

w6-1 0715

convolution neural network

Vs Deep neural network

Deep Neural Network

데이터들을 배열로 만든다 →
각 데이터의 크기에 따른 **hidden layer** 들을 생성
→ 분류

Convolution Neural Network

데이터들을 배열로 만든다 →
각 데이터를 적당히 **분할**한 후 **필터**를 입혀서 **부표본(subsample)** 생성 →
샘플들을 또 적당히 **분할**한 후 **필터**를 입혀서 **부표본(subsample)** 생성 →
(deep) neural network
→ 분류

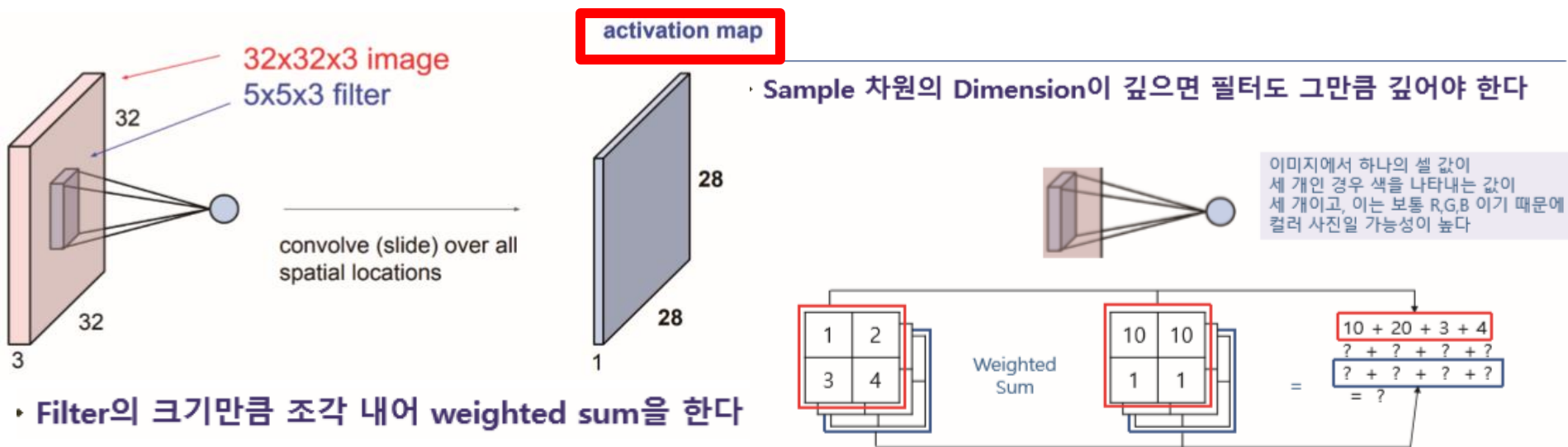
convolution neural network

1. convolution layer

완전연결 신경망, 국소연결 신경망, 가중치공유신경망, 국소연결 가중치 공유 신경망

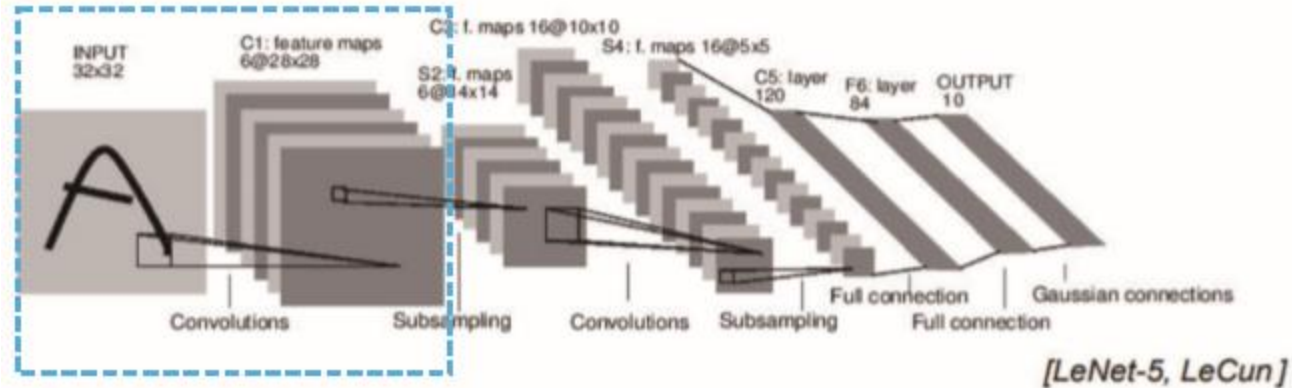
neural network는 이미지 데이터가 데이터 사이즈가 커 너무 많은 파라미터를 가지게 되어서 쉽게 오버피팅 되는 문제를 해결하기 위해 고안되었다.

