

AI504: Programming for Artificial Intelligence

Week 7: Convolutional Neural Network

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr

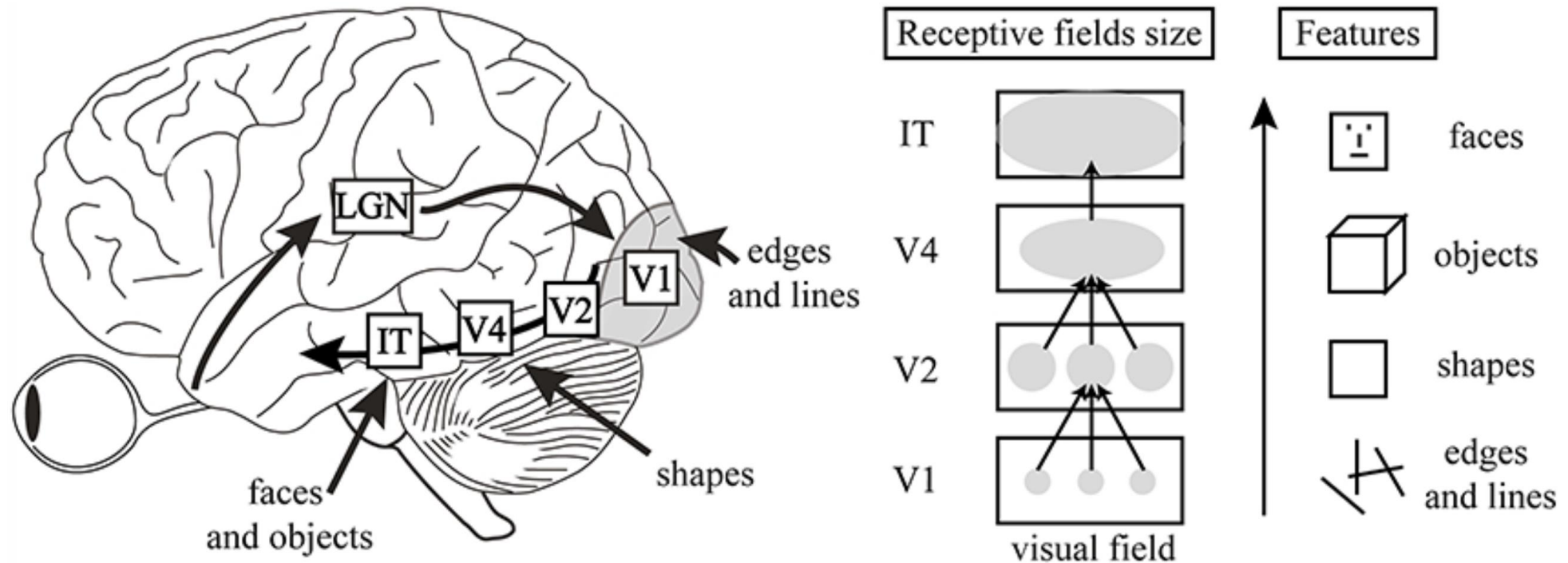
Today's Topic

- CNN
 - Filter, Strides, Pooling
 - 1D, 3D CNN
- Training Technique
 - BatchNorm, Dropout
- CNN Architectures
 - VGG, Inception, ResNet
- Project 1

Convolutional Neural Network

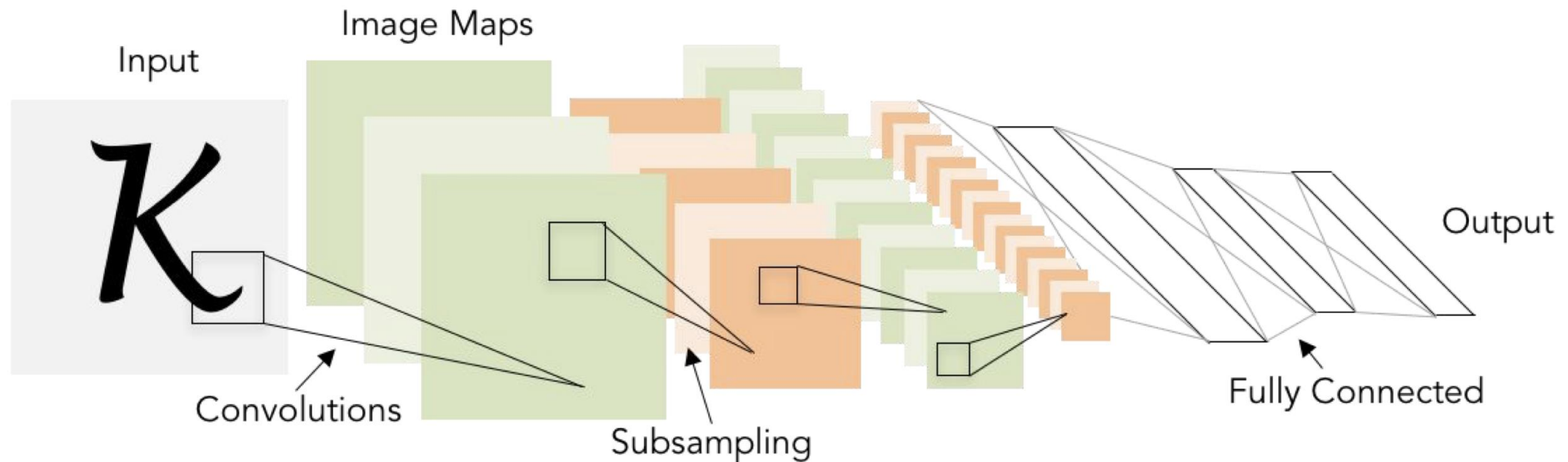
Biological Motivation

- Human visual perception



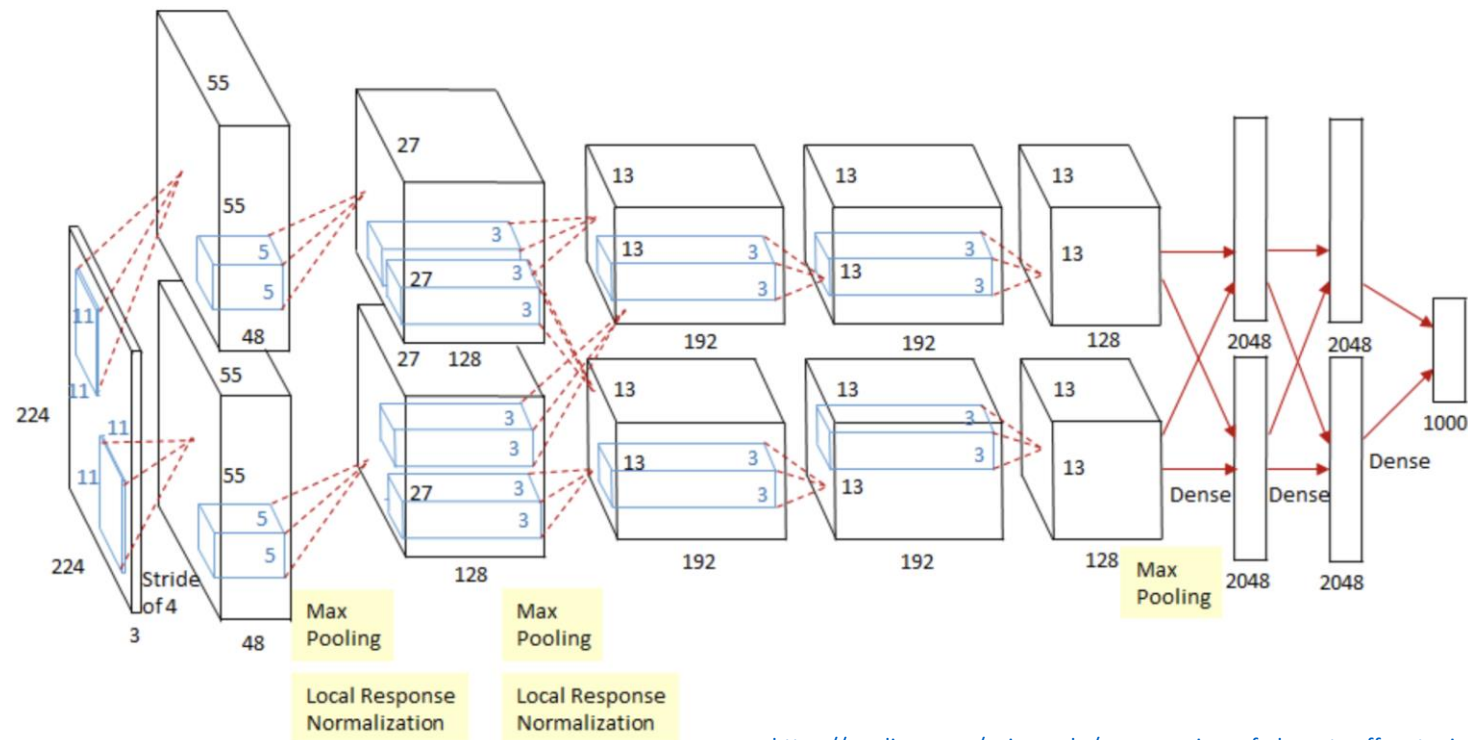
History

- LeNet-5
 - Gradient-based learning applied to document recognition
 - [LeCun, Bottou, Bengio, Haffner 1998]



History

- AlexNet
 - ImageNet Classification with Deep Convolutional Neural Networks
 - [Krizhevsky, Sutskever, Hinton, 2012]



Modern ConvNets

Classification

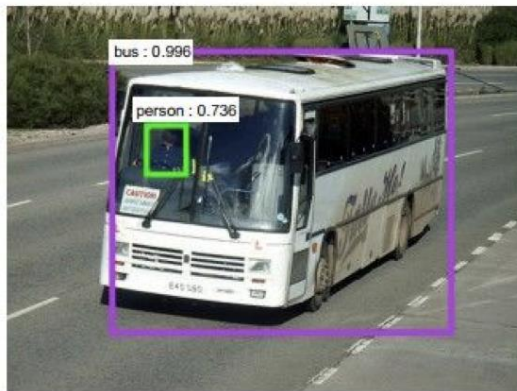
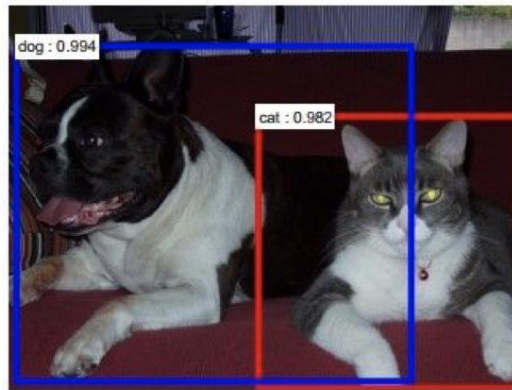
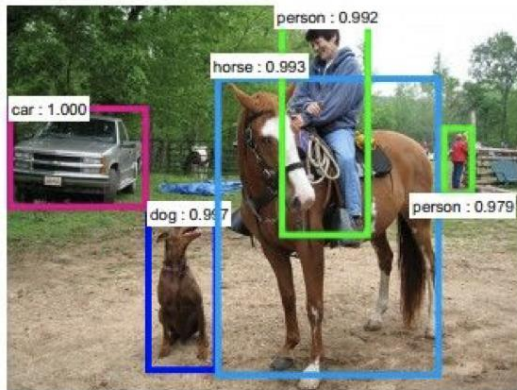


