Convolutional Neural Network Architecture

2023-2 KUBIG 방학세션
DL

Category

1. Convolution layer
2. Max Pooling layer
3. AlexNet- Top Network from ImageNet Classification
4. Q & A

0. Paper Review

0. Paper Review

PyTorch

Normalization in Pytorch

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

nn.BatchNorm2d(num_features, eps=1e-5, affine=True)

```
# With Learnable Parameters
m = nn.BatchNorm2d(100)
# Without Learnable Parameters
m = nn.BatchNorm2d(100, affine=False)
input = torch.randn(20, 100, 35, 45)
output = m(input)
```

nn.BatchNorm3d(num_features, ,,,)

```
# With Learnable Parameters
m = nn.BatchNorm3d(100)
# Without Learnable Parameters
m = nn.BatchNorm3d(100, affine=False)
input = torch.randn(20, 100, 35, 45, 10)
output = m(input)
```

nn.LayerNorm(normalized_shape, eps=1e-5, elementwise_affine=True)

```
# NLP Example
batch, sentence_length, embedding_dim = 20, 5, 10
embedding = torch.randn(batch, sentence_length, embedding_dim)
layer_norm = nn.LayerNorm(embedding_dim)
# Activate module
layer_norm(embedding)
```

0. CNNs are Everywhere

[Ciresan et al., 2013] [Sermanet et al., 2011]

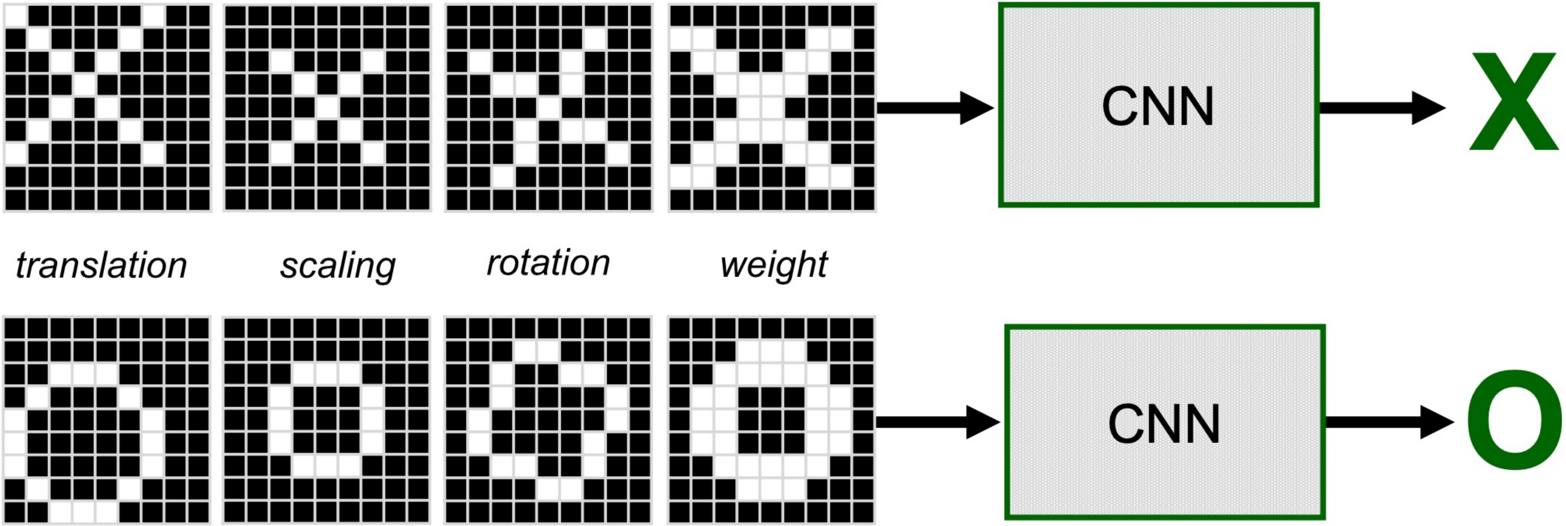
[Citation] Lecture 13: Convolutional Neural Networks -1, 데이터 학습과 지능, 정태수, 고려대학교

0. CNNs are Everywhere

[Citation] Lecture 13: Convolutional Neural Networks -1, 데이터 학습과 지능, 정태수, 고려대학교

0. CNNs are Everywhere

▶ Trickier cases



[Citation] Lecture 13: Convolutional Neural Networks -1, 데이터 학습과 지능, 정태수, 고려대학교

0. CNNs are Everywhere

▶ Computers are literal

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

≠

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

[Citation] Lecture 13: Convolutional Neural Networks -1, 데이터 학습과 지능, 정태수, 고려대학교

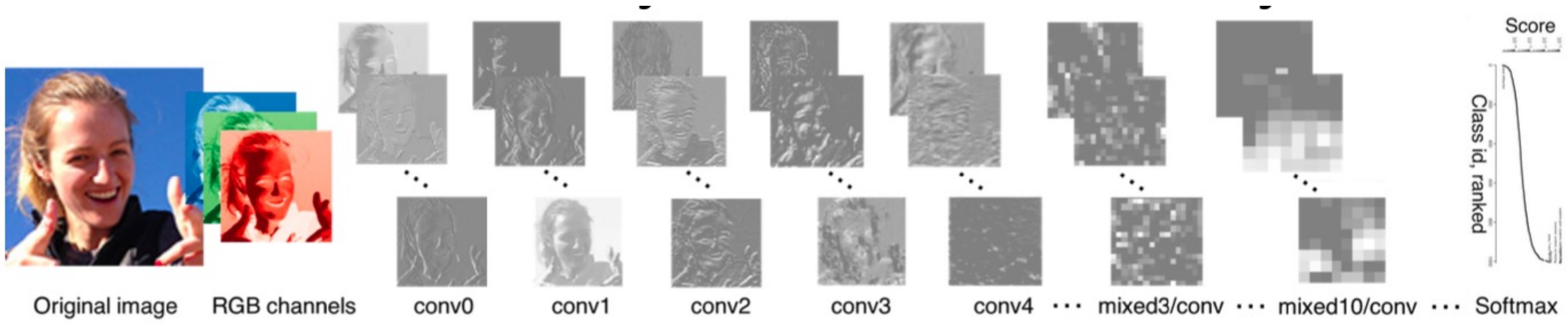
1. Convolution Layer

* Recap: FC layer

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

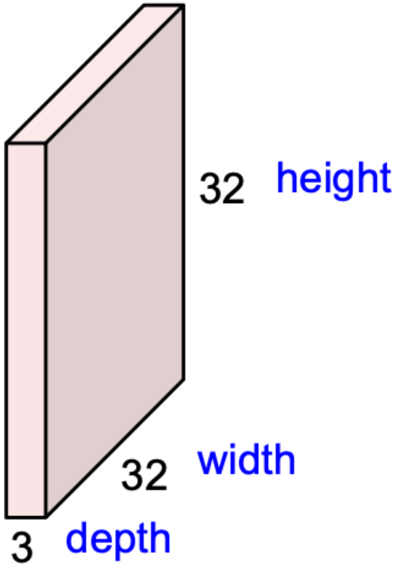
* Convolution = " 합성곱"
* Similar to fully connected layer but somewhat different
* Use "filter" to "convolve" input data



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

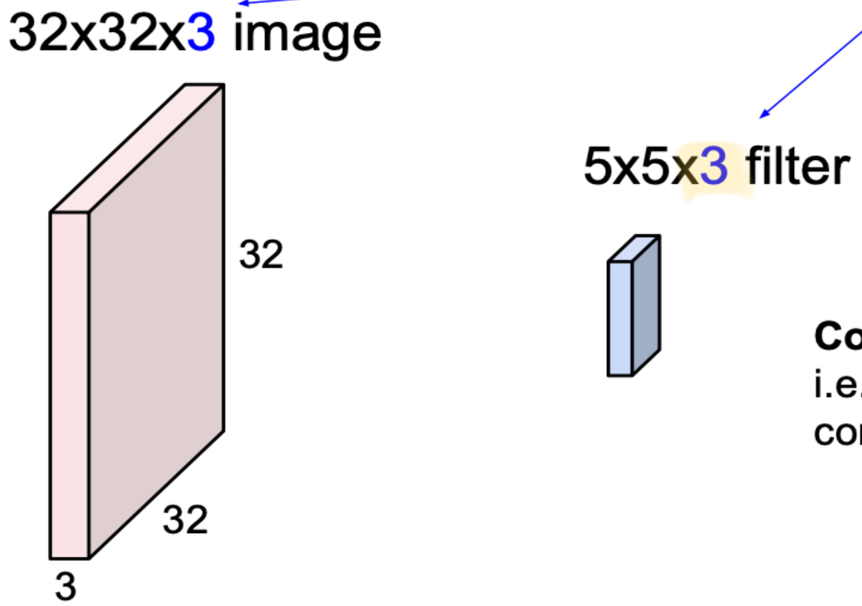
32x32x3 image -> preserve spatial structure