Convolution Neural Network(CNN, ConvNet)는 **multilayer neural network**를 통해 최소한의 전처리 프로세싱을 이용해 오버피팅 문제를 해결함.

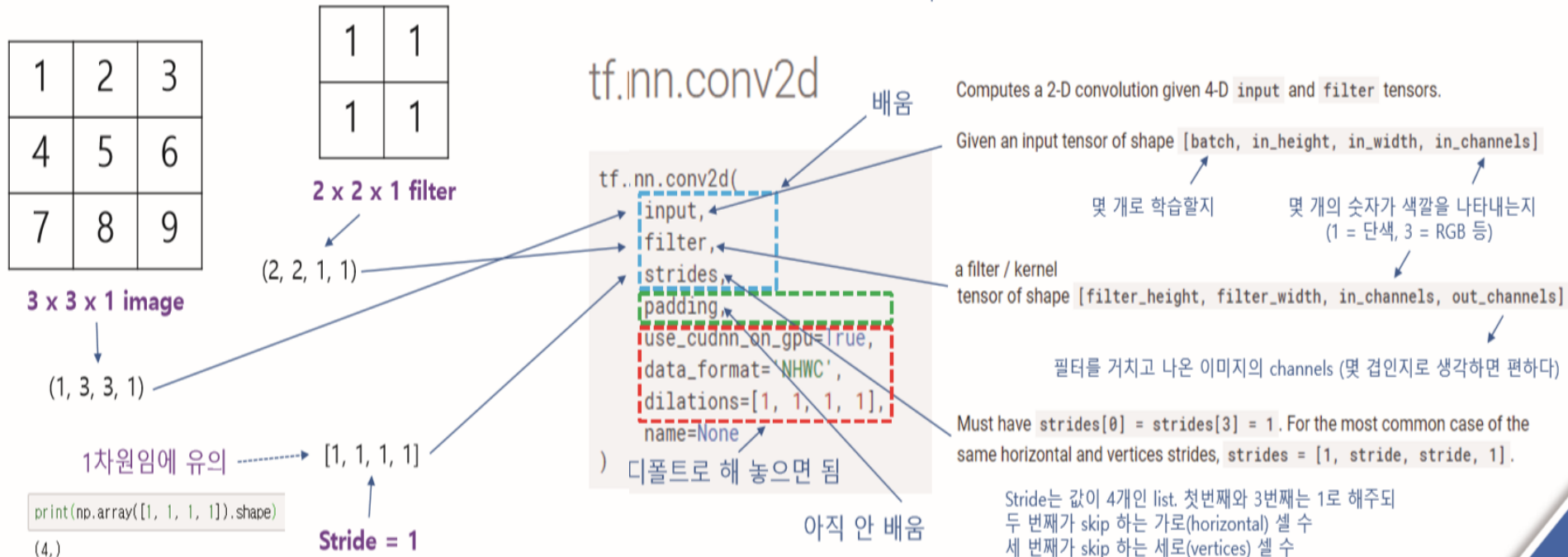


convolution neural network

tf.nn.conv2d 구성



tensorflow 에서 activation map을 만들어주는 함수 conv2d 가 다음 형태의 파라미터를 가진다



convolution neural network

1) input / output(activation map)

activation map의 값이 계산한 값이랑 다르게 나옴 → swapaxes 필요
(pooling으로 바로 들어갈 수 있는 이유)

Given an input tensor of shape [batch, in_height, in_width, in_channels]

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

3 x 3 x 1 image
with padding = 'SAME'

(1, 3, 3, 1)
-> (1, 4, 4, 1)

conv2d_img.shape (1, 3, 3, 3)

1	1
1	1

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

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

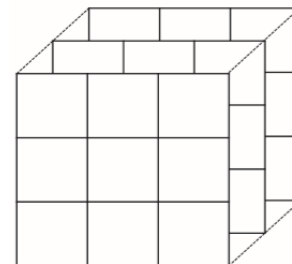
12	16	9
24	28	12
15	17	9

Activation map
(1, 3, 3, 1)

[[[12. 24. 36.]
[16. 32. 48.]
[9. 18. 27.]

