7. CNN

2018년 8월 28일 화요일 오전 10:48

■ **CNN (Convolution neural network)**

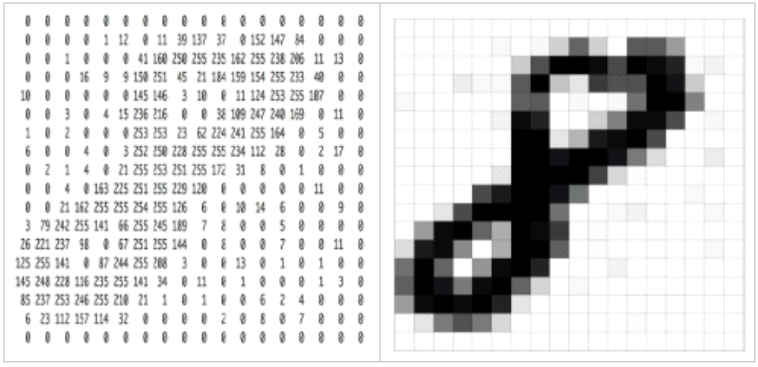
합성곱 신경망 ?

" Convolution 층과 pooling 층을 포함하는 신경망 "

기존 신경망과의 차이?

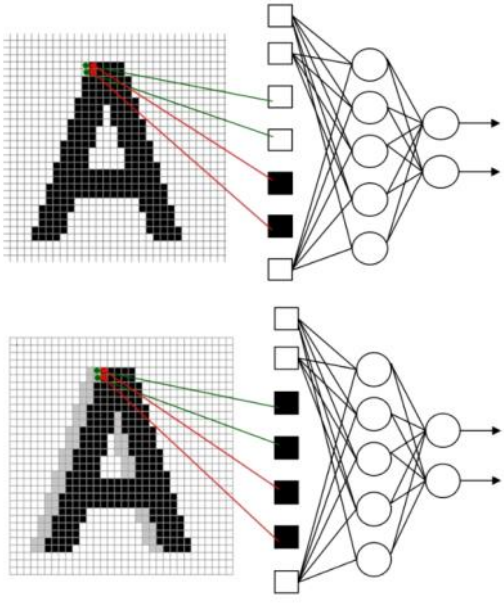
- 기존방법 : Affine ----> ReLu
- CNN : Conv ----> ReLu ----> Pooling

1. 아래의 숫자 8 필기체 784개의 픽셀값이 신경망에 입력이 되는데



2. CNN 을 이용하지 않았을때의 문제점은 ?

전체 글자에서 단지 2픽셀 값만 달라지거나 2픽셀씩 이동만 하더라도 새로운 학습 데이터로 처리를 해줘야 하는 문제점이 있다.

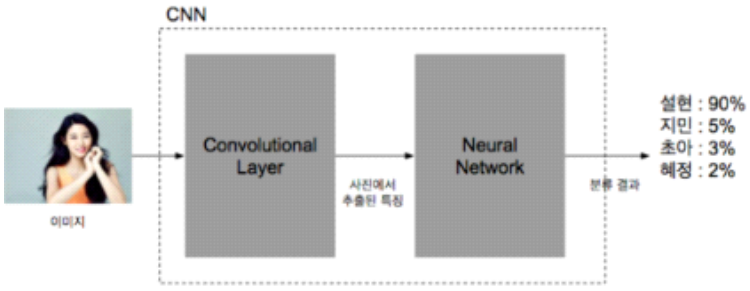


또한 글자의 크기가 달라지거나, 글자가 회전하거나, 글자에 변형 (distortion)이 조금만 생기더라도 새로운 학습 데이터를 넣어주지 않으면 좋은 결과를 기대하기 어렵다.

결론적으로 기존 multi - layerd neural network는 글자의 topology는 고려하지 않고, 말 그대로 raw data에 대해 직접적으로 처리하기 때문에 엄청나게 많은 학습 데이터를 필요로 하고, 또한 거기에 따른 학습 시간을 대가로 지불해야되는 문제점이있다.

3. 그래서 CNN을 사용하게 되면?

cnn은 전통적인 뉴럴 네트워크 앞에 여러 계층의 컨볼루셔널 계층을 붙인 모양이 되는데, 그 이유는 다음과 같다.
cnn은 앞의 컨볼루셔널 계층을 통해서 입력 받은 이미지에 대한 특징(Feature)를 추출하게 되고, 이렇게 추출된 특징을 기반으로 기존의 뉴럴 네트워크를 이용하여 분류를 해내게 된다.

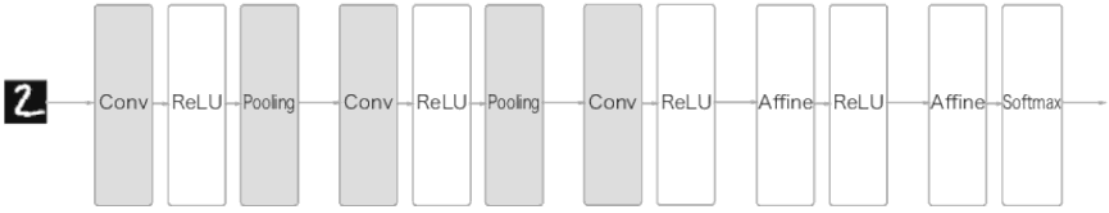


형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하기 때문에 이미지가 갖는 본질적인 패턴을 읽지 못한다. ---> 그래서 합성곱이 필요하다.

결국 원본 이미지에서 조금만 모양이 달라져도 같은 이미지로 인식하지 못하는 문제를 합성곱이 해결해줄 수 있다.

** 어떻게 해결하는가?
원본이미지를 가지고 여러개의 feature map을 만들어서 분류하는 완전 연결 계층에 입력한다.

■ 합성곱 계층



"feature map을 만들고 그 feature map을 선명하게 해주는 층"

** 합성곱 연산 ? 이미지 3차원 (세로,가로,색상) data의 형상을 유지하면서 연산하는 작업

합성곱층 ----> 풀링층

** 합성곱층의 역할 ? 이미지에서 특징 (feature map)을 추출하는 작업
** pooling 층의 역할 ? 이미지를 더 선명하게

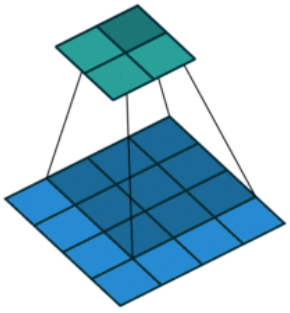
"입력 데이터에 필터를 적용한 것이 합성곱 연산이다. "

- 합성곱 연산을 컴퓨터로 구현하는 방법

```
1 2 3 0          2 0 1
0 1 2 3   ◎      0 1 2    =  15  16
3 0 1 2          1 0 2        6   15
2 3 0 1
```

입력 데이터 (4,4) 필터 (3,3) (2,2)

■ Striding 1 의 Convolution 연산 (합성곱 연산)



** 출력크기 공식

입력크기를 (H,W), 필터크기를 (FH,FW),
출력크기를 (OH,OW), 패딩을 P, 스트라이드를 S라고 할 때 출력 크기의 공식

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

----> P(패딩) = (OH -1) * S -H +FH / 2

■ 3차원의 합성곱 (p235)

이미지의 색깔이 보통은 흑백이 아니라 RGB 컬러이므로 RGB(Red,Green,Blue) 컬러에 대해서 합성곱을 해야한다.

■ 합성곱 총정리

합성곱 ? 이미지의 특징 (featurs amp)을 추출하는 과정
--> filter(가중치)를 이용해서 추출한다.

원본이미지 1장 * 필터 50개 = 50개의 feature mpa의 갯수

3. 합성곱 연산

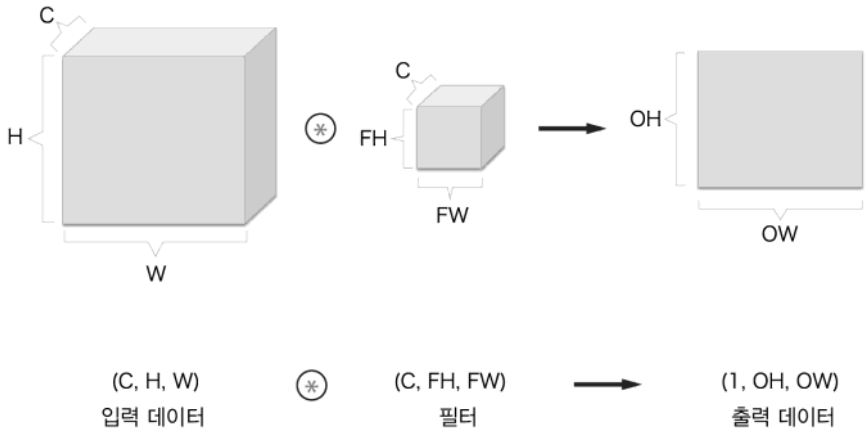
" 합성곱층의 역할을 컴퓨터로 구현 "
원본 이미지를 필터의 크기로 스트라이드 하면서 feature map을 출력하는 연산

4. 3차원 합성곱

" RGB 이미지를 RGB 필터로 합성곱 연산 "

■ 블록으로 생각하기 (p237)

3차원 합성곱 연산은 데이터와 필터를 직육면체 블럭이라고 생각하면 쉽다.

