Convolutional Neural Networks

Jisang Han

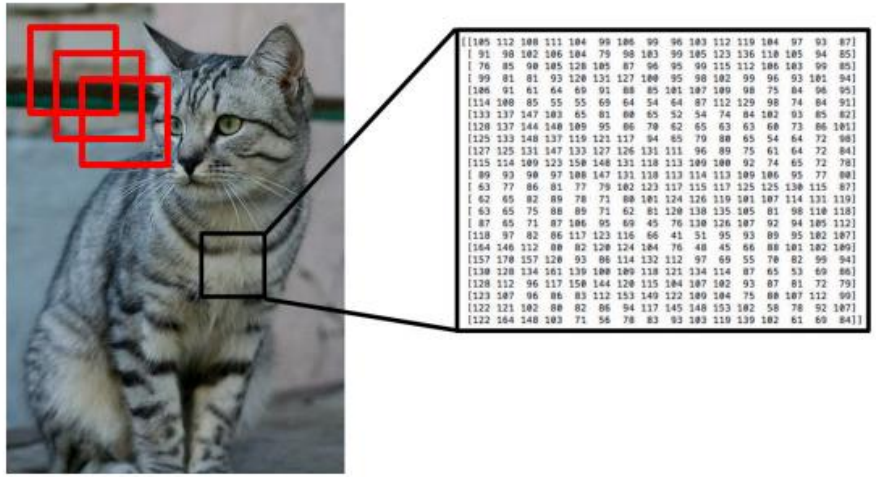
onground@korea.ac.kr

KUGODS

Department of Computer Science and Engineering, Korea University

Korea University GOogle Developer Students
KUGODS

What is Convolution?

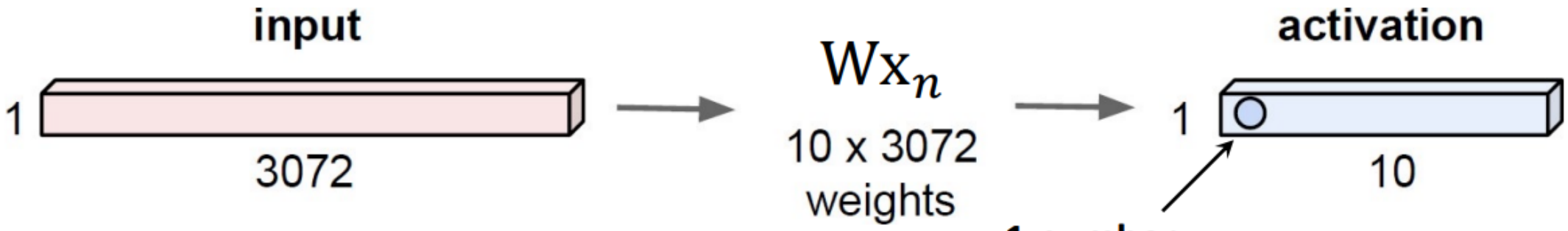


- **Local** processing of data
- Parameter Sharing
- Sparsity of Connections

Lecture Note from COSE474

What is Convolution?

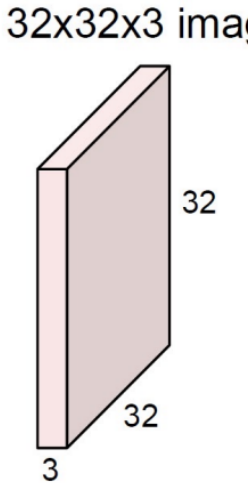
- **The Perceptron:** $32 \times 32 \times 3$ image $\rightarrow$ stretch to 3072×1



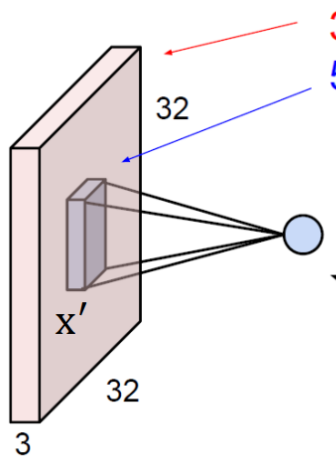
- Stretching 2D image into 1D vector is not suitable in considering a spatial context of 2D image.
- Too many network parameters are required even for an image of moderate size.

Lecture Note from COSE474

What is Convolution?

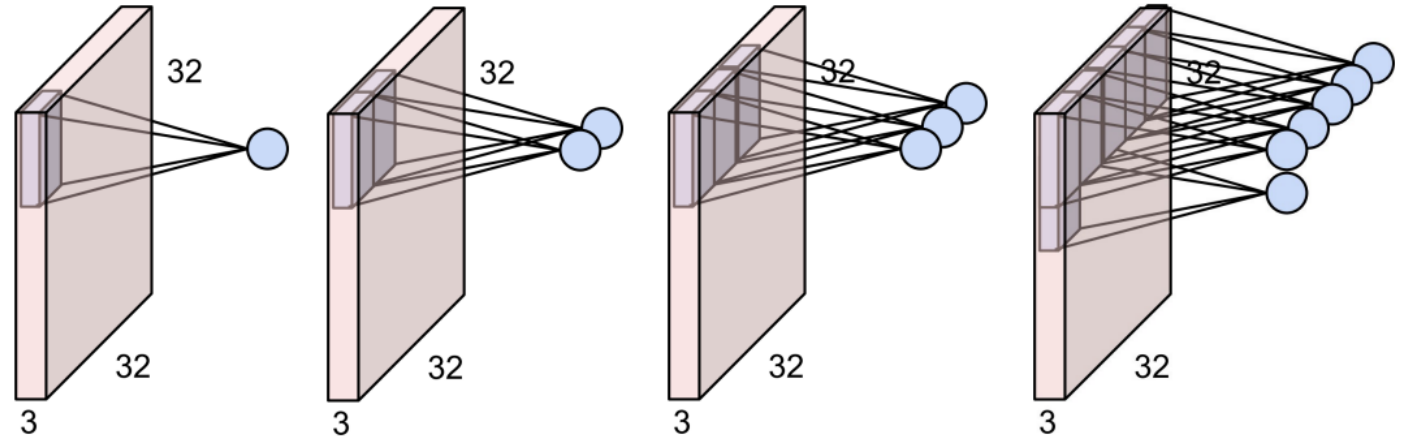


Convolve the filter with the image, i.e., "slide over the image spatially, computing dot products"



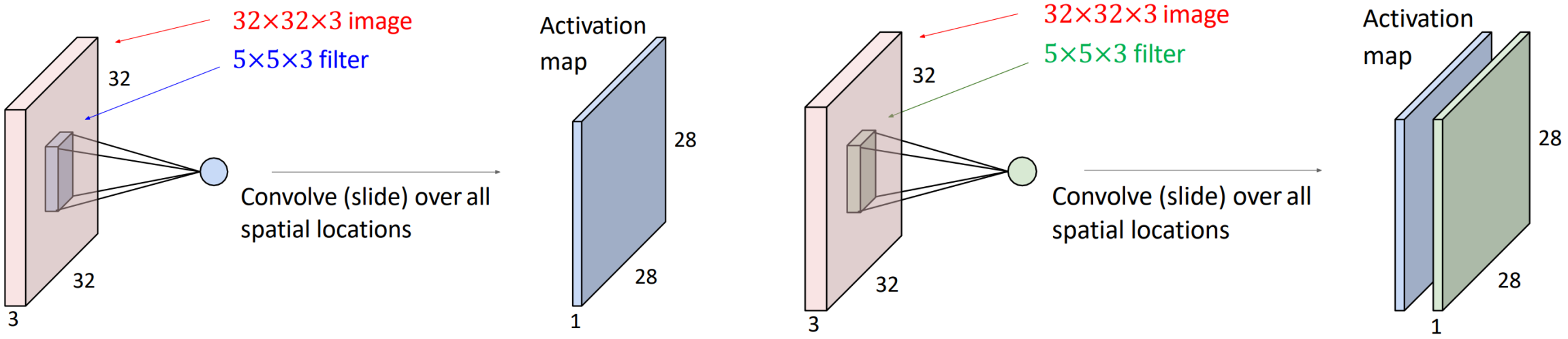
1 number:
the result of taking a dot product between the filter and a small $5 \times 5 \times 3$ chunk of the image x'
(i.e., $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$Wx' + b$$

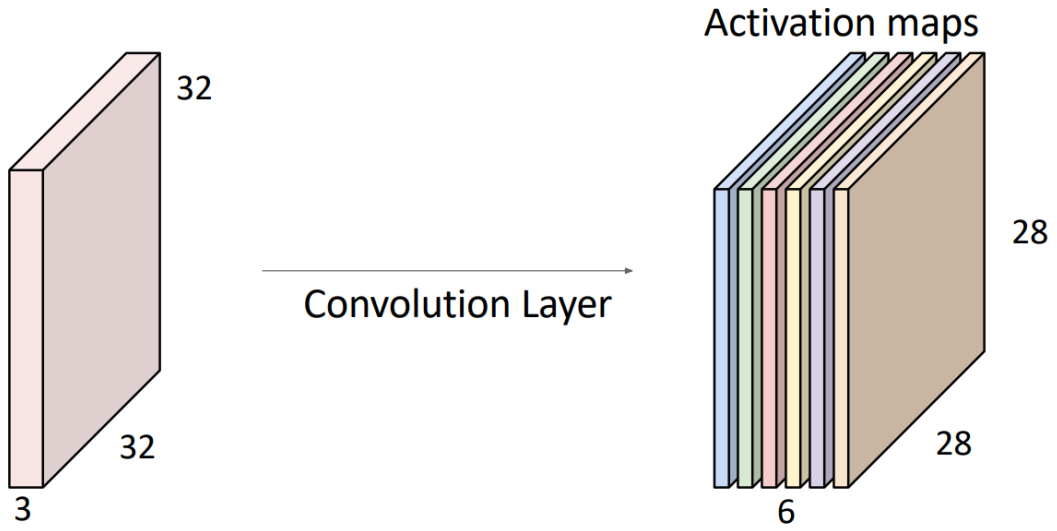


Lecture Note from COSE474

What is Convolution?



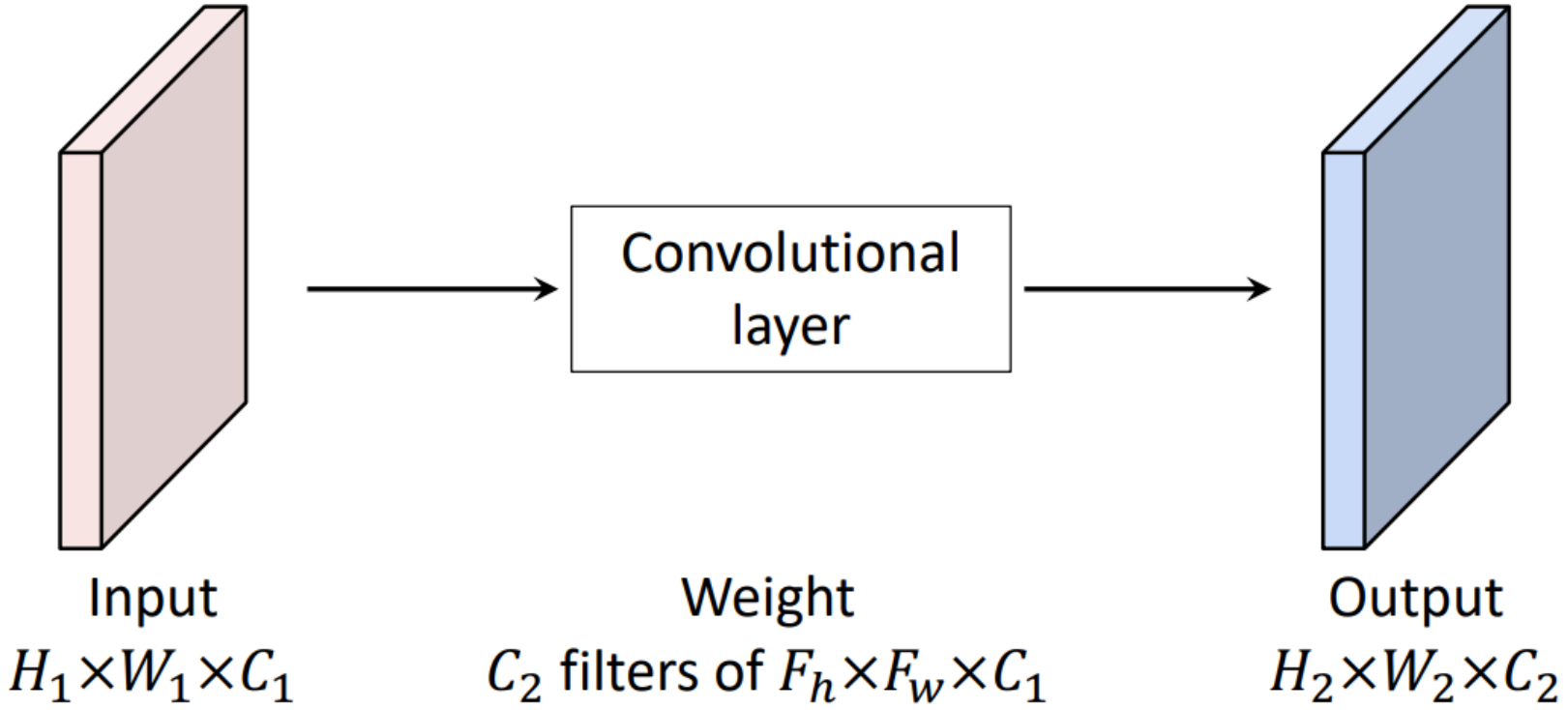
For example, if we had 6 5×5 filters, we'll get 6 separate activation maps.



Lecture Note from COSE474

What is Convolution?

The number of parameters in convolutional layer

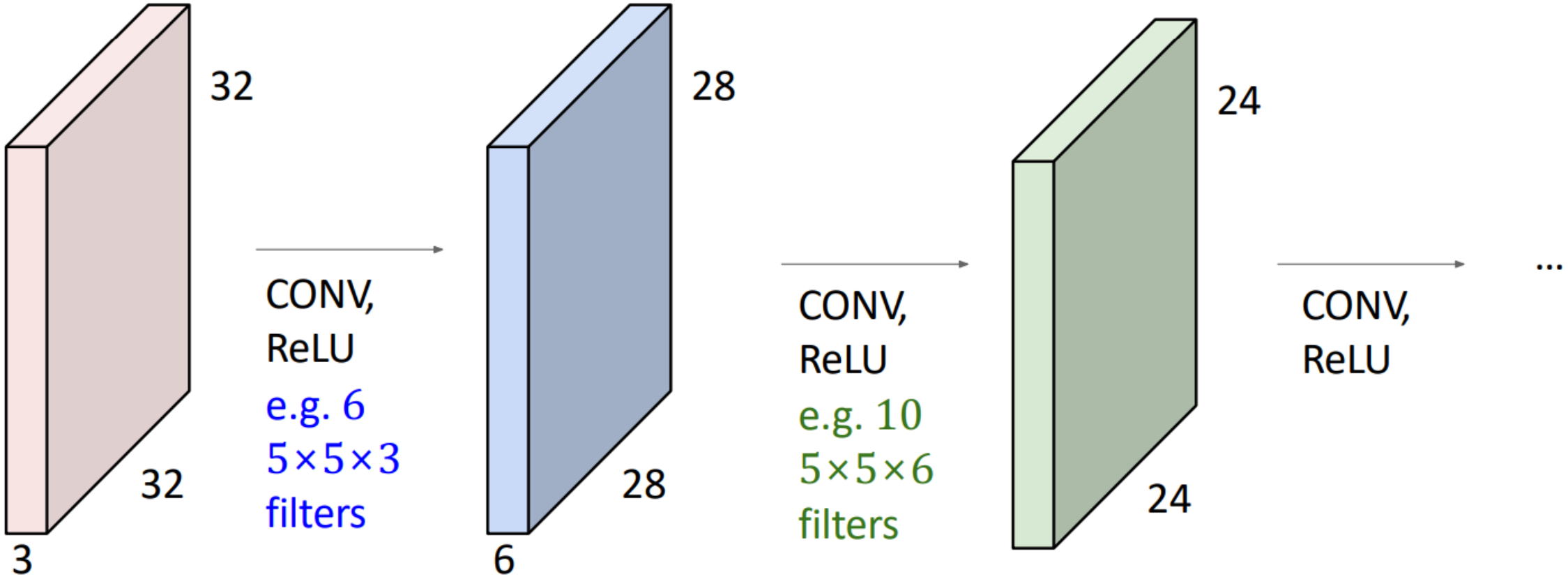


→ The number of weights: $C_2 \times (C_1 \times F_h \times F_w)$, The number of bias: C_2

Lecture Note from COSE474

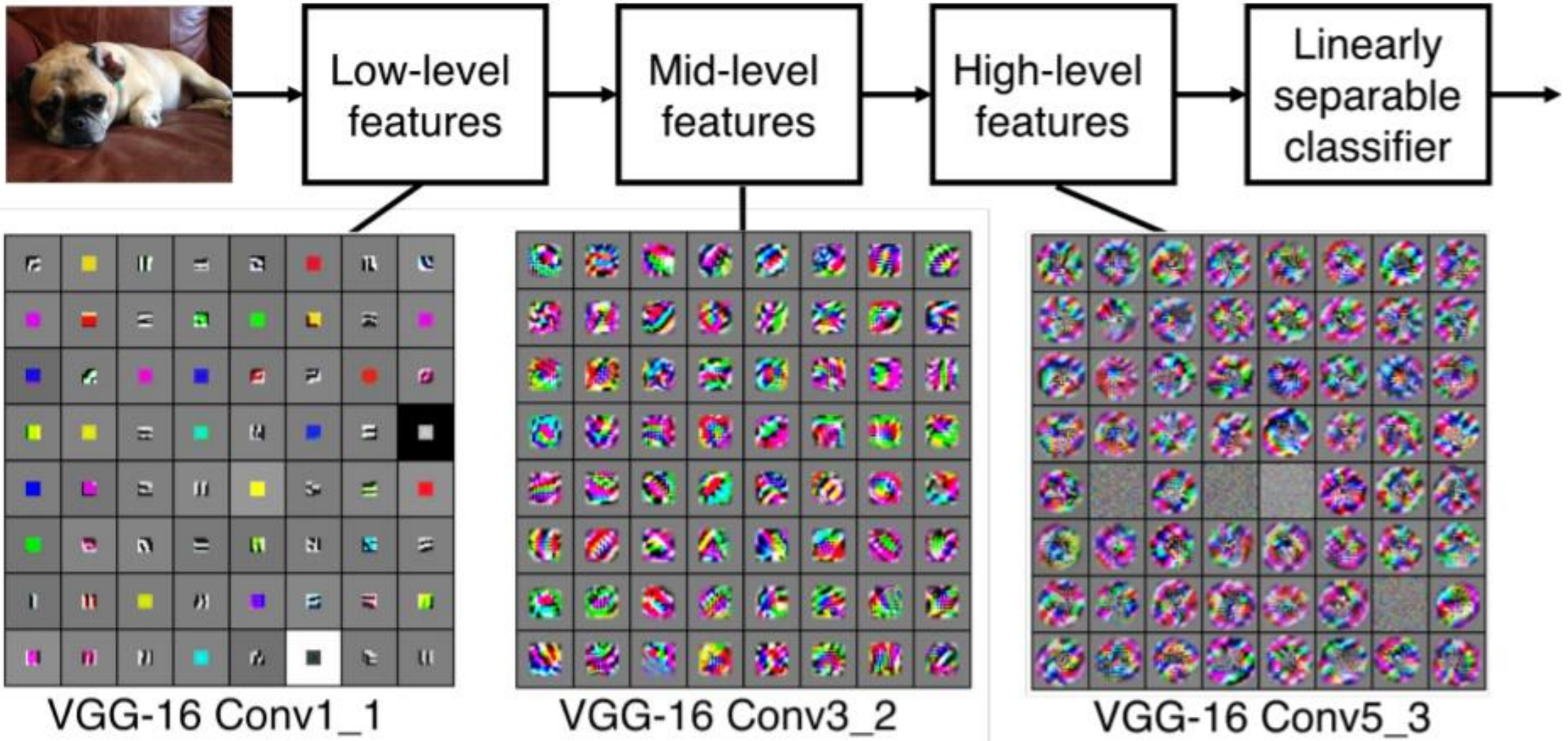
What is Convolution?

- **ConvNet** is a sequence of convolution layers, interspersed with non-linear activation functions (e.g., ReLU)



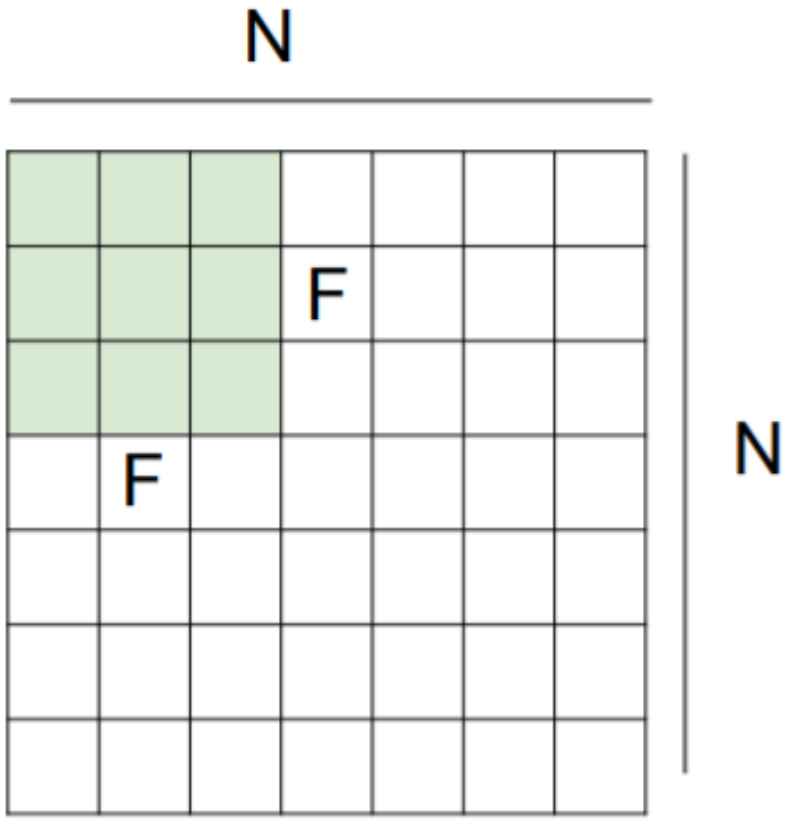
Lecture Note from COSE474

What is Convolution?



Lecture Note from CS231n

Activation Map

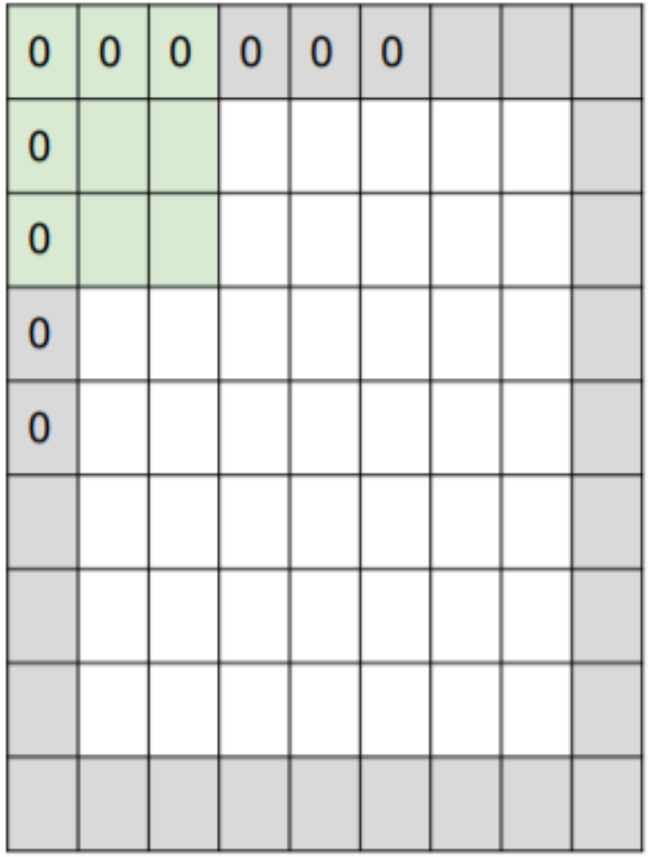


Output size:
(N - F) / stride + 1

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 :\

Lecture Note from CS231n

Activation Map



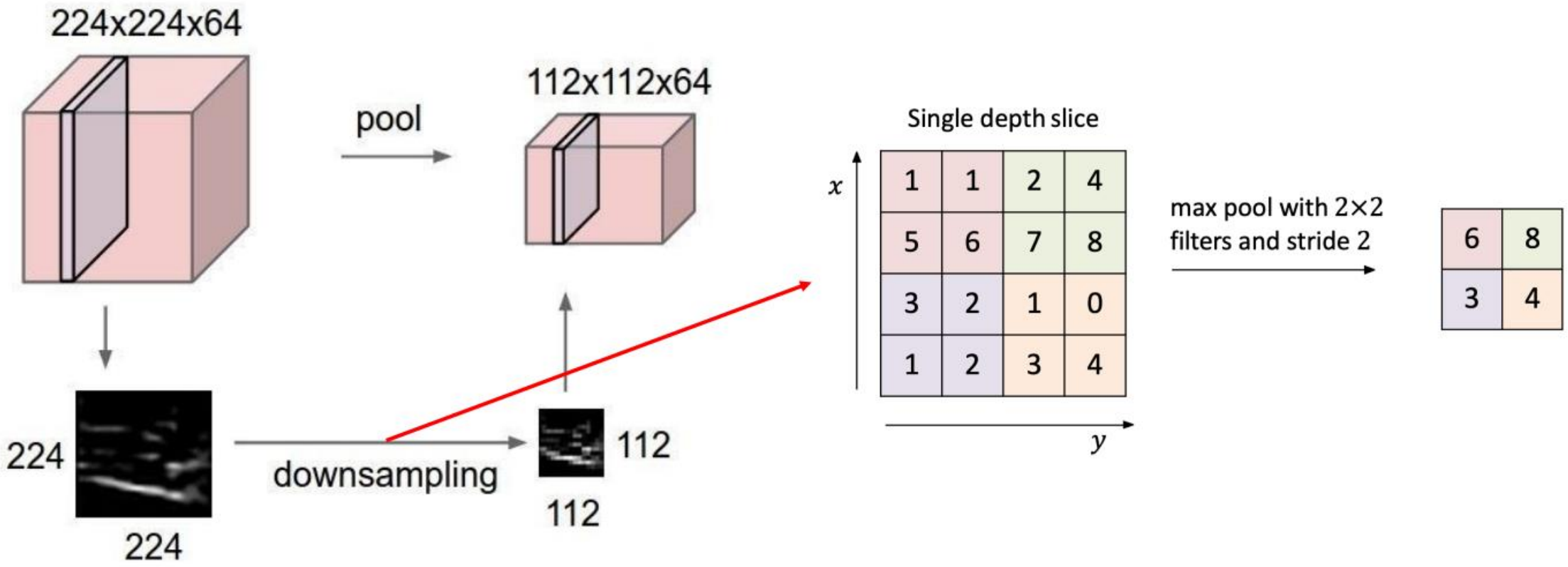
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Lecture Note from CS231n

Pooling Layer

- **Max Pooling** / **Average Pooling**
- Makes the representations **smaller** and more **manageable**.



Lecture Note from CS231n

PyTorch

- **Conv Layer**

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

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=1,
    out_channels=64,
    kernel_size=3,
    stride=1,
    padding=1
)
```

```
self.conv1 = nn.Conv2d(1, 64, 3, 1, 1)
```

PyTorch

- **MaxPooling**

https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html

MAXPOOL2D

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

```
nn.MaxPool2d(
    kernel_size=2,
    stride=2
)
```

```
self.maxpool1 = nn.MaxPool2d(2, 2)
```

PyTorch

- **ConvNet**

```
self.convlayers = nn.Sequential(
            nn.Conv2d(1, 64, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(128, 256, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(7, 1)
        )
```

→ Input : 1 x 28 x 28

→ Conv : 64 x 28 x 28

→ MaxPool : 64 x 14 x 14

→ Conv : 128 x 14 x 14

→ MaxPool : 128 x 7 x 7

→ Conv : 256 x 7 x 7

→ MaxPool : 256 x 1 x 1

→ Output : 256 x 1 x 1

PyTorch

- **Classifier**

```
self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )
```

→ Input : 256 x 1 x 1

→ Flatten : 256
→ Linear : 128

→ Linear : 10

PyTorch

- **Naïve ConvNet**

```python
class Naive_ConvNet(nn.Module):
    def __init__(self):
        super().__init__()

        self.convlayers = nn.Sequential(
            nn.Conv2d(1, 64, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(128, 256, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(7, 1)
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.convlayers(x)
        x = self.classifier(x)
        return x
```

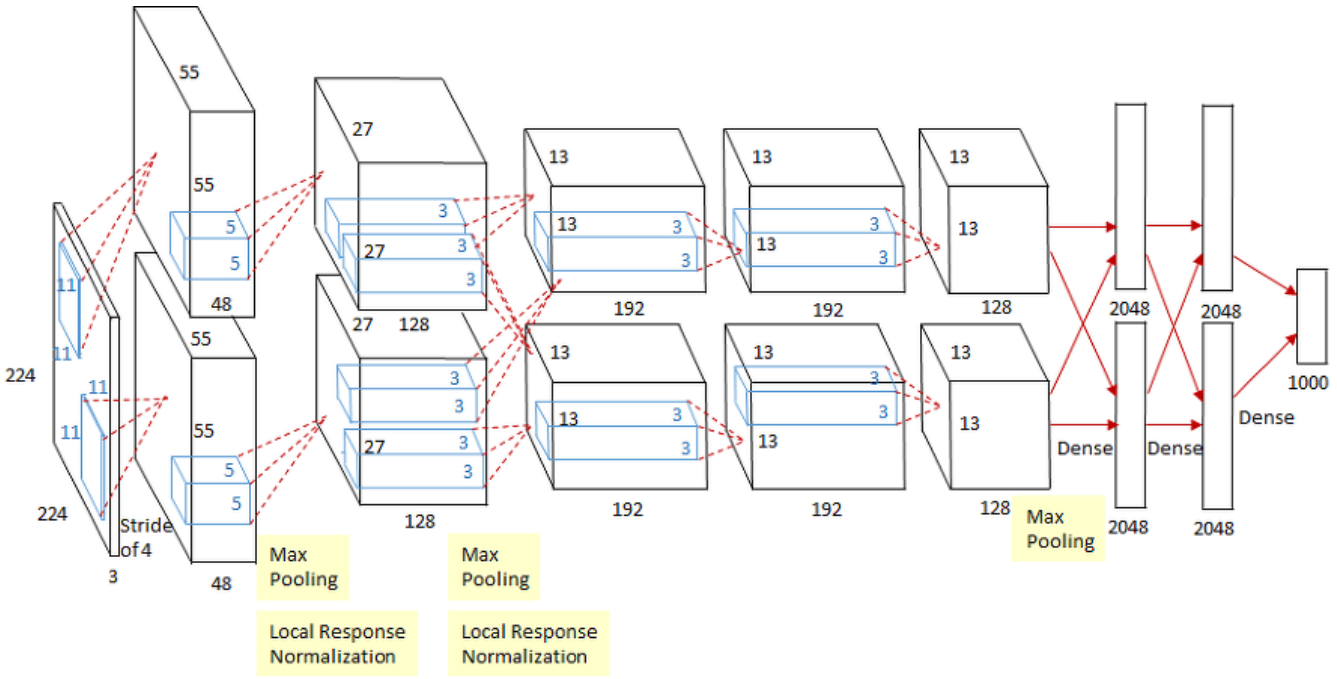
PyTorch

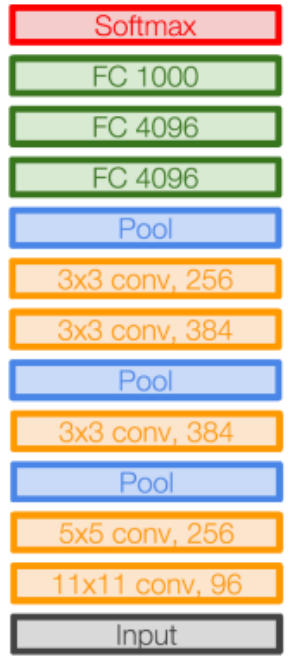
- **Naïve ConvNet**

```python
class Naive_ConvNet(nn.Module):
    def __init__(self):
        super().__init__()

        self.convlayers = nn.Sequential(
            nn.Conv2d(1, 64, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(128, 256, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(7, 1)
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.convlayers(x)
        x = self.classifier(x)
        return x
```

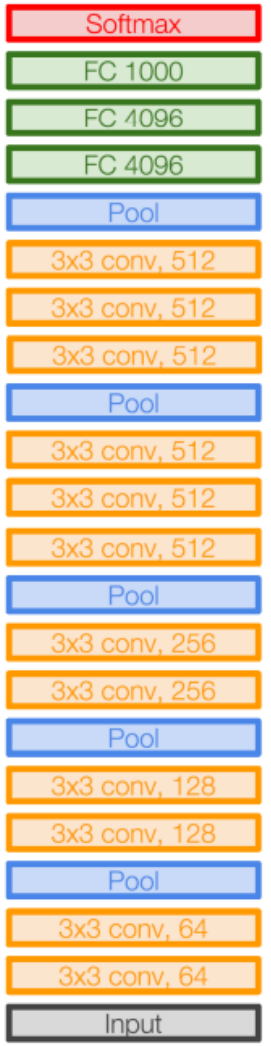
Assignment

- Implementation of **AlexNet** and **VGG16**





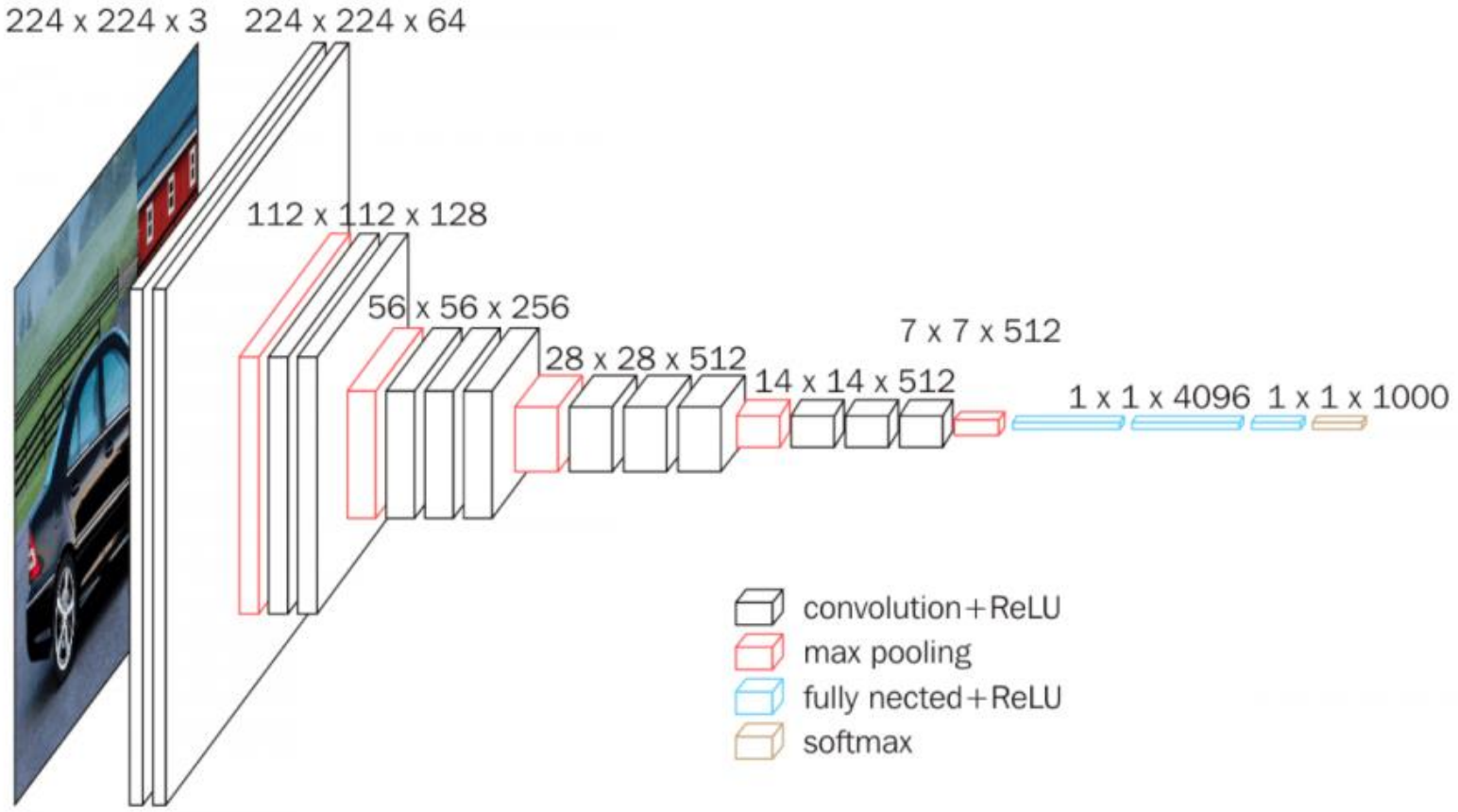
AlexNet

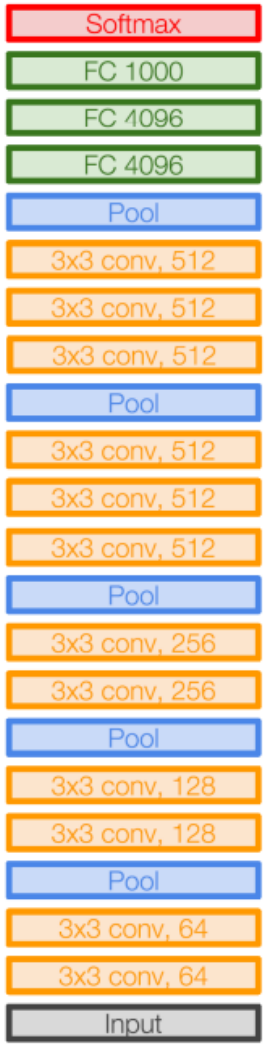


VGG16

Assignment

- Implementation of **AlexNet** and **VGG16**





VGG16

Assignment

- Implementation of **AlexNet** and **VGG16**
- You should implement **at least one architecture**
- **Due Date 11/13**

- Lectures, Exercises, and Assignments will be uploaded in Github (https://github.com/ONground-Korea/KUGODS-2022-GDSC-Deeplearning-Session)

Thank you!
Q & A