Machine Learning and MLP

Honggyu An

hg010303@korea.ac.kr

Artificial Intelligence in KU (AIKU)

Department of Computer Science and Engineering, Korea University

AIKU

Image Classification

(assume given a set of labels)
{dog, cat, truck, plane, ...}

cat
dog
bird
deer
truck

Image Classification

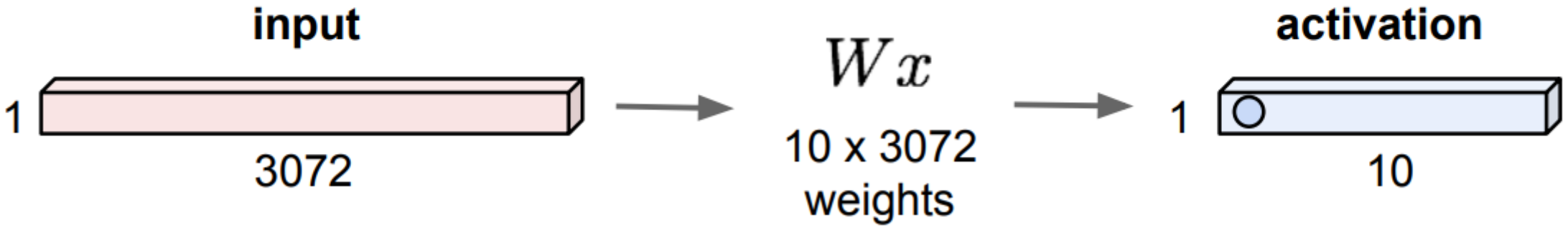
What we see

25 43 11 04 70 87 12 31 43 10 05 77 12 06 45 09 29 30 02
56 22 75 03 22 96 45 12 23 03 77 67 81 45 22 04 90 22 21
32 45 41 91 87 62 35 02 00 11 62 25 43 11 04 70 87 12 61
31 43 10 05 77 12 06 45 09 29 30 56 22 75 03 22 96 45 05
12 23 03 77 67 81 45 22 04 90 22 32 45 41 91 87 62 35 44
02 00 11 62 25 43 11 04 70 87 12 31 43 10 05 77 12 06 10
45 09 29 30 56 22 75 03 22 96 45 12 23 03 77 67 81 45 55
22 04 90 22 32 45 41 91 87 62 35 02 00 11 62 25 43 11 80
04 70 87 12 31 43 10 05 77 12 06 45 09 29 30 56 22 75 08
03 22 96 45 12 23 03 77 67 81 45 22 04 90 22 32 45 41 99
91 87 62 35 02 00 11 62 22 01 00 72 65 23 01 00 22 04 30
90 22 32 45 41 91 87 62 35 02 00 11 62 25 43 11 04 70 42
87 12 31 43 10 05 77 12 06 45 09 29 30 56 22 75 03 22 91
96 45 12 23 03 77 67 81 45 22 04 90 22 32 45 41 91 87 40
62 35 02 00 11 62 22 01 00 72 65 23 01 00 56 22 75 03 67
22 96 45 12 23 03 77 67 81 45 22 04 90 22 32 45 41 91 22

What computers see

Image Classification with MLP

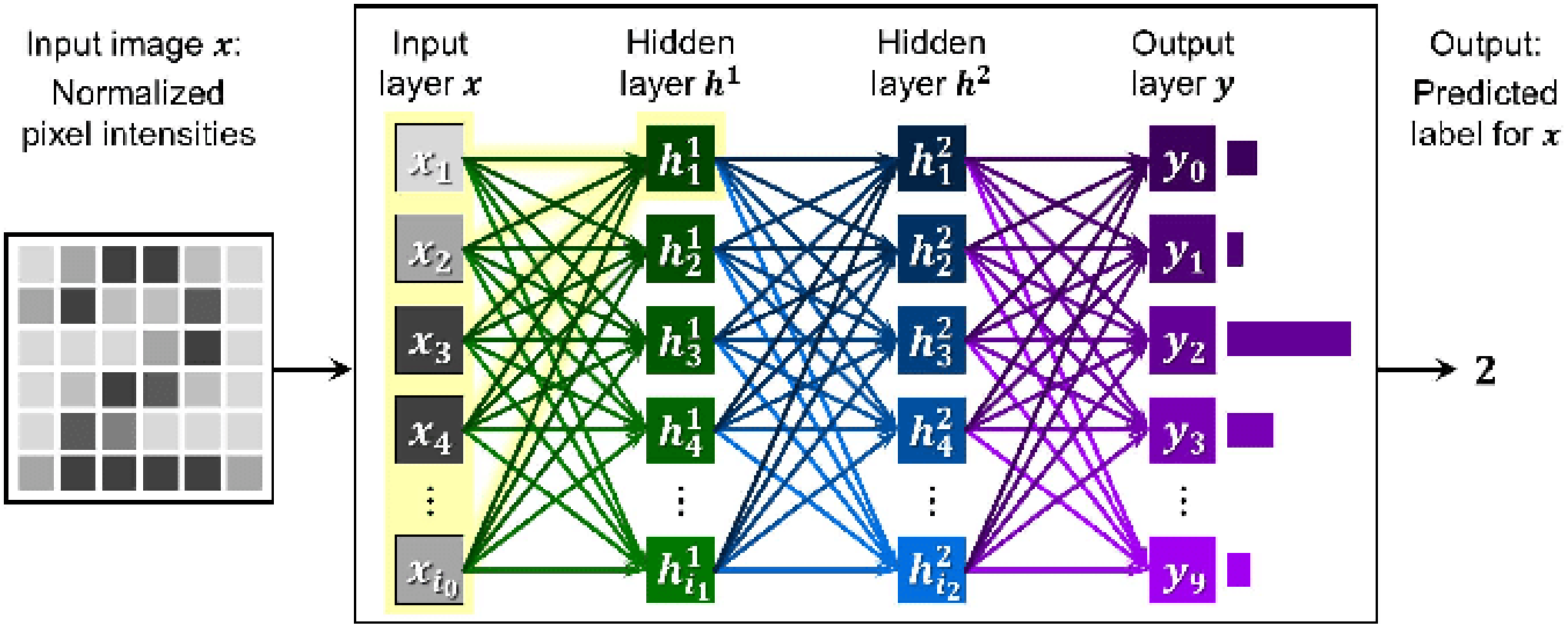
32x32x3 image -> stretch to 3072 x 1



Problem

1. Spatial component of image is destroyed
2. When image resolution is large, Computational Costs would be higher.

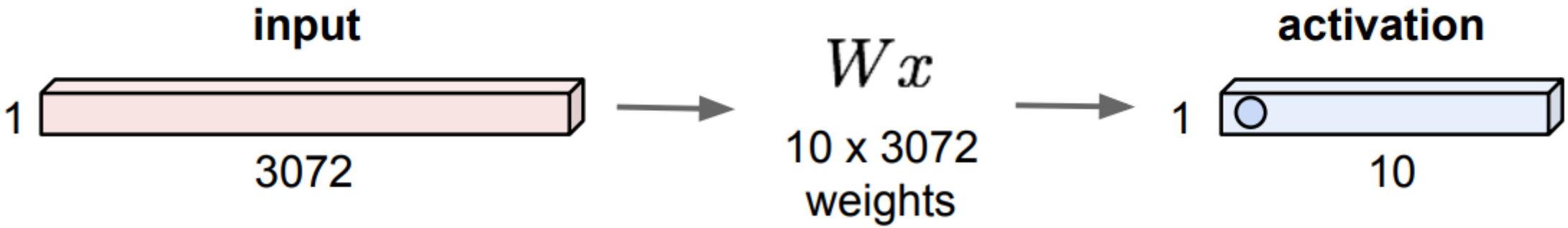
Image Classification with MLP



- Example of Classification of MLP to predict hand-writing number.

Image Classification with MLP

32x32x3 image -> stretch to 3072 x 1



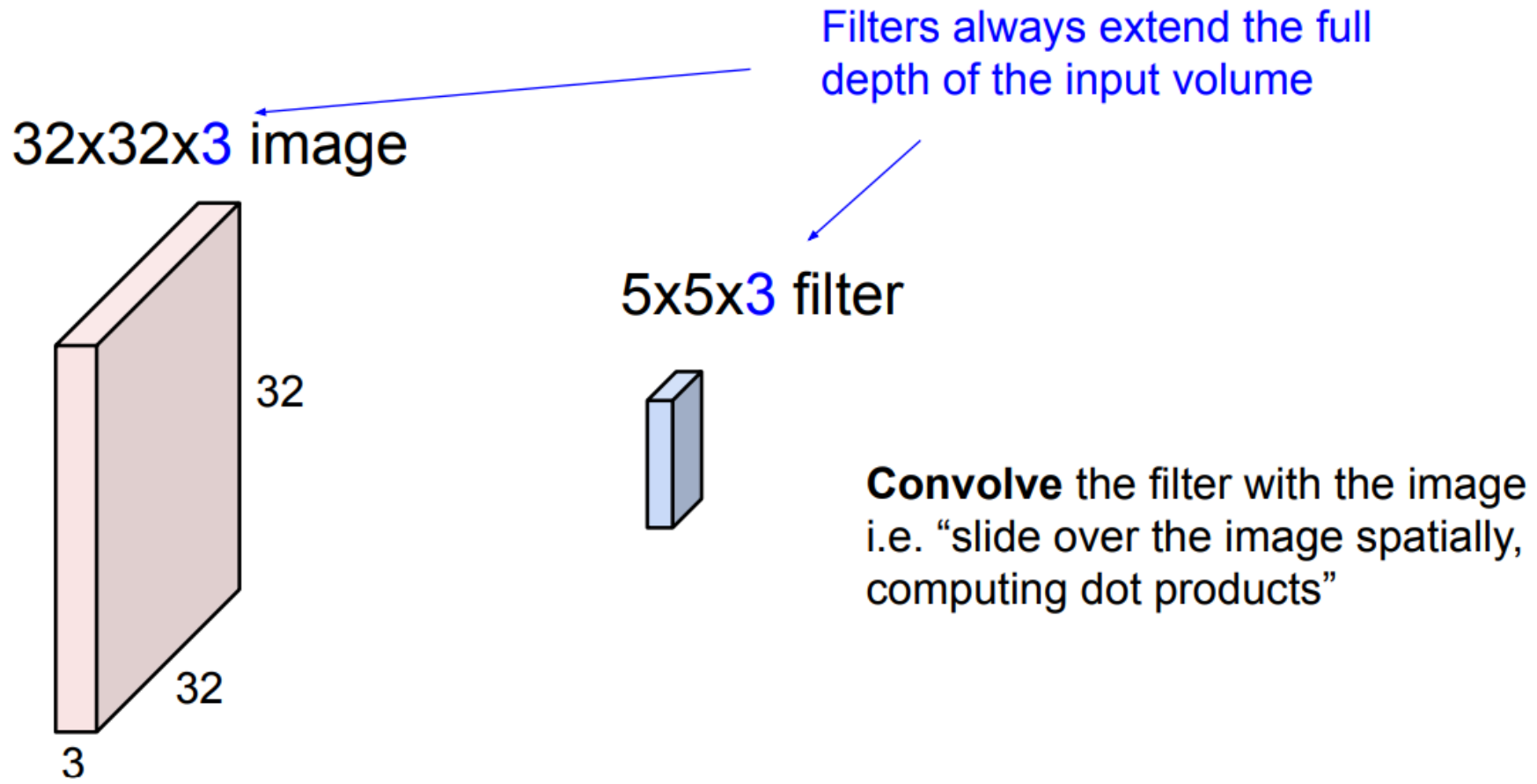
Problem

1. Spatial component of image is destroyed
2. When image resolution is large, Computational Costs would be higher.

Solution

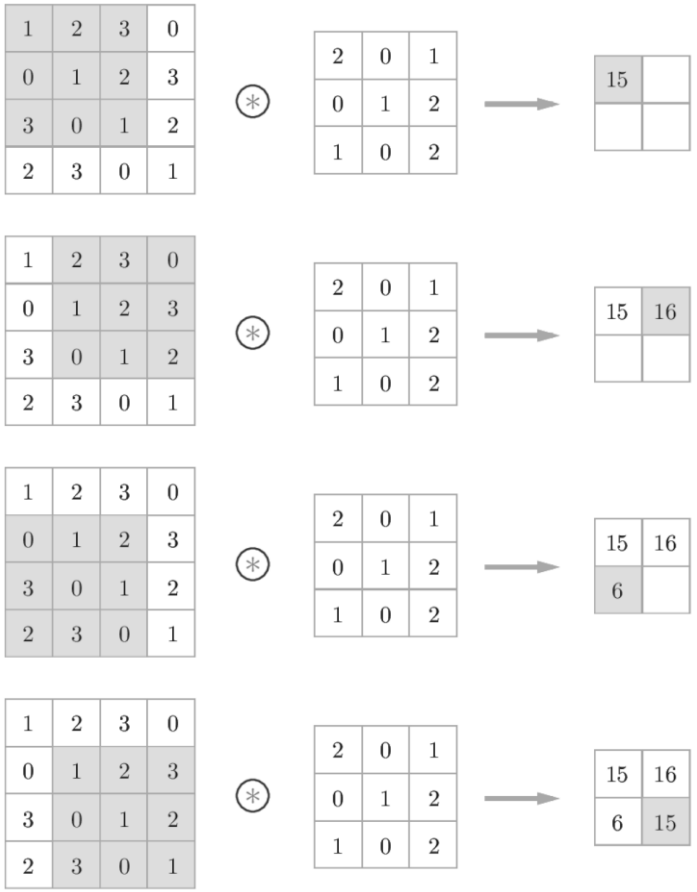
Convolution layer!

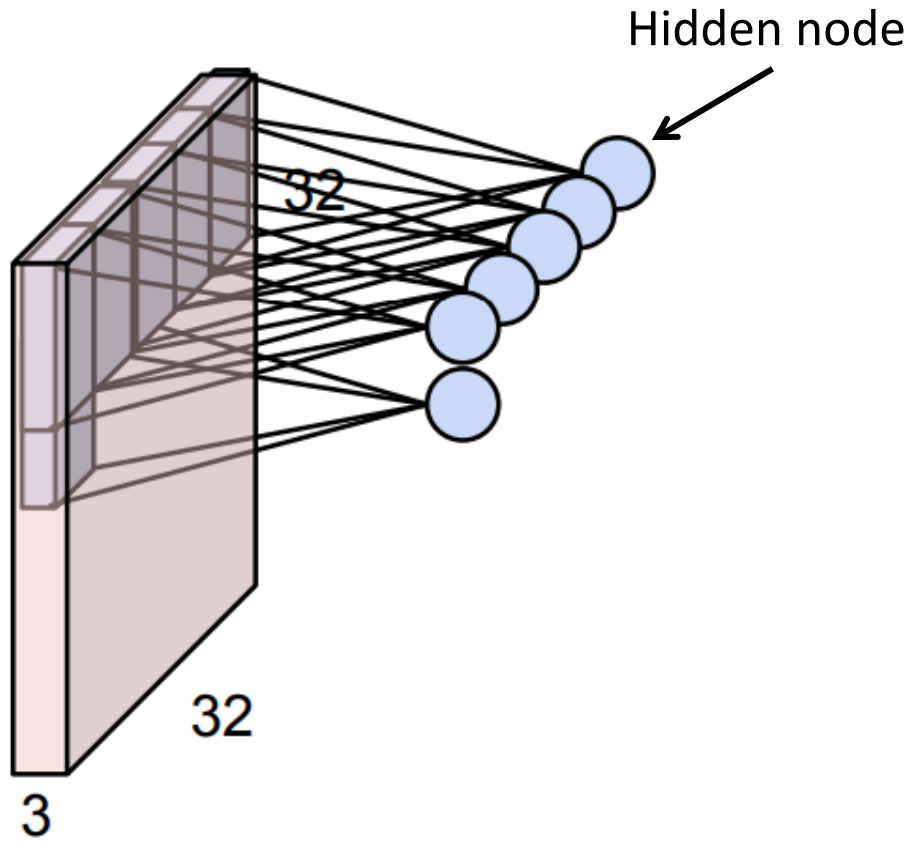
Convolution Network



- Unlike MLP, the concept of a filter appears on convolution layer.

Convolution Network – filter operation





- By performing a filter operation while rotating each image, a **hidden node** is generated for each part

(Note) Filter

-1	0	+1
-2	0	+2
-1	0	+1

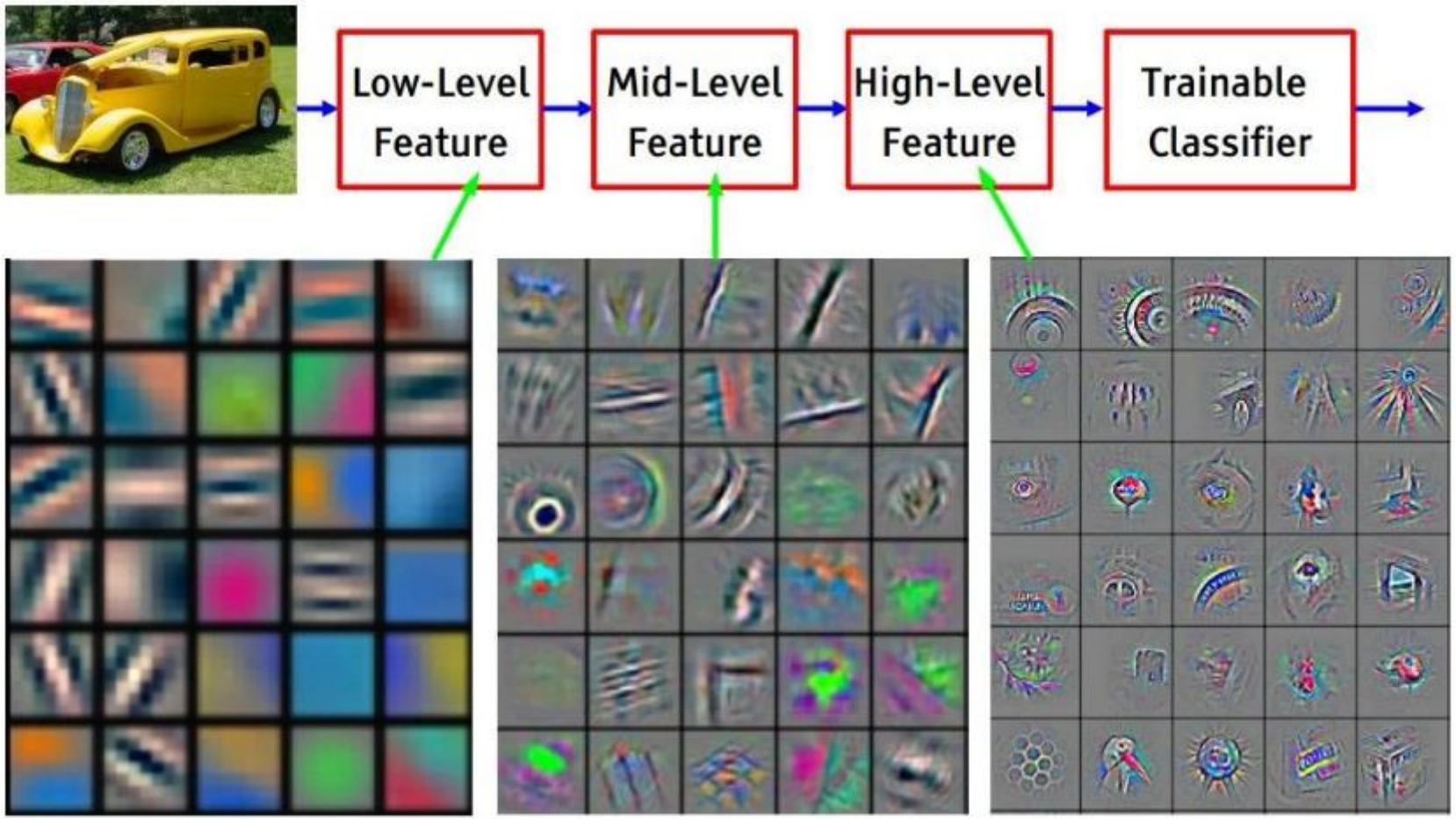
Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

