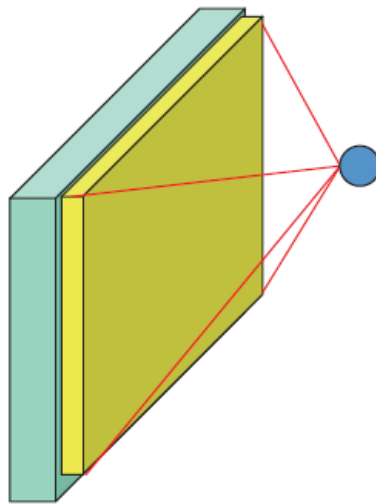


9장 컨버전션 신경망

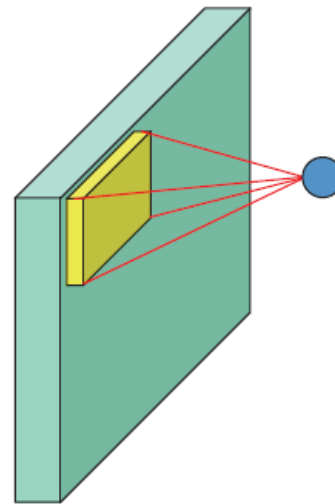


컨볼루션 신경망(CNN)

- 기존의 신경망 : 측정 레이어의 유닛은 앞단의 모든 유닛과 완전히 연결(fully connected)
- 컨볼루션(Convolution Neural Network: CNN) 신경망 : 하위 레이어의 노드들과 상위 레이어의 노드들이 부분적으로만 연결



(a) 완전연결 신경망



(b) 컨볼루션 신경망

컨볼루션 신경망(CNN)

- 컨볼루션 신경망은 Hubel과 Wiesel이 발견한 고양이의 시각 세포에서 영감을 얻었다.
- 제한된 시야 영역에서만 자극에 반응

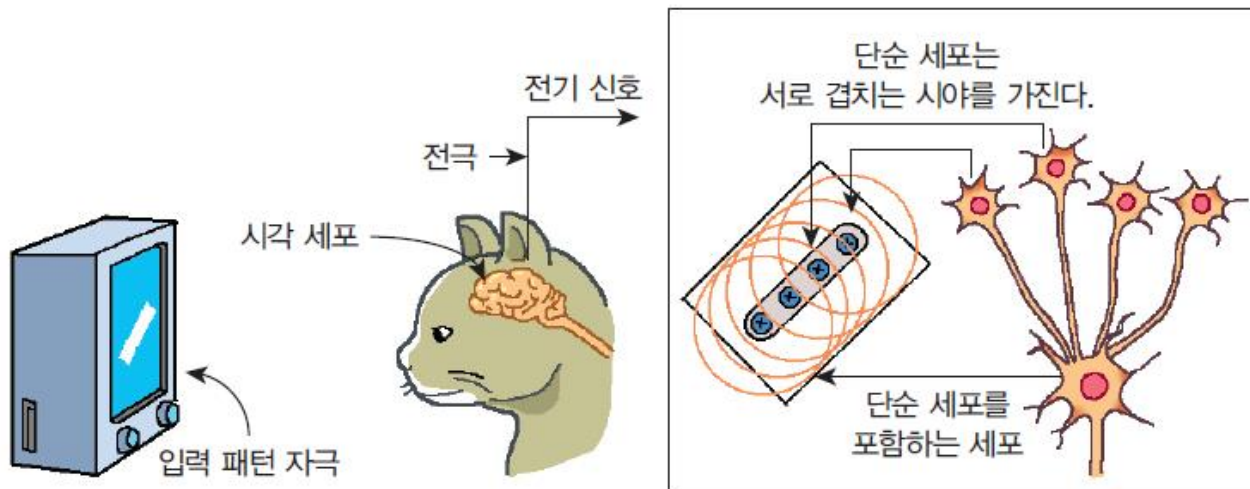


그림 9-2 시각 피질 뉴런의 구조(출처: Hubel's book, Eye, Brain, and Vision)



네오코그니트론(Neocognitron)

- 1980년 후쿠시마(Kunihiko Fukushima)에 의해 소개된 신경망 구조
- 허벨과 위젤의 작업에서 영감을 받았다

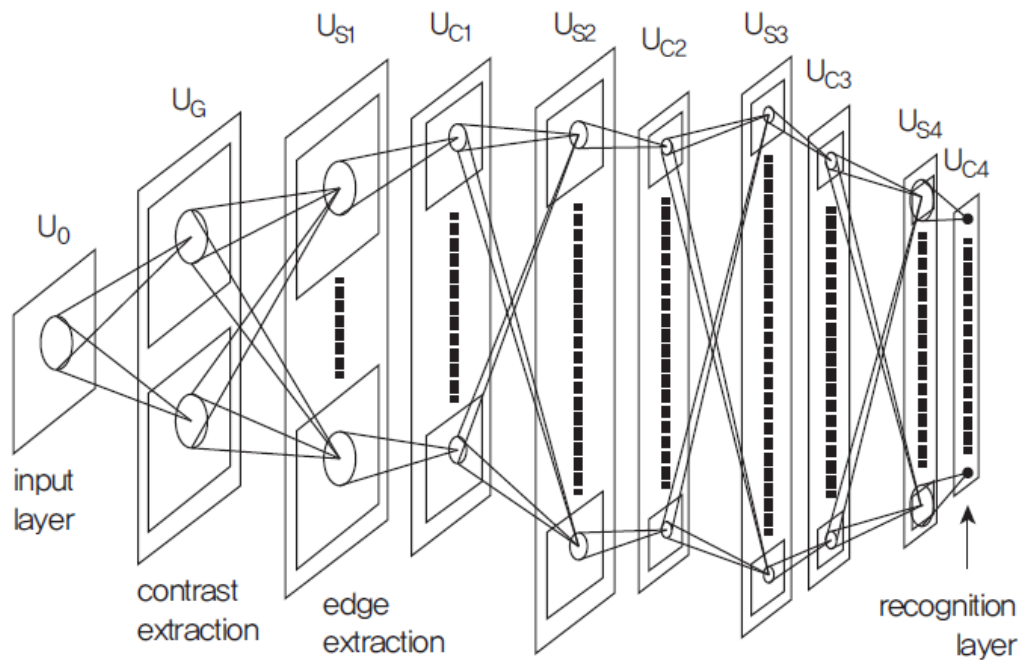
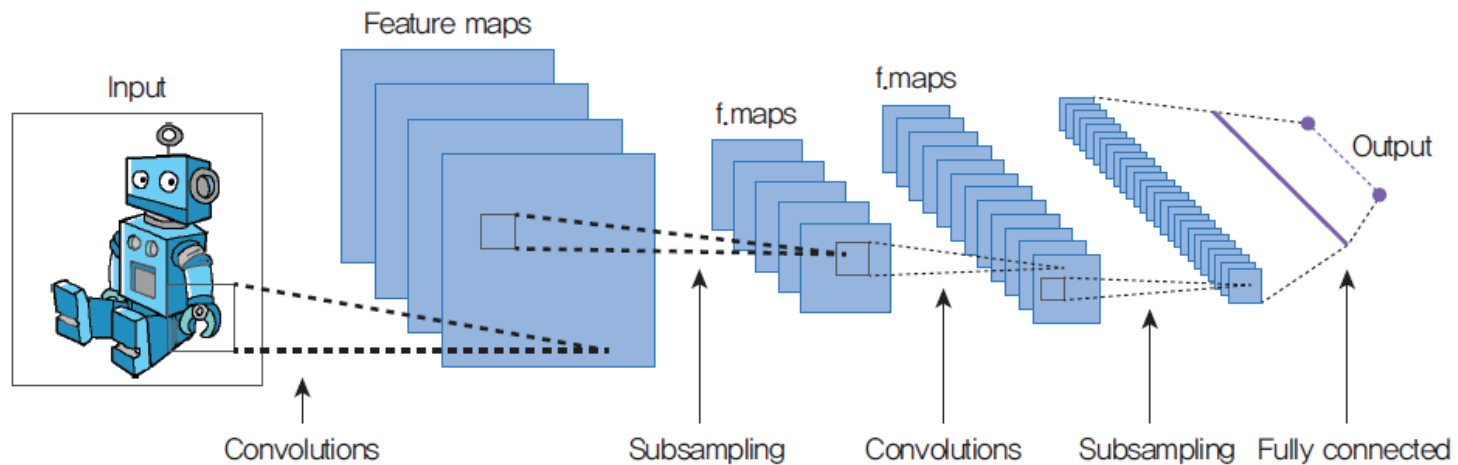