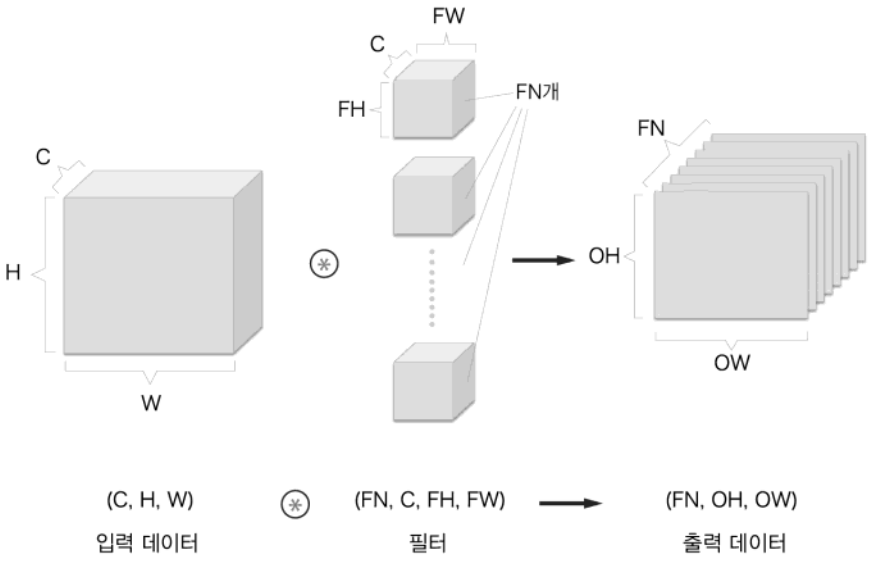


#필터 한 장당 feature map 한 장 나옴

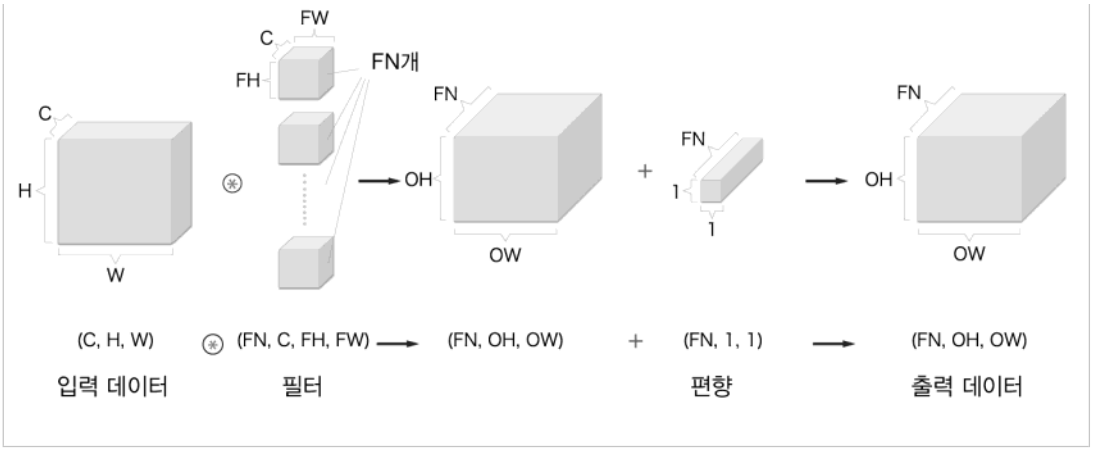
위의 그림은 feature map이 한 개가 나오고 있는데 실제로는 아이린 사진 한 장에 대해서 여러개의 feature map이 필요하다. 여러개의 feature map을 출력하려면 어떻게 해야하는가?

채널 수 : C
높이 : H
너비 : W
FH : 필터 높이
FW : 필터 너비

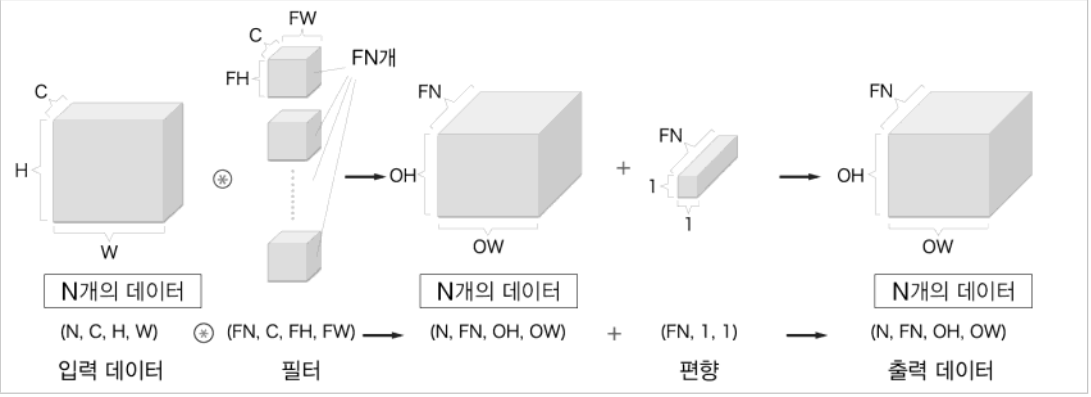
여러개의 feature map을 출력하려면 어떻게 해야하는가?
-->filter 의 갯수를 늘린다



합성 곱 연산에서도 편향이 쓰이므로 편향을 더하면 어떤 그림일까?



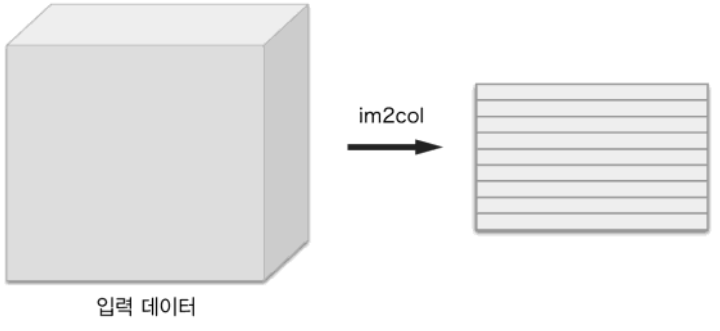
위의 그림은 이미지를 1장씩 넣어서 학습 시키는 것이므로 학습속도가 느리므로 여러장의 이미지를 한 번에 입력해서 학습시키면 (mini batch) 아래의 그림이 된다.

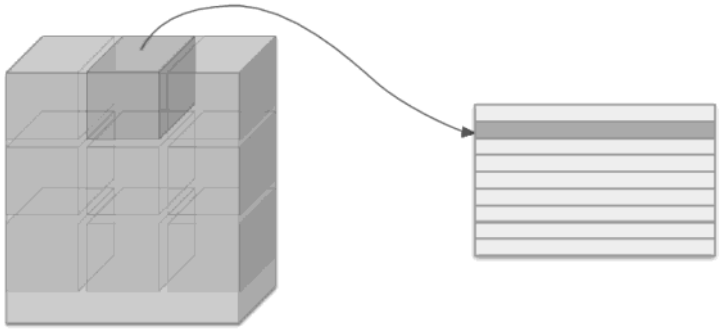


그러면 결국 합성곱 계층을 구현할 때 흘러가는 4차원 행렬이 연산되는데, 그러면 연산 속도가 느리므로 행렬 **연산을 빠르게 하려면 4차원이 아니라 2차원으로 차원 축소**를 해야한다.

그래서 필요한 함수 ?

" im2col 함수 " <---- p 234
4차원 ------> 2차원





합성곱 연산을 빠르게 진행하기 위해 im2col 함수를 이용했다.

--2차원으로 변경해야 할 행렬 2가지

1. **원본 이미지**를 필터 사이즈에 맞게 2차원으로 변경한 행렬 (im2col 함수 사용)

   원본이미지 (장 수, 채널, 높이 , 너비)[4차원] ---- im2col함수 -----> (높이, 너비) [2차원]

2. 4차원 **필터 행렬**을 2차원으로 변경

   원본 필터 (필터의 개수, 채널, 높이, 너비) ---- reshape함수 ------> (높이, 너비) [2차원]

■ convolution 클래스 내에서 일어나는 일

1. 원본 이미지를 im2col로 2차원 행렬로 변경 한다.
2. filter 를 reshape함수를 이용해서 2차원 행렬로 변경한다.
3. 2차원 행렬로 변경한 두 행렬을 내적한다.
4. 내적한 결과인 2차원 행렬을 다시 4차원으로 변경한다.

■ pooling

풀링층의 역할은 말 그대로 출력 값에서 일부분만 취하는 기능이다.
convolution이 이렇게 저렇게 망쳐 놓은 그림들을 각 부분에서 대표들을 뽑아 사이즈가 작은 이미지를 만드는 것이다. 마치 사진을 축소하면 해상도가 좋아지는 듯한 효과와 비슷하다.

풀링의 종류 3가지

a. 최대 풀링 : 컨볼루션 데이터에서 가장 큰 값을 대표값으로 선정한다.

a. 평균 풀링 : 컨볼루션 데이터에서 모든 값의 평균 값을 대표값으로 선정
b. 확률적 풀링 : 컨볼루션 데이터에서 임의 확률로 한 개를 선정

■ CNN 층의 구조

conv 층 -----> pooling 층 -----> fully connected 층

convolution : 이미지의 특징[feature map]을 추출하는 층
pooling 층 :
fully connected (완전 연결 계층 [Affine]) :

문제 173.	아래의 두 행렬을 만들고 합성곱 한 결과인 15를 파이썬으로 출력 하시오.

```
import numpy as np

a = np.array([[1,2,3],[0,1,2],[3,0,1]])
b = np.array([[2,0,1],[0,1,2],[1,0,2]])

print(a*b)
print(np.sum(a*b))
```

****결과**

```
[[2 0 3]
 [0 1 4]
 [3 0 2]]
15
```

문제 174.	아래의 4x4행렬에서 아래의 3x3 행렬만 추출 하시오. [[1 2 3 0] [0 1 2 3] [3 0 1 2] [2 3 0 1]] ----> [[1 2 3] [0 1 2] [3 0 1]]

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])

b = np.array([a[0][:3],a[1][:3],a[2][:3]])
print(b)
```

****결과**

```
[[1 2 3]
 [0 1 2]
 [3 0 1]]
```

문제 175.	아래의 행렬에서 아래의 결과 행렬을 추출 하시오. [[1 2 3 0] [0 1 2 3] [3 0 1 2] [2 3 0 1]]

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
print(a[0:3,0:3])
print()
print(a[0:3,1:4])
print()
print(a[1:4,0:3])
print()
```

```
print(a[1:4,1:4])
```

**결과

```
[[1 2 3]
 [0 1 2]
 [3 0 1]]
[[2 3 0]
 [1 2 3]
 [0 1 2]]
[[0 1 2]
 [3 0 1]
 [2 3 0]]
[[1 2 3]
 [0 1 2]
 [3 0 1]]
```

문제 176. 아래의 4x4 행렬에서 아래의 결과를 출력 하시오.