[[24. 48. 72.]
[28. 56. 84.]
[15. 30. 45.]

[[15. 30. 45.]
[17. 34. 51.]
[9. 18. 27.]



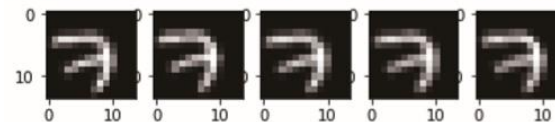
결과가 맞지 않다

(1, 3, 3, 3) 가 (3, 3, 3, 1) 이 되어야 함 (나중에 3, 3, 3 으로 reshape)
=> 0축과 3축을 바꾼다

```
conv2d_img = conv2d.eval()
conv2d_img = np.swapaxes(conv2d_img, 0, 3).reshape(3, 3, 3)
print("conv2d_img.shape", conv2d_img.shape)
print(conv2d_img)
```

```
conv2d_img = conv2d.eval()
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(14,14), cmap='gray')
    # collection의 인덱스 번호와 값 반환
sess.close()
```

Tensor("Conv2D_16:0", shape=(1, 14, 14, 5), dtype=float32)



convolution neural network

2) filter

: 정해진 필터를 이용해 이미지를 바꿔주거나 학습을 통해 값을 적절히 바꾸거나

: neural network의 weight와 대응됨

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: Blur



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



(We'll assume the kernel is normalized before convolution so the entries sum to 1)

(GIMP documentation)

Filter: Sharpen



$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$



(GIMP documentation)

Filter: Edge-Detect



$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$



(GIMP documentation)

Filter: Emboss



$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} =$$



(GIMP documentation)

convolution neural network

2) filter

Graph 만들기

* node_1 = **tf.constant**(3.0, shape=[2,3]) #암시적으로 float

→ Tensor("Const_1:0", shape=(2, 3), dtype=float32)

#Tensor는 print()함수 써도 이 모양확인이 안됨 π

[[-3. -3. -3.]

[-3. -3. -3.]]

* node_2 = tf.constant([1,2,3,4,5])

→ Tensor("Const_3:0", shape=(5,), dtype=int32)

[1 2 3 4 5]

* node_3 = tf.add(node, node) #add도 사용 가능

→ Tensor("Add:0", shape=(), dtype=dtype=float32)

```
tf.constant(  
    value,  
    dtype=None,  
    shape=None,  
    name='Const',  
    verify_shape=False  
)
```

convolution neural network

2) filter

Graph 실행하기 in session

1) **tf.Session** 이용

```
sess = tf.Session()
```

```
sess.run(node1)
```

```
sess.close()
```

#세션 만들고 사용할 때 변수명 .run()

close 필요

2) **with** 이용

```
with tf.Session() as sess :
```

```
    sess.run(node1)
```

변수명.run() , close 필요 X

3) **tf.InteractiveSession()** 이용

```
sess= tf.InteractiveSession()
```

```
node1.eval()
```

```
sess.close()
```

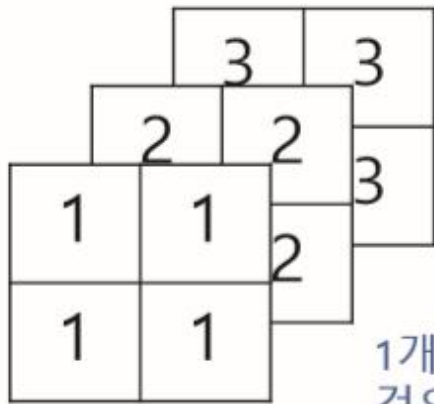
#close 안에서 사용할 때 변수명X .eval()

close 필요

convolution neural network

2) filter

filter 설정하기.