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

Filters always extend the full depth of the input volume



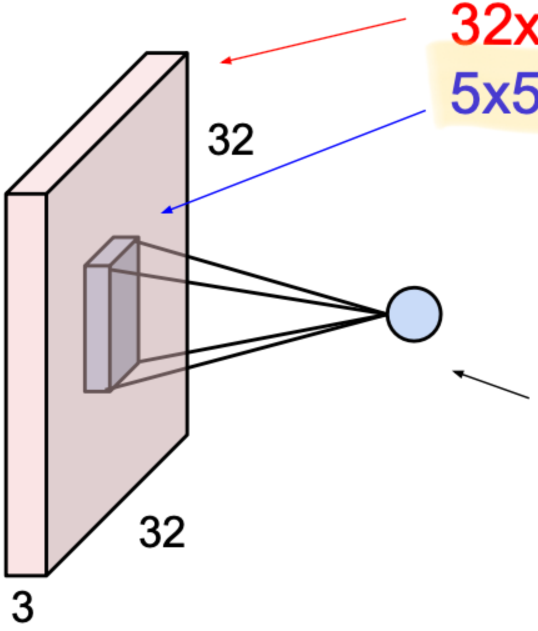
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

32x32x3 image

5x5x3 filter w

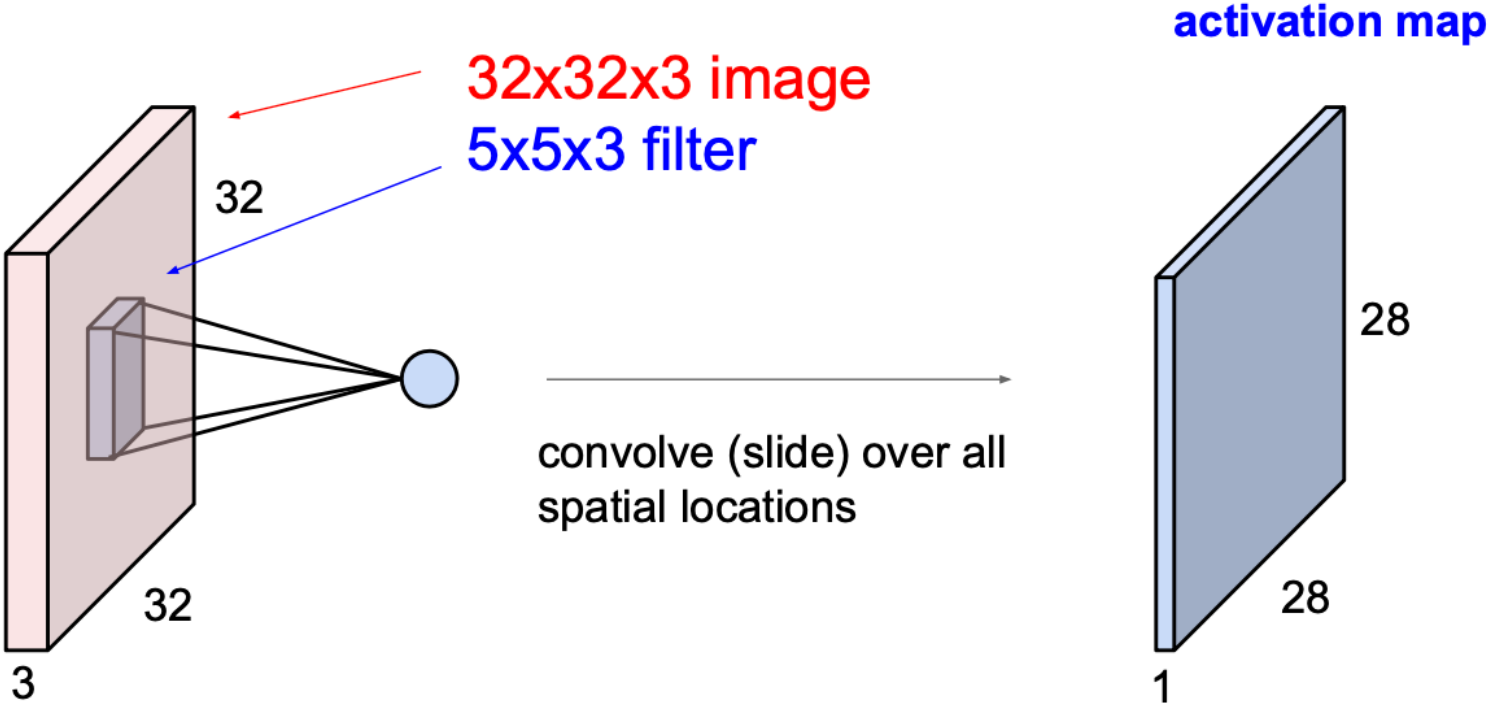


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + $\boxed{\text{bias}}$)

$$w^T x + b$$

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

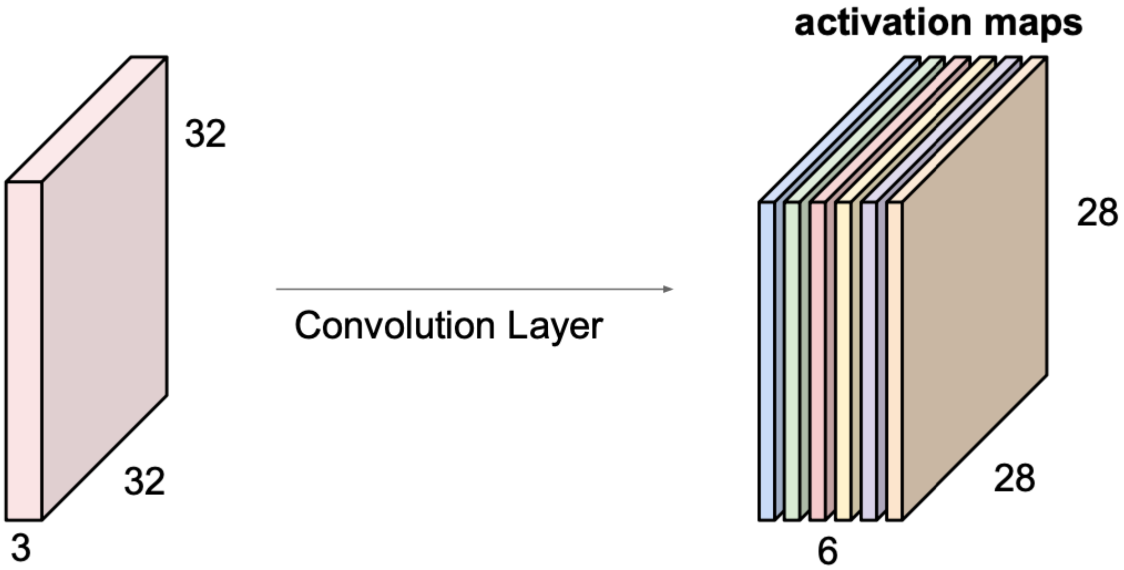
1. Convolution Layer



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

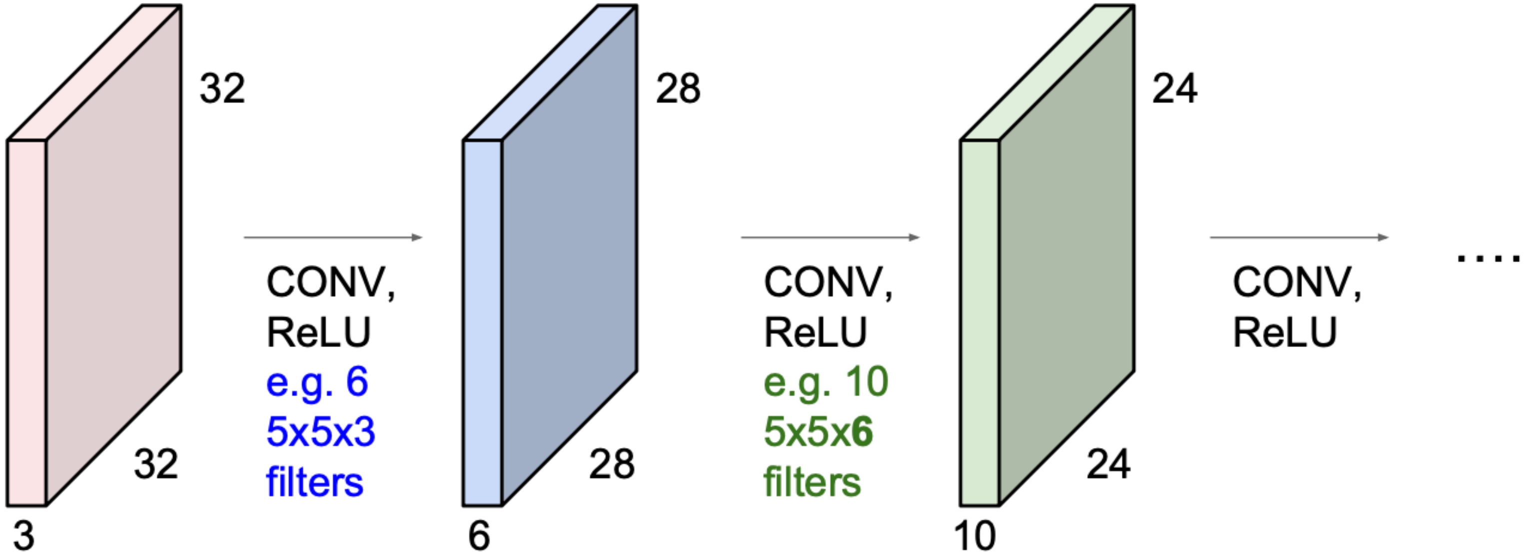


We stack these up to get a "new image" of size 28x28x6!

