인공지능시스템

CNN Modules

Convolution

- Convolution(합성곱) 연산 :
  이미지와 필터(Filter 또는 Kernel)와의 합성곱을 이용하여 이미지의 인접한 공간적/지역적 특징 정보를 추출할 수 있음

- 합성곱 예시

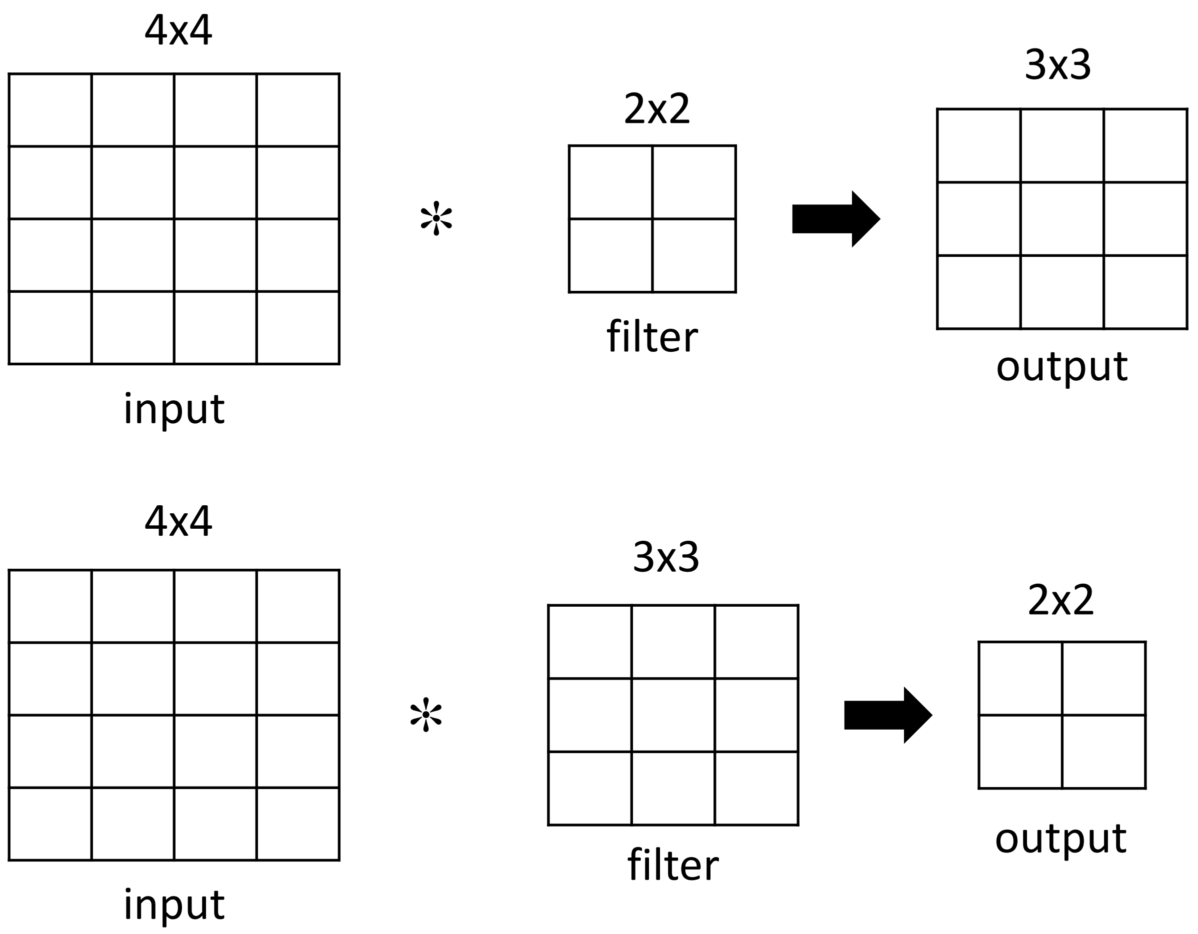
= 1*1+2*2+4*3+5*4

= 2*1+3*2+5*3+6*4

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 37 & 47 \\ 67 & 77 \end{bmatrix}$$

Conv filter - size

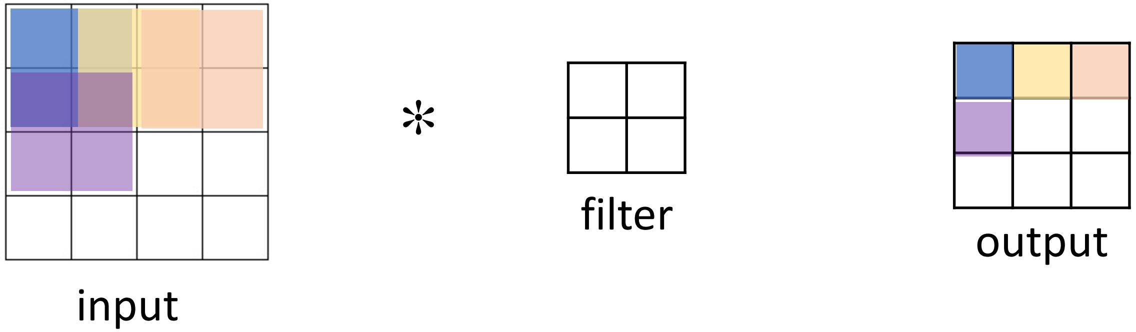
- Input size & Filter size & Output size