[[1 2 3 0] [0 1 2 3] [3 0 1 2] [2 3 0 1]]	◎	[[2 0 1] [0 1 2] [1 0 2]]	[15, 16, 6, 15]

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
print(a,end='₩n₩n')
print(a[0:3,0:3],end='₩n₩n')
print(a[0:3,1:4],end='₩n₩n')
print(a[1:4,0:3],end='₩n₩n')
print(a[1:4,1:4],end='₩n₩n')

n=3
lst=[]
for i in range(len(a[0])):
    for j in range(len(a)):
        if i+n <= len(a) and j+n <= len(a[0]):
            lst.append(np.sum(a[i:i+n,j:j+n]))

print(lst)
```

**결과

```
[[1 2 3 0]
 [0 1 2 3]
 [3 0 1 2]
 [2 3 0 1]]
[[1 2 3]
 [0 1 2]
 [3 0 1]]
[[2 3 0]
 [1 2 3]
 [0 1 2]]
[[0 1 2]
 [3 0 1]
```

```
 [2 3 0]]
[[1 2 3]
 [0 1 2]
 [3 0 1]]
[13, 14, 12, 13]
```

문제 177. 아래의 합성곱을 파이썬으로 구현 하시오.

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
b = np.array([[2,0,1],[0,1,2],[1,0,2]])

n=3
lst=[]
for i in range(len(a[0])):
    for j in range(len(a)):
        if i+n <= len(a) and j+n <= len(a[0]):
            c= a[i:i+n,j:j+n] * b
            lst.append(np.sum(c))

print(lst)
```

****결과**

```
[15, 16, 6, 15]
```

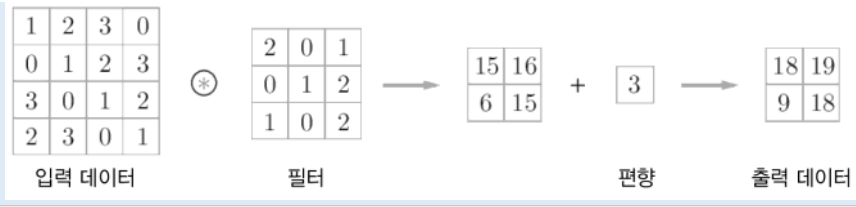
문제 178. 위에서 출력한 결과인 1차원 배열 [15, 16, 6, 15] 결과를 2x2 행렬로 변경 하시오.

```
import numpy as np
a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
filter = np.array([[2,0,1],[0,1,2],[1,0,2]])

result =  []
for  rn  in  range(len(a) - 2):
    for  cn  in  range(len(a) -2):
        result.append( np.sum(a[rn:rn+3,cn:cn+3] * filter) )

print (result)

result = np.array(result).reshape(2,2)
print(result)
```

****결과**

```
[15, 16, 6, 15]
[[15 16]
 [ 6 15]]
```

문제 179. 아래의 그림의 convolution 연산을 파이썬으로 구현 하시오.




```
import numpy as np
a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
filter = np.array([[2,0,1],[0,1,2],[1,0,2]])
b=3
result =  []
for  rn  in  range(len(a) - 2):
    for  cn  in  range(len(a) -2):
        result.append( np.sum(a[rn:rn+3,cn:cn+3] * filter) )

print (result)

result = np.array(result).reshape(2,2)
print(result+b)
```

****결과**

```
[15, 16, 6, 15]
[[18 19]
 [ 9 18]]
```

문제 180.	위에서 출력한 2x2 행렬에 제로패딩 1을 수행 하시오.

```
import numpy as np

result = np.array([[15,16],[6,15]])
result_pad = np.pad(result, pad_width=1,mode='constant',constant_values=0)

print(result_pad)
```

****결과**

```
[[ 0  0  0  0]
 [ 0 15 16  0]
 [ 0  6 15  0]
 [ 0  0  0  0]]
```

문제 181.	4x4 행렬에 3x3 필터를 적용해서 결과로 4x4 행렬이 출력되게 하려면 제로패딩을 몇으로 해줘야 하는가?

```
import numpy as np

result = np.array([[15,16],[6,15]])
result_pad = np.pad(result, pad_width=1,mode='constant',constant_values=0)

print(result_pad)
```

****결과**

```
[[ 0  0  0  0]
 [ 0 15 16  0]
 [ 0  6 15  0]
```

```
[ 0  0  0  0]]
```

문제 182.	패딩을 답으로 해서 아래의 식을 풀어보시오. $$OH = \frac{H + 2P - FH}{S} + 1$$ $$OW = \frac{W + 2P - FW}{S} + 1$$ 입력크기를 (H,W), 필터크기를 (FH,FW), 출력크기를 (OH,OW), 패딩을 P, 스트라이드를 S라고 할 때 출력 크기의 공식

```
P = (OH -1) * S -H +FH / 2
```

문제 183.	입력 이미지 4x4 행렬에 필터 3x3 행렬을 합성곱 한 출력 결과 행렬이 4x4 행렬이 되려면 패딩이 몇인지 출력 하시오.

```
P = (4 -1) *1 -4 +3 / 2
**출력결과
1
```

문제 184.	위의 패딩 공식을 구현하는 파이썬 함수를 구현 하시오.

```
def padding(h,s,oh,fh):
    return (s*(oh-1)-h+fh)//2
```

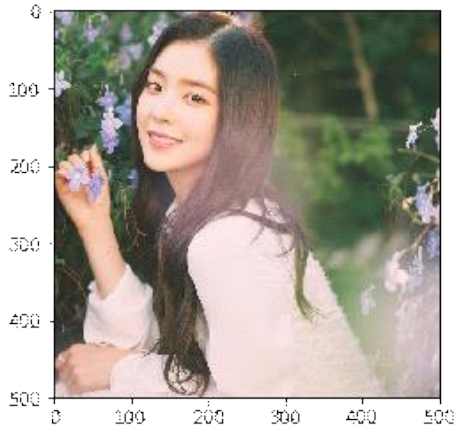
문제 185.	입력행렬(6x6), 스트라이드 1, 출력행렬 (6x6), 필터행렬 (3x3)의 패딩이 어떻게 되는지 출력 하시오.

```
def padding(h,s,oh,fh):
    return (s*(oh-1)-h+fh)//2

print(padding(6,1,6,3))

**결과
1
```

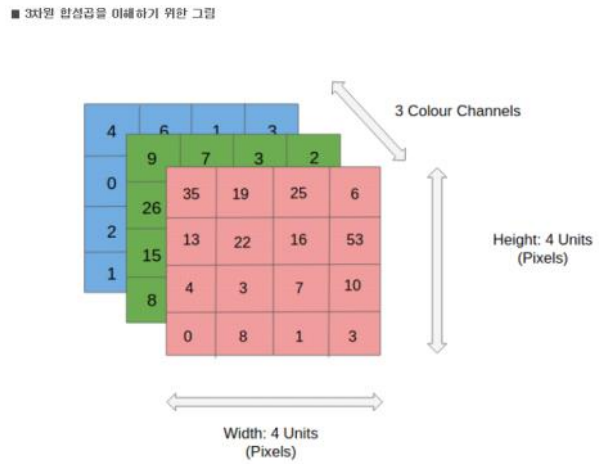
문제 186.	레드벨벳의 아이린 사진을 3차원 행렬로 변환하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

##5.1. 원본 이미지 불러오기

img = Image.open('c:₩₩data₩₩irin.jpg')
img_pixel = np.array(img)
print(img_pixel)
print(img_pixel.shape)
plt.imshow(img_pixel)
**결과
```

```
[[[ 79  96 104]
  [113 118 138]
  [147 137 172]
  ...
  [ 50  72  51]
  [ 50  72  51]
  [ 50  72  51]]
[[101 118 128]
  [137 142 162]
  [169 159 194]
  ...
  [165 150 147]
  [176 160 160]
  [184 168 169]]]
(500, 500, 3)
```



##설명



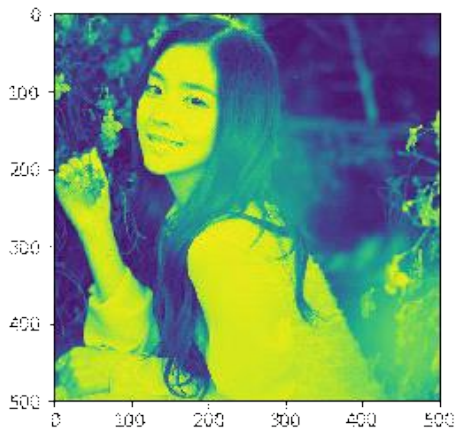
레드,그린,블루의 3차원 배열로 이루어져 있다.

문제 187.	레드벨벳의 아이린 사진에서 Red 행렬만 출력하고 Red행렬만 시각화 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = Image.open('c:₩₩data₩₩irin.jpg')
```

```
img_pixel = np.array(img)
plt.imshow(img_pixel[:,:,0])
```

****결과**



##설명

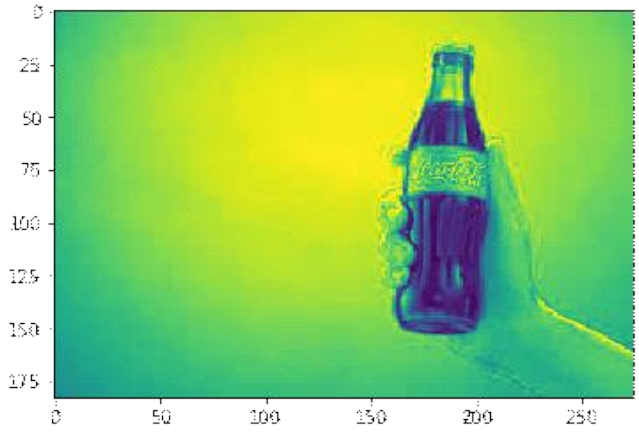
```
plt.imshow(img_pixel[:,:,0])   #   0 : 레드, 1 : 그린, 2: 블루
```

문제 188.	원하는 사진을 선택해서 RGB를 확인 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = Image.open('c:₩₩data₩₩tt.jpg')

img_pixel = np.array(img)
print(img_pixel.shape)
plt.imshow(img_pixel[:,:,0])
```

****결과**

```
(183, 275, 3)
```



문제 189.	R,G,B 행렬을 이해하기 위한 아래의 numpy array 를 이해하시오 !