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

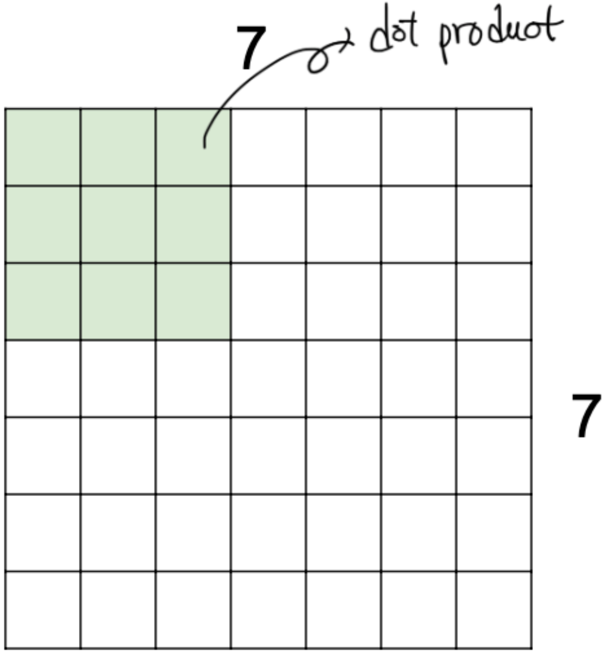
1. Convolution Layer

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

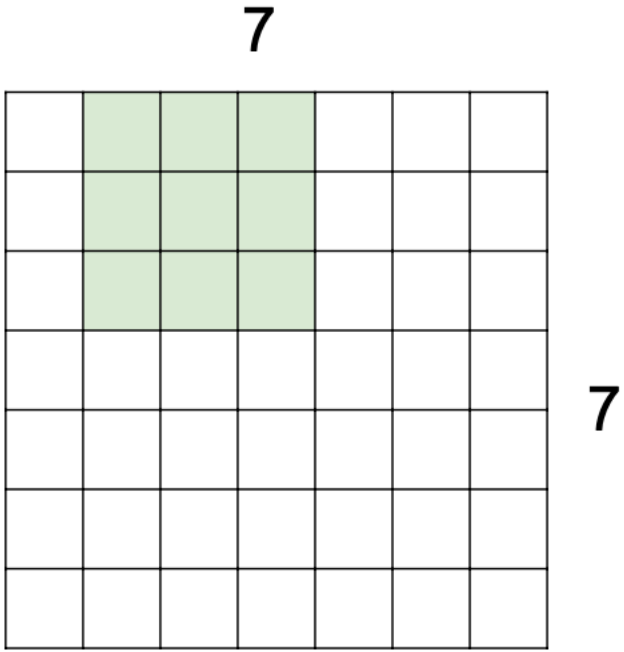
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

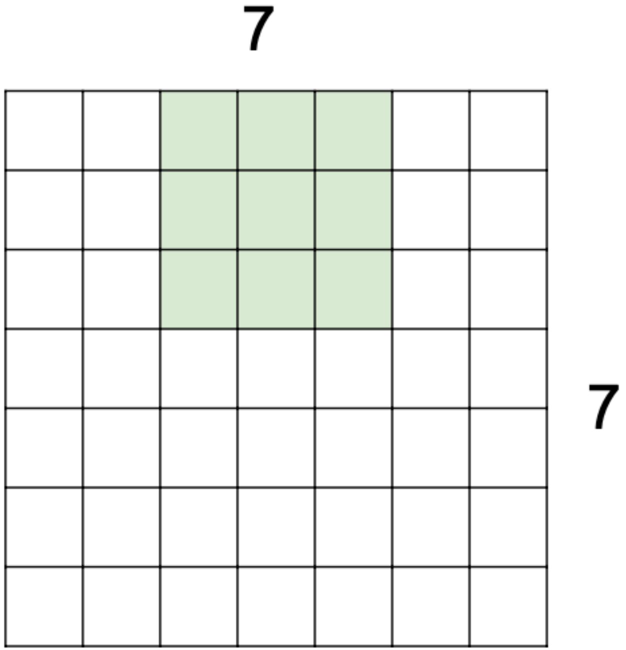
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

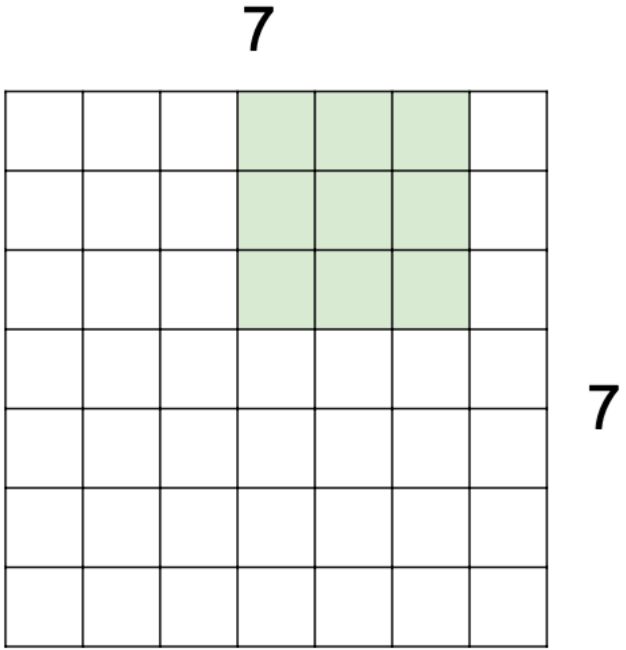
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li &
Justin Johnson & Serena Yeung

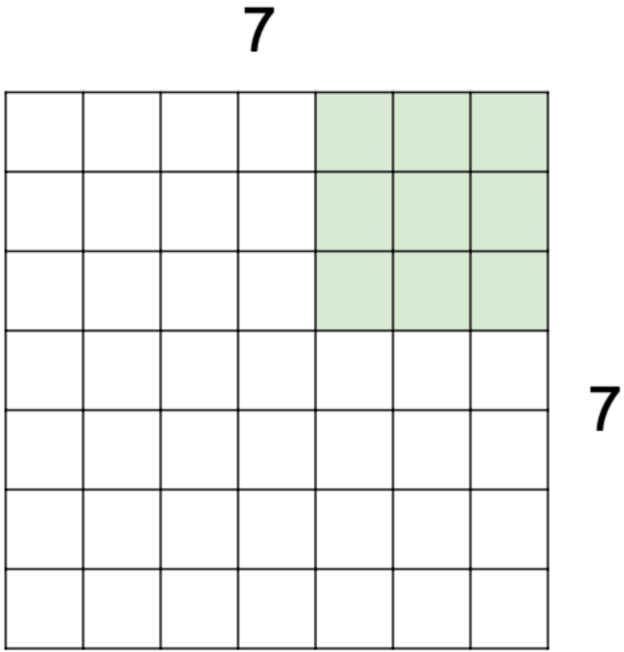
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

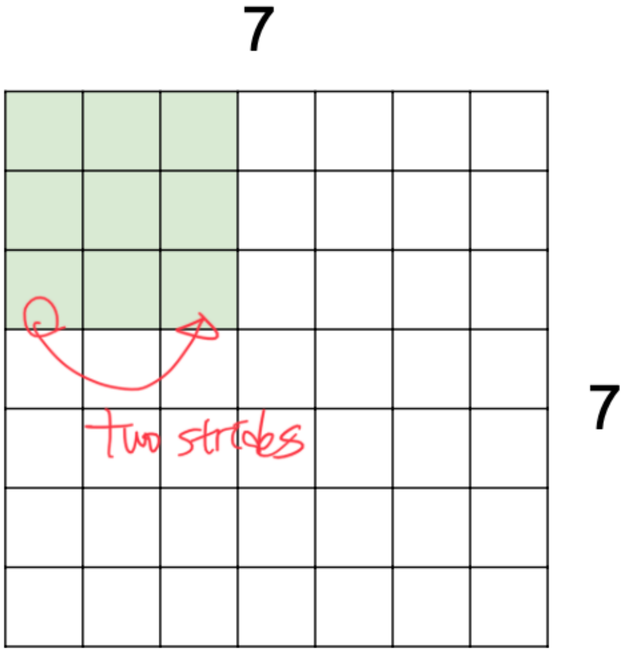


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

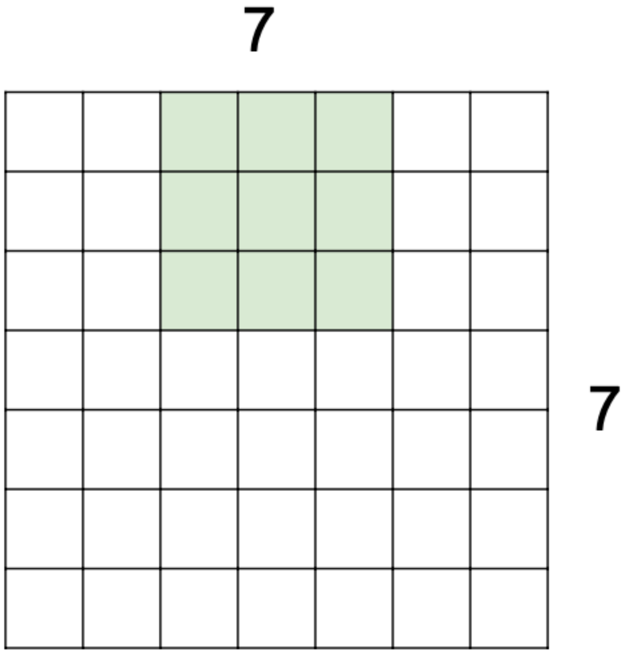
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