3 x 2 x 2



```
filter = np.array([[[[1,1],[1,1]],[[2,2],[2,2]],[[3,3],[3,3]]]])  
print(filter.shape)
```

```
filter = np.array([[[[1,1],[1,1]],[[2,2],[2,2]],[[3,3],[3,3]]]])  
print(filter.shape)
```

(1, 3, 2, 2)

1개를 의미하는
것으로 임시 설정

Filter의 개수
= output channel

height
width

a filter / kernel
tensor of shape [filter_height, filter_width, in_channels, out_channels]

(1, 3, 2, 2) 가 (2, 2, 1, 3) 이 되어야 함
=> 0축과 2축을 바꾼다, 1축과 3축을 바꾼다

```
filter = np.swapaxes(filter, 0, 2)  
filter = np.swapaxes(filter, 1, 3)
```

convolution neural network

3) stride

: filter가 한 step당 sample에서 건너뛰는 크기

❖ Stride

- step당 건너뛰는 크기

1

1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6
7	7	7	7	7	7	7

Stride = 1

0	0	0
1	1	1
2	2	2

Filter

$$\begin{aligned} &0 + 0 + 0 + \\ &2 + 2 + 2 + \\ &6 + 6 + 6 \end{aligned}$$

24	24			

Activation map

convolution neural network

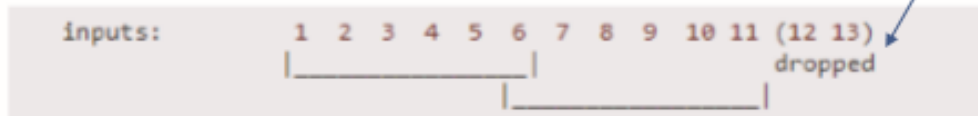
4) padding

- : 이미지의 외각에 지정된 값만큼 특정 값으로 채워 넣는 작업
- weighted sum을 데이터의 drop 없이 모두 수행할 수 있다.
- sample의 크기가 너무 빨리 줄어드는 것을 방지할 수 있다.
- 경계면의 정보를 살릴 수 있다.
(0으로 둘러싸지 않는다면 어느 곳이 경계면인지 알기 힘들 수 있다.)
- : **종류** : padding은 딱 = "SAME"과 = "VALID" 두 종류만 지원
- **VALID** : padding을 지정하지 않음. 데이터를 잃을 수 있음
- **SAME** : 잃어버리는 데이터가 없을때까지 적당히 padding해줌.
 - 오른쪽, 아래 → 왼쪽 위쪽 순서

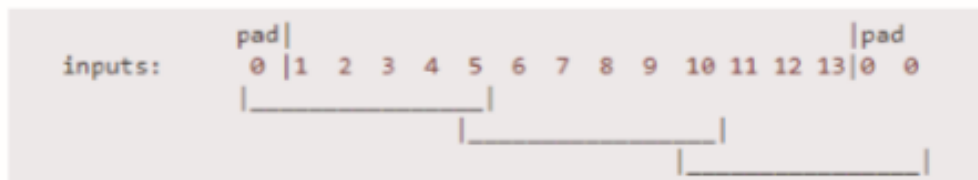
If you like ascii art:

Stride 값에 따라 벗어날 경우 버려지는 점 주목

- "VALID" = without padding:



- "SAME" = with zero padding:



convolution neural network

5) pooling

: 데이터의 사이즈를 줄이거나 강조할 때 사용,
특징맵의 차원을 적극적으로 줄이고 특징들을 뚜렷하게 하려고 사용.
(stride, filter, padding 개념 사용)

: 종류 – Max pooling, Average pooling

Max pooling : stride의 크기만큼 이동할 때 해당 filter에서 가장 큰 값을 뽑아냄.

Average pooling : 평균값을 뽑아냄.
(`tf.nn.avg_pool`)

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

원본 4 × 4 × 1

2 × 2 × 1 filter

Stride = 2

6	8
14	16

tf.nn.max_pool

```
tf.nn.max_pool(  
  value,  
  ksize,  
  strides,  
  padding,  
  data_format='NHWC',  
  name=None  
)
```

Max pooling 할 sample

Filter

ksize : A list or tuple of 4 ints.