Retrieval

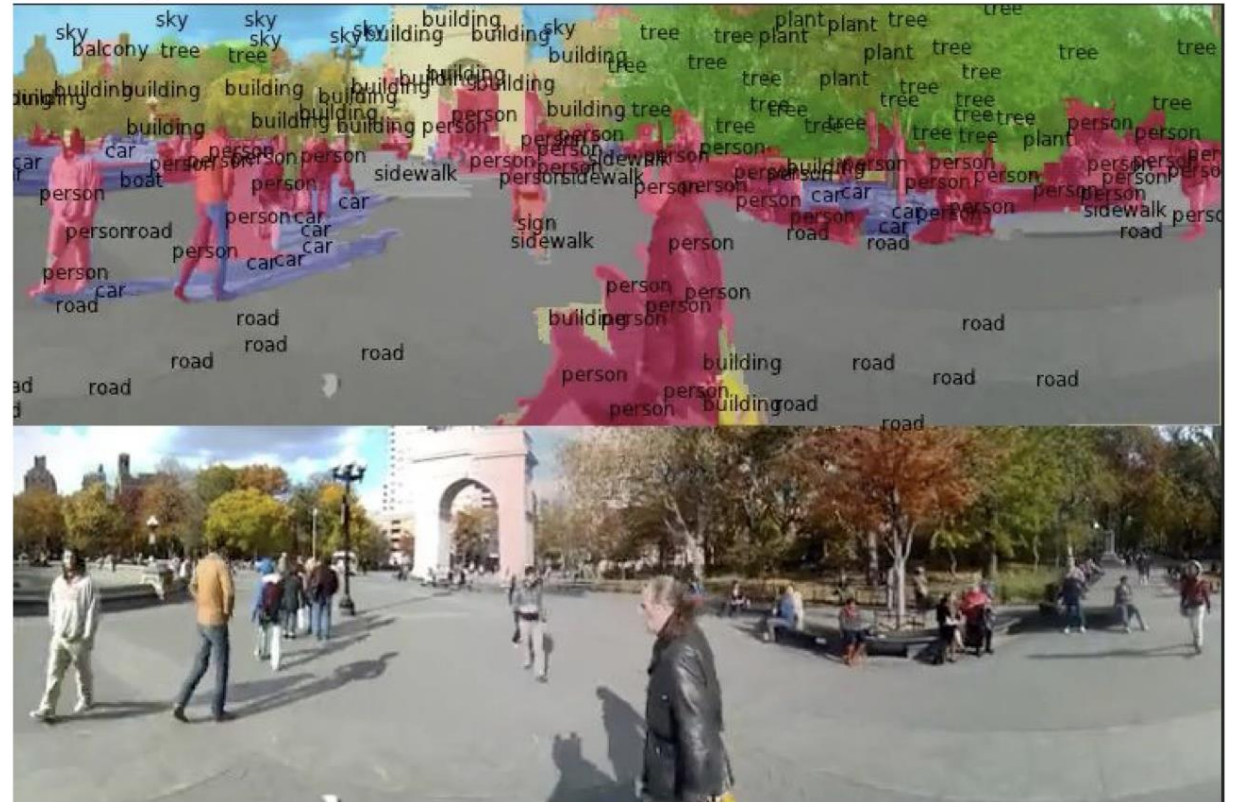


Modern ConvNets

Detection



Segmentation



Modern ConvNets

No errors



A white teddy bear sitting in the grass

Minor errors



A man in a baseball uniform throwing a ball

Somewhat related



A woman is holding a cat in her hand



A man riding a wave on top of a surfboard

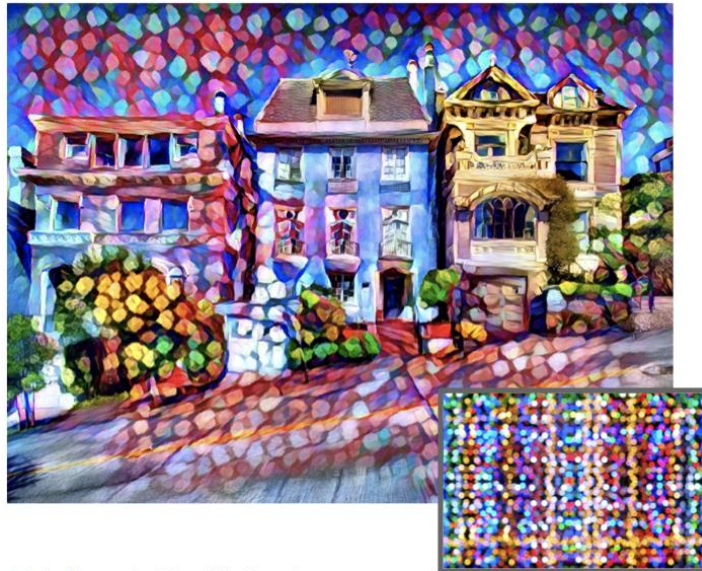


A cat sitting on a suitcase on the floor



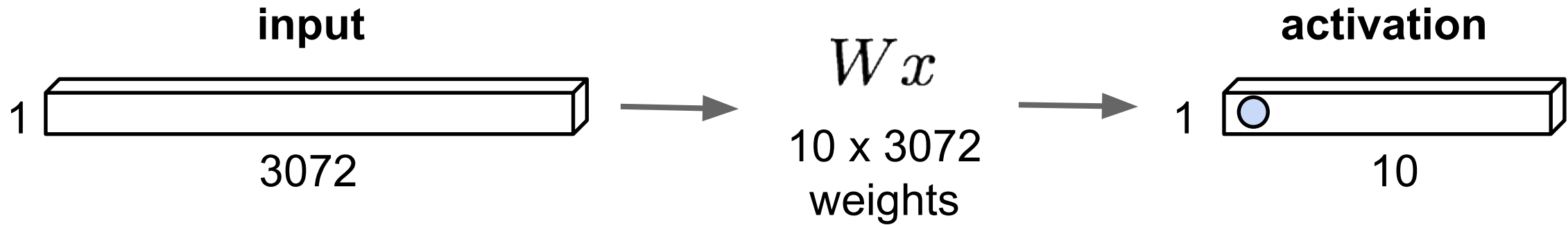
A woman standing on a beach holding a surfboard

Modern ConvNets



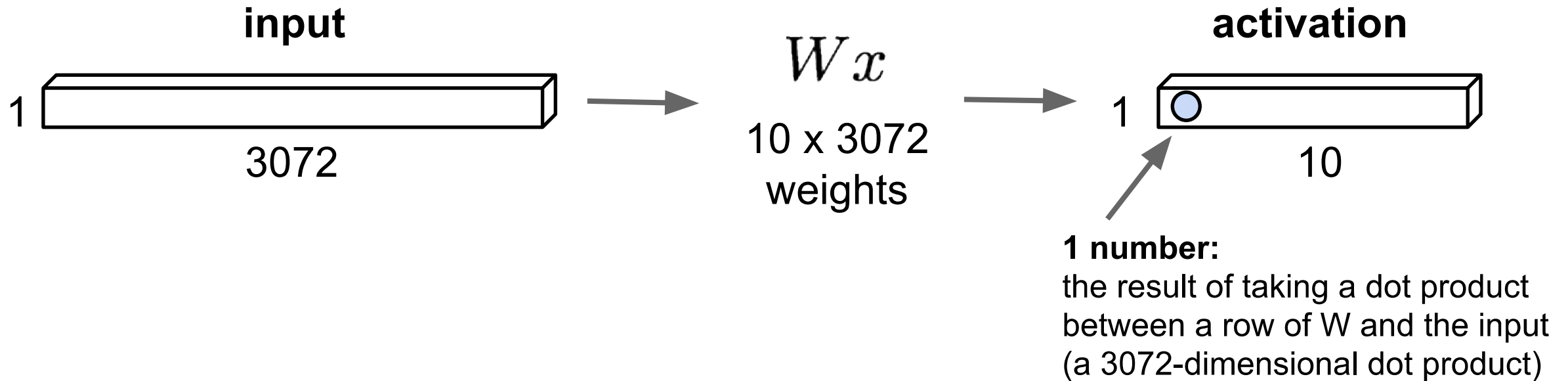
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



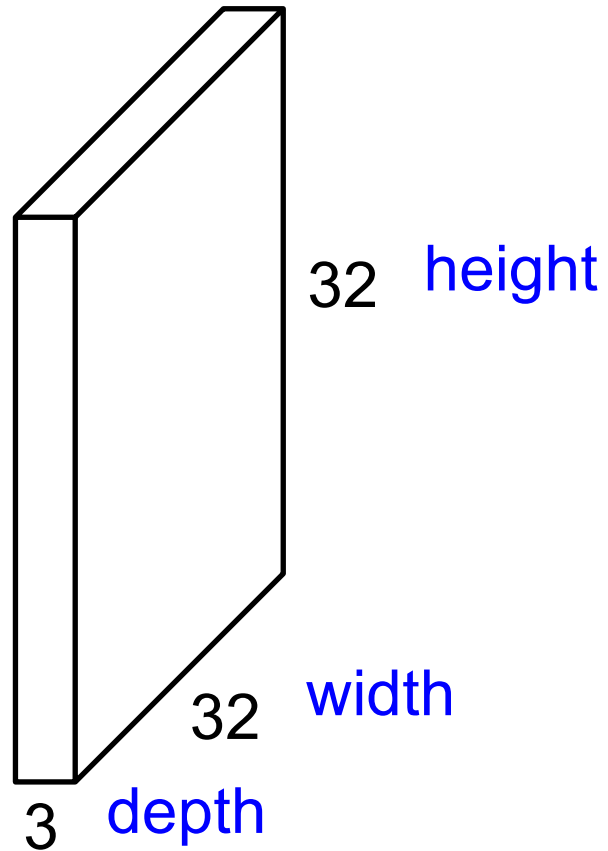
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



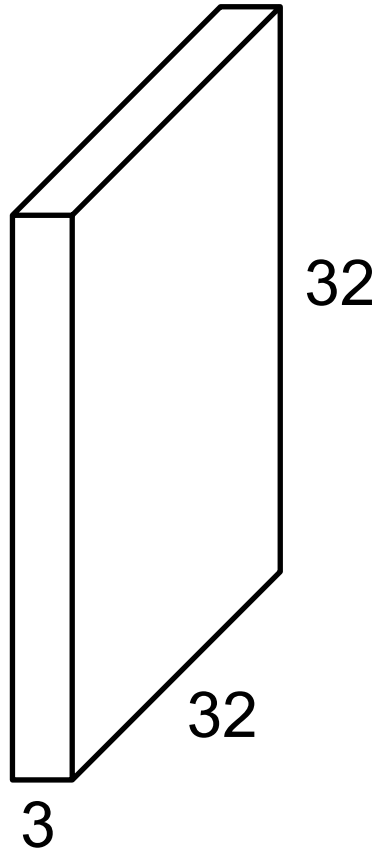
Convolution Layer

32x32x3 image -> preserve spatial structure

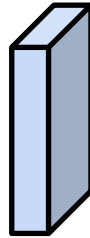


Convolution Layer

32x32x3 image



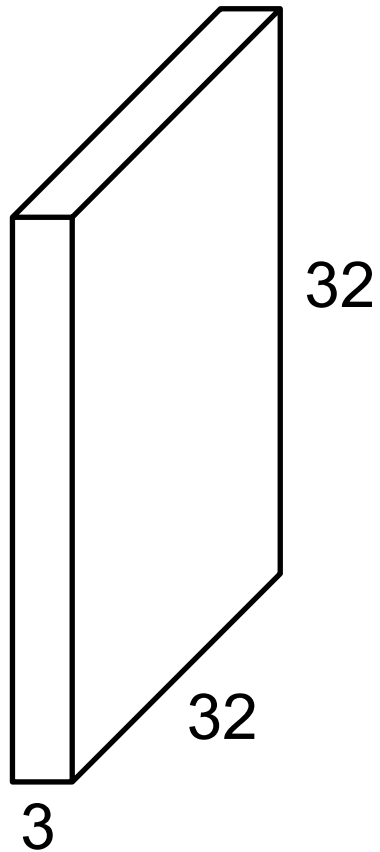
5x5x3 filter (a.k.a kernel)



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

Convolution Layer

32x32x3 image



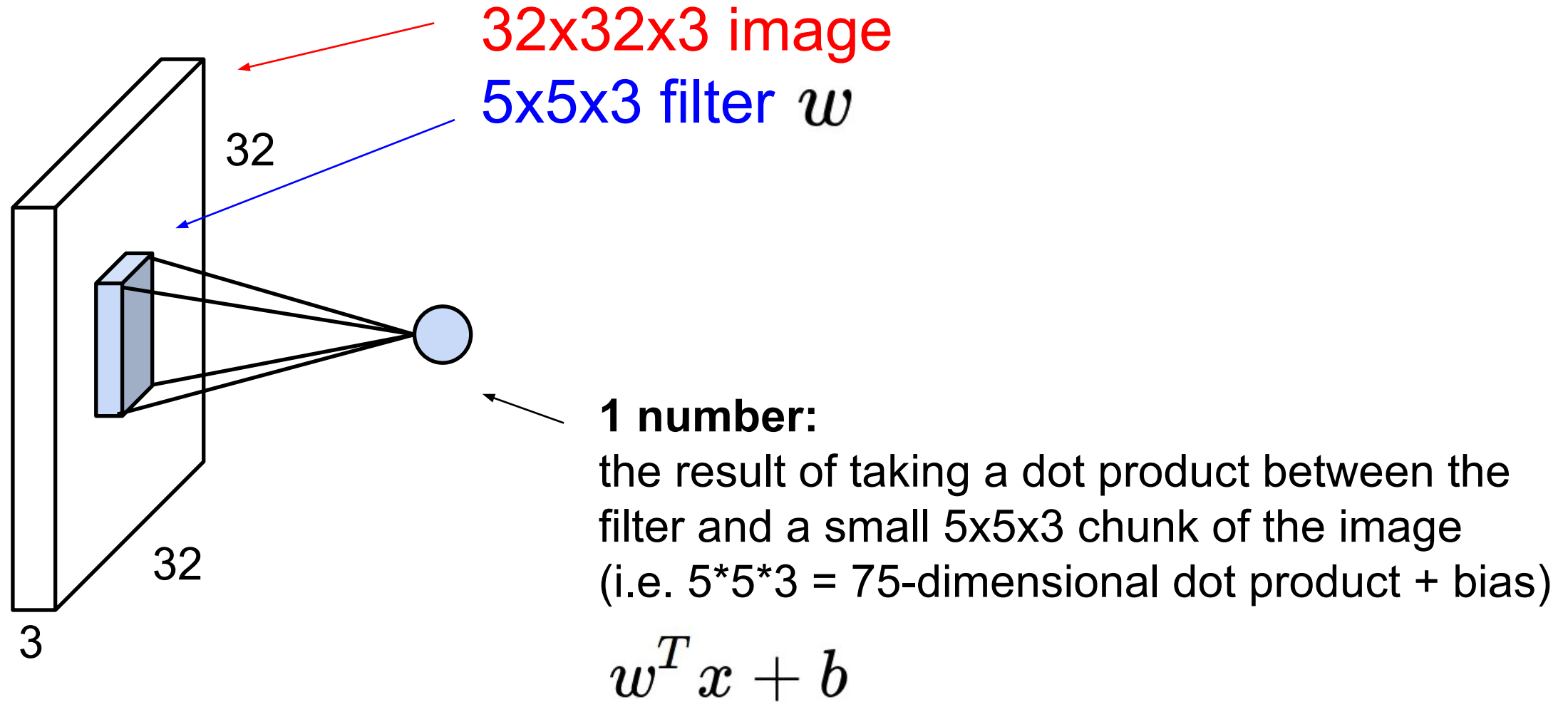
Filters always extend the full depth of the input volume