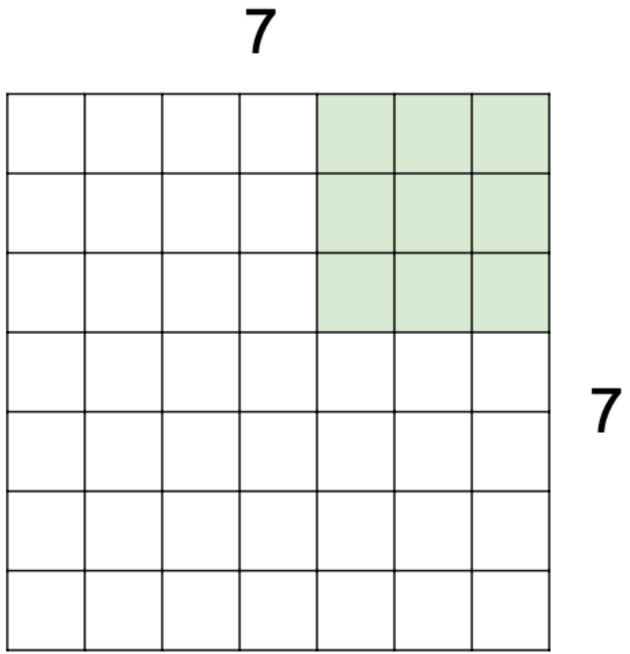
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

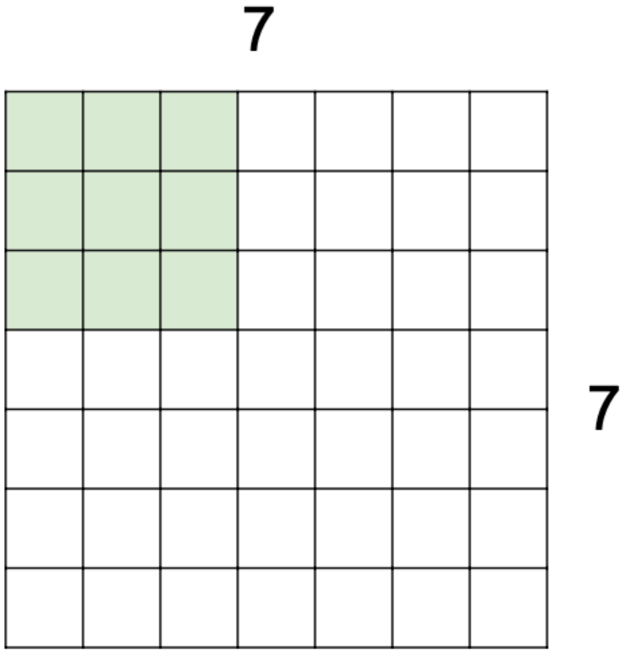
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> **3x3 output!**

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

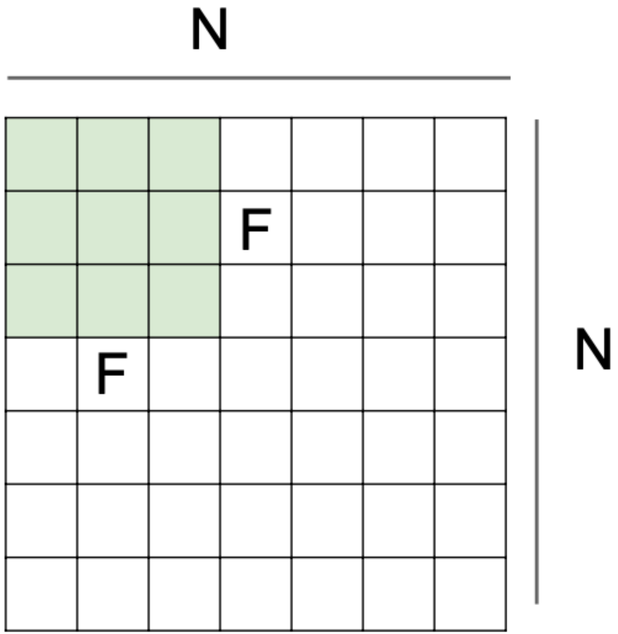
1. Convolution Layer



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer



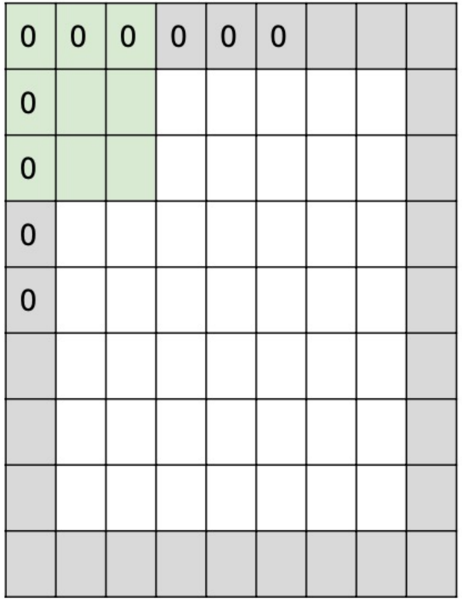
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

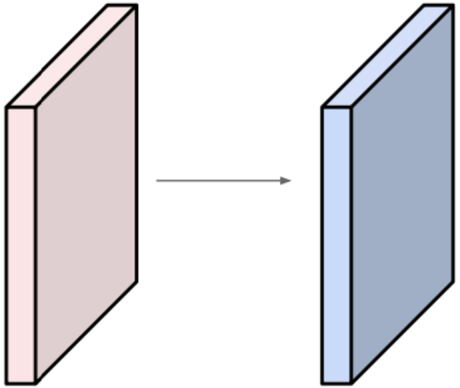
[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

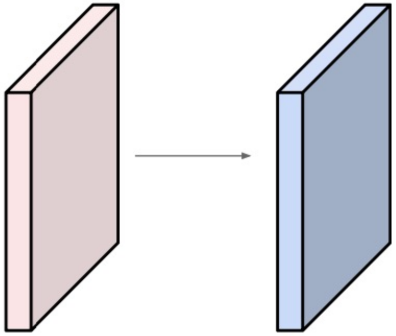
Output volume size: ?



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:

Input padding 양옆 Filter

(32+2*2-5)/1+1 = 32 spatially, so

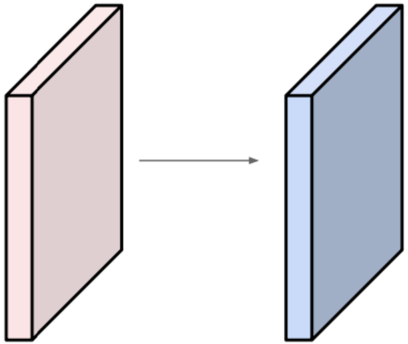
stride

32x32x10

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

Examples time:



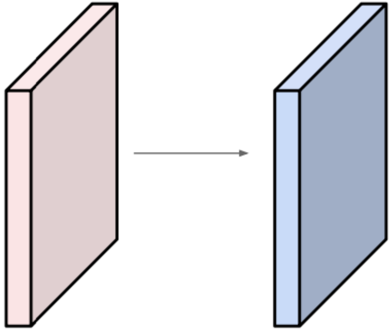
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

Examples time:



Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = 760

ten filters

$W^T x + b$

One

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li &
Justin Johnson & Serena Yeung

1. Convolution Layer(Summary)

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

1. Convolution Layer

PyTorch

```
# In[11]:
conv = nn.Conv2d(3, 16, kernel_size=3)
conv

# Out[11]:
Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
```

Instead of the shortcut kernel_size=3, we could equivalently pass in the tuple that we see in the output: kernel_size=(3, 3).

```
# In[12]:
conv.weight.shape, conv.bias.shape

# Out[12]:
(torch.Size([16, 3, 3, 3]), torch.Size([16]))
```

[Citation] Deep Learning with Pytorch, Eli Stevens, Luca, Thomas, 2020

1. Convolution Layer

PyTorch

```
# In[16]:
conv = nn.Conv2d(3, 1, kernel_size=3, padding=1)        <-- Now with padding
output = conv(img.unsqueeze(0))
img.unsqueeze(0).shape, output.shape

# Out[16]:
(torch.Size([1, 3, 32, 32]), torch.Size([1, 1, 32, 32]))
```

nn.ConstantPad1d(padding, value)
nn.ConstantPad2d(padding, value)
nn.ConstantPad3d(padding, value)

