



# 신경망 부흥의 시작, 합성곱 신경망

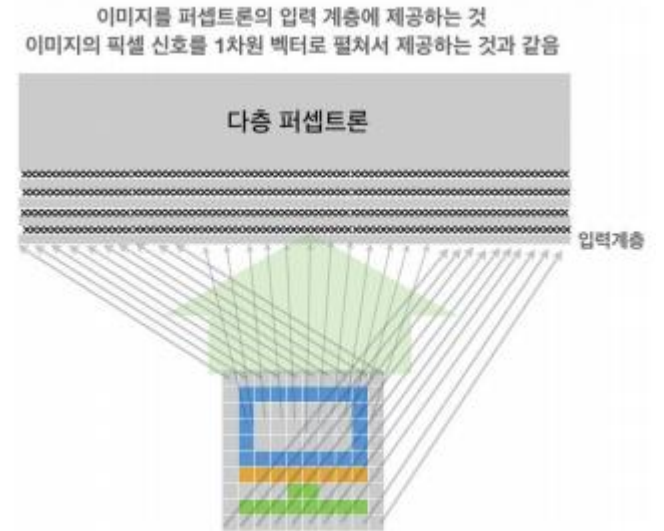
## 이장에서 배울 것들

- 이미지 인식에 신경망을 적용할 때의 문제는 무엇일까.
- 합성곱 신경망은 어떤 원리로 동작하는가.
- 합성곱 신경망이 잘 동작하는 이유는 무엇인가.
- 패딩과 풀링은 신경망의 학습에 어떤 영향을 미치는가.

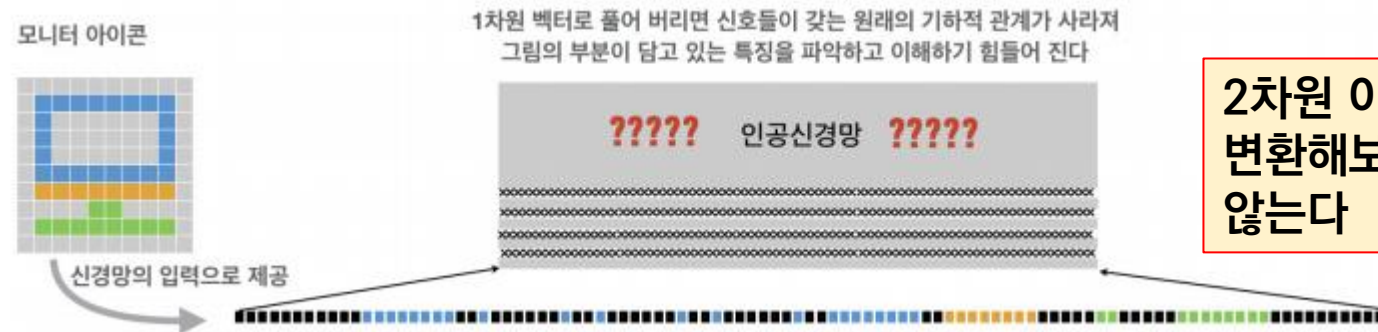
# 시각 정보 처리와 합성곱의 필요성

- 이미지 분석 정확도를 획기적으로 개선하며 신경망 분야의 새로운 도약을 이끈 **컨볼루션 신경망**Convolutional neural network:CNN 혹은 **합성곱 신경망**을 살펴보자
- 인간이나 동물이 어떤 방식으로 눈으로 본 사물을 '인식'하는가에 대해서는 오랜 시간동안 사람들의 궁금증을 자아냄
- 1950년대 말 데이비드 허벨David Hubel과 토르스텐 비셀Torsten Wiesel은 고양이에 대한 실험을 통해 시각 정보가 들어올 경우, **시각 피질**visual cortex에 있는 뉴런들이 시야의 일부 범위 안에 있는 영역에 대해 활성화되어 정보를 받아들인다는 것을 알게 됨
- 하나의 뉴런을 활성화시키는 데에 영향을 미치는 시각 정보의 영역을 **수용장**receptive field
  - 예를 들면 어떤 뉴런은 수평선 이미지에 반응하고, 어떤 뉴런은 수직선 이미지에 반응하며 이러한 낮은 수준의 이미지에 대한 반응을 조합하는 더 큰 수용장이 패턴에 반응한다는 결과

- 합성곱 신경망은 이러한 실험의 결과로 알게된 동물의 시각정보 처리 방식을 이미지 인식에 적용한 것
  - 시각 수용장의 역할을 합성곱 혹은 **컨볼루션convolution**이라는 연산이 수행해 그 결과가 신경세포로 전달
  - 이 방법은 다른 영상 분류 알고리즘에 비해 상대적으로 입력에 대한 전처리가 거의 필요하지 않음.
- 신경망은 이미지를 잘 다루기 위해 특징을 추출하여 입력으로 만드는 단계가 필요
- 합성곱 신경망은 학습 과정에 입력을 특징을 추출하는 방법도 함께 학습
  - **특징 추출해 내는 과정feature engineering**이 필요하지 않으며 이것이 주 장점
- 기존의 신경망이 가진 문제를 그림으로 살펴 보자

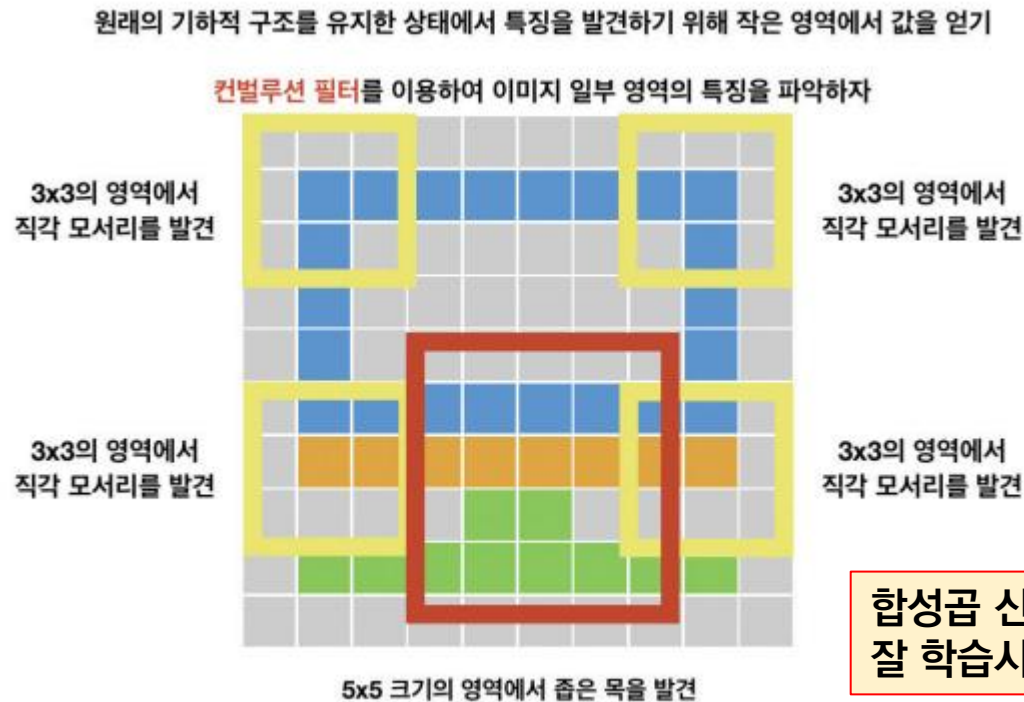


- 펼쳐친 1차원 벡터 형태에서는 모니터의 특징을 찾기가 쉽지 않다



2차원 이미지를 1차원 벡터로  
변환해보니 특징이 잘 보이지  
않는다

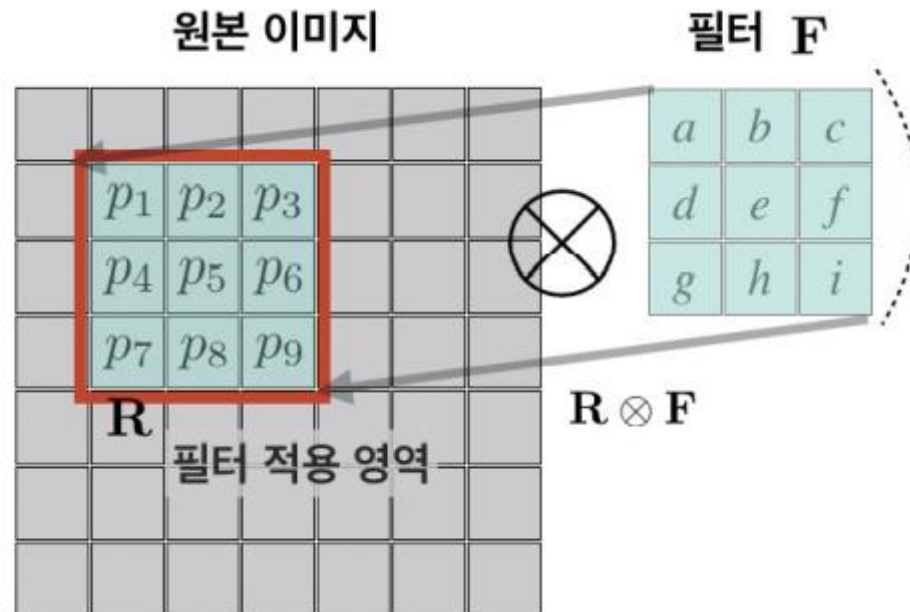
- 이미지 내의 특정한 영역에 대해 원래의 이미지가 가지는 기하적 관계를 유지한 채로 살펴보는 것이 필요하다.
  - 특징을 파악하기 위해 살펴보는 작은 영역을 동물의 시각 처리 방식에서는 수용장이라 할 수 있음
  - 이미지 처리 분야에서 특정한 영역 내에 있는 모든 픽셀pixel 정보로 하나의 값을 생성하는 일을 합성곱이라고 함
- 합성곱은 결국 특정한 영역을 하나의 특징으로 변환하는 역할을 하며 합성곱과 함께 현재는 풀링pooling이라고 부르는 서브샘플링subsampling 계층을 도입



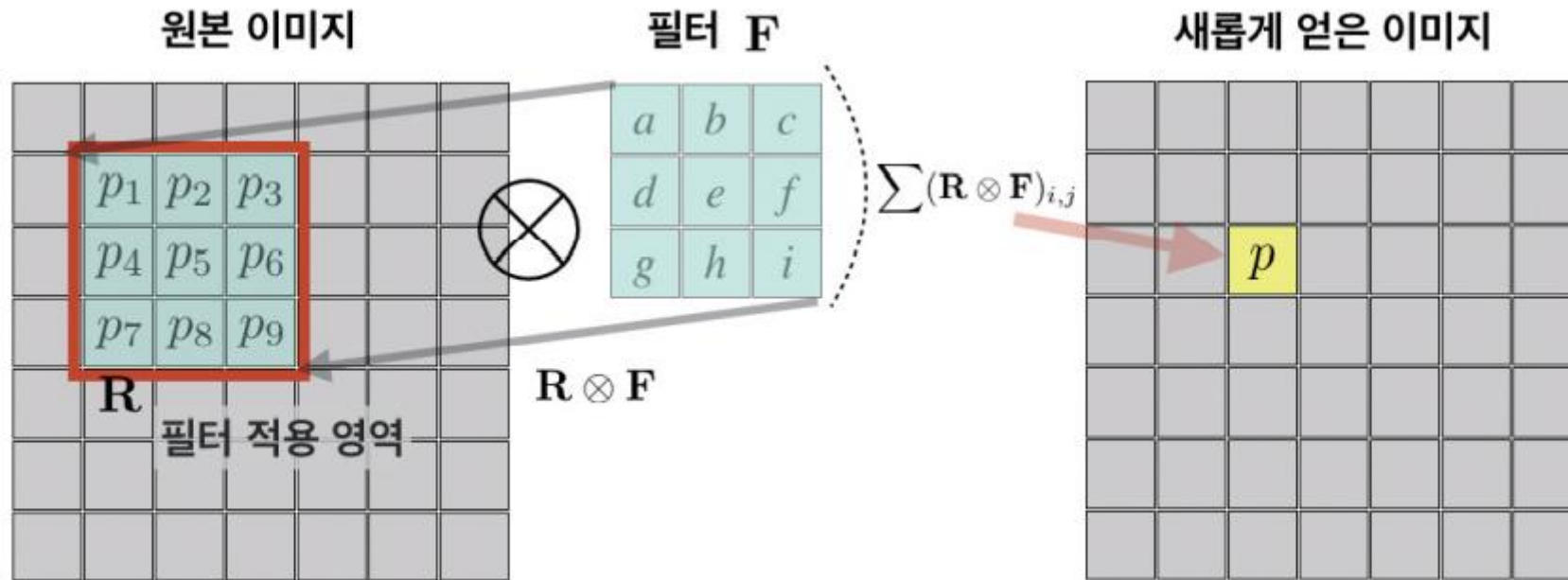
합성곱 신경망 : 2차원 필터를 잘 학습시키는 것이 더 나을 것이다

# 합성곱의 기본 개념 이해

- 이미지를 조작할 때 사용하는 방법 중에 **커널** kernel 혹은 **필터** filter라고 불리는 작은 행렬을 사용하여 새로운 값을 얻는 방법이 있음
- SVM을 다룰 때, 커널이라는 개념을 사용했기 때문에, 혼돈을 피하기 위해 이미지를 처리할 때 사용하는 이 행렬은 필터라고 부름

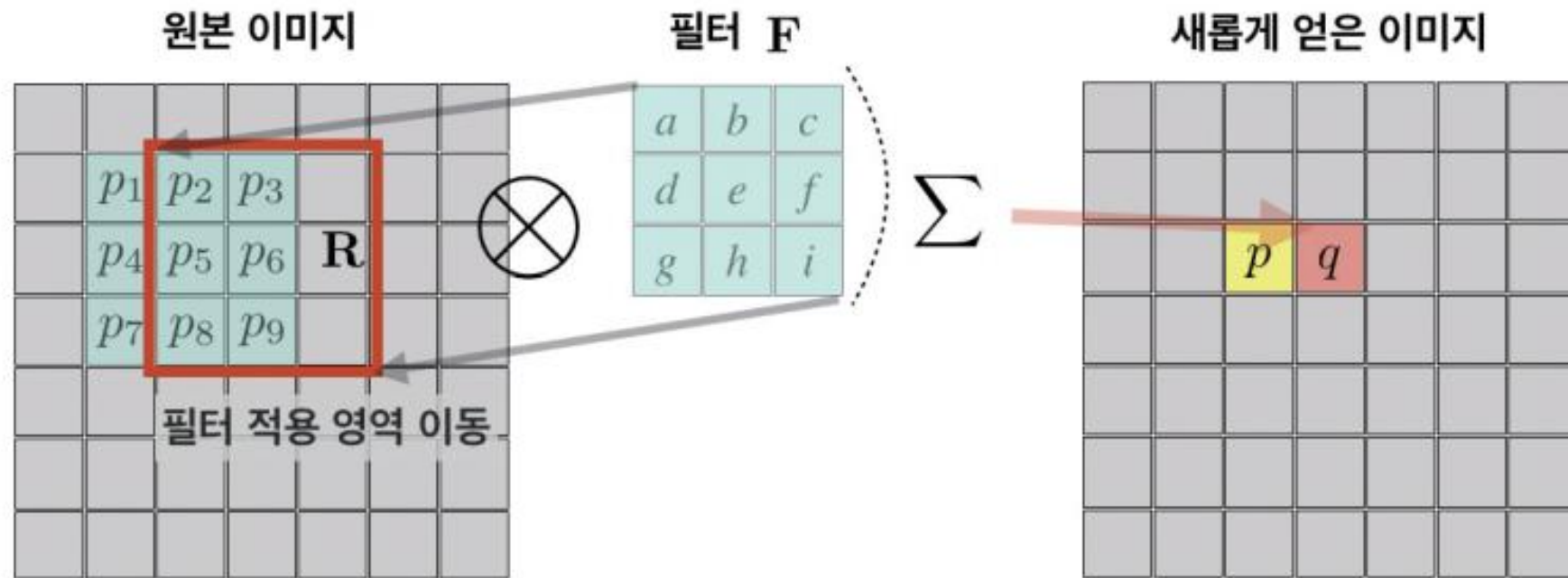


- 필터를 통해 이미지를 변형하는 것은 다음 그림과 같은 방식을 사용
- 필터는  $F$ 로 표시된 작은 크기의 파란색 행렬이며, 이 영역을 원본 이미지에 덮었을 때, 필터가 덮이는 영역을 표시 하는 부분 행렬이  $R$
- 현재  $p_5$  위치의 픽셀 값을 필터가 적용된 이미지의  $p$ 로 바꾸는 작업이 진행되고 있다. 즉, 아래의 수식과 같이 필터와 원본 이미지의 대응되는 값들을 서로 곱한 뒤에 모두 더하여 새로운 이미지의 픽셀값  $p$ 를 구하는 것이며 이러한 연산을 **합성곱 연산**, 혹은 **컨볼루션**이라 함



$$p = a \cdot p_1 + b \cdot p_2 + c \cdot p_3 + d \cdot p_4 + e \cdot p_5 + f \cdot p_6 + g \cdot p_7 + h \cdot p_8 + I \cdot p_9$$