5x5x3 filter

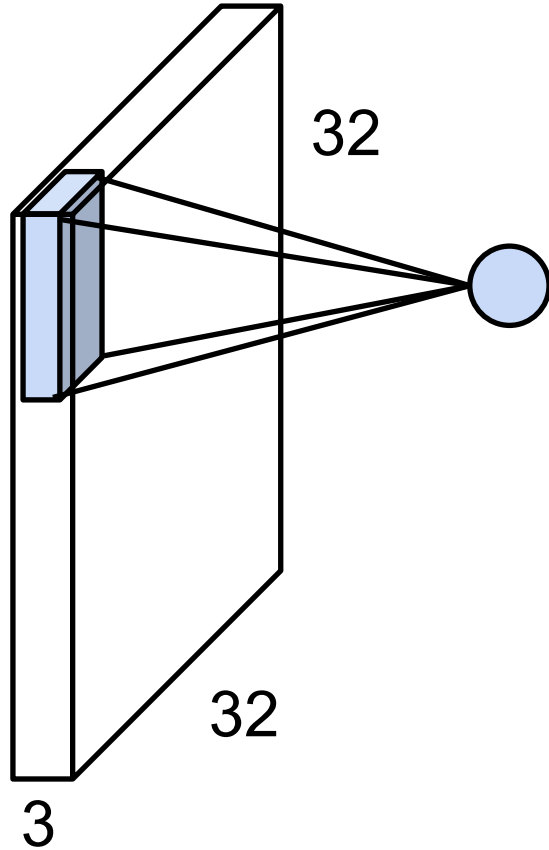


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

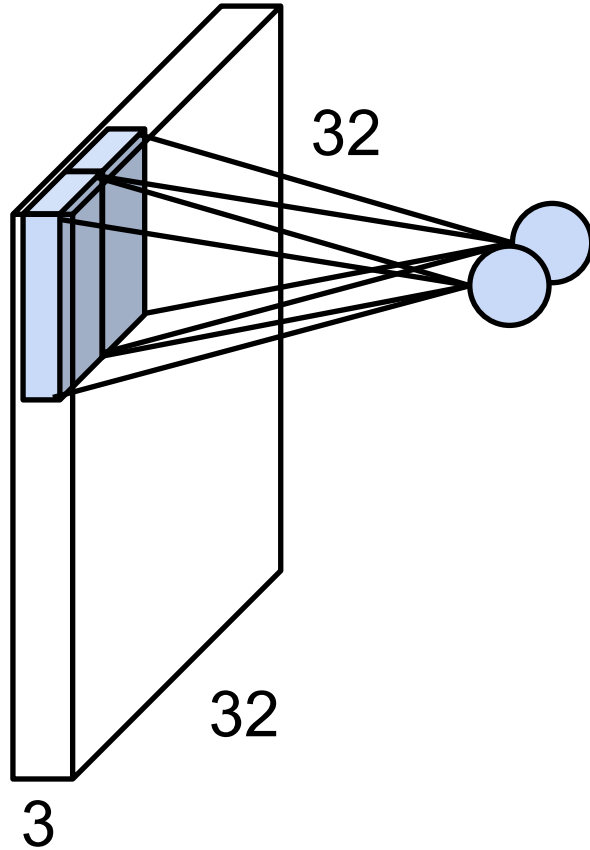
Convolution Layer



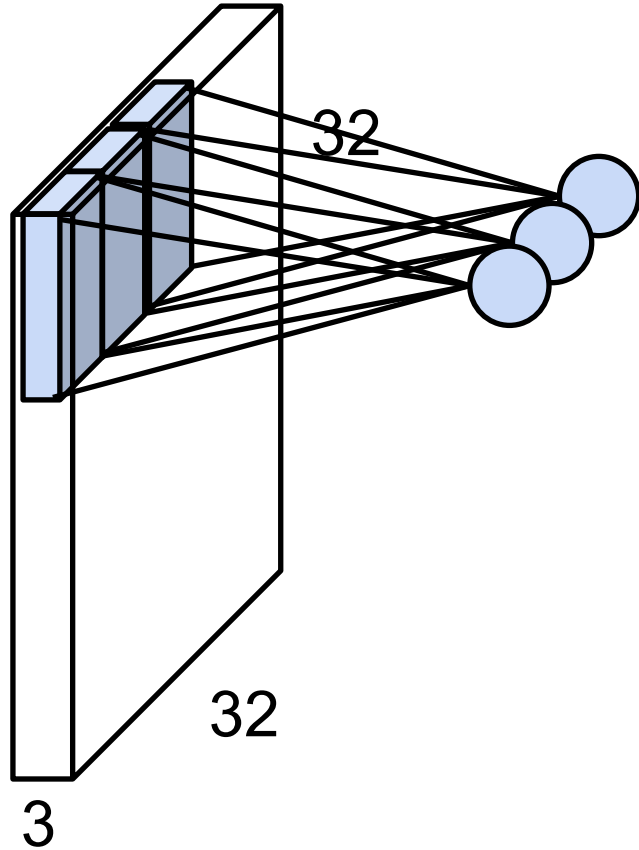
Convolution Layer



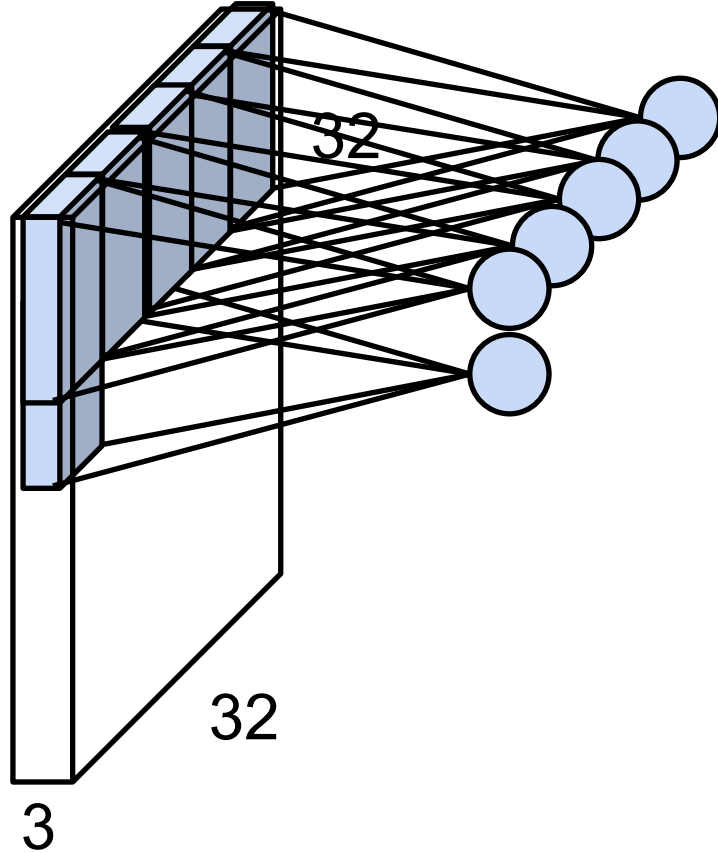
Convolution Layer



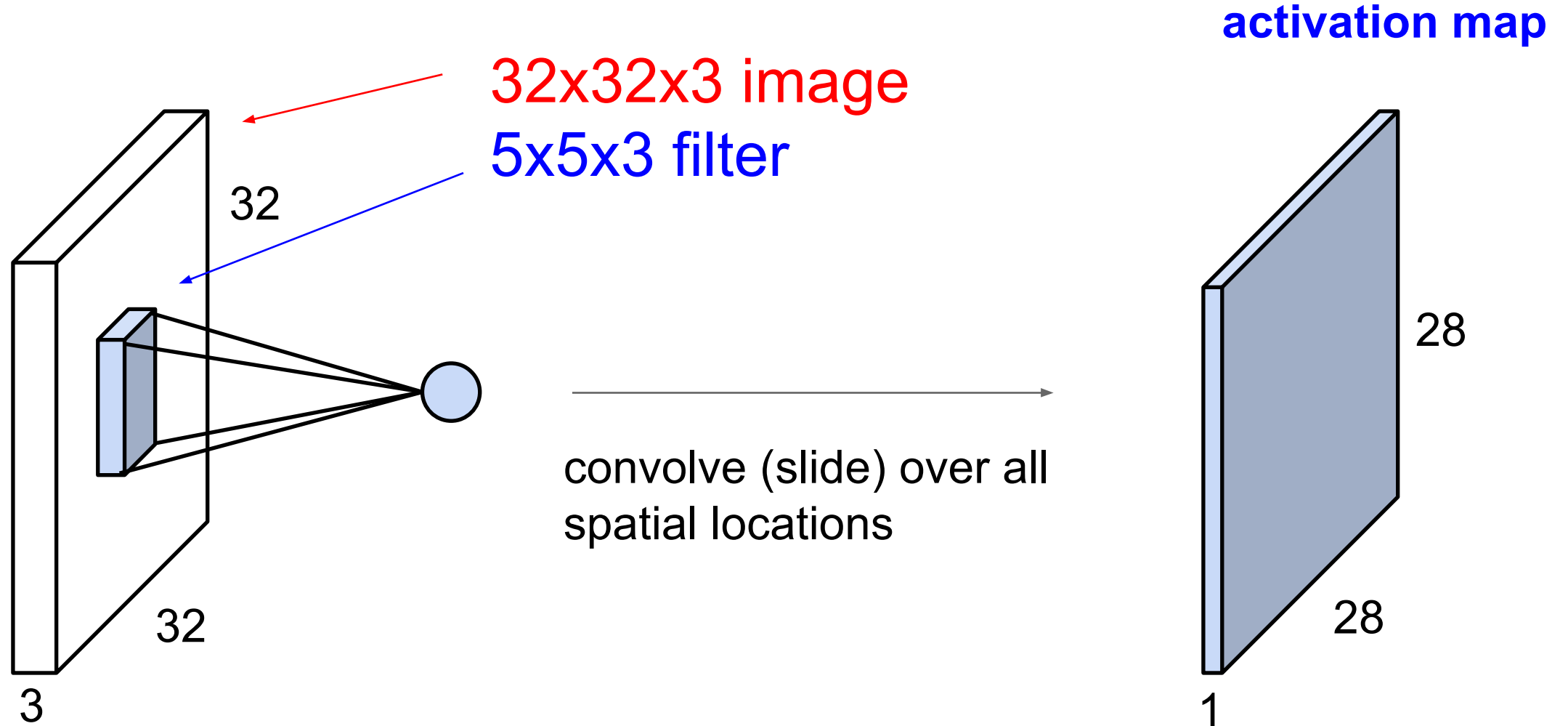
Convolution Layer



Convolution Layer

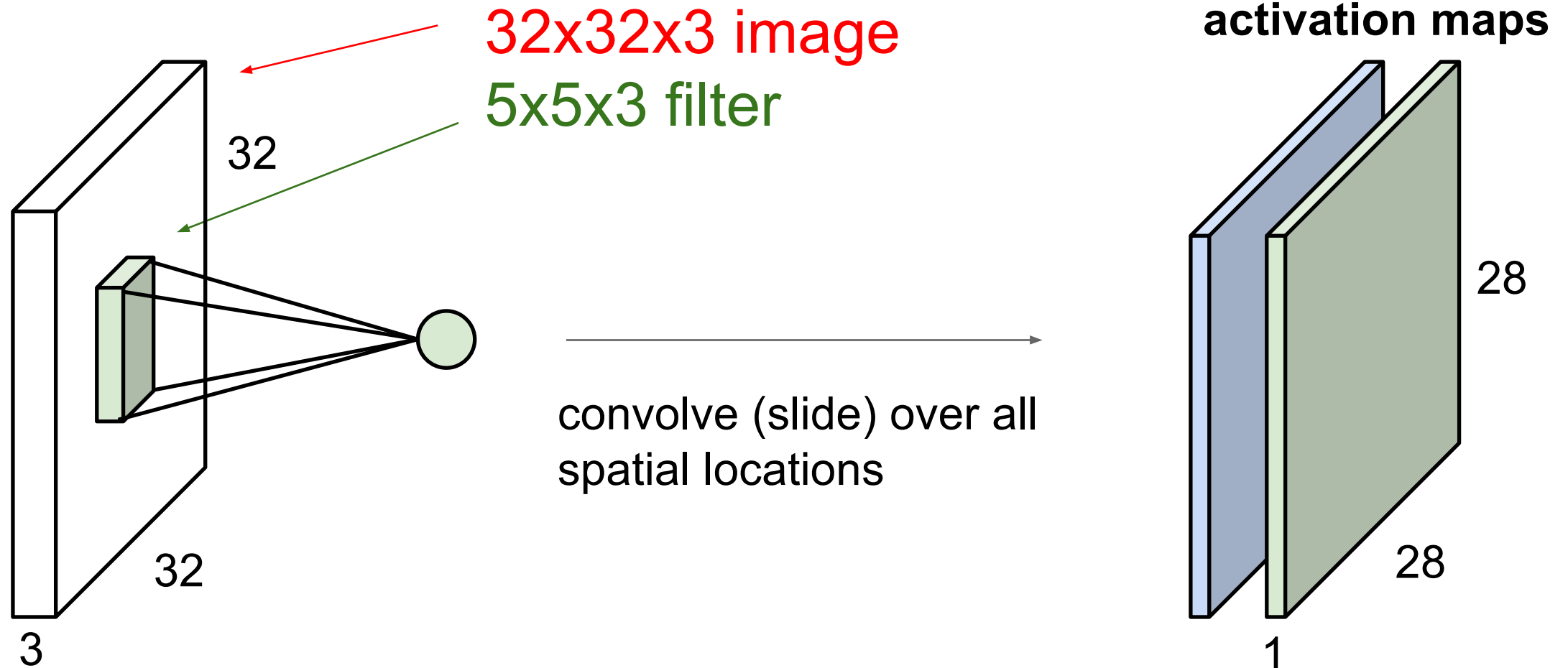


Convolution Layer

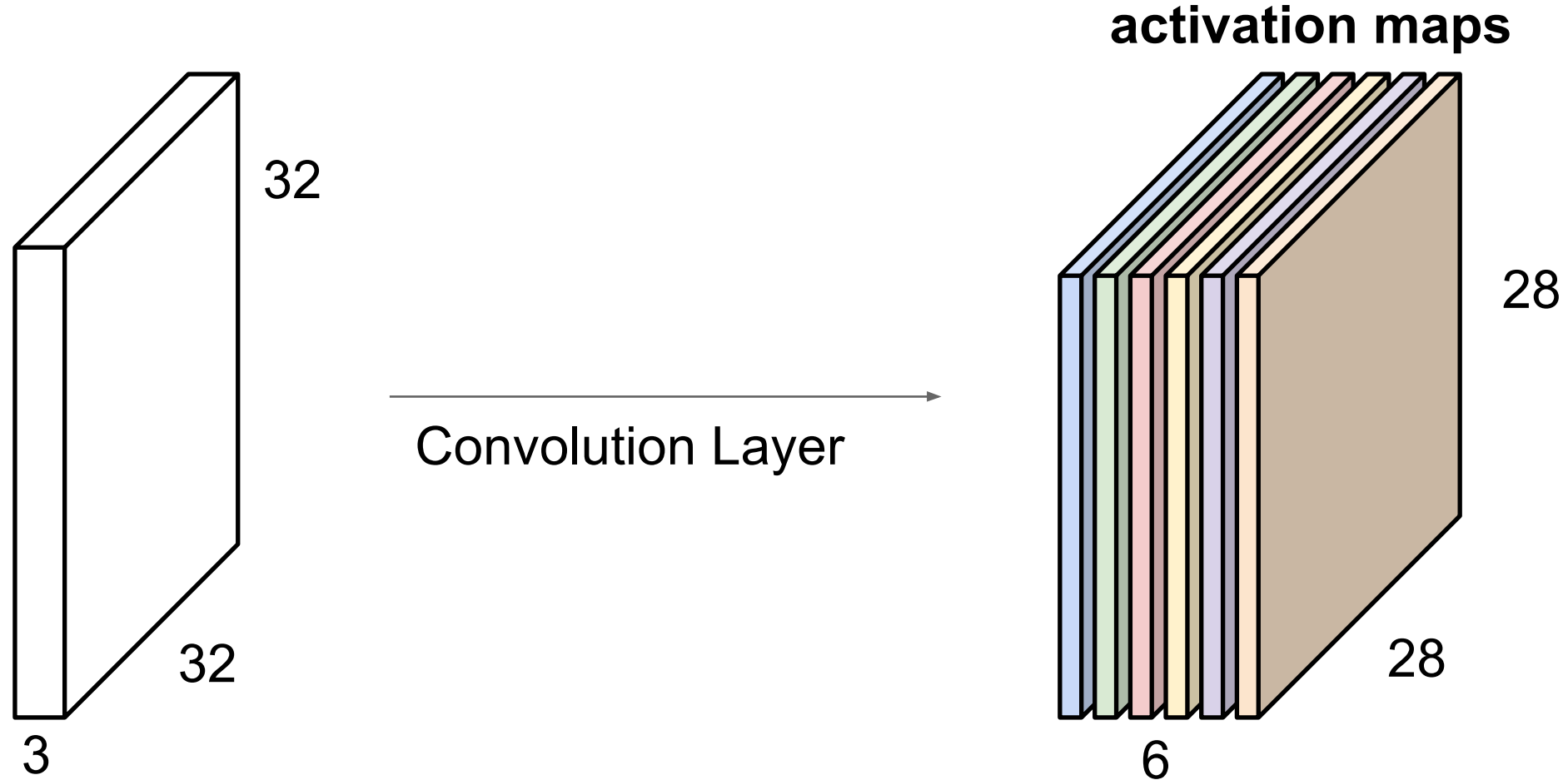


Convolution Layer

consider a second, **green** filter

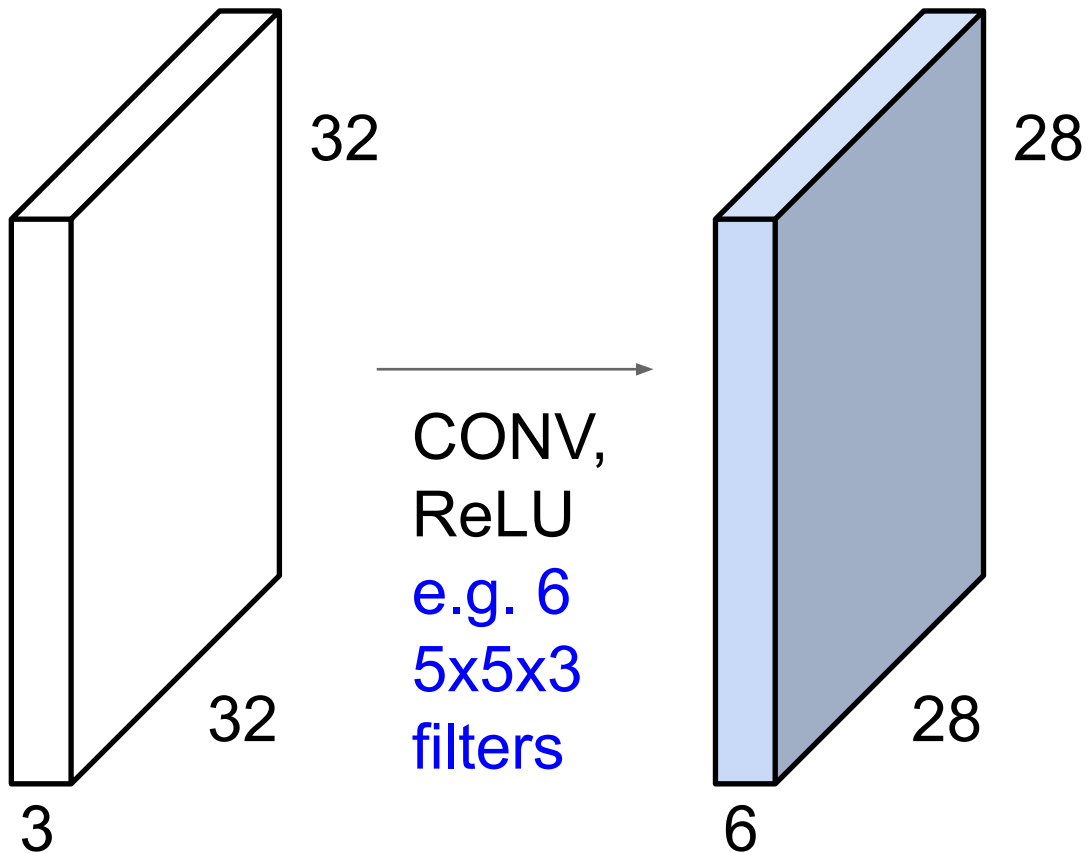