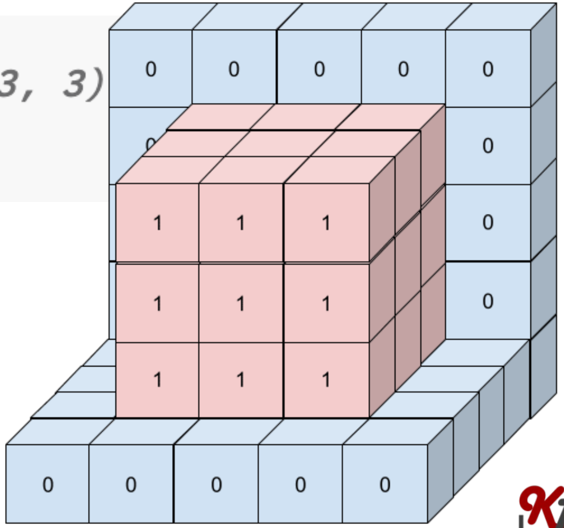
[Citation] Deep Learning with Pytorch, Eli Stevens, Luca, Thomas, 2020

1. Convolution Layer

PyTorch

torch.nn.functional.pad(input, pad, mode='constant', value=None)
-> padding starts from the last dimension
-> pad: int or tuple_(left,right) or (left,right, top, bottom)
or (left, right, top, bottom, front back)

```python
t4d = torch.empty(3, 3, 4, 2)
p3d = (0, 1, 2, 1, 3, 3) # pad by (0, 1), (2, 1), and (3, 3)
out = F.pad(t4d, p3d, "constant", 0)
print(out.size())
```



*How about Size?

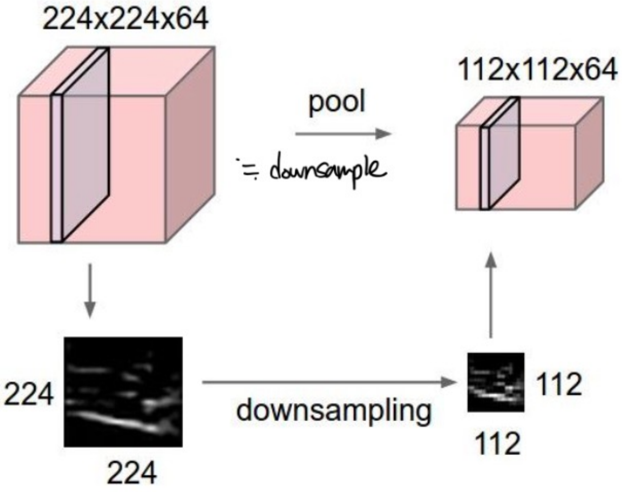
[Citation] Deep Learning with Pytorch, Eli Stevens, Luca, Thomas, 2020

2. Max Pooling Layer

2. Max Pooling Layer

Pooling layer

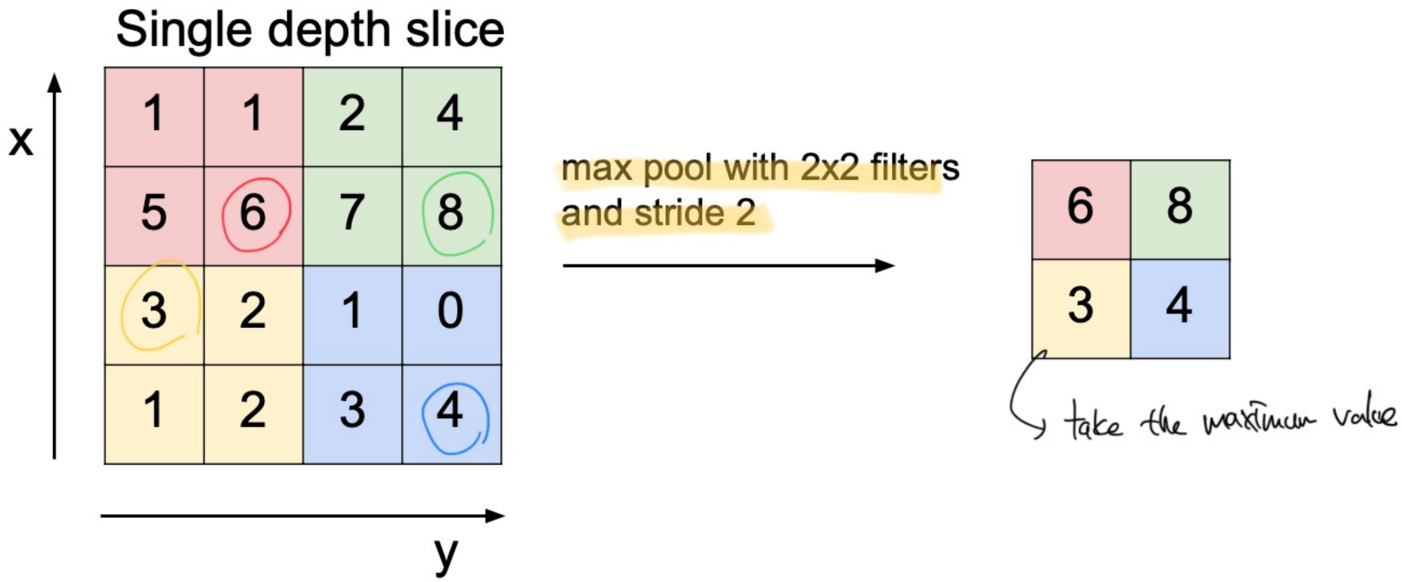
- makes the representations smaller and more manageable
- operates over each activation map independently:



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

2. Max Pooling Layer

MAX POOLING



[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

2. Max Pooling Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F, : Filter size
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

2. Max Pooling Layer

PyTorch

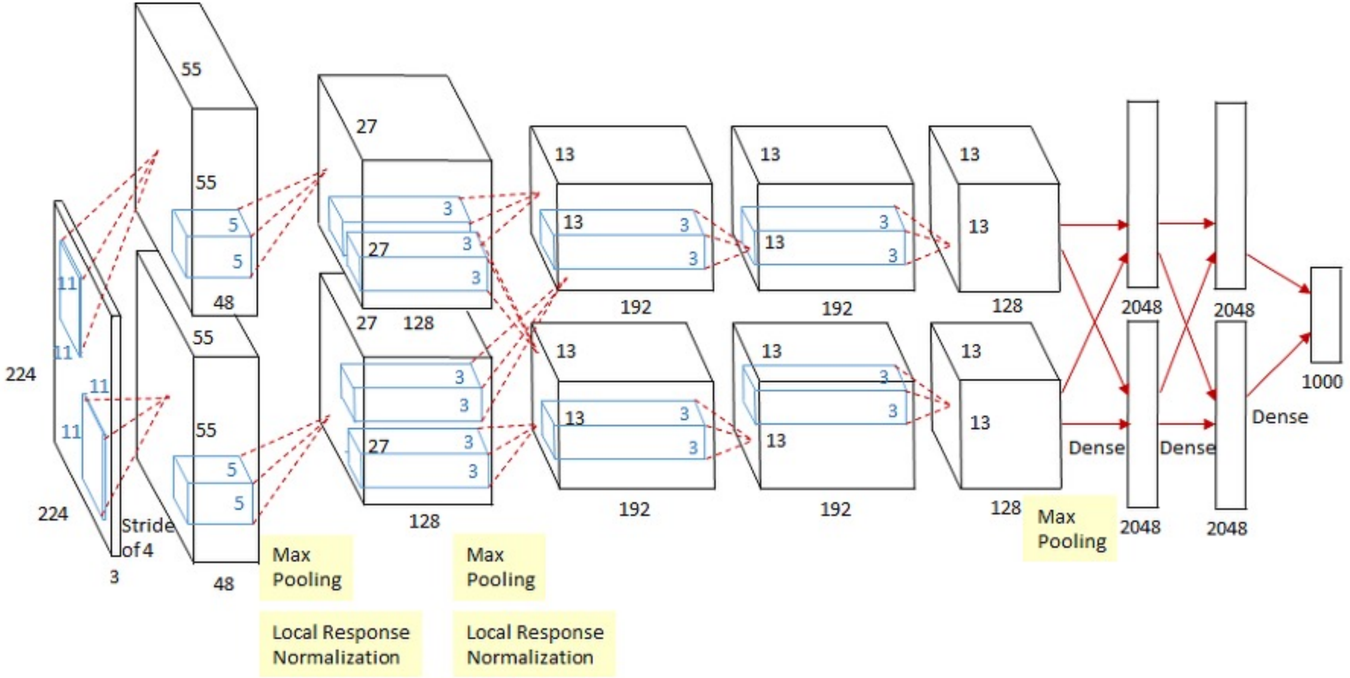
```
# In[21]:
pool = nn.MaxPool2d(2)
output = pool(img.unsqueeze(0))

img.unsqueeze(0).shape, output.shape

# Out[21]:
(torch.Size([1, 3, 32, 32]), torch.Size([1, 3, 16, 16]))
```

```
>>> # pool of square window of size=3, stride=2
>>> m = nn.MaxPool2d(3, stride=2)
>>> # pool of non-square window
>>> m = nn.MaxPool2d((3, 2), stride=(2, 1))
>>> input = torch.randn(20, 16, 50, 32)
>>> output = m(input)
```