- 다음 픽셀  $q$ 를 구하는 일은 필터를 옆으로 한 칸 옮기면 됨

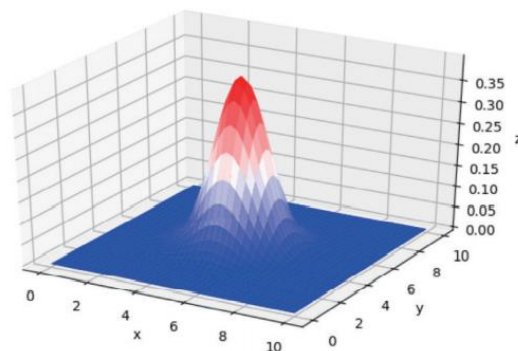




- 가장 간단한 필터링은 하나의 픽셀 값을 변경할 때 주위의 값을 고려하여 평균을 취하는 방법
- 필터 중에서 이렇게 특정 픽셀 주변의 픽셀 들이 가진 값의 평균을 취하는 필터를 **평균 필터**average filter 혹은 **상자 필터**box filter

$$M = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- 가우스 필터는 필터 영역의 모든 픽셀에 동일 한 중요성을 부여하는 것이 아니라 중심 픽셀에는 더 높은 중요도를 부여하고, 중심에서 멀어질 수록 중요성을 낮게함
  - 이 중요도를 **가중치**weight라고 부르고, 가우스 함수를 통해 그 값을 결정
  - 가우스 함수는 왼쪽 그림과 같은 **종 모양**bell shaped 함수 혹은 **정규 분포**normal distribution 함수





원본 이미지

상자 필터를 이용한 합성곱

가우스 필터를 이용한 합성곱

이미지와 영상 처리 분야에서는 필터를 사용하여 이미지를 가공하는 것이 일반적이다

# 합성곱을 통한 특징 추출

- 윤곽선을 찾는 것은 색이 연속적이지 않고 갑작스럽게 변하는 픽셀들을 찾아내는 것
- 이미지는 픽셀 위치를 **정의역**domain으로 하고 픽셀의 색상을 **치역**range으로 하는 함수
  - 픽셀의 각 지점마다 색상 값이 변하는 방향으로 기울기를 구할 수 있음
- 기울기를 구하는 델 연산자는  $\nabla$ (이 기호의 이름은 나블라)
- 값이 크게 변하는 곳에서는 이 기울기 벡터의 크기가 커지며 이를 감지하기 위해 **라플라시안**Laplacian 연산 적용
- 두 번 이상 미분할 수 있는 함수  $f$ 가 있을 때 라플라시안 연산  $\Delta$ 
  - 기울기 연산으로 얻은 값을 자기 자신과 **내적**dot product하는 연산 = 기울기 벡터의 변화가 큰 **윤곽선** 지점에서 큰 값

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$$

$$\nabla = \left( \frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n} \right)$$

$$\Delta f = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}$$

- 각 차원의 변수를  $x_1, x_2, \dots, x_n$ 이라고 부를 수 있는데, 이미지는 2차원 공간에서 정의되므로  $n=2$
- 이미지에 라플라시안 연산을 적용하는 것은 각 픽셀에 대해 각 축으로 두 번 미분한 값을 합산하는 것
- 이미지 생성 함수를 찾아 미분하는 일은 불가능하기 때문에 실제 함수 미분의 근사치가 되는 수치 미분을 적용
  - 픽셀 사이의 간격을 단위 길이 1이라고 가정하고 인접 픽셀의 차를 구하면 바로 1차 미분의 근사치
  - 만들어진 행렬에서 인접 원소의 차를 구하면 2차 미분이다. 따라서  $i$ 행,  $j$ 열에 있는 특정한 픽셀  $I_{i,j}$ 를 행 방향을 의미하는  $x_1$ 축 방향으로 두 번 미분한 값의 근사치는

$$\frac{\partial^2 I_{i,j}}{\partial x_1^2} = (I_{i+1,j} - I_{i,j}) - (I_{i,j} - I_{i-1,j}) = I_{i+1,j} - 2I_{i,j} + I_{i-1,j}$$

- 비슷한 방식으로 열 방향을 의미하는  $x_2$  축 방향으로 두 번 미분한 값

$$\frac{\partial^2 I_{i,j}}{\partial x_2^2} = (I_{i+1,j} - I_{i,j}) - (I_{i,j} - I_{i,j-1}) = I_{i,j+1} - 2I_{i,j} + I_{i,j-1}$$

- 이 두 값을 더한 것이 특정 픽셀의 라플라시안 연산 결과

$$\Delta I_{i,j} = \frac{\partial^2 I_{i,j}}{\partial x_1^2} + \frac{\partial^2 I_{i,j}}{\partial x_2^2} = I_{i,j+1} + I_{i,j-1} + I_{i+1,j} + I_{i-1,j} - 4I_{i,j}$$

- 필터로 표현하면 자기 자신은 -4, 대각선으로 이웃한 값은 1이 되는 다음과 같은 모양의 행렬이다.

$$\mathbf{K}_{Laplacian} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- 필터를 적용하면 아래 그림과 같이 이미지에서 윤곽선 역할을 하는 픽셀들이 강하게 반응
- 시각 정보를 처리할 때 이러한 필터를 적용하여 윤곽선 특징을 추출해 낼 수 있다는 것을 의미



왼쪽 이미지의 윤곽선 특성을 잘 추출한 것이 오른쪽 이미지

- 합성곱 연산을 할 때는 필터로 사용되는 행렬만 바꾸면 여러 가지 다른 결과를 얻을 수 있음
  - 다른 필터를 사용하는 것 = 다른 특징을 추출하는 것
  - 필터의 적용은 필터의 크기와 동일한 크기를 가진 부분 영역에 대해 특정한 특징을 얻는 것
- 이것을 설계하지 않고 기계가 스스로 좋은 필터를 만들어내는 것은 어떨까?
  - 이것은 필터를 모델의 파라미터로 만드는 일
  - 합성곱 신경망의 기본 아이디어

# 합성곱을 만들어보자

## 실습 목표

필터를 다양하게 생성해서 이미지와 합성곱을 구현해보자. 실습을 통해 합성곱 연산의 원리에 대해 깊이 이해해 보자.



## 힌트

맷플롯립은 png 파일을 읽을 수 있는 기능을 제공한다. 이것을 이용하여 이미지를 읽으면 넘파이 배열 형태로 반환이 되는데, 넘파이 배열로 필터를 만들어 이 필터가 이미지를 표현하는 배열에 곱해지도록 한다. 이 책에서는 다양한 이미지 포맷을 다루거나 복잡한 이미지 처리를 하는 것이 목표가 아니므로 맷플롯립만을 사용하지만, 더욱 고급 이미지 처리를 원할 경우에는 Pillow나 OpenCV 패키지를 활용하도록 하라.

교재의 해답 코드를 하나하나 단계별로 수행해 봅니다



## 합성곱 실습

▶ #convolution

```
[ ] import matplotlib.pyplot as plt
import urllib.request
import PIL.Image
import numpy as np
```

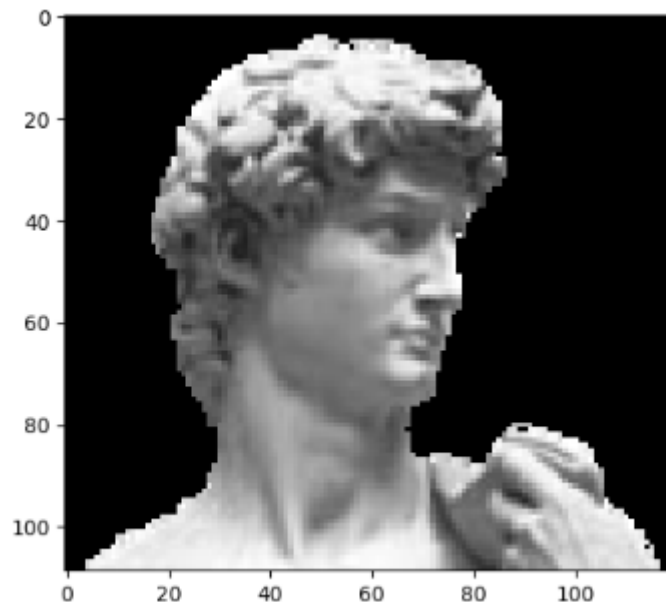
```
[ ] url='https://github.com/dknife/ML/raw/main/data/image/david.png'
```

```
[ ] img = np.array( PIL.Image.open( urllib.request.urlopen(url) ) )
img.shape
```

↗ (109, 120, 2)


```
[ ] plt.imshow(img[:, :, 0], cmap='gray')
```

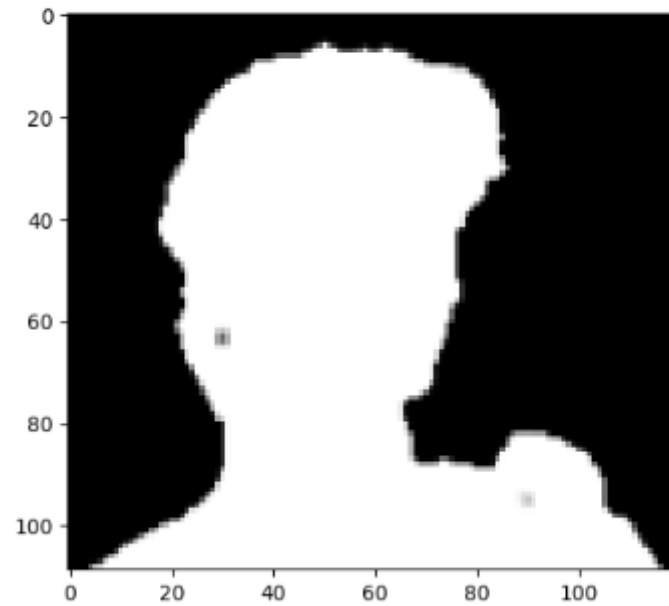
↗ <matplotlib.image.AxesImage at 0x7b1b6c554f40>



## 합성곱 실습

```
[ ] plt.imshow(img[:, :, 1], cmap='gray')
```

 <matplotlib.image.AxesImage at 0x7b1b6c585db0>



# 합성곱 실습

```
[ ] url='https://github.com/dknife/ML/raw/main/data/image/book.png'  
img = np.array( PIL.Image.open( urllib.request.urlopen(url) ) )  
img.shape
```

(533, 800, 3)

```
[ ] plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x7b1b6caa87f0>



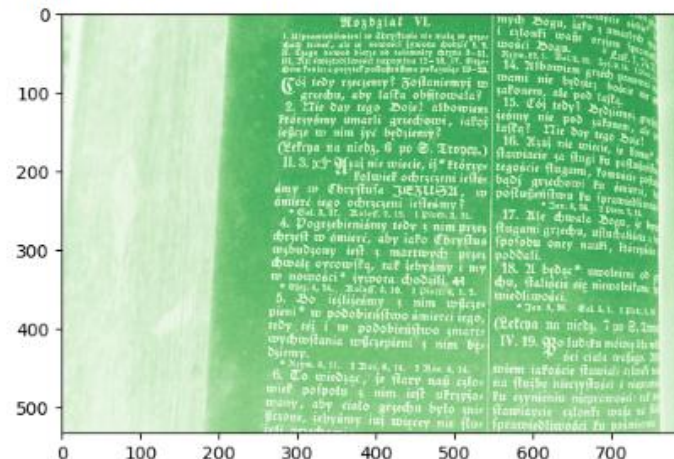
```
[ ] plt.imshow(img[:, :, 0], cmap='Reds')
```

<matplotlib.image.AxesImage at 0x7b1b85a1f670>



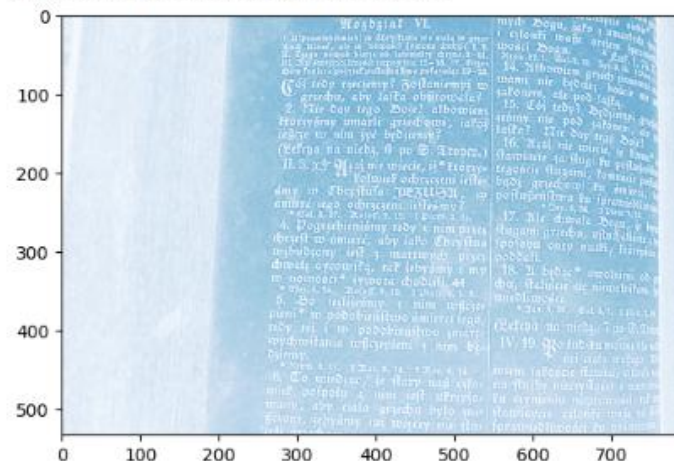
```
[ ] plt.imshow(img[:, :, 1], cmap='Greens')
```

<matplotlib.image.AxesImage at 0x7b1b5fe63070>



```
[ ] plt.imshow(img[:, :, 2], cmap='Blues')
```

<matplotlib.image.AxesImage at 0x7b1b6c110910>

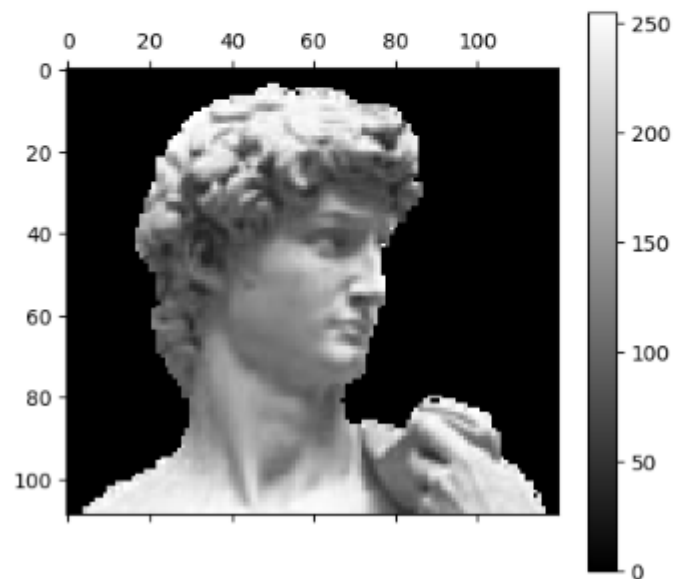


## 합성곱 실습 - 정규화