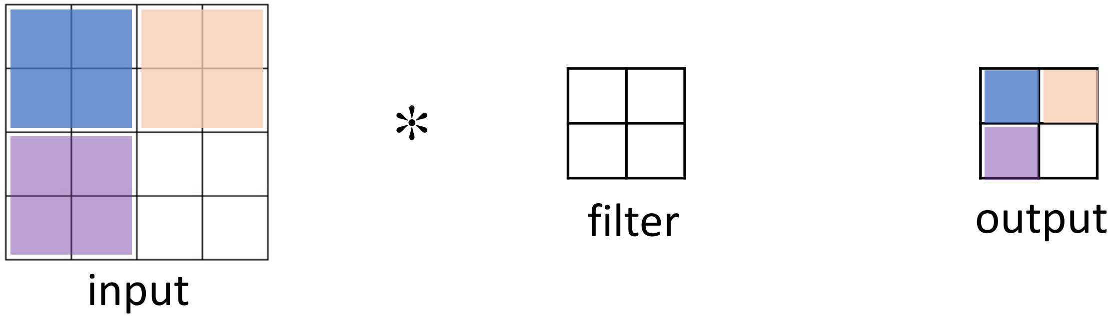
Conv filter - stride

- Stride: 합성곱을 계산하는 간격

stride = 1



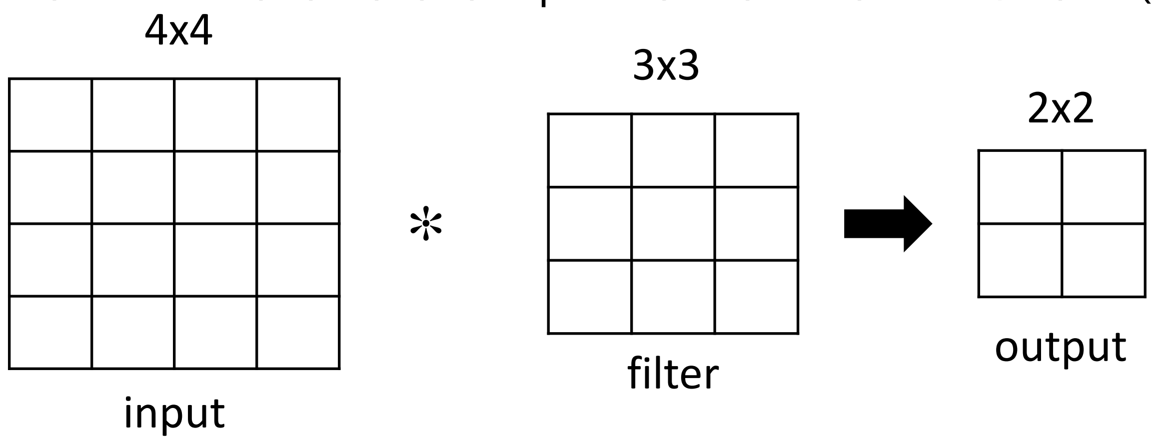
stride = 2

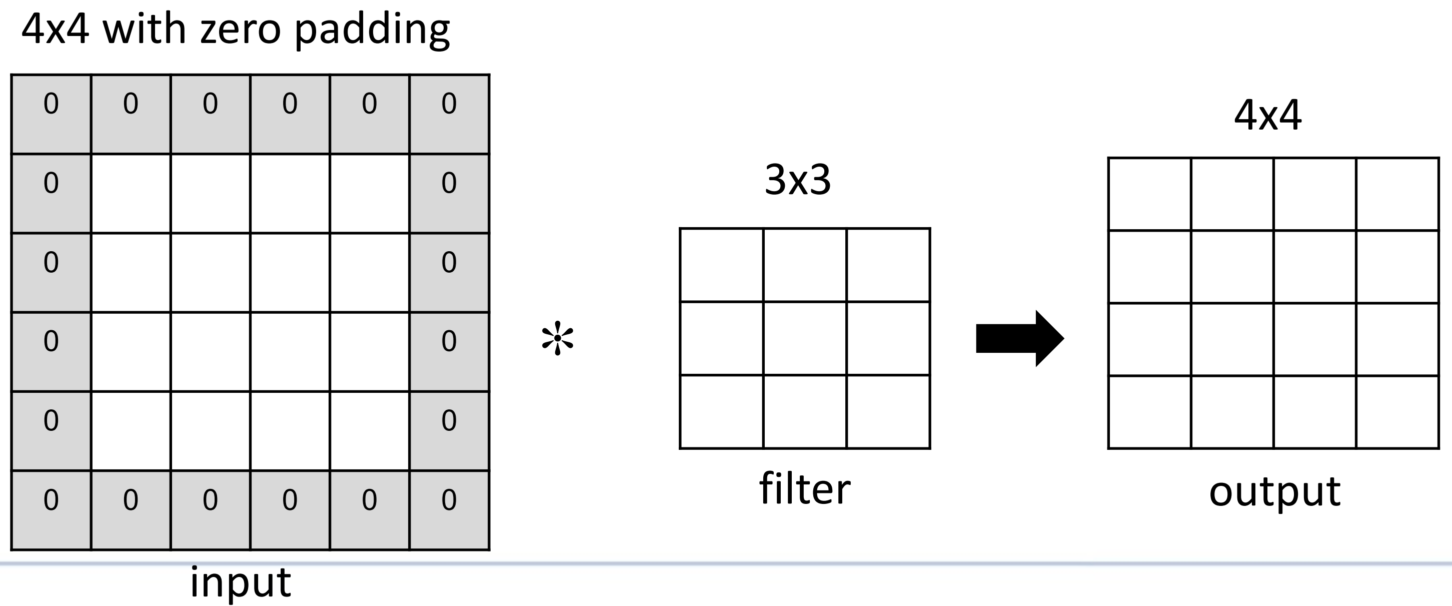


Conv filter - padding

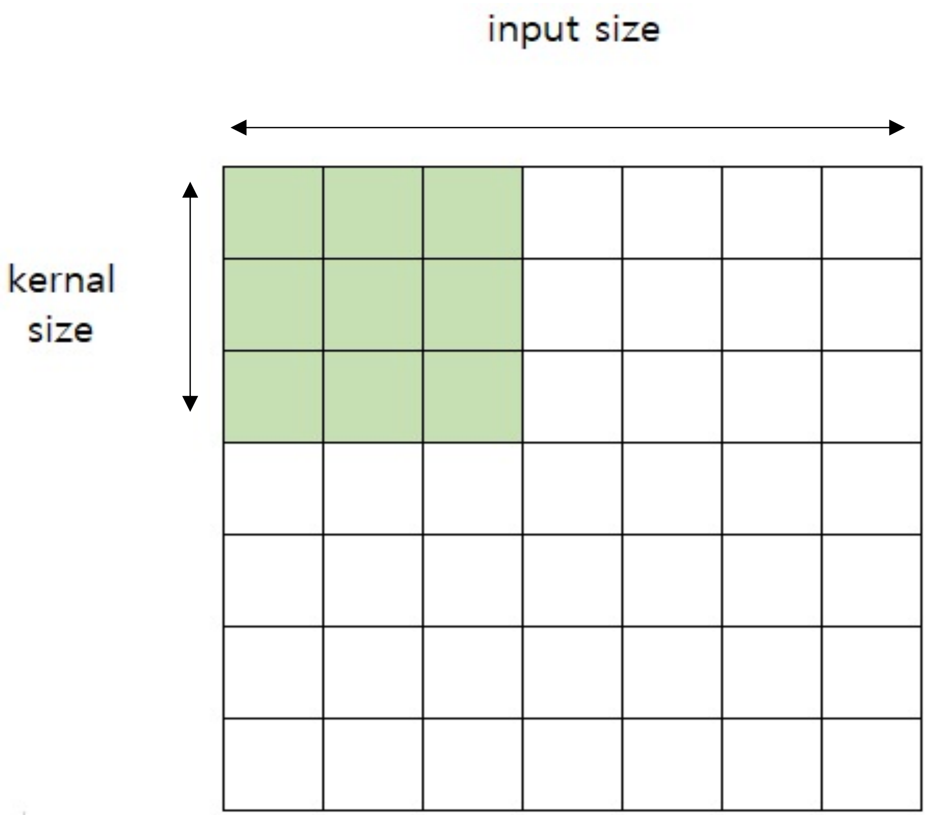
- Padding: 합성곱 연산을 수행하면 input에 비해서 output의 크기가 줄어들게 됨

  이를 보상하기 위하여 input 바깥부분에 0을 추가함 (zero padding)





Conv filter – Output size



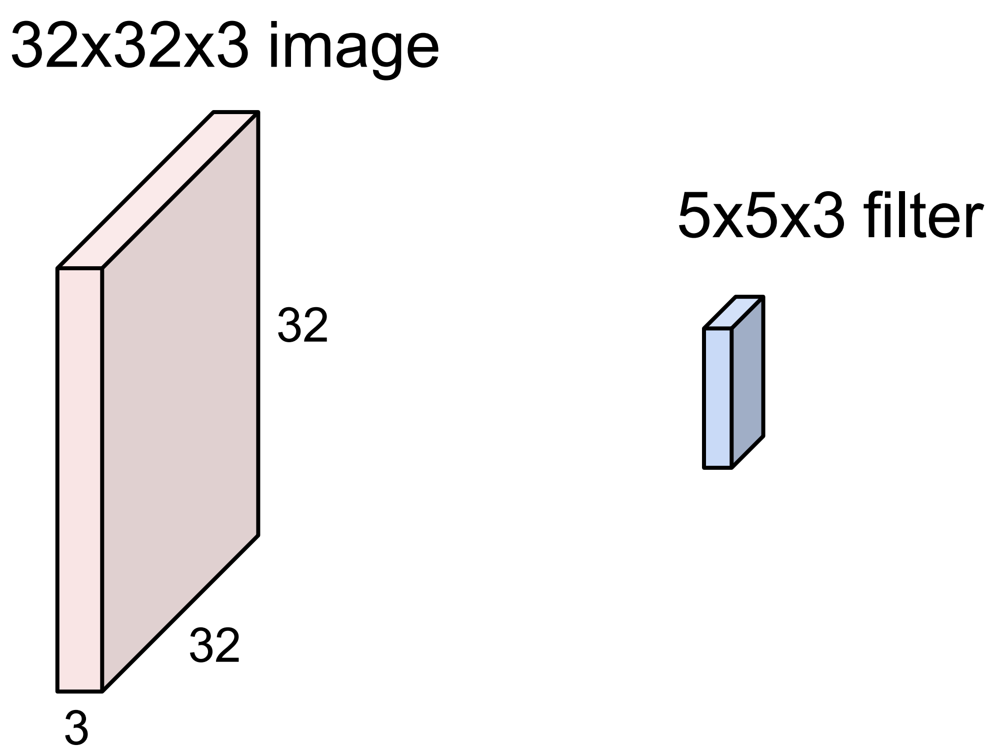
Quiz1) input_size=7,
kernel_size=3, stride=1 → 5x5