[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

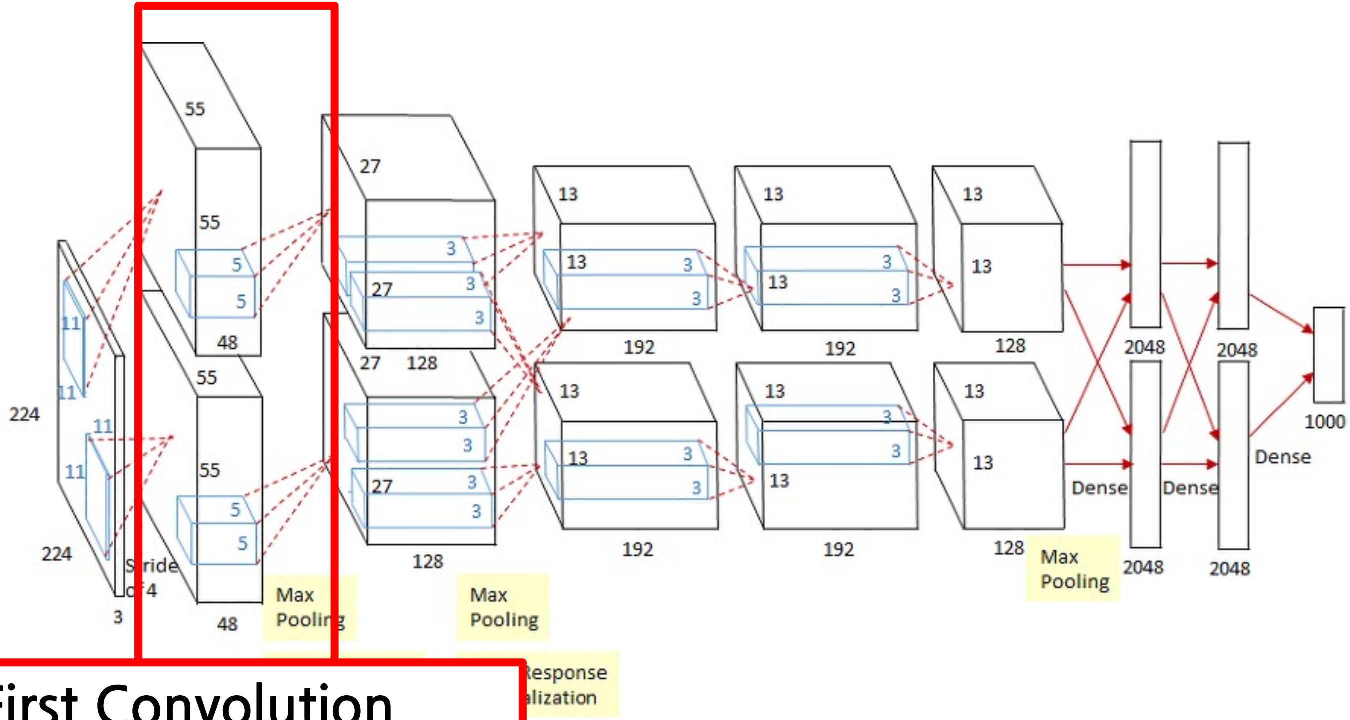
3. AlexNet
– 2012 Top Network from ImageNet Classification

3. AlexNet -2012 Top Network from ImageNet Classification



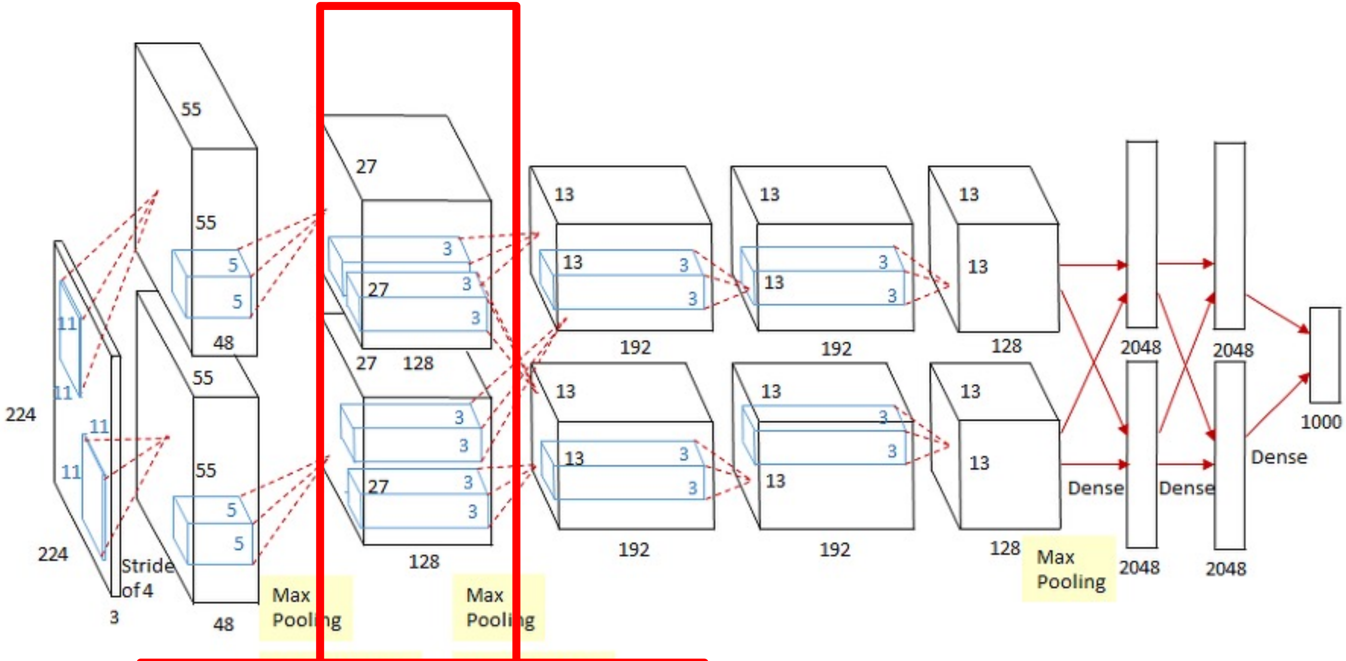
[Citation] Lecture 5: Convolutional Neural Networks, Fei-Fei-Li & Justin Johnson & Serena Yeung