```
url='https://github.com/dkknife/ML/raw/main/data/image/david.png'  
img = np.array( PIL.Image.open( urllib.request.urlopen(url) ) )  
img.shape  
  
david_intensity = img[:, :, 0]
```

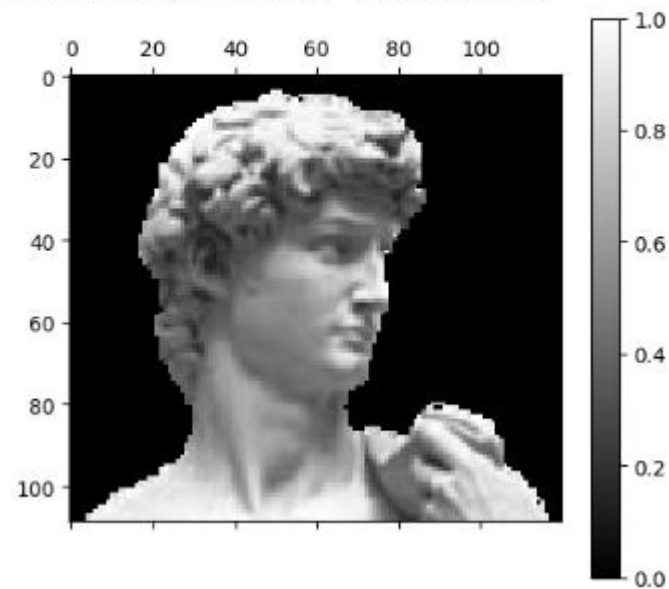
```
[ ] plt.matshow(david_intensity, cmap='gray')  
plt.colorbar()
```

<matplotlib.colorbar.Colorbar at 0x7b1b6c1ef2b0>



```
#정규화 이미지  
david_normalized = david_intensity / 255.0  
plt.matshow(david_normalized, cmap='gray')  
plt.colorbar()
```

<matplotlib.colorbar.Colorbar at 0x7b1b5fb6d270>



# 합성곱 실습

```
[ ] box_filter = np.array([
    [1/9, 1/9, 1/9],
    [1/9, 1/9, 1/9],
    [1/9, 1/9, 1/9]
])
```

```
[ ] def apply_filter(small_region, filter) :
    conv = np.multiply(small_region, filter)
    return np.sum(conv)
```

```
def convolution(img, filter) :
    # 이미지의 행, 열의 수와 채널 수 확인
    if (len(img.shape)==3) : # multichannel
        row, col, channels = img.shape
    else :
        row, col = img.shape
        channels = 1

    # 필터의 행과 열의 수 확인
    rp, cp = filter.shape

    # 컨볼루션을 적용할 수 있는 시작 픽셀 확인
    th_r, th_c = (rp-1)//2, (cp-1)//2

    # 예를 들어 3x3 필터를 사용하면 0,0 픽셀에 적용 불가하고
    # 1,1 픽셀부터 시작해야 함
    # 5x5 필터를 사용하면 2,2 픽셀부터 시작
    # 필터가 적용되는 마지막 픽셀은???
    # 이미지의 마지막 픽셀이 row-1-th_r, col-1-th_c

    start = np.array( [th_r, th_c] )
    end = np.array( [row-1, col-1] ) - start
    print(start, end, rp, cp)

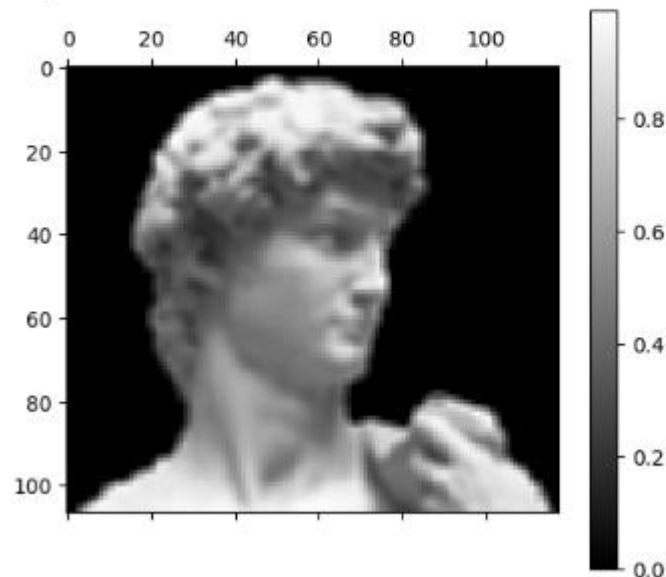
    if channels > 1:
        conv_img = np.zeros( (row - th_r*2, col - th_c*2, channels) )
    else :
        conv_img = np.zeros( (row - th_r*2, col - th_c*2) )

    # 컨볼루션 적용
    for channel in range(channels) :
        for r in range(start[0], end[0]+1) :
            for c in range(start[1], end[1]+1) :
                if channels > 1 :
                    conv_img[r-th_r, c-th_c, channel] = apply_filter(img[r-th_r:r+th_r+1, c-th_c:c+th_c+1, channel], filter)
                else :
                    conv_img[r-th_r, c-th_c] = apply_filter(img[r-th_r:r+th_r+1, c-th_c:c+th_c+1], filter)

    return conv_img
```

```
conv_img = convolution(david_normalized, box_filter)
plt.matshow(conv_img, cmap='gray')
plt.colorbar()
```

```
[1 1] [107 118] 3 3
<matplotlib.colorbar.Colorbar at 0x7b1b4e62d8a0>
```

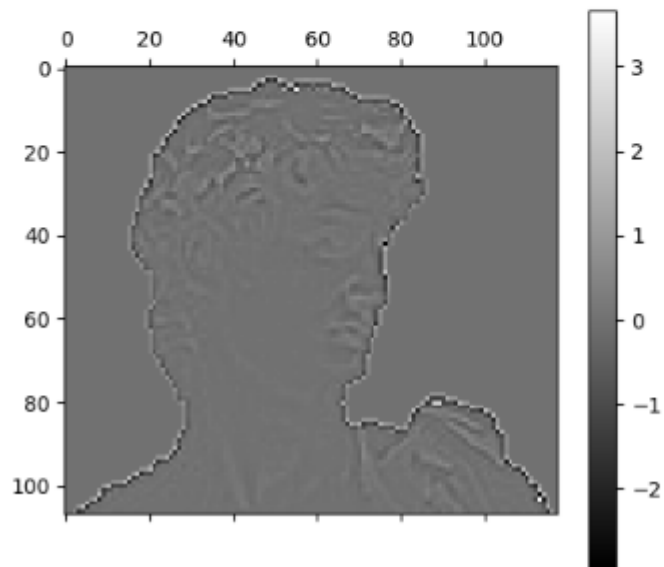


## 합성곱 실습

```
laplacian_filter = np.array([
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]
])
```

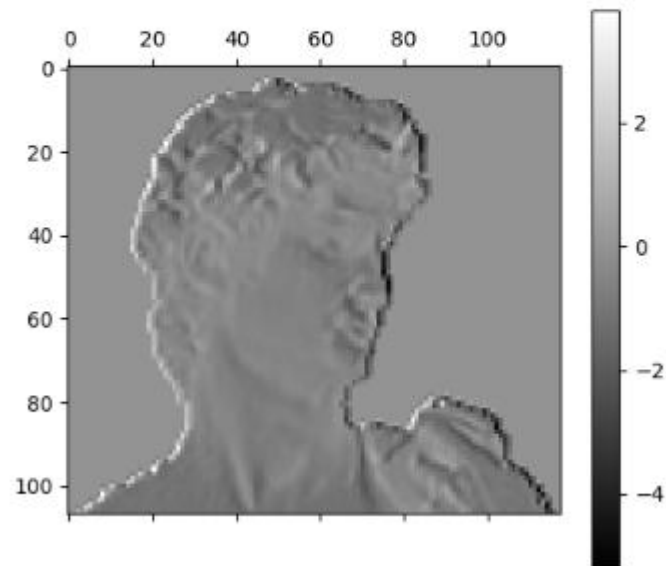
```
conv_img = convolution(david_normalized, laplacian_filter)
plt.matshow(conv_img, cmap='gray')
plt.colorbar()
```