그림 9-3 네오코그니트론(출처: 후쿠시마의 논문)



컨버루션 신경망의 중요성

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



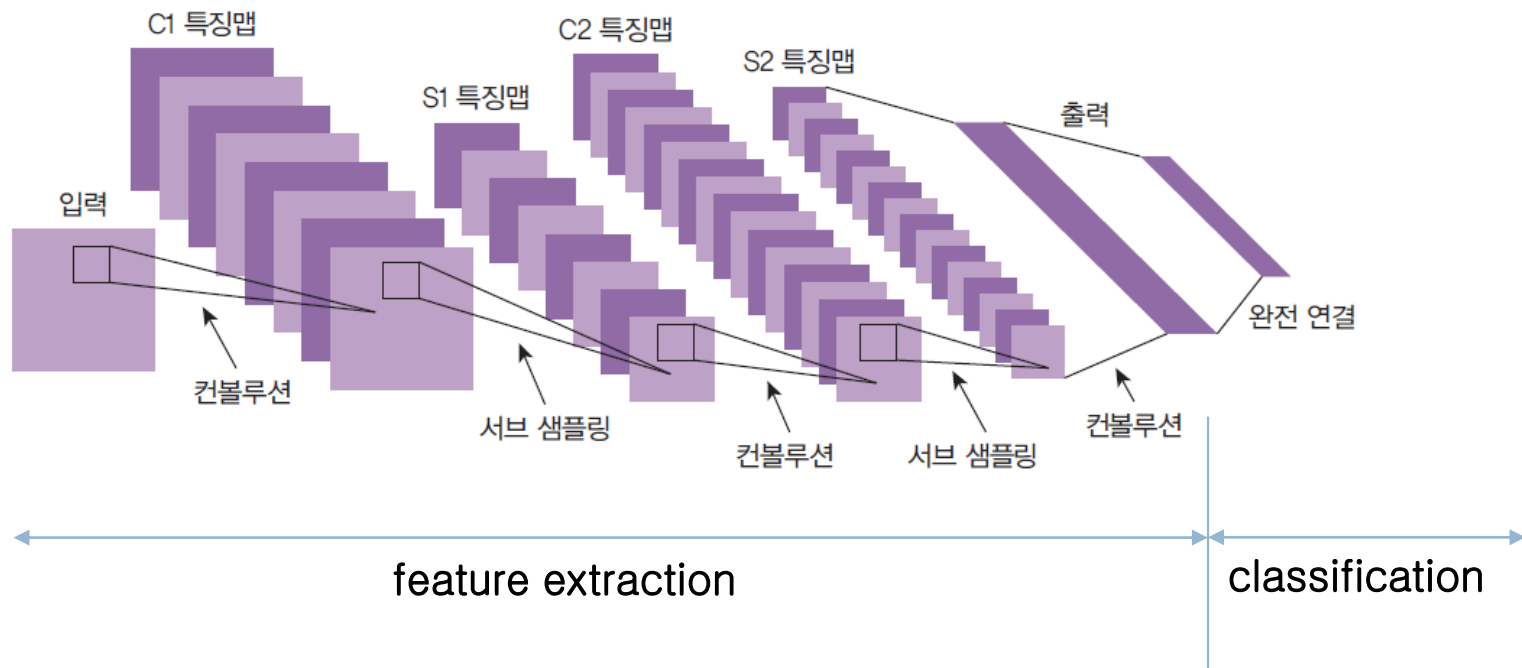


컨벌루션 신경망의 중요성

- 2012년도 영상 인식 경진 대회 ILSVRC에서 우승하여, 딥러닝의 우수성을 전 세계에 알리고 인공지능 부활의 신호탄이 되었던 신경망 모델도 컨벌루션 신경망이었다



- 컨볼루션 신경망도 여러 레이어를 연결하여 신경망을 구축한다





영상 처리에서의 컨볼루션 영상

- 이해에 도움이 되는 링크
- [File:2D Convolution Animation.gif](#)
- [Animations of Convolution and Deconvolution](#)
- [All Convolution Animations Are Wrong \(Neural Networks\)](#)



영상 처리에서의 컨볼루션 영상

- 컨볼루션 연산: 필터링 연산. 이미지로부터 어떤 특징값을 얻을 때.
- 커널(kernel) = 필터(filter) = 마스크(mask) : 가중치를 갖는 2차원 배열(3x3, 5x5, 7x7, ...)
- 커널은 입력영상에서 각 화소를 중심으로 덮어 씌어진다. 커널 아래에 있는 화소들은 각각 해당되는 마스크의 값들과 곱해져서 더해진다. 계산값은 출력영상의 동일한 위치에 저장된다. 현 화소 처리가 끝나면 마스크는 한 칸 이동한다.

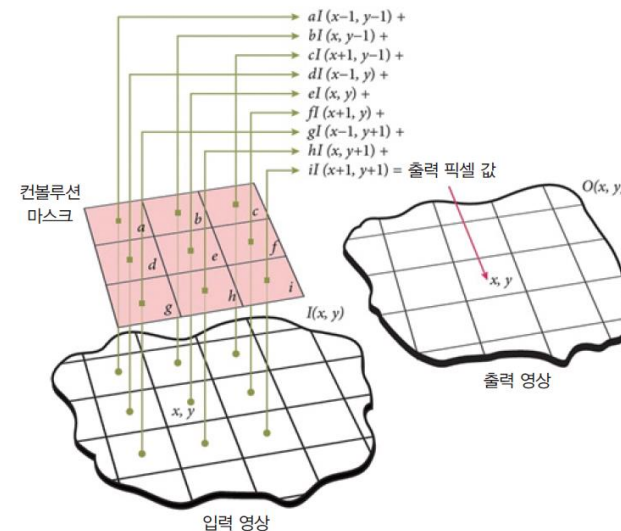


그림 9-6 영상 처리에서 컨볼루션 연산



영상 처리에서의 컨볼루션 영상

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

가중치 커널

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 레이어

	4.88						

출력 레이어

$$\text{ReLU}(1/9 * 3 + 1/9 * 6 + 1/9 * 6 + 1/9 * 3 + 1/9 * 4 + 1/9 * 3 + 1/9 * 5 + 1/9 * 7 + 1/9 * 7) = 4.88$$



영상 처리에서의 컨볼루션 영상

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

가중치 마스크

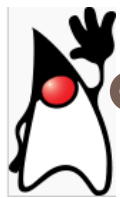
3	6	6	4	7	8	2	1
3	4	3	8	8	3	3	2
5	7	7	7	7	4	3	2
8	9	9	9	9	9	3	2
8	3	3	4	3	2	1	1
8	9	9	8	8	3	3	2
6	4	3	8	8	3	3	2
7	4	3	8	8	3	3	2

입력 레이어

	4.88	5.77					

다음 레이어

$$\text{ReLU}(1/9 * 6 + 1/9 * 6 + 1/9 * 4 + 1/9 * 4 + 1/9 * 3 + 1/9 * 8 + 1/9 * 7 + 1/9 * 7 + 1/9 * 7) = 5.77$$



영상 처리에서의 컨벌루션 연산

- 라플라시안 필터(가운데만 8이고 나머지는 -1)를 영상에 적용하면 영상에서 에지(edge)를 찾을 수 있다.
- 이와 같이 컨벌루션 연산을 하면 영상에서 어떤 특징을 뽑아낼 수 있기 때문에 컨벌루션을 수행한 결과는 특징맵(feature map)이라고 불린다

