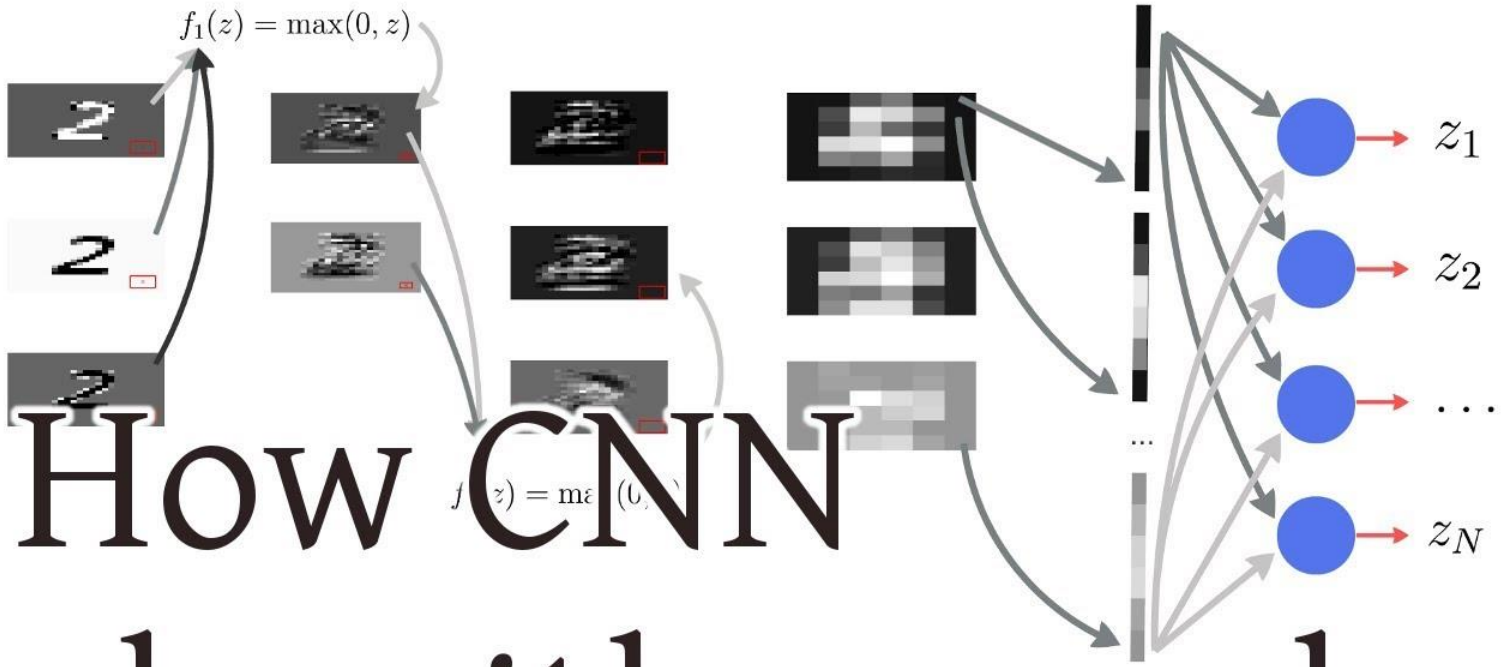
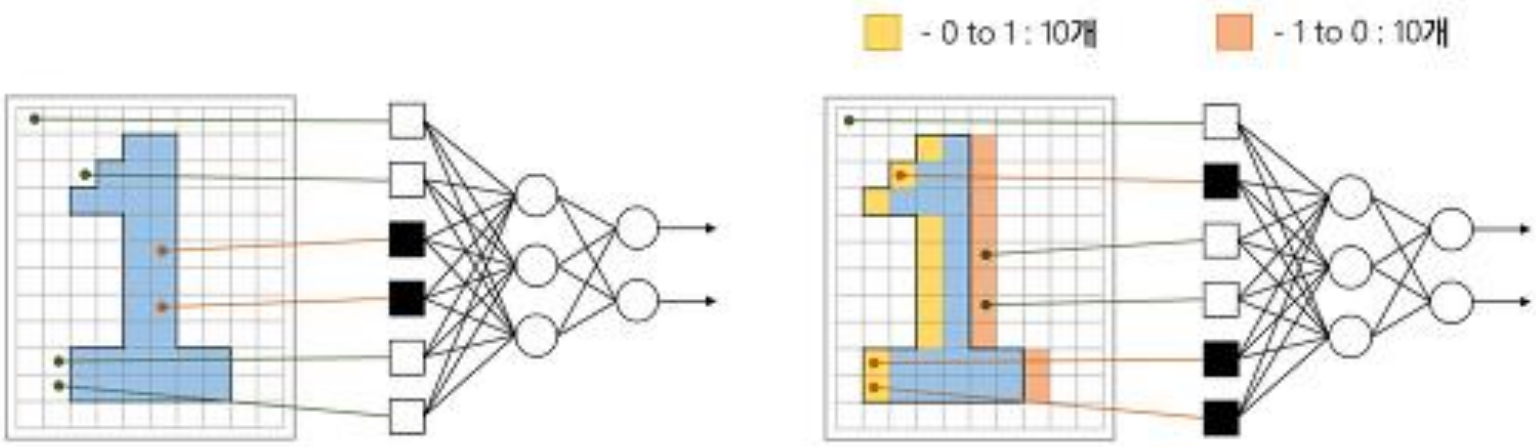
CNN 알고리즘



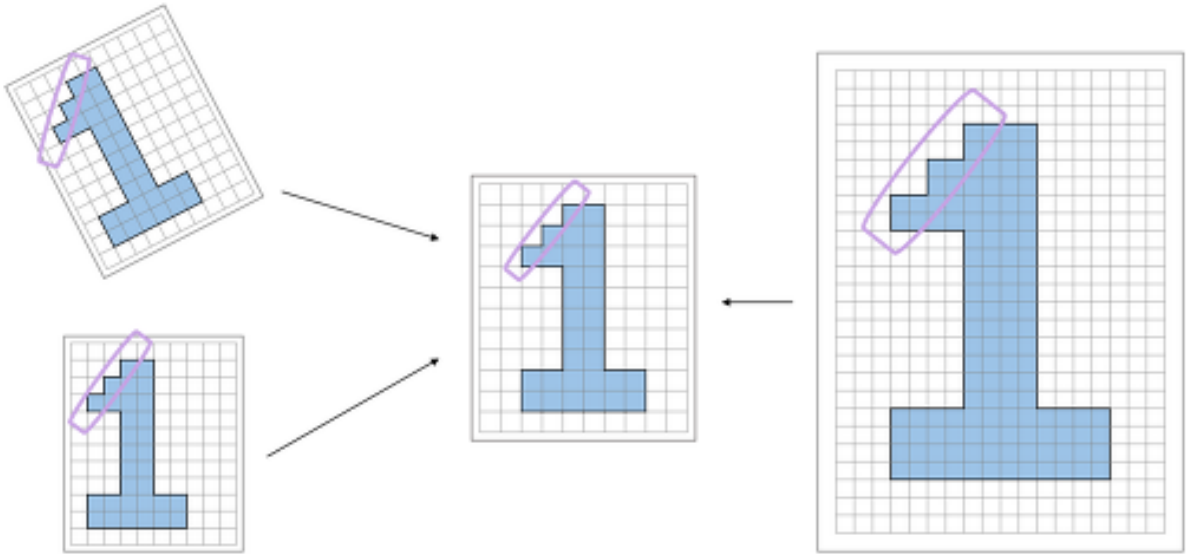
How CNN algorithm works

박경규

MLP의 문제점



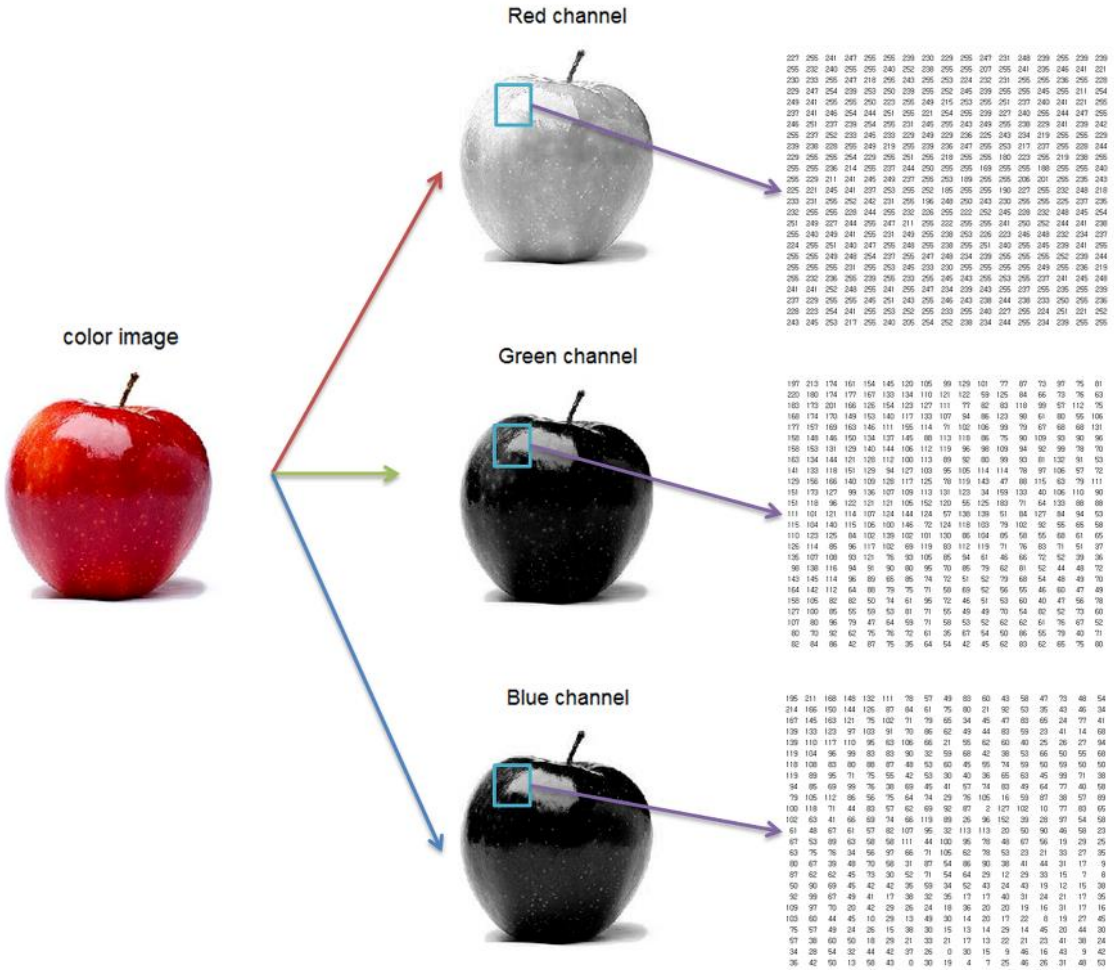
한 칸씩만 움직였는데
변화하는 인풋값이 20개



https://gomguard.tistory.com/188?category=712467

이미지 데이터

- 컬러 이미지는 3개의 채널로 표현됩니다.
- 3개의 채널은 Red, Green, Blue 3원색입니다.
- 각 채널은 0~255사이의 값으로
  빨강의 정도,
  녹색의 정도,
  파랑의 정도를 각각 나타냅니다.