```
[1 1] [107 118] 3 3
<matplotlib.colorbar.Colorbar at 0x7b1b4e4fa920>
```



```
rand_filter = np.random.randn(3, 3)
rand_image = convolution(david_normalized, rand_filter)
plt.matshow(rand_image, cmap='gray')
plt.colorbar()
```

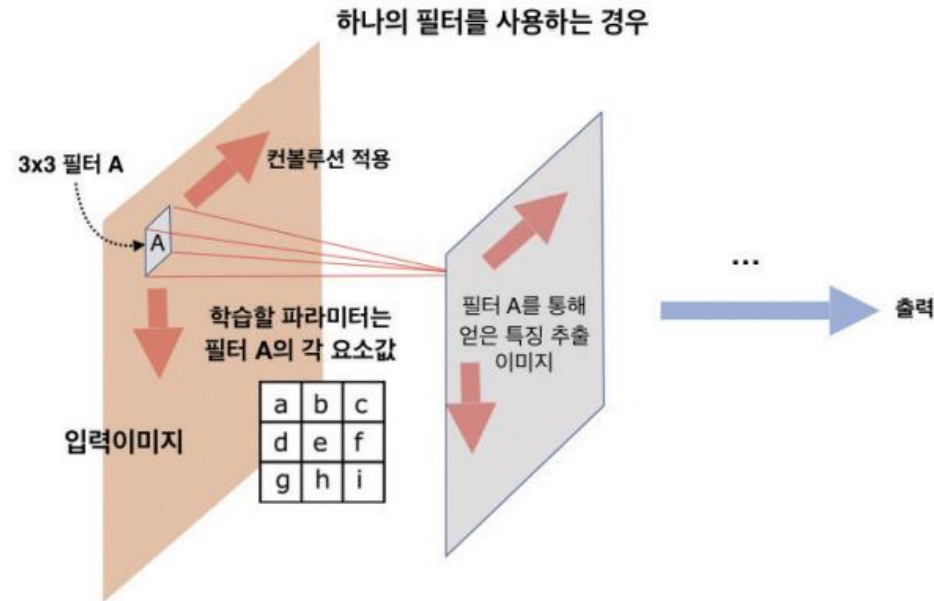
```
[1 1] [107 118] 3 3
<matplotlib.colorbar.Colorbar at 0x7b1b4da4d330>
```



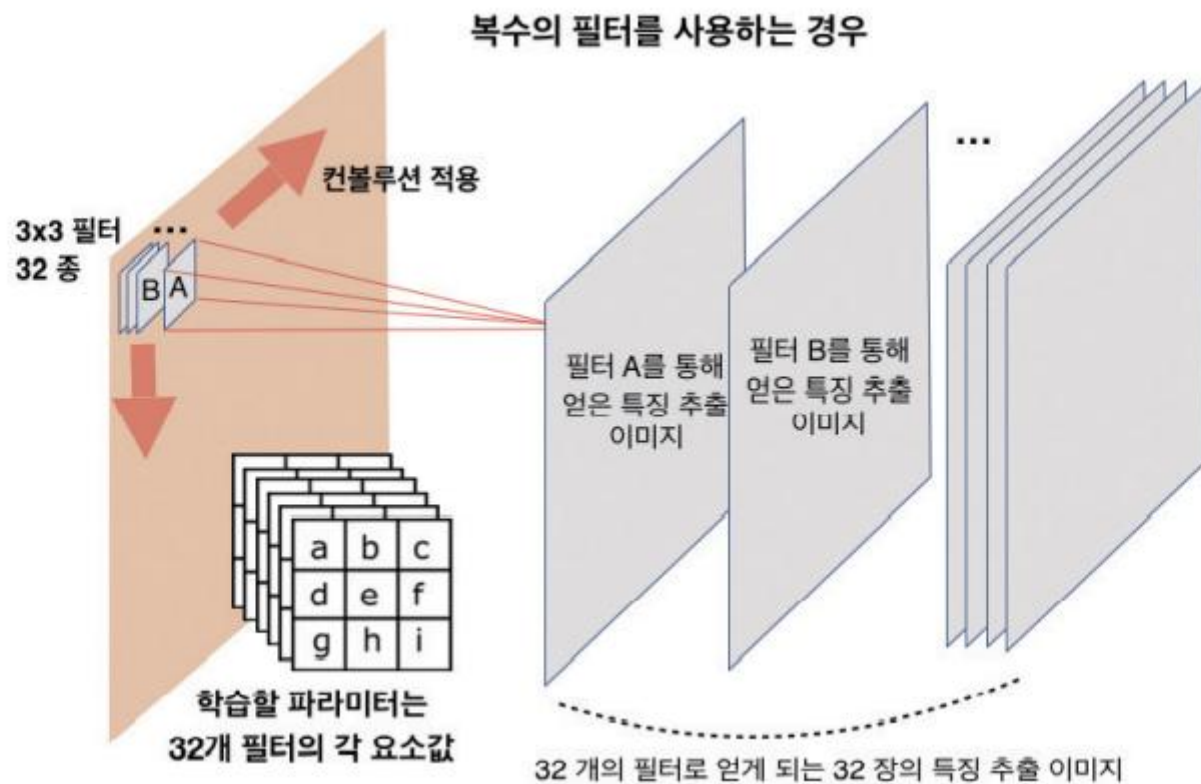


# 합성곱 수행 신경망의 기본 구조와 문제

- 그림은 입력 이미지에 필터 A 를 이용하여 합성곱을 적용하여 특징을 추출한 이미지를 얻어 신경망의 다음 단계로 연결하는 개념을 보임
- 학습의 대상이 되는 것은 필터가  $3 \times 3$ 의 크기라면 9개의 요소를 찾는 일이며 입력을 받아 특징 이미지를 만드는 계층의 연결강도는 9개의 파라미터로 정의



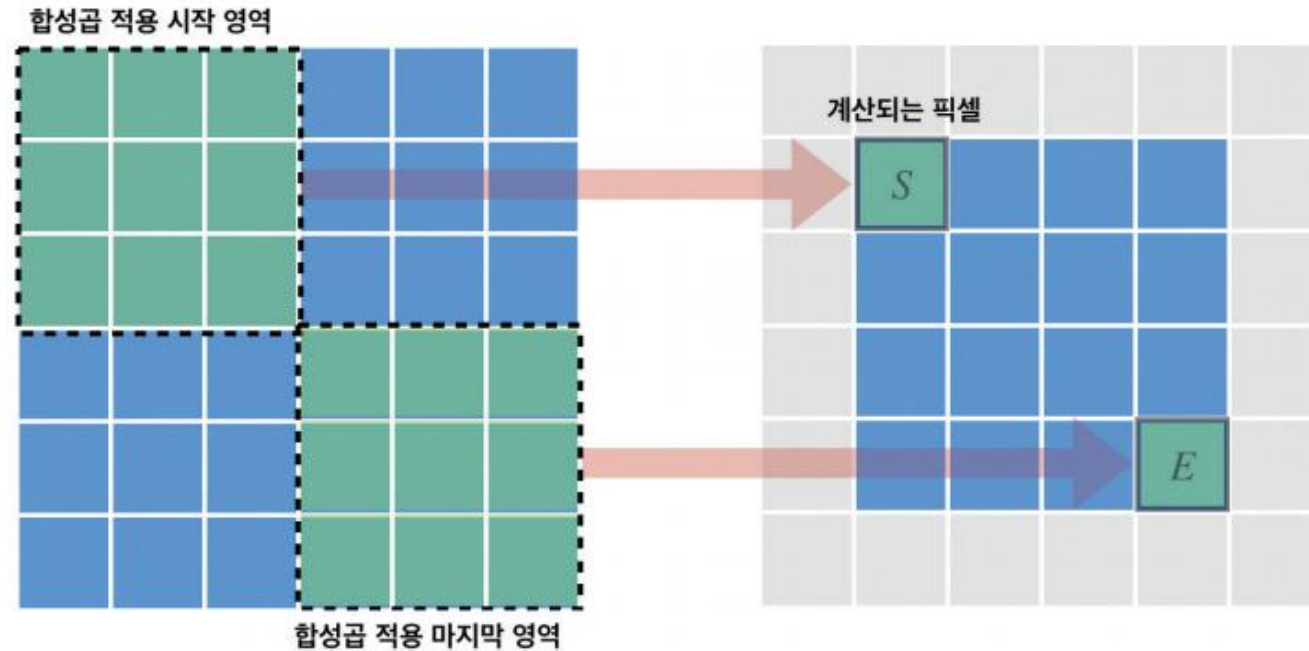
- 하나의 필터만으로는 이미지의 특징을 다양하게 추출할 수 없음
- 그림과 같이 32개의 필터를 모델의 파라미터로 두고 학습한다고 가정했을 때 이미지 하나에 대해 모두 32장의 특징 추출 결과를 얻으며 **특성 맵** feature map 이라 함



학습을 통해 좋은 필터를 만들어 보자



- 이런 구조의 연결을 통해 입력 이미지를 처리하고 얻는 결과는 이미지
- 이 이미지에 대해 다시 합성곱을 수행하는 계층을 계속해서 이어나갈 수 있지만, 이때 발생 할 수 있는 문제에 대해 생각해 보자
- 첫 번째 문제는 합성곱을 적용하면 이미지가 작아진다는 것
- 원래의 이미지가 왼쪽의 파란색 픽셀로 표시된 영역인데, 이 이미지에  $3 \times 3$ 크기의 녹색 점선으로 표시된 필터를 사용하여 합성곱 연산을 적용할 수 있음
- 결과로 얻게되는 픽셀은 오른쪽 그림에 표시된  $S$ 픽셀이 될 것이며 연결망을 계속해서 쌓아나가면 출력에 가까운 곳에서는 이미지가 지나치게 작은 크기로 줄어들게 됨

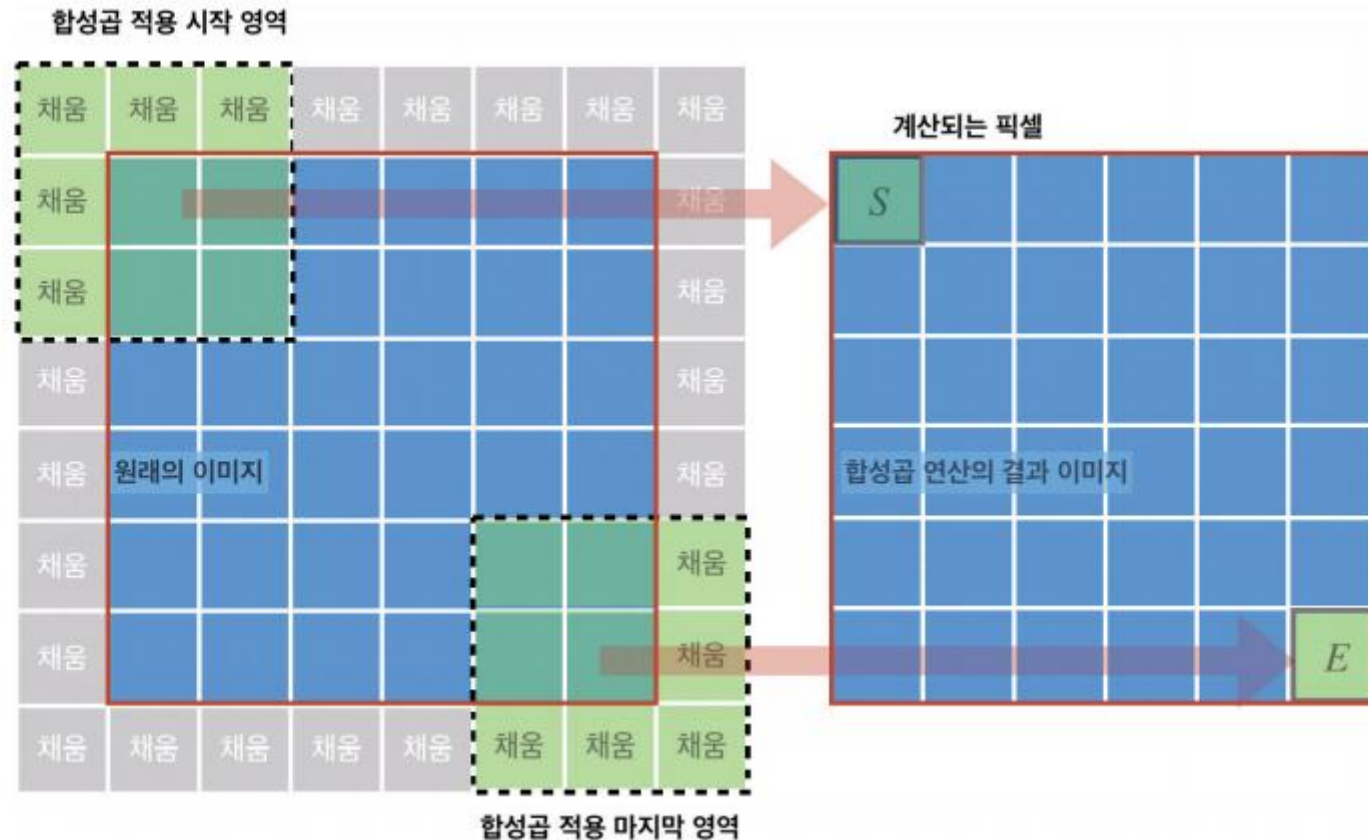


- 합성곱을 거치면서 이미지의 크기가 작아지는 것이 이미지 전체의 특성을 요약한 것이 아님
- 이미지의 가장자리 정보들이 소실되고 중심부의 정보만 살아남는 것이기 때문에 정보를 추상화시키는 과정도 아님
- 정보를 요약하지 않으면 입력 단계의 작은 잡음이 가진 영향력도 계층을 거치며 사라지지 않고 계속해서 전파
- 합성곱에 의해 이미지가 작아지는 것이 아니고, 정보를 요약하는 계층을 통해 이미지를 적절한 크기로 줄이는 과정이 필요 이러한 일은 풀링pooling이라는 기법을 통해 이루어짐

# 패딩, 스트라이딩, 다중 채널

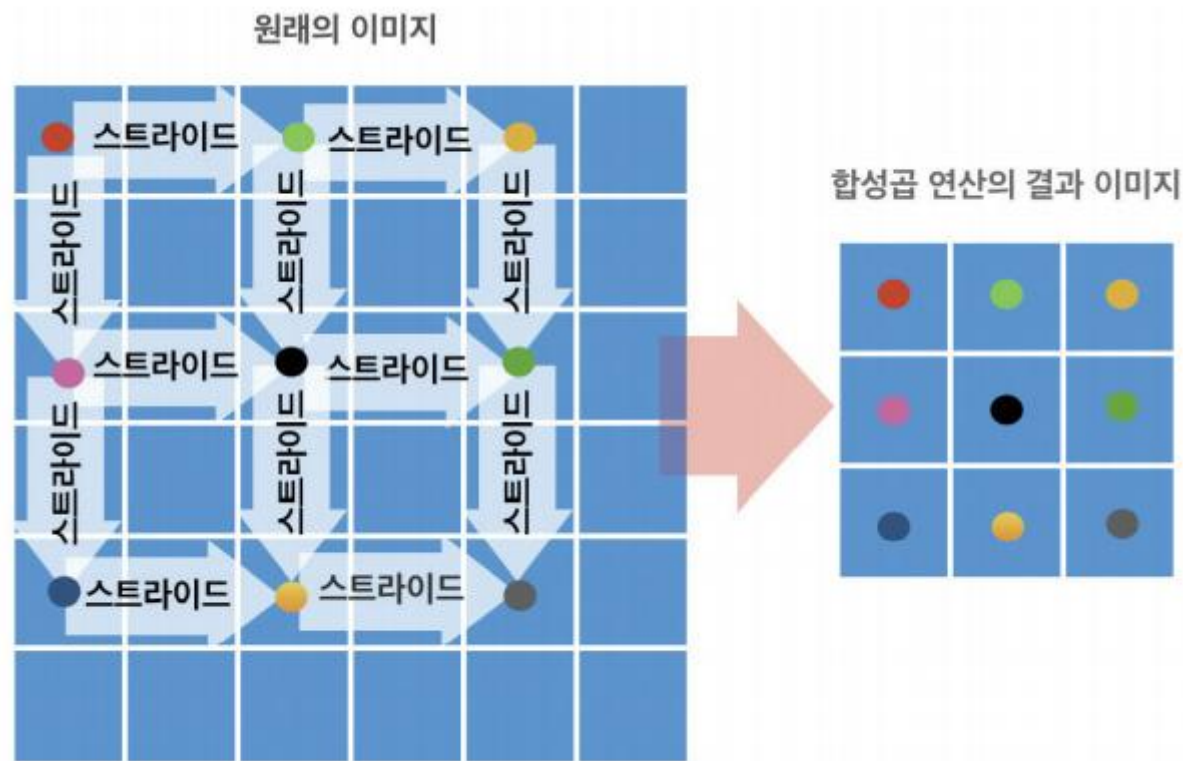
- 합성곱 연산을 사용하면 이미지의 크기가 작아짐
  - 이를 방지하기 위해 입력 이미지의 주변에 값을 덧대어 채워주는 일을 **패딩padding**

- 아래 그림처럼 빨간색으로 표시된 원래 이미지 영역 외부에 추가적인 픽셀(회색으로 표시됨) 값을 채워주면 첫 번째 픽셀부터 필터를 적용할 수 있으므로, 그 결과는 오른쪽과 같이 원래의 이미지와 같은 크기의 이미지

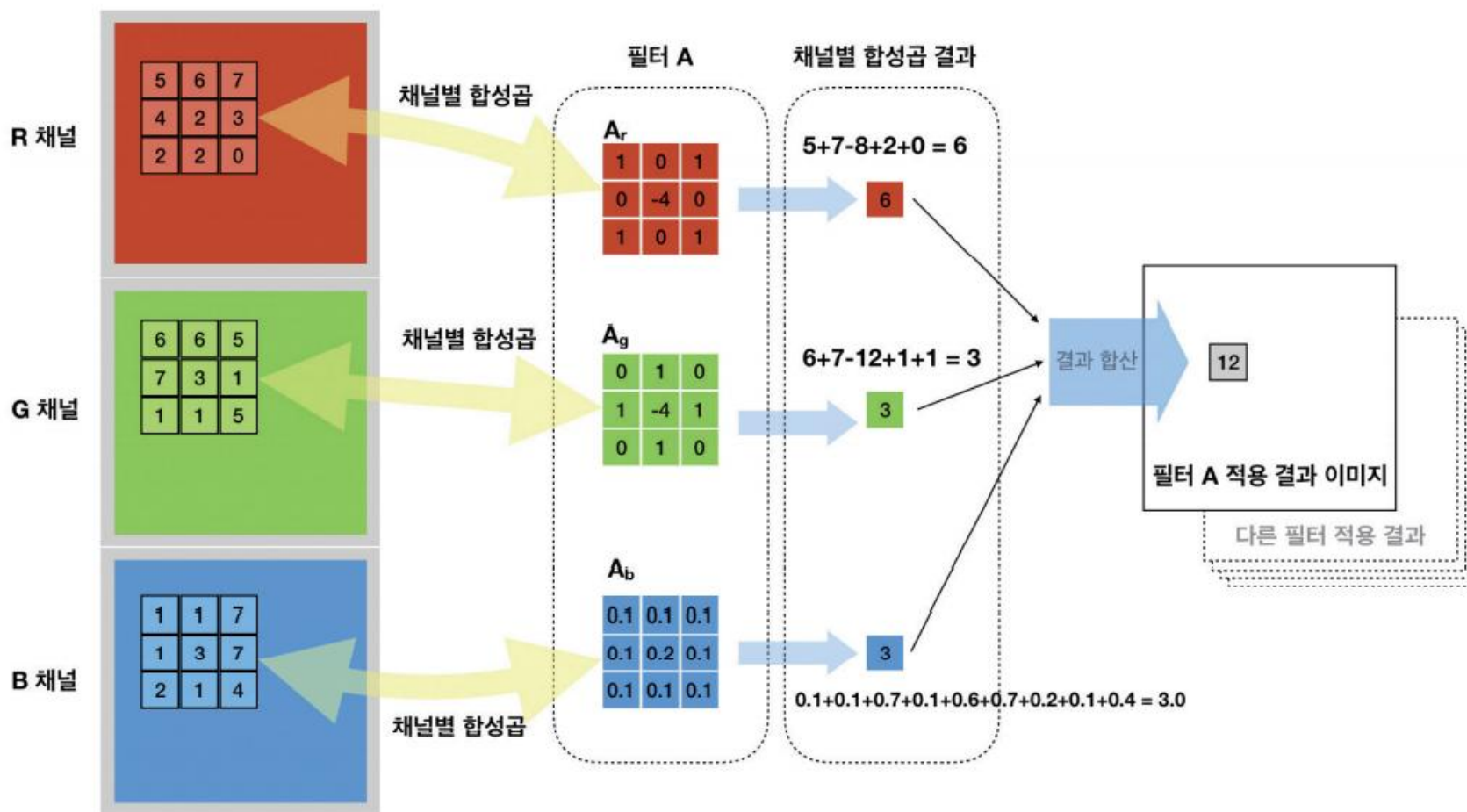


패딩을 하지 않으면 원본 이미지는 계속 작은 이미지로 변할 것이다

- 필터가 적용되는 영역을 윈도우window라고 부르는데, 이 윈도우는 한 픽셀 씩 움직이는 것이 아니라 두 픽셀 혹은 세 픽셀 씩 성큼 성큼 움직일 수도 있으므로 움직이는 보폭을 스트라이드stride라고 함



- 컬러 이미지는 하나의 채널이 아니라 다수의 채널로 구성된 이미지
- 그 외의 다양한 이유로 이미지를 표현할 때 2차원 행렬 하나가 아니라 복수로 제공될 수 있으며 이런 **다중 채널** multi-channel 이미지에 대해 합성곱 신경망 모델이 필터를 적용하는 방식은 그림과 같음

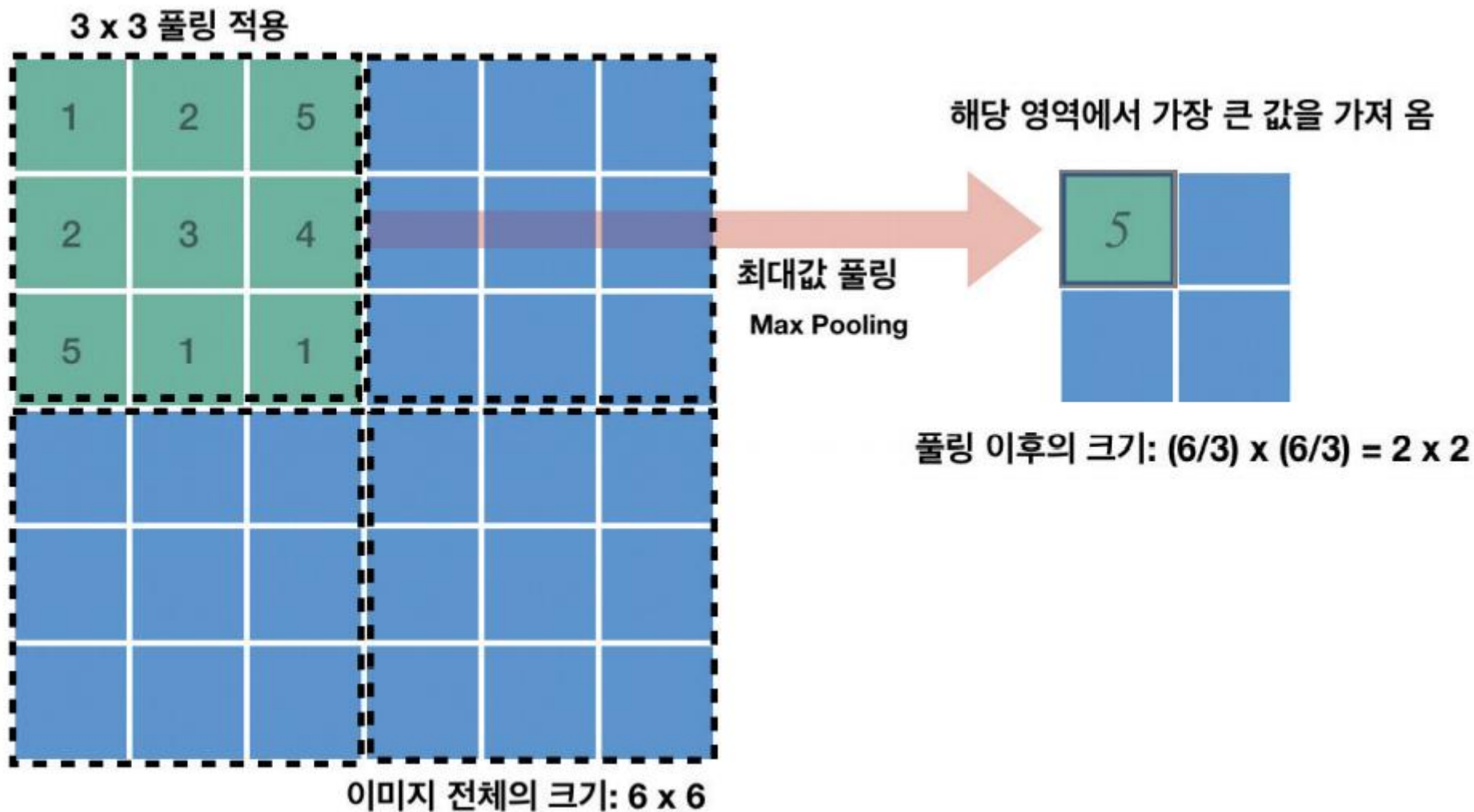


# 강건한 모델을 만드는 풀링

- 풀링(pooling)은 이미지의 일정한 영역 내의 픽셀들이 가진 값을 하나로 축소하는 연산
- 이것은 합성곱과 달리 윈도우를 중첩시켜 이동시키지 않는 것이 일반적
- 원래 이미지의 크기가  $n \times m$ 이고 풀링에 사용하는 영역의 크기가  $n_p \times m_p$  이라면 일반적 스트라이드를 사용한 풀링 이후에 얻게 되는 이미지의 크기는 다음과 같음

$$\frac{n}{n_p} \times \frac{m}{m_p}$$

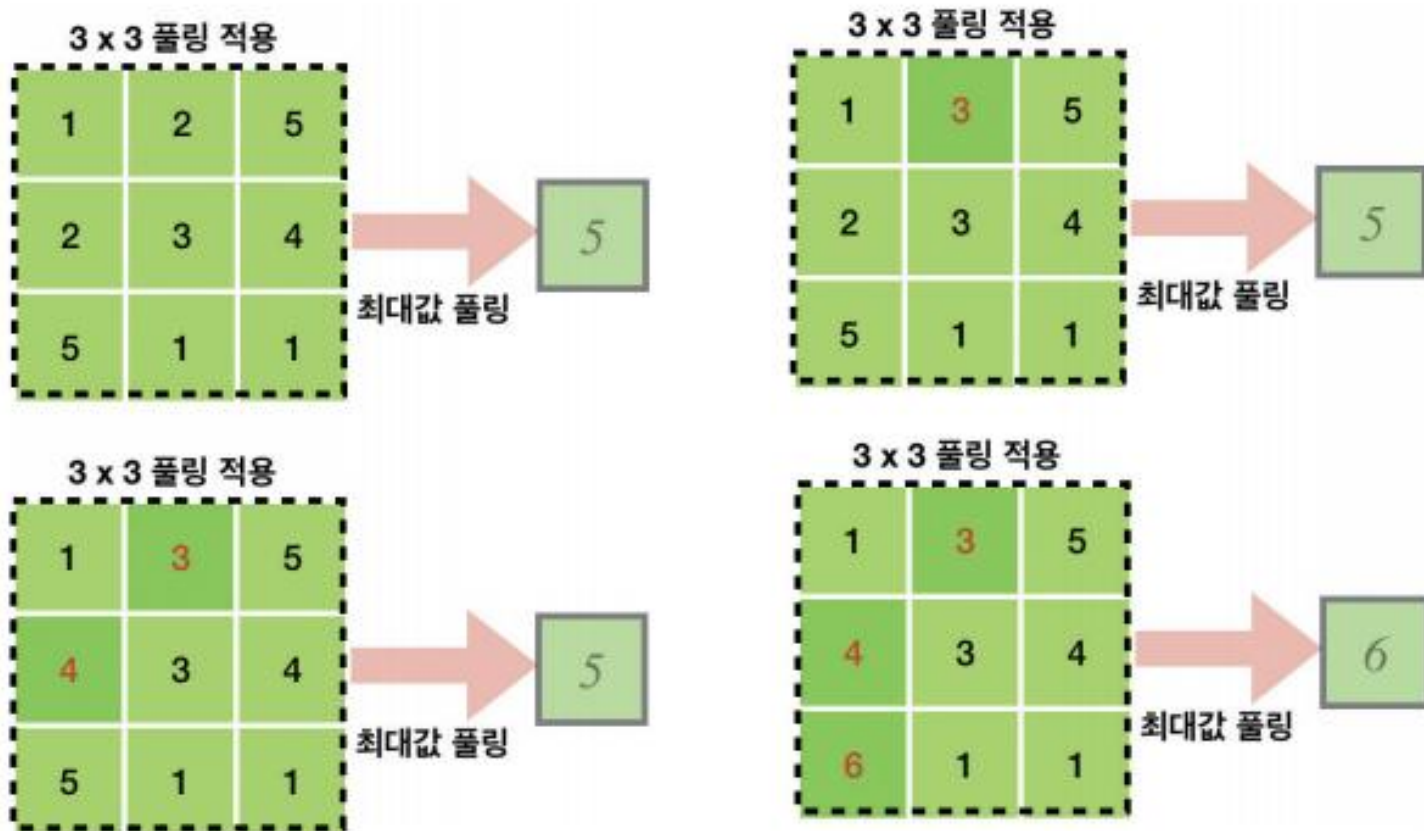
- 풀링은 값을 어떻게 결정하는냐에 따라 **최대값 풀링** max pooling, **평균값 풀링** mean pooling 등이 있음
- 최대값 풀링은 풀링 적용 영역 내에서 가장 큰 값을 결과로 선택하는 것이고, 평균값 풀링은 평균을 구해 결과로 삼는 것





- 풀링을 사용하면 이미지의 크기가 줄어들지만 합성곱 연산을 패딩 없이 적용했을 때 이미지가 줄어드는 것과는 다르다.
  - 합성곱 연산에 의한 축소는 이미지의 주변부 정보를 잃는 일이지만, 풀링은 다수의 픽셀 정보를 통합하여 하나로 만드는 것으로 원래 신호에 존재하는 잡음<sup>noise</sup> 요소를 제거하는 일
- 풀링은 최대값 추출이나 평균값 추출과 같이 미리 정해진 기능을 수행하므로 학습 단계에서 파라미터를 최적화할 필요가 없는 계층
  - 원래 이미지가 가지고 있던 채널을 그대로 유지하면서 공간만 줄이는 일을 수행
- 다수의 픽셀 정보를 통합하여 하나의 픽셀을 생성하기 때문에 정보를 요약할 수 있으며 이 과정에서 입력의 변화에 대해 덜 민감한 신호 전달을 달성

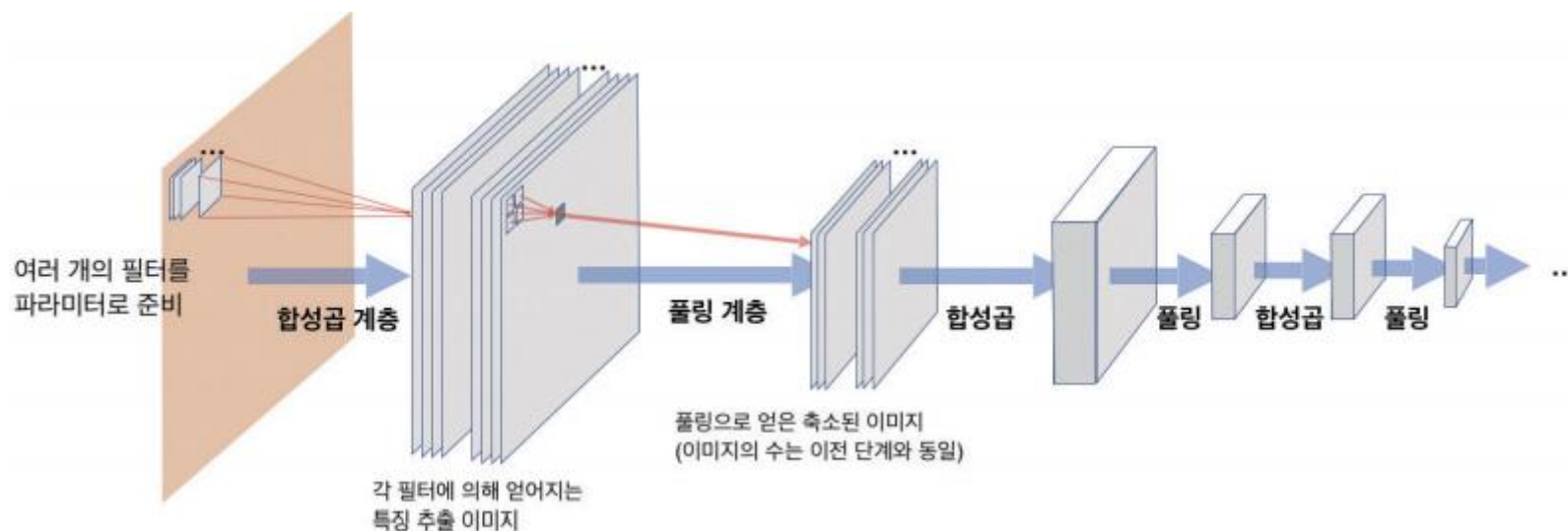
- 픽셀의 값을 일부 변경하여도 최대값 풀링의 결과는 잘 바뀌지 않음
  - 이동 불변성<sup>translation invariance</sup> 이라하며, 이동 불변성을 가진 모델을 **강건한<sup>robust</sup> 모델**이라고 함



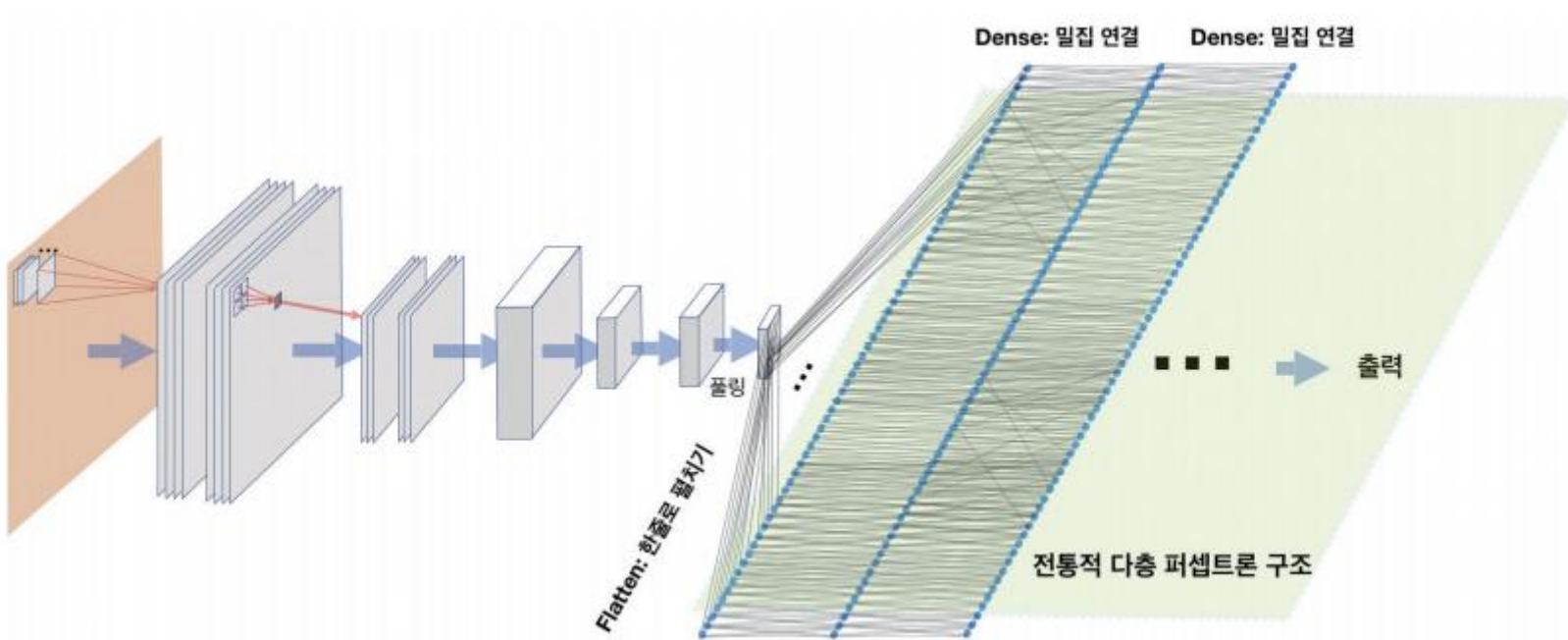
- 인공 신경망 계층에서 합성곱 연산을 수행하는 계층을 **합성곱 계층**convolution layer
- 풀링 연산을 수행하는 계층을 **풀링 계층**pooling layer
- 두 종류의 계층을 통해, 모델 파라미터 개수를 효율적으로 줄여주어 전체 모델 복잡도가 감소하는 효과
- 합성곱 신경망은 이 두 종류의 계층을 쌓아 나가는 형태로 인공 신경망을 구성

# 합성곱 신경망 모델의 구성

- 패딩을 고려한 합성곱과 풀링이 순차적으로 이루어지게 하는 것이 합성곱 신경망의 신호 연결이며 파라미터로 사용되는 것이 합성곱 필터의 가중치
- 몇 차례의 합성곱을 한 뒤에 풀링을 할 수도 있고, 풀링 없이 합성곱으로만 연결 가능
- 풀링만으로 연결될 수는 없는데 풀링은 정보를 줄이기만 할 뿐 학습 가능한 파라미터가 없기 때문



- 최종 이미지를 다층 퍼셉트론<sup>MLP</sup> 형태의 전통적인 신경망에 연결
- 합성곱 신경망의 앞 부분에서 얻은 이미지 정보들을 1차원 벡터로 만드는 **평탄화**<sup>flatten</sup> 과정 필요



- 케라스 API는 밀집 연결이나 평탄화 계층처럼 합성곱과 풀링을 위한 계층도 미리 만들어 제공
- 다음과 같이 Sequential 클래스를 이용하여 모델을 하나 만들고 합성곱과 풀링 계층을 추가