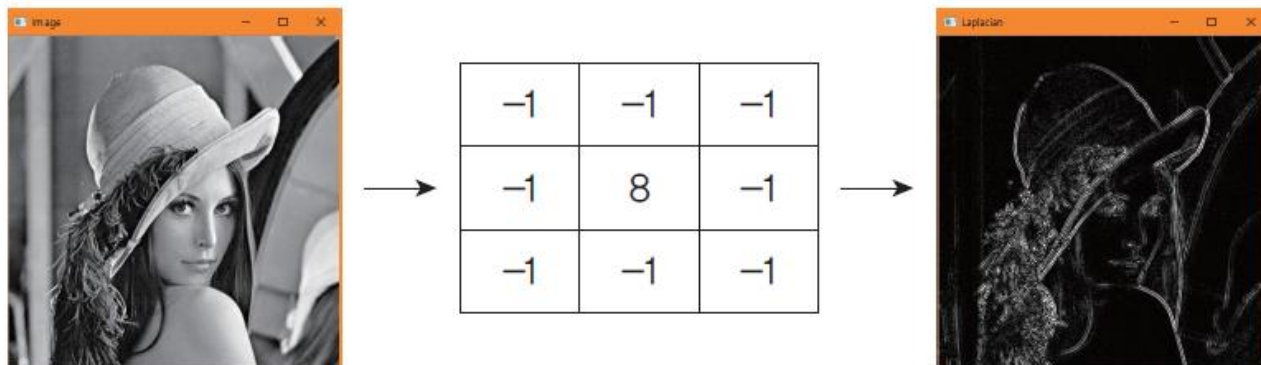


그림 9-9 | 영상 처리에서의 컨벌루션 연산



영상 처리에서의 컨벌루션 연산

- 컨벌루션 신경망에서는 필터의 가중치가 미리 결정되는 것이 아니다. 백지 상태에서 출발하여 샘플을 이용한 훈련(학습)과정을 통해 필터의 가중치를 구해야 한다.

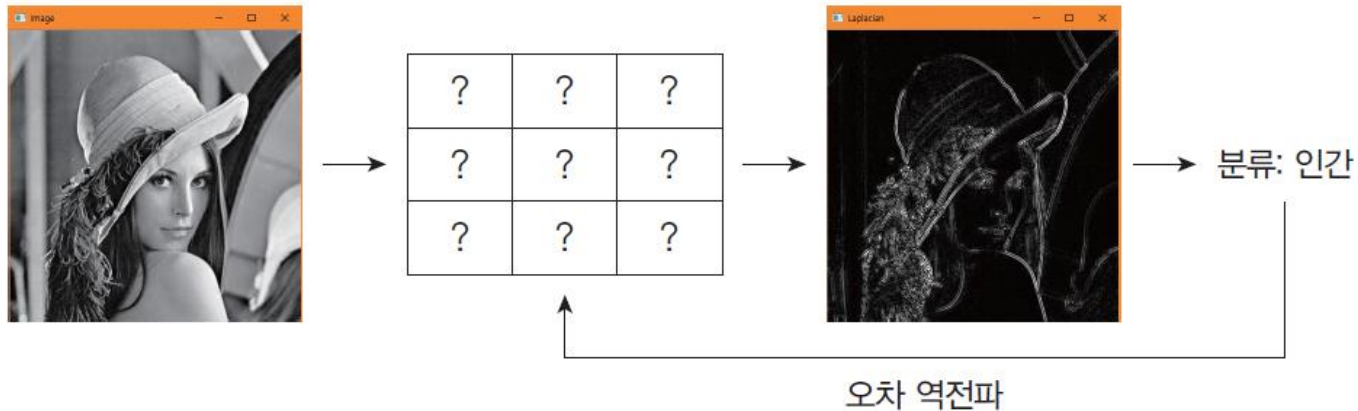


그림 9-10 컨벌루션 신경망에서는 커널의 가중치들이 학습된다.



컨벌루션 신경망에서의 컨벌루션 연산

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

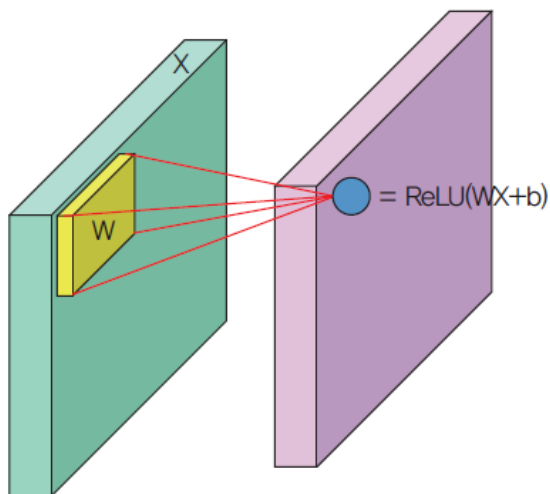
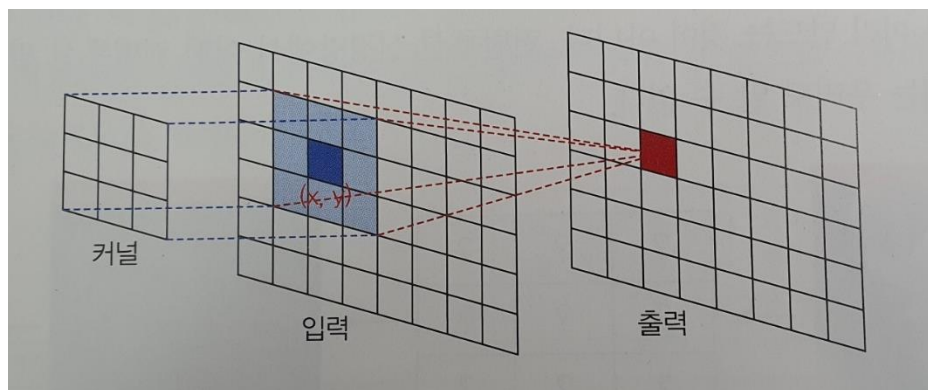


그림 9-11 신경망에서의 컨벌루션 연산





컨벌루션 신경망에서의 컨벌루션 연산

- 가중치 필터는 한 칸씩 이동하여 동일한 연산을 되풀이 한다.