Quiz2) input_size=7,
kernel_size=3, stride=2 → 3x3

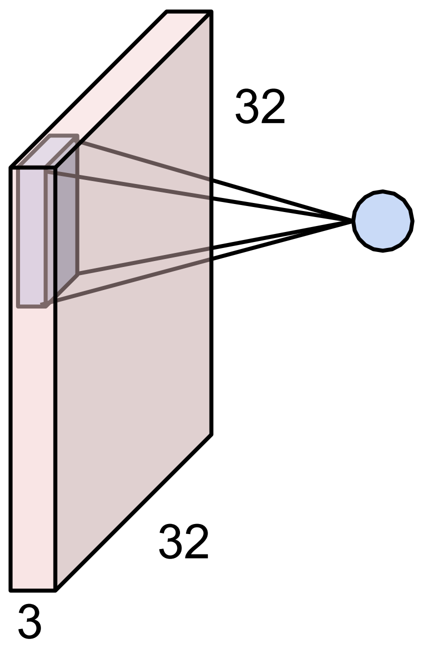
- output size = (input_size – kernel_size)/stride +1

  ※ input_size: input에 padding을 적용한 후의 size

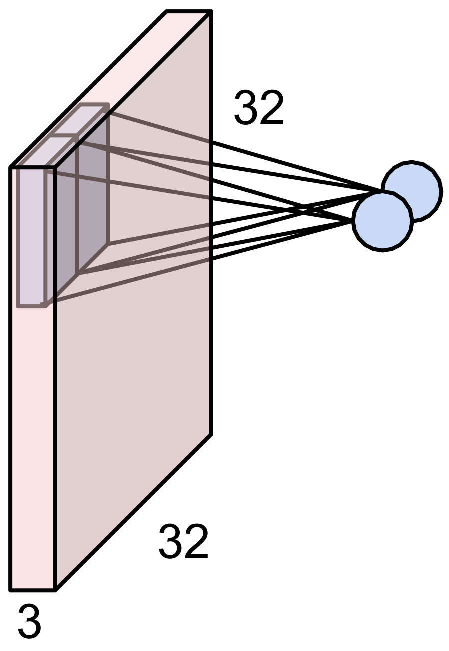
Convolution operations



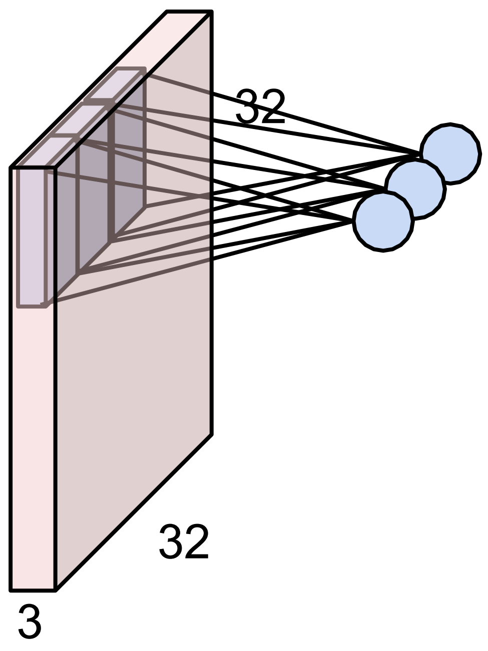
Convolution operations



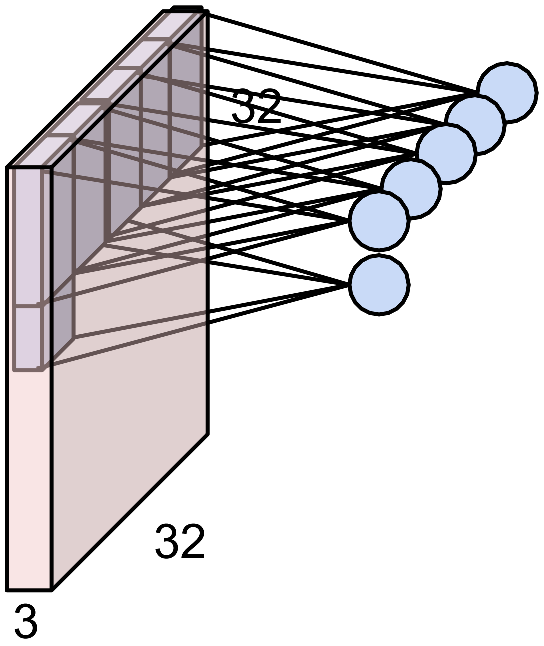
Convolution operations



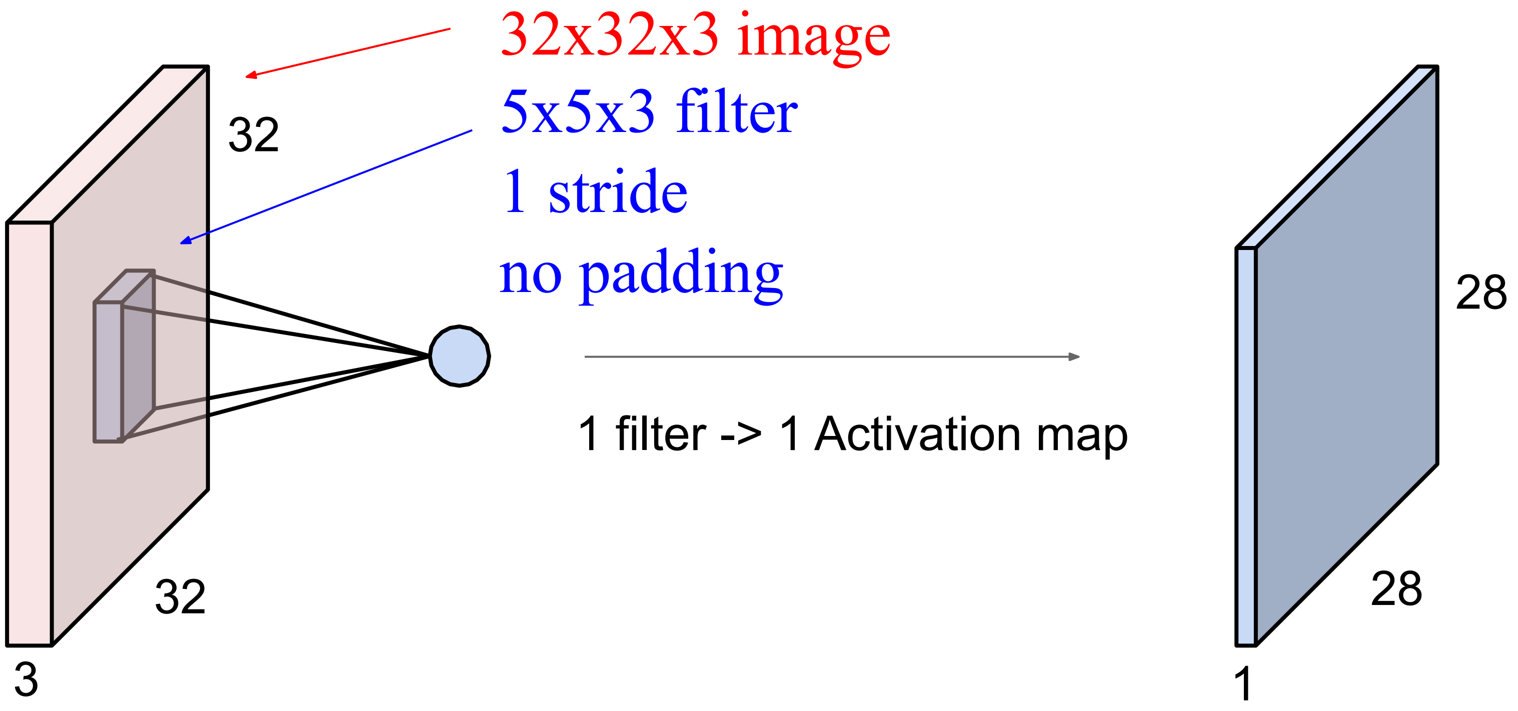
Convolution operations



Convolution operations

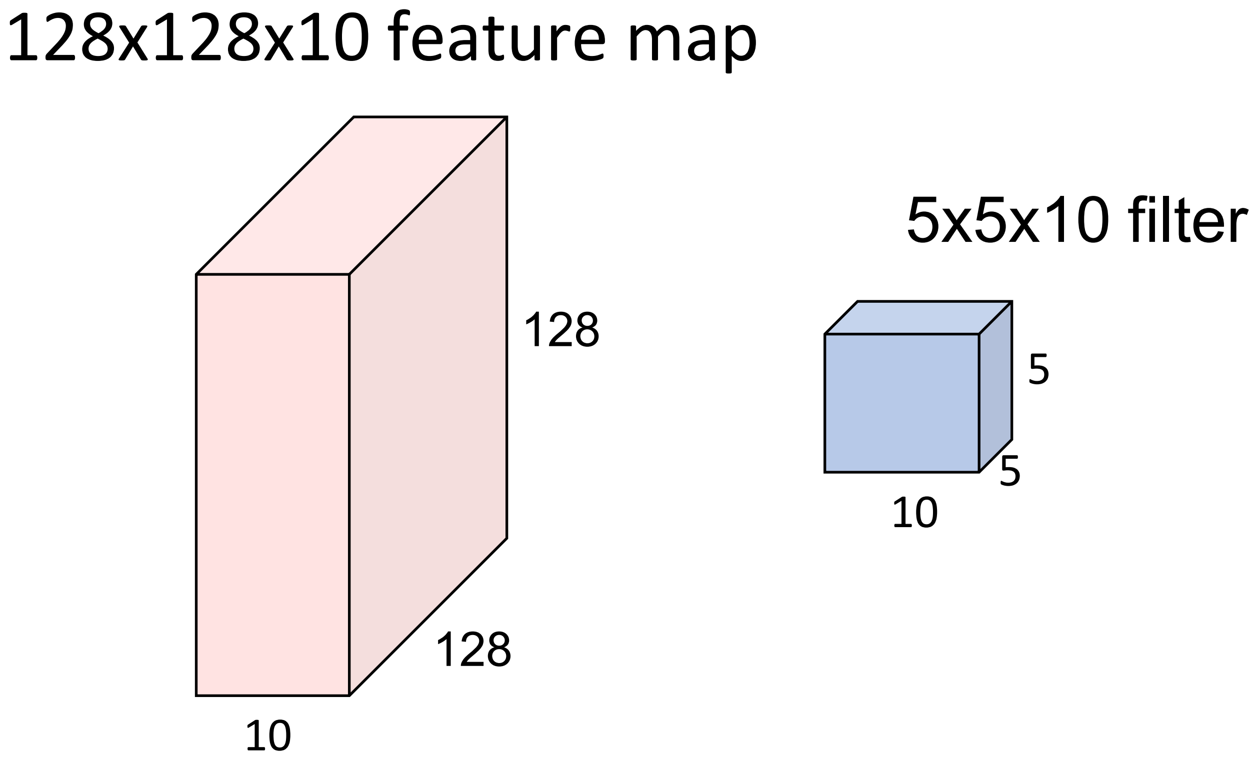


Convolution operations



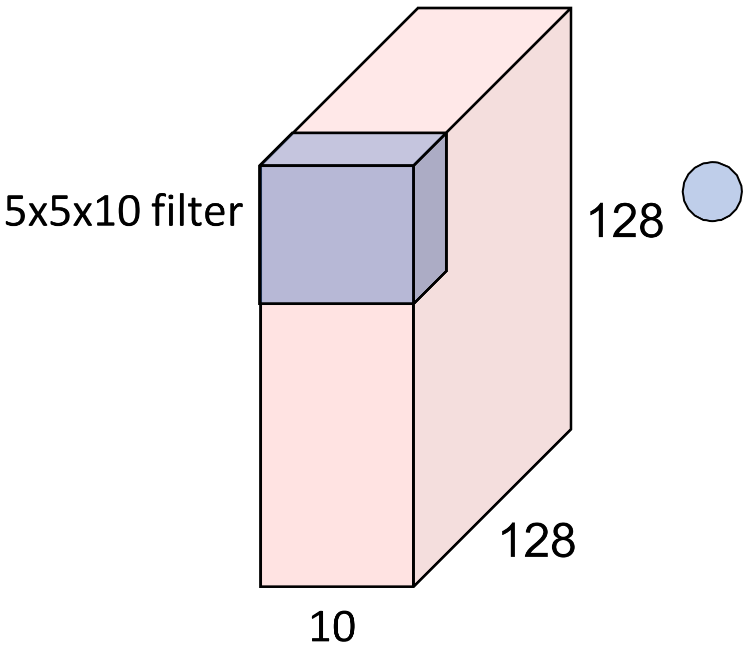
Activation map

Convolution operations



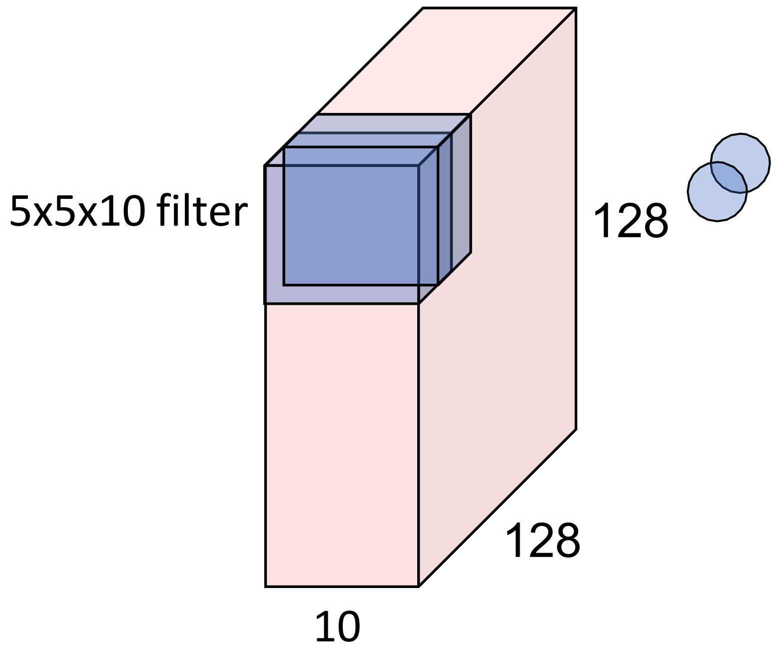
Convolution operations

128x128x10 feature map



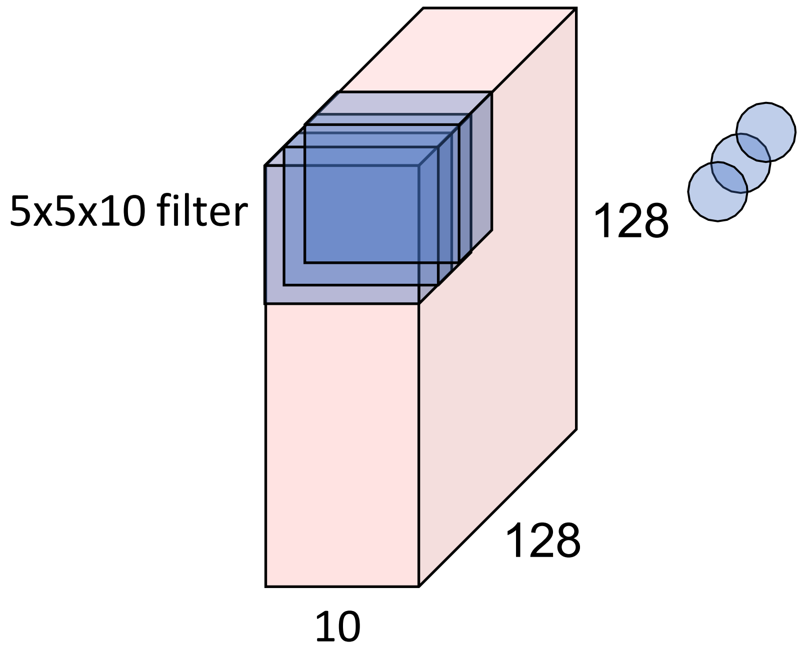