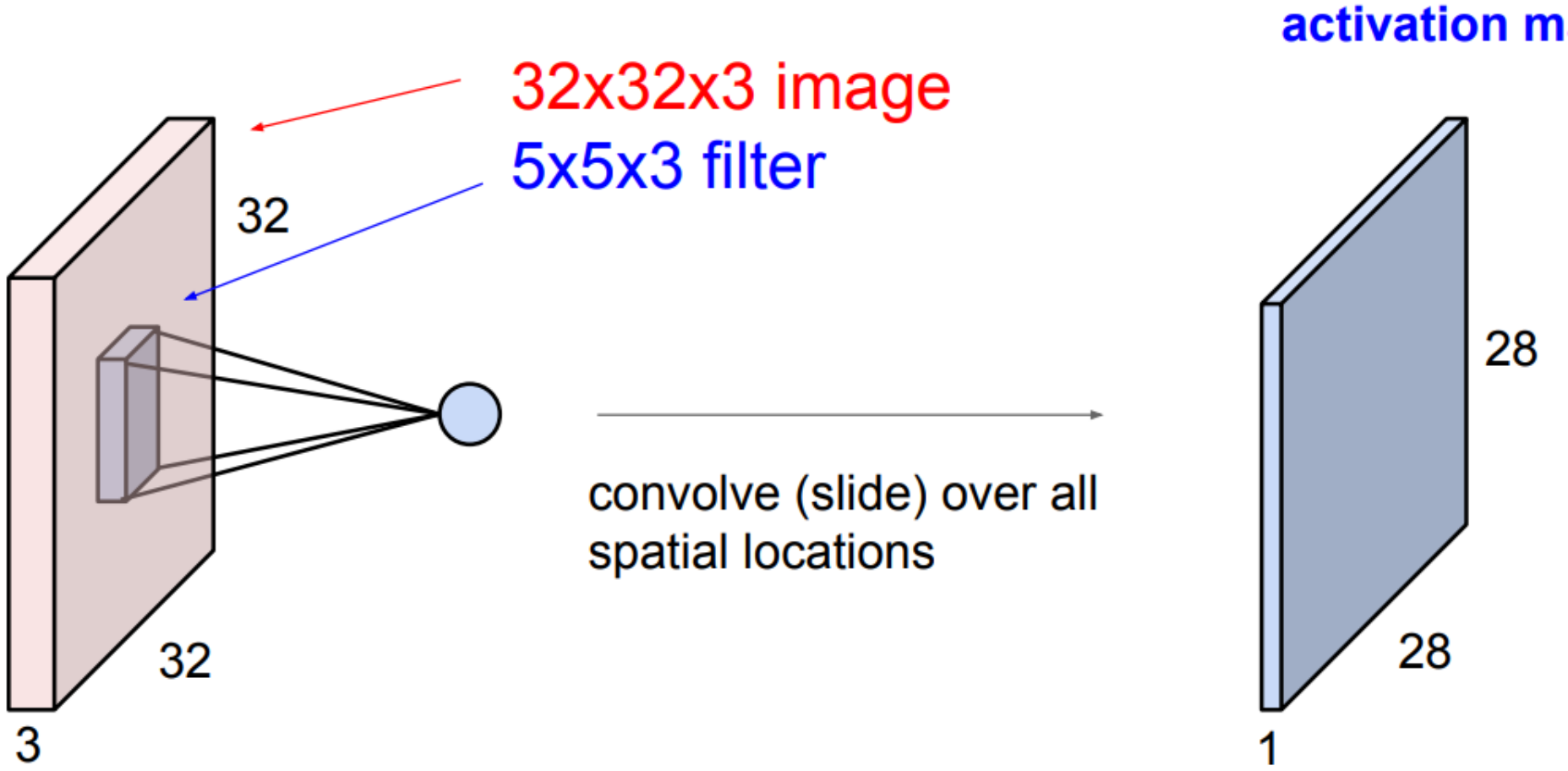
- Sobel filter, which is the traditional method to get edges.

(Note) Filter



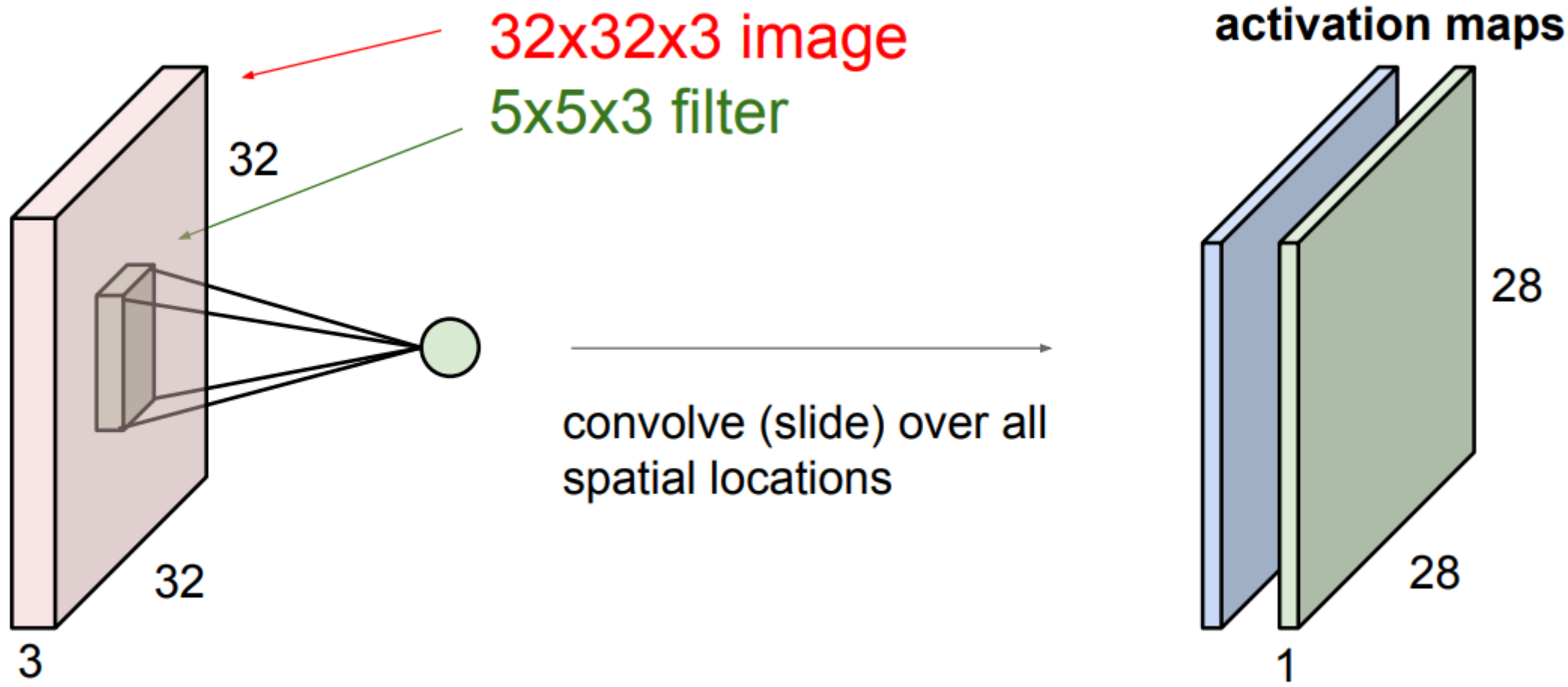
- In low-level feature, we can see filters which is like Sobel Filter

Convolution Network



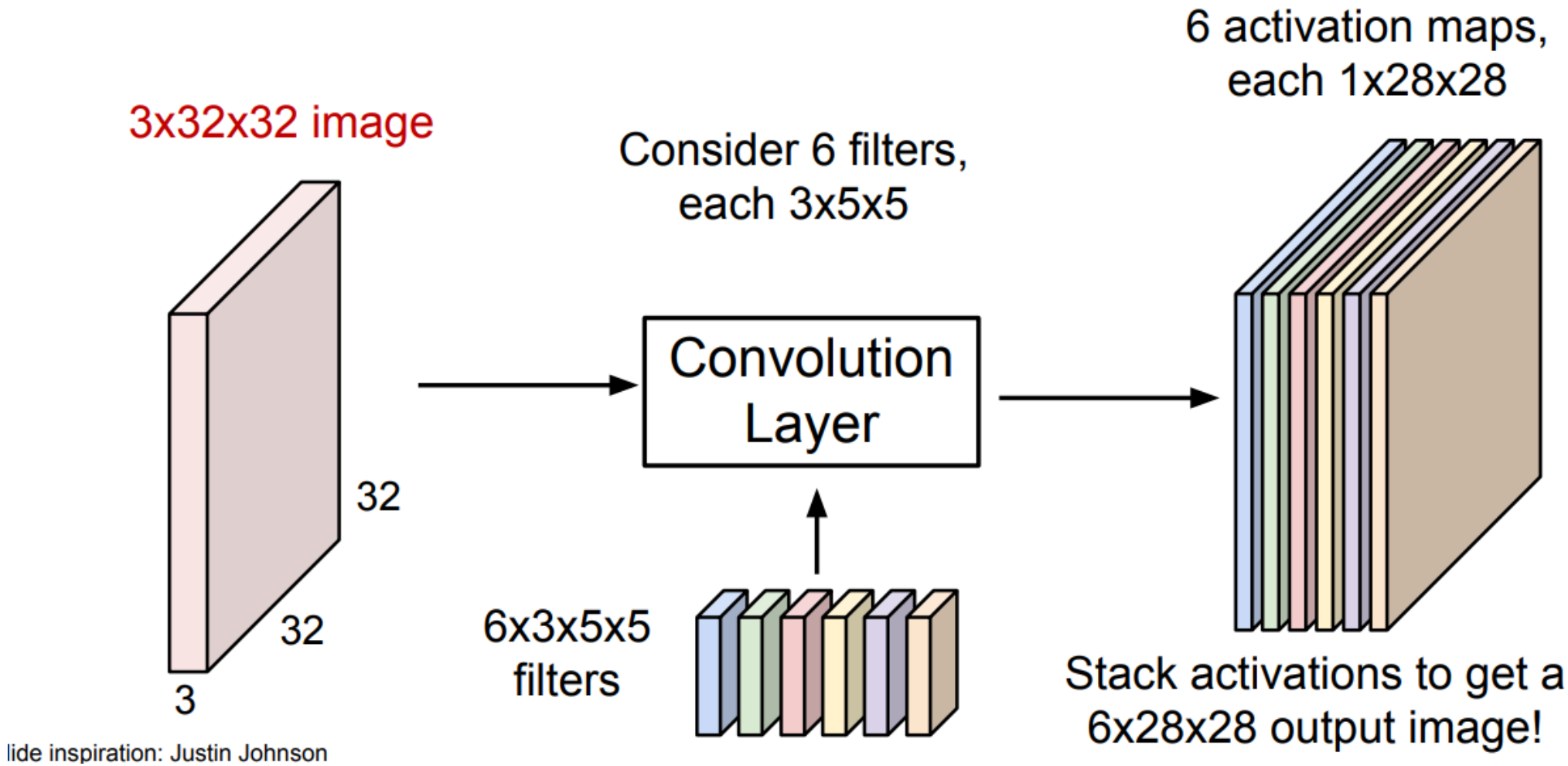
- Hidden nodes gather to form an **activation map.**

Convolution Network



- We can use more filter, then they make other **activation maps** which contains new contents.

Convolution Network

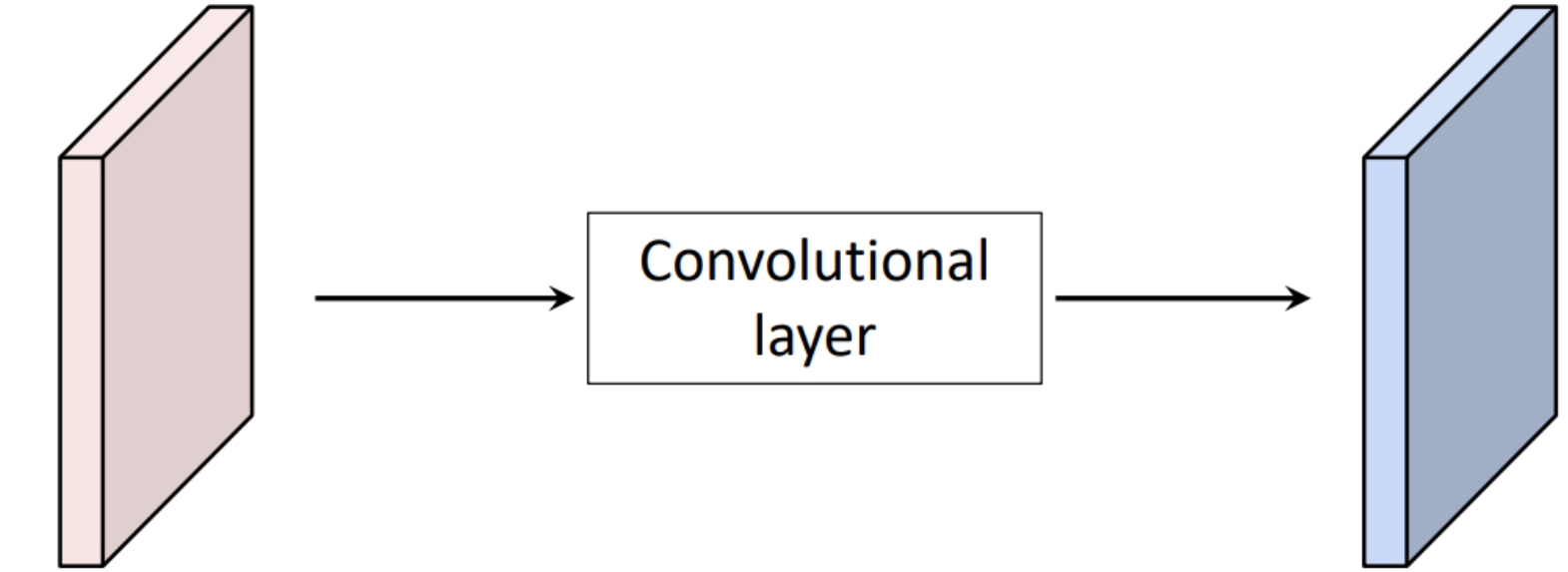


lide inspiration: Justin Johnson

- The overall convolution network has the following structure.

Convolution Network

The number of parameters in convolutional layer



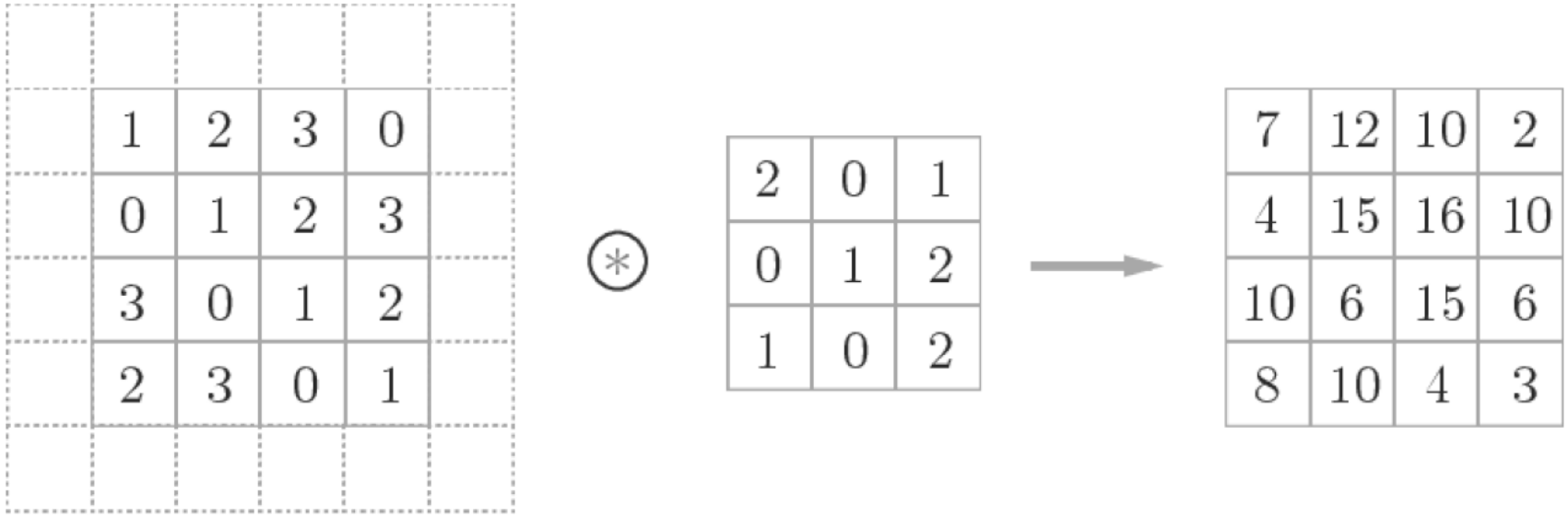
Input
$H_1 \times W_1 \times C_1$

Weight
C_2 filters of $F_h \times F_w \times C_1$

Output
$H_2 \times W_2 \times C_2$

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

Convolution Network - padding



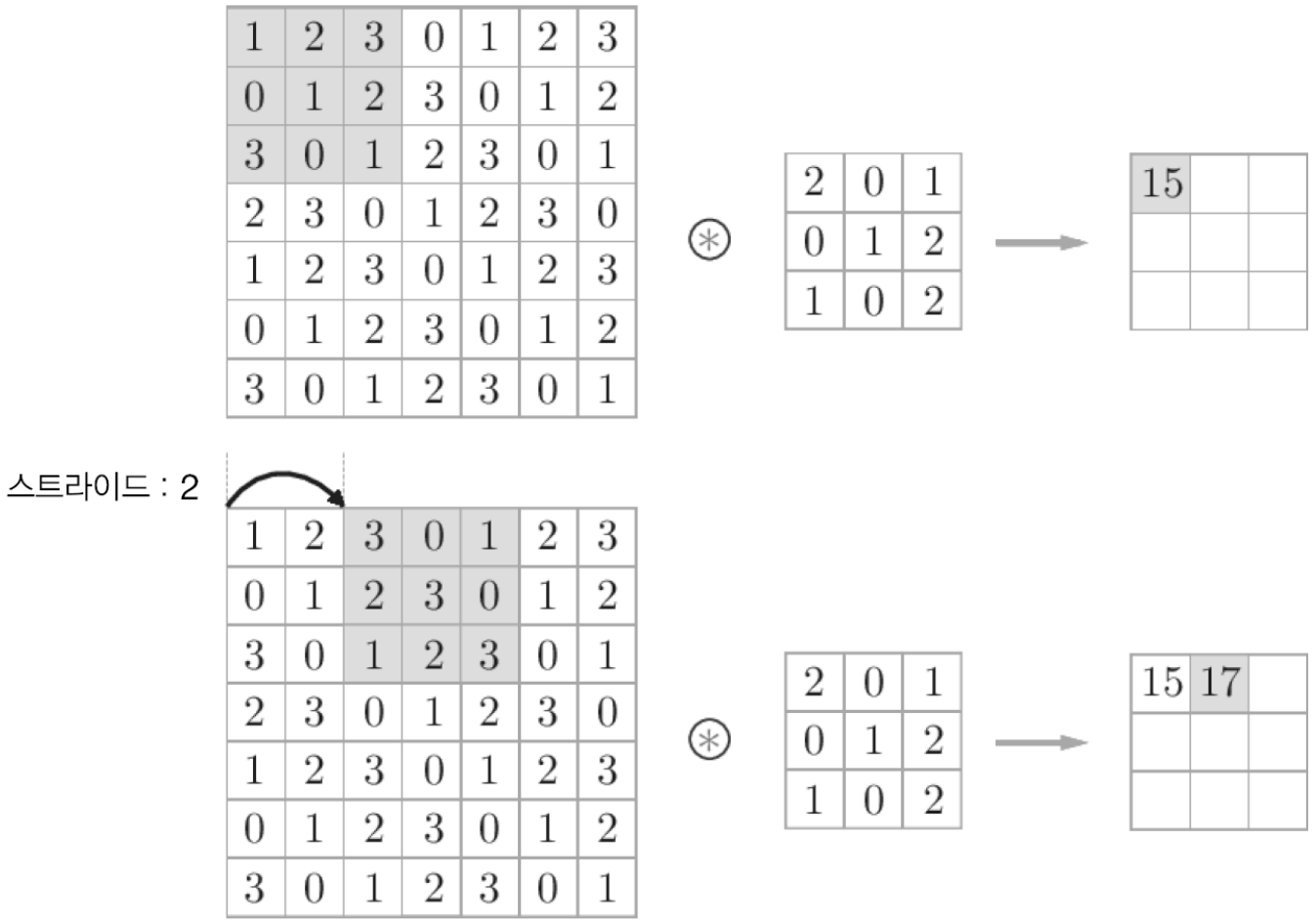
(4, 4)
입력 데이터(패딩 : 1)

(3, 3)
필터

(4, 4)
출력 데이터

- **Padding** means filling the surroundings with zero to maintain the image size.