```
data = np.array(
    [
```

```
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],
         [1, 0, 1, 0, 1],   #레드

        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2],  #그린

        [[4, 2, 1, 2],
         [0, 1, 0, 4],
         [3, 0, 6, 2],
         [4, 2, 4, 5]]   #블루
    ])
```

문제 190.	문제 189번 data행렬의 shape를 확인 하시오.

```
import numpy as np

data = np.array(
    [
      [[2, 2, 1, 1, 0],
       [0, 0, 1, 0, 0],
       [0, 2, 0, 0, 1],
       [1, 2, 1, 1, 1],
       [1, 0, 1, 0, 1]],

      [[2, 0, 0, 0, 1],
       [0, 2, 2, 0, 1],
       [0, 0, 0, 0, 2],
       [0, 1, 2, 0, 1],
       [2, 0, 2, 2, 2]],

      [[4, 2, 1, 2,1],
       [0, 1, 0, 4,1],
       [3, 0, 6, 2,0],
       [4, 2, 4, 5,2],
      [2, 0, 2, 2, 2]]
    ])

print(data.shape)
```

****결과**

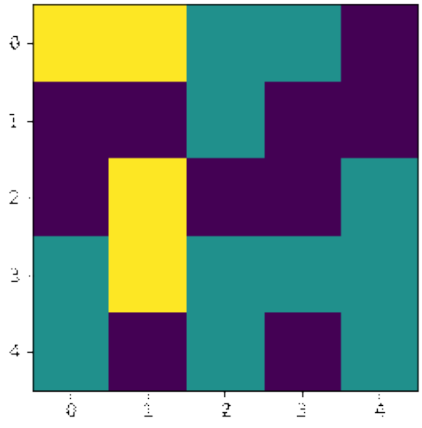
```
(3, 5, 5)
```

문제 191.	위의 data 행렬에서 Red 행렬만 출력 하고 시각화 하시오.

```
data = np.array(
    [
```

```
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],
         [1, 0, 1, 0, 1]],

        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],

        [[4, 2, 1, 2,1],
         [0, 1, 0, 4,1],
         [3, 0, 6, 2,0],
         [4, 2, 4, 5,2],
        [2, 0, 2, 2, 2]]
       ])

print(data[0])
plt.imshow(data[0])
```

****결과**



문제 192.	아래의 numpy array의 행렬을 확인 하시오.

```
import numpy as np

Filter = np.array([[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]])

print(Filter)
print(Filter.shape)
```

****결과**

```
[[[ 1  1 -1 -1  0  0  1  1  0]
  [-1 -1  0  0 -1  1  0 -1  0]
  [-1  1  1 -1  1 -1  0  0 -1]]]
(1, 3, 9)
```

문제 193. 위의 행렬을 (3,3,3)행렬로 변환 하시오.

```
import numpy as np

Filter = np.array([[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

print(Filter)
print(Filter.shape)
```

****결과**

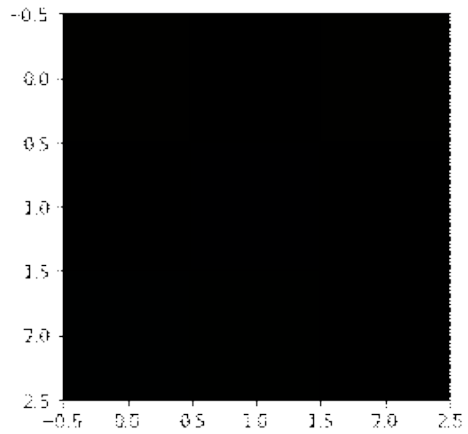
```
[[[ 1  1 -1]
  [-1  0  0]
  [ 1  1  0]]
[[-1 -1  0]
  [ 0 -1  1]
  [ 0 -1  0]]
[[-1  1  1]
  [-1  1 -1]
  [ 0  0 -1]]]
(3, 3, 3)
```

문제 194. 위의 (3,3,3) 행렬을 시각화 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

Filter = np.array([[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

print(Filter)
print(Filter.shape)

plt.imshow(Filter)
```

****결과**

```
[[[ 1  1 -1]
  [-1  0  0]
  [ 1  1  0]]
[[-1 -1  0]
  [ 0 -1  1]
  [ 0 -1  0]]
[[-1  1  1]
  [-1  1 -1]
  [ 0  0 -1]]]
(3, 3, 3)
```

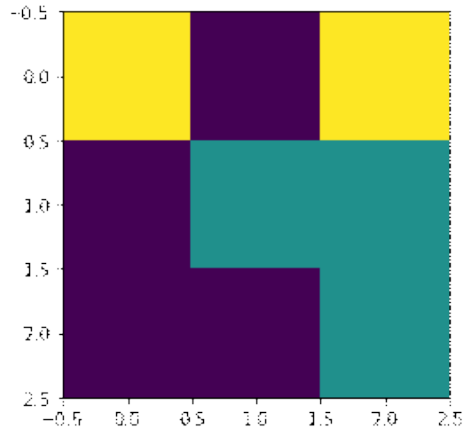


0, -1 만 존재하므로 검정색만 출력된다.

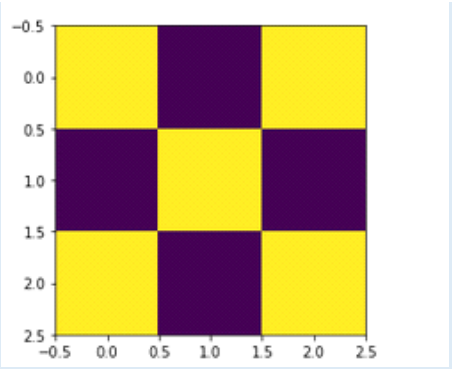
문제 195.	위의 (3,3,3) 행렬의 Red, Green, Blue 행렬을 각각 시각화 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

Filter = np.array([[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

plt.imshow(Filter[:,:,0])
plt.show()
plt.imshow(Filter[:,:,1])
plt.show()
plt.imshow(Filter[:,:,2])
plt.show()
```

****결과**



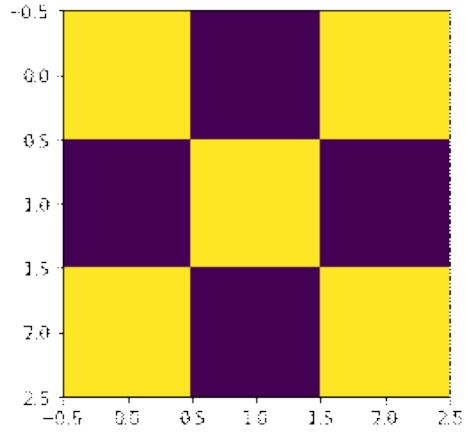
문제 196.	아래의 필터를 생성 하시오.



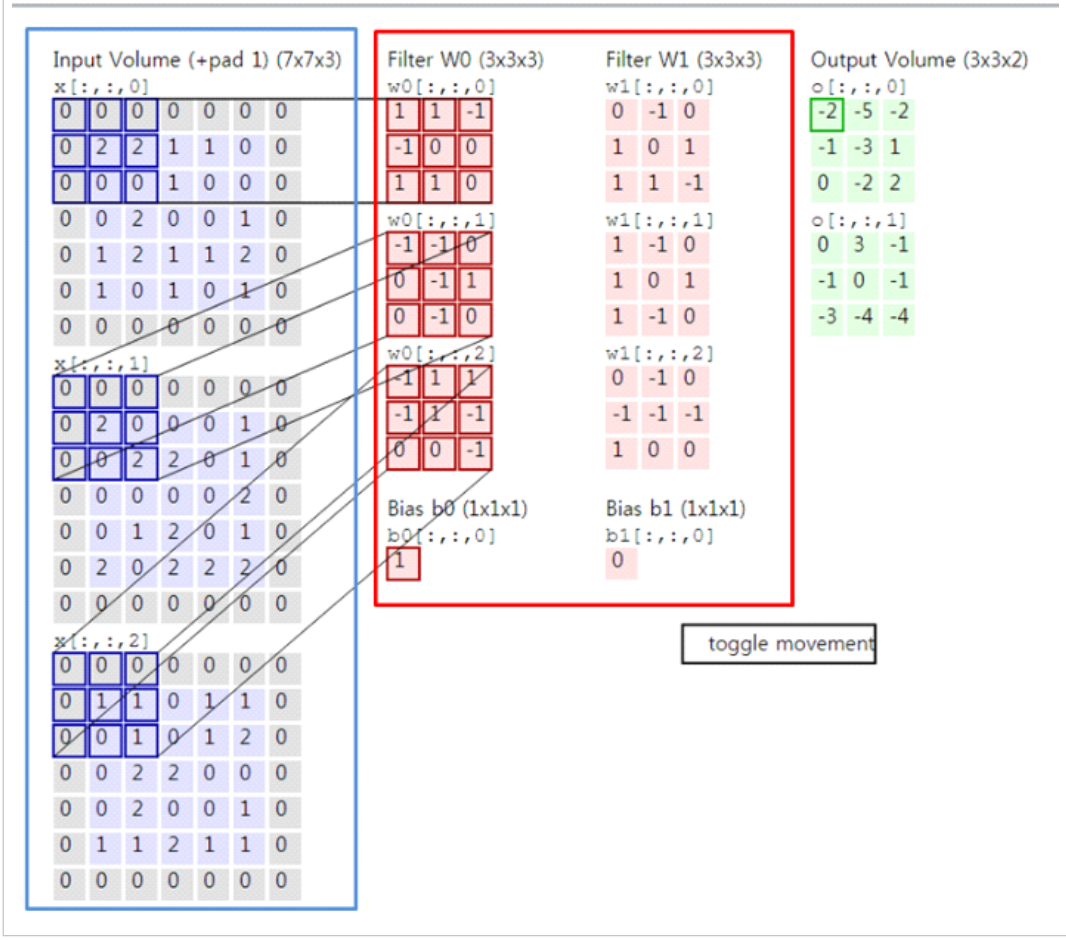