3. AlexNet -2012 Top Network from ImageNet Classification



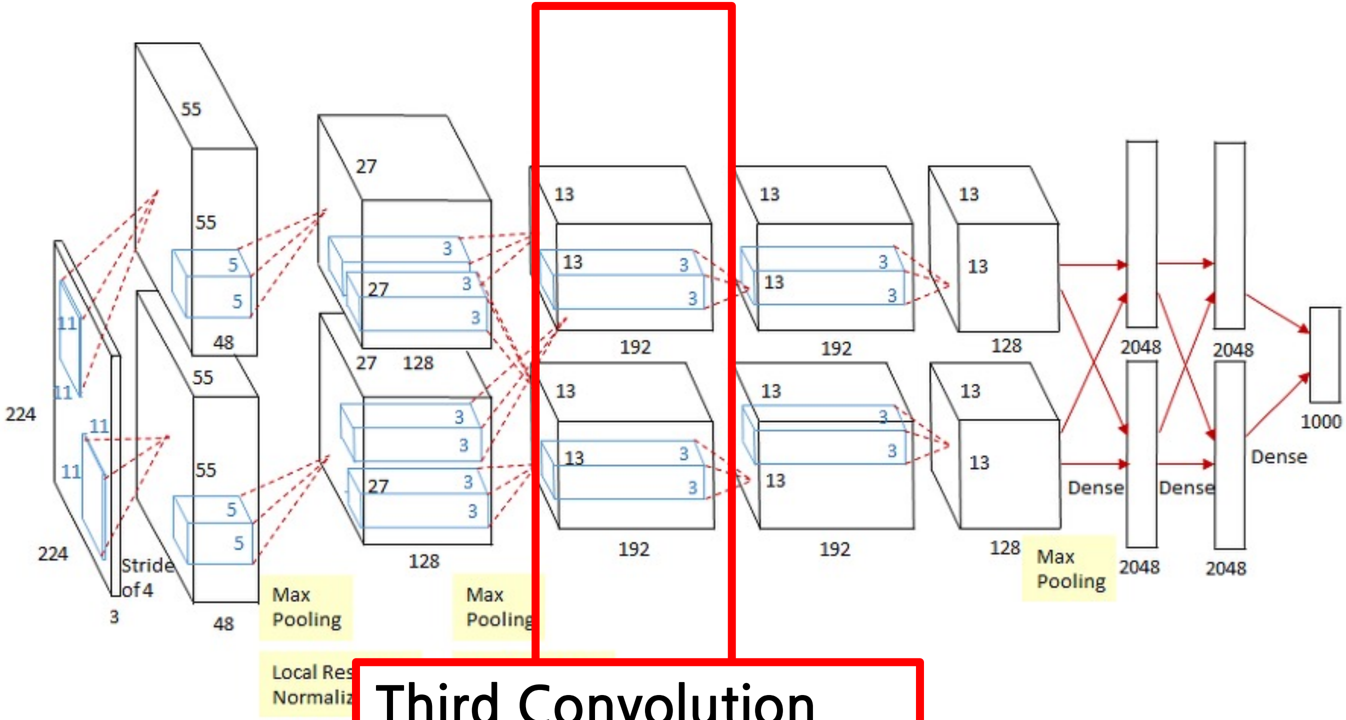
First Convolution
with 96 Filters 11*11

3. AlexNet -2012 Top Network from ImageNet Classification



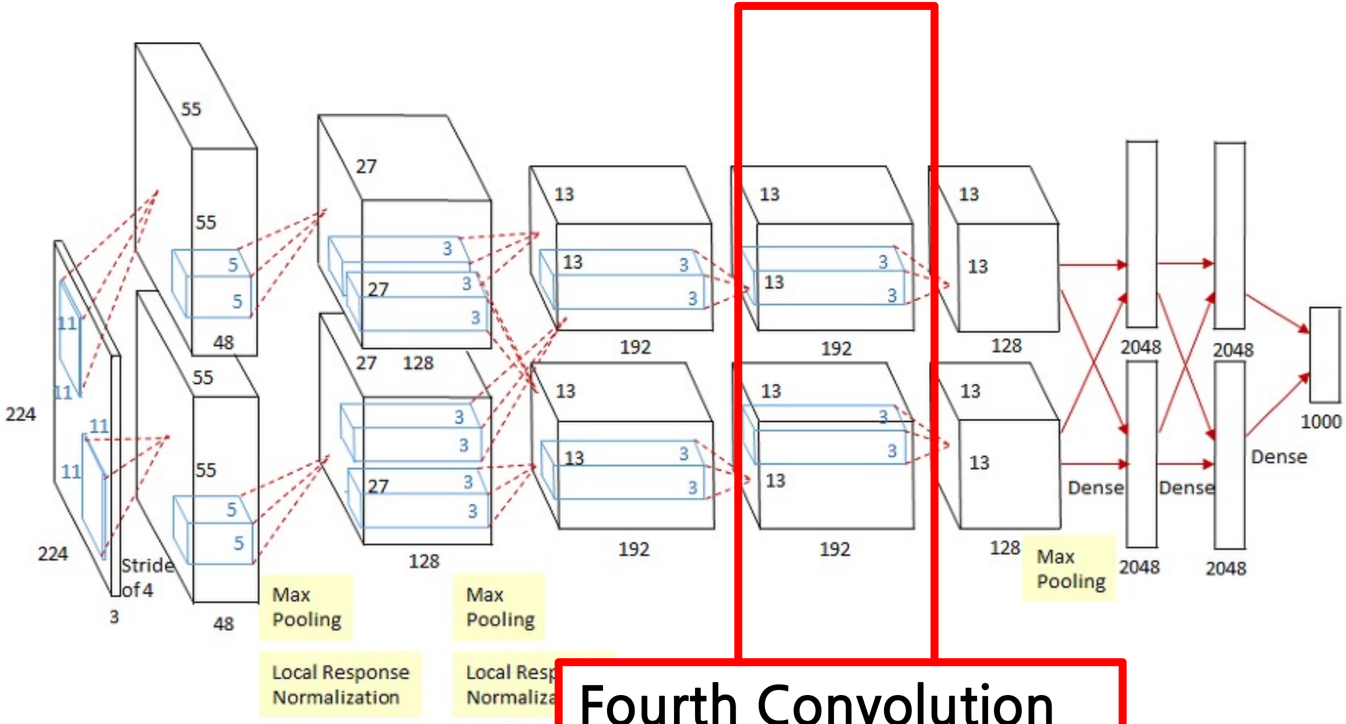
Second Convolution
with 256 Filters 5*5

3. AlexNet -2012 Top Network from ImageNet Classification



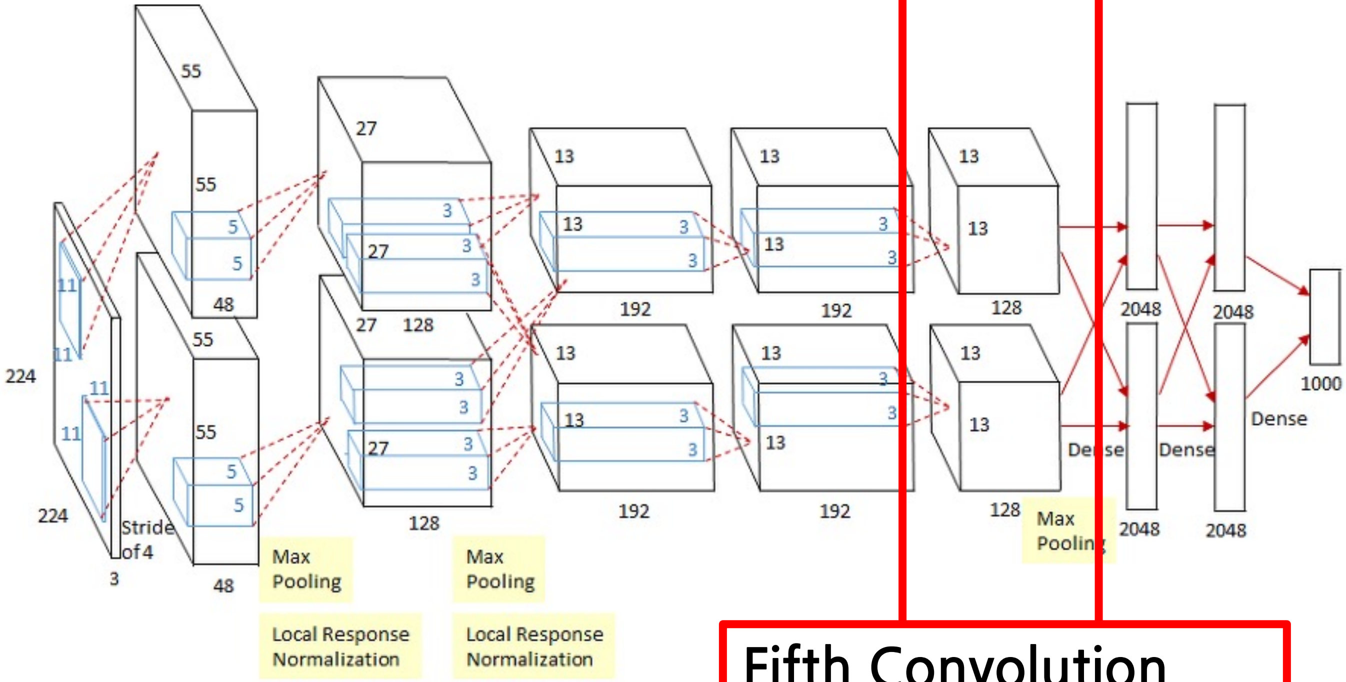
Third Convolution with 384 Filters 3*3

3. AlexNet -2012 Top Network from ImageNet Classification



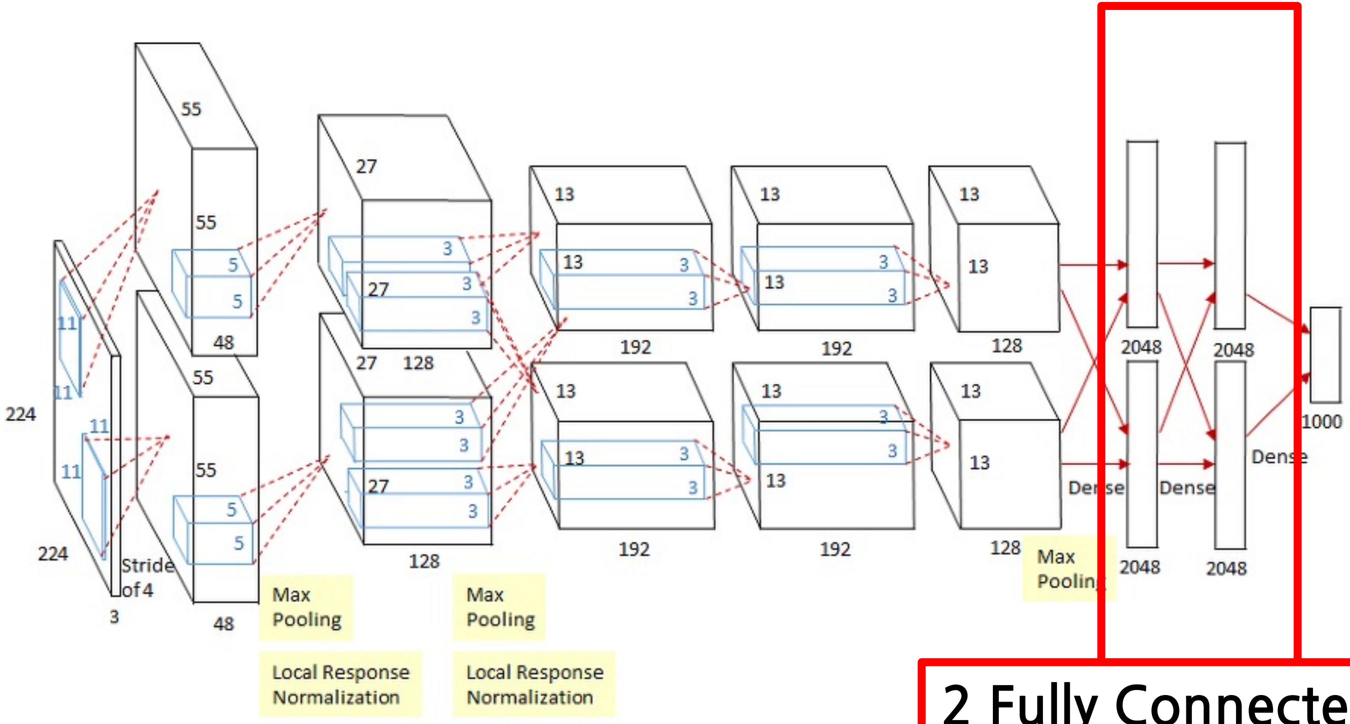
Fourth Convolution with 384 Filters 3*3

3. AlexNet -2012 Top Network from ImageNet Classification



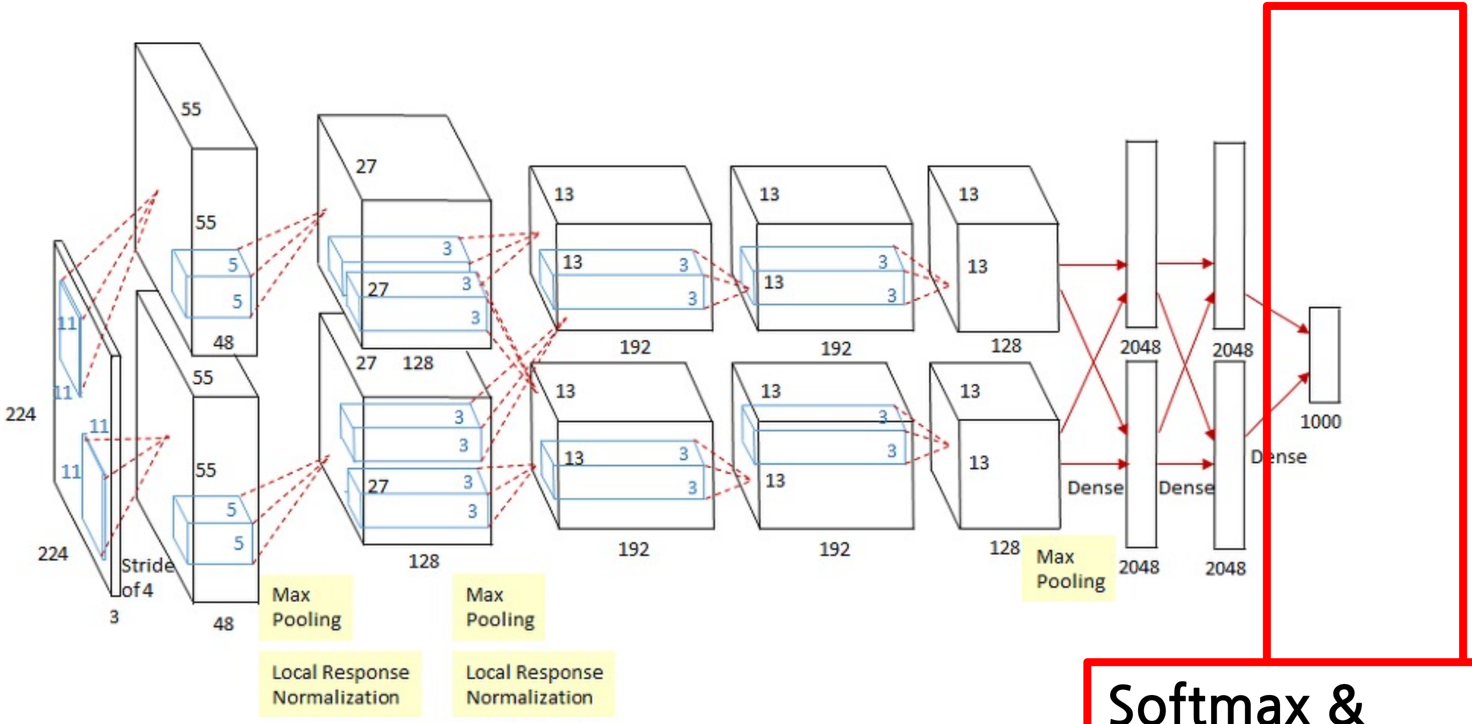
Fifth Convolution with 256 Filters 3*3

3. AlexNet -2012 Top Network from ImageNet Classification



2 Fully Connected layer with 4096-dim

3. AlexNet -2012 Top Network from ImageNet Classification



Softmax &
Backpropagation !

4. Other Remarks

ConvBlock = stacking Conv layer, Pooling layer, FC layer

* General Structure of ConvNet
- (Conv-ReLU)*N → Pool(or not) → FC layer → ReLU → FC or Softmax

