooling2D 층의 출력은 (높이, 너비, 채널) 크기의 3D 텐서입니다. 높이와 너비 차원은 네트워크가 깊어질수록 감소하는 경향을 가집니다. Conv2D 층에서 출력 채널의 수는 첫 번째 매개변수에 의해 결정됩니다 (예를 들면, 32 또는 64). 일반적으로 높이와 너비가 줄어듦에 따라 (계산 비용 측면에서) Conv2D 층의 출력 채널을 늘릴 수 있습니다.

### 4. 마지막에 Dense 층 추가하기

모델을 완성하려면 마지막 합성곱 층의 출력 텐서(크기 (3, 3, 64))를 하나 이상의 Dense 층에 주입하여 분류를 수행합니다. Dense 층은 벡터(1D)를 입력으로 받는데 현재 출력은 3D 텐서입니다. 먼저 3D 출력을 1D로 펼치겠습니다. 그다음 하나 이상의 Dense 층을 그 위에 추가하겠습니다. MNIST 데이터는 10개의 클래스가 있으므로 마지막에 Dense 층에 10개의 출력과 소프트맥스 활성화 함수를 사용합니다.

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

최종 모델의 구조를 확인해 보죠.

```
model.summary()
```

```
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 (MaxPooling2 (None, 5, 5, 64)          0
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

여기에서 볼 수 있듯이 두 개의 Dense 층을 통과하기 전에 (3, 3, 64) 출력을 (576) 크기의 벡터로 펼쳤습니다.

## 5. 모델 컴파일과 훈련하기

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

model.fit(train_images, train_labels, epochs= 5)
```

Train on 60000 samples

Epoch 1/5

60000/60000 [==============================] - 12s 202us/sample - loss: 0.4771 - accuracy: 0.8474

Epoch 2/5

. . .

Epoch 5/5

60000/60000 [==============================] - 6s 95us/sample - loss: 0.0264 - accuracy: 0.9927

<tensorflow.python.keras.callbacks.History at 0x7fdec0407780>

## 6. 모델 평가

```
test loss, test_acc = model.evaluate(test_images,  test_labels, verbose= 2)
print(test_acc)
```

0.987

결과에서 보듯이 간단한 CNN 모델이 99%의 테스트 정확도를 달성합니다. 몇 라인의 코드치고 나쁘지 않네요! (케라스의 서브클래싱 API와 GradientTape를 사용하여) CNN을 만드는 또 다른 방법은 여기를 참고하세요.