The size of the window for each dimension of the input tensor.

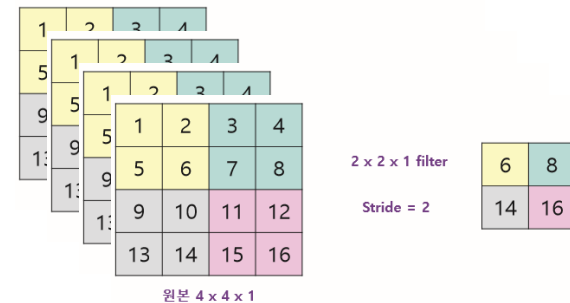
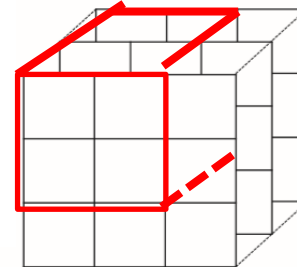
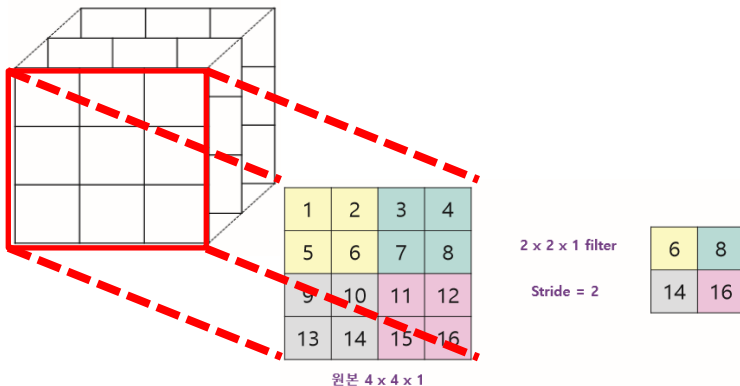
Stride : 동일

Padding : 동일

convolution neural network

5) pooling

pooling할 때 swapaxes해줘야 하는 이유 :
channel이 많이 있으면 한 channel마다 pooling을 찍어주는 것이 아니라,
모든 채널의 같은 자리를 뺄어와서 필터도 그만큼을 만들어서 계산해서 생
성된 각 자리를 나중에 합쳐 줘야함. (batch를 swapaxes해줘야하는 이유)



tf.nn.max_pool

```
tf.nn.max_pool(  
    value, ← Max pooling 할 sample  
    ksize, ← Filter  
    strides, ← Stride : 동일  
    padding, ← Padding : 동일  
    data_format='NHWC',  
    name=None  
)
```

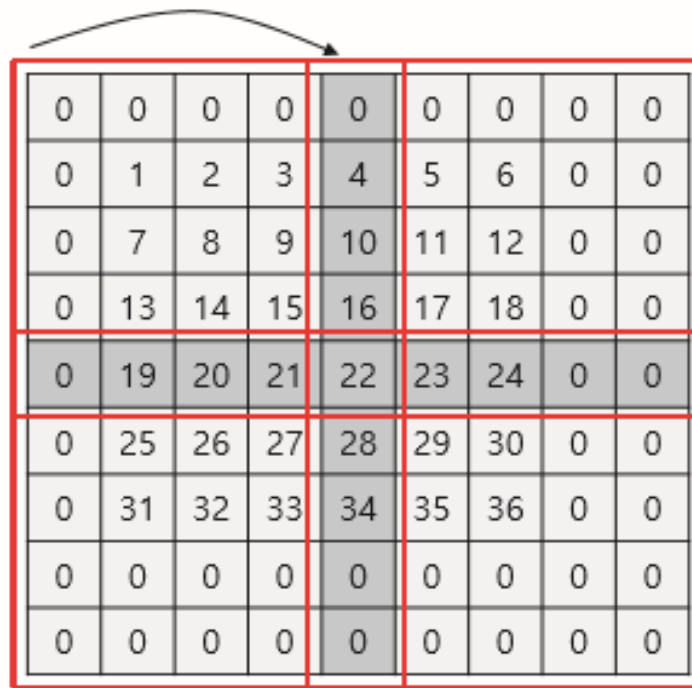
Filter
ksize : A list or tuple of 4 ints.
The size of the window for each dimension of the input tensor.