* Training Technique
- Batch Normalization, LR Scheduler, Optimizer, Dropout, Data Augmentation
- SOTA: Transformer,,

* Limitation
- Deeper Conv layers don’t guarantee better performance

4. Other Remarks

```python
# In[17]:
with torch.no_grad():
    conv.bias.zero_()

with torch.no_grad():
    conv.weight.fill_(1.0 / 9.0)
```

Blurred image due to convolution

```python
# In[18]:
output = conv(img.unsqueeze(0))
plt.imshow(output[0, 0].detach(), cmap='gray')
plt.show()
```

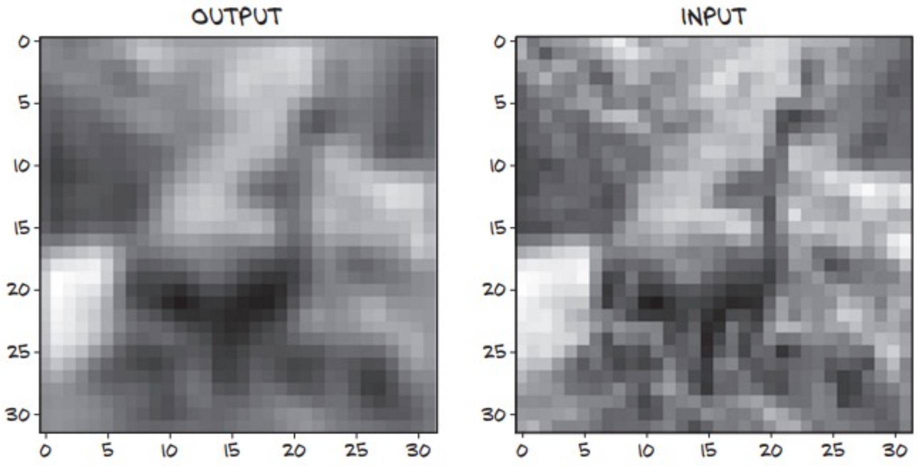


Figure 8.4 Our bird, this time blurred thanks to a constant convolution kernel

4. Other Remarks

```
# In[19]:
conv = nn.Conv2d(3, 1, kernel_size=3, padding=1)

with torch.no_grad():
    conv.weight[:] = torch.tensor([[-1.0, 0.0, 1.0],
                                   [-1.0, 0.0, 1.0],
                                   [-1.0, 0.0, 1.0]])
    conv.bias.zero_()
```

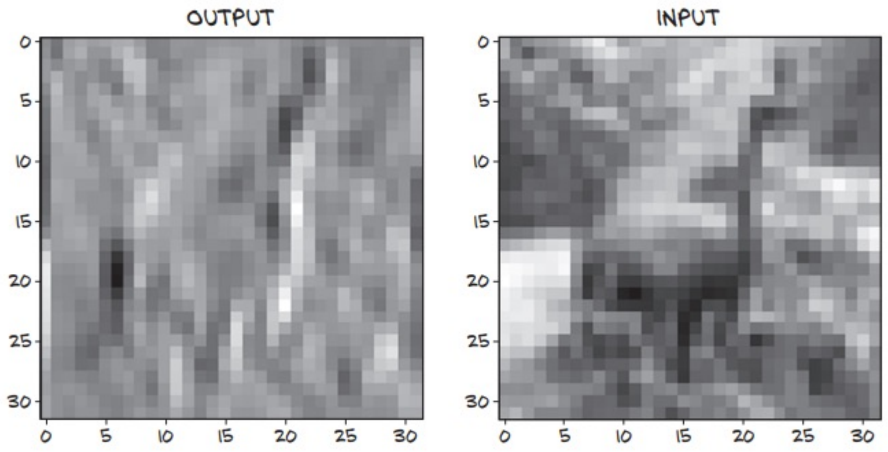


Figure 8.5 Vertical edges throughout our bird, courtesy of a handcrafted convolution kernel

Conv layer with Vertical Edge

Q&A

Have a nice week🥰