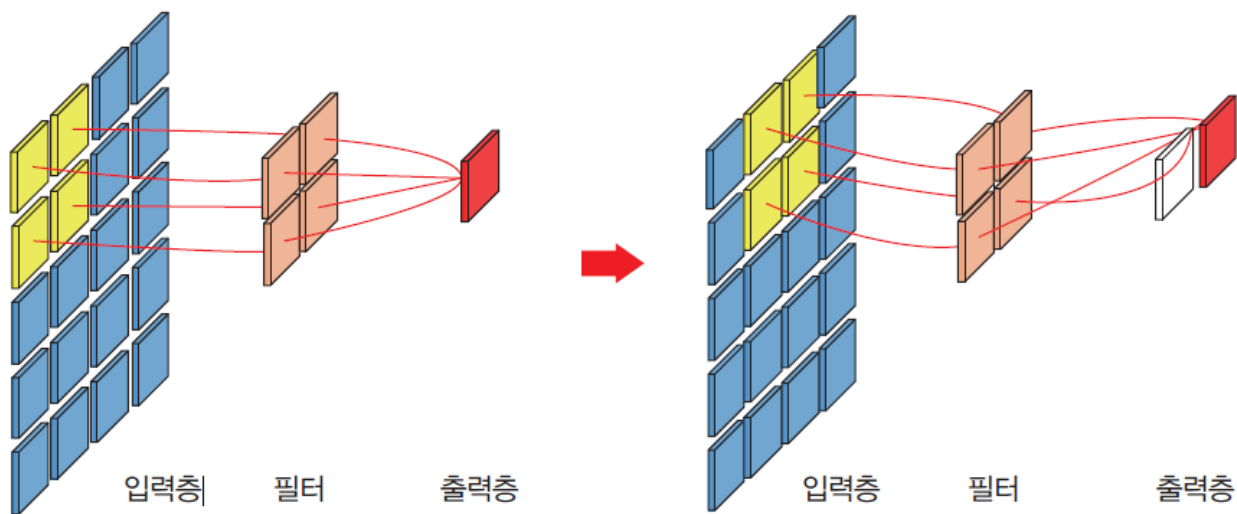
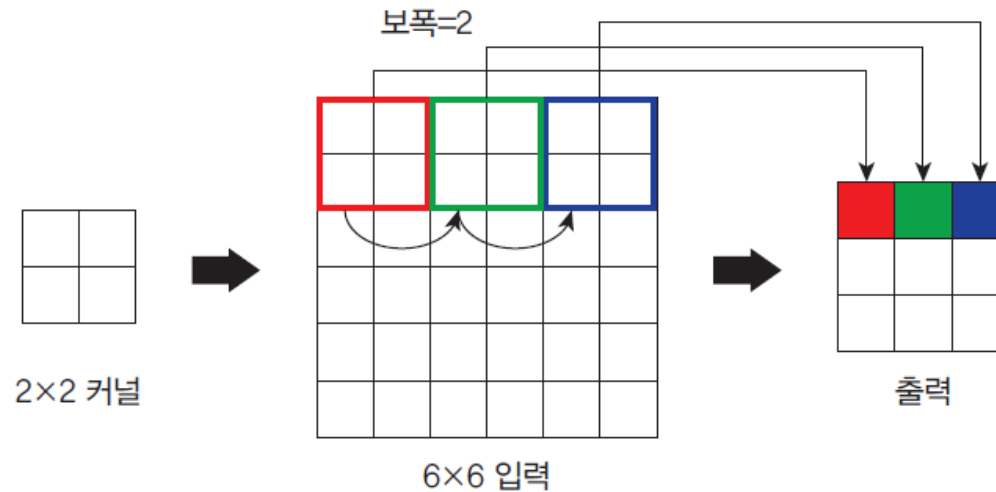


그림 9-12 컨벌루션 연산

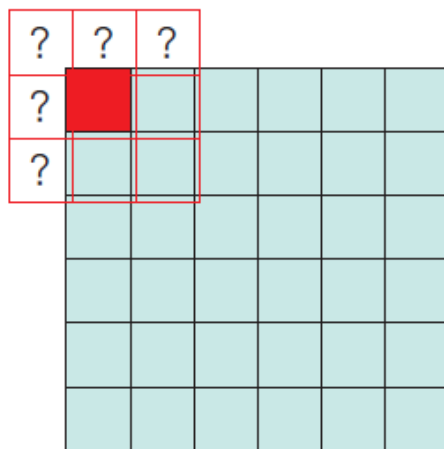
보폭(stride)

- 커널을 적용하는 거리. 보폭이 1이면 커널을 한 번에 1픽셀씩 이동하면서 커널을 적용한다.
- 출력크기 = $\frac{\text{입력크기} - \text{커널크기}}{\text{보폭}} + 1$



패딩(padding)

- 이미지의 가장자리를 처리하기 위한 기법



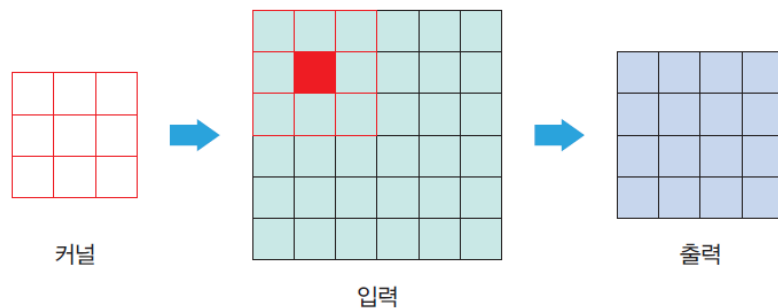
이미지의 가장 자리에 커널을
적용하려니, 커널 아래에 픽셀이
없네요. 어떻게 해야 할까요?



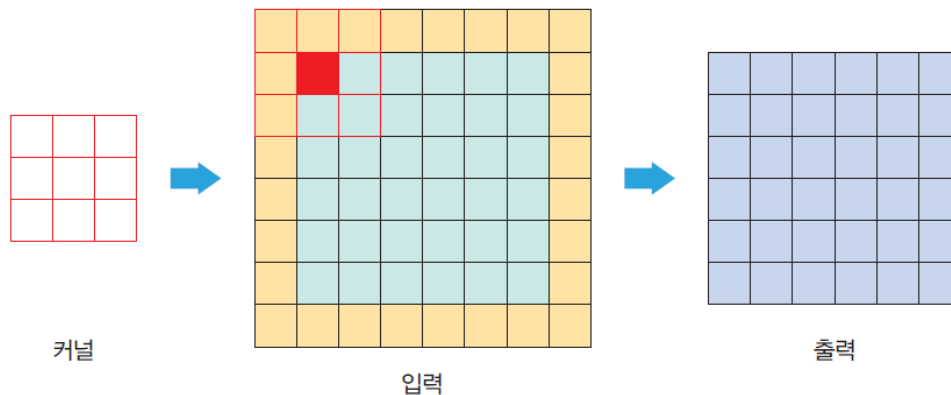
그림 9-13 패딩이 필요한 이유

2가지 패딩 방법

- **valid** : 커널을 입력 이미지 안에서만 움직인다. 컨벌루션 후에 출력은 작아진다.

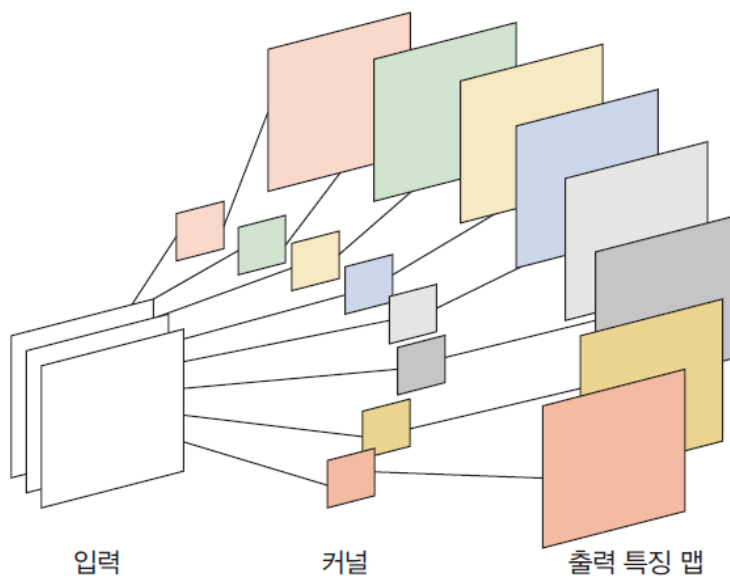


- **same** : 입력 이미지의 주변을 특정값(예를 들면 0, 또는 이웃 픽셀값)으로 채우는 것. 컨벌루션 후에 입력과 출력의 크기는 같아진다.



커널의 개수

- 하나의 커널은 하나의 특징만 추출한다.
- 입력 영상이 하나의 특징만 갖는 경우는 없기 때문에 여러 개의 커널을 동시에 학습하는 경우가 일반적



커널이 많아지면
특징맵의 개수도 많아진다.