Convolution Network - stride

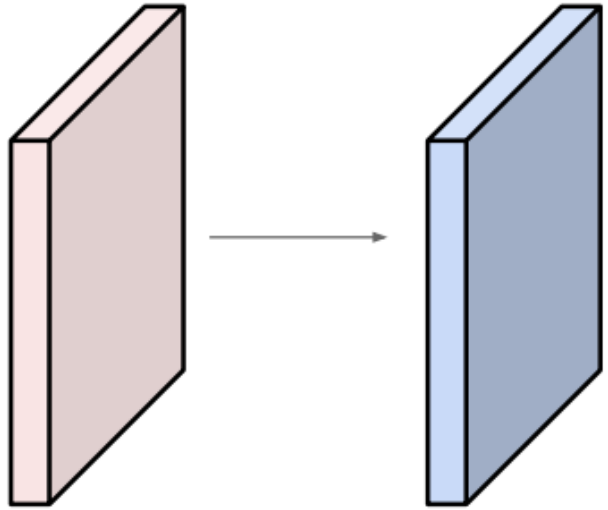


- **stride** determines the interval at which the filter is applied. (default: 1)

Convolution Network – filter size

Examples time:

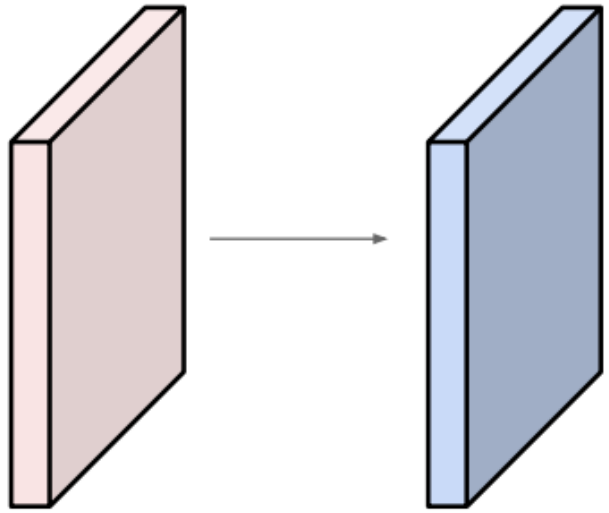
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output volume size: ?

Convolution Network – filter size

Examples time:



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

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

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

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

Receptive field

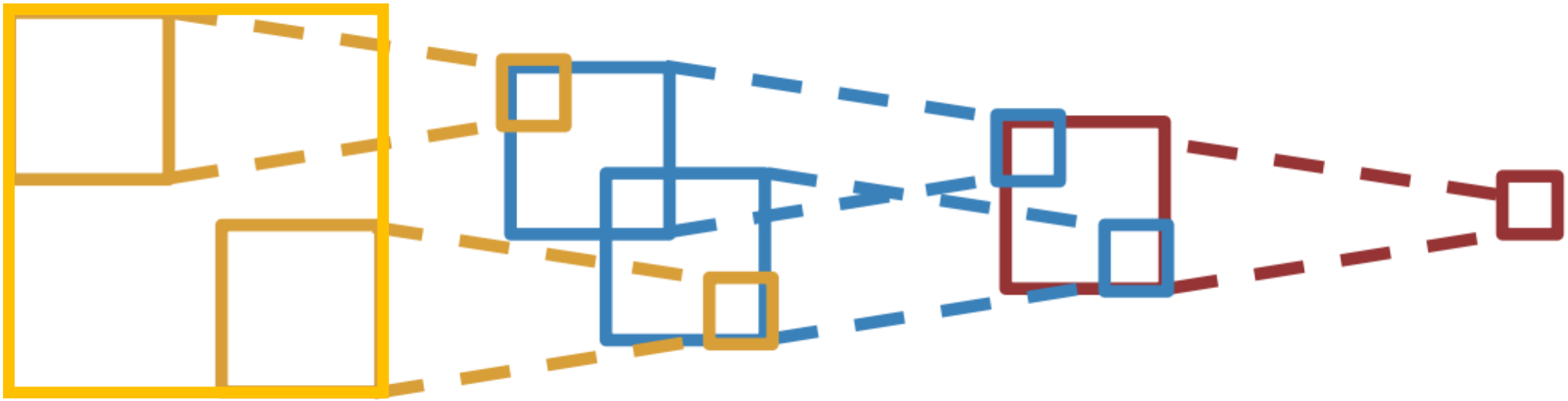
For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input

Input Output

- **Kernel Size:** filter size
- **Receptive field:** image field that affects the hidden node or output node.

Receptive field

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)

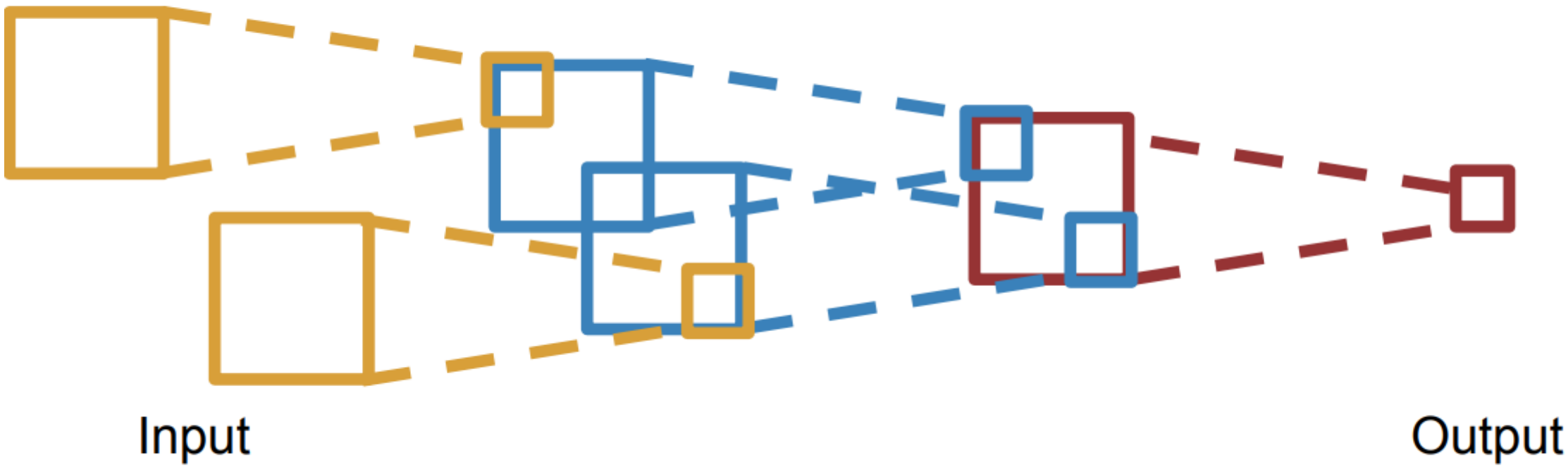


Be careful – ”receptive field in the input” vs. “receptive field in the previous layer”

- If multiple convolution layers are stacked, the receive field becomes larger.

Receptive field

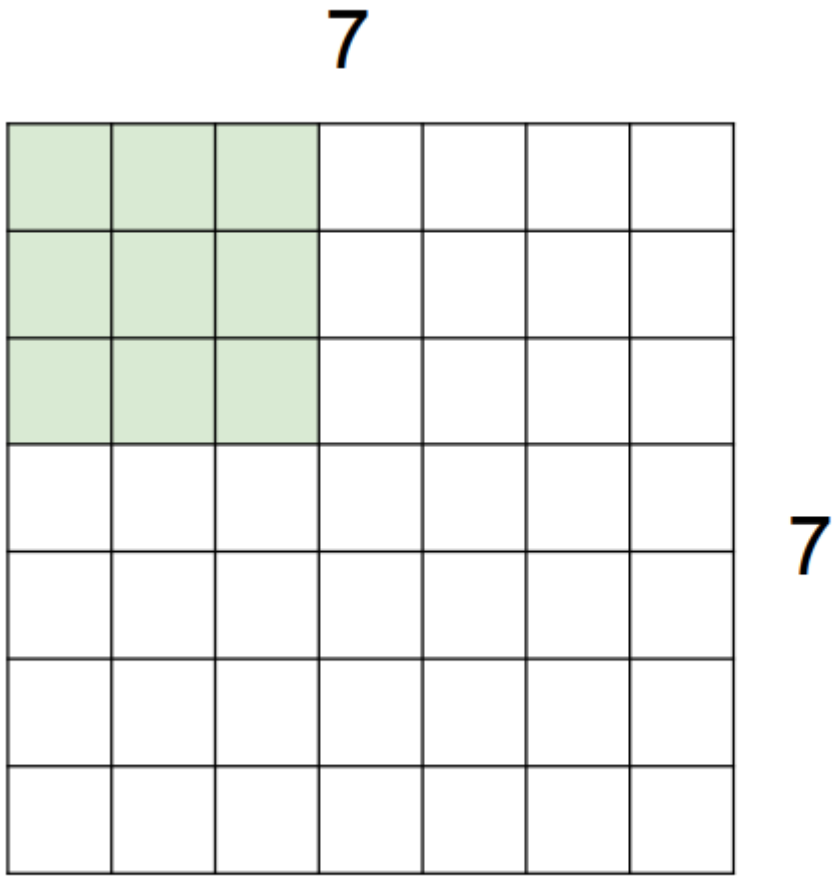
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Problem

1. Output may not see whole image, so we should reduce image size.

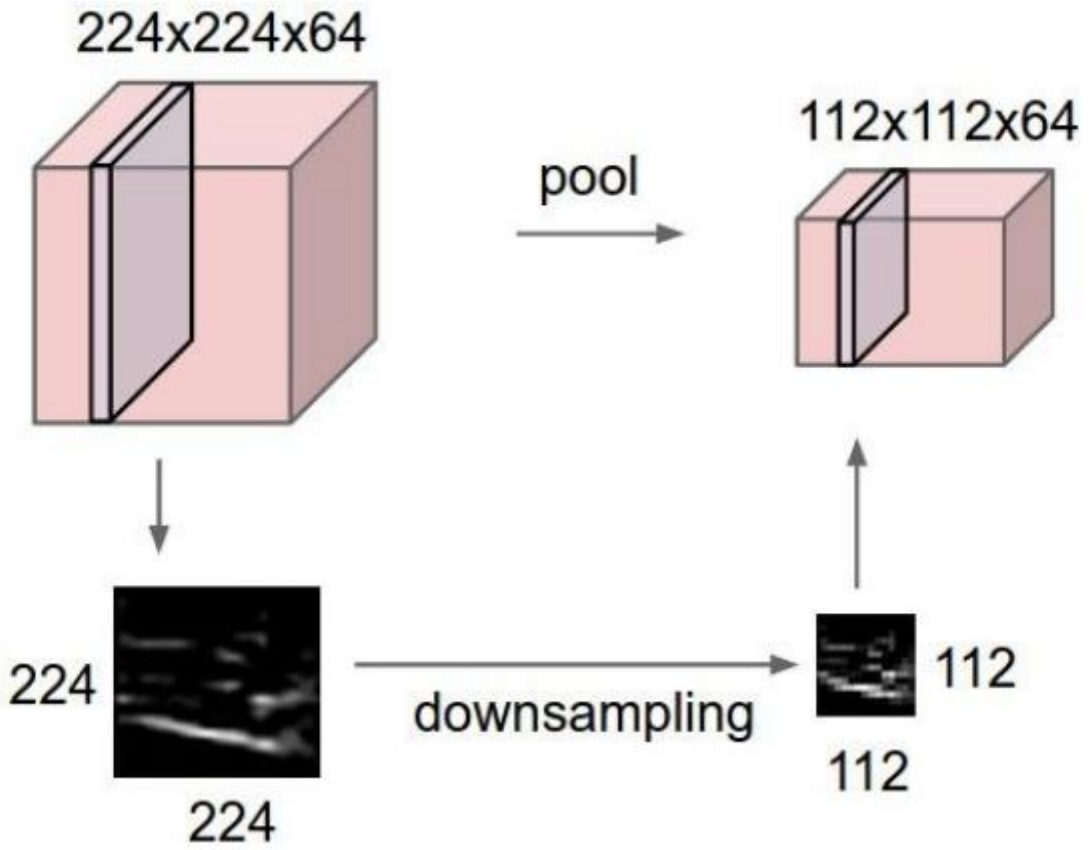
Receptive field - Solution



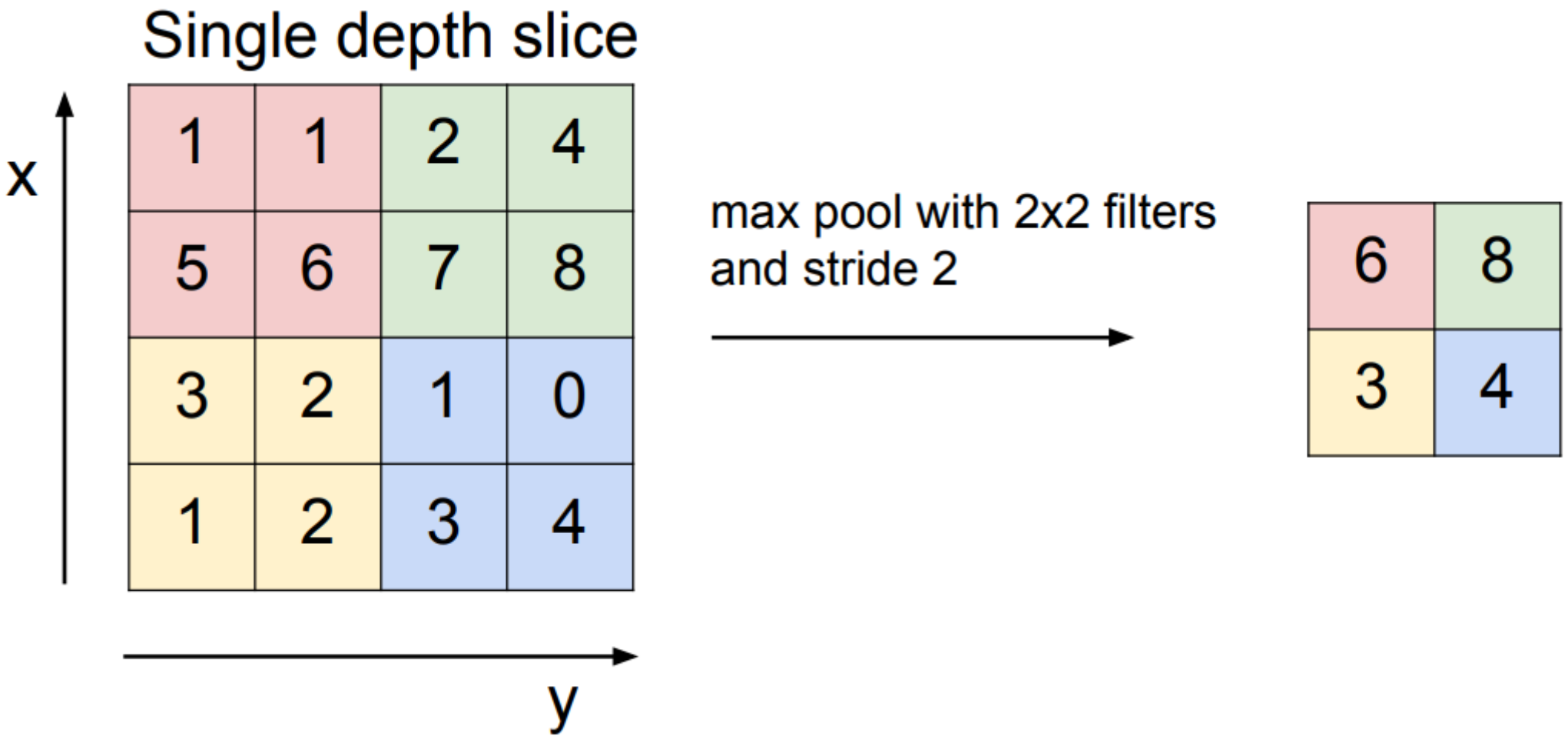
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Pooling Layer

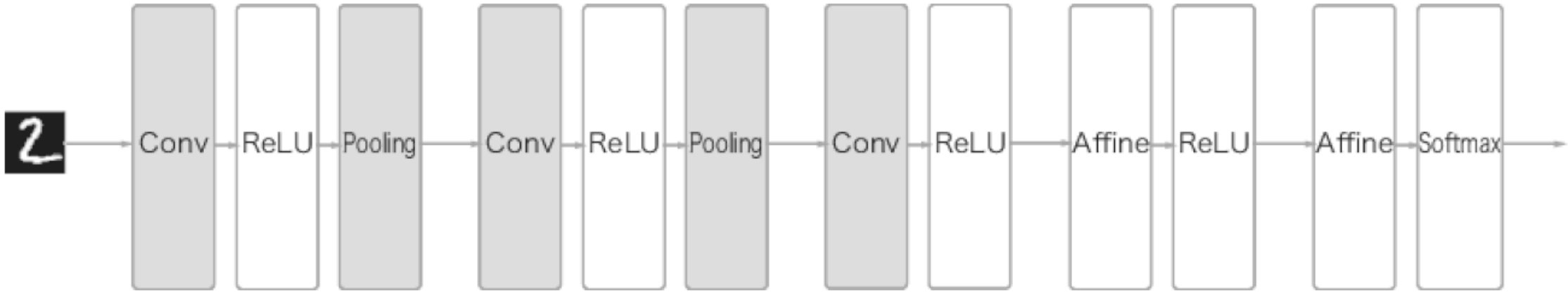
- makes the representations smaller and more manageable
- operates over each activation map independently



Pooling Layer



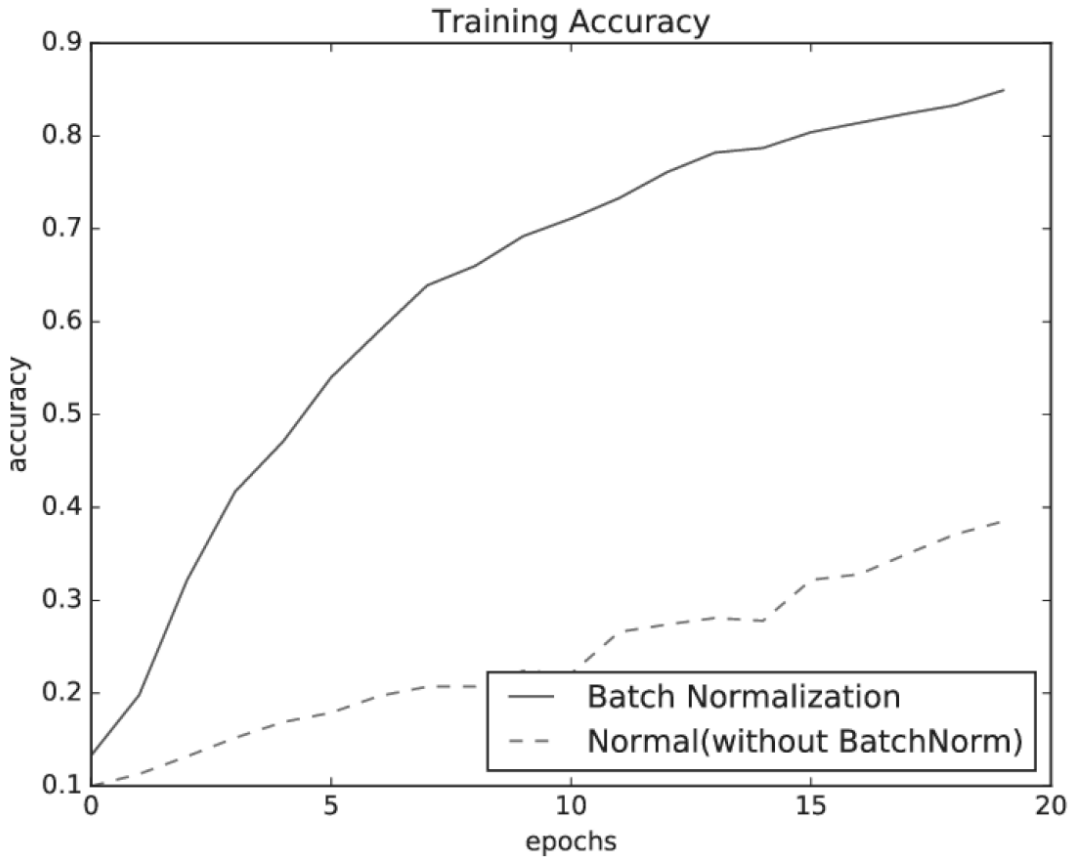
Pooling Layer



Batch Normalization Layer

Pros

1. Learning speed becomes faster.
2. It does not highly depend on Initialization.
3. Suppress overfitting



Batch Normalization Layer

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

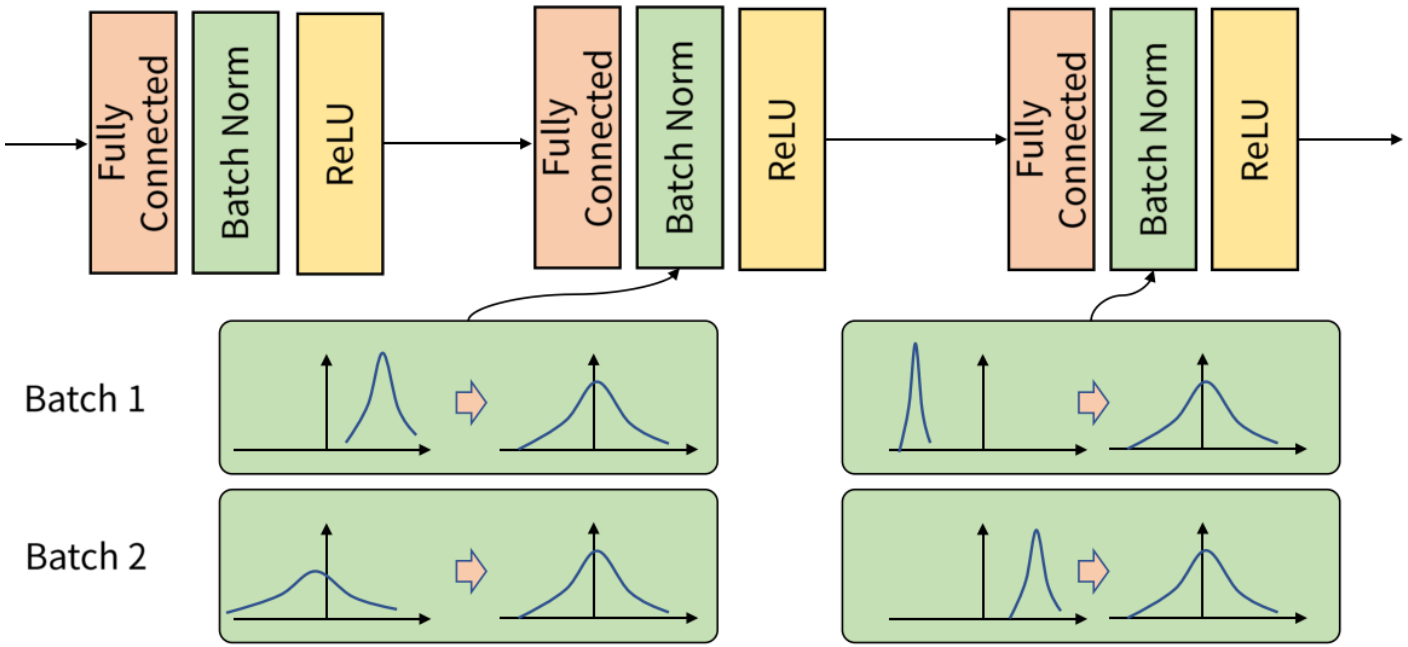
Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D



When parameter is changed, hidden layer distribution would be changed, and it makes training harder.