```
model = keras.models.Sequential( [  
    keras.layers.Conv2D(input_shape = (64, 64, 3),  
                        kernel_size = (3,3), filters = 32),  
    keras.layers.MaxPooling2D((2, 2), strides=2),  
    keras.layers.Conv2D(kernel_size = (3,3), padding='same' filters = 64),  
    ... ])
```

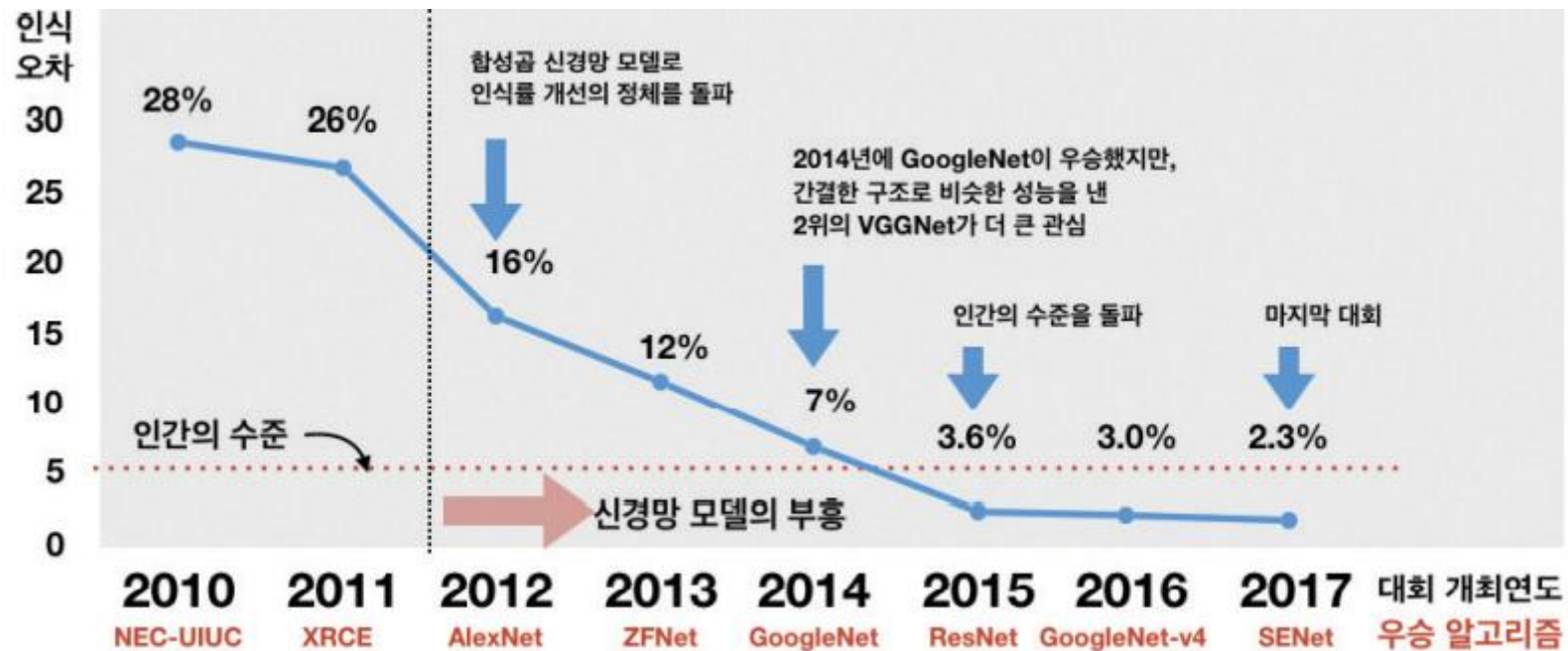


# 합성곱 신경망 모델의 성공

- 고전적인 손글씨 데이터베이스 뿐만 아니라 여러 종류의 사물을 인식하도록 하기 위하여 대규모 이미지 데이터베이스를 구축하는 작업은 스탠퍼드 대학교의 페이페이리<sup>Fei-Fei Li</sup>교수가 주관
- 2만2천개의 범주로 구분된 1,500만장의 방대한 이미지로 구성된 **이미지넷**<sup>ImageNet</sup> 데이터베이스는 오늘날의 딥러닝 기술을 발전시키는데 크게 기여

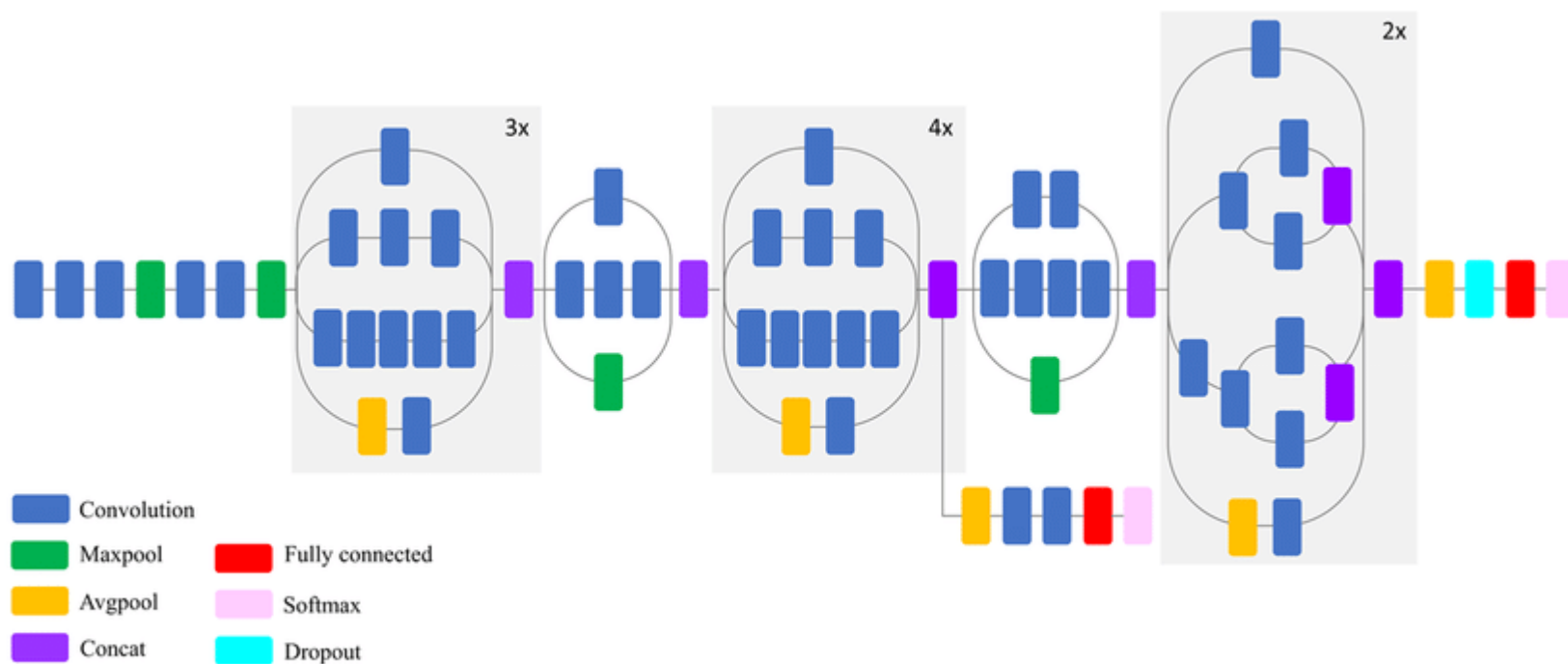


- 합성곱 신경망 모델이 각광을 받게 된 계기는 2012년 이미지넷의 경진대회에서 합성곱 신경망 구조로 AlexNet 이 압도적 성적으로 우승한 일
- 이 경진대회는 흔히 ILSVRC라고 부르며, 이는 **이미지넷 대규모 시각 인지 경진대회** ImageNet large scale visual recognition challenge의 머릿글자





- GoogleNet의 코드명은 **인셉션Inception**
  - 이 모델은 합성곱 신경망을 매우 깊게 구성하는데, 연산을 줄일 수 있도록하는 기능인 인셉션 모듈을 도입
  - 2015년부터는 인간의 능력을 뛰어넘는 인식률을 보이면서 2017년을 마지막으로 이 경진대회는 더 열리지 않고 있음



인셉션 아키텍처

- 합성곱 신경망이 이전의 신경망을 크게 개선할 수 있었던 것은 다음과 같은 핵심적인 개념들 때문
  - 희소 상호작용sparse interaction 합성곱 모델은 국소적인 영역에 작은 필터를 적용하여 특징을 추출한다. 입력된 이미지의 모든 픽셀들 사이의 관계를 다룰 필요가 없어 입력 신호들 사이의 연결 개수를 크게 줄일 수 있음
  - 파라미터 공유parameter sharing 여러 영역에 적용되는 합성곱 필터는 많은 수의 데이터를 처리하여도 작은 크기로 유지될 수 있고, 파라미터의 수도 제한적이다. 신경망 연산의 핵심이 연결 강도를 곱하는 것이라는 점에서 파라미터 공유는 연산량 감소에 큰 역할을 하고, 더 깊은 층을 쌓을 수 있도록 함
  - 등변성 표현equivariant representation 등변성은 어떤 함수의 입력에 특정한 변경을 적용하면 출력도 같은 방식으로 변하는 것을 의미한다. 즉  $f(g(x)) = g(f(x))$ 의 성질을 갖는 함수  $f$ 를 등변 함수equivariant function라고 하며 합성곱 신경망은 이미지의 일부가 이동하면, 그 결과가 같은 방식으로 이동하여 나타나게 된다. 이것은 외곽선 추출과 같은 국소적인 함수가 어디서나 효과를 발휘한다는 것을 의미
  - 유의할 점은 이미지의 크기 변화나 회전 등에 대해서는 합성곱 연산이 등변성을 보이지 못한다는 것이다. 제프리 힌튼Geoffrey Hinton 교수는 이러한 문제를 해결하기 위해 캡슐capsule 신경망 개념을 제안

# 합성곱 신경망으로 패션 MNIST 분류를 개선하기

## 실습 목표

CNN을 이용하여 패션 MNIST 데이터를 분류하는 작업을 시도하자. 이 방법과 다층 퍼셉트론을 사용한 방법을 비교하고, 얼마나 성능이 개선되는지 확인해 보자.



## 힌트

다층 퍼셉트론을 이용하여 MNIST 데이터의 분류를 수행했던 LAB<sup>8-1</sup>의 해답 코드와 큰 차이는 없다. 모델만 합성곱 신경망으로 바꾸면 된다. 이 과정에서 합성곱 신경망이 요구하는 입력은 이미지의 채널 정보가 추가적으로 요구되기 때문에 이러한 형태로 변형하는 작업이 필요하다.

교재의 해답 코드를 하나하나 단계별로 수행해 봅니다

## 나의 CNN 모델 만들기

[ 합성곱 신경망 모델은 다중 채널 이미지를 기본으로 사용. 채널이 1인 우리 데이터는 채널 1이라는 값을 명시적으로 갖도록 변형