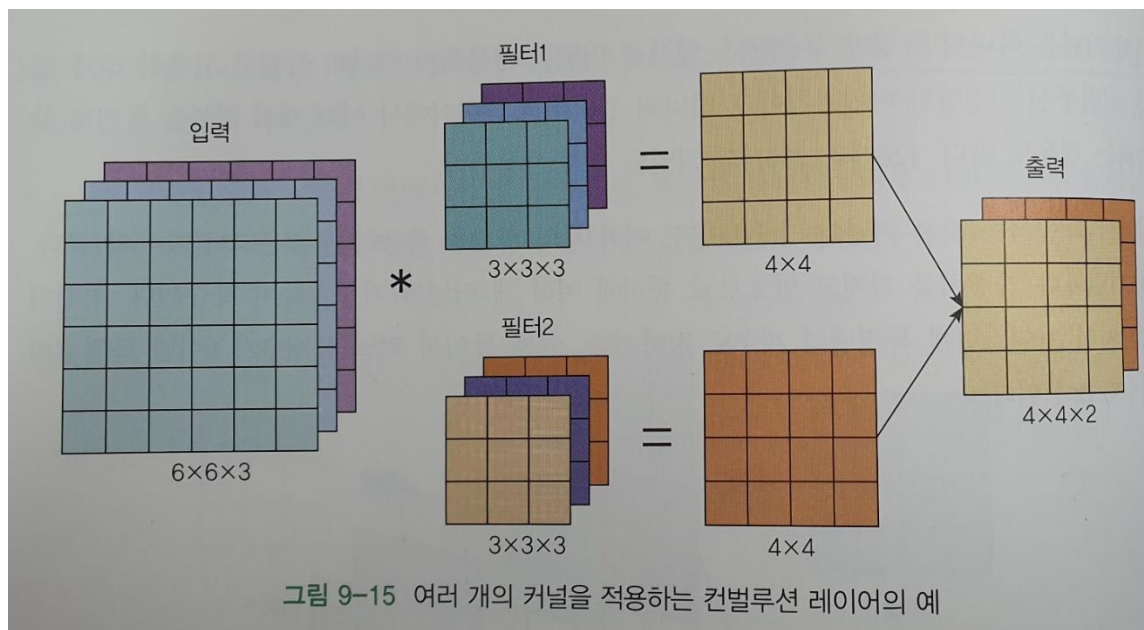
그림 9-14 필터가 여러 개일 때의 컨볼루션 레이어



필터가 여러 개일 때의 컨벌루션 레이어

예.

입력 = 컬러영상. 6×6 . (R, G, B) 채널. 커널 1개 = 3×3 . (R, G, B) 채널.
valid 패딩이면 출력 = 4×4 . \rightarrow 커널이 2개이면 전체 출력은 $4 \times 4 \times 2$



풀링(서브 샘플링)

- 입력 데이터의 크기를 줄이는 것이다.

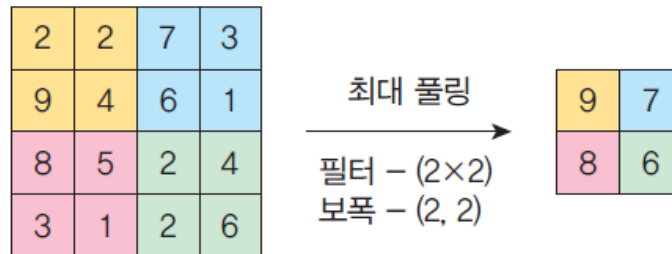


그림 9-17 풀링 레이어

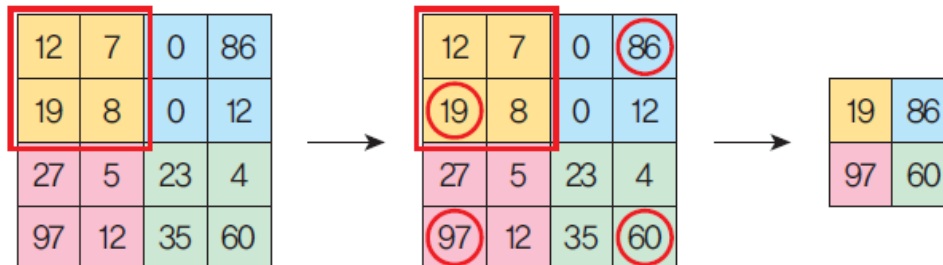


그림 9-18 풀링 연산

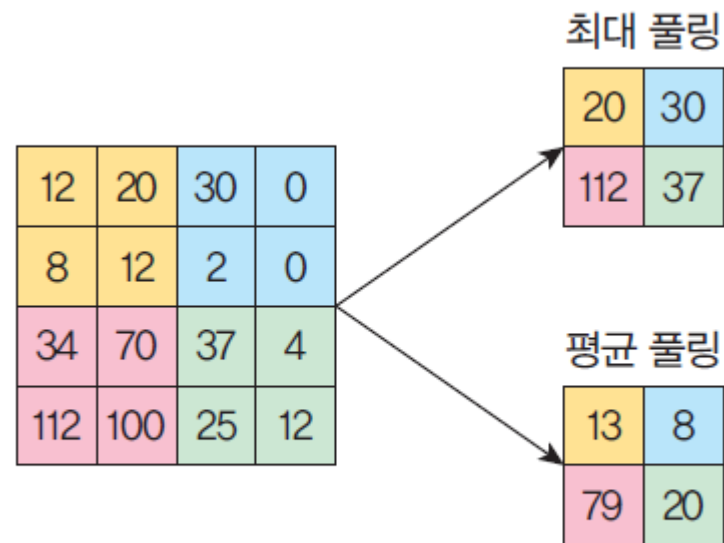


그림 9-19 폴링의 종류

- 레이어의 크기가 작아지므로 계산이 빨라진다.
- 공간에서 물체의 이동이 있어도 결과는 변하지 않는다. 즉 물체의 공간이동에 대하여 둔감해지게 된다.

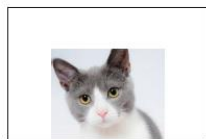
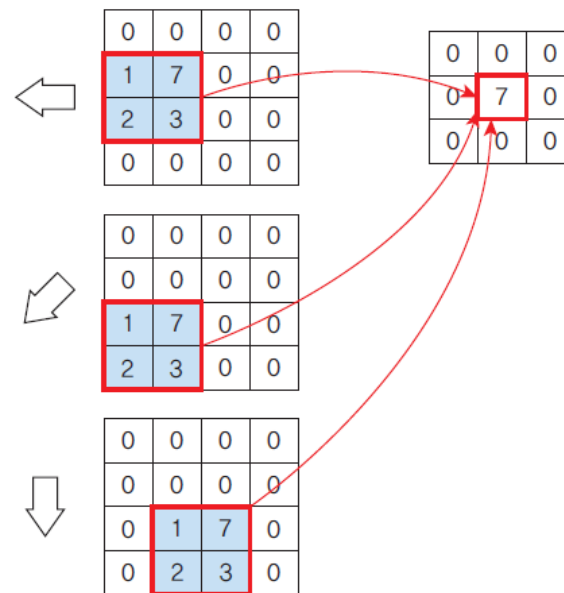