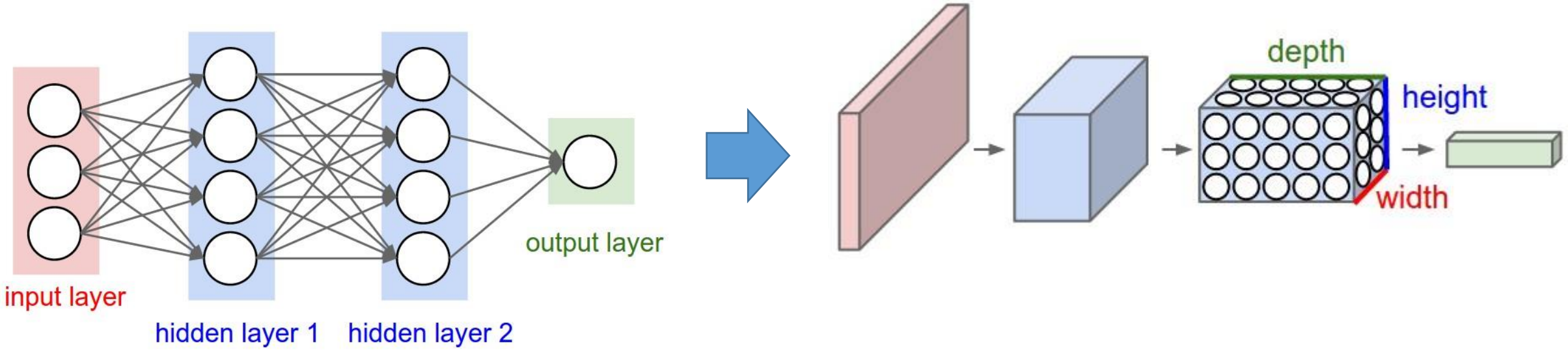


- 2차원 구조인 이미지 데이터를 1차원 숫자 배열로 바꾸면 이미지의 시각적 구조가 상당히 유실 됩니다.
- 200×200 픽셀 RGB 이미지의 경우에, 밀집층의 뉴런 하나에 12만1개의 파라미터가 필요하고,
  64개의 뉴런을 가진 밀집층에서의 가중치 개수는 760만개가 되며 계산복잡도가 증가합니다.

이미지 높이(픽셀수)	이미지 너비(픽셀수)	채널수	Neuron 당 weight 개수	bias 개수	Neuron 당 파라미터 개수	Layer 개수	Layer 당 파라미터 개수
200	200	3	120,000	1	120,001	64	7,680,064

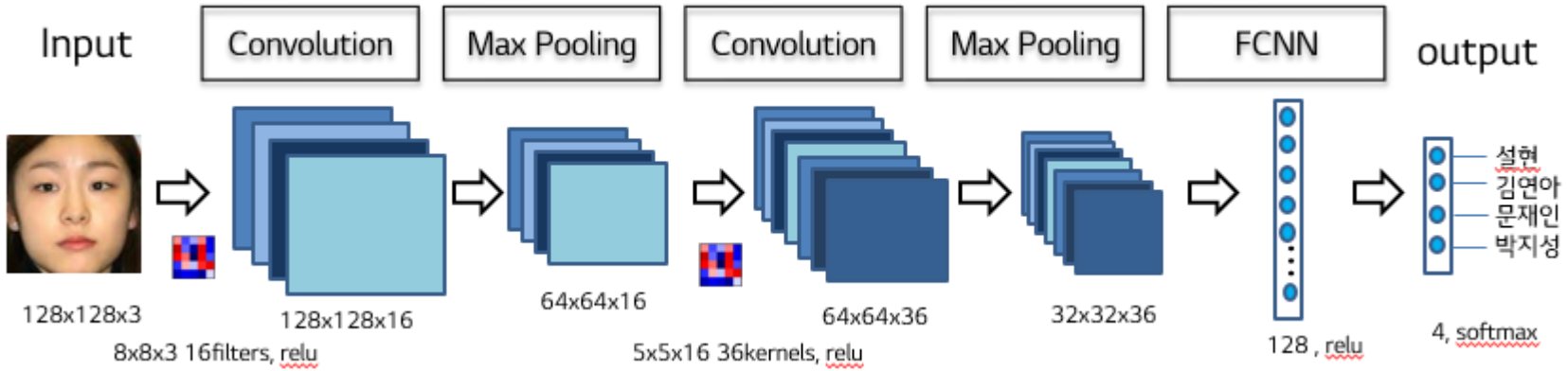
이미지 데이터 처리

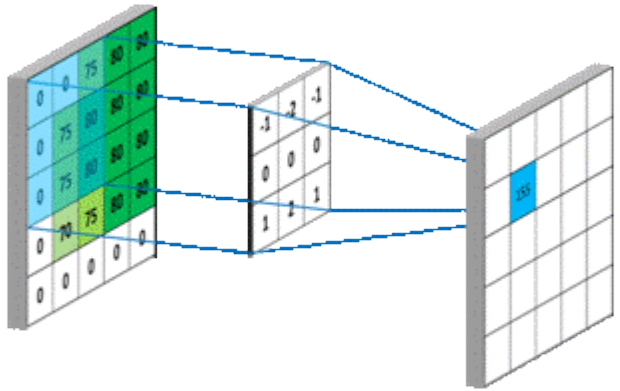
- 200x200x3 크기 이미지는 첫 번째 hidden layer의 1개의 뉴런에 대해 200x200x3 + 1 =120,001개의 가중치를 필요로 하여, MLP는 이미지를 다루기에 적합하지 않습니다.
- ConvNet은 입력이 이미지로 이뤄져 있다는 특징을 살려 좀 더 합리적인 방향으로 아키텍처를 구성하였습니다.
- MLP와 CNN의 가장 큰 차이점은 이미지 Feature 추출방법입니다.
  - MLP는 이미지의 픽셀 값을 Input으로 사용하는 것이고,
  - CNN은 이미지의 Region Feature를 Convolution Layer와 Pooling Layer를 이용해 추출하고 그 Feature를 MLP의 Input으로 사용하는 것입니다.
- CNN이 Computer Vision에서 성능이 좋은 이유는 Region Feature를 추출할 수 있기 때문입니다.



CNN(Convolutional Neural Network)

- CNN은 두뇌의 시각 피질 메커니즘에 영감을 받아 설계된 이미지, 영상등을 인식하는 신경망 모델입니다.
- Convolution Layer(합성곱 층)은 여러 개의 필터로 구성되며, 필터는 이미지를 왼쪽 위에서 오른쪽 아래까지 스캔 하는 작은 윈도우입니다.
- 필터가 각 위치에서 합성곱 연산(Convolution Operation)을 수행합니다.
- 필터는 밀집층과 마찬가지로 역전파로 학습되는 가중치로 구성됩니다.
- 필터의 일반적인 크기는 3×3 이며, 크기는 다양하게 설정합니다.

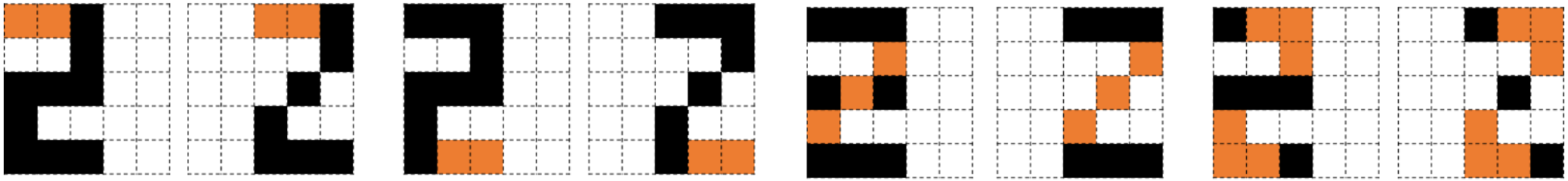




CNN 모델 개념

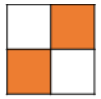
CNN은 뇌가 사물을 구별하듯 생김새 정보로 사물을 학습하고 구별해 낸다.

■ 숫자 2에서 공통적으로 얻을 수 있는 생김새 정보

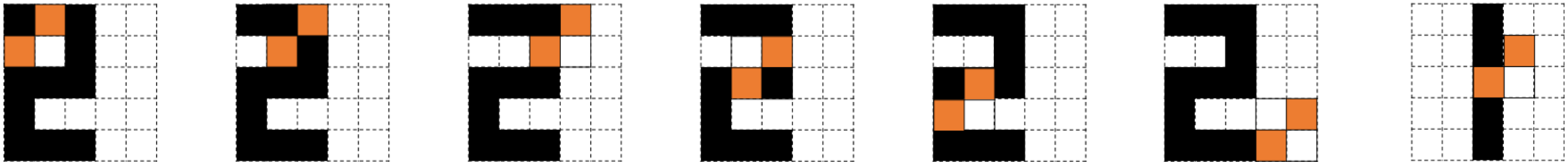


머리 꼬리 이음새 모서리

■ CNN은 어떻게 특징을 찾아 내는가?



필터(커널)



대각선 필터는 숫자 2로부터 두 곳의 대각선 특징을 감지하지만, 숫자 1에서는 대각선 특징을 발견하지 못한다.