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

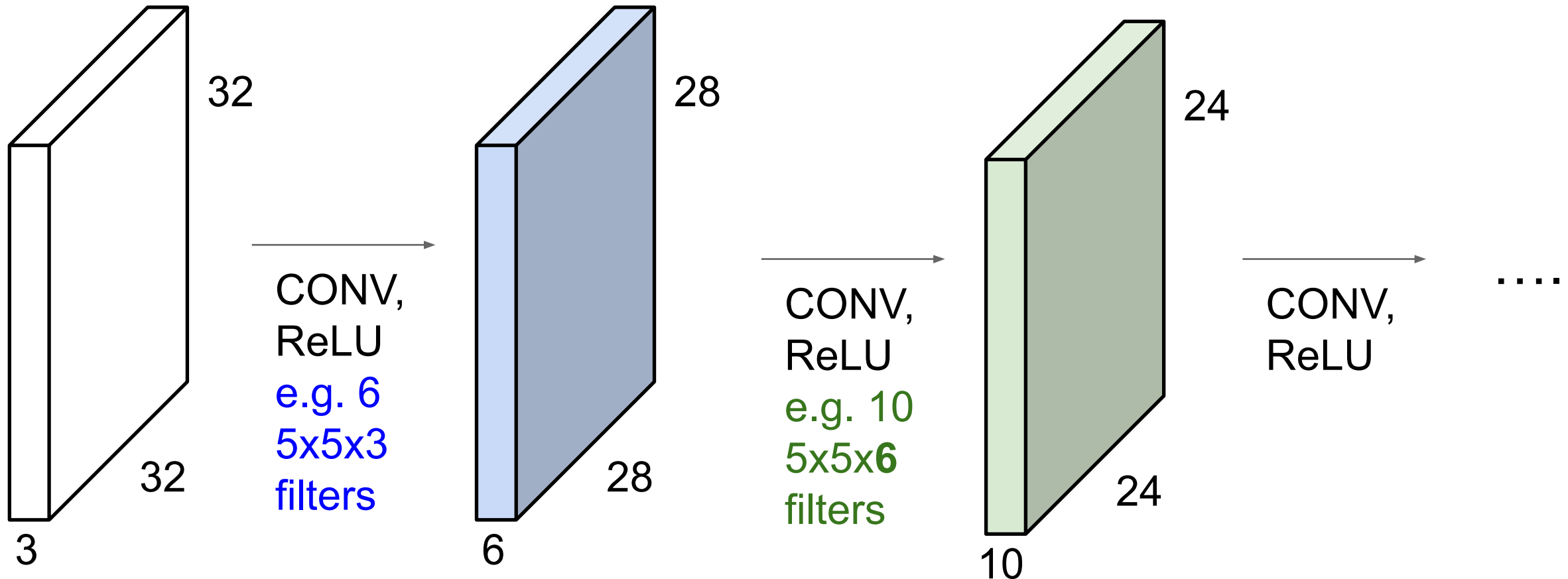


We stack these up to get a “new image” of size 28x28x6!

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

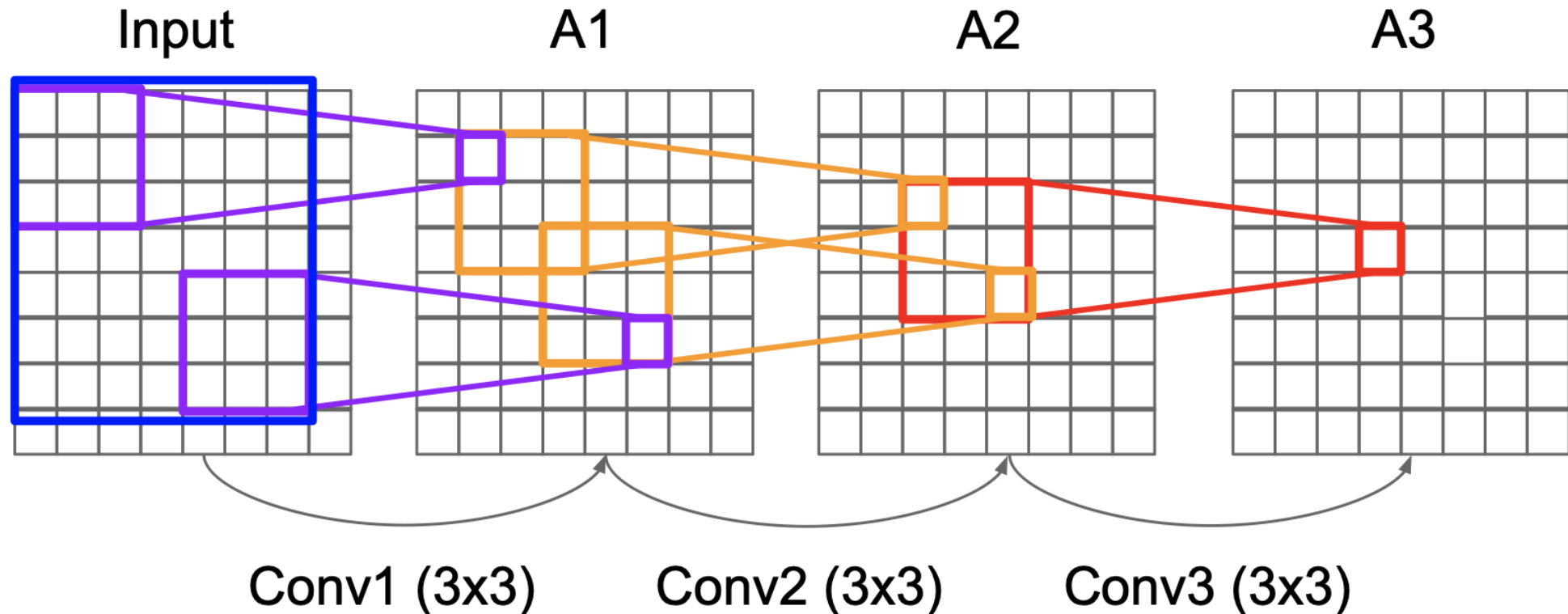


Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Receptive Field

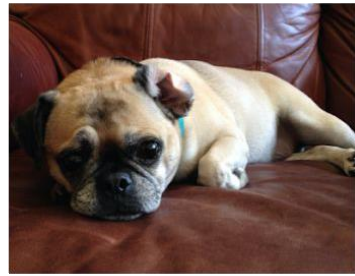
- Repeated Conv layers increase the receptive field.
 - 3 layers of 3x3 filter has the same receptive field as a single 7x7 filter.
 - The former requires less params, and is deeper!



Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

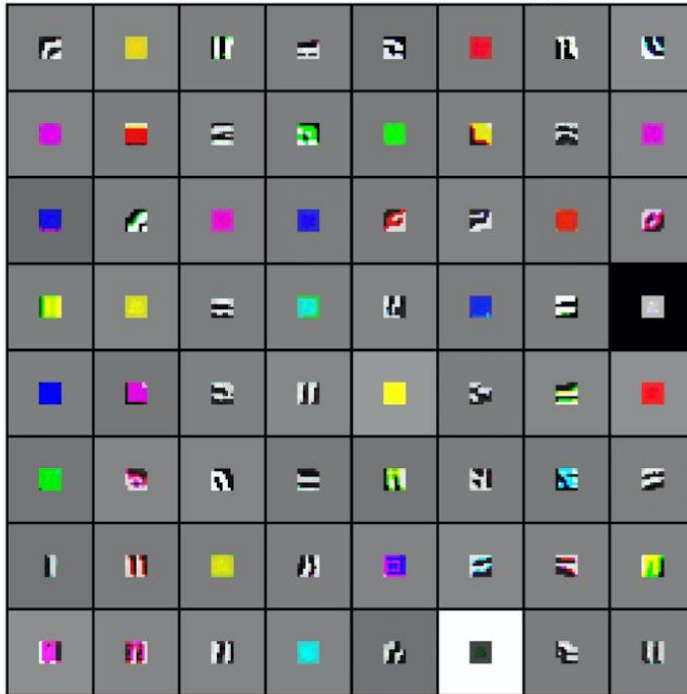


Low-level features

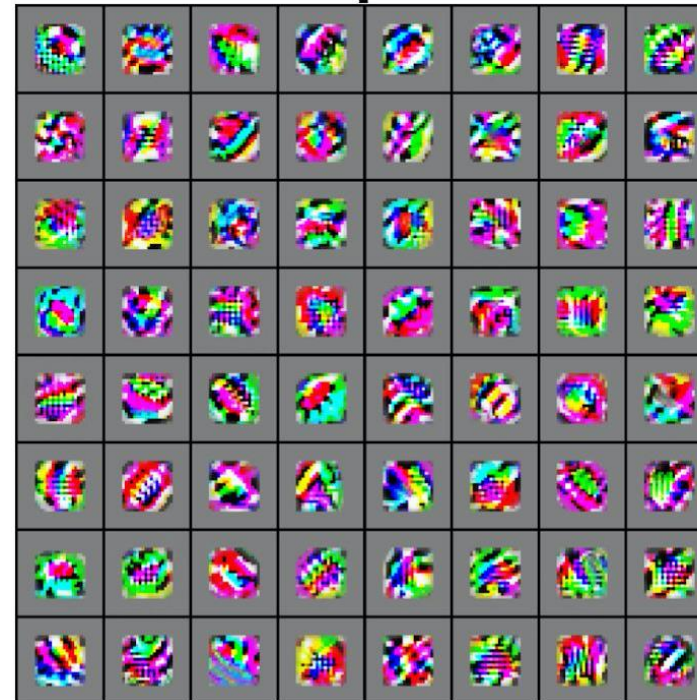
Mid-level features

High-level features

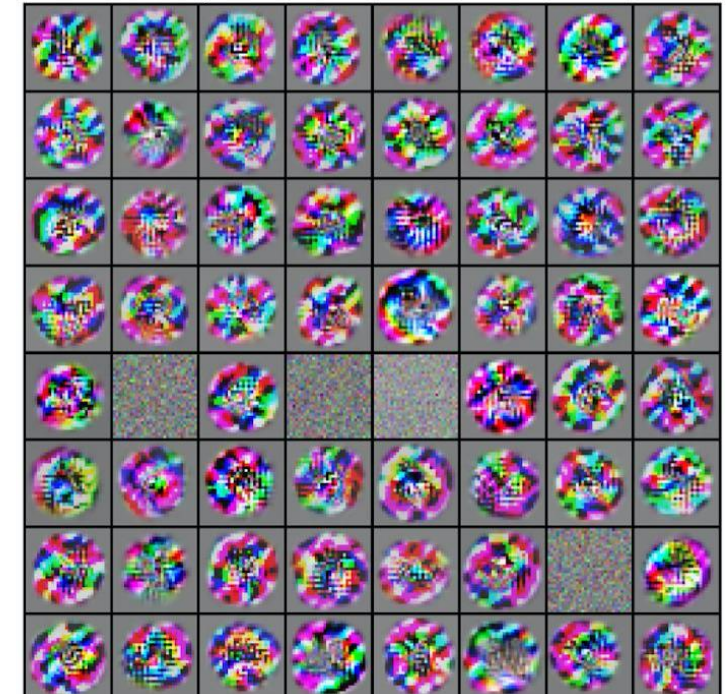
Linearly separable classifier



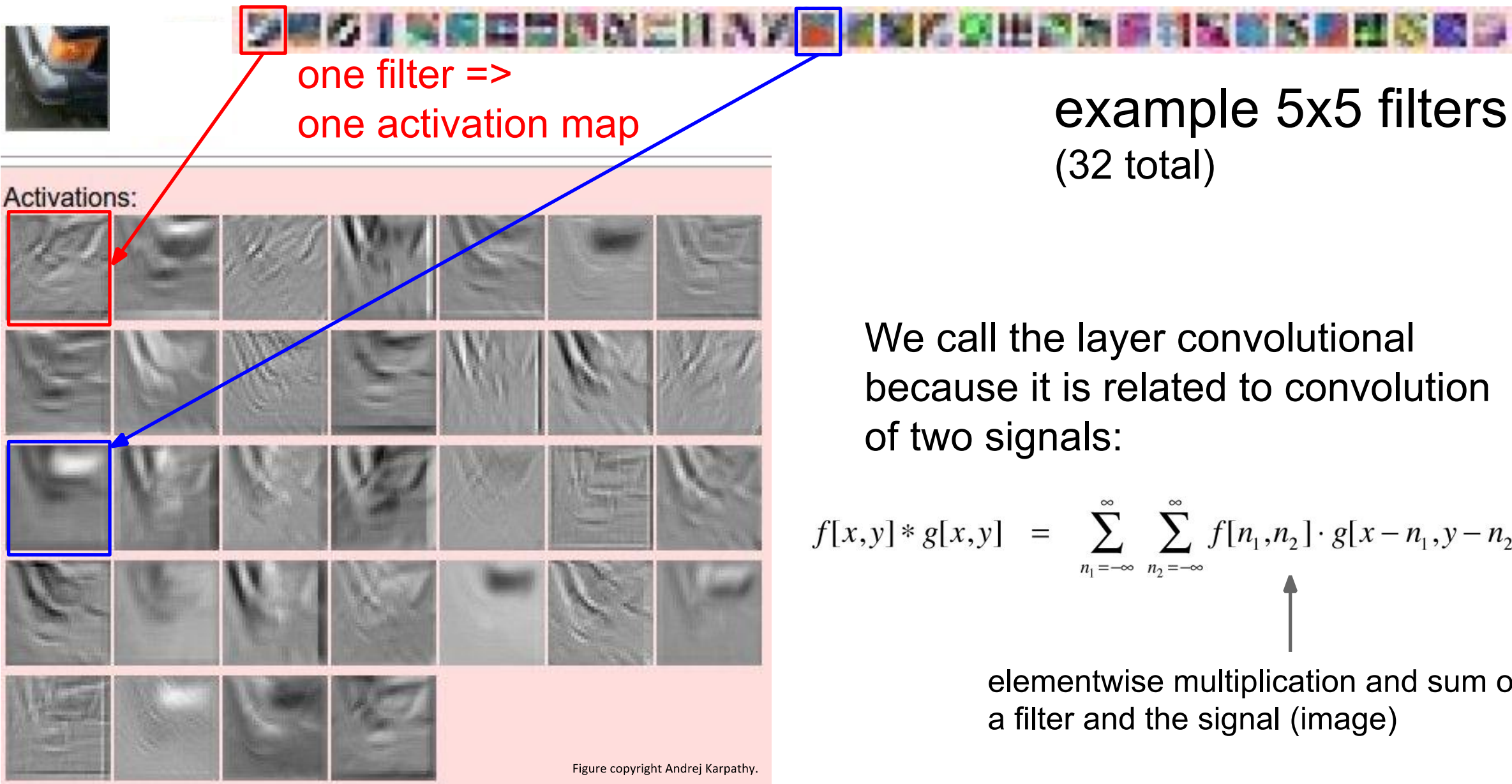
VGG-16 Conv1_1



VGG-16 Conv3_2



VGG-16 Conv5_3



We call the layer convolutional because it is related to convolution of two signals:

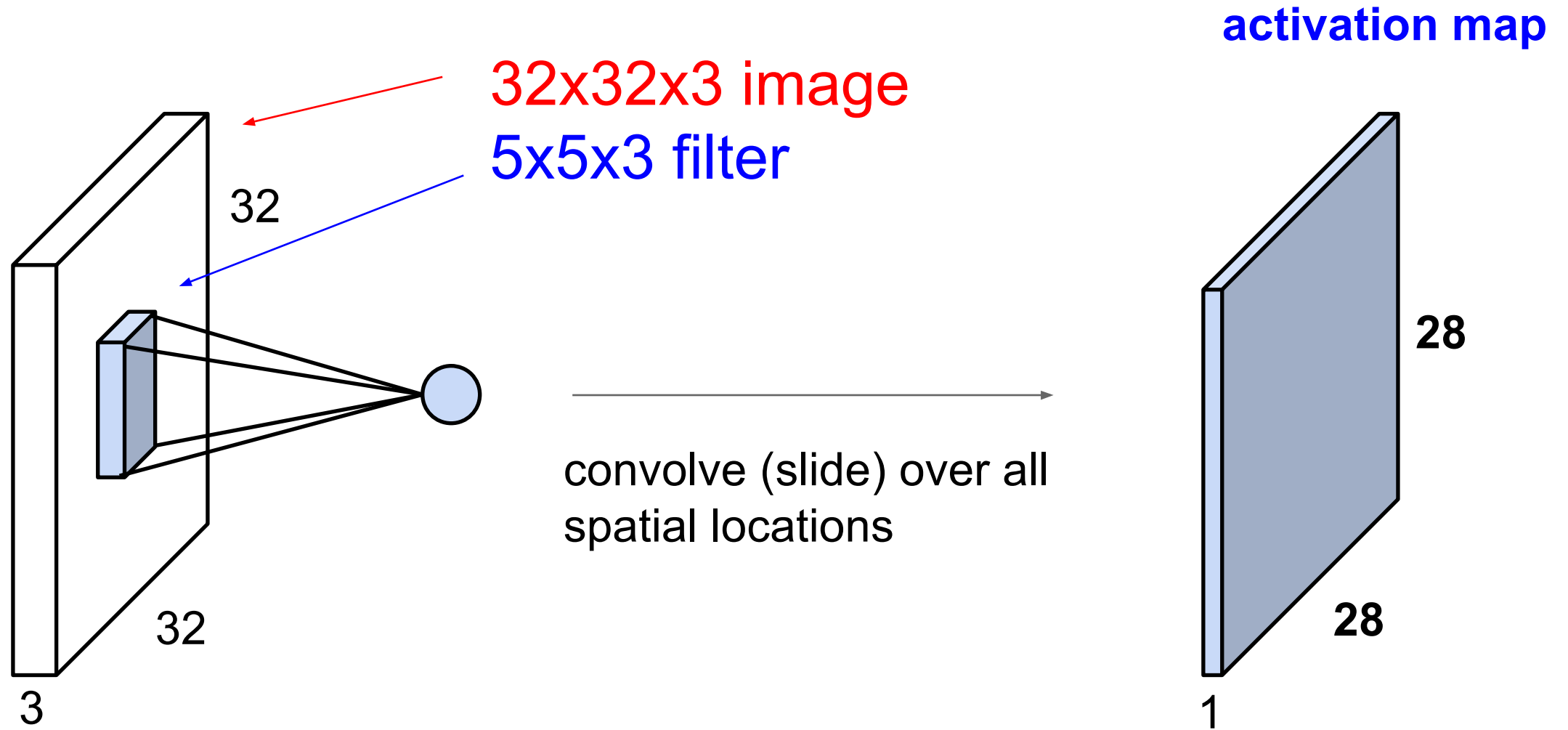
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

↑
elementwise multiplication and sum of
a filter and the signal (image)

A black Mazda3 is shown from a front-three-quarter view, parked on a light-colored paved surface. The car is positioned in front of a low, light-colored concrete wall. Behind the wall, there is a landscaped hillside with rocks and some dry vegetation. In the far background, a cityscape is visible under a hazy sky. The car's license plate is partially visible and appears to be a California plate.