convolution neural network

padding/pooling 심화

❖ Pooling - padding 고급

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

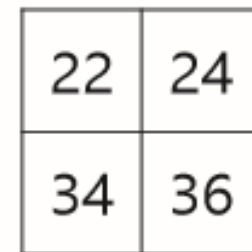


0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	0	0	0
0	7	8	9	10	11	12	0	0	0
0	13	14	15	16	17	18	0	0	0
0	19	20	21	22	23	24	0	0	0
0	25	26	27	28	29	30	0	0	0
0	31	32	33	34	35	36	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

원본 6 x 6 x 1

5 x 5 x 1 filter

Stride = 4



22	24
34	36

(2, 2, 1)

convolution neural network

[심화과정] Activation Map(Feature Map)의 크기는 어떻게 계산할까?

$$\text{Output Size} = \frac{(\text{Input Size} + 2 \times \text{Padding} - \text{Filter Size})}{\text{Stride}} + 1$$

Padding이 있을 경우 반영된 Input 크기

(Stride가 1일때) Input 크기에서 Filter 크기를 빼면, 남은 filter의 convolving 횟수가 나옴

(Stride가 1이 아닐때) filter의 가능한 convolving 횟수를 stride값으로 나눔

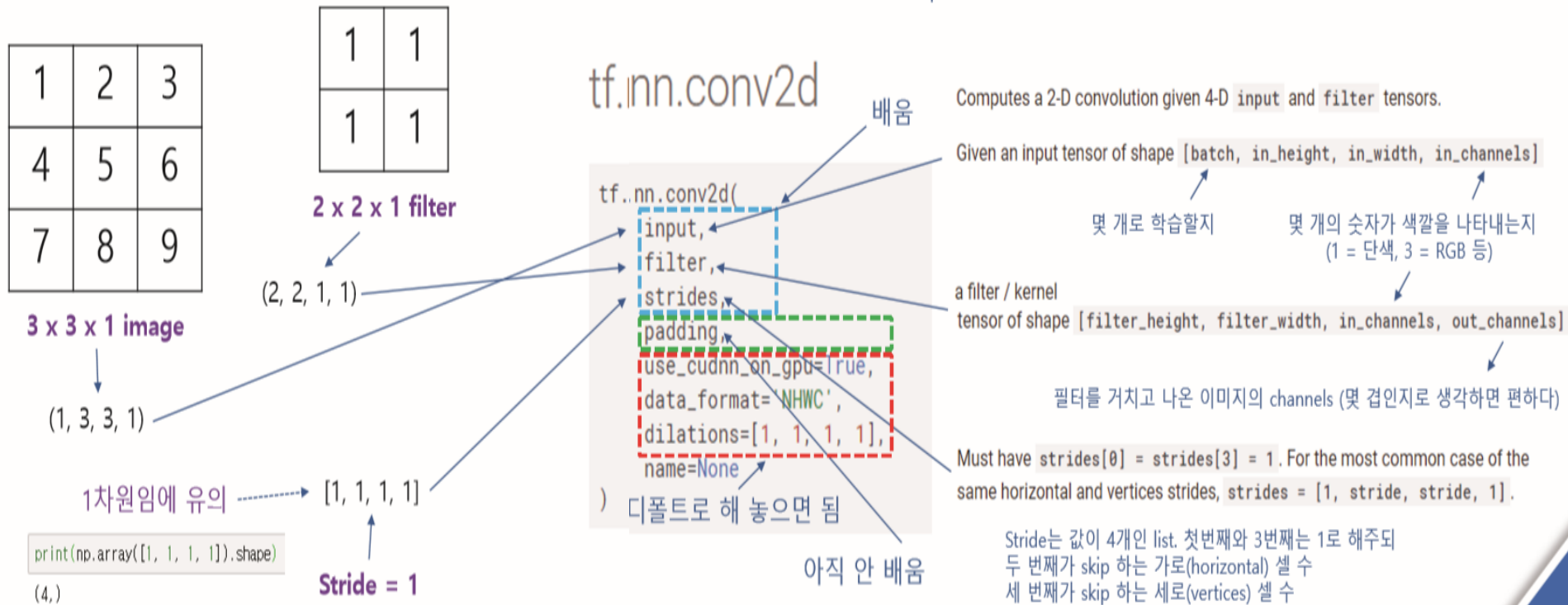
Convolving하기 전, 첫번째 filter가 input과 inner product를 수행한 횟수