CNN 모델 개념

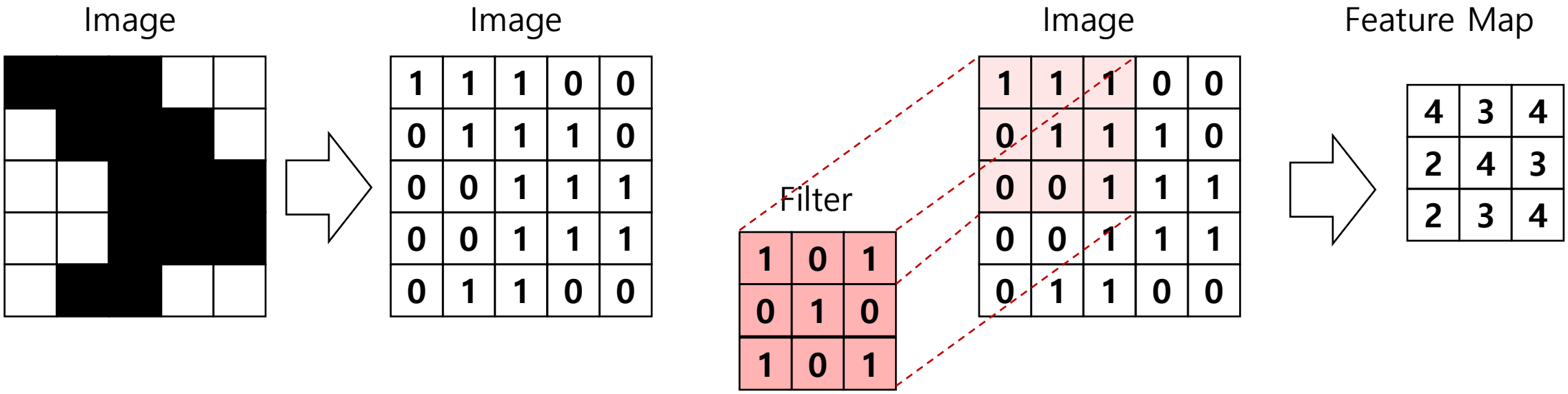
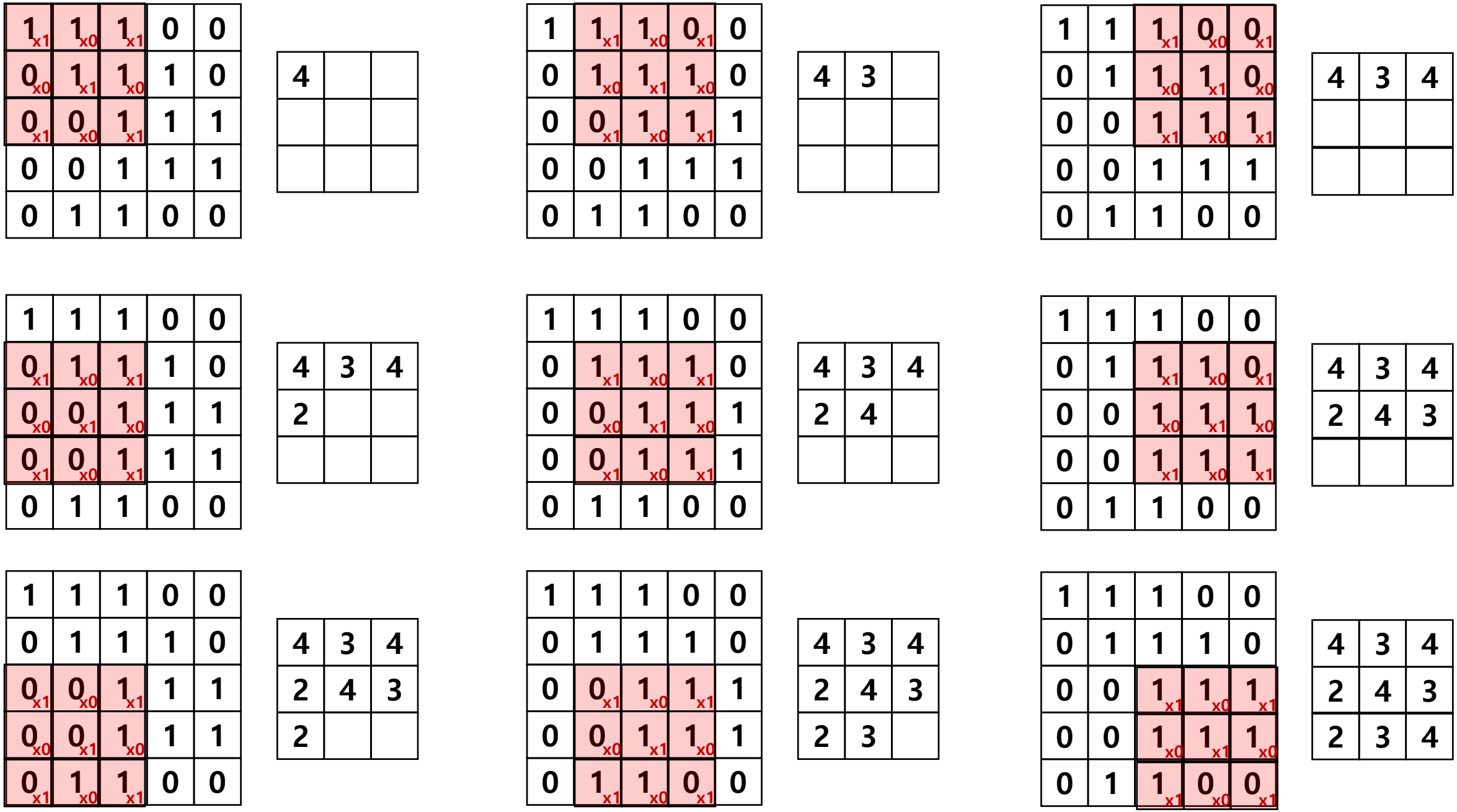


Image
Image
Image
Feature Map
Filter
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
1 0 1
0 1 0
1 0 1
4 3 4
2 4 3
2 3 4


1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
2
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
2 4
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
2 4 3
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
2 4 3
2
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
2 4 3
2 3
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 1
0 1 1 0 0
4 3 4
2 4 3
2 3 4