To keep the distribution constant, we make output normalized. (e.g., mean=0, std = 1)

For each batch, a mean and a variance are obtained separately to computation effectively.

Batch Normalization Layer

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

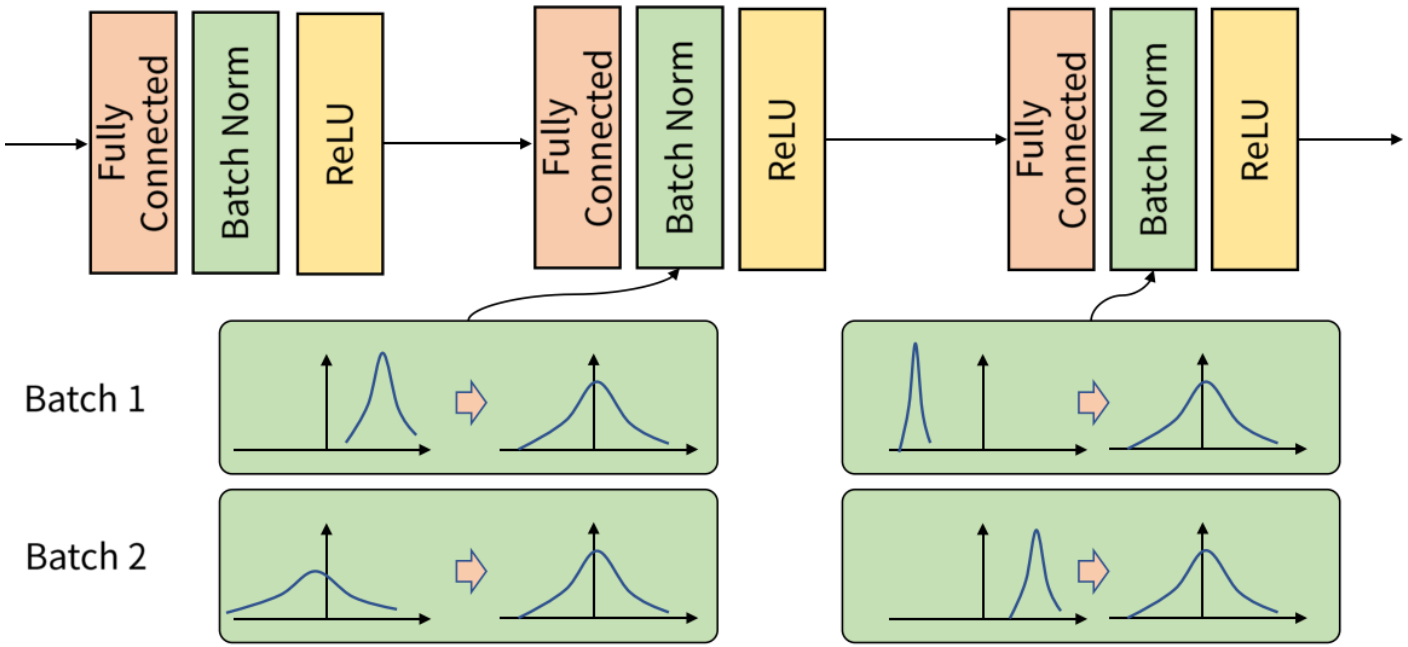
Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D



Problem

- Too much constraints -> make distribution more flexible.

Batch Normalization Layer

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$,
$\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization Layer – test time

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

During testing batchnorm becomes a linear operator!
Can be fused with the previous fully-connected or conv layer

$$\mu_j = \text{(Running) average of values seen during training}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \text{(Running) average of values seen during training}$$

Per-channel var, shape is D

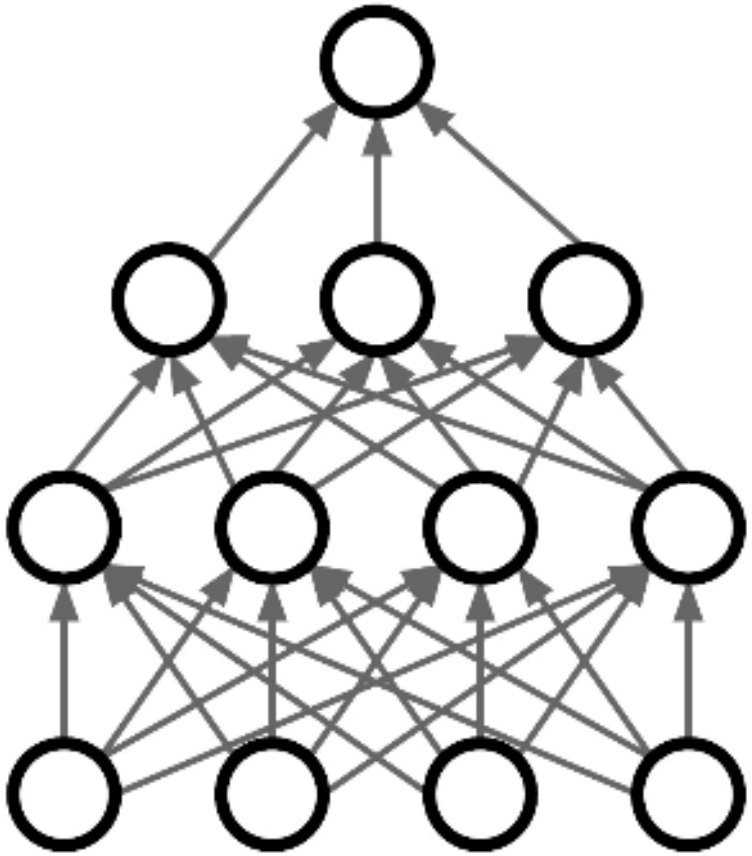
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

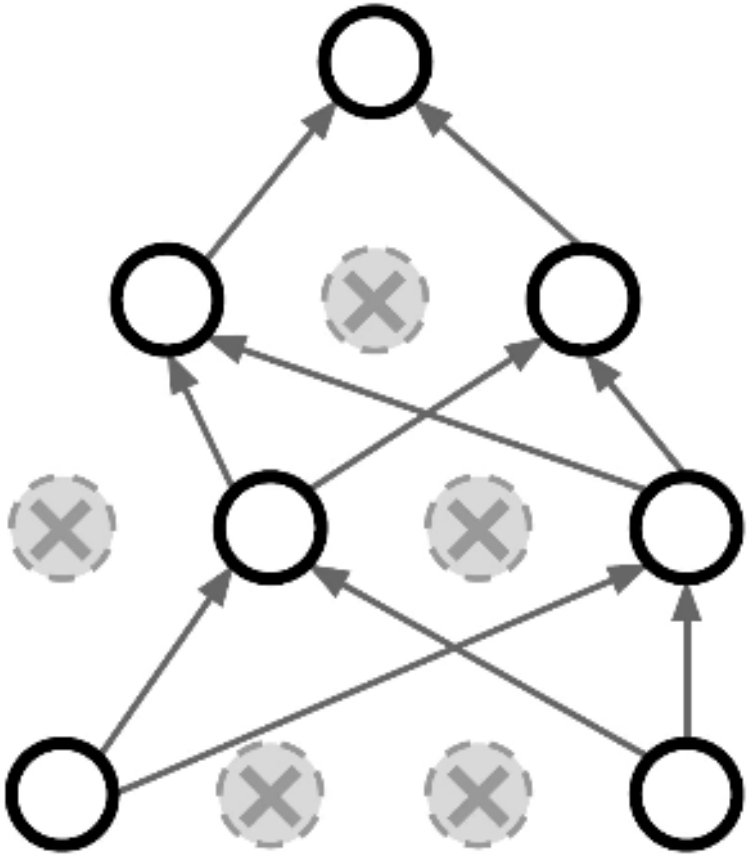
Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

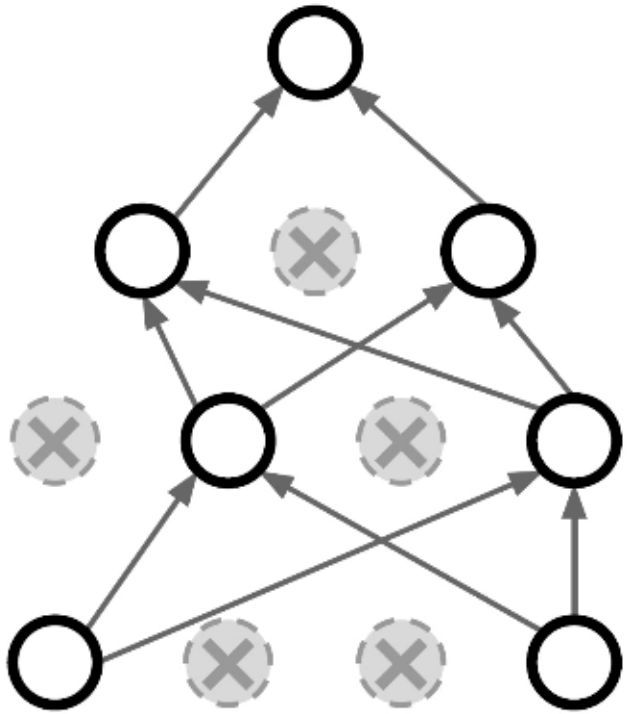
Output, Shape is N x D

Recall) Regularization: Dropout





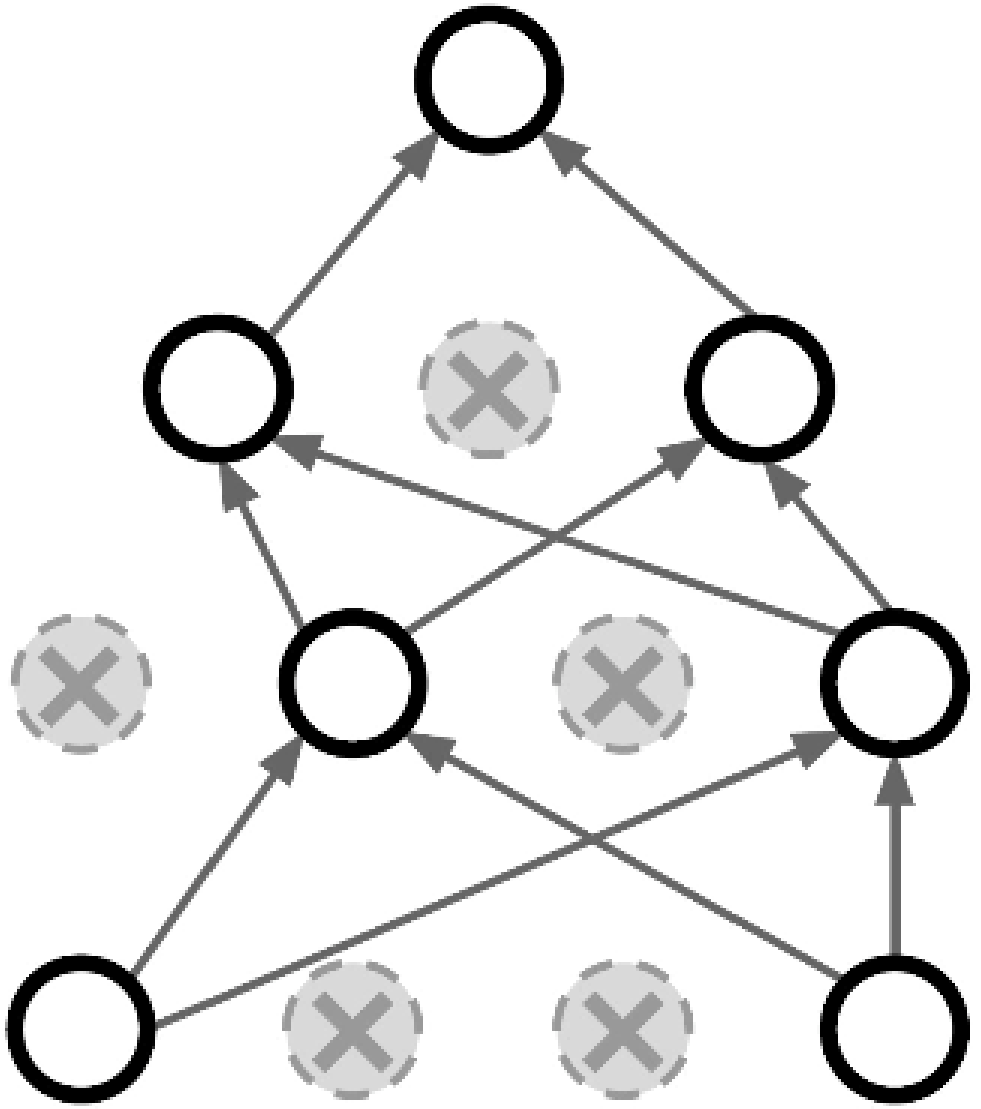
Recall) Regularization: Dropout



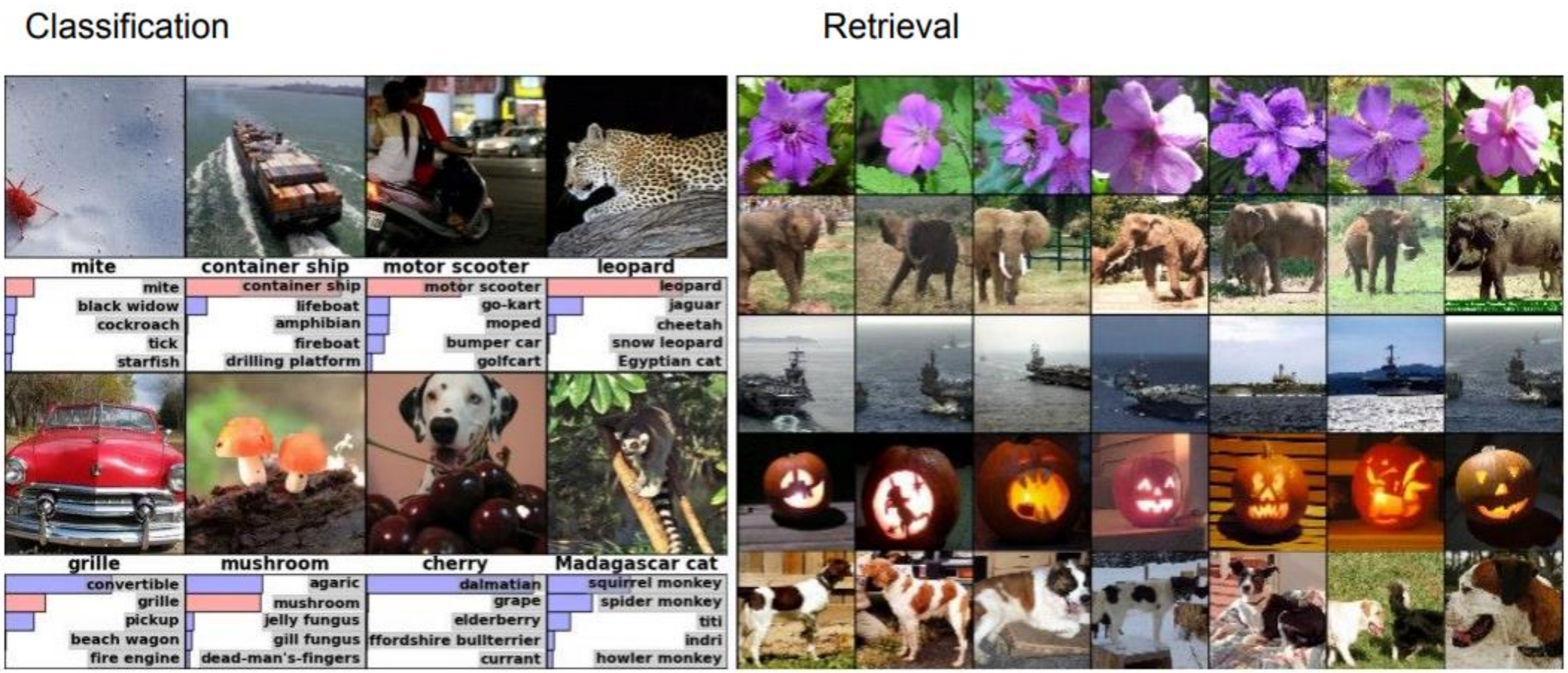
Forces the network to have a redundant representation;
Prevents co-adaptation of features

Recall) Regularization: Dropout

Dropout이 파라미터를 공유하는 큰 Ensemble 모델을 학습하는 것과 같다고 보는 관점도 있다.



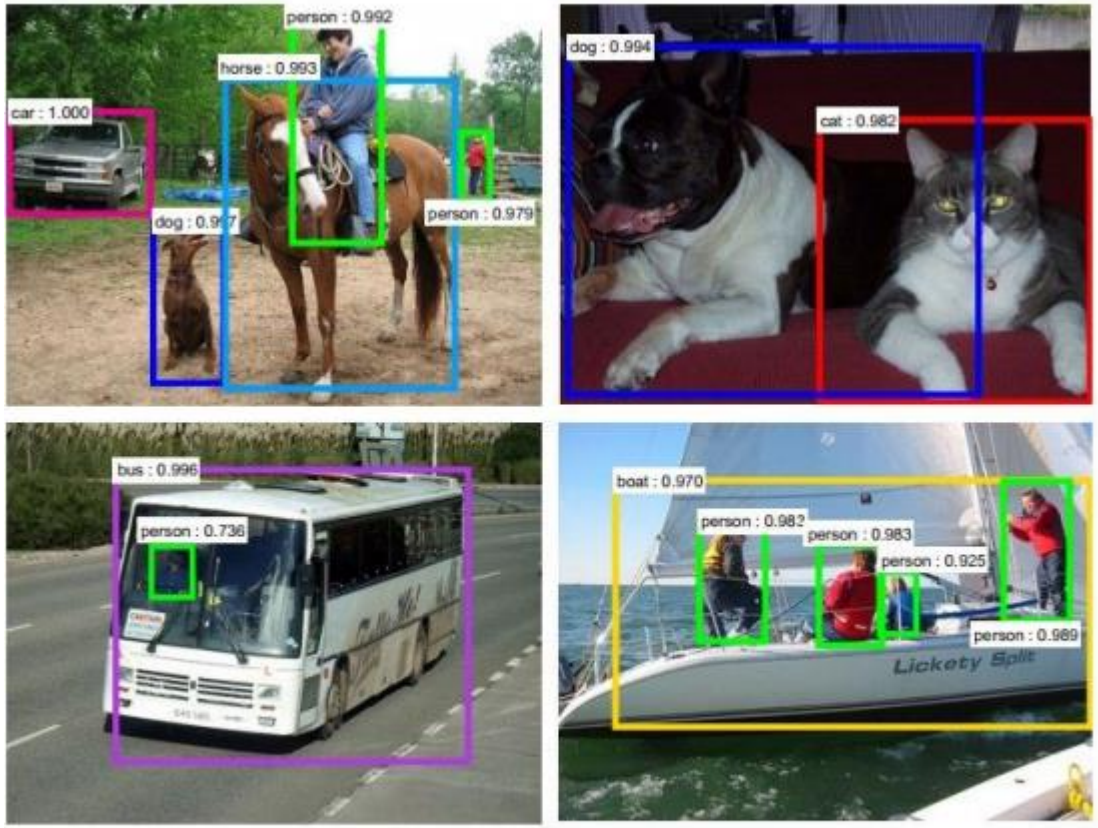
Example of Convolution Network



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Example of Convolution Network

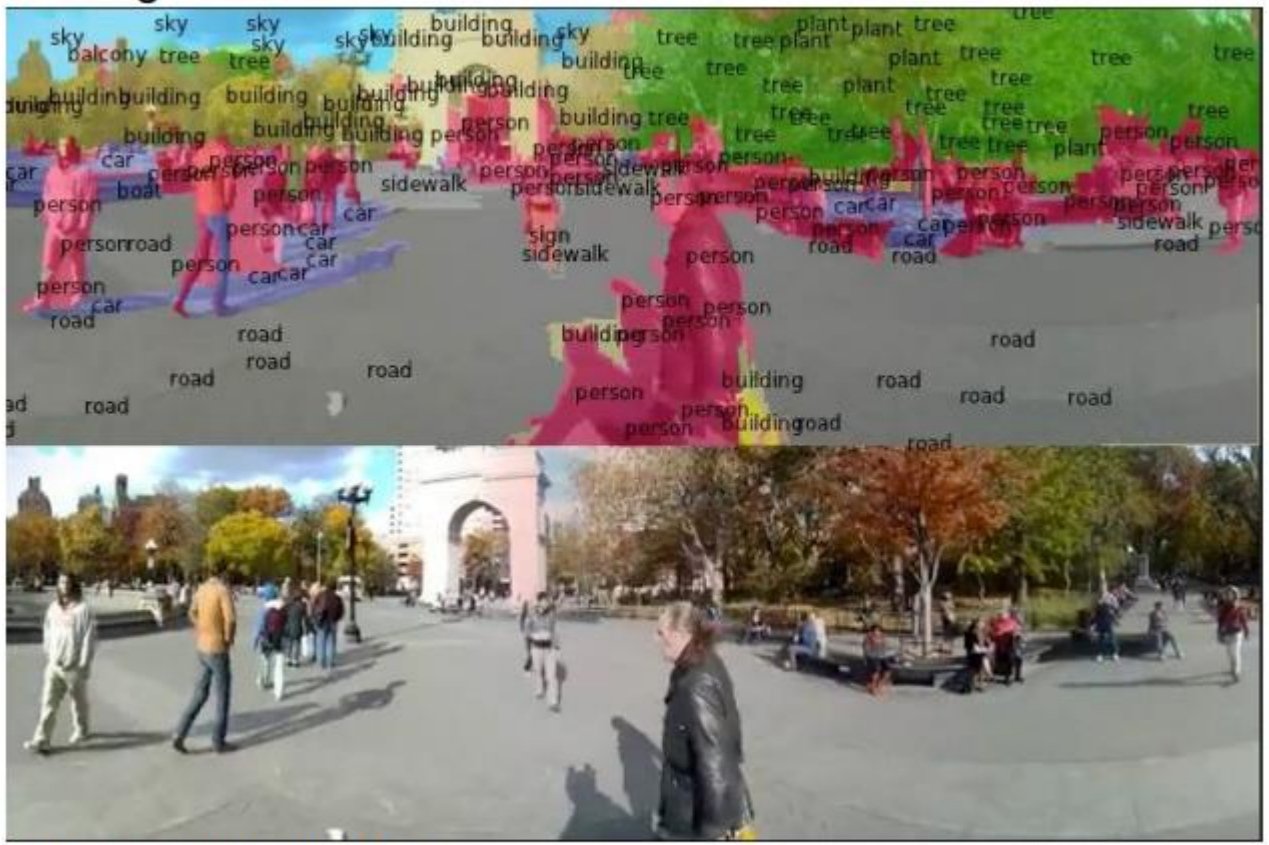
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]

Example of Convolution Network

Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Example of Convolution Network

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Example of Convolution Network

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

Example of Convolution Network

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

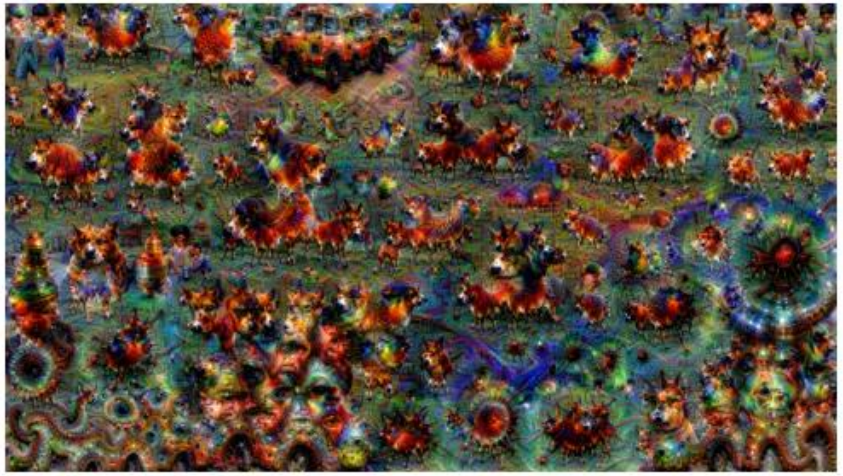
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

Example of Convolution Network

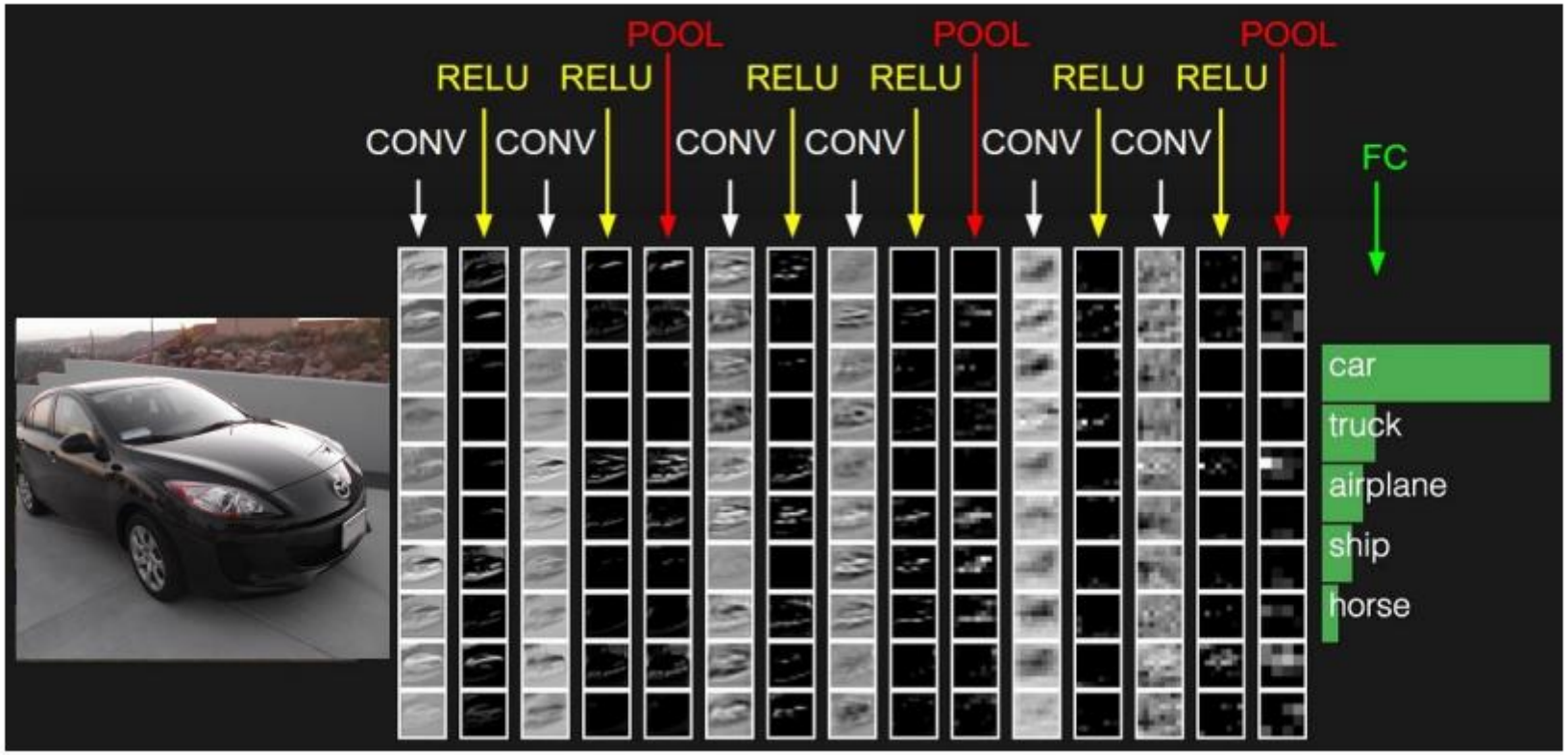


Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

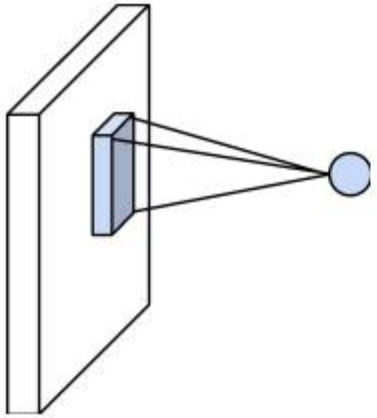
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

CNN Architecture

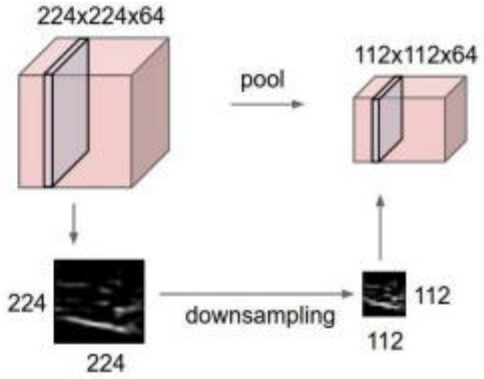


Component of CNNs

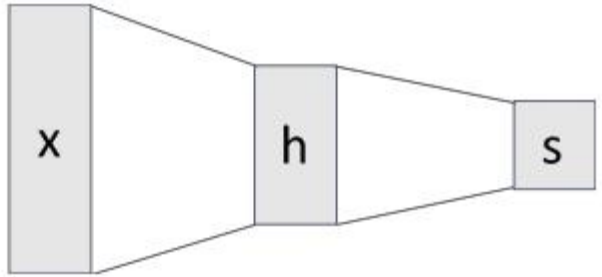
Convolution Layers



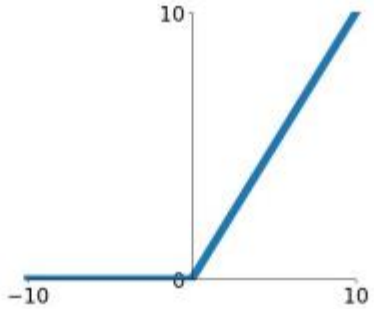
Pooling Layers



Fully-Connected Layers



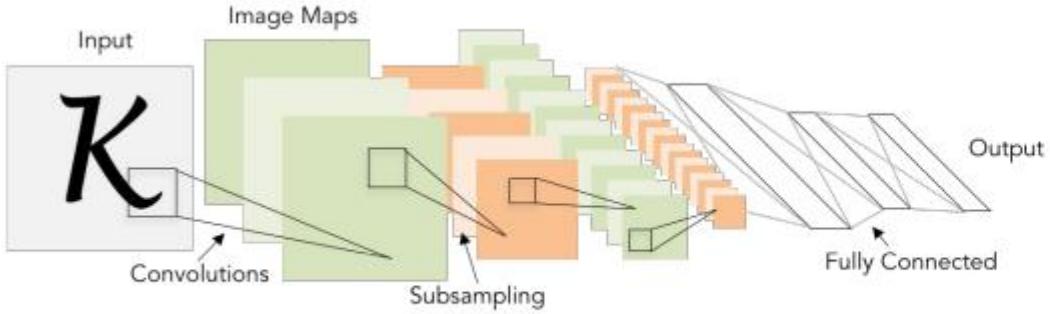
Activation Function

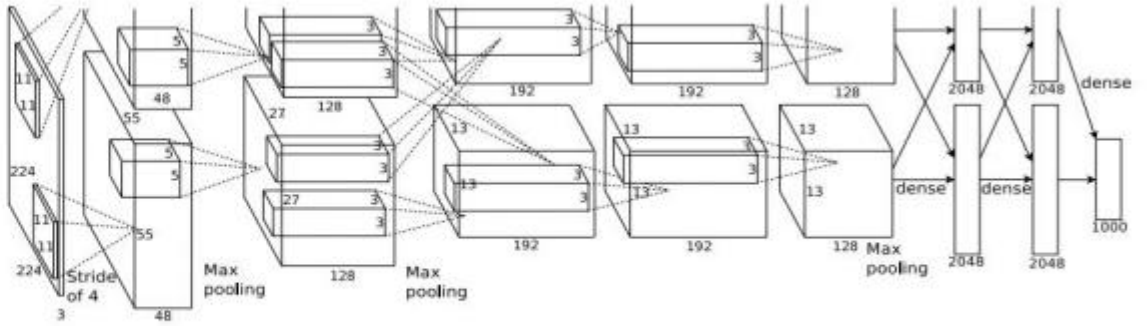


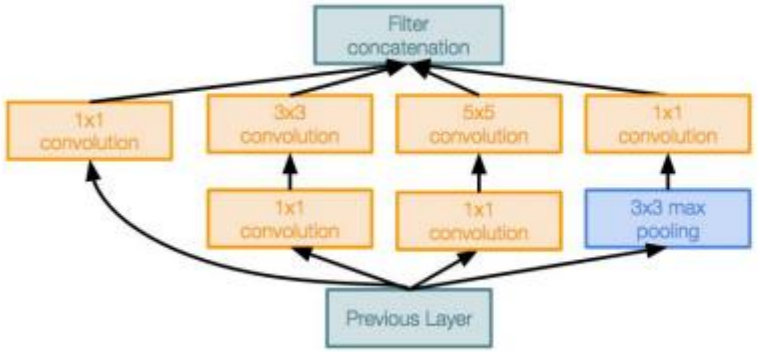
Normalization

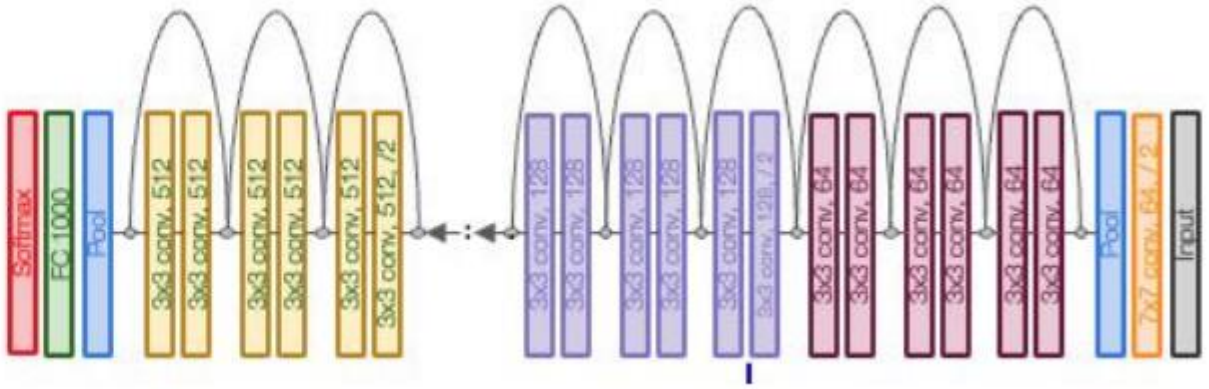
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

CNN Architecture

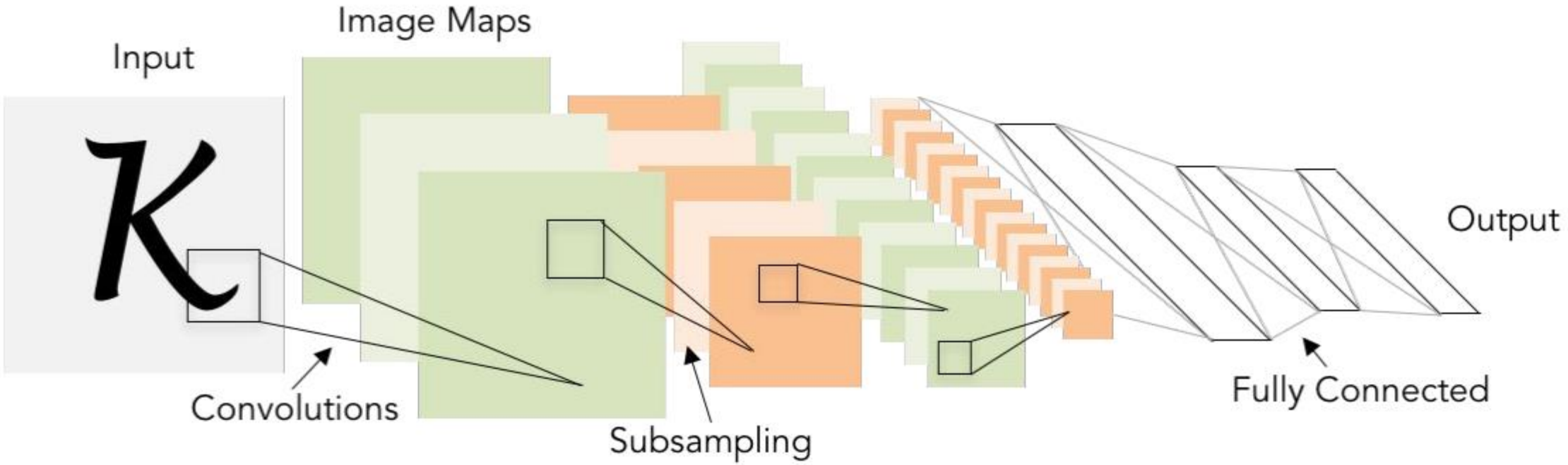






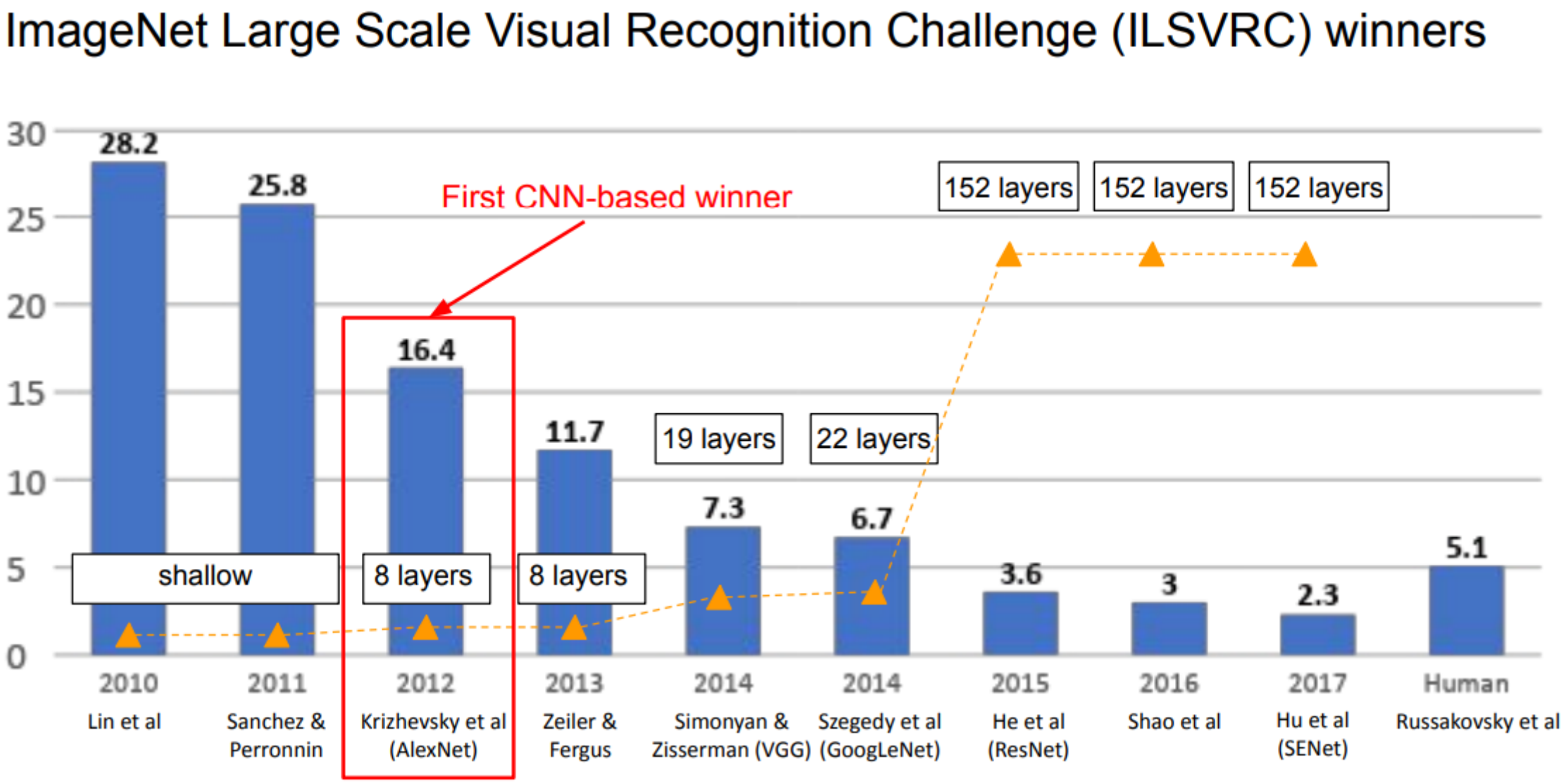


LeNet



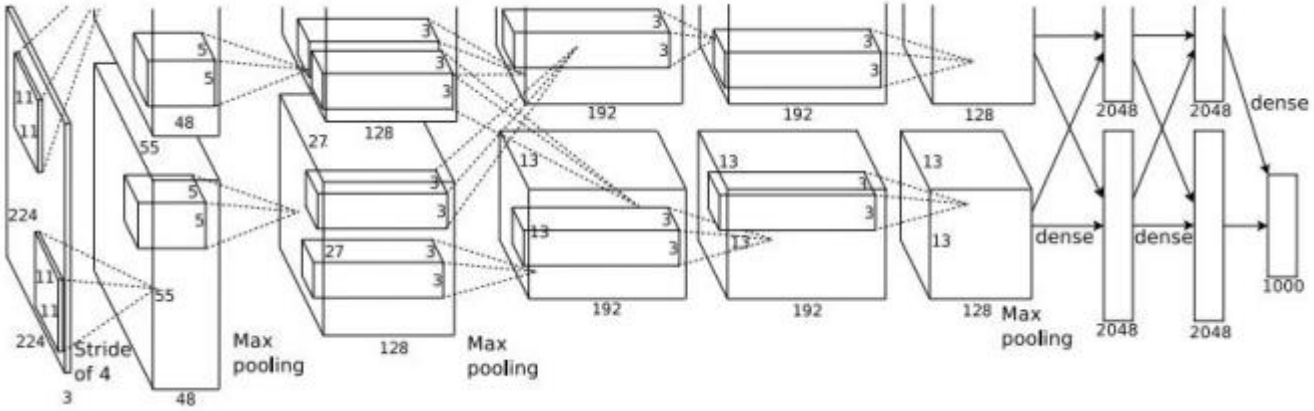
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

ImageNet Challenge



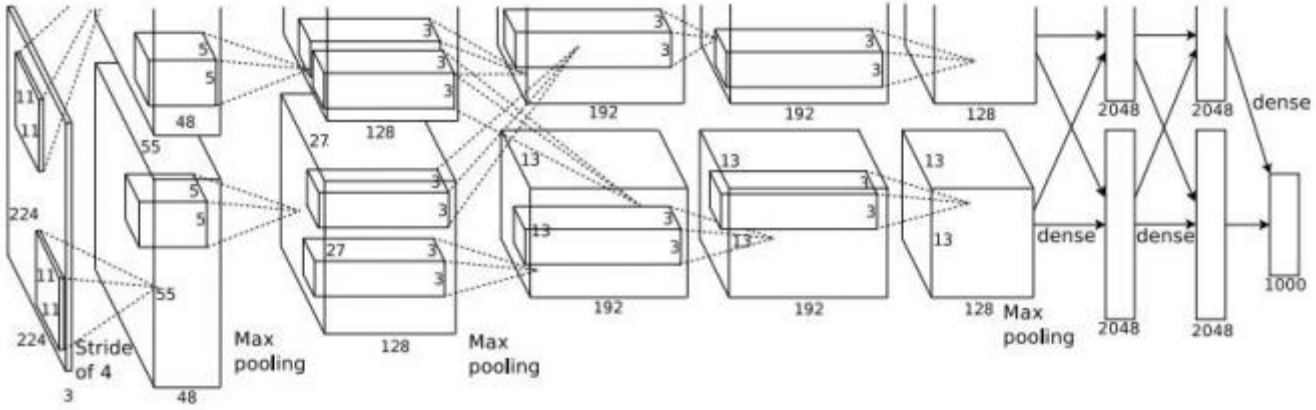
AlexNet

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



Usually, activation appears after convolution layer or fully connected layer

AlexNet



Input: 227x227x3 images

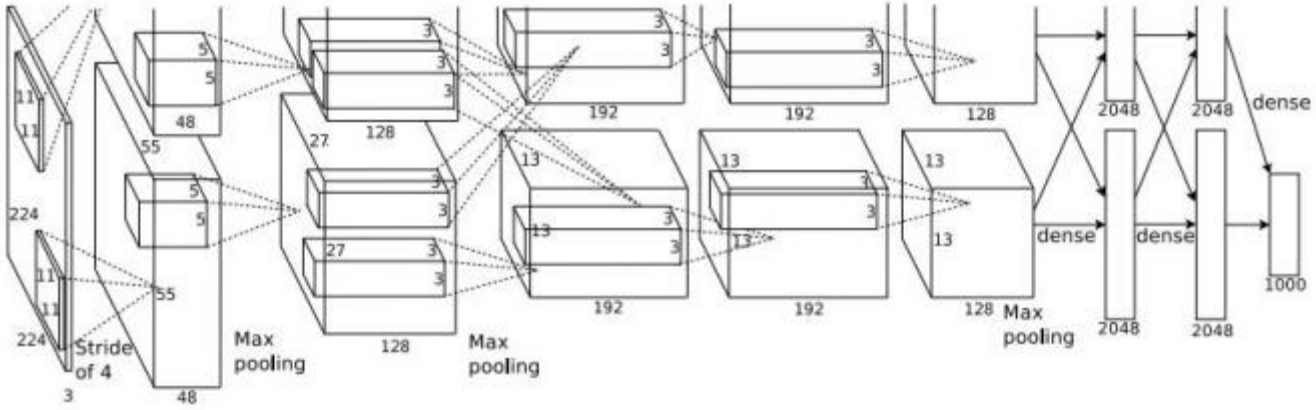
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

W' = (W - F + 2P) / S + 1

AlexNet



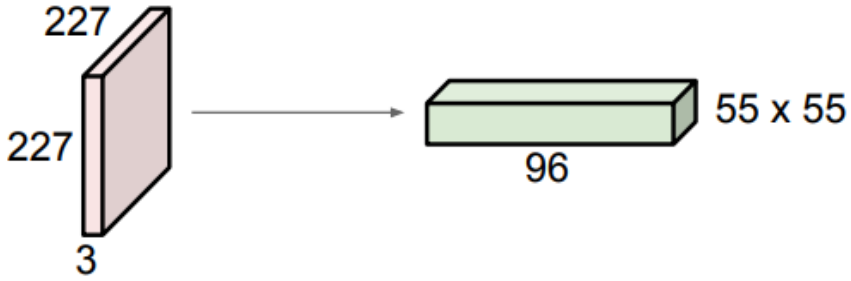
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

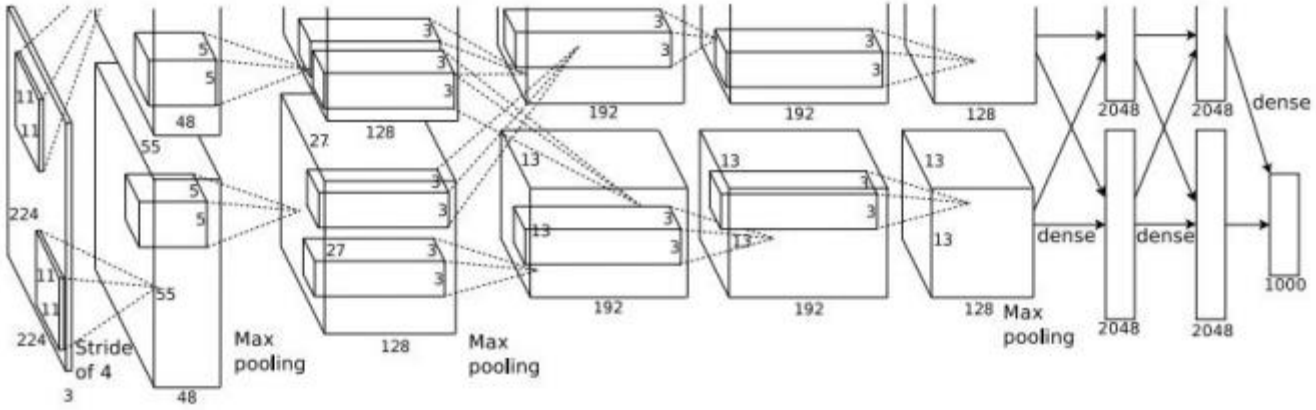
=>

Output volume **[55x55x96]**

W' = (W - F + 2P) / S + 1



AlexNet



Input: 227x227x3 images
After CONV1: 55x55x96

$W' = (W - F + 2P) / S + 1$

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

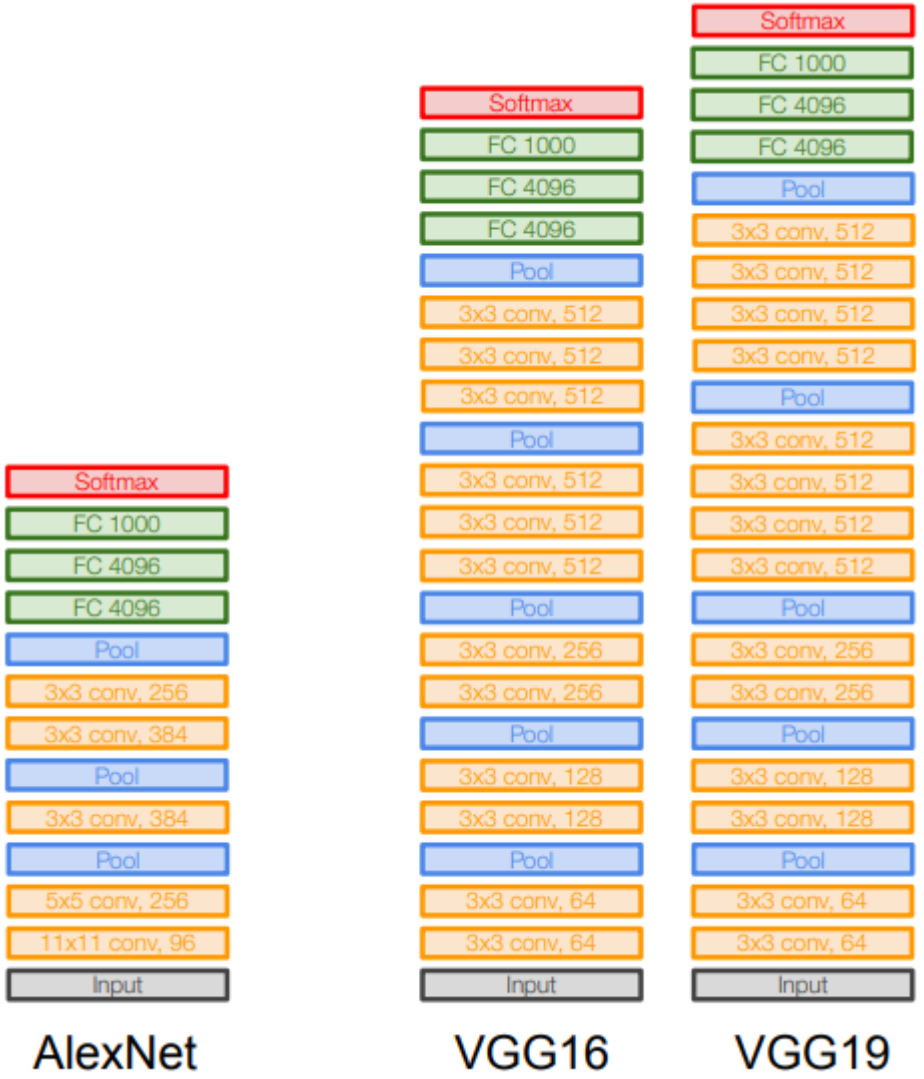
VGGNet

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

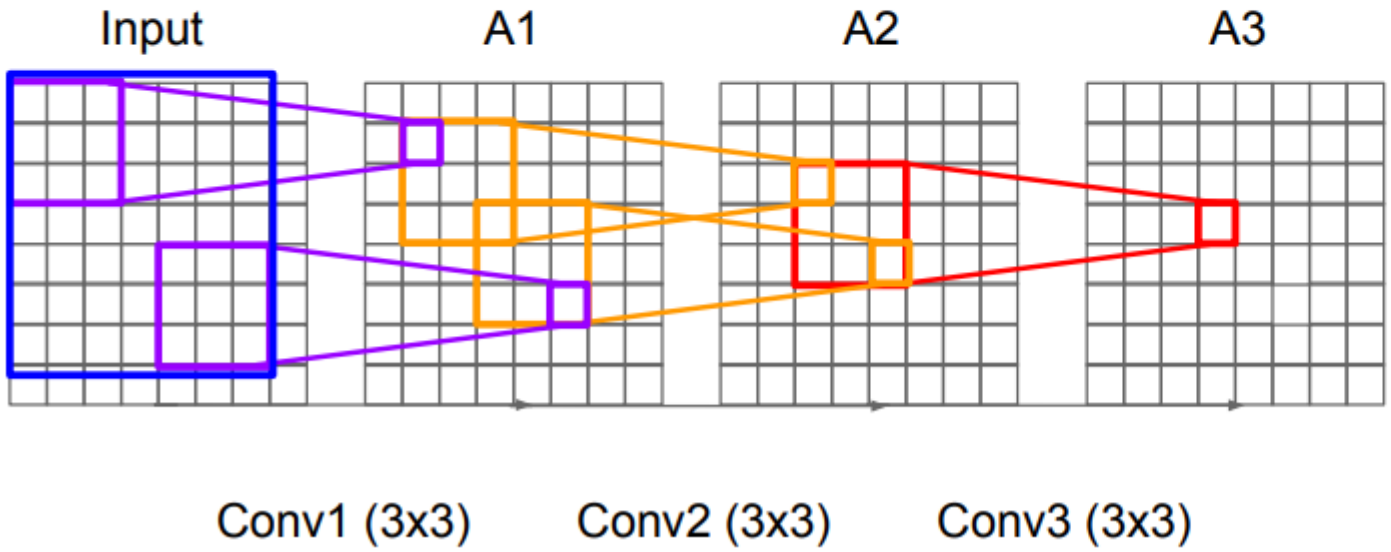
11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



VGGNet

Q: Why use smaller filters? (3x3 conv)

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



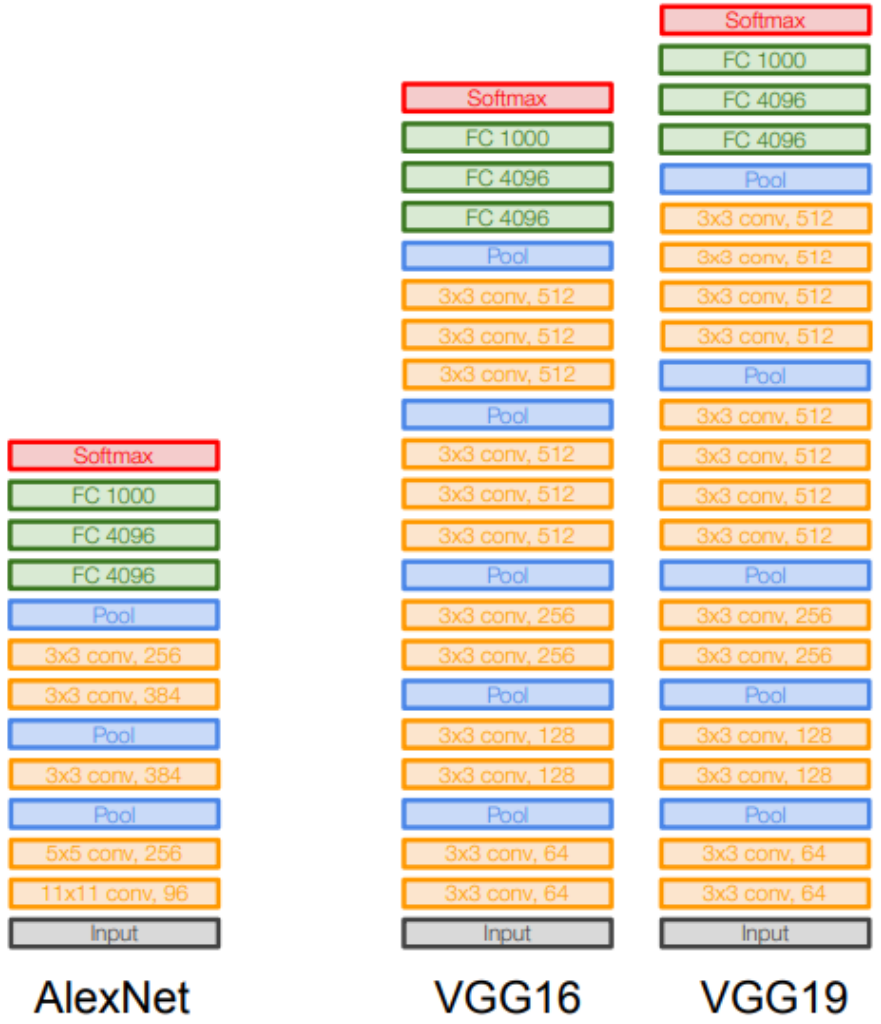
VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

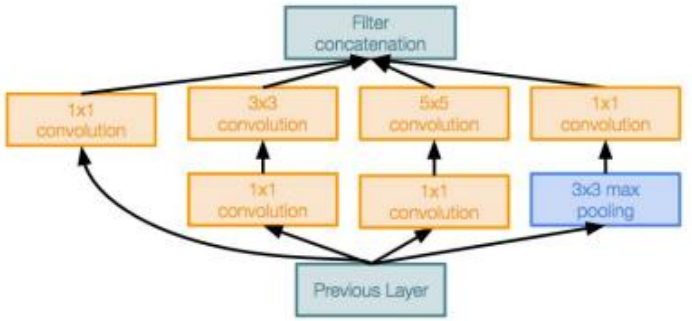
And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer



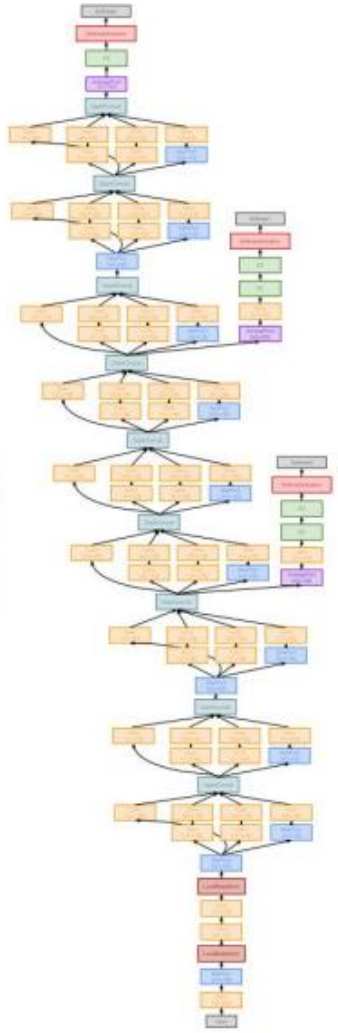
GoogleNet

Deeper networks, with computational efficiency

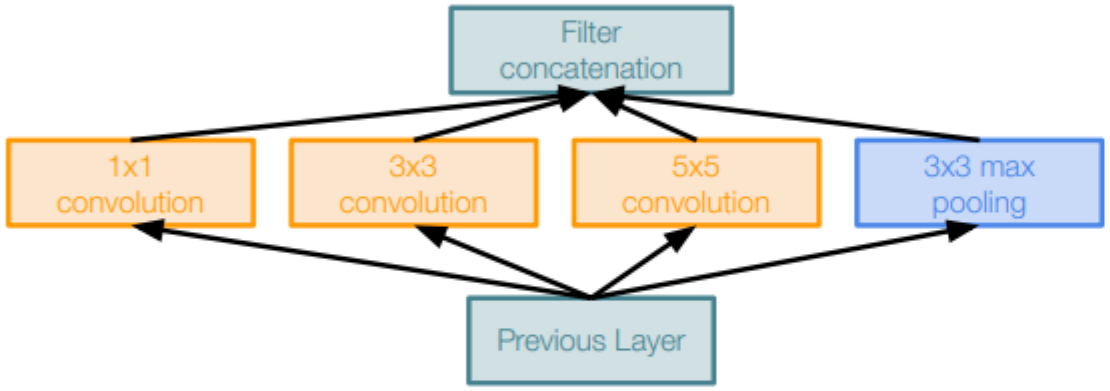
- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient "Inception" module
- No FC layers



Inception module



GoogleNet



Naive Inception module

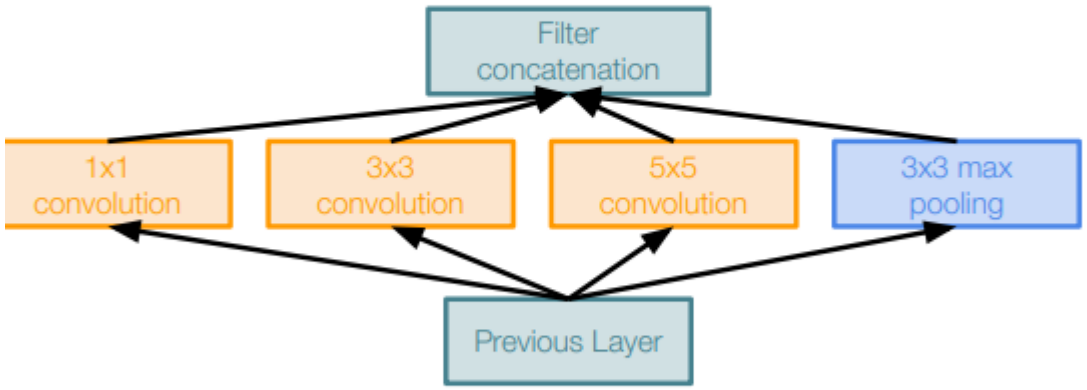
Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

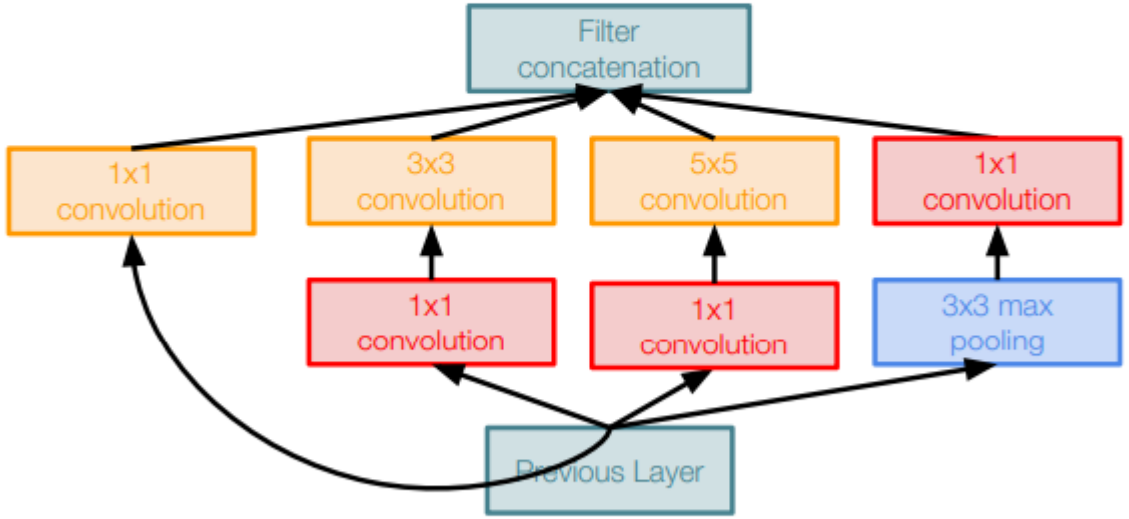
Q: What is the problem with this? [Hint: Computational complexity]

GoogleNet



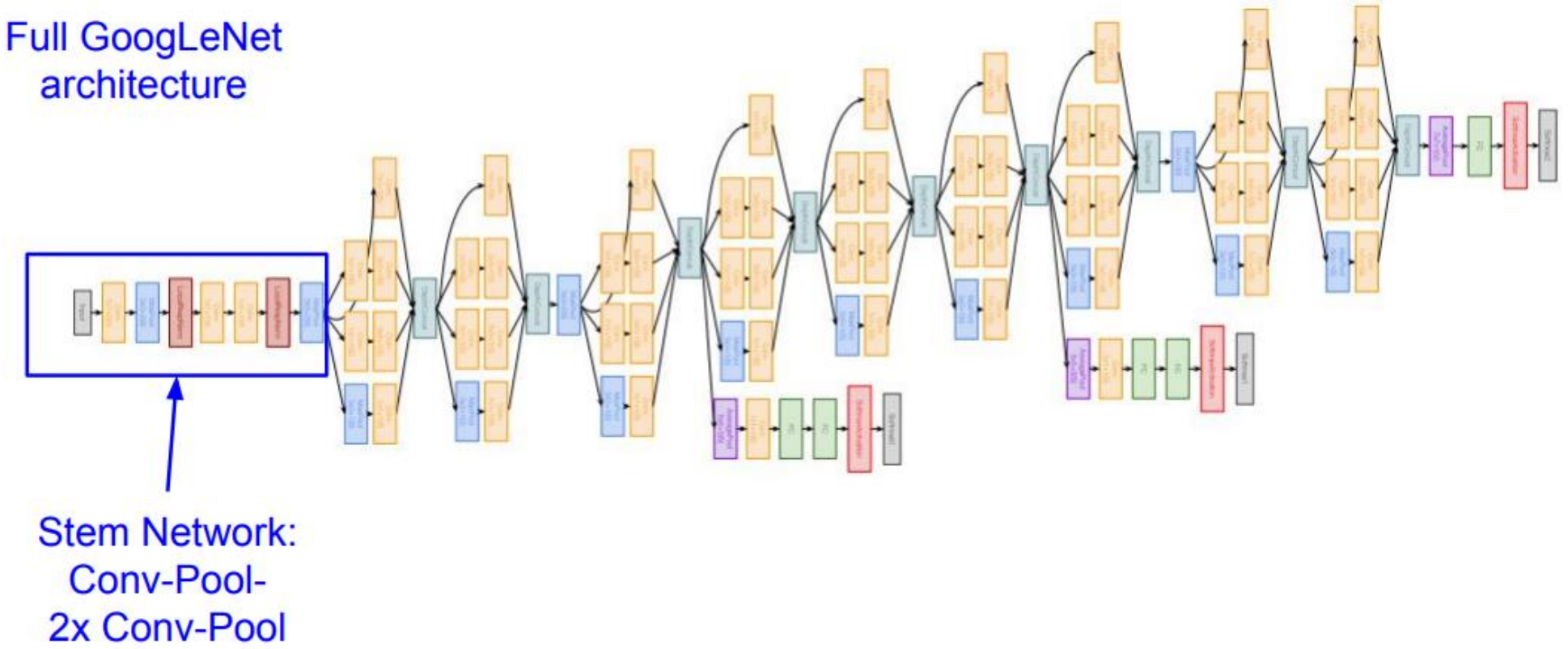
Naive Inception module

1x1 conv "bottleneck" layers



Inception module with dimension reduction

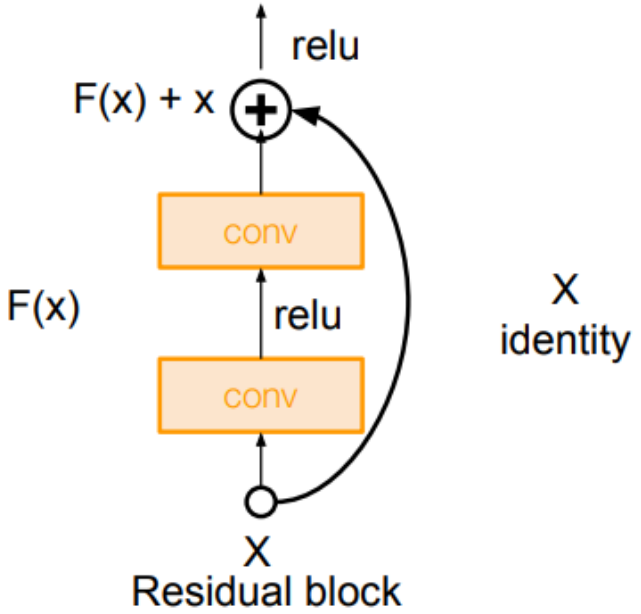
GoogleNet

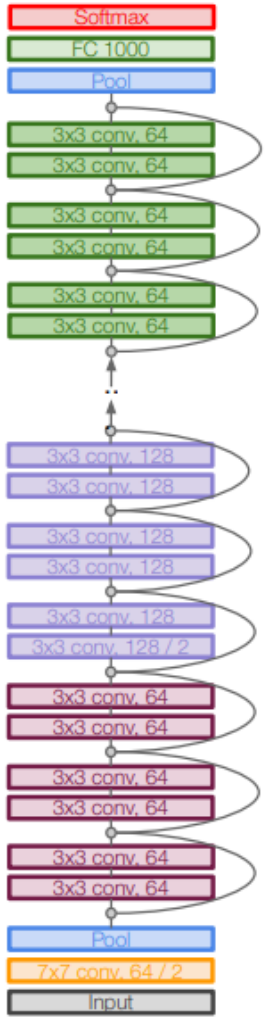

Full GoogLeNet architecture
Stem Network: Conv-Pool- 2x Conv-Pool

ResNet

Very deep networks using residual connections

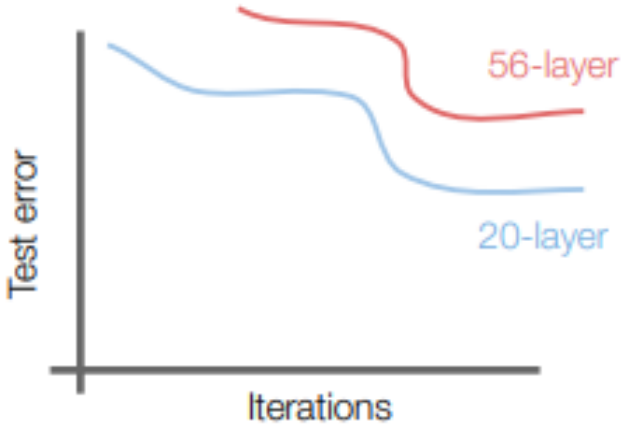
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

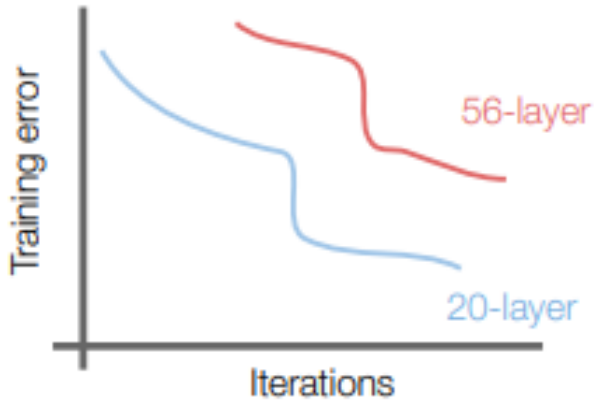




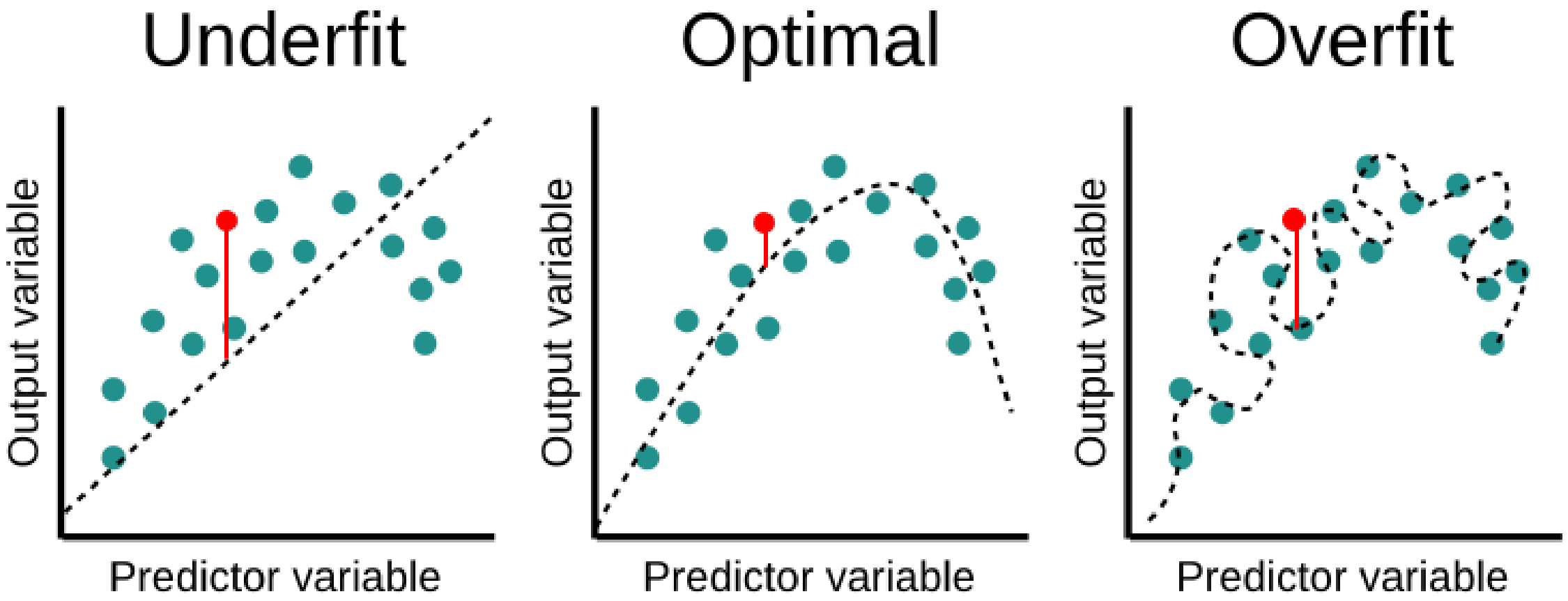
ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

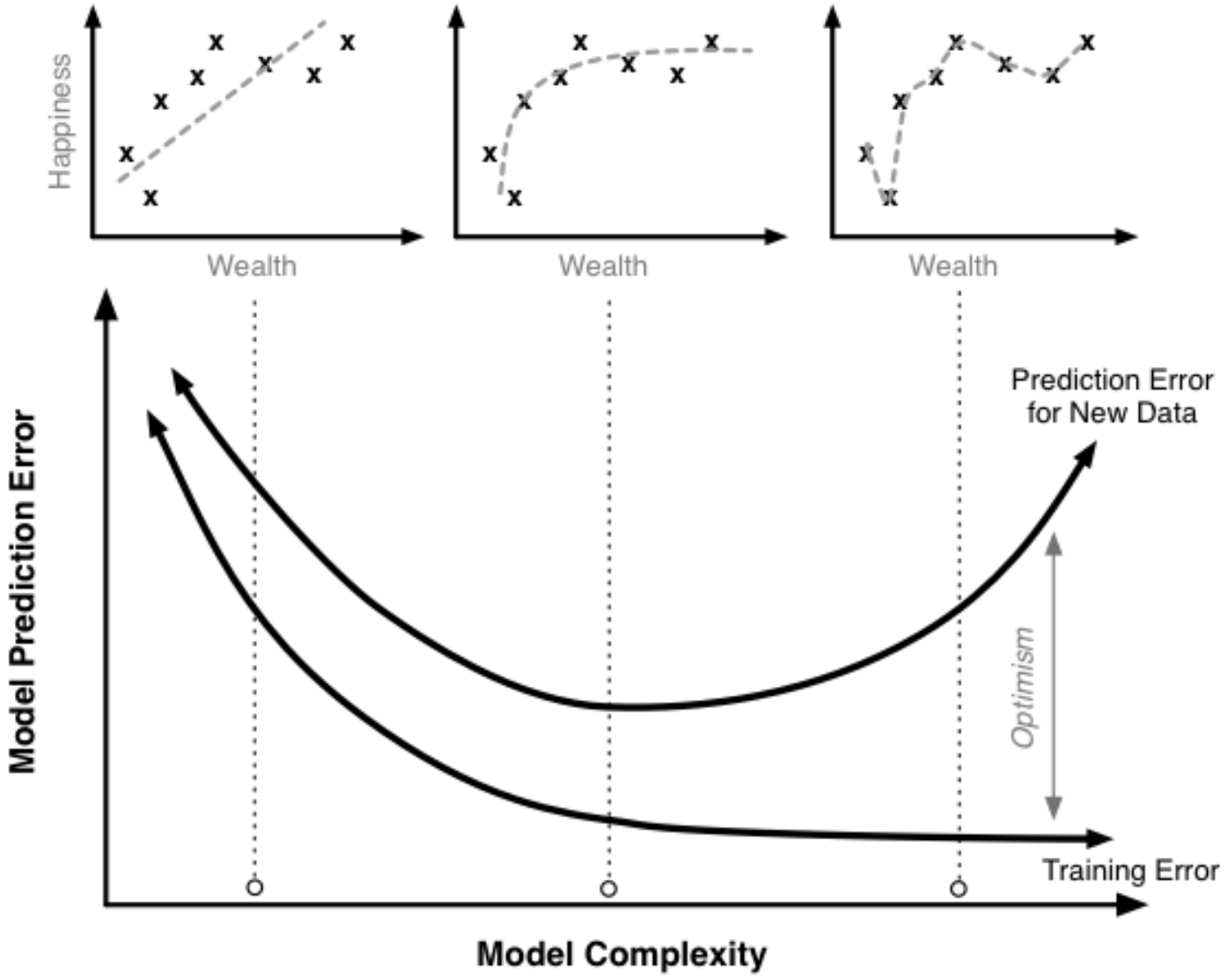




(Recap) Overfitting

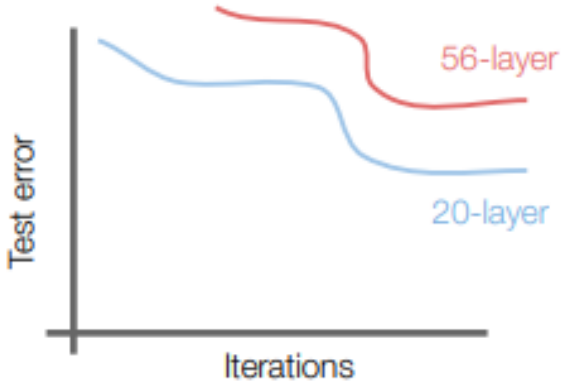


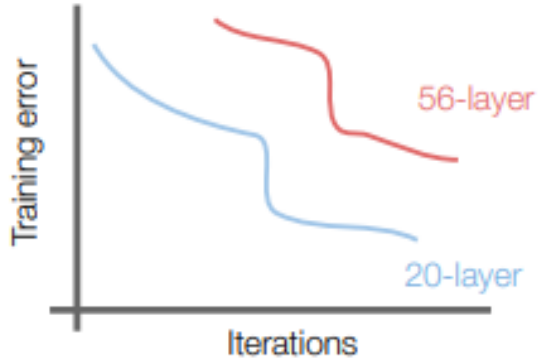
(Recap) Overfitting



ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

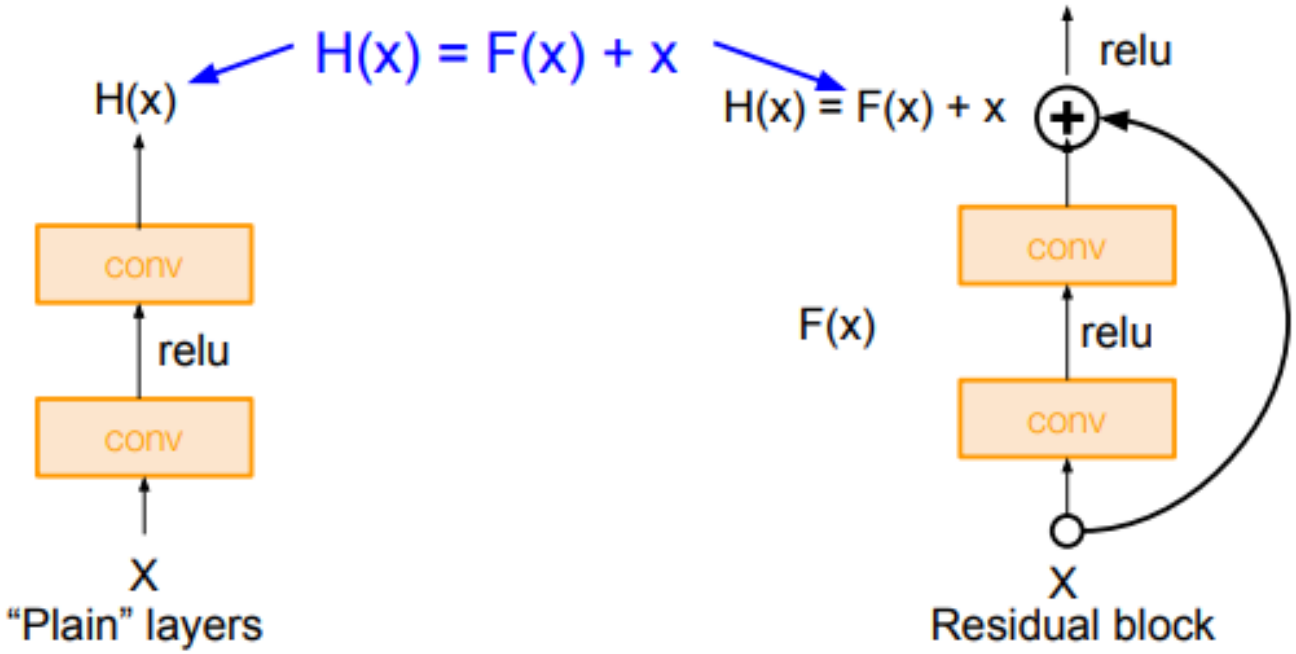




56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

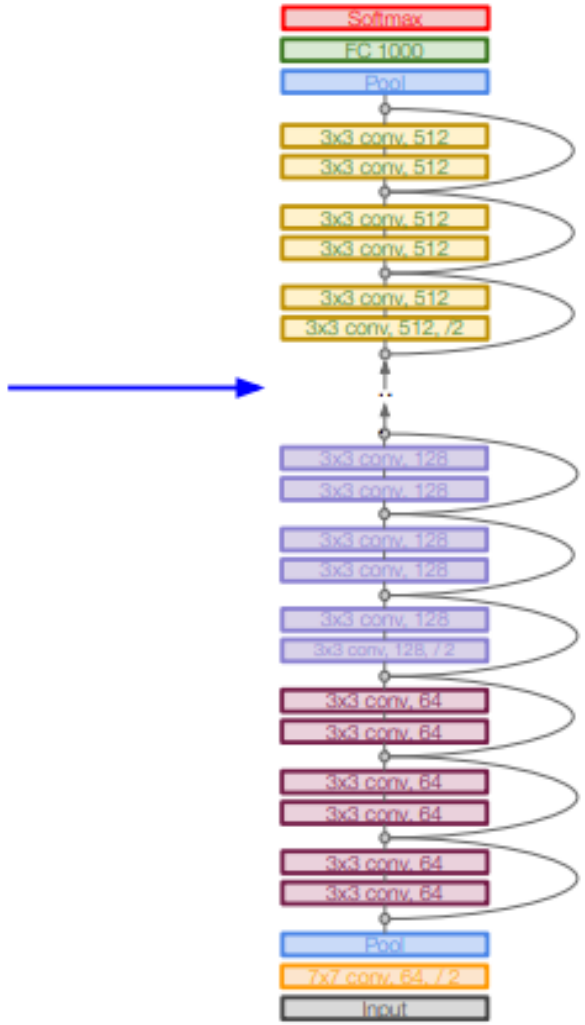


Identity mapping:
H(x) = x if F(x) = 0

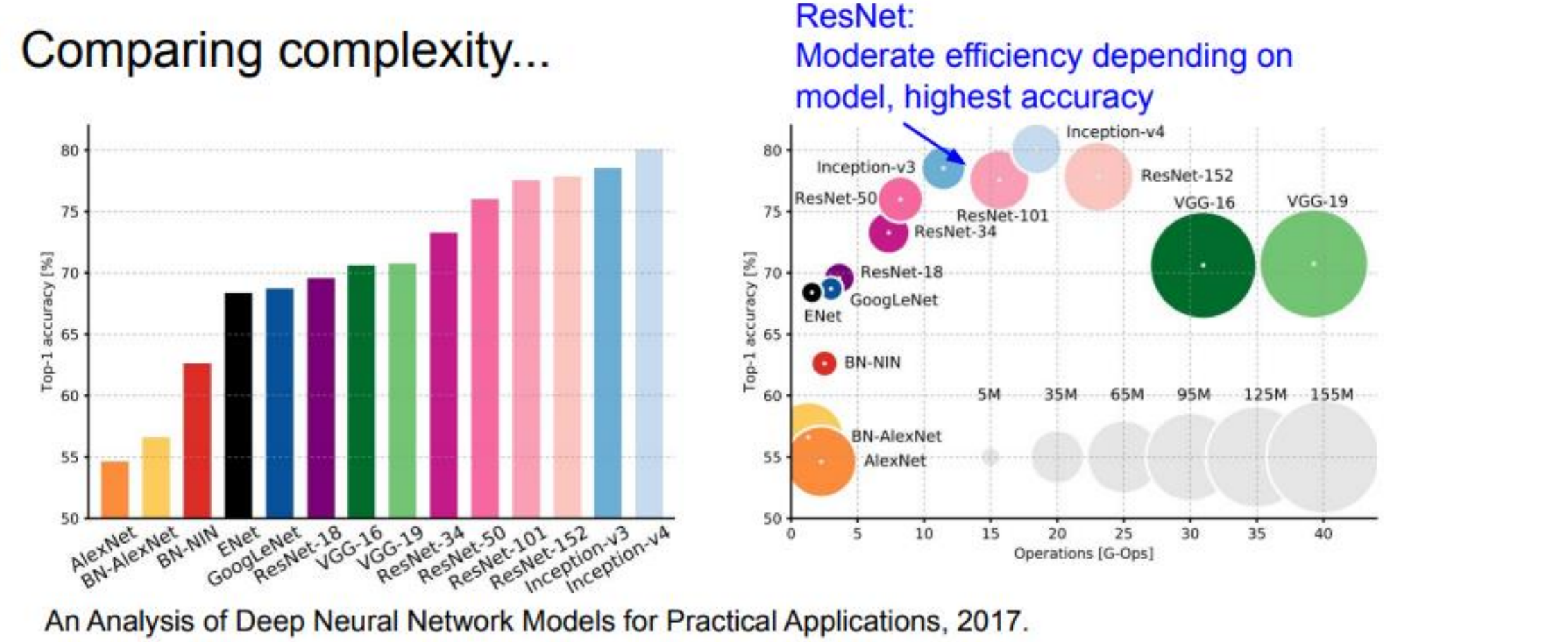
Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly

ResNet

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet

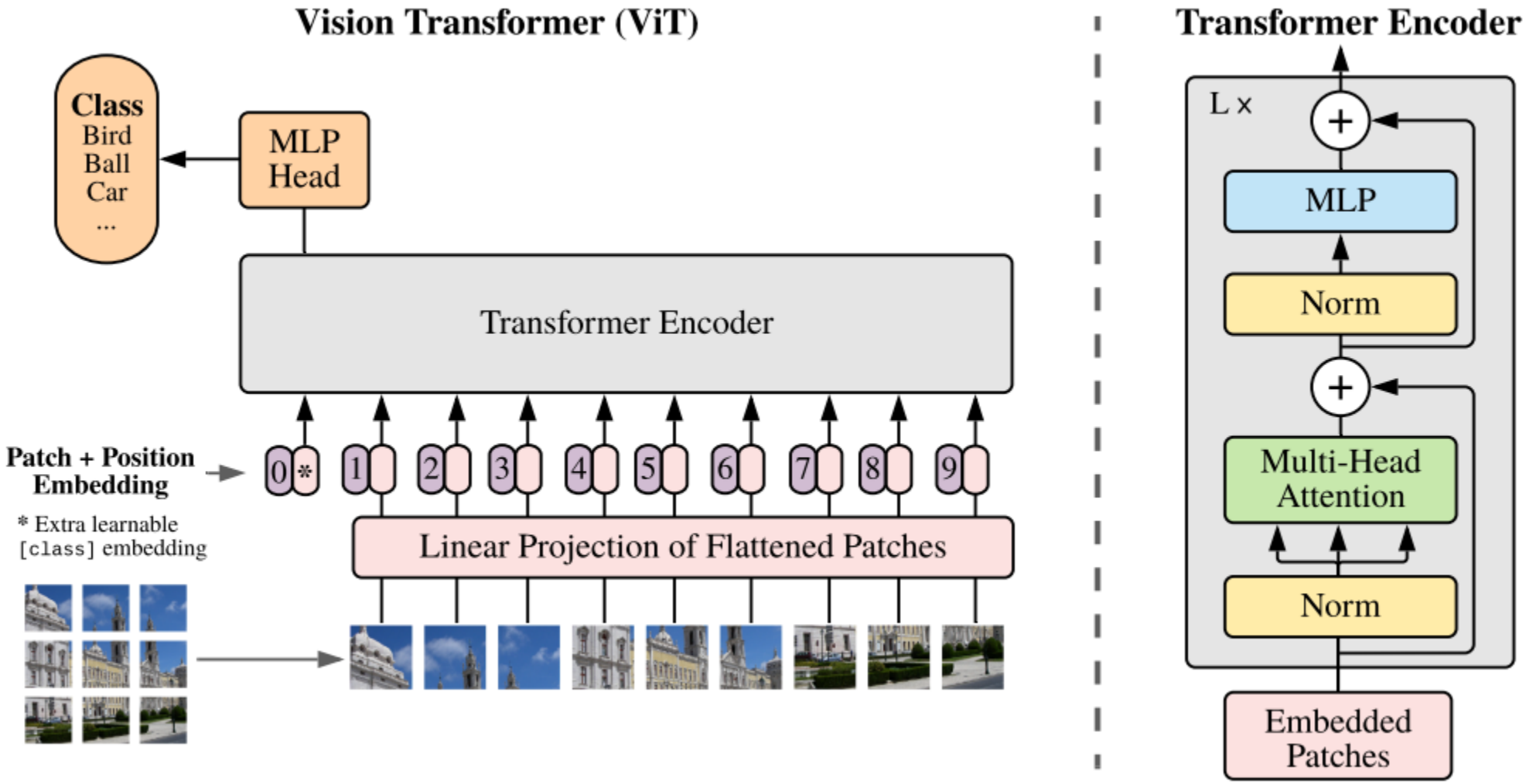


ResNet



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Now then…


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

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

- **BatchNorm**

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

BATCHNORM2D

CLASS torch.nn.BatchNorm2d(*num_features*, *eps=1e-05*, *momentum=0.1*, *affine=True*, *track_running_stats=True*, *device=None*, *dtype=None*) [SOURCE]

- **DropOut**

https://pytorch.org/docs/stable/generated/torch.nn.DropOut2d.html

DROPOUT2D

CLASS torch.nn.Dropout2d(*p=0.5*, *inplace=False*) [SOURCE]

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**

```
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
```

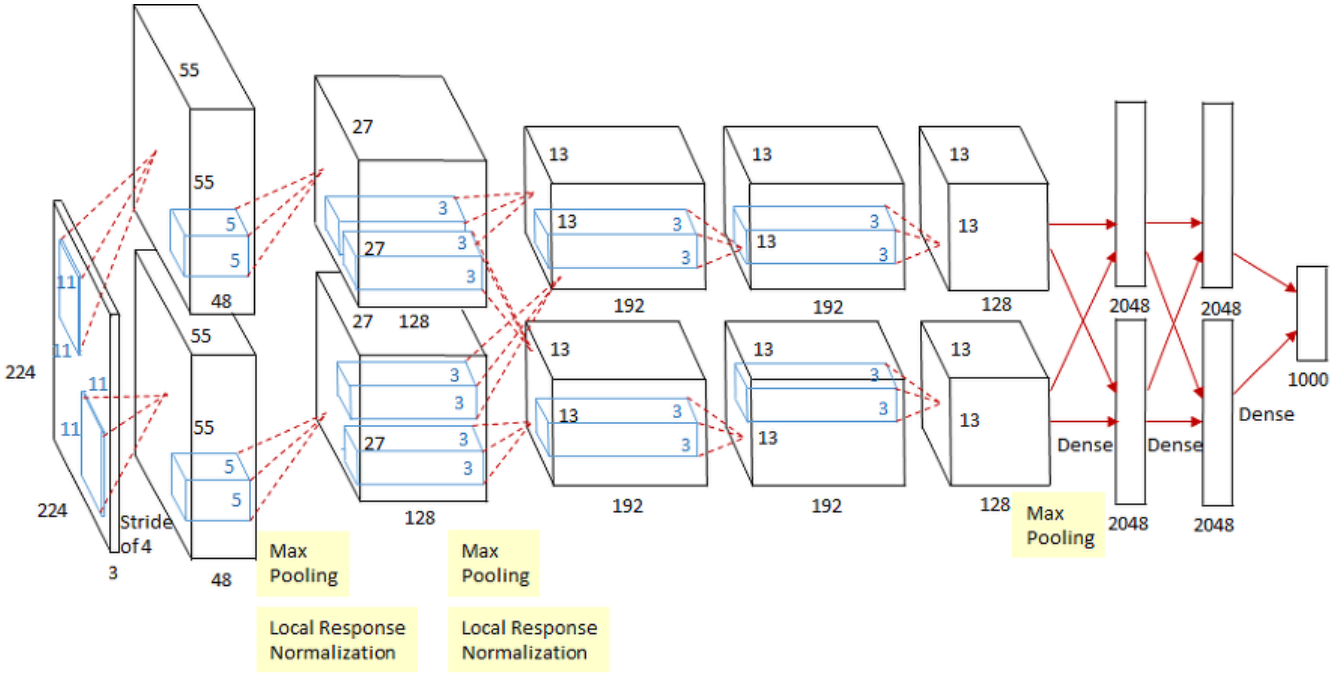
PyTorch

- **Naïve ConvNet**

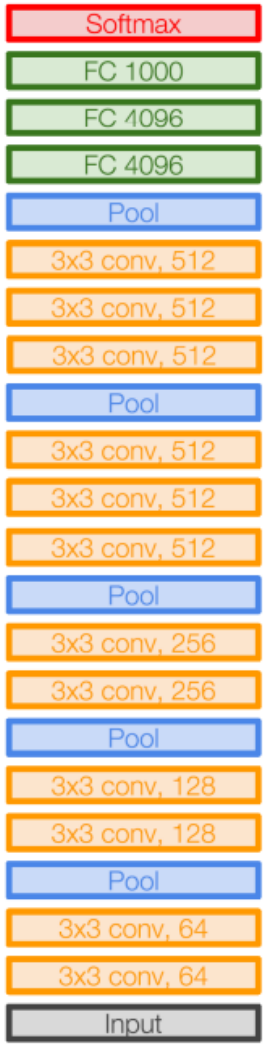
```
class Naive_ConvNet(nn.Module):
    def __init__(self):
        super().__init__()

        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**



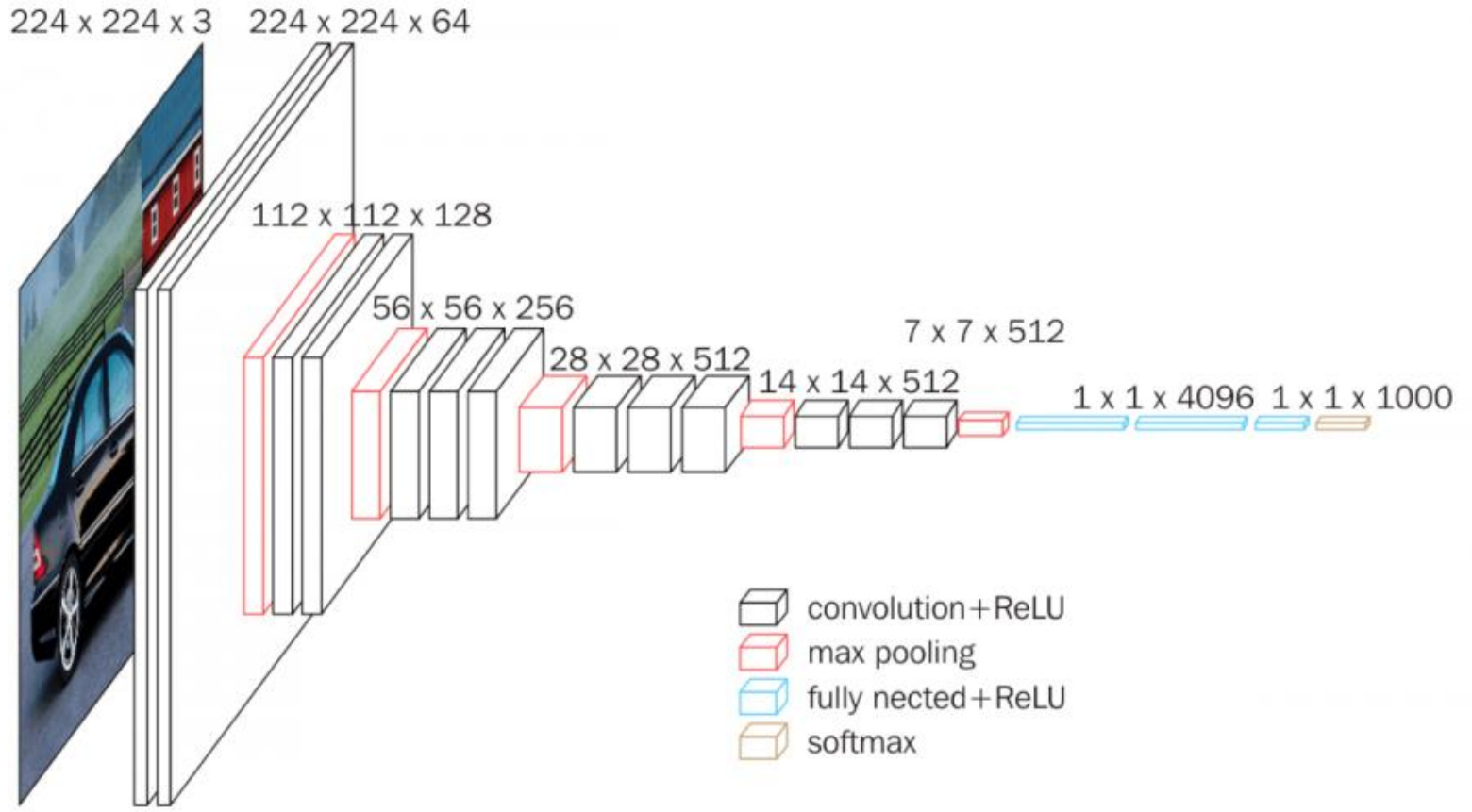
AlexNet

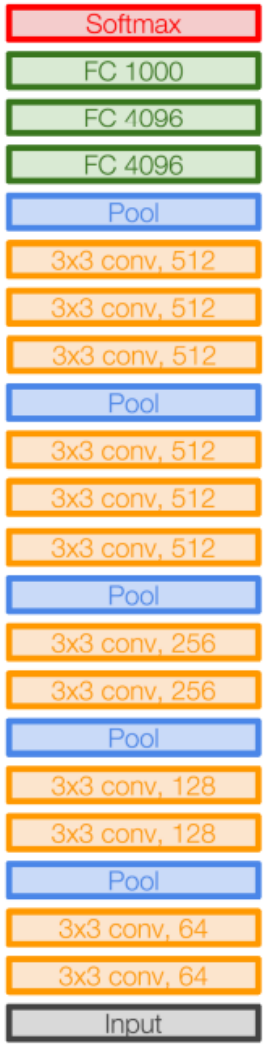


VGG16

Assignment

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





VGG16

Assignment

- Implementation of **AlexNet** and **VGG16**
- You should implement **at least one architecture**
- **Due Date 1/26(Thu) 3 p.m.**

- Lectures, Exercises, and Assignments will be uploaded in Notion

Thank you!
Q & A