Convolution operations

128x128x10 feature map



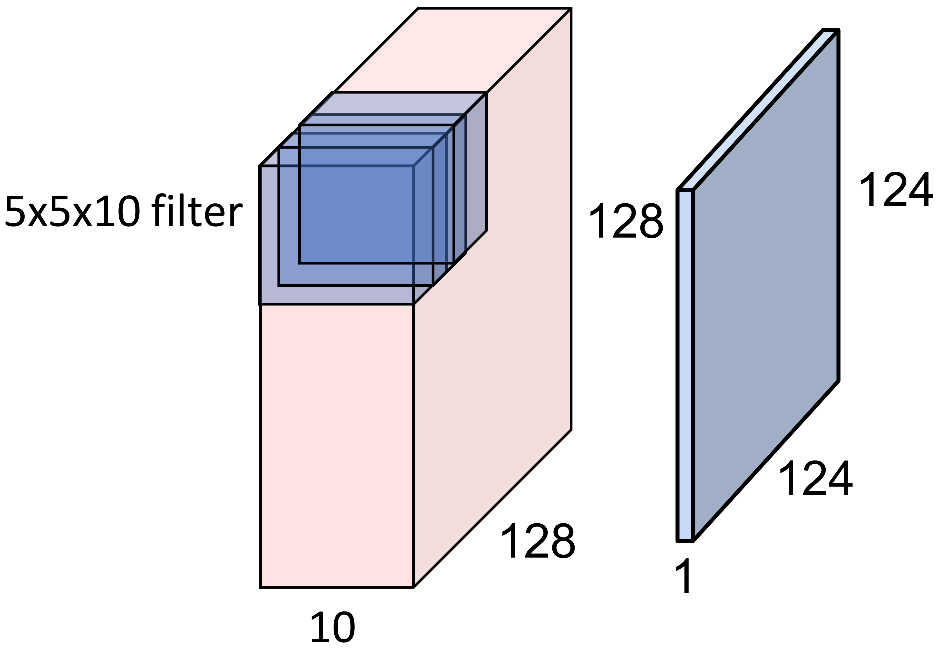
Convolution operations

128x128x10 feature map

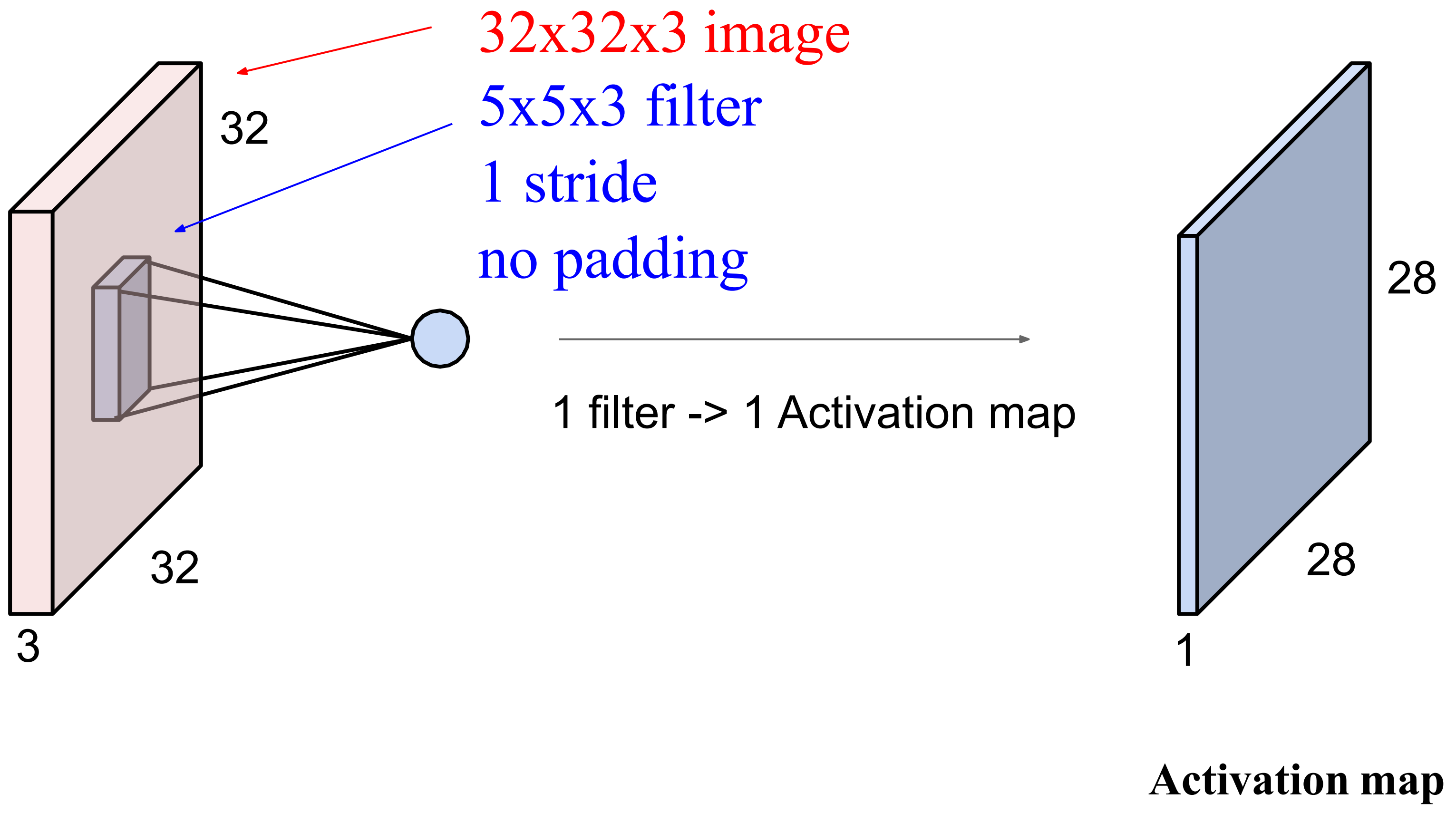


Convolution operations

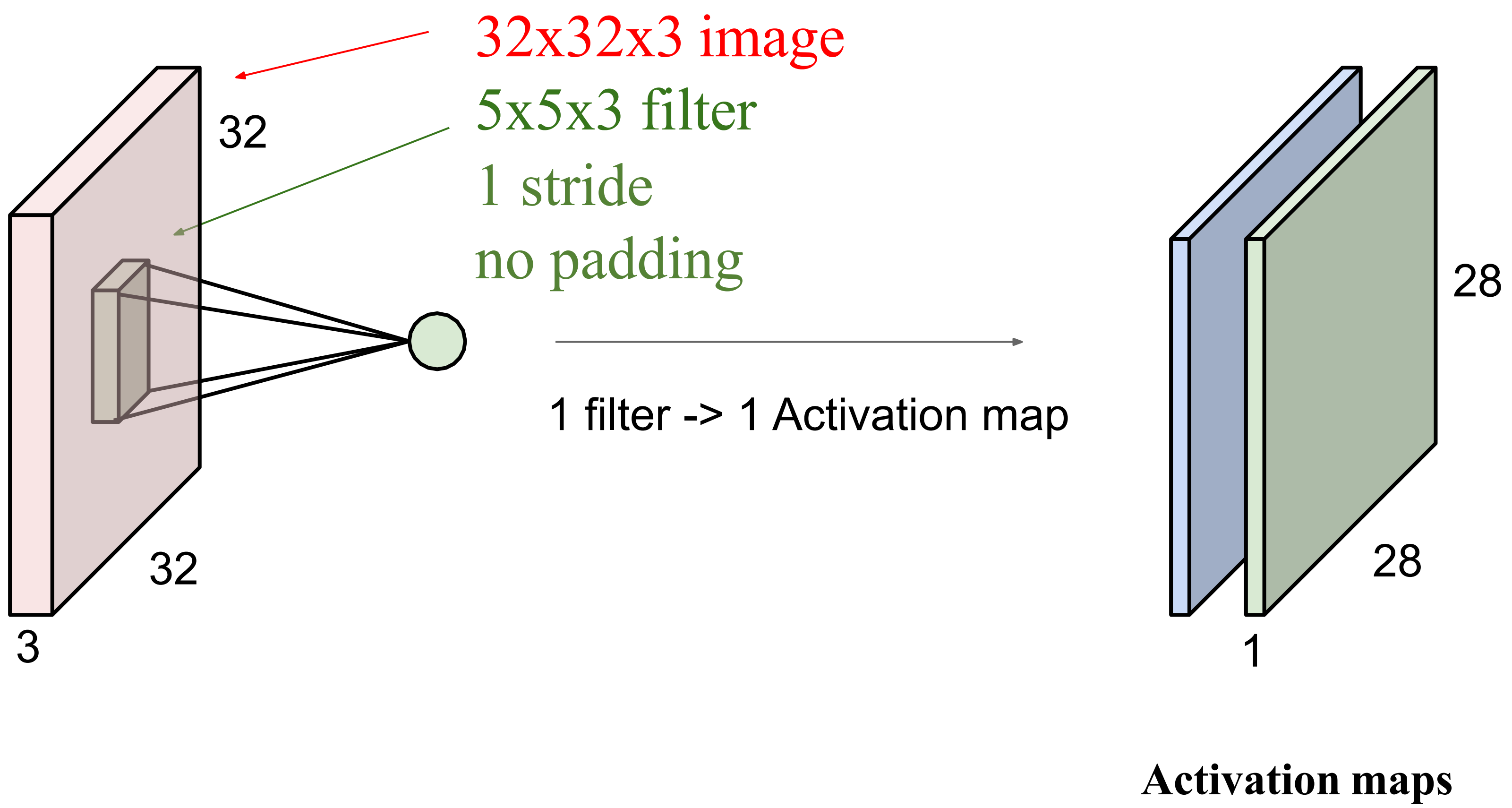
128x128x10 feature map



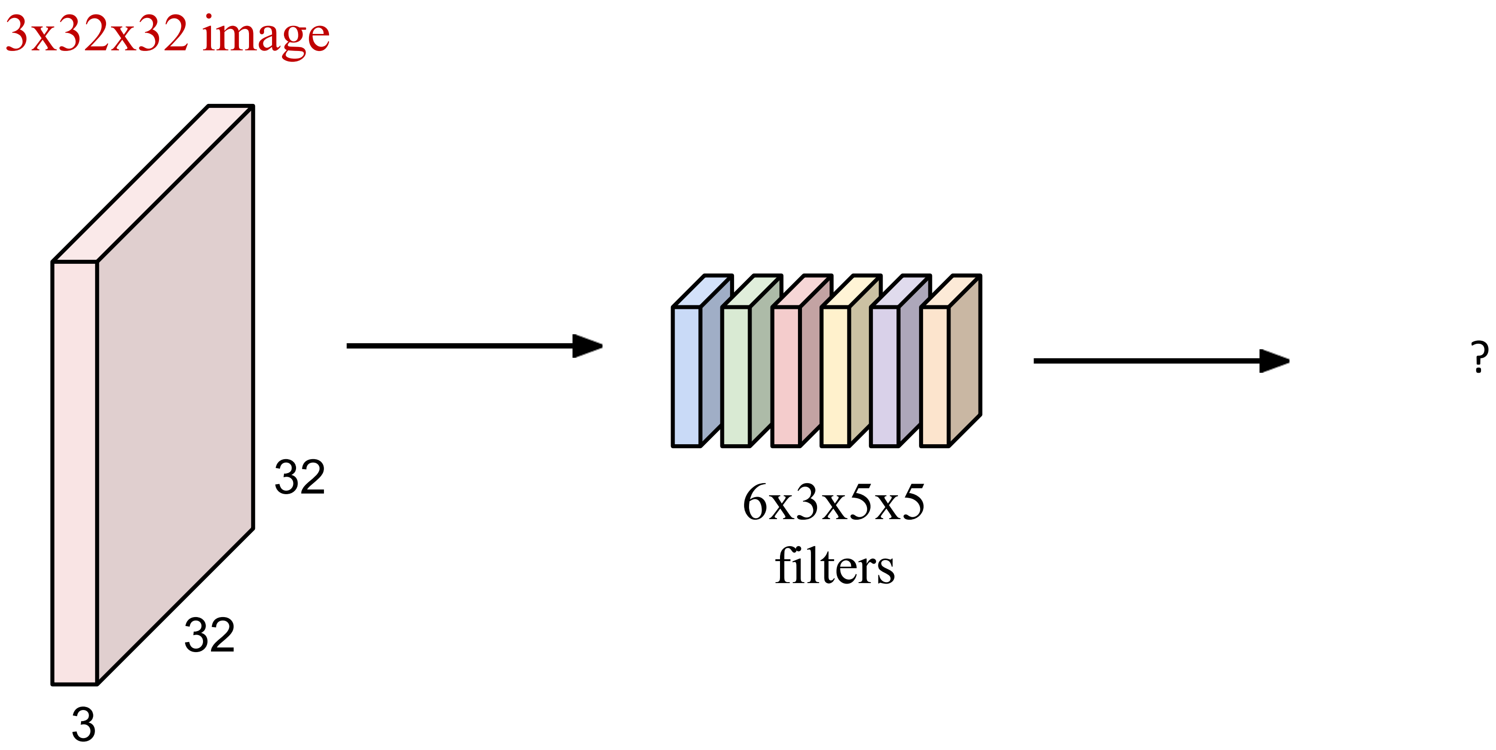
Convolution operations



Convolution operations



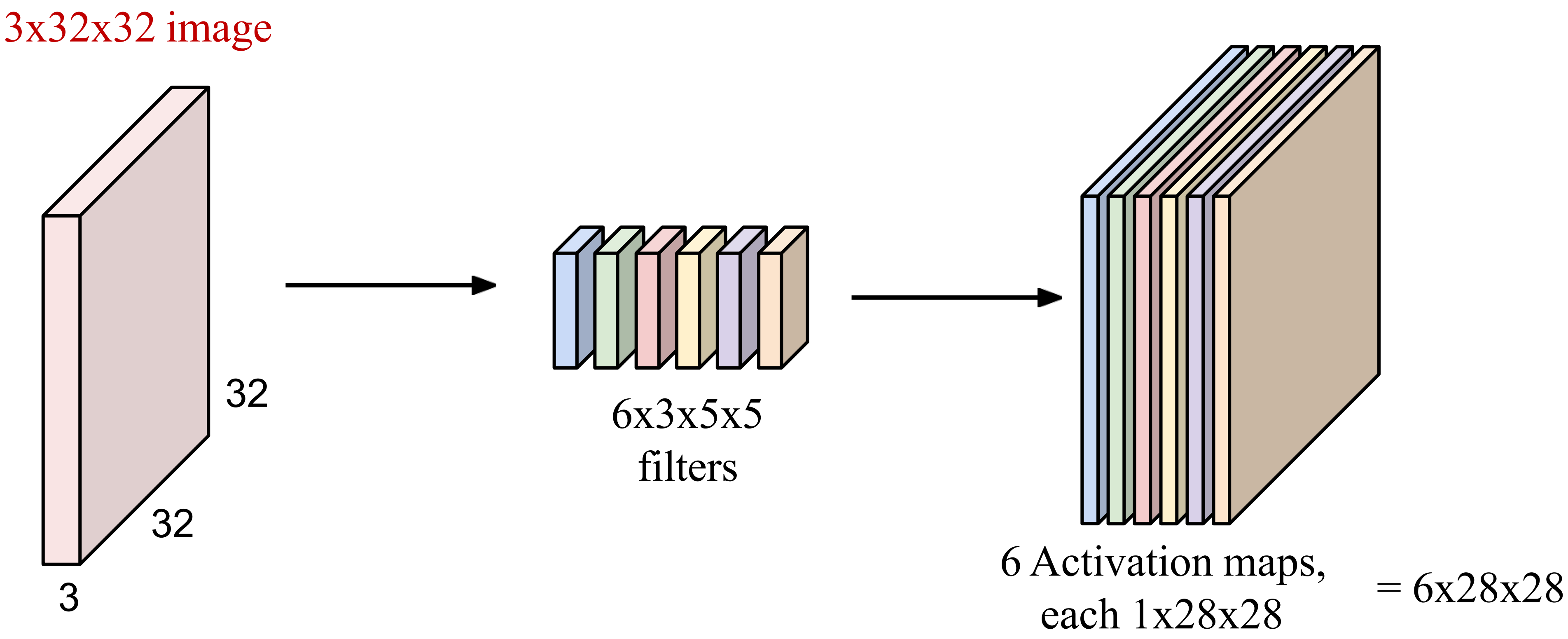
Convolution operations



Pytorch Convention

(#Batch, #Channel, H, W)

Convolution operations



Pytorch Convention

(#Batch, #Channel, H, W)

2d Conv layer -Pytorch

CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

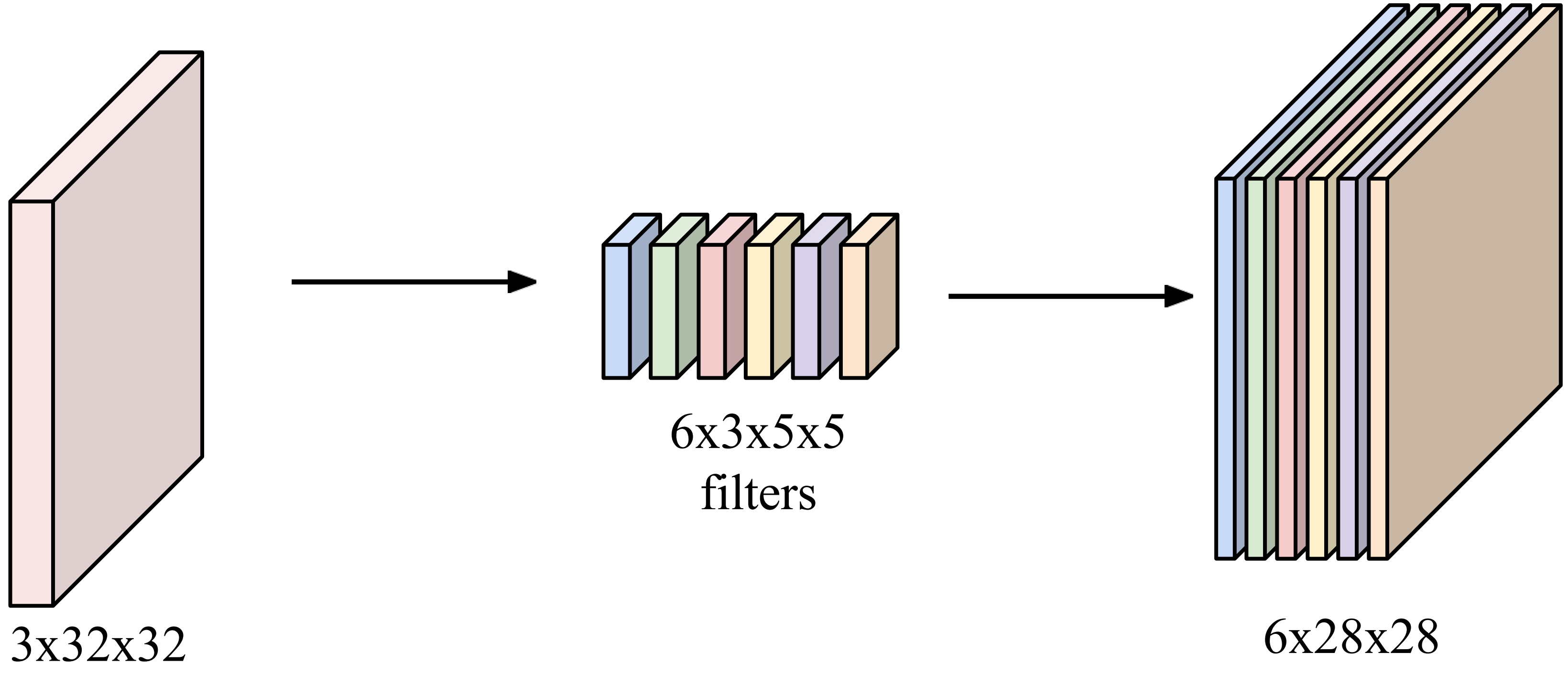
```
nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3,padding=0, stride=1)