Padding

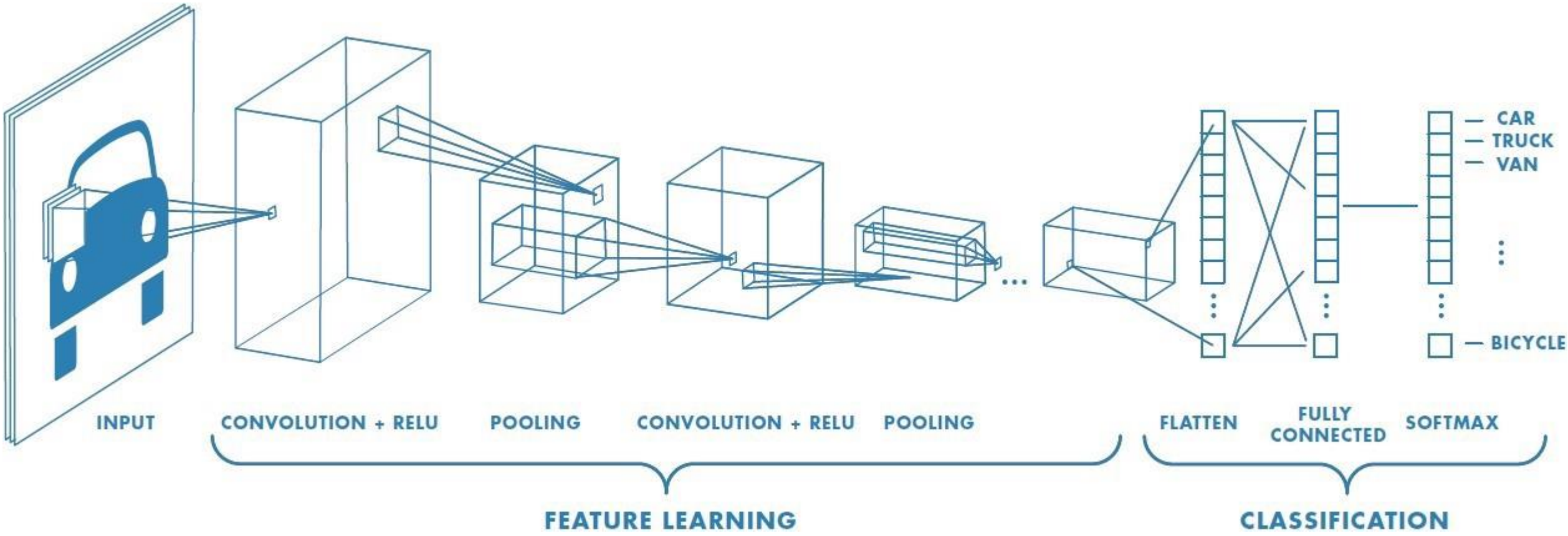
패딩(padding)을 1만큼 적용한 이미지 데이터 행렬

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

Pooling

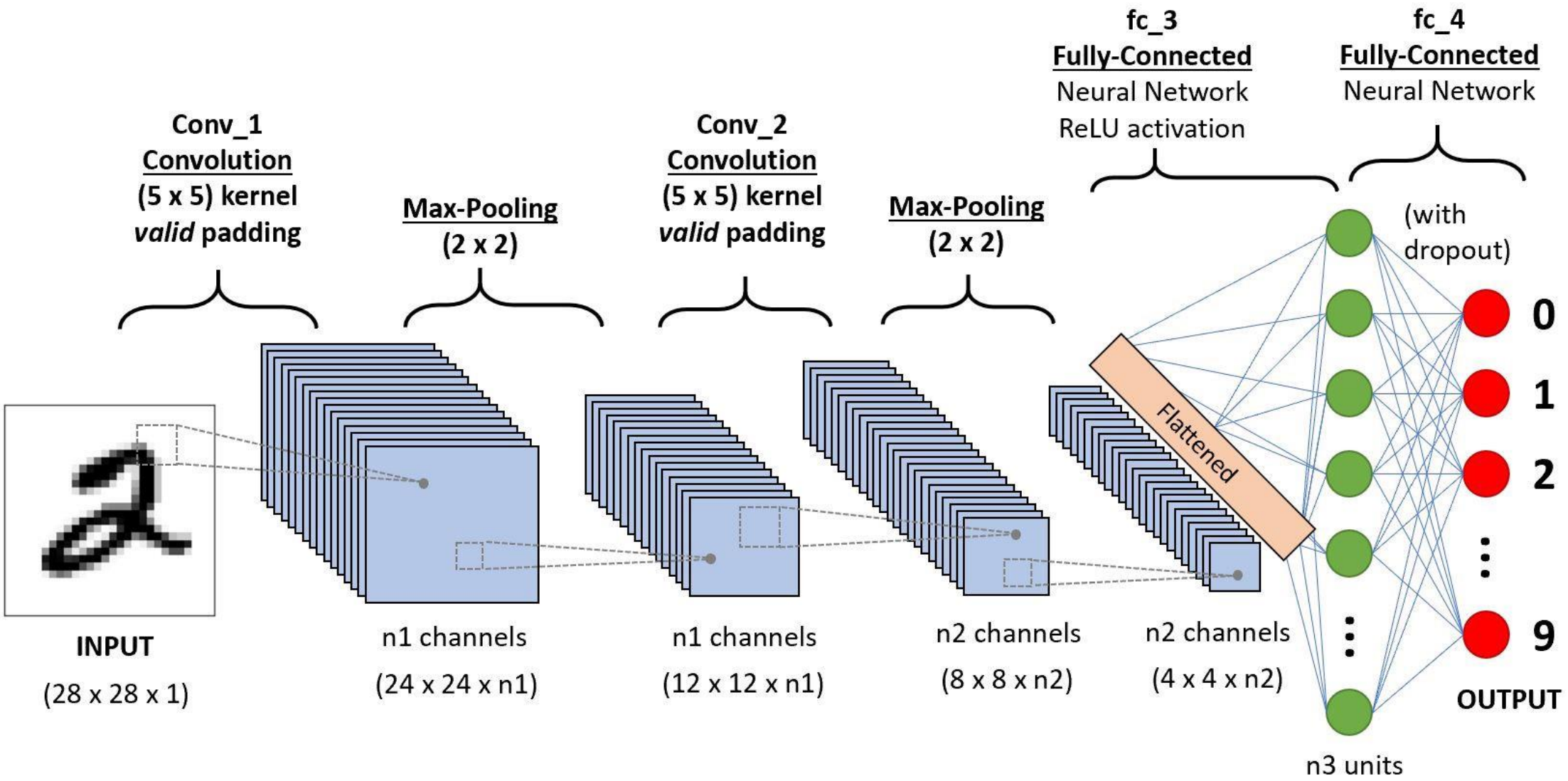
1	4	**8**	3
6	**9**	2	1
11	13	6	7
8	**19**	**8**	2