그림 9-20 평행이동된 영상





컨벌루션 신경망을 해석해보자

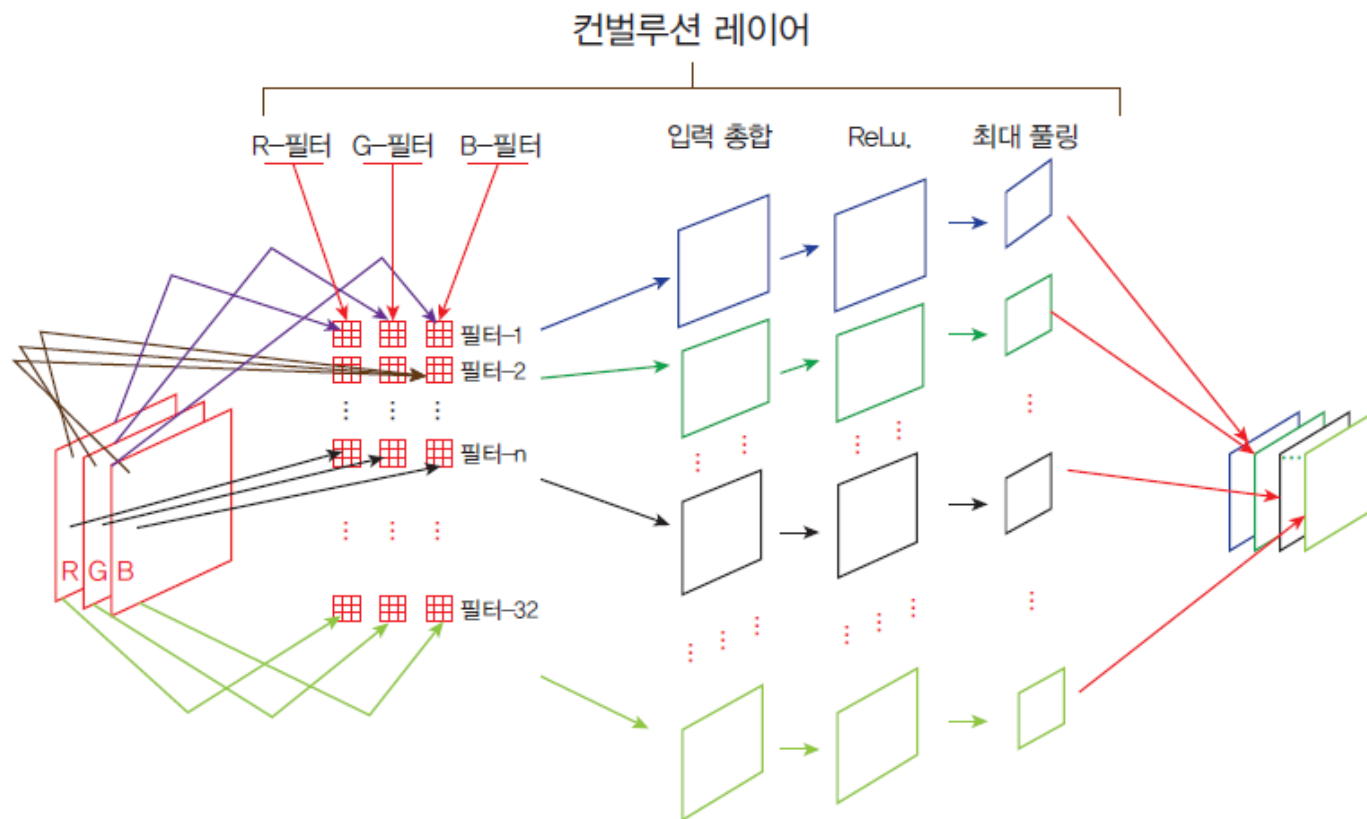


그림 9-21 컨벌루션 레이어의 분석



AlexNet 신경망의 해석

- 2012년도에 경이적인 영상 인식을 달성
- 입력 데이터: 깊이 3, 크기 227×227
- 8개의 레이어(5개의 컨볼루션 레이어, 3개의 완전연결 레이어)

Full (simplified) AlexNet architecture:

[$227 \times 227 \times 3$] INPUT

[$55 \times 55 \times 96$] **CONV1**: 96 11×11 filters at stride 4, pad 0

[$27 \times 27 \times 96$] **MAX POOL1**: 3×3 filters at stride 2

[$27 \times 27 \times 96$] **NORM1**: Normalization layer

[$27 \times 27 \times 256$] **CONV2**: 256 5×5 filters at stride 1, pad 2

[$13 \times 13 \times 256$] **MAX POOL2**: 3×3 filters at stride 2

[$13 \times 13 \times 256$] **NORM2**: Normalization layer

[$13 \times 13 \times 384$] **CONV3**: 384 3×3 filters at stride 1, pad 1

[$13 \times 13 \times 384$] **CONV4**: 384 3×3 filters at stride 1, pad 1

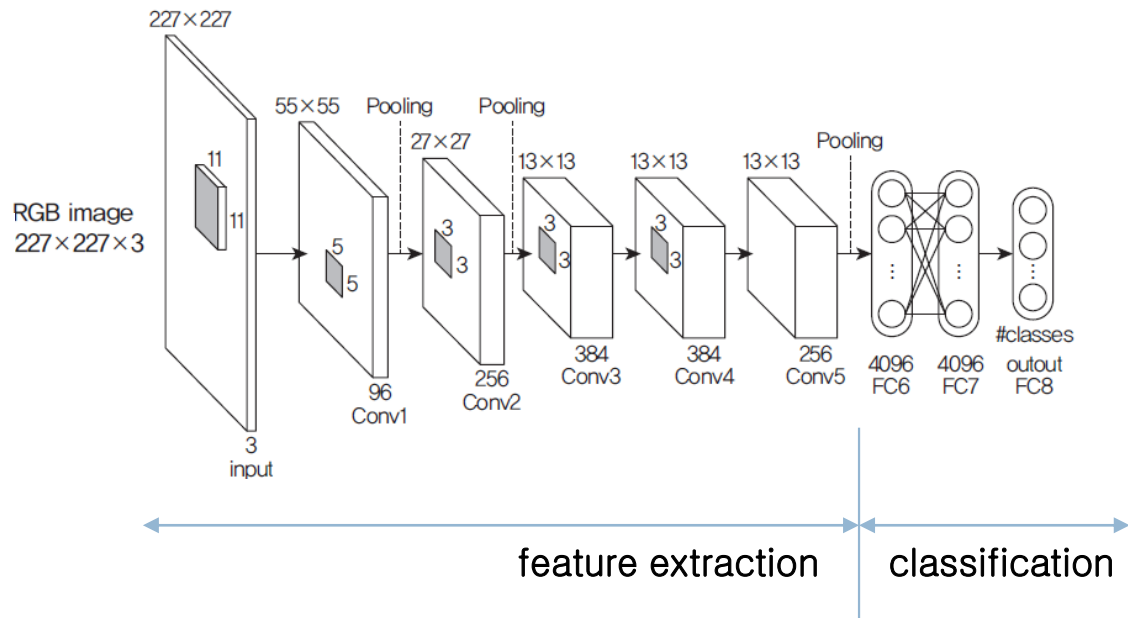
[$13 \times 13 \times 256$] **CONV5**: 256 3×3 filters at stride 1, pad 1

[$6 \times 6 \times 256$] **MAX POOL3**: 3×3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



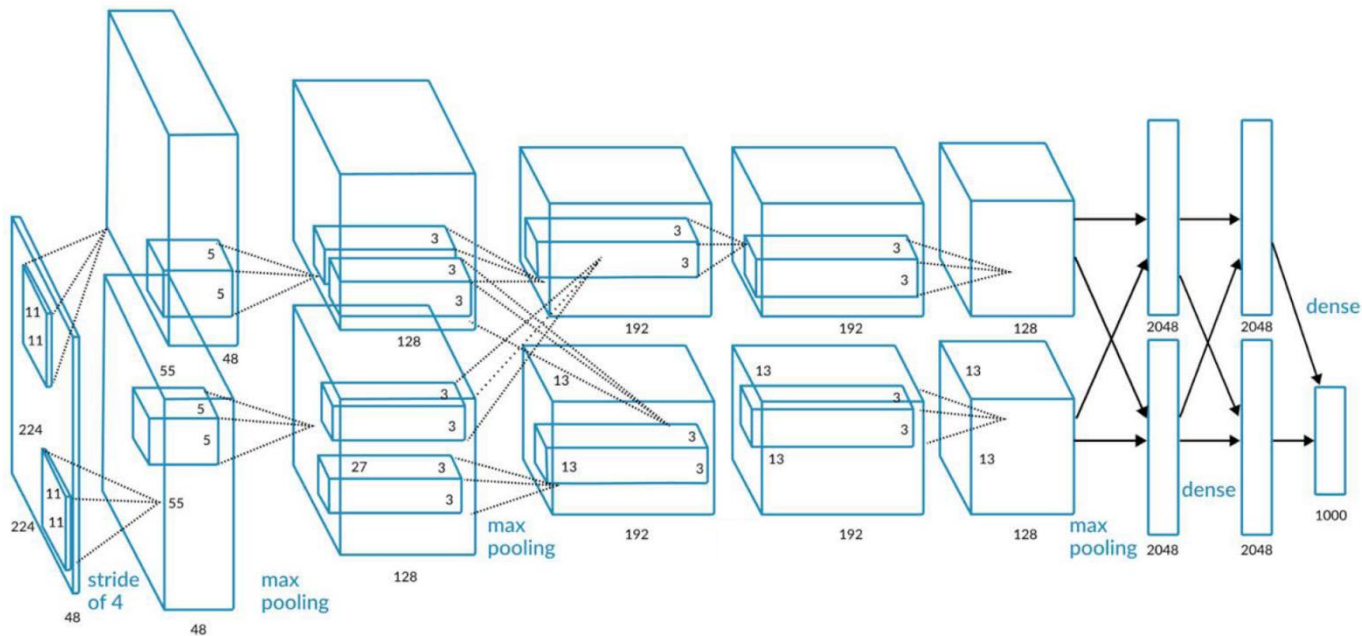


- AlexNet, GoogLeNet, ResNet, InceptionNet ...