```

2d Conv layer -Pytorch

CONV2D

CLASS torch.nn.Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None*) [SOURCE]

```
nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3,padding=0, stride=1)
```



CNN – Pooling Pytorch

MAXPOOL2D

CLASS `torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)` [SOURCE]

AVGPOOL2D

CLASS `torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)` [SOURCE]

Input_preprocess

- torch.permute() 를 이용하여
  입력 데이터 Tensor의 순서를 BHWC→BCHW 순서로 바꾸어 준다.

```
def input_preprocess(x , hflip=True):
    x = x.float()                     # input data (integer 0 ~ 255)
    x = x/255.0                       # 0~1사이의 float로 바꿔주기
    if hflip:                         # hfip 이 True이면 수행
        half = int(x.shape[0]/2)  # mini-batch의 절반의 크기를 계산
        x[0:half, :] = torch.flip(x[0:half, :], dims=[2])  # 앞에서부터 절반까지는 수평뒤집기(좌우반전)

    x = torch.permute(x, dims=(0,3,1,2))     # 차원의 순서를 바꿔주기 B,H,W,C -> B,C,H,W
    return x
```

Convolutional Neural Network

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3, 3), padding=0, stride=(1, 1))
        self.conv2 = nn.Conv2d(32, 64, (4, 4))
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(64*6*6, 10)

    def forward(self, x):       # 3x32x32
        x = self.conv1(x)       # 32x30x30
        x = F.relu(x)
        x = F.max_pool2d(x, 2)  # 32x15x15

        x = self.conv2(x)       # 64x12x12
        x = F.relu(x)
        x = F.max_pool2d(x, 2)  # 64x6x6

        x = torch.flatten(x, start_dim=1) # Fully Connected Layer를 통과할 수 있도록 쭉 펼침
        x = self.dropout(x)
        x = self.fc1(x)

        return x
```

Convolutional Neural Network

```python
class CNNet(nn.Module):
    def __init__(self):
        super(CNNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3,padding=0, stride=1)
        self.conv2 = nn.Conv2d(32,64,4)
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(64*6*6, 10)

    def forward(self, x):        # 3x32x32
        x = self.conv1(x)        # 32x30x30
        x = F.relu(x)
        x = F.max_pool2d(x, 2)   # 32x15x15

        x = F.relu(x)
        x = F.max_pool2d(x, 2)   # 64x6x6

        x = torch.flatten(x, 1)  # fully connected layer를 통과할 수 있도록 쭉 펼침
        x = self.dropout(x)
        x = self.fc1(x)