CNN



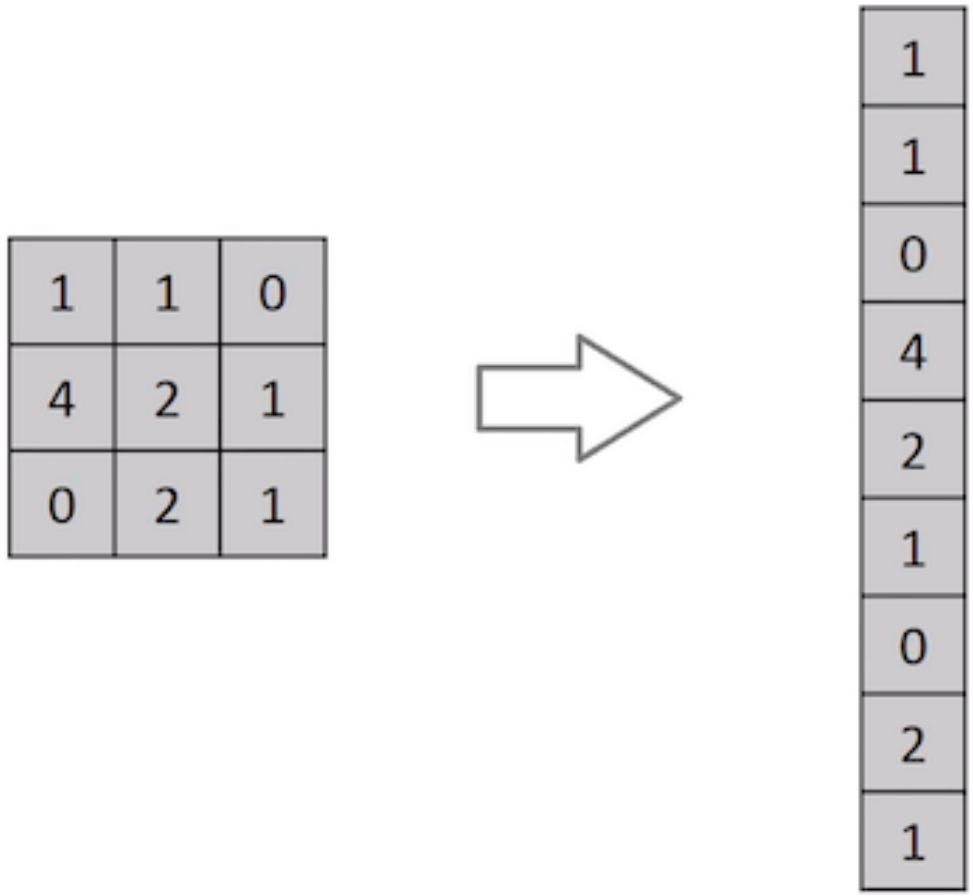
https://bit.ly/2RVDoFI

CNN



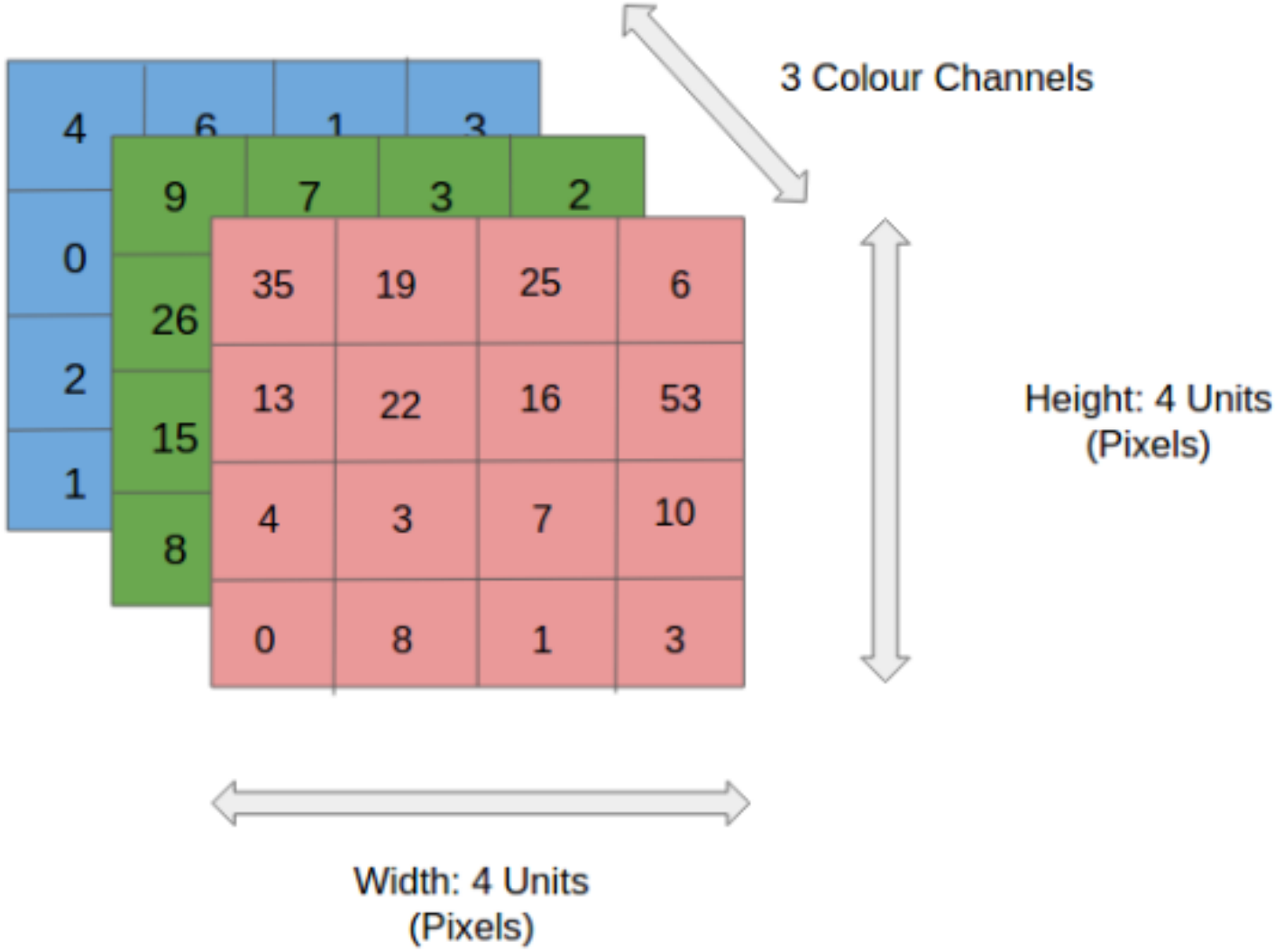
https://bit.ly/2RVDoFI

ConvNet 필요 이유



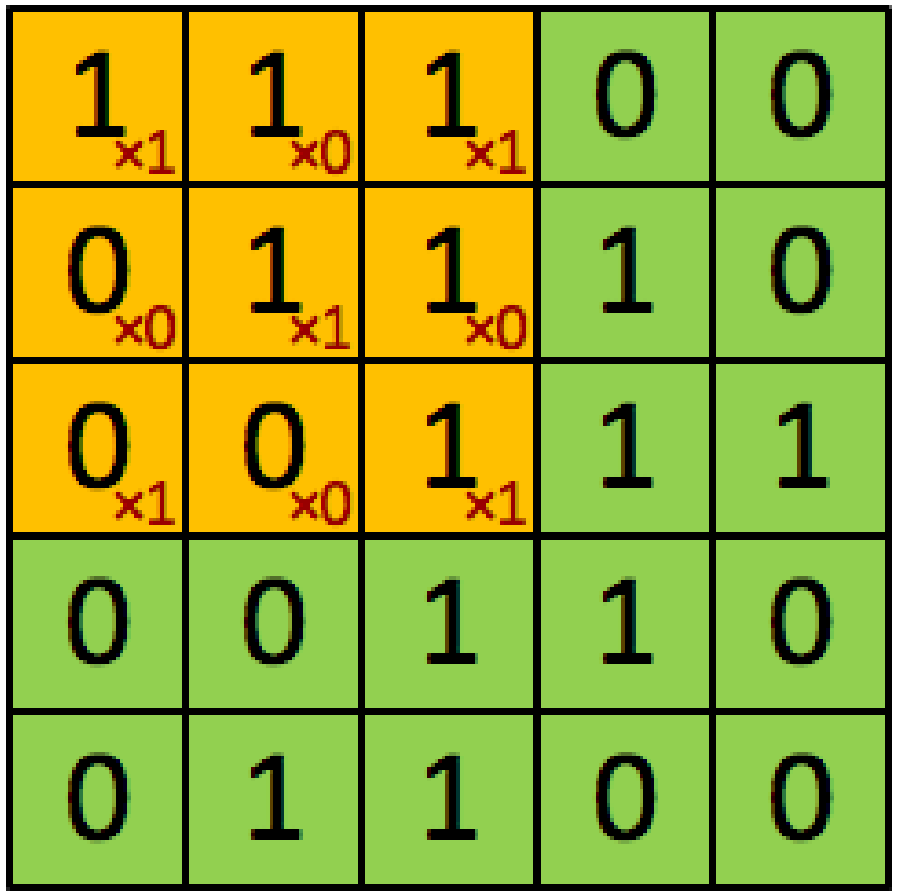
https://bit.ly/2RVDoFI

Input Image

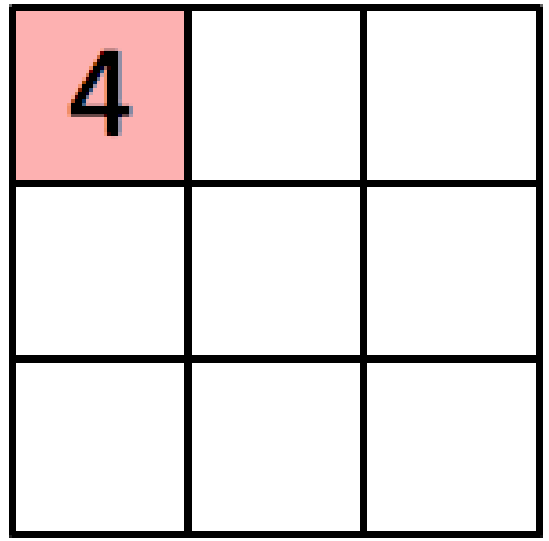


https://bit.ly/2RVDoFI

Convolution Layer - Kernel



Image



Convolved Feature

https://bit.ly/2RVDoFI

Convolution Layer - Kernel

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

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

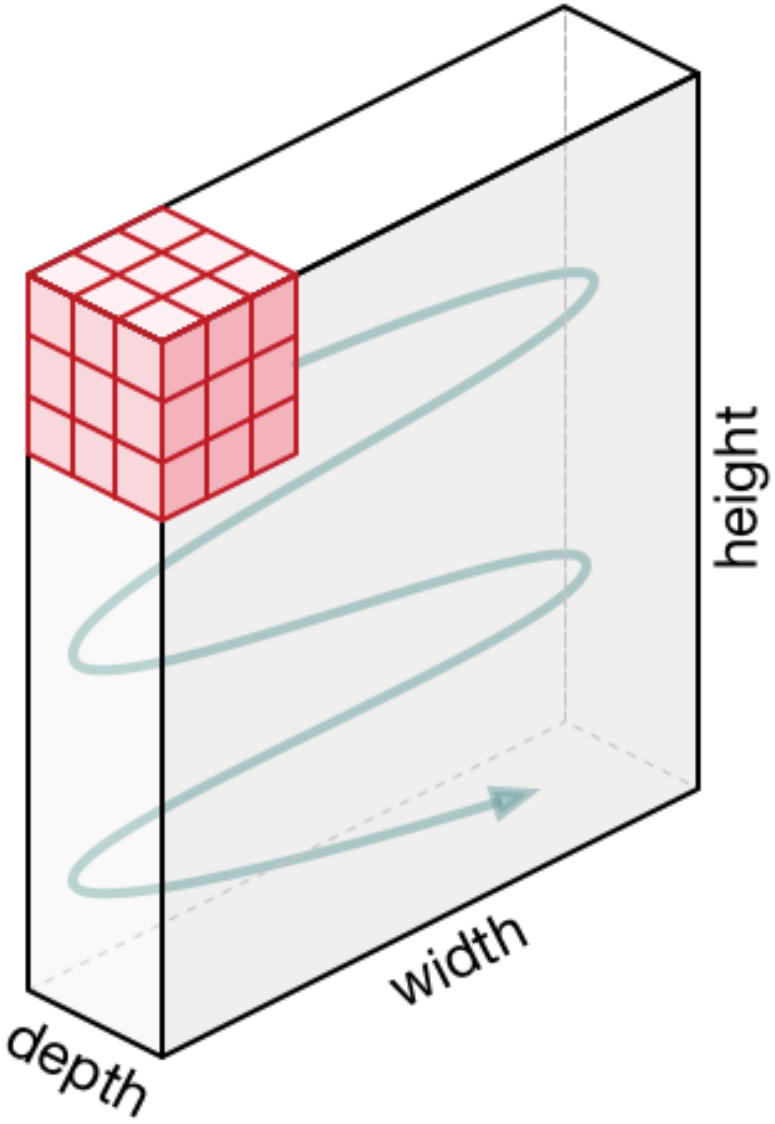
Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

https://bit.ly/2RVDoFI

Convolution Layer - Kernel



합성곱 필터 하이퍼파라미터

■ 필터(커널) 크기

- 머신비전에서 3 x 3, 5 x 5, 7 x 7 필터크기가 효과적이라고 알려져 있습니다.
- 이미지에 비해 필터가 너무 크면, 수용장(receptive field)에서 많은 특성이 과도하게 경쟁하기 때문에 합성곱 층이 효과적으로 학습하기 어렵게 만듭니다.

■ 스트라이드(Stride) 크기

- 스트라이드는 필터가 이미지 위를 지나가는 스텝 크기입니다.
- 1픽셀 크기와 2픽셀 크기의 스트라이드를 널리 사용하며 3픽셀 스트라이드도 사용합니다.
- 스트라이드가 큰 경우 이미지 영역을 건너뛸 수 있으므로 최적이 아닐 수 있습니다.
- 스트라이드를 늘리면 계산량이 줄어들어 속도가 빨라집니다.

■ 패딩(Padding)

- 입력데이터 주변을 특정값으로 채워 늘리는 것으로 출력데이터의 크기를 조절하기 위해 사용합니다.
- 주로 zero-padding을 사용합니다.

활성화 맵 크기 계산

$$\text{활성화 맵} = \frac{D - F + 2P}{S} + 1$$

D : 이미지 크기(이미지 너비)