convolution neural network

tf.nn.conv2d

왜 input 의 형태가 (1, 3, 3, 1) 인가?

tensorflow 에서 activation map을 만들어주는 함수 conv2d 가 다음 형태의 파라미터를 가진다



CNN : 활성화 함수

: 입력 신호의 합에 적절한 변화를 주어 출력 신호로 반환하기 위한 함수. 입력 신호의 총합이 활성화를 일으키는지 아닌지를 정해주는 역할을 함.

신경망과 퍼셉트론의 차이는 활성화 함수의 존재 여부

- 퍼셉트론은 임계값을 경계로 출력이 바뀜

: 계단 함수 사용 (입력값이 0 보다 클 때 1 작을 때 0을 반환)

활성화 함수의 종류

1) 시그모이드 함수 :

- 계단 함수와 달리 값이 0 또는 1로 나뉘지 않고 출력값이 0에서 1 사이의 값을 가진다.

- 비선형 함수 : 비선형

- 신경망이 깊어지다 $f(t) = \frac{1}{1+e^{-t}}$

깊어지는 학습에 의미를 준다.

생기다가 0으로 수렴하는 Gradient

Vanishing 문제가 생긴다.

2) 이 문제를 해결하기 위해 ReLU 함수가 널리 이용됨

: 하지만 중심값이 0이 아니어서 최적화 과정이 느려지는 문제가 생길 수 있음.

3) tanh 함수(중심값이 0), Leaky ReLU, PReLU, Exponential Linear Unit, Maxout 함수 등 여러 함수가 등장.

TENSORFLOW 기본 함수들

tf.기본 함수

tf.placeholder :

선언 후 데이터가 제공된 다음 값을 전달. 할당이 아니라 텐서를 placeholder에 맵핑 시키는 것.

dtype 데이터 타입, shape 입력 데이터의 형태,

```
placeholder(  
    dtype,  
    shape=None,  
    name=None  
)
```

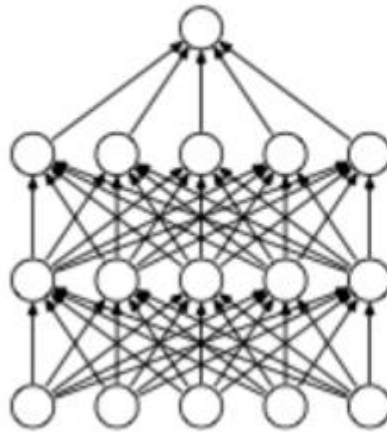
tf.layers.함수

tf.layers.dropout() :

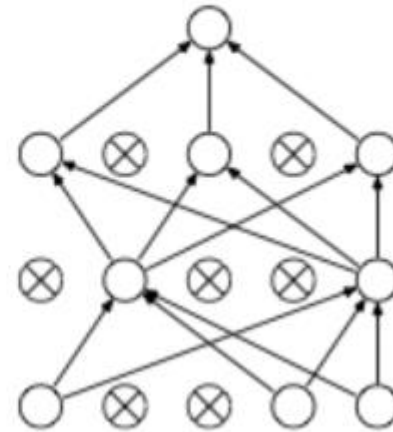
inputs 텐서, rate=(0-1사이 버릴 비율), training : [True(dropout을 쓰겠다), False(dropout을 안쓰겠다 - inference mode)], noise_shape = 버릴 때 마스크, name : The name of the layer (string)

```
tf.layers.dropout(  
    inputs,  
    rate=0.5,  
    noise_shape=None,  
    seed=None,  
    training=False,  
    name=None  
)
```

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]



(a) Standard Neural Net



(b) After applying dropout.