        return x
```

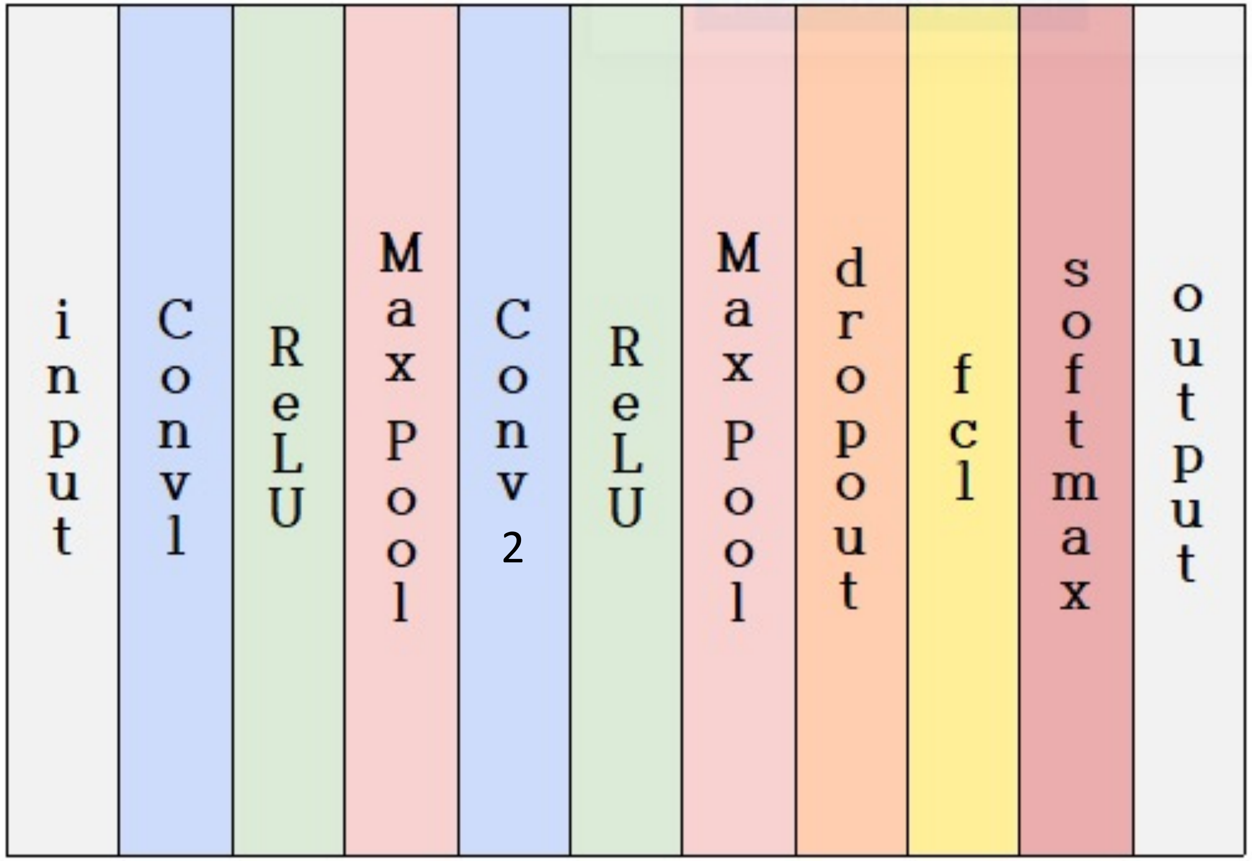
- nn module 의 Conv2d 로 conv1, conv2를 각각 정의함
  (channels size에 주의)
- Dropout 은 학습되는 파라미터가 아니라 그냥 p 확률로 feature의 일부를 off한다.
- Linear layer의 input 을 잘 계산할 것!
  (입력으로 들어오는 feature를 vectorize 한 크기를 계산)

Convolutional Neural Network

- forward는 레고 블럭처럼 이어 붙인다.
- `import torch.nn.functional as F`
  F에서 relu와 max_pool2d를 사용할 수 있음
- torch.flatten으로 tensor를 vectorize 한다.

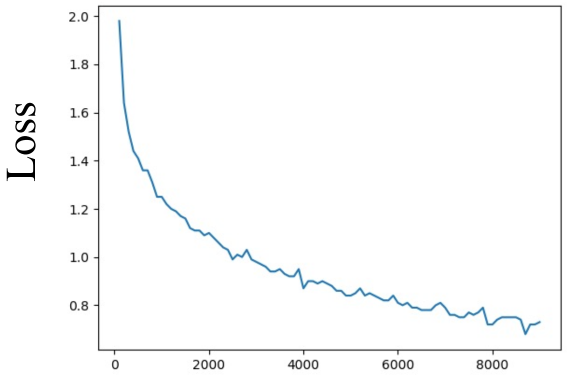
```python
def forward(self, x):        # 3x32x32
    x = self.conv1(x)        # 32x30x30
    x = F.relu(x)
    x = F.max_pool2d(x, 2)   # 32x15x15

    x = self.conv2(x)        # 64x12x12
    x = F.relu(x)
    x = F.max_pool2d(x, 2)   # 64x6x6

    x = torch.flatten(x, 1) # Fully Connected Layer를 통과할 수 있도록 쭉 펼침
    x = self.dropout(x)
    x = self.fc1(x)

    return x
```

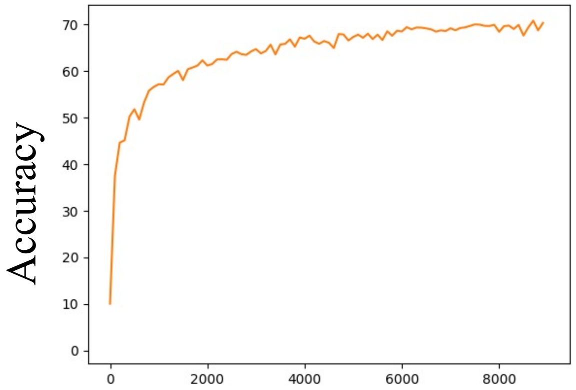
Convolutional Neural Network



학습 결과

- Option1 (Adam)

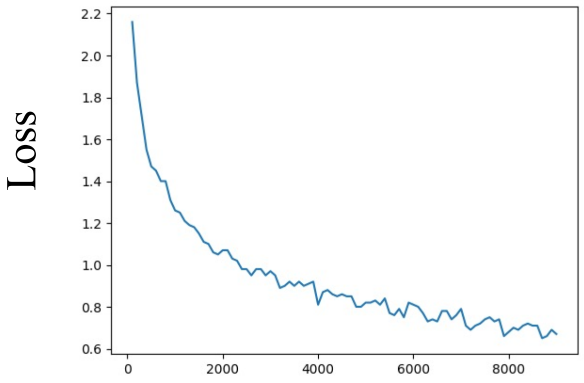


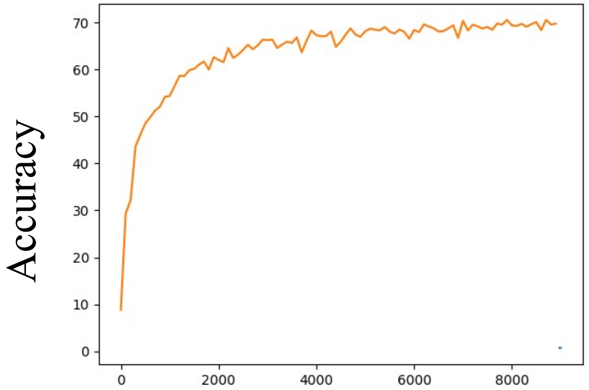


iterations

- Accuracy: 70.51%

- Option2 (SGD + StepLR)





iterations

- Accuracy: 70.16%

thank you

본 과제(결과물)는 교육부와 한국연구재단의 재원으로 지원을 받아 수행된
디지털신기술인재양성 혁신공유대학사업의 연구결과입니다.