F : 필터의 크기

P : 패딩의 양

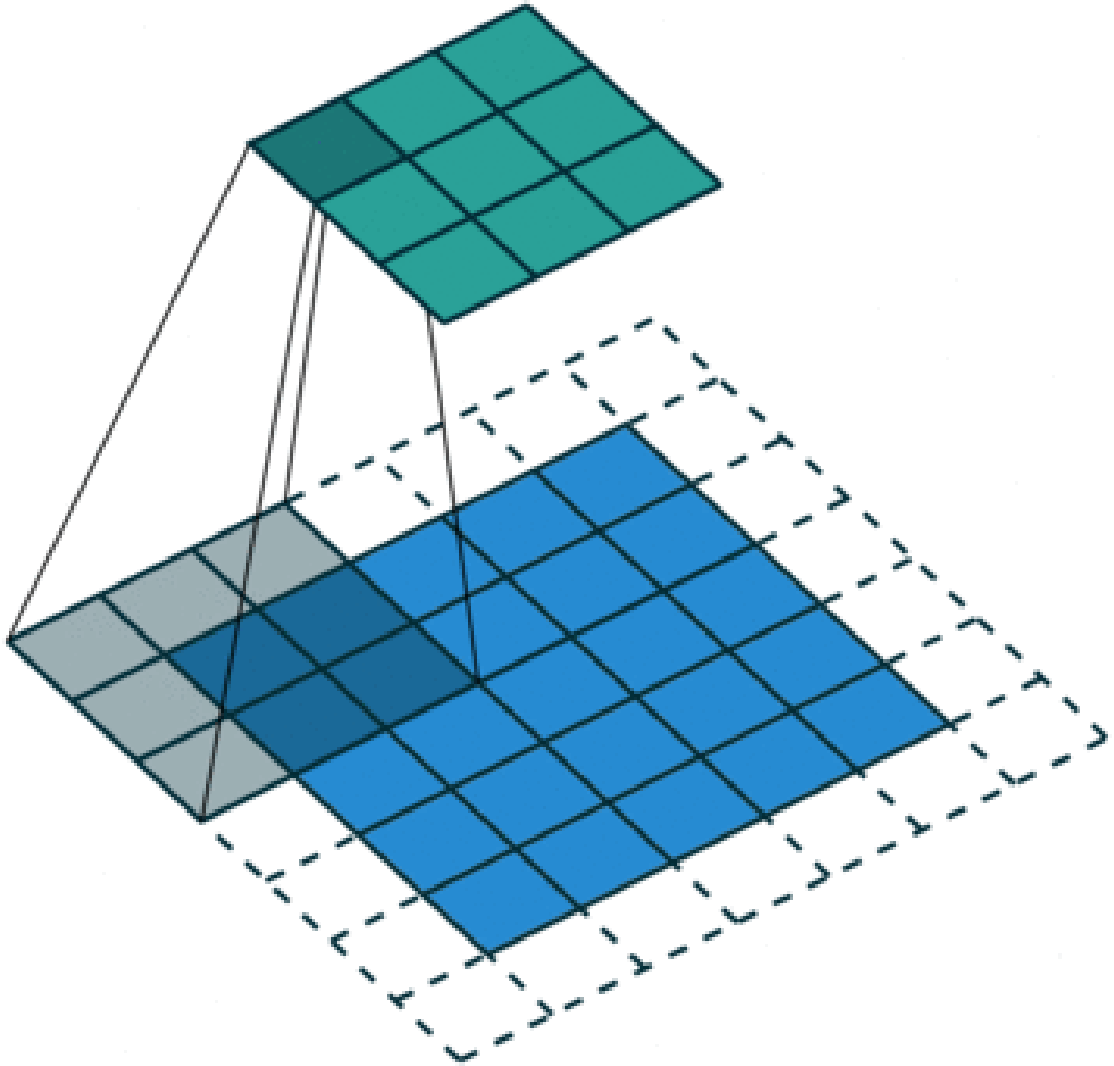
S : 스트라이드 크기

- 이미지 크기 : 28 x 28
- 필터 크기 : 5 x 5
- 패딩의 양 : 2
- 스트라이드 크기 : 1

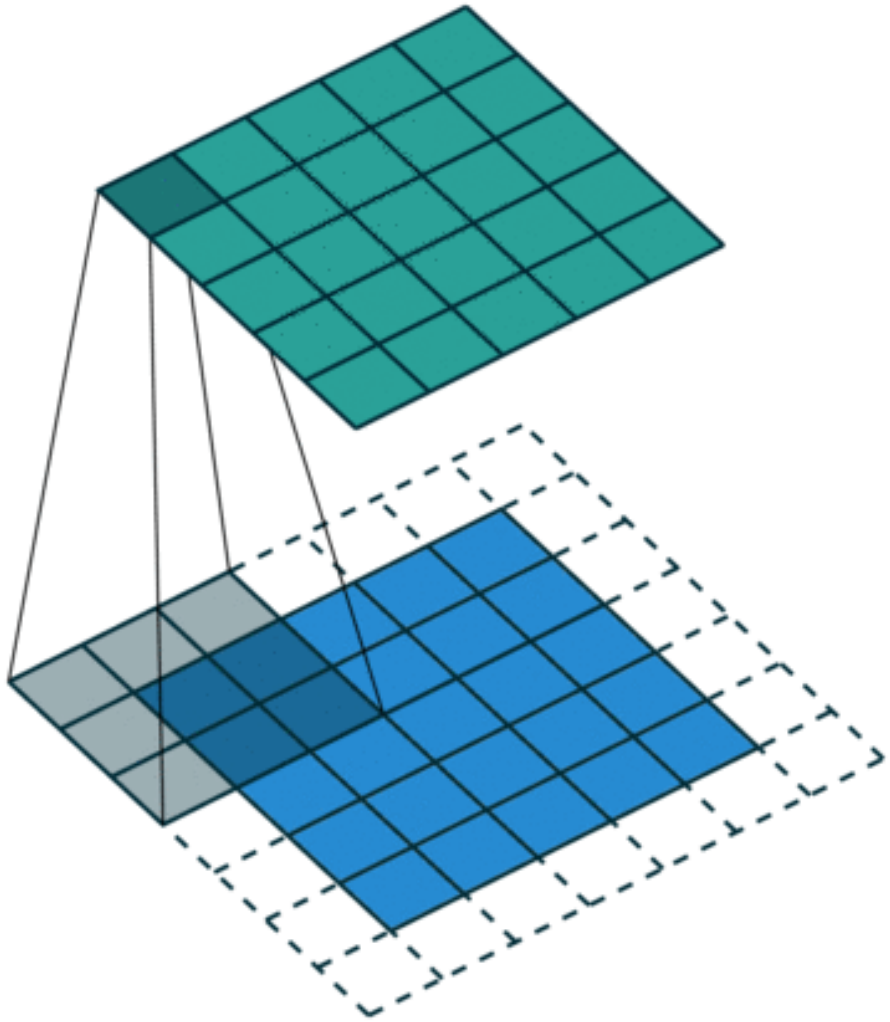
$$\frac{D - F + 2P}{S} + 1 = \frac{28 - 5 + 2 \times 2}{1} + 1 = 28$$

Convolution Layer

Convolution Operation with Stride Length = 2



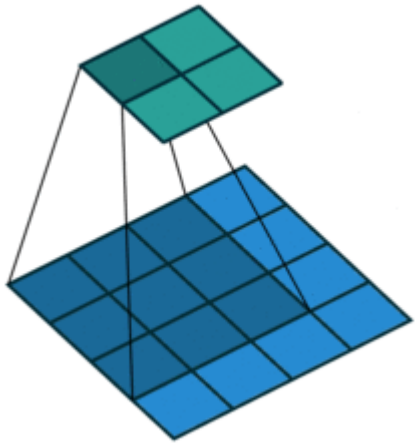
SAME padding



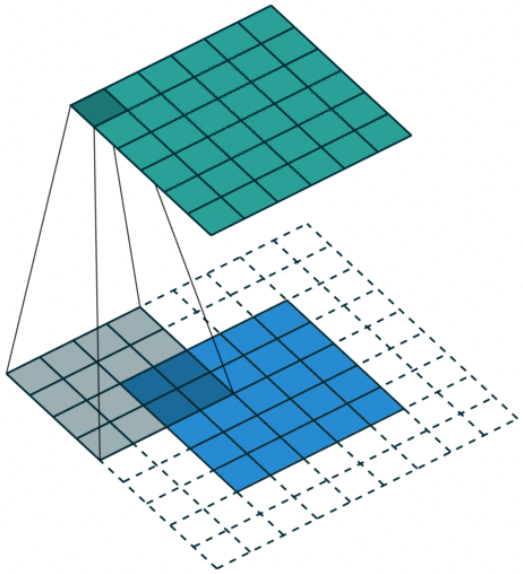
y/2RVDoFI

Convolution Layer - The Kernel

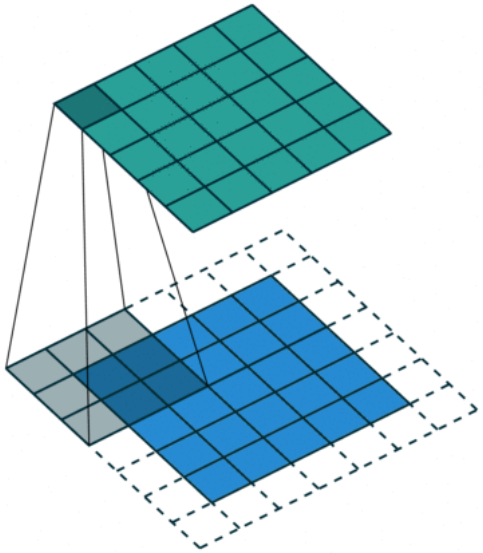
No padding, no strides



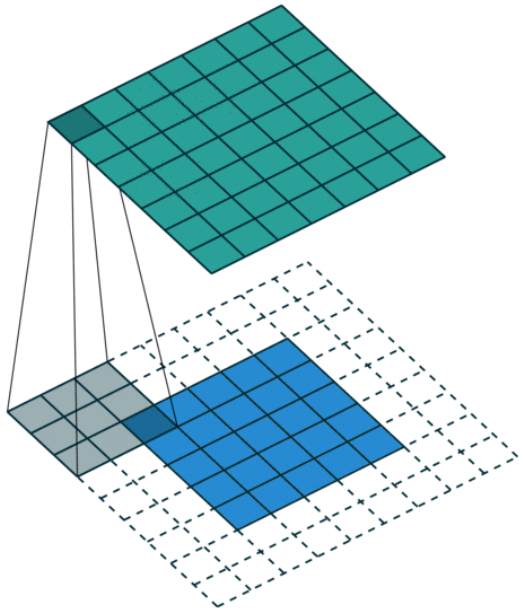
Arbitrary padding, no strides



Half padding, no strides



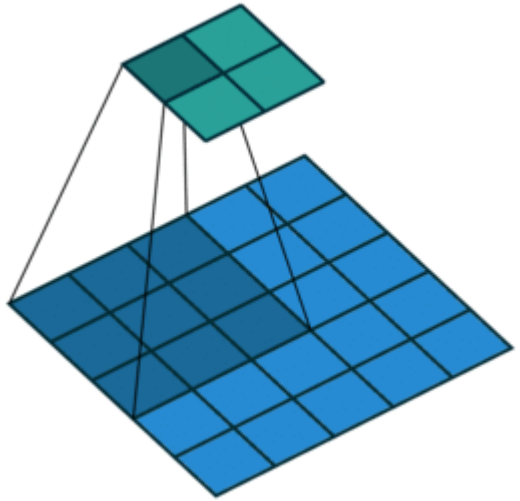
Full padding, no strides



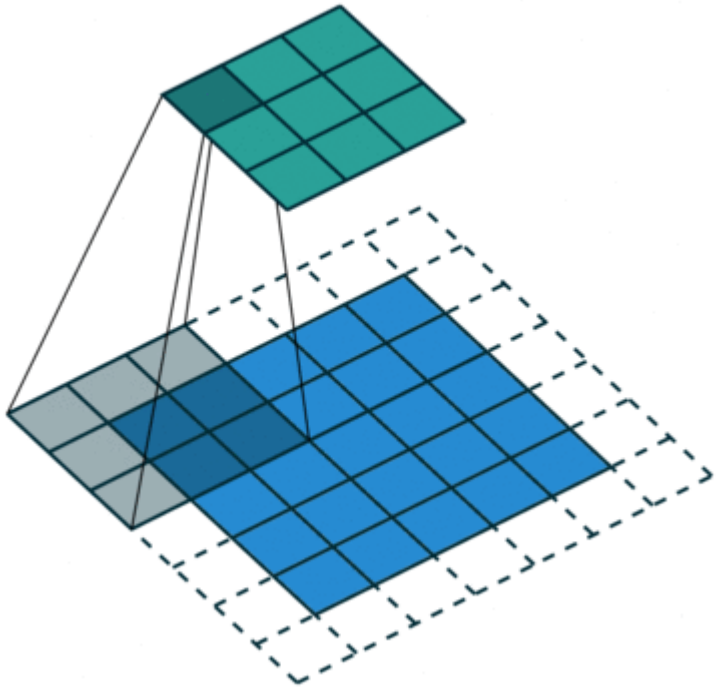
https://github.com/vdumoulin/conv_arithmetic