```
[ ] train_images = train_images[:, :, :, np.newaxis]
    test_images = test_images[:, :, :, np.newaxis]
    train_images.shape, test_images.shape
```

```
↔ ((60000, 28, 28, 1), (10000, 28, 28, 1))
```

```
[ ] ##### my CNN model
```

```
[
    model = keras.models.Sequential(
        [
            keras.layers.Conv2D(input_shape=(28, 28, 1), kernel_size=(3,3), padding='same', filters=64),
            keras.layers.MaxPooling2D( (2,2), strides = 2),
            keras.layers.Conv2D(kernel_size=(3,3), padding='same', filters=32),
            keras.layers.MaxPooling2D( (2,2), strides = 2),
            keras.layers.Conv2D(kernel_size=(3,3), padding='same', filters=16),
            keras.layers.Flatten(),
            keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(10, activation='softmax')
        ]
    )
```

## 나의 CNN 모델 만들기

합성곱 신경망 모델은 다중 채널 이미지를 기본으로 사용. 채널이 1인 우리 데이터는 채널 1이라는 값을 명시적으로 갖도록 변형

```
[ ] train_images = train_images[:, :, :, np.newaxis]
    test_images = test_images[:, :, :, np.newaxis]
    train_images.shape, test_images.shape
```

```
↩ ((60000, 28, 28, 1), (10000, 28, 28, 1))
```

```
[ ] ##### my CNN model

model = keras.models.Sequential(
    [
        keras.layers.Conv2D(input_shape=(28, 28, 1), kernel_size=(3,3), padding='same', filters=64),
        keras.layers.MaxPooling2D( (2,2), strides = 2),
        keras.layers.Conv2D(kernel_size=(3,3), padding='same', filters=32),
        keras.layers.MaxPooling2D( (2,2), strides = 2),
        keras.layers.Conv2D(kernel_size=(3,3), padding='same', filters=16),
        keras.layers.Flatten(),
        keras.layers.Dense(256, activation='relu'),
        keras.layers.Dense(128, activation='relu'),
        keras.layers.Dense(10, activation='softmax')
    ]
)
```

# 나의 CNN 모델 만들기

```
▶ model.summary()
```

↔ Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 28, 28, 64)	640
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_7 (Conv2D)	(None, 14, 14, 32)	18464
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_8 (Conv2D)	(None, 7, 7, 16)	4624
flatten_2 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 256)	200960
dense_7 (Dense)	(None, 128)	32896
dense_8 (Dense)	(None, 10)	1290

=====  
Total params: 258874 (1011.23 KB)  
Trainable params: 258874 (1011.23 KB)  
Non-trainable params: 0 (0.00 Byte)  
=====

## 나의 CNN 모델 만들기

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

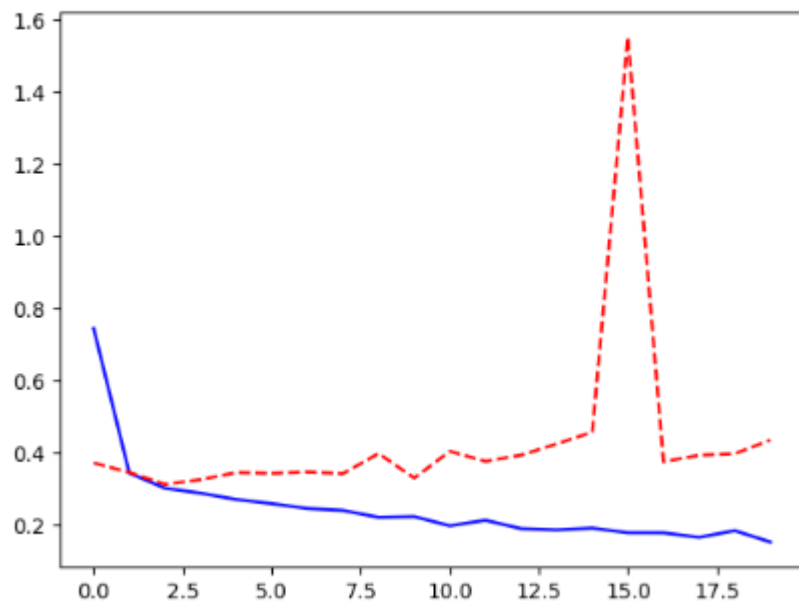
```
history = model.fit(train_images, train_labels, epochs=20, validation_split=0.25)
```

```
Epoch 1/20  
1407/1407 [=====] - 11s 6ms/step - loss: 0.7440 - accuracy: 0.8158 - val_loss: 0.3715 - val_accuracy: 0.8677  
Epoch 2/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.3434 - accuracy: 0.8767 - val_loss: 0.3450 - val_accuracy: 0.8757  
Epoch 3/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.3014 - accuracy: 0.8902 - val_loss: 0.3122 - val_accuracy: 0.8902  
Epoch 4/20  
1407/1407 [=====] - 8s 6ms/step - loss: 0.2876 - accuracy: 0.8954 - val_loss: 0.3250 - val_accuracy: 0.8879  
Epoch 5/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.2703 - accuracy: 0.9010 - val_loss: 0.3448 - val_accuracy: 0.8794  
Epoch 6/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.2586 - accuracy: 0.9080 - val_loss: 0.3425 - val_accuracy: 0.8902  
Epoch 7/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.2452 - accuracy: 0.9117 - val_loss: 0.3465 - val_accuracy: 0.8901  
Epoch 8/20  
1407/1407 [=====] - 8s 6ms/step - loss: 0.2398 - accuracy: 0.9154 - val_loss: 0.3418 - val_accuracy: 0.8931  
Epoch 9/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.2207 - accuracy: 0.9208 - val_loss: 0.3977 - val_accuracy: 0.8735  
Epoch 10/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.2227 - accuracy: 0.9230 - val_loss: 0.3294 - val_accuracy: 0.8965  
Epoch 11/20  
1407/1407 [=====] - 9s 6ms/step - loss: 0.1972 - accuracy: 0.9309 - val_loss: 0.4038 - val_accuracy: 0.8779  
Epoch 12/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.2126 - accuracy: 0.9284 - val_loss: 0.3759 - val_accuracy: 0.8941  
Epoch 13/20  
1407/1407 [=====] - 8s 5ms/step - loss: 0.1893 - accuracy: 0.9348 - val_loss: 0.3930 - val_accuracy: 0.8961  
Epoch 14/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.1858 - accuracy: 0.9372 - val_loss: 0.4242 - val_accuracy: 0.8961  
Epoch 15/20  
1407/1407 [=====] - 8s 6ms/step - loss: 0.1907 - accuracy: 0.9367 - val_loss: 0.4569 - val_accuracy: 0.8785  
Epoch 16/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.1779 - accuracy: 0.9403 - val_loss: 1.5522 - val_accuracy: 0.8717  
Epoch 17/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.1777 - accuracy: 0.9418 - val_loss: 0.3749 - val_accuracy: 0.9041  
Epoch 18/20  
1407/1407 [=====] - 8s 6ms/step - loss: 0.1651 - accuracy: 0.9465 - val_loss: 0.3927 - val_accuracy: 0.8997  
Epoch 19/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.1838 - accuracy: 0.9406 - val_loss: 0.3970 - val_accuracy: 0.9047  
Epoch 20/20  
1407/1407 [=====] - 7s 5ms/step - loss: 0.1522 - accuracy: 0.9510 - val_loss: 0.4348 - val_accuracy: 0.9048
```

## 나의 CNN 모델 만들기

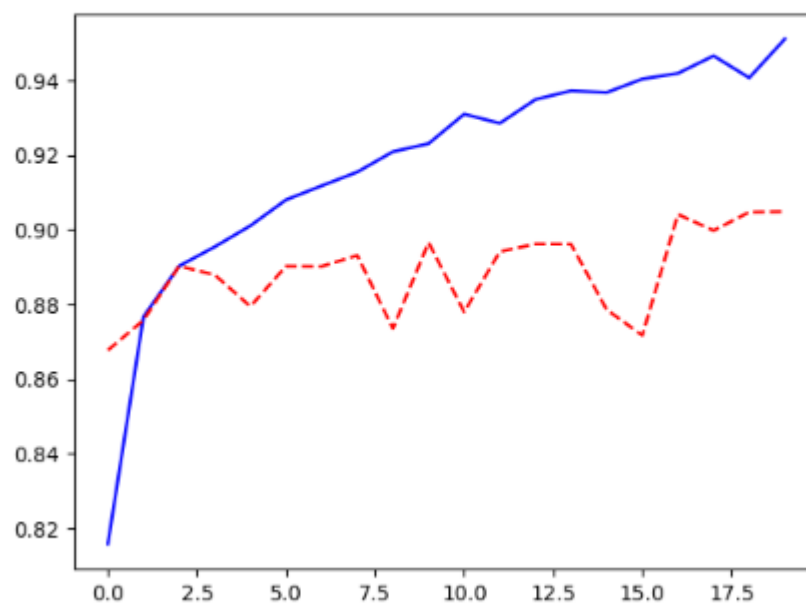
```
plt.plot(history.history['loss'], 'b-')  
plt.plot(history.history['val_loss'], 'r--')
```

[<matplotlib.lines.Line2D at 0x7b1a6df79f90>]