```
filter= np.array([[[1,-1,1],[-1,1,-1],[1,-1,1]]])
print(filter.shape)
plt.imshow(filter[0,:,:])
plt.show()
```

****결과**

(1, 3, 3)



문제 197.	아래의 원본 이미지인 data 행렬과 filter 행렬을 3차원 합성곱 하는 그림을 이해하시오.



문제 198. 아래의 원본 이미지의 zero 패딩 1한 결과를 출력 하시오.

```
data = np.array(
    [
      [[2, 2, 1, 1, 0],
       [0, 0, 1, 0, 0],
       [0, 2, 0, 0, 1],
       [1, 2, 1, 1, 1],    #  --->  Red
       [1, 0, 1, 0, 1]],
      [[2, 0, 0, 0, 1],
       [0, 2, 2, 0, 1],
       [0, 0, 0, 0, 2],    # ---->  Green
       [0, 1, 2, 0, 1],
       [2, 0, 2, 2, 2]],
      [[4, 2, 1, 2,2],
       [0, 1, 0, 4,1],      # ---->  Blue
       [3, 0, 6, 2,1],
       [4, 2, 4, 5,4],
       [0, 1, 2, 0, 1]]
    ])

data_pad = np.pad(data, pad_width=1,mode='constant',constant_values=0)[1:4]
#data_pad = np.pad(data, pad_width=((0,0),(1,1),(1,1)),mode='constant',constant_values=0)  # 또 다른 방법
print(data_pad)
```

****결과**

```
[[[0 0 0 0 0 0 0]
  [0 2 2 1 1 0 0]
```

```
  [0 0 0 1 0 0 0]
  [0 0 2 0 0 1 0]
  [0 1 2 1 1 1 0]
  [0 1 0 1 0 1 0]
  [0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0]
  [0 2 0 0 0 1 0]
  [0 0 2 2 0 1 0]
  [0 0 0 0 0 2 0]
  [0 0 1 2 0 1 0]
  [0 2 0 2 2 2 0]
  [0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0]
  [0 4 2 1 2 2 0]
  [0 0 1 0 4 1 0]
  [0 3 0 6 2 1 0]
  [0 4 2 4 5 4 0]
  [0 0 1 2 0 1 0]
  [0 0 0 0 0 0 0]]]
```

문제 199.	아래의 코드를 이해하시오.

```
data = np.array(
       [
         [[2, 2, 1, 1, 0],
          [0, 0, 1, 0, 0],
          [0, 2, 0, 0, 1],
          [1, 2, 1, 1, 1],    #  --->  Red
          [1, 0, 1, 0, 1]],
         [[2, 0, 0, 0, 1],
          [0, 2, 2, 0, 1],
          [0, 0, 0, 0, 2],    # ---->  Green
          [0, 1, 2, 0, 1],
          [2, 0, 2, 2, 2]],
         [[4, 2, 1, 2,2],
          [0, 1, 0, 4,1],      # ---->  Blue
          [3, 0, 6, 2,1],
          [4, 2, 4, 5,4],
          [0, 1, 2, 0, 1]]
       ])

data_pad = np.pad(data, pad_width=((0,0),(1,1),(1,1)),mode='constant',constant_values=0)  # 또 다른 방법
print(data_pad)

**결과
[[[0 0 0 0 0 0 0]
  [0 2 2 1 1 0 0]
  [0 0 0 1 0 0 0]
  [0 0 2 0 0 1 0]
  [0 1 2 1 1 1 0]
  [0 1 0 1 0 1 0]
  [0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0]
  [0 2 0 0 0 1 0]
```

```
 [0 0 2 2 0 1 0]
 [0 0 0 0 0 2 0]
 [0 0 1 2 0 1 0]
 [0 2 0 2 2 2 0]
 [0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0]
 [0 4 2 1 2 2 0]
 [0 0 1 0 4 1 0]
 [0 3 0 6 2 1 0]
 [0 4 2 4 5 4 0]
 [0 0 1 2 0 1 0]
 [0 0 0 0 0 0 0]]]
```

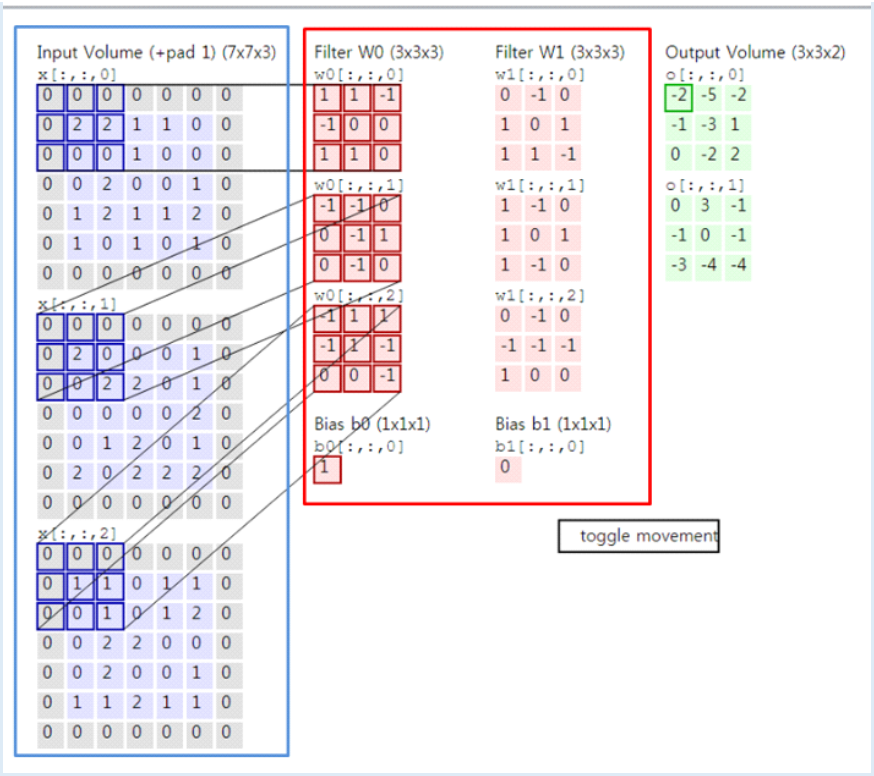
문제 200.	한 개의 zero 패딩한 Red 행렬에서 아래의 행렬만 추출 하시오.

```
data = np.array(
       [
         [[2, 2, 1, 1, 0],
          [0, 0, 1, 0, 0],
          [0, 2, 0, 0, 1],
          [1, 2, 1, 1, 1],    #  --->  Red
          [1, 0, 1, 0, 1]],
         [[2, 0, 0, 0, 1],
          [0, 2, 2, 0, 1],
          [0, 0, 0, 0, 2],    # ----> Green
          [0, 1, 2, 0, 1],
          [2, 0, 2, 2, 2]],
         [[4, 2, 1, 2,2],
          [0, 1, 0, 4,1],      # ----> Blue
          [3, 0, 6, 2,1],
          [4, 2, 4, 5,4],
          [0, 1, 2, 0, 1]]
       ])

data_pad = np.pad(data, pad_width = ((0,0),(1,1),(1,1)), mode = 'constant', constant_values = 0)

print(data_pad[0,:3,:3])

print(data[0][0:3,0:3])
```

****결과**

```
[[0 0 0]
 [0 2 2]
 [0 0 0]]
```

문제 201.	위의 3x3 행렬의 R,G,B 행렬을 각각 출력하면 ?




```
[[0 0 0]
 [0 2 2]
 [0 0 0]]

[[0 0 0]
 [0 2 0]
 [0 0 2]]

[[0 0 0]
 [0 1 1]
 [0 0 1]]
```

```
import numpy as np

data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],  # --->  Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],  # ---->  Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2],  # ---->  Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
```

```
            mode='constant', constant_values=0)
print(pad[0,:3,:3])
print(pad[1,:3,:3])
print(pad[2,:3,:3])
```

**결과

```
[[0 0 0]
 [0 2 2]
 [0 0 0]]
[[0 0 0]
 [0 2 0]
 [0 0 2]]
[[0 0 0]
 [0 1 1]
 [0 0 1]]
```

문제 202.	아래의 Filter 에서 아래의 행렬을 추출 하시오.