<https://transcranial.github.io/keras-js/#/>

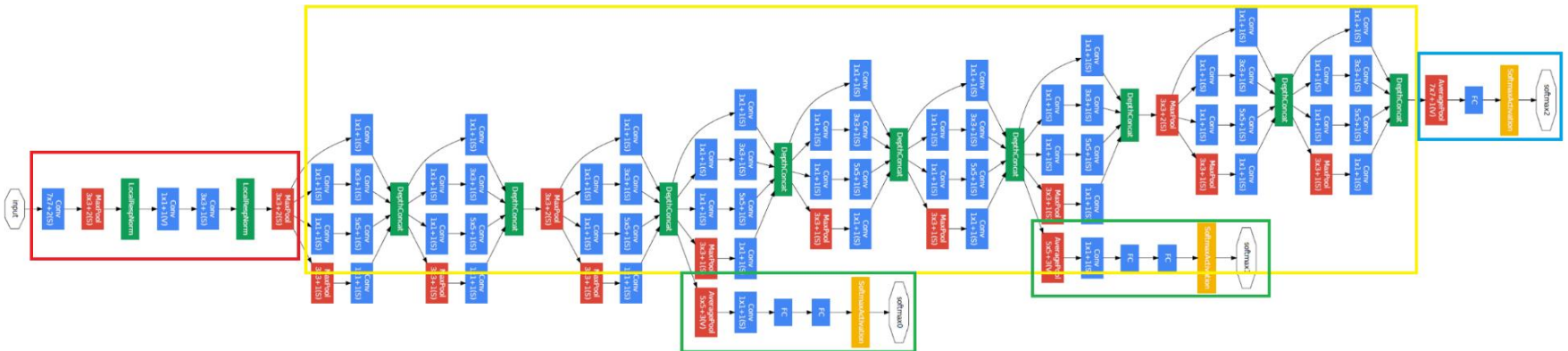
AlexNet

- ImageNet Challenge에서 사실상 최초로 만든 Deep Neural Network 모델
- 2012년, 8 layers, 오류율 16.4%
- 2개의 cpu로 병렬연산 구조



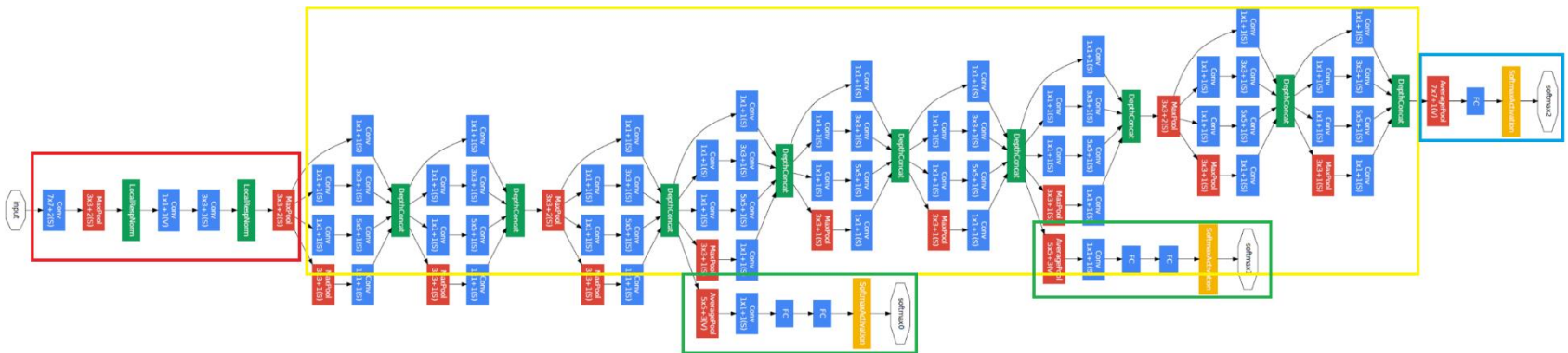
GoogLeNet

- InceptionNet으로도 부른다. 모델이 점점 더 깊어진다는 뜻에서 inception module이라는 것을 사용
- Google에서 만들었으며 한 layer에서 여러 크기의 Convolution filter를 사용하여 성능을 높였다.
- 2014년, 19 layers, 오류율 6.7%.



ResNet

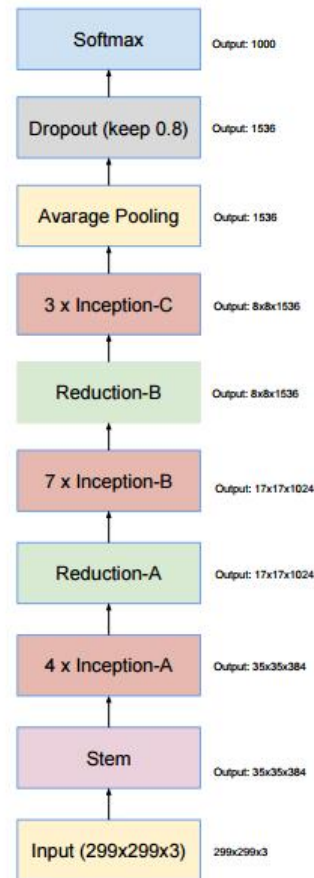
- Microsoft에서 개발한 모델
- 2015년, 152 layers, 오류율 3.6%, 최초로 사람의 분류 성능을 뛰어넘은 모델로 평가
- 매우 깊은 모델(152층)이며, 이 이후부터는 매우 깊고 큰 모델을 사용하게 된다





InceptionNet v2,3, v4

- GoogleNet(Inception)을 개량한 버전으로 ResNet보다 더 높은 성능을 보여준다.
- 2017년





케라스로 컨벌루션 신경망 구현하기

	클래스 이름	설명
컨벌루션 레이어	Conv1D, Conv2D, Conv3D, SeparableConv1D, SeparableConv2D, DepthwiseConv2D, Conv2DTranspose, Conv3DTranspose	컨벌루션 연산을 구현하는 레이어이다.
풀링 레이어	MaxPooling1D, MaxPooling2D AveragePooling1D, AveragePooling2D GlobalMaxPooling1D, GlobalMaxPooling2D GlobalAveragePooling1D, GlobalAveragePooling2D	몇 개의 값을 하나로 합치는 레이어이다.



- `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. 디폴트 = 1
 - `activation`: 유닛의 활성화 함수이다.
 - `input_shape`: 입력 배열의 형상.(배치크기 x 이미지높이 x 이미지너비 x 채널수)
 - `Padding`: 패딩 방법을 선택한다. 디폴트는 “valid”이다.



컨버루션 레이어

디폴트 사용(padding=valid, strides=1)

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)
```

$$\frac{28 - 3}{1} + 1 = 26$$

(4, 26, 26, 2)

padding="same" 사용

```
input_shape = (4, 28, 28, 3)
```

```
x = tf.random.normal(input_shape)
```

```
y = tf.keras.layers.Conv2D(2, 3, activation='relu', padding="same",
```

```
input_shape=input_shape[1:])(x)
```

```
print(y.shape)
```

$$\frac{30 - 3}{1} + 1 = 28$$

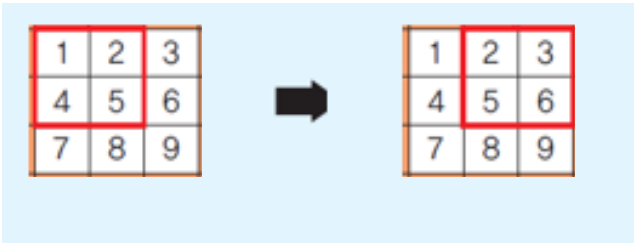
(4, 28, 28, 2)



- `tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid")`
 - `pool_size`: 풀링 윈도우의 크기, 정수 또는 2개 정수의 튜플이다. (2, 2) 라면 2x2 풀링 윈도우에서 최대값을 추출한다.
 - `strides`: 보폭, 각 풀링 단계에 대해 풀링 윈도우가 이동하는 거리를 지정한다.
 - `padding`: "valid"나 "same" 중의 하나이다. "valid"는 패딩이 없음을 의미한다. "same"은 출력이 입력과 동일한 높이 / 너비 치수를 갖도록 입력의 왼쪽 / 오른쪽 또는 위 / 아래에 균일하게 패딩한다.

```
# padding='valid'로 풀링
```

```
x = tf.constant([[1., 2., 3.], [4., 5., 6.], [7., 8., 9.]])  
x = tf.reshape(x, [1, 3, 3, 1])  
max_pool_2d = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1),  
                                              padding='valid')  
print(max_pool_2d(x))
```



```
[[[[5.]  
   [6.]  
   [8.]  
   [9.]]],  
 shape=(1, 2, 2, 1),  
 dtype=float32)
```

padding='same'로 풀링

```
x = tf.constant([[1., 2., 3.], [4., 5., 6.], [7., 8., 9.]])
x = tf.reshape(x, [1, 3, 3, 1])
max_pool_2d = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1),
                                             padding='same')
print(max_pool_2d(x))
```

tf.Tensor(
[[[5.]
[6.]
[6.]

[[8.]
[9.]
[9.]]

[[8.]
[9.]

[9.]]], shape=(1, 3, 3, 1), dtype=float32)

1	2	3	
4	5	6	
7	8	9	

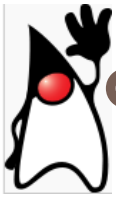


1	2	3	
4	5	6	
7	8	9	

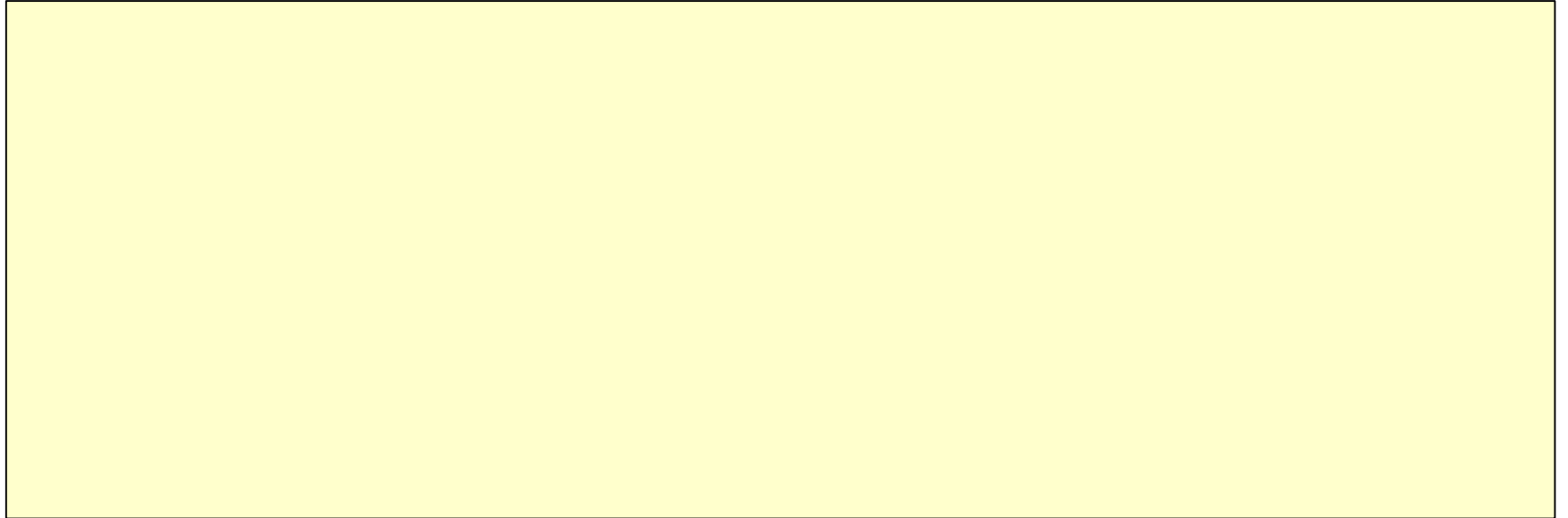


1	2	3	
4	5	6	
7	8	9	

결과가 맞는지
확인해봅시다.



예제: MNIST 필기체 숫자 인식



```

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

B model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2))) # strides 생략하면 pool size와 같은 값이 된다.

C model.add(layers.Conv2D(64, (3, 3), activation='relu'))

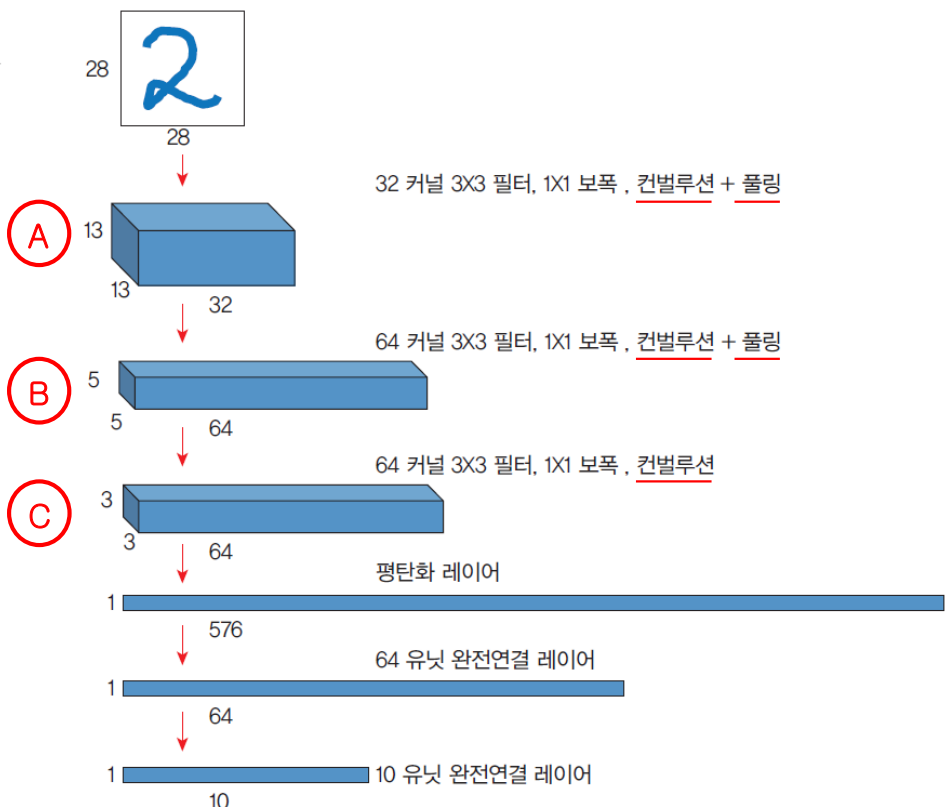
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

```

Model: "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
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: 93322 (364.54 KB)
 Trainable params: 93322 (364.54 KB)





<http://taewan.kim/post/cnn/> CNN, Convolutional Neural Network 요약