Convolution Layer - The Kernel

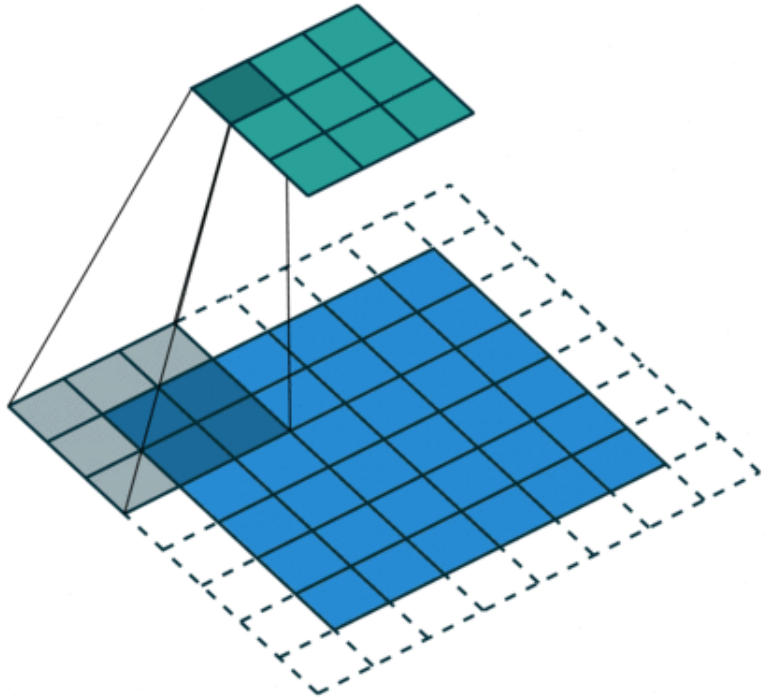
No padding, strides



Padding, strides



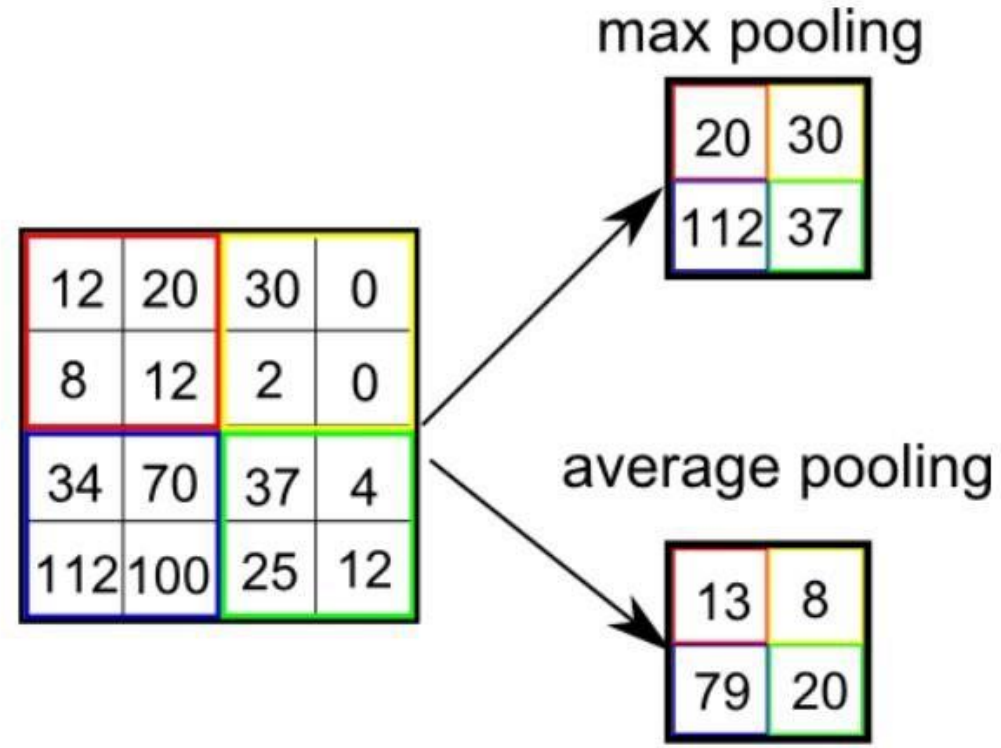
Padding, strides (odd)



https://github.com/vdumoulin/conv_arithmetic

Pooling Layer

- 폴링층(Pooling Layer)은 신경망의 전체 파라미터 개수와 복잡도를 줄이는 역할을 합니다.
- 계산속도를 높이고 과대적합을 피하는 데 도움이 됩니다.
- 폴링층은 활성화 맵의 공간 차원을 축소 시키고 깊이 차원은 그대로 유지합니다.
- 폴링층에서 자주 사용하는 연산은 max이고 이런 층을 max pooling 층이라고 합니다.
- 전형적으로 폴링층의 필터 크기는 2 x 2이고 스트라이드 크기는 2입니다.



https://bit.ly/2RVDoFI

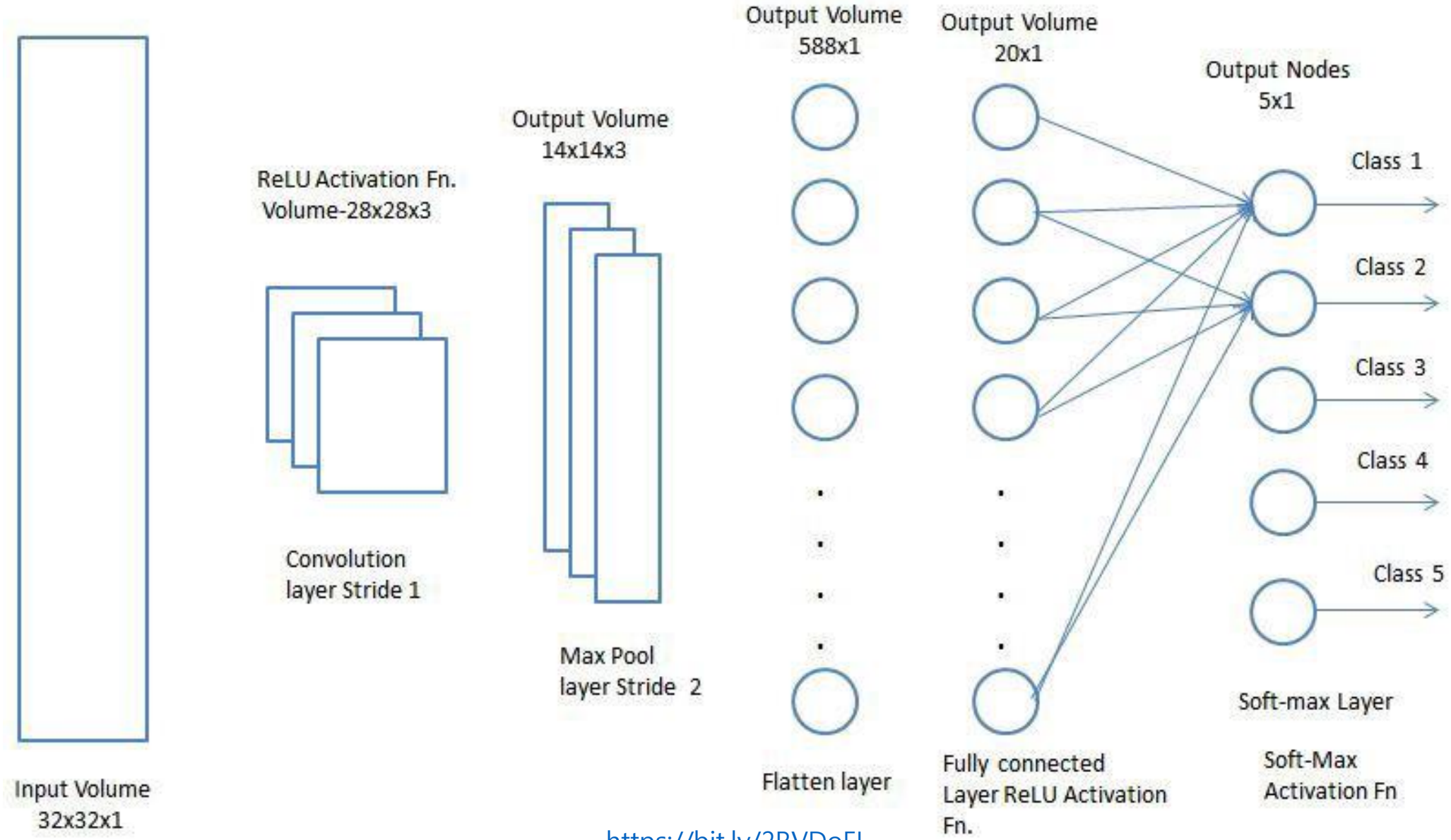
Pooling Layer

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

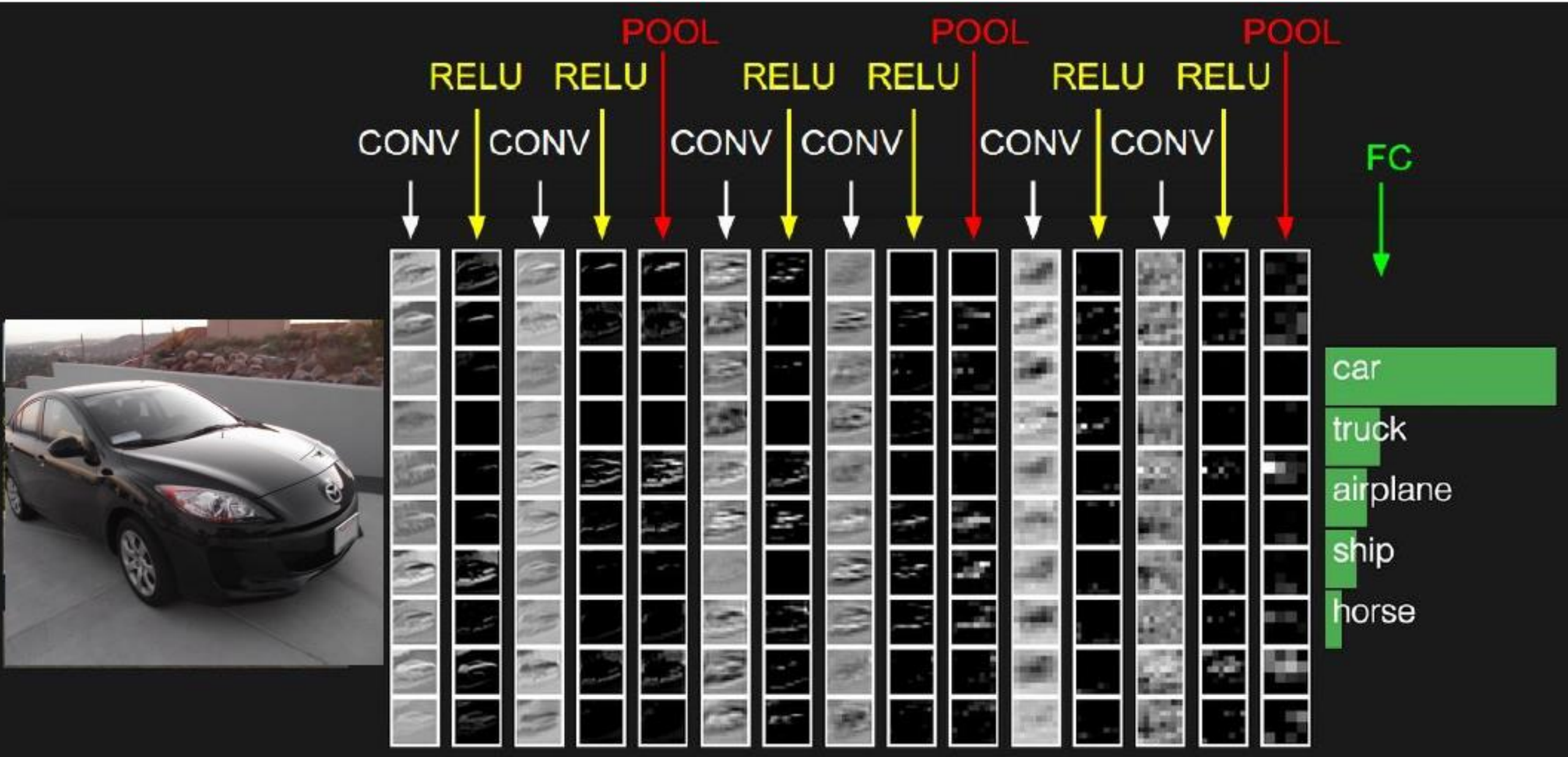
https://bit.ly/2RVDoFI

FC Layer(Fully Connected Layer)