```
Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],   ₩
                  [-1,-1,0,0,-1,1,0,-1,0], ₩
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

print(Filter[0,:,:])
```

**결과

```
[[ 1  1 -1]
 [-1  0  0]
 [ 1  1  0]]
```

문제 203.	위의 3개의 행렬 중 원본 이미지의 Red 행렬과 아래의 filter의 Red 행렬과의 곱을 수행하시오. [[0 0 0] 1 1 -1 0 0 0 [0 2 2] * -1 0 0 = 0 0 0 [0 0 0]] 1 1 1 0 0 0

```
data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],  # --->  Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],  # ---->  Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2],  # ---->  Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])
```

```
pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],    ₩
                  [-1,-1,0,0,-1,1,0,-1,0], ₩
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

print(pad[0,:3,:3]*Filter[0,:,:])
```

****결과**

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

문제 204.	아래의 원본 이미지 RGB 3개의 행렬과 아래의 필터 RGB 3개의 행렬을 각각 행렬곱 한 후 그 원소들을 다 합친 결과 숫자 하나를 출력 하시오. (3차원 합성곱 연산)

```
data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],  # --->  Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],  # ---->  Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2],  # ---->  Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],    ₩
                  [-1,-1,0,0,-1,1,0,-1,0], ₩
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

r=pad[0,:3,:3]*Filter[0,:,:]
b=pad[1,:3,:3]*Filter[1,:,:]
g=pad[2,:3,:3]*Filter[2,:,:]

rst = r+b+g
print(np.sum(rst))
```

****결과**

```
-3
```

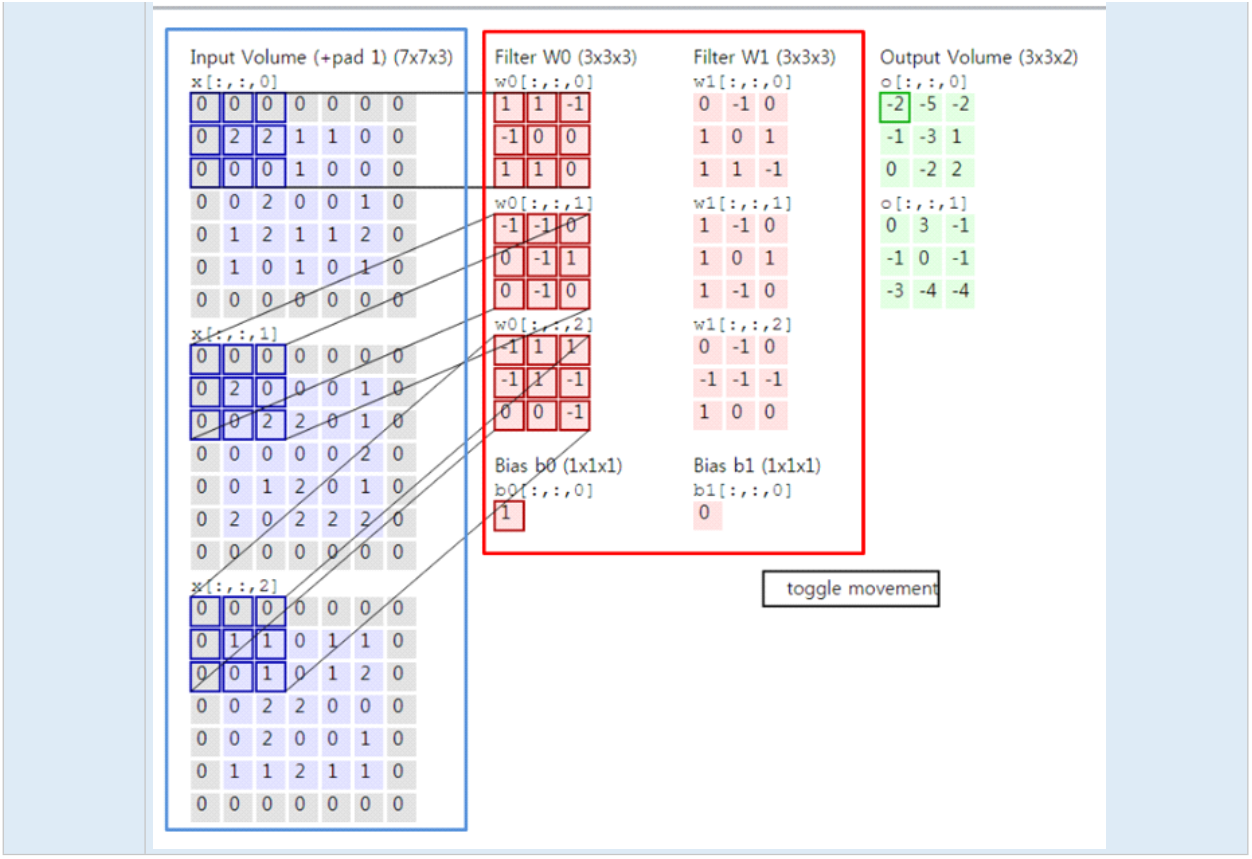
문제 205.	1칸 스트라이드 한 원본 이미지의 3x3 행렬 RGB와 Filter RGB와의 합성곱을 구하시오. [[0 0 0] 1 1 -1 0 0 0 [2 2 1] * -1 0 0 = -2 0 0 [0 0 1]] 1 1 0 0 0 0 [[0 0 0] -1 -1 0 0 0 0 [2 0 0] * 0 -1 1 = 0 0 0 [0 2 2]] 0 -1 0 0 -2 0 [[0 0 0] -1 1 1 0 0 0 [1 1 0] * -1 1 -1 = -1 1 0 [0 1 0]] 0 0 -1 0 0 0

```
data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],  # --->  Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],  # ---->  Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2],  # ---->  Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],    ₩
                  [-1,-1,0,0,-1,1,0,-1,0], ₩
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

r=pad[0,:3,1:4]*Filter[0,:,:]
b=pad[1,:3,1:4]*Filter[1,:,:]
g=pad[2,:3,1:4]*Filter[2,:,:]

rst = r+b+g
print(np.sum(rst))
```

****결과**

-4

문제 206.	아래의 합성곱 연산 (25번 연산)을 하는 결과를 파이썬으로 구현 하시오.



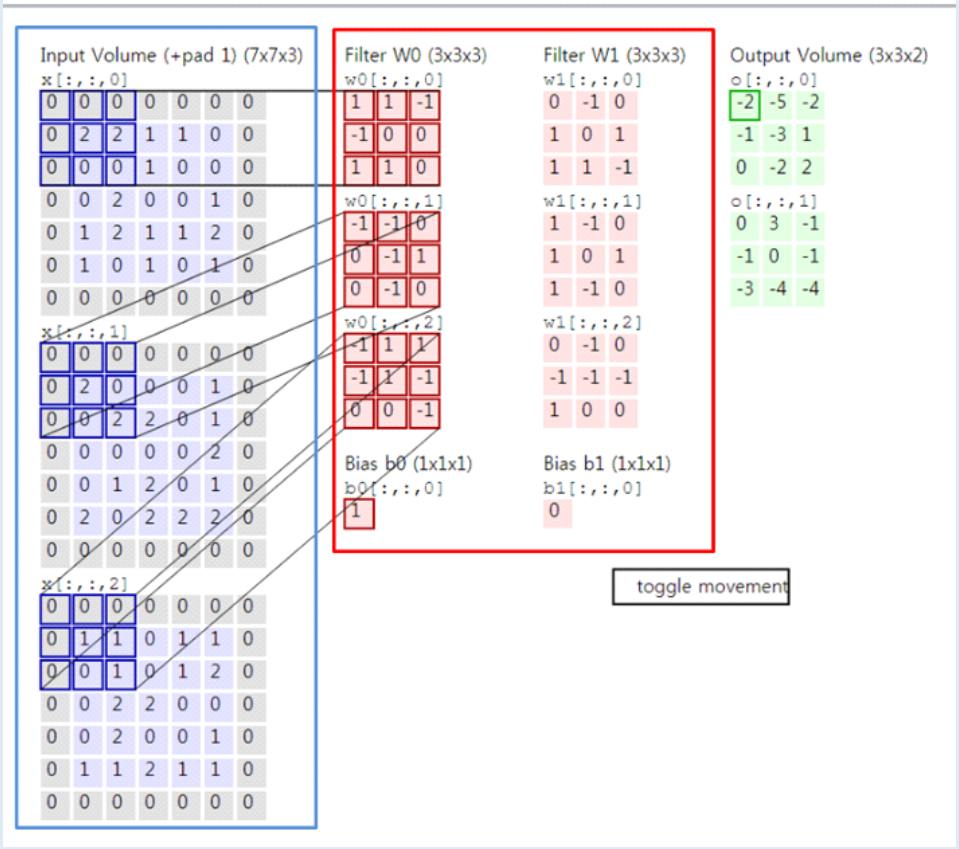

```
data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],  # --->  Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],  # ---->  Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2],  # ---->  Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],    ₩
                  [-1,-1,0,0,-1,1,0,-1,0], ₩
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)
rst=[]
for d in range(len(pad)):
    for i in range(len(pad[0])-len(Filter)+1):
        for j in range(len(pad[0])-len(Filter)+1):
            r=pad[d,i:i+len(Filter),j:j+len(Filter)]*Filter[d,:,:]
            rst.append(np.sum(r))
```

```
rst=np.array(rst).reshape(3,5,5)
print(rst,end='₩n₩n')

print(rst[0]+rst[1]+rst[2])
```

**결과

```
[[[ 0 -2 -1  0 -1]
  [ 0  5  4  1  2]
  [ 1  2  2  3  2]
  [-1  2  1 -1  1]
  [-1  1  2  0  2]]
[[-2 -2 -2  1 -2]
  [ 0 -2 -2  1 -4]
  [ 0 -3 -6  0 -4]
  [-1  1 -4 -1 -5]
  [-2  1 -3 -2 -3]]
[[-1  0 -3 -2  0]
  [-1 -1 -2  1  1]
  [-3  1  0  0  1]
  [-1  4 -3 -4  1]
  [ 2  0 -2 -1  1]]]
[[-3 -4 -6 -1 -3]
 [-1  2  0  3 -1]
 [-2  0 -4  3 -1]
 [-3  7 -6 -6 -3]
 [-1  2 -3 -3  0]]
```

문제 207. 아래와 같이 입력행렬과 필터행렬과 스트라이드와 패딩을 입력받아 출력 행렬의 shape를 출력하는 함수를 구하시오.