A closer look at spatial dimensions:



A closer look at spatial dimensions:

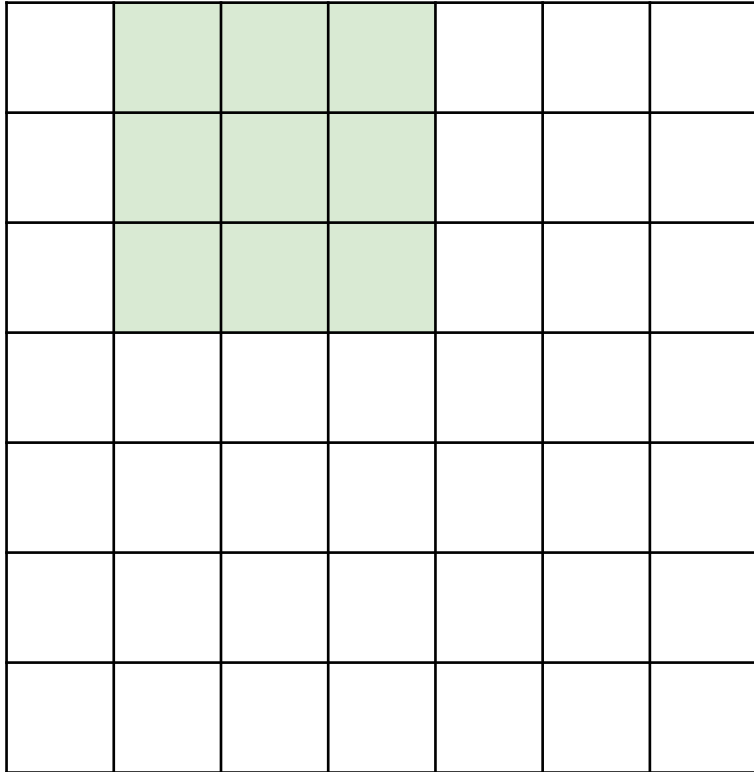
7

7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

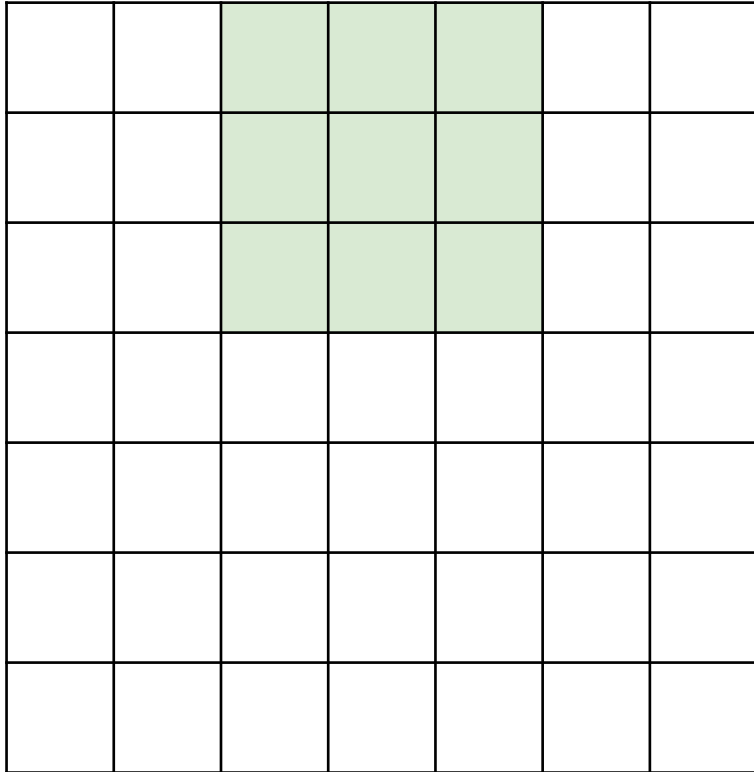


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7



7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

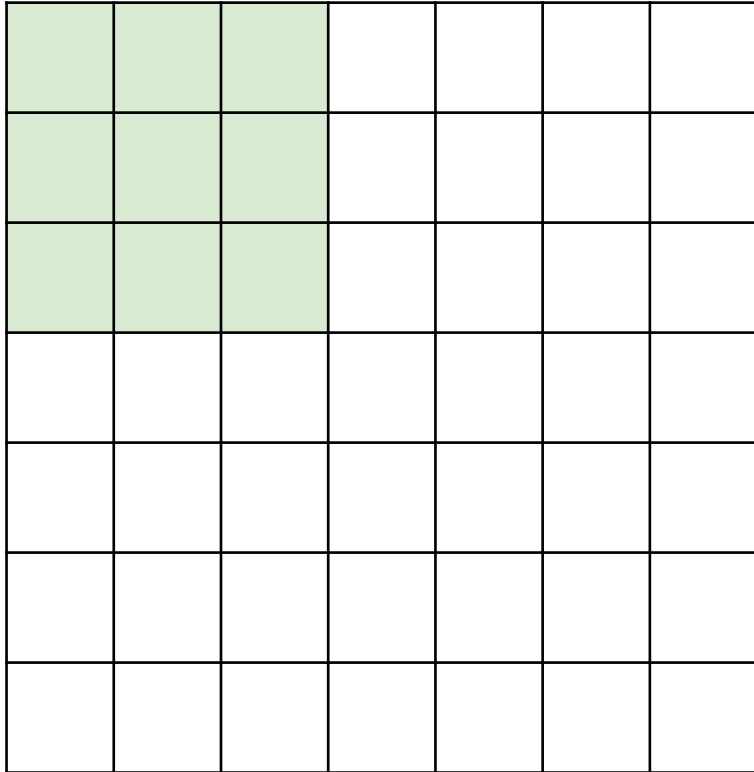
7

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

A closer look at spatial dimensions:

7

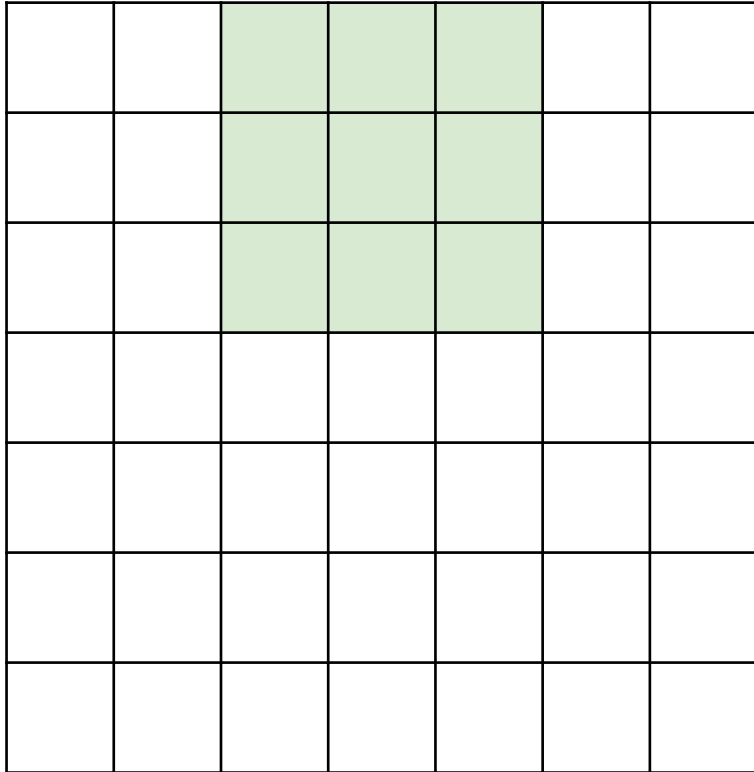


7

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

A closer look at spatial dimensions:

7

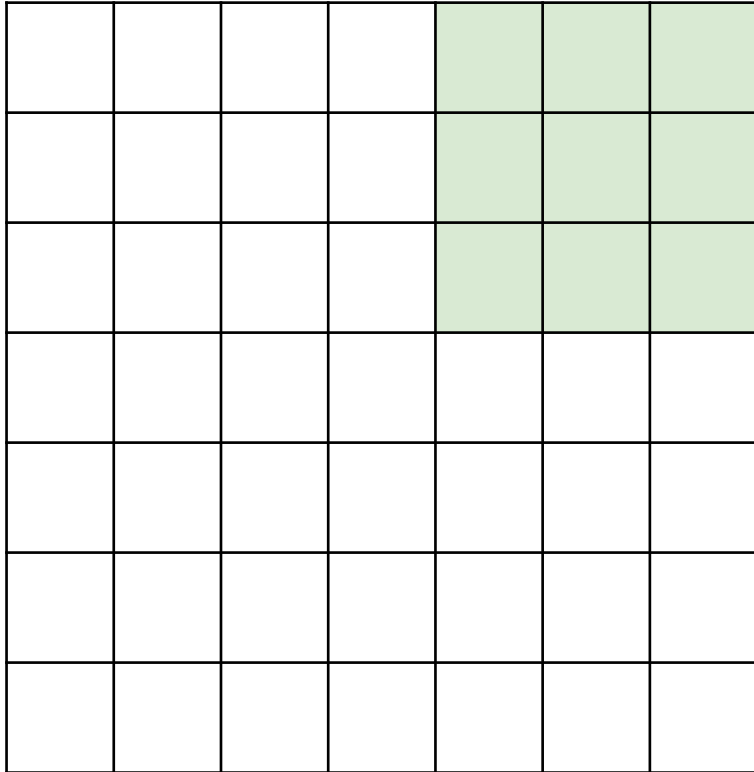


7

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

A closer look at spatial dimensions:

7

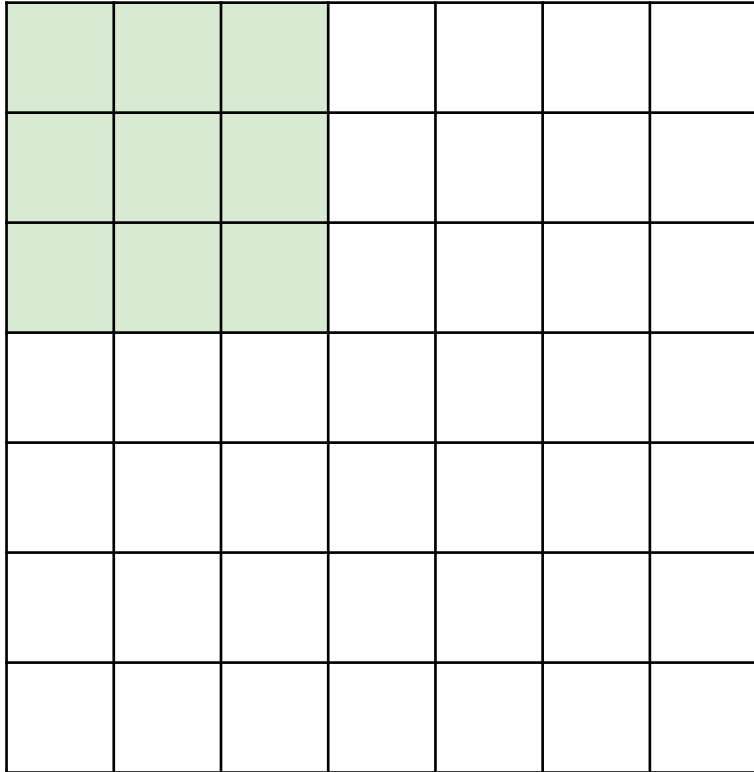


7

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

A closer look at spatial dimensions:

7

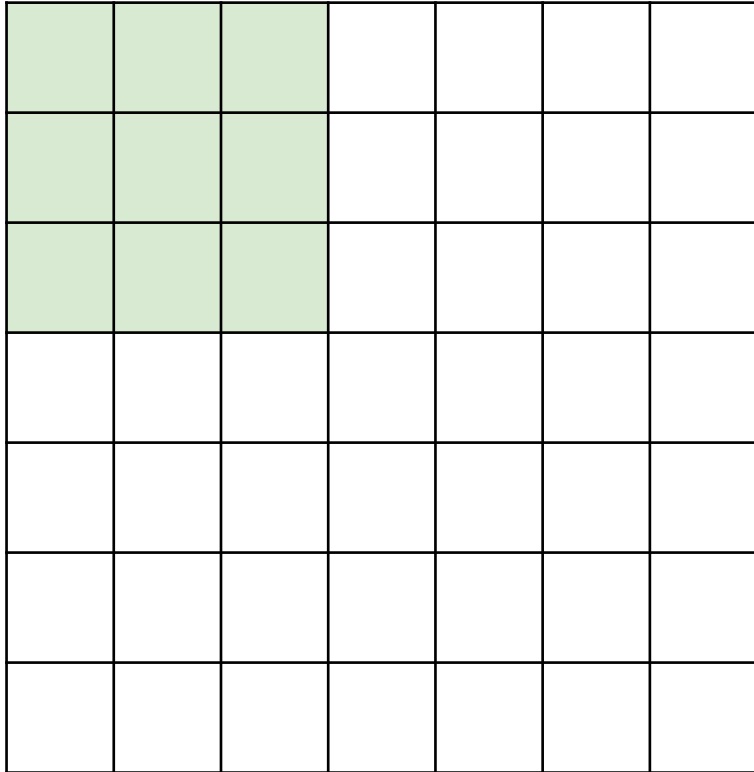


7

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

A closer look at spatial dimensions:

7

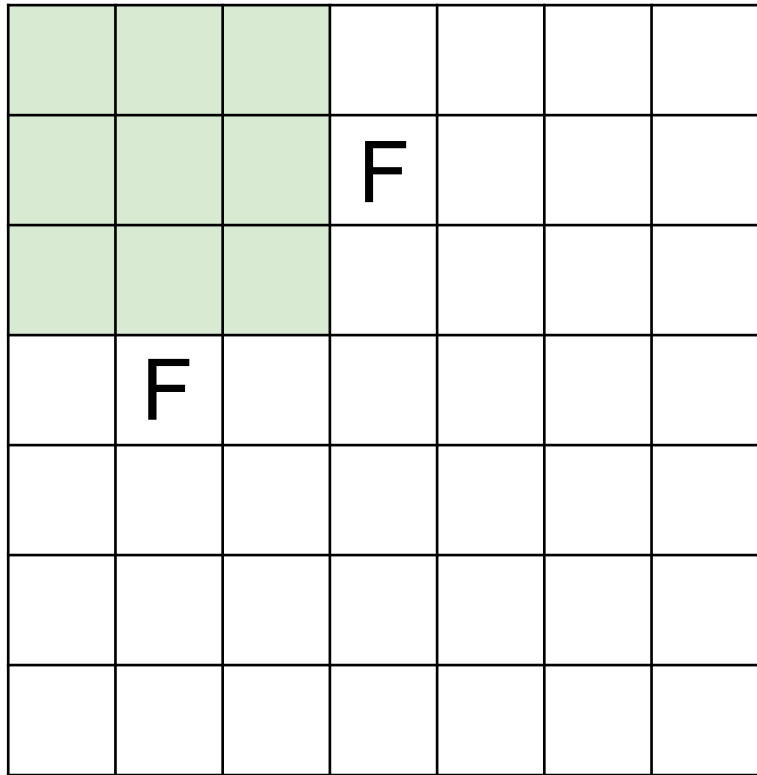


7

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

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

N



N

Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7, F = 3$:

stride 1 $\Rightarrow (7 - 3) / 1 + 1 = 5$

stride 2 $\Rightarrow (7 - 3) / 2 + 1 = 3$

stride 3 $\Rightarrow (7 - 3) / 3 + 1 = 2.33 : \backslash$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

(recall:)

$$(N + 2P - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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 $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

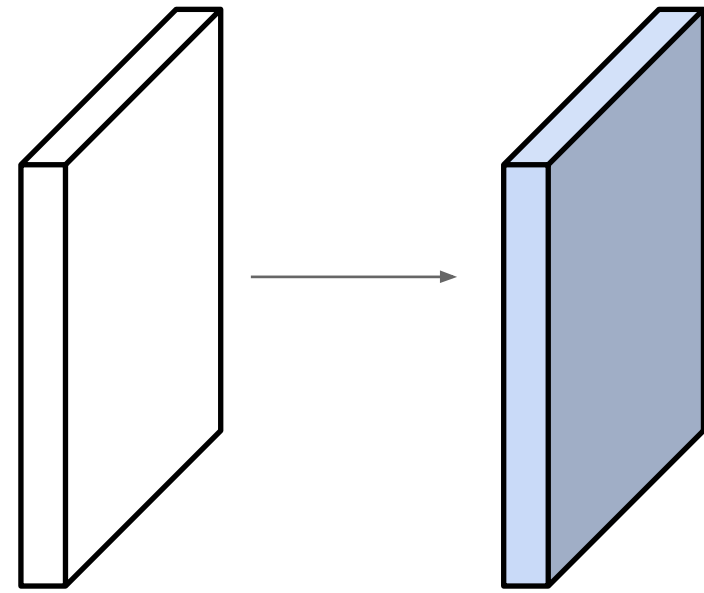
$F = 7 \Rightarrow$ zero pad with 3

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:

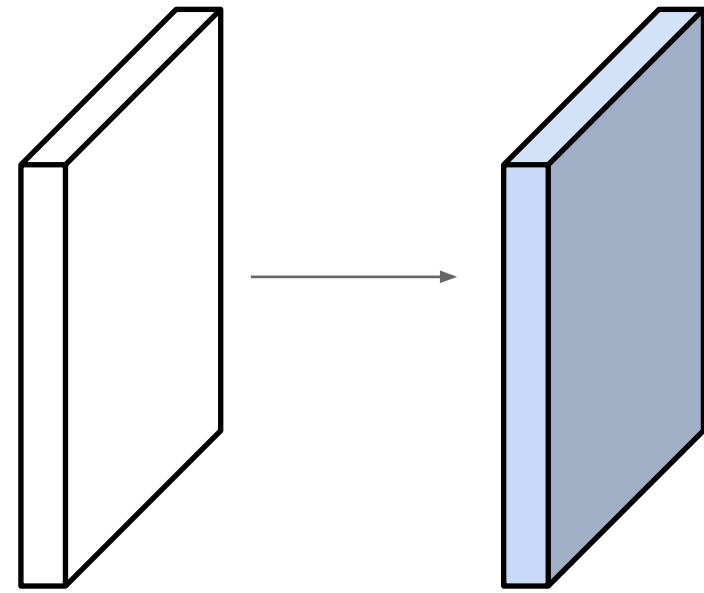
Input volume: **32x32x3**

10 **5x5** filters with stride **1**, pad **2**

Output volume size:

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

32x32x10

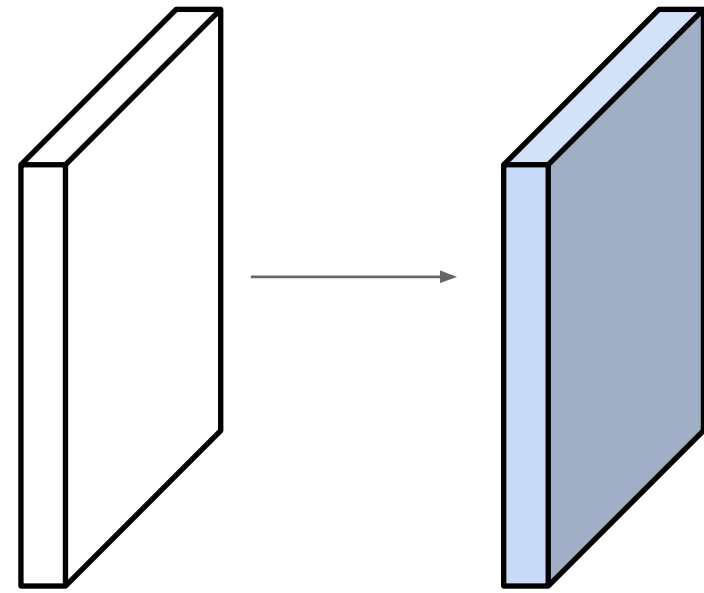


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

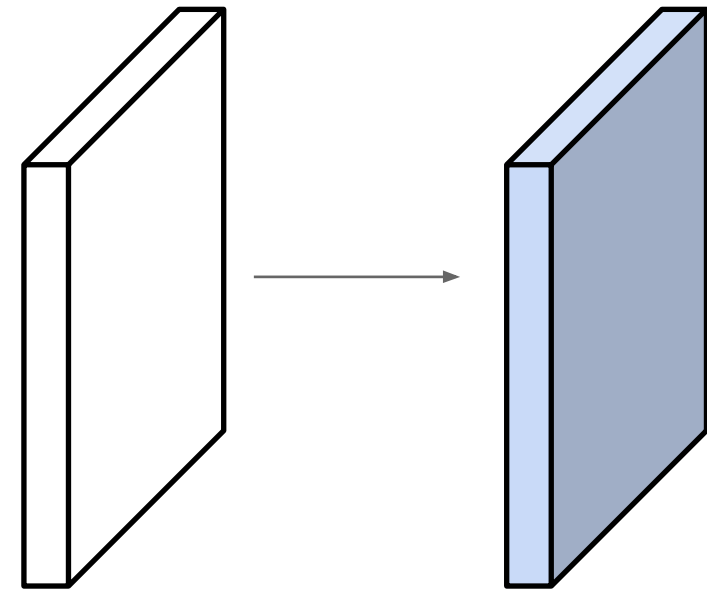
Number of parameters in this layer?



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)

$\Rightarrow 76*10 = 760$

Convolution layer: summary

Common settings:

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

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

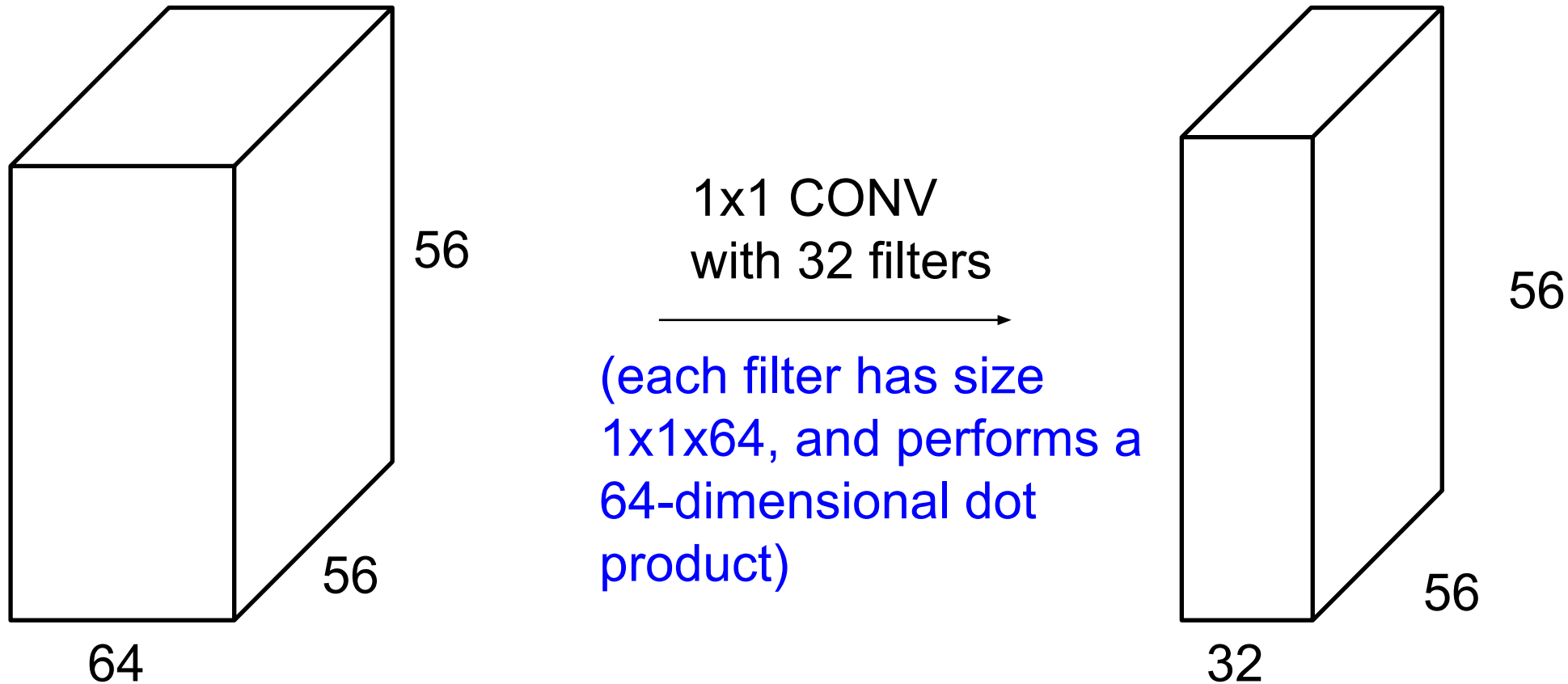
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ (whatever fits)
- $F = 1, S = 1, P = 0$

This will produce an output of $W_2 \times H_2 \times K$
where:

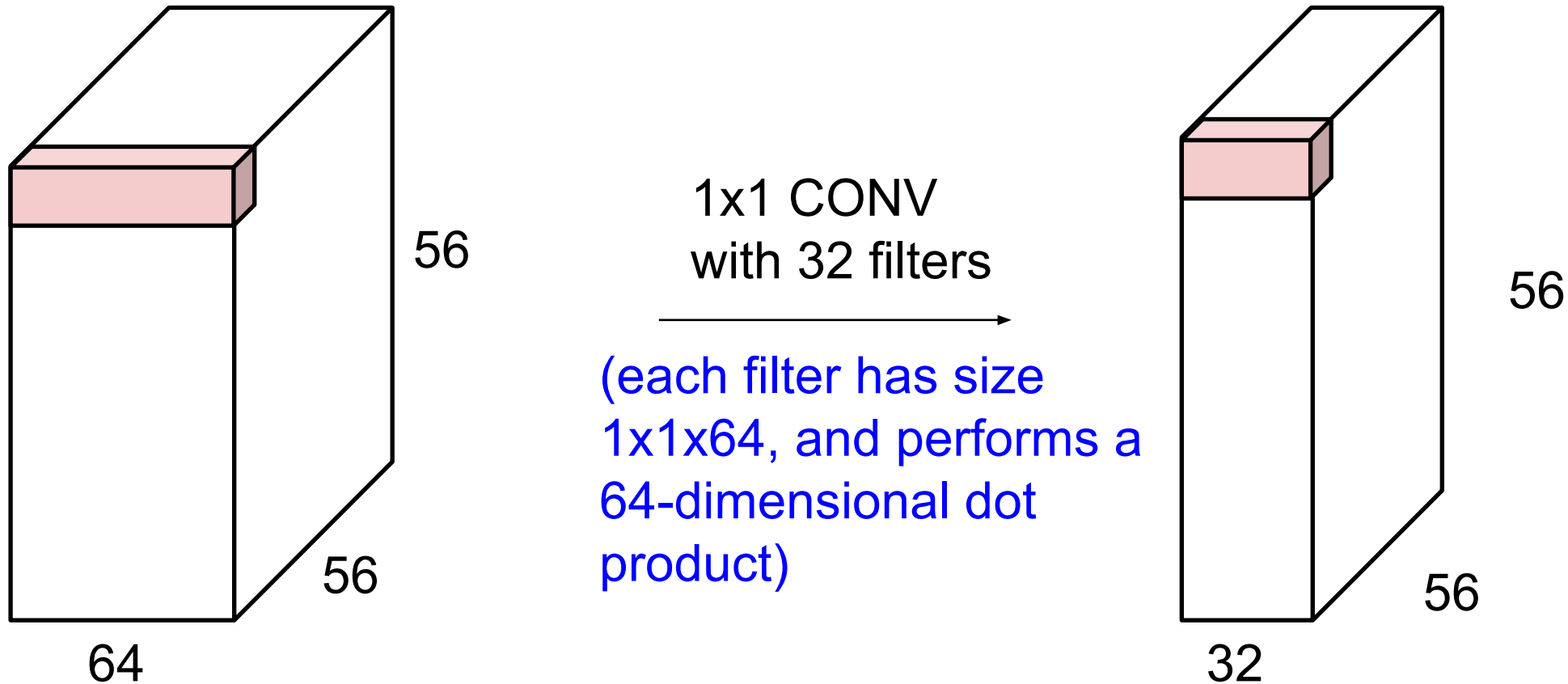
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

(btw, 1x1 convolution layers make perfect sense)



(btw, 1x1 convolution layers make perfect sense)



Example: CONV layer in PyTorch

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
```

[SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{out}, H_{out}, W_{out})$ can be precisely described as:

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k)$$

where \star is the valid 2D **cross-correlation** operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this [link](#) has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
 - At `groups=1`, all inputs are convolved to all outputs.
 - At `groups=2`, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
 - At `groups= in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{out}}{C_{in}} \right\rfloor$.

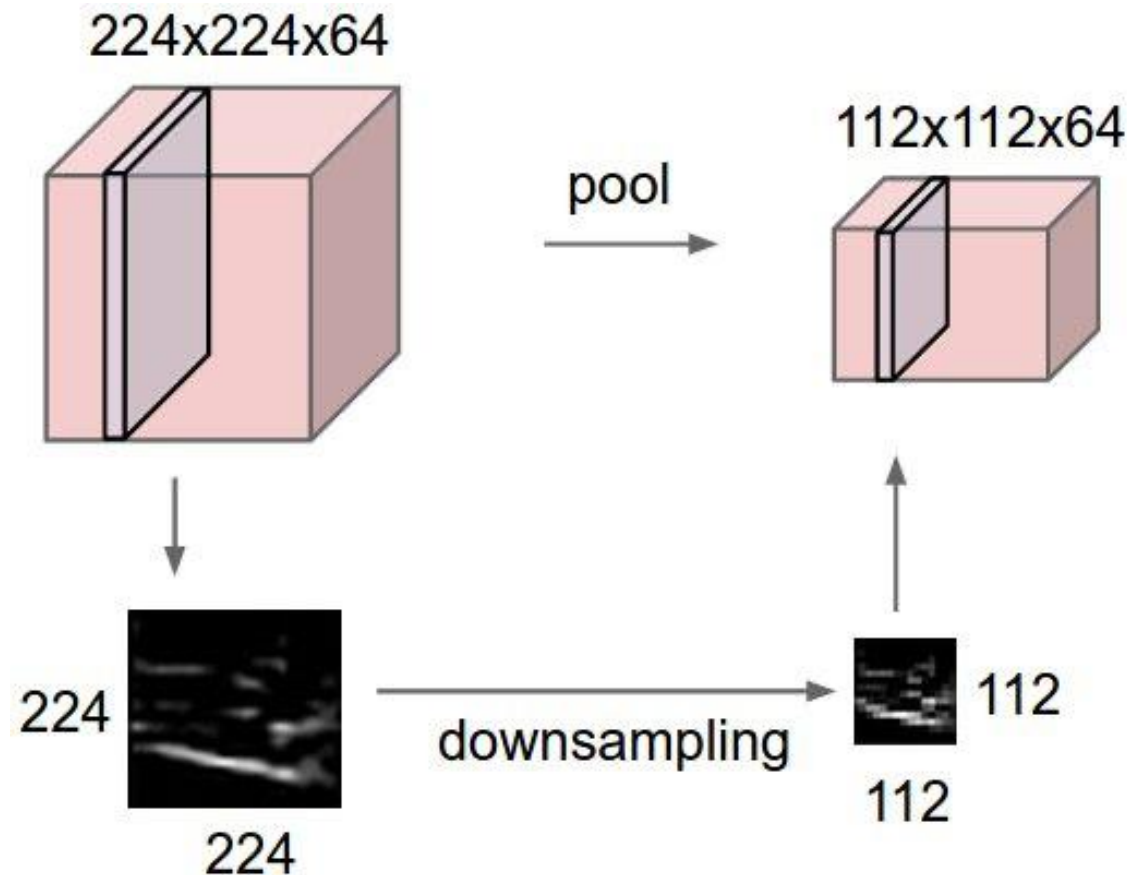
The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

PyTorch is licensed under [BSD 3-clause](#).

Pooling layer

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



MAX POOLING

Single depth slice

x ↑

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→ y

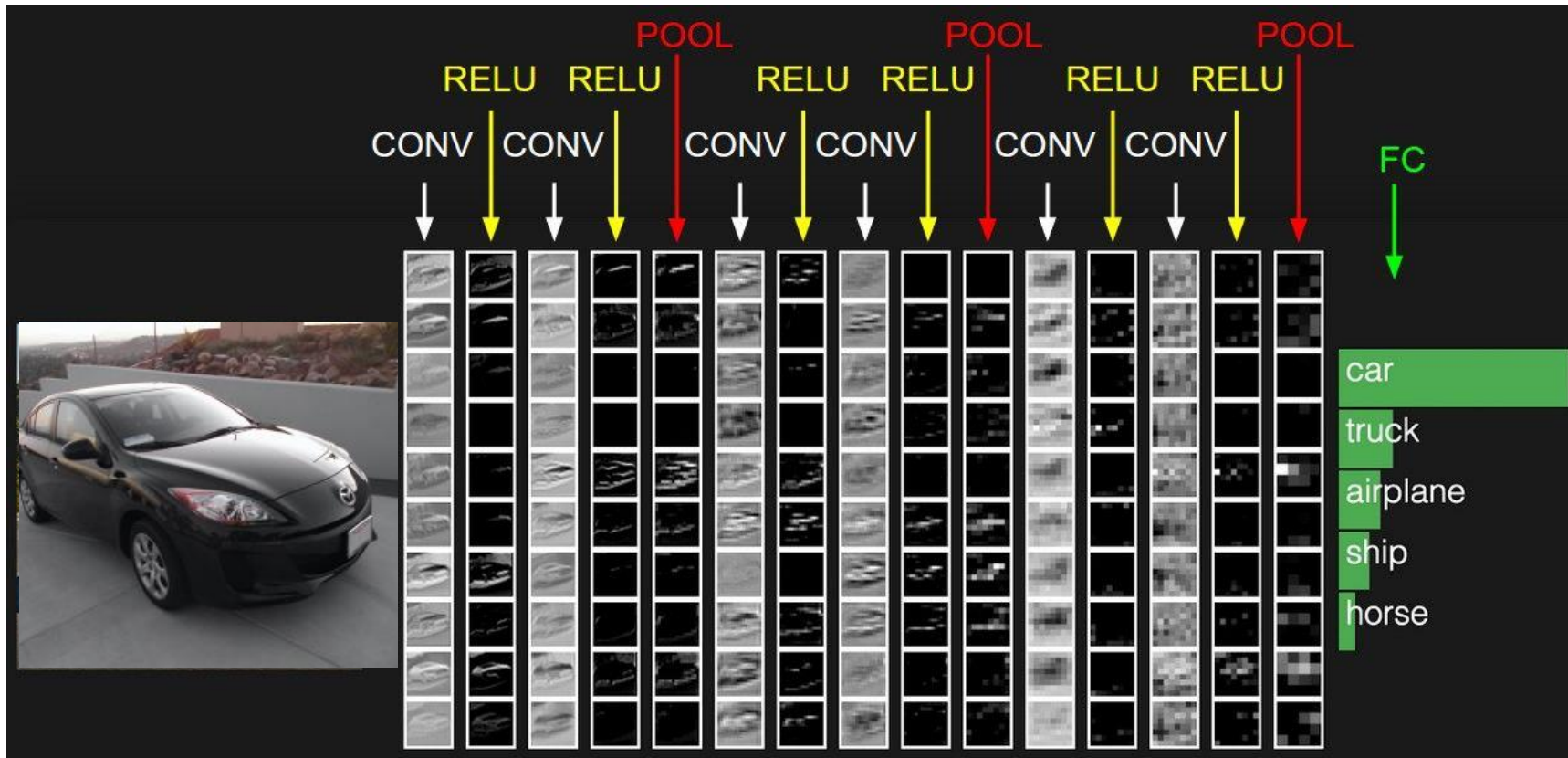
max pool with 2x2 filters
and stride 2



6	8
3	4

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

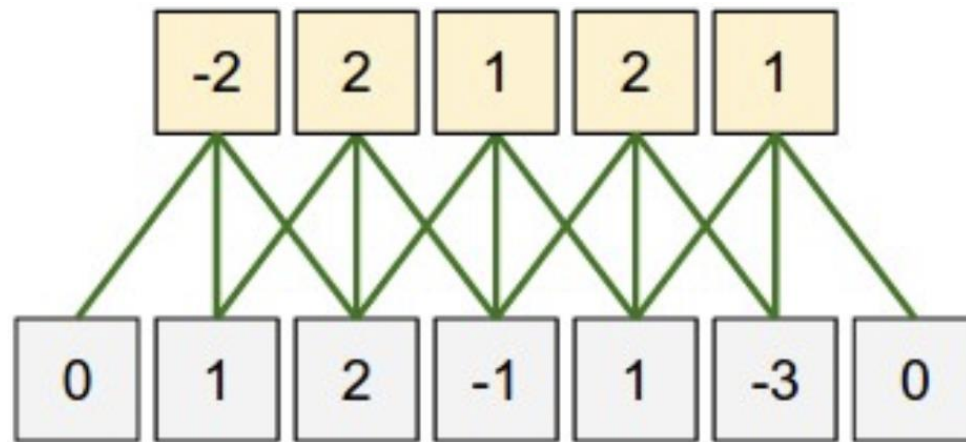


Summary

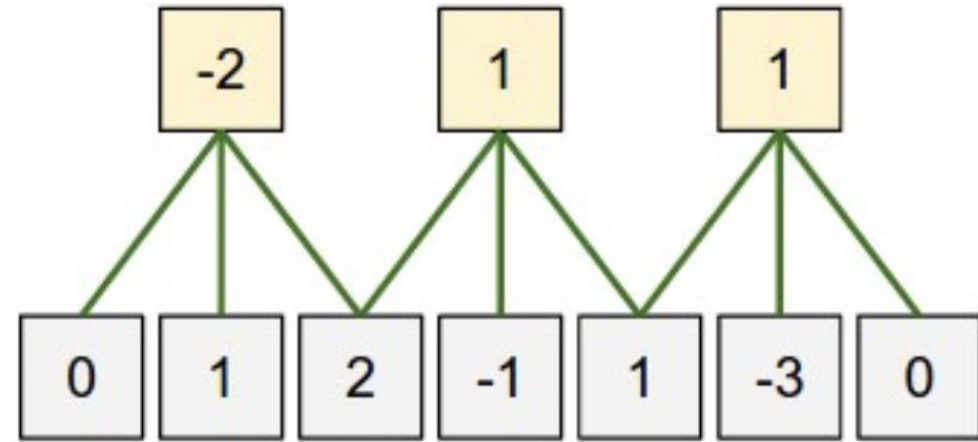
- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
 $[(\text{CONV-RELU})^N\text{-POOL?}]^M\text{-(FC-RELU)}^K, \text{SOFTMAX}$
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet have challenged this paradigm

1D ConvNet

- Used for processing sequences
 - Sentences, audios, time-series