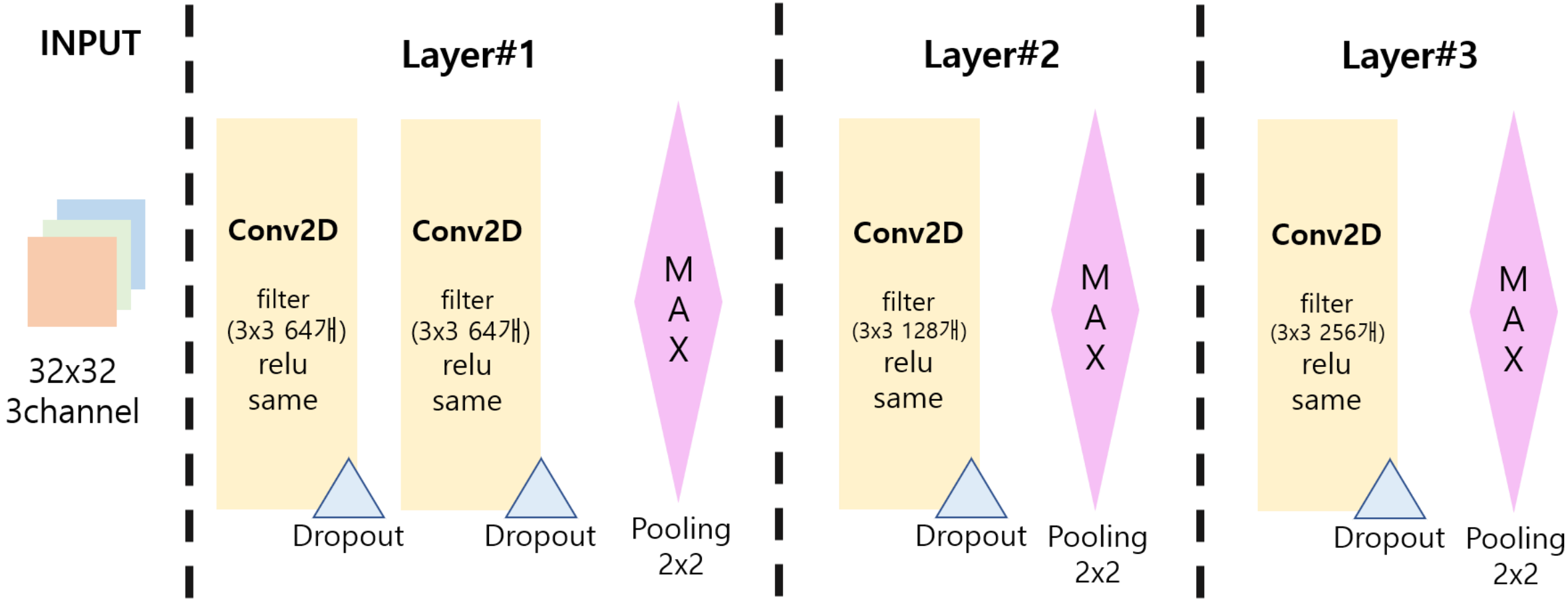


https://bit.ly/2RVDoFI

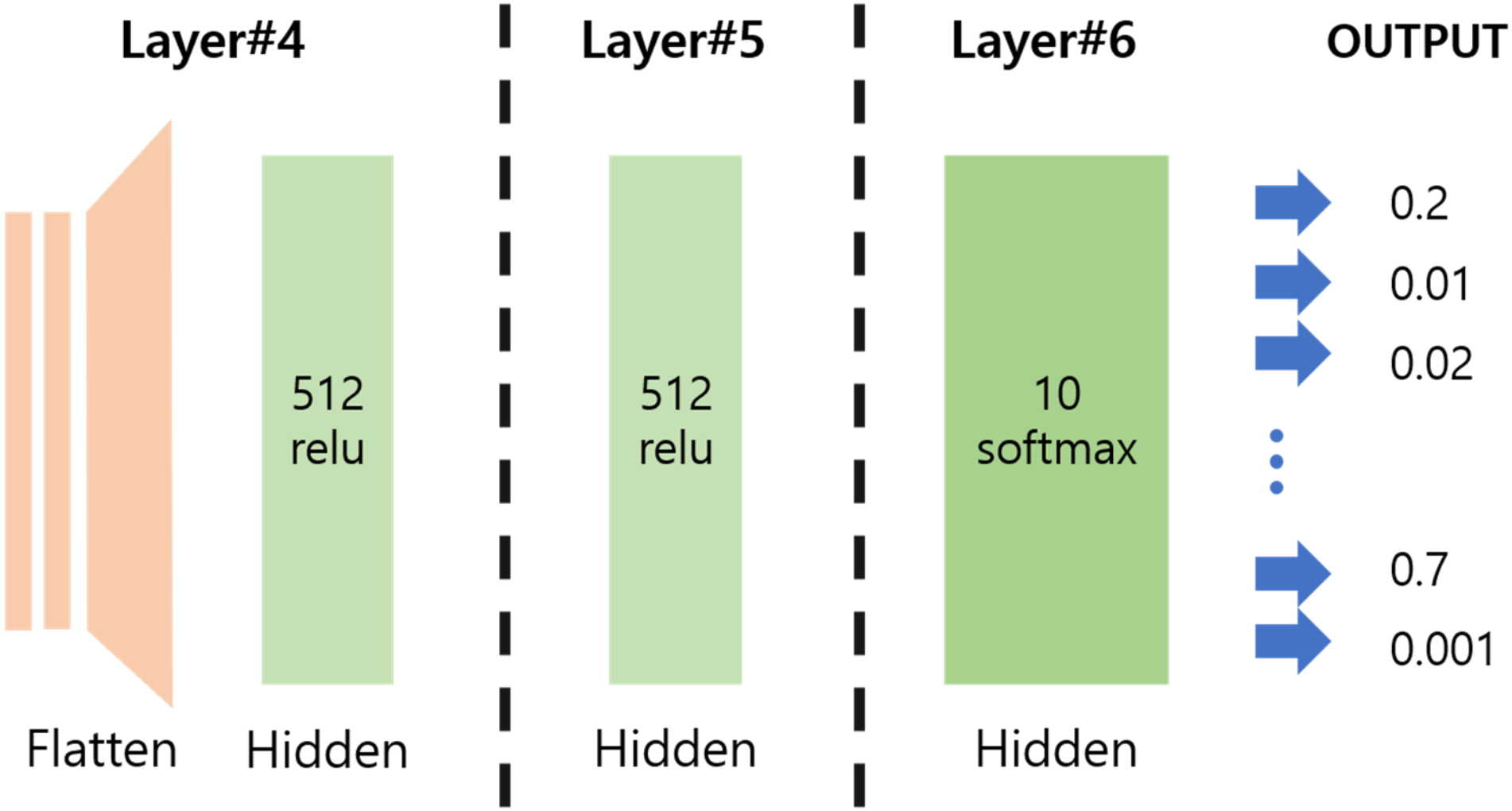
CNN 아키텍처


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

CNN 아키텍처 - Feature Extraction



CNN 아키텍처 - Classification



LeNet-5 CNN 모델 구현

라이브러리 임포트

```
[1]  import numpy as np
     import pandas as pd
     from matplotlib import pyplot as plt

     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.datasets import mnist
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout
     from tensorflow.keras.layers import Flatten, Conv2D, MaxPooling2D
```

LeNet-5 CNN 모델 구현

LeNet-5 CNN 모델

```
[7]  model = Sequential()

     model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
     model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.25))
     model.add(Flatten())

     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.5))

     model.add(Dense(n_classes, activation='softmax'))
```

LeNet-5 CNN 모델 구현

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

LeNet-5 CNN 모델 구현

```
Epoch 1/10
469/469 [==============================] - 39s 15ms/step - loss: 0.2381 - accuracy: 0.9282 - val_loss: 0.0540 - val_accuracy: 0.9830
Epoch 2/10
469/469 [==============================] - 7s 15ms/step - loss: 0.0857 - accuracy: 0.9747 - val_loss: 0.0391 - val_accuracy: 0.9875
Epoch 3/10
469/469 [==============================] - 7s 15ms/step - loss: 0.0629 - accuracy: 0.9811 - val_loss: 0.0427 - val_accuracy: 0.9860
Epoch 4/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0529 - accuracy: 0.9837 - val_loss: 0.0305 - val_accuracy: 0.9902
Epoch 5/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0447 - accuracy: 0.9860 - val_loss: 0.0291 - val_accuracy: 0.9908
Epoch 6/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0387 - accuracy: 0.9874 - val_loss: 0.0283 - val_accuracy: 0.9912
Epoch 7/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0362 - accuracy: 0.9887 - val_loss: 0.0281 - val_accuracy: 0.9914
Epoch 8/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0300 - accuracy: 0.9903 - val_loss: 0.0339 - val_accuracy: 0.9897
Epoch 9/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0280 - accuracy: 0.9908 - val_loss: 0.0261 - val_accuracy: 0.9916
Epoch 10/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0262 - accuracy: 0.9919 - val_loss: 0.0266 - val_accuracy: 0.9913
```

LeNet-5 CNN 모델 구현

cnn_lenet.ipynb

Thank you

kgpark88@gmail.com