```
def out(x,f, s, p):

    FH = f.shape[-2]
    FW = f.shape[-1]
    H = x.shape[-2]
    W = x.shape[-1]

    OH = int(((H+2*p-FH)/s)+1)
    OW = int(((W+2*p-FW)/s)+1)

    pad=np.pad(x,pad_width=((0,0),(p,p),(p,p)),
           mode='constant', constant_values=0)
    rst=[]
    for d in range(len(pad)):
        for i in range(int((len(pad[0])-len(f))/s+1)):
            for j in range(int((len(pad[0])-len(f))/s+1)):
                r=pad[d,i:i+len(Filter),j:j+len(Filter)]*Filter[d,:,:]
                rst.append(np.sum(r))
    print('###shape : OH :',OH, ' OW :',OW)
    rst=np.array(rst)
    return rst.reshape(int(len(rst)/(OH*OW)),OH,OW)

data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],  # --->  Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],  # ---->  Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2],  # ---->  Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])


Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],
                  [-1,-1,0,0,-1,1,0,-1,0],
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)  # 개수, 행, 열

out(data,Filter,1,1)
```

****결과**

```
###shape : OH : 5  OW : 5
array([[[ 0, -2, -1,  0, -1],
        [ 0,  5,  4,  1,  2],
        [ 1,  2,  2,  3,  2],
        [-1,  2,  1, -1,  1],
```

```
        [-1,  1,  2,  0,  2]],
        [[-2, -2, -2,  1, -2],
         [ 0, -2, -2,  1, -4],
         [ 0, -3, -6,  0, -4],
         [-1,  1, -4, -1, -5],
         [-2,  1, -3, -2, -3]],
       [[-1,  0, -3, -2,  0],
        [-1, -1, -2,  1,  1],
        [-3,  1,  0,  0,  1],
        [-1,  4, -3, -4,  1],
        [ 2,  0, -2, -1,  1]]])
```

문제 208.	설현 사진 50장과 아이린 사진 50장, 총 100장의 사진을 신경망에 입력해서 설현과 아이린을 구분(분류) 하는 신경망을 만든다고 할 때 RGB필터를 30개 사용하면 Feature map은 총 몇개 일까 ?

100장 ----> 3000장

문제 209.	칠판에 나온 아이린 사진 한 장의 3차원 행렬을 만드시오. (RGB 7x7 행렬 1장)

```
import numpy as np

x1 = np.random.rand(1,3,7,7)
print(x1)
print(x1.shape)
```

****결과**

```
[[[[0.04146522 0.75111956 0.3096085  0.70458906 0.78363936 0.52925284
    0.03027472]
   [0.84720142 0.9510707  0.71125988 0.42433048 0.75146977 0.92061094
    0.39863146]
   [0.84406455 0.76879374 0.6406376  0.87873444 0.97485125 0.98397857
    0.7300184 ]
   [0.10790605 0.85568982 0.50554661 0.60595911 0.55054506 0.7559521
    0.53092377]
   [0.46822921 0.18695024 0.68807195 0.04881204 0.55263676 0.87514535
    0.36401664]
   [0.16108935 0.34637151 0.69821563 0.92129002 0.9331873  0.25258835
    0.27808362]
   [0.76620471 0.92373867 0.48886042 0.89476054 0.26297989 0.62433865
    0.76431686]]
[[0.20595574 0.10288318 0.36168794 0.81557768 0.90056976 0.2643665
    0.08590275]
   [0.84876066 0.13131874 0.15082708 0.77715371 0.42781956 0.44429663
    0.16916134]
   [0.38209403 0.81437037 0.71543096 0.20895893 0.26314861 0.73417797
    0.48657193]
   [0.45288346 0.68045069 0.40808002 0.74165353 0.64106329 0.462943
    0.64699237]
   [0.87769469 0.87687895 0.15368963 0.8452533  0.9300904  0.29249455
    0.4777687 ]
   [0.26382278 0.51740736 0.30171081 0.94858638 0.24544676 0.66503829
    0.51625294]
   [0.66949584 0.49436196 0.80933866 0.41411674 0.04888242 0.503647
```

```
    0.85257683]]
[[0.13118409 0.34162458 0.26272157 0.5251492  0.97501604 0.54128155
    0.78062057]
   [0.41335574 0.92593055 0.65730433 0.49590699 0.77932026 0.71341426
    0.83236066]
   [0.10497102 0.19541296 0.88130376 0.04252038 0.32205706 0.40305427
    0.11580042]
   [0.6650085  0.79719574 0.89253871 0.64035887 0.30685577 0.08802892
    0.54560454]
   [0.88878912 0.33629819 0.93316654 0.97246735 0.61050836 0.86916122
    0.91330019]
   [0.59210703 0.29721845 0.77380886 0.76259301 0.64428312 0.33293676
    0.43756581]
   [0.83783129 0.55232062 0.40546976 0.91759246 0.15869097 0.04373689
    0.66651265]]]]
(1, 3, 7, 7)
```

문제 210.	im2col 함수를 이용해서 아래의 4차원을 2차원 행렬로 변경 하시오. (필터는 5x5의 RGB 행렬을 사용함)

```
import numpy as np

def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).

    Parameters
    ----------
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -------
    col : 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2 * pad - filter_h) // stride + 1
    out_w = (W + 2 * pad - filter_w) // stride + 1

    img = np.pad(input_data, [(0, 0), (0, 0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N * out_h * out_w, -1)
    return col

x1 = np.random.rand(1,3,7,7)   # 입력데이터 (원본) 4차원
```

```
col = im2col(x1,5,5,stride=1,pad=0)    # 입력데이터를 2차원 행렬로 변환
print(col.shape)
```

**결과
(9, 75) # 한장만 im2col 했을 때

문제 211.	이미지 10장을 랜덤으로 생성 하시오.

```
x10 = np.random.rand(10,3,7,7)
print(x10.shape)
```

**결과
(10, 3, 7, 7)

문제 212.	아이린 사진 10장을 im2col 함수에 넣어서 2차원 행렬로 변환 시키시오. (필터는 5x5의 RGB 행렬을 사용함) --> 입력 데이터(4차원)을 2차원 데이터로 변형

```
x10 = np.random.rand(10,3,7,7)
print(x10.shape)
col = im2col(x10,5,5,stride=1,pad=0)
print(col.shape)
```

**결과
(10, 3, 7, 7)
(90, 75)

문제 213.	mlist 데이터 100장을 im2col 함수에 넣었을 때, 나오는 출력 예상하시오. (필터의 크기 : 5X5 RGB 채널, mlist 이미지 크기 : 28x28 흑백채널)

```
x10 = np.random.rand(10,3,7,7)
print(x10.shape)
col = im2col(x10,5,5,stride=1,pad=0)
print(col.shape)
```

**결과
(100, 1, 28, 28)
(57600, 25)

문제 214.	아래의 filter를 생성하고 shape를 확인하고 전치 시키시오.

```
Filter = np.array([[[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                   [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                   [[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
                   [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                   [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]]], dtype = np.uint8)

print(Filter.shape) # (5,5,3)
```

**결과
(5, 5, 3)

문제 215.	Filter (3,5,5) 행렬을 (3,25) 행렬로 변경 하시오.

```
Filter = np.array([[[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                   [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                   [[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
                   [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                   [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]]], dtype = np.uint8)
print(Filter.shape) # (5,5,3)
a = Filter.T.reshape(3,-1)
print(a.shape)
```

**결과

```
(5, 5, 3)
(3, 25)
```

문제 216.	아래의 4차원 행렬의 filter를 numpy의 random을 이용해서 만드시오.

```
f = np.random.rand(10,3,5,5)
print(f.shape)
```

**결과

```
(10, 3, 5, 5)
```

문제 217.	아래의 4차원 행렬을 3차원으로 변경 하시오. (10,3,5,5) [4차원] -----> (10,3,25) [3차원]

```
f = np.random.rand(10,3,5,5)
print(f.shape)
f = f.reshape(10,3,-1)
print(f.shape)
```

**결과

```
(10, 3, 5, 5)
(10, 3, 25)
```

문제 218.	아래의 3차원 행렬을 2차원 행렬로 변경 하시오.

```
f = np.random.rand(10,3,5,5)
print(f.shape)
f = f.reshape(10,3,-1)
print(f.shape)
f = f.reshape(10,-1)
print(f.shape)
```

**결과

```
(10, 3, 5, 5)
(10, 3, 25)
(10, 75)
```

문제 219.	(10,75) ---> (75, 10)으로 변경 하시오.

```
f = np.random.rand(10,3,5,5)
print(f.shape)
f = f.reshape(10,3,-1)
print(f.shape)
f = f.reshape(10,-1)
print(f.shape)
print(f.T.shape)
```

****결과**

```
(10, 3, 5, 5)
(10, 3, 25)
(10, 75)
(75, 10)
```

문제 220.	책 246페이지에 나오는 Convolution 클래스를 생성 하시오.

```
class Convolution :
    def __init__(self,W,b,stride=1,pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape

        out_h = int(1 + (H +2*self.pad - FH)/self.stride)
        out_w = int(1 + (W +2*self.pad - FW)/self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T
        out = np.dot(col, col_W) + self.b

        out = out.reshape(N, out_h, out_w, -1).transpose(0,3,1,2)

        return out
```

문제 221.	위에서 만든 Convolution 클래스를 객체화 시켜서 칠판에 나온 convolution층을 구현 하시오.

```
class Convolution :
    def __init__(self,W,b,stride=1,pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape

        out_h = int(1 + (H +2*self.pad - FH)/self.stride)
        out_w = int(1 + (W +2*self.pad - FW)/self.stride)
```

```
        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T
        out = np.dot(col, col_W) + self.b

        out = out.reshape(N, out_h, out_w, -1).transpose(0,3,1,2)  # 입력행렬의 모양으로 바꿔주기 위해서

        return out

x1 = np.arange(1470).reshape(10,3,7,7)
w1 = np.arange(750).reshape(10,3,5,5)
b1 = 1

conv = Convolution(w1,b1)
f = conv.forward(x1)
print('f.shape =',f.shape)
```

**결과

f.shape = (10, 10, 3, 3)

문제 222. 아래의 이미지를 손으로 최대풀링 하시오.

```
21 8 8 12
12 19 9 7
8 10  4  3
18 12 9 10
```

```
21 8 8 12
12 19 9 7
8 10  4  3
18 12 9 10
```

--->

```
21 12
18 10
```

문제 223. max_pooling 함수를 이용해서 4x4 행렬을 2x2 행렬로 변경 하시오.

```
def max_pooling(array):
    res = []
    a = array.flatten()
    for i in range(0,12,2):
        if i==4 or i==6:
            continue
        temp=np.array([a[i:i+2], a[i+4:i+6]])
        res.append(np.max(temp))
    res = np.array(res).reshape(2,2)
    return res

x = np.array([[21,8,8,12],
              [12,19,9,7],
              [8,10,4,3],
              [18,12,9,10]])
```

```
print(x,end='₩n₩n')

max_p = max_pooling(x)
print(max_p)
```

**결과

```
[[21  8  8 12]
 [12 19  9  7]
 [ 8 10  4  3]
 [18 12  9 10]]
[[21 12]
 [18 10]]
```

문제 224.	책 249 페이지의 Pooling 클래스 생성 하시오.

```
class Pooling :
    def __init__(self,pool_h,pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        N,C,H,W = x.shape
        out_h = int(1 +(H-self.pool_h)/self.stride)
        out_w = int(1 +(W-self.pool_w)/self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h * self.pool_w)

        out = np.max(col,axis=1)

        out = out.reshape(N, out_h, out_w, C).transpose(0,3,1,2)

        return out
```

문제 225.	mnist (28 x 28) 데이터가 convolution 층을 통과했을 때 출력이미지의 사이즈를 알아내시오. (필터 사이즈 : 5x5 , stride : 1, padding : 0)

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

----> P(패딩) = (OH -1) * S -H +FH / 2

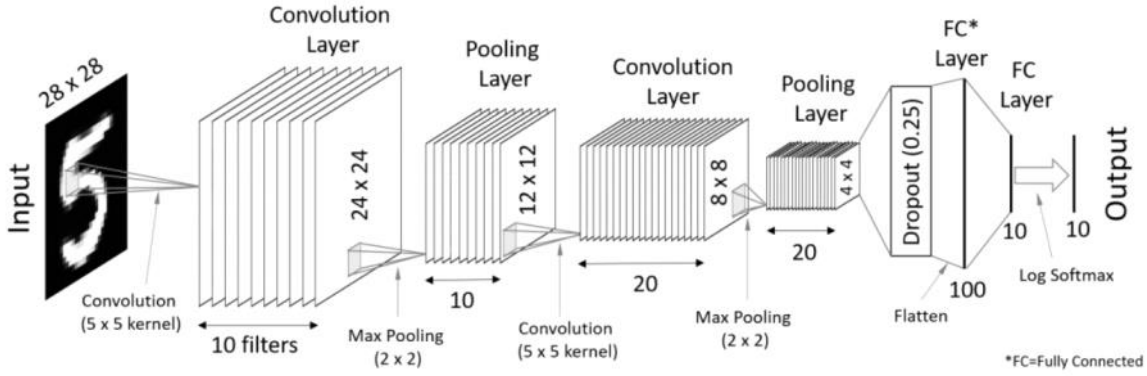
oh = (28 + 2*0 - 5)/ 1 + 1 = 24
ow = (28 + 2*0 - 5)/ 1 + 1 = 24

oh x ow = 24 x 24

문제 226.	입력 이미지 (24x24) 행렬이 풀링층을 통과했을 때 출력되는 이미지의 크기가 어떻게 되는가? (pool_h : 2, stride : 2)

11.5인데 int(11.5) 이므로 12가 된다. # 파이썬을 짝수로 반올림

문제 227. mnist 데이터를 cnn으로 구현했을때의 신경망을 그림으로 확인 하시오.



문제 228. 칠판에 그린 CNN 구현코드를 구현 하시오.

```
# coding: utf-8
import sys, os

sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import pickle
import numpy as np
from collections import OrderedDict
from common.layers import *
from common.gradient import numerical_gradient
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.trainer import Trainer


class SimpleConvNet:
    """단순한 합성곱 신경망

    conv - relu - pool - affine - relu - affine - softmax

    Parameters
    ----------
    input_size : 입력 크기 ( MNIST의 경우엔 784 )
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 ( e.g. [100, 100, 100] )
    output_size : 출력 크기 ( MNIST의 경우엔 10 )
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
    weight_init_std : 가중치의 표준편차 지정 ( e.g. 0.01 )
        'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
        'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
    """

    def __init__(self, input_dim=(1, 28, 28),
                 conv_param={'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
```

```
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2 * filter_pad) / filter_stride + 1
        pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size / 2))

        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size, output_size)

        self.params['b3'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
                                           conv_param['stride'], conv_param['pad'])
        self.layers['Relu1'] = Relu()
        self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])

        self.last_layer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    def loss(self, x, t):
        """손실 함수를 구한다.

        Parameters
        ----------
        x : 입력 데이터
        t : 정답 레이블
        """
        y = self.predict(x)
        return self.last_layer.forward(y, t)

    def accuracy(self, x, t, batch_size=100):
        if t.ndim != 1: t = np.argmax(t, axis=1)

        acc = 0.0

        for i in range(int(x.shape[0] / batch_size)):
            tx = x[i * batch_size:(i + 1) * batch_size]
            tt = t[i * batch_size:(i + 1) * batch_size]
            y = self.predict(tx)
```

```
        y = np.argmax(y, axis=1)
        acc += np.sum(y == tt)

    return acc / x.shape[0]



def numerical_gradient(self, x, t):
    """기울기를 구한다 ( 수치미분 ) .

    Parameters
    ----------
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -------
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1']、grads['W2']、... 각 층의 가중치
        grads['b1']、grads['b2']、... 각 층의 편향
    """
    loss_w = lambda w: self.loss(x, t)

    grads = {}
    for idx in (1, 2, 3):
        grads['W' + str(idx)] = numerical_gradient(loss_w, self.params['W' + str(idx)])
        grads['b' + str(idx)] = numerical_gradient(loss_w, self.params['b' + str(idx)])

    return grads



def gradient(self, x, t):
    """기울기를 구한다(오차역전파법).

    Parameters
    ----------
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -------
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1']、grads['W2']、... 각 층의 가중치
        grads['b1']、grads['b2']、... 각 층의 편향
    """
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.last_layer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
```

```
        for layer in layers:
            dout = layer.backward(dout)

        # 결과 저장
        grads = {}
        grads['W1'], grads['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
        grads['W2'], grads['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
        grads['W3'], grads['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

        return grads



    def save_params(self, file_name="params.pkl"):
        params = {}
        for key, val in self.params.items():
            params[key] = val
        with open(file_name, 'wb') as f:
            pickle.dump(params, f)



    def load_params(self, file_name="params.pkl"):
        with open(file_name, 'rb') as f:
            params = pickle.load(f)
        for key, val in params.items():
            self.params[key] = val

        for i, key in enumerate(['Conv1', 'Affine1', 'Affine2']):
            self.layers[key].W = self.params['W' + str(i + 1)]
            self.layers[key].b = self.params['b' + str(i + 1)]



# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)

# 시간이 오래 걸릴 경우 데이터를 줄인다.
# x_train, t_train = x_train[:5000], t_train[:5000]
# x_test, t_test = x_test[:1000], t_test[:1000]

max_epochs = 20

network = SimpleConvNet(input_dim=(1, 28, 28),
                        conv_param={'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)


# 매개변수 보존
network.save_params("params.pkl")
print("Saved Network Parameters!")


# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
```

```
train_size = x_train.shape[0] # 60000 개
batch_size = 100  # 미니배치 크기
learning_rate = 0.1
train_loss_list = []
train_acc_list = []
test_acc_list = []


# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득  # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]


    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)
    grad = network.gradient(x_batch, t_batch)
    # 매개변수 갱신

    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]


    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고
    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
        print(x_train.shape) # 60000,784
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개  16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc)  # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))


# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

문제 229.	위의 cnn 신경망에 가중치 초기화 선정인 Xavier를 적용해서 테스트 하시오.

- 1. Xavier 적용

(60000, 1, 28, 28)
train acc, test acc | 0.9863833333333333, 0.9833
(60000, 1, 28, 28)
train acc, test acc | 0.9877166666666667, 0.9843

문제 230.	위의 cnn을 이용한 3층 신경망에 배치정규화를 적용 하시오.

```
        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
                                           conv_param['stride'], conv_param['pad'])
        self.layers['BatchNorm1']=BatchNormalization(gamma=1.0, beta=0.)
        self.layers['Relu1'] = Relu()
        self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['BatchNorm2']=BatchNormalization(gamma=1.0, beta=0.)
        self.layers['Relu2'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])

# 층이 깊어질 수록 배치정규화의 필요성이 커진다. (가중치 값들이 퍼트려져 있는 것을 강제화 하기위해)
```

문제 231.	위의 cnn을 이용한 3층 신경망에 합성곱층과 pooling 층을 하나 더 추가하시오.

conv --> relu --> pooling --> conv --> relu --> pooling --> affine1 --> relu --> affine2 -->softmax