```
plt.plot(history.history['accuracy'], 'b-')  
plt.plot(history.history['val_accuracy'], 'r--')
```

[<matplotlib.lines.Line2D at 0x7b1a6d66b100>]





## 나의 CNN 모델 만들기

```
[ ] class_name = [ 'T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Boots' ]
```

```
def plot_images( images, labels, images_per_row = 5 ) :  
    n_images = len(images)  
    n_rows = (n_images - 1) // images_per_row + 1  
    fig, ax = plt.subplots( n_rows, images_per_row,  
                           figsize = (images_per_row * 2, n_rows * 2))  
  
    for i in range(n_rows) :  
        for j in range(images_per_row) :  
            if i*images_per_row + j >= n_images : break;  
            im_idx = i * images_per_row + j  
            if n_rows > 1:  
                axis = ax[i, j]  
            else :  
                axis = ax[j]  
            axis.imshow(images[im_idx], cmap='gray', interpolation = 'nearest')  
            axis.axis('off')  
            axis.set_title(class_name[labels[im_idx]])
```

## 나의 CNN 모델 만들기

```
rand_idx = np.random.randint(0, 10000)

images = test_images[rand_idx: rand_idx + 25]
plot_images(images, test_labels[rand_idx:rand_idx+25], images_per_row = 5)
```

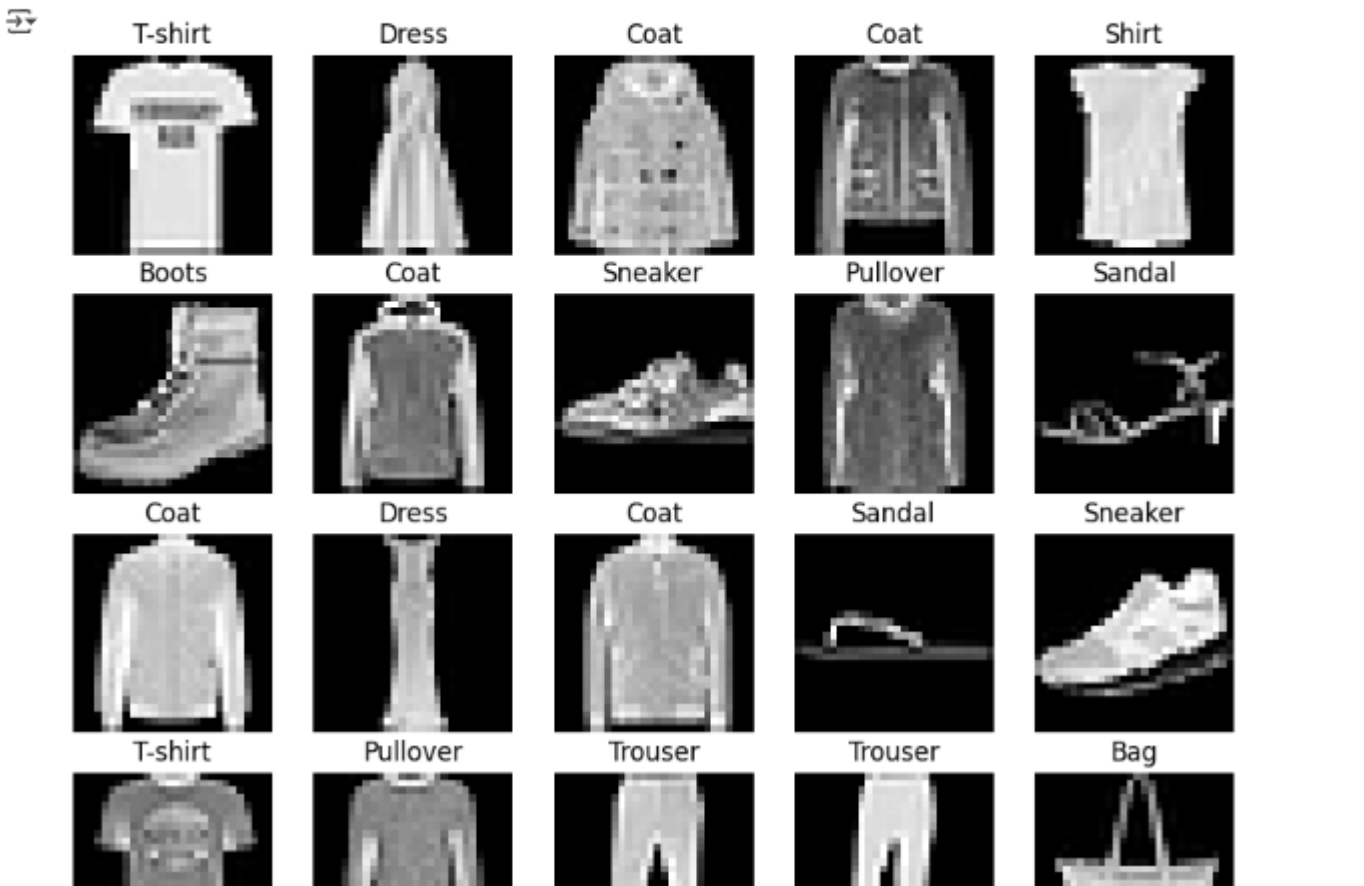


# 나의 CNN 모델 만들기

```
▶ pred = np.argmax(model.predict(images), axis=1)
pred
```

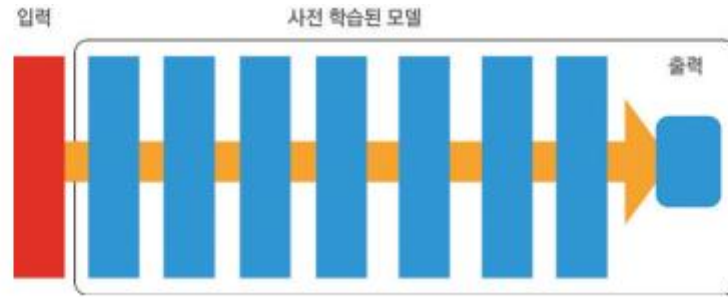
```
↗ 1/1 [=====] - 0s 19ms/step
array([0, 3, 4, 4, 6, 9, 4, 7, 2, 5, 4, 3, 4, 5, 7, 0, 2, 1, 1, 8, 7, 3,
       2, 0, 5])
```

```
[ ] plot_images(images, pred)
```



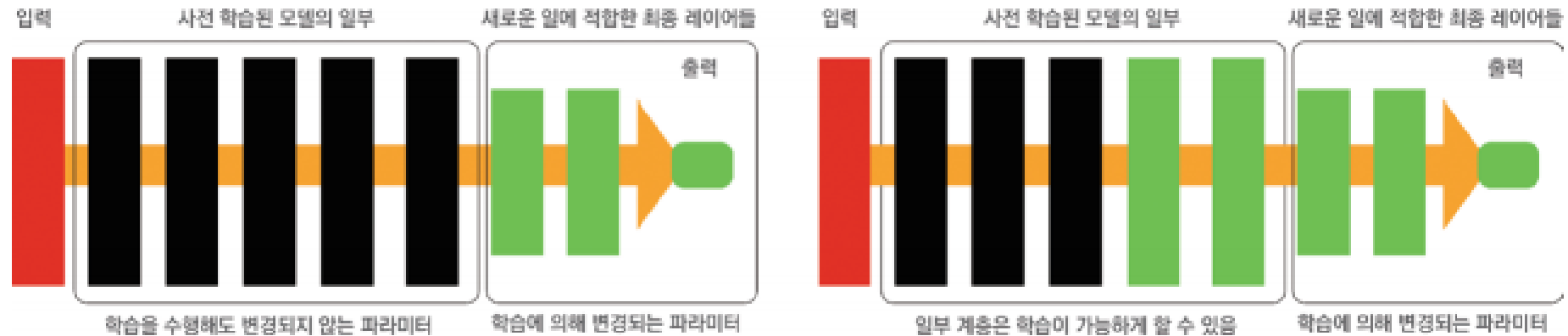
# 전이학습 - 이미 훈련된 모델을 고쳐 쓰기

- 전이학습 *transfer learning*은 이미 학습이 완료된 모델을 다른 목적에 맞춰 조금만 고쳐서 사용하는 것



- 이미 학습된 모델을 가져다가 특징을 추출하는 용도로 사용하고, 이 특징 추출기의 출력을 이용하여 우리가 원하는 작업을 수행하는 다음과 같은 구조로 바꿀 수 있으면 좋을 것

- 사전 학습된 모델을 가져와 일부분을 우리의 새로운 일에 활용하려면 일부 계층을 잘라서 가져오고, 이 계층들이 가지고 있는 파라미터들은 변경되지 않도록 만들어야 함
  - 그림에 검정색으로 표시된 계층들이 변경되지 않는 층
- 이 계층들 이후에는 우리의 새로운 작업에 맞는 출력을 내도록 새롭게 구성해서 붙이는 것
- 붙인 부분이 학습을 통해 파라미터가 변경되는 영역으로 만들면 되며 물론 그림 오른쪽과 같이 사전 학습된 모델의 일부를 학습 가능하게 만들 수 있음



# 성능 좋은 CNN을 가져다 써보자

## 실습 목표

텐서플로우는 이미 많은 학자들과 개발자들이 개발하여 검증한 훌륭한 CNN 모델을 제공하고 있다. 우리는 구글사에서 만든 Inception V3라는 뛰어난 모델을 사용하여 MNIST 데이터의 분류를 할 것이다. 결과는 99%의 매우 높은 정확도를 보일 것이다.



## 힌트

`keras.applications.inception_v3`라는 서브 모듈을 이용하자. keras에는 ImageNet 경진대회에서 성능이 입증된 Xception, VGG16, VGG19, ResNet, ResNetV2, ResNeXt, InceptionV3, InceptionResNetV2, MobileNet, MobileNetV2, DenseNet, NASNet을 제공한다. 각각의 모델에 대한 자세한 특징은 keras 어플리케이션 웹 문서<sup>7)</sup>를 참고하면 된다.

교재의 해답 코드를 하나하나 단계별로 수행해 봅니다

- 우선 Inception V3 모델을 가져오면서 **이미지넷** imagenet 데이터를 이용하여 학습한 파라미터를 가져 오도록 함



```
# 사전 훈련된 모델을 가져온다. - 이미지넷 데이터로 학습된 모델
pre_trained_model = InceptionV3(input_shape=(75, 75, 3),
                                include_top=False,
                                weights='imagenet') # 반드시 훈련된 값을 사용
```

- 사전 학습된 모델의 모든 계층에 대해 파라미터를 고정
  - trainable 옵션을 False로 지정하면 된다.



```
for layer in pre_trained_model.layers:
    layer.trainable = False
```

- 사전 훈련된 모델의 능력을 활용하므로 뒤쪽 두 계층은 64개 노드와 10개 노드로 매우 간단하게 만듦
  - 모델 요약을 보면 훈련을 위한 파라미터가 40만개 정도로 대폭 줄었음



```
last_layer = pre_trained_model.get_layer('mixed7')
last_output = last_layer.output

x = layers.Flatten()(last_output)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dense(10, activation='softmax')(x)

model = Model(pre_trained_model.input, x)
model.compile(optimizer=RMSprop(lr=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

...

=====

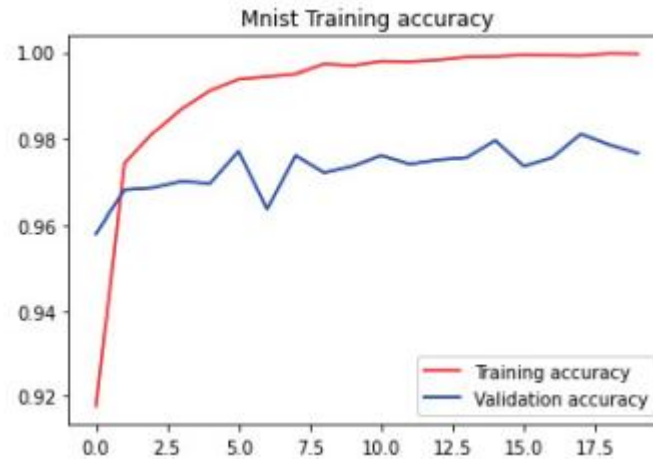
Total params: 9,418,346

Trainable params: 443,082

Non-trainable params: 8,975,264



- 학습되는 파라미터의 수가 대폭 줄었기 때문에 학습의 속도도 대폭 향상



- 특히 훈련 초기부터 검증 데이터에 대한 정확도가 높은 수준으로 나타남을 유의
- 전이학습을 이용하면 데이터 인스턴스의 수가 적어서 훈련이 어려운 경우에도 효과적으로 특징을 추출하여 좋은 결과를 낼 수 있음

## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

### 목표

SVM을 이용했던 미니 프로젝트 B1의 결과를 개선하기 위해 미리 만들어 놓은 CNN을 적용해 보자. 텐서플로우는 이미검증된 훌륭한 CNN 모델을 함께 제공하고 있다. 우리는 구글에서 만든 Inception V3라는 뛰어난 CNN을 모델과 옥스퍼드 대학의 연구자들이 만든 VGG16을 사용할 것이다.

## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
[ ] import matplotlib.pyplot as plt
    import numpy as np
```

```
[ ] from skimage.io import imread
    from skimage.transform import resize
```

```
[ ] from keras.applications.inception_v3 import preprocess_input
    from keras.applications.inception_v3 import InceptionV3
    from keras import models, layers
    from keras.models import Model
```

```
[ ] url = 'https://github.com/dknife/ML/raw/main/data/Proj2/faces/'
```

```
[ ] face_images = []
    for i in range(15):
        file = url + 'img{0:02d}.jpg'.format(i+1)
        img = imread(file)
        img = resize(img, (75,75))
        face_images.append(img)

    np.array(face_images).shape
```



(15, 75, 75, 3)

## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
[ ] def plot_images(nRow, nCol, img) :  
    fig = plt.figure()  
    fig, ax = plt.subplots(nRow, nCol, figsize=(nCol, nRow))  
    for i in range(nRow):  
        for j in range(nCol):  
            if nRow <= 1: axis = ax[j]  
            else: axis = ax[i, j]  
            axis.imshow(img[i*nCol + j])  
            axis.set_xticks([])  
            axis.set_yticks([])
```

```
[ ] plot_images(3, 5, face_images)
```

🔗 <Figure size 640x480 with 0 Axes>



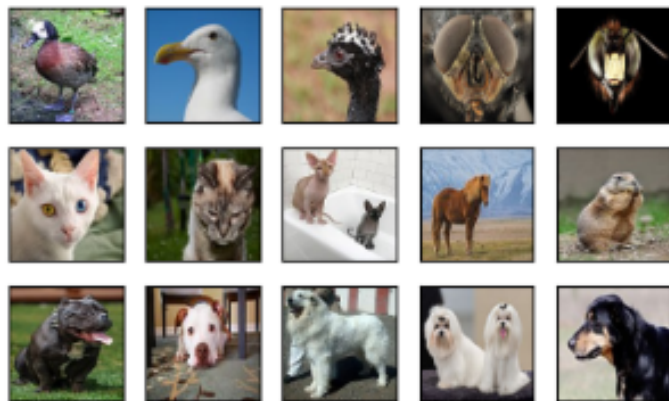
## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
▶ url = 'https://github.com/dknife/ML/raw/main/data/Proj2/animals/'  
animal_images = []  
for i in range(15):  
    file = url + 'img{0:02d}.jpg'.format(i+1)  
    img = imread(file)  
    img = resize(img, (75,75))  
    animal_images.append(img)  
  
np.array(animal_images).shape
```

↗ (15, 75, 75, 3)

```
[ ] plot_images(3, 5, animal_images)
```

↗ <Figure size 640x480 with 0 Axes>



## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
X = face_images + animal_images
y = [[1]] * len(face_images) + [[0]] * len(animal_images)
X = np.array(X)
y = np.array(y)
print(y)
```

[illegible]

## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
[ ] X.shape, y.shape
```

```
((30, 75, 75, 3), (30, 1))
```

▶ # 전이학습: inception

```
pre_trained_model = InceptionV3(input_shape = (75, 75, 3),  
                                include_top=False,  
                                weights = 'imagenet')
```

```
pre_trained_model.trainable = False  
pre_trained_model.summary()
```

Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/inception\\_v3/inception\\_v3\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_notop.h5](https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5)  
87910968/87910968 [=====] - 4s 0us/step  
Model: "inception\_v3"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 75, 75, 3)]	0	[]
conv2d (Conv2D)	(None, 37, 37, 32)	864	['input_1[0][0]']
batch_normalization (Batch Normalization)	(None, 37, 37, 32)	96	['conv2d[0][0]']
activation (Activation)	(None, 37, 37, 32)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 35, 35, 32)	9216	['activation[0][0]']
batch_normalization_1 (Batch Normalization)	(None, 35, 35, 32)	96	['conv2d_1[0][0]']
activation_1 (Activation)	(None, 35, 35, 32)	0	['batch_normalization_1[0][0]']
conv2d_2 (Conv2D)	(None, 35, 35, 64)	18432	['activation_1[0][0]']
batch_normalization_2 (Batch Normalization)	(None, 35, 35, 64)	96	['conv2d_2[0][0]']

## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
[ ] last_layer = pre_trained_model.get_layer('mixed7')
last_output = last_layer.output

my_layer = layers.Flatten()(last_output)
my_layer = layers.Dense(512, activation='relu')(my_layer)
my_layer = layers.Dense(128, activation='relu')(my_layer)
my_layer = layers.Dense(256, activation='relu')(my_layer)
my_layer = layers.Dense(1, activation='sigmoid')(my_layer)
```

```
▶ model = Model(pre_trained_model.input, my_layer)
model.compile(
    optimizer = 'adam',
    loss = 'mse',
    metrics = ['accuracy']
)
```

```
📄 model.summary()
```

↔ Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 75, 75, 3)]	0	[]
conv2d (Conv2D)	(None, 37, 37, 32)	864	['input_1[0][0]']
batch_normalization (Batch Normalization)	(None, 37, 37, 32)	96	['conv2d[0][0]']
activation (Activation)	(None, 37, 37, 32)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 35, 35, 32)	9216	['activation[0][0]']
batch_normalization_1 (Batch Normalization)	(None, 35, 35, 32)	96	['conv2d_1[0][0]']
activation_1 (Activation)	(None, 35, 35, 32)	0	['batch_normalization_1[0][0]']
conv2d_2 (Conv2D)	(None, 35, 35, 64)	18432	['activation_1[0][0]']
batch_normalization_2 (Batch Normalization)	(None, 35, 35, 64)	192	['conv2d_2[0][0]']
activation_2 (Activation)	(None, 35, 35, 64)	0	['batch_normalization_2[0][0]']
max_pooling2d (MaxPooling2D)	(None, 17, 17, 64)	0	['activation_2[0][0]']
conv2d_3 (Conv2D)	(None, 17, 17, 80)	5120	['max_pooling2d[0][0]']

[+ Code](#)[+ Text](#)



## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
history = model.fit(X, y, epochs=150, shuffle=True)
```

1/1 ————— 0s 148ms/step - accuracy: 1.0000 - loss: 8.976e-13  
Epoch 49/150  
1/1 ————— 0s 131ms/step - accuracy: 1.0000 - loss: 8.7157e-13  
Epoch 50/150  
1/1 ————— 0s 62ms/step - accuracy: 1.0000 - loss: 8.4855e-13  
Epoch 51/150  
1/1 ————— 0s 61ms/step - accuracy: 1.0000 - loss: 8.2819e-13  
Epoch 52/150  
1/1 ————— 0s 63ms/step - accuracy: 1.0000 - loss: 8.1017e-13  
Epoch 53/150  
1/1 ————— 0s 139ms/step - accuracy: 1.0000 - loss: 7.9416e-13  
Epoch 54/150  
1/1 ————— 0s 60ms/step - accuracy: 1.0000 - loss: 7.7991e-13  
Epoch 55/150  
1/1 ————— 0s 61ms/step - accuracy: 1.0000 - loss: 7.6721e-13  
Epoch 56/150  
1/1 ————— 0s 141ms/step - accuracy: 1.0000 - loss: 7.5587e-13  
Epoch 57/150  
1/1 ————— 0s 62ms/step - accuracy: 1.0000 - loss: 7.4573e-13  
Epoch 58/150  
1/1 ————— 0s 62ms/step - accuracy: 1.0000 - loss: 7.3665e-13  
Epoch 59/150  
1/1 ————— 0s 151ms/step - accuracy: 1.0000 - loss: 7.2851e-13  
Epoch 60/150  
1/1 ————— 0s 66ms/step - accuracy: 1.0000 - loss: 7.2122e-13  
Epoch 61/150  
1/1 ————— 0s 60ms/step - accuracy: 1.0000 - loss: 7.1470e-13  
Epoch 62/150  
1/1 ————— 0s 60ms/step - accuracy: 1.0000 - loss: 7.0884e-13  
Epoch 63/150  
1/1 ————— 0s 60ms/step - accuracy: 1.0000 - loss: 7.0357e-13  
Epoch 64/150  
1/1 ————— 0s 60ms/step - accuracy: 1.0000 - loss: 6.9883e-13  
Epoch 65/150  
1/1 ————— 0s 140ms/step - accuracy: 1.0000 - loss: 6.9455e-13  
Epoch 66/150  
1/1 ————— 0s 61ms/step - accuracy: 1.0000 - loss: 6.9070e-13  
Epoch 67/150  
1/1 ————— 0s 141ms/step - accuracy: 1.0000 - loss: 6.8722e-13  
Epoch 68/150  
1/1 ————— 0s 140ms/step - accuracy: 1.0000 - loss: 6.8409e-13  
Epoch 69/150  
1/1 ————— 0s 64ms/step - accuracy: 1.0000 - loss: 6.8126e-13  
Epoch 70/150  
1/1 ————— 0s 146ms/step - accuracy: 1.0000 - loss: 6.7870e-13  
Epoch 71/150  
1/1 ————— 0s 60ms/step - accuracy: 1.0000 - loss: 6.7640e-13  
Epoch 72/150  
1/1 ————— 0s 141ms/step - accuracy: 1.0000 - loss: 6.7431e-13  
Epoch 73/150

## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
url = 'https://github.com/dknife/ML/raw/main/data/Proj2/test_data/'
test_images = []
for i in range(10):
    file = url + 'img{0:02d}.jpg'.format(i+1)
    img = imread(file)
    img = resize(img, (75,75))
    test_images.append(img)
test_images = np.array(test_images)

plot_images(2, 5, test_images)
```

<Figure size 640x480 with 0 Axes>

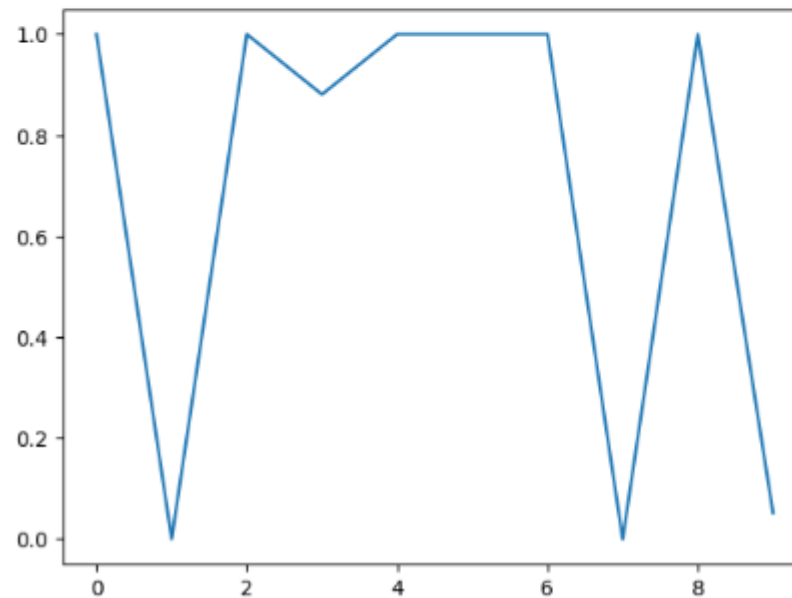


```
print(test_images.shape)
test_result = model.predict(test_images)
print(test_result)
plt.plot(test_result)
```

(10, 75, 75, 3)  
1/1 6s 6s/step

```
[[9.9995911e-01]
 [1.5610392e-04]
 [1.0000000e+00]
 [8.8080585e-01]
 [1.0000000e+00]
 [9.9999988e-01]
 [1.0000000e+00]
 [5.5548958e-06]
 [1.0000000e+00]
 [5.2164912e-02]]
```

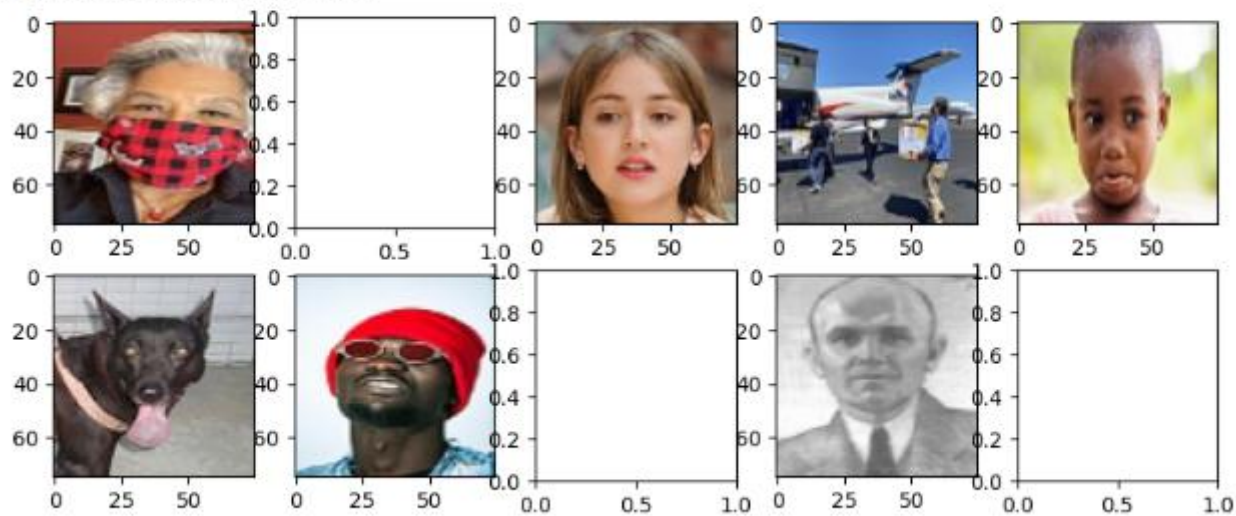
[<matplotlib.lines.Line2D at 0x782a7932ea90>]



## 미니 프로젝트 얼굴 찾기: 전이학습 활용하기

```
fig = plt.figure()
fig, ax = plt.subplots(2, 5, figsize=(10, 4))
for i in range(2):
    for j in range(5):
        if test_result[i*5+j] > 0.5:
            ax[i, j].imshow(test_images[i*5+j])
```

<Figure size 640x480 with 0 Axes>



# 핵심 정리

- 인간과 동물의 감각공간의 일부인 수용장은 시각 시스템으로부터 유입되는 자극에 반응하여 특별한 역치 이상의 자극에 대해서 활성화된다.
- 동물의 시각피질의 구조는 시각 세포로부터 자극이 전달될 때 여러 시각피질 층을 통과하여 계층적으로 전달되는 특징을 가진다. 이러한 구조에 영감을 받아 만들어진 심층 신경망 모델이 합성곱 신경망 모델이다.
- 생물체의 수용장 역할을 하는 것이 인공 신경망에서는 합성곱 혹은 컨볼루션 연산이다.
- 이미지가 입력 데이터로 주어지면 이 데이터에 대하여 커널 혹은 필터라는 작은 행렬을 이용하여 합성곱 연산을 수행할 수 있는데, 이 합성곱 연산을 통해서 특징을 추출해 낼 수 있다.
- 필터는 2차원 행렬을 사용하기 때문에 평탄화된 1차원 심층 신경망에서 처리할 수 없는 신호들의 기하적인 특징을 신경망 학습에 사용할 수 있다.
- 필터는 전통적인 이미지 처리 시스템에서 널리 이용되어 오던 개념이다.
- 좋은 필터가 있을 경우 이미지의 특징을 잘 부각시킬 수 있는데, 학습을 통해서 좋은 필터를 생성하도록 만들 수 있다. 이 아이디어가 합성곱 신경망의 핵심 아이디어이다.

# 핵심 정리

- 합성곱 연산을 계층적으로 반복하면 이미지가 점점 작아질 수 있기 때문에 이미지의 주변에 추가적인 값을 덧붙여 주는데 이를 **패딩**이라 한다. 또한 필터가 적용되는 영역을 **윈도우**라고 하며, 이 윈도우를 움직이는 보폭을 **스트라이드**라고 한다.
- **풀링**은 이미지 내의 일정한 영역 안에 있는 픽셀들이 가지는 여러 값을 하나의 대표 값으로 축소하는 연산으로 원래 이미지가 가지는 **정보를 요약하는 역할**을 하며 강건한 모델을 만드는데 도움이 된다.
- 풀링은 **최대값 풀링**, **평균값 풀링**등의 기법을 사용하여 여러 픽셀의 정보를 통합할 수 있다.
- 이미지 처리를 연구해온 연구자들은 **이미지넷**이라는 학습 데이터 집합을 구축하여 모델의 성능을 평가하는 척도로 삼아왔다. 이미지넷의 방대한 데이터를 사용하여 합성곱 신경망의 성능을 개선하는 시도가 오랫동안 이루어졌다.
- 합성곱 신경망을 오랫동안 연구해온 연구자들에 의해서 좋은 신경망 모델이 발표되었으며 이미 학습이 완료된 모델을 목적에 맞게 고쳐쓰는 것을 **전이학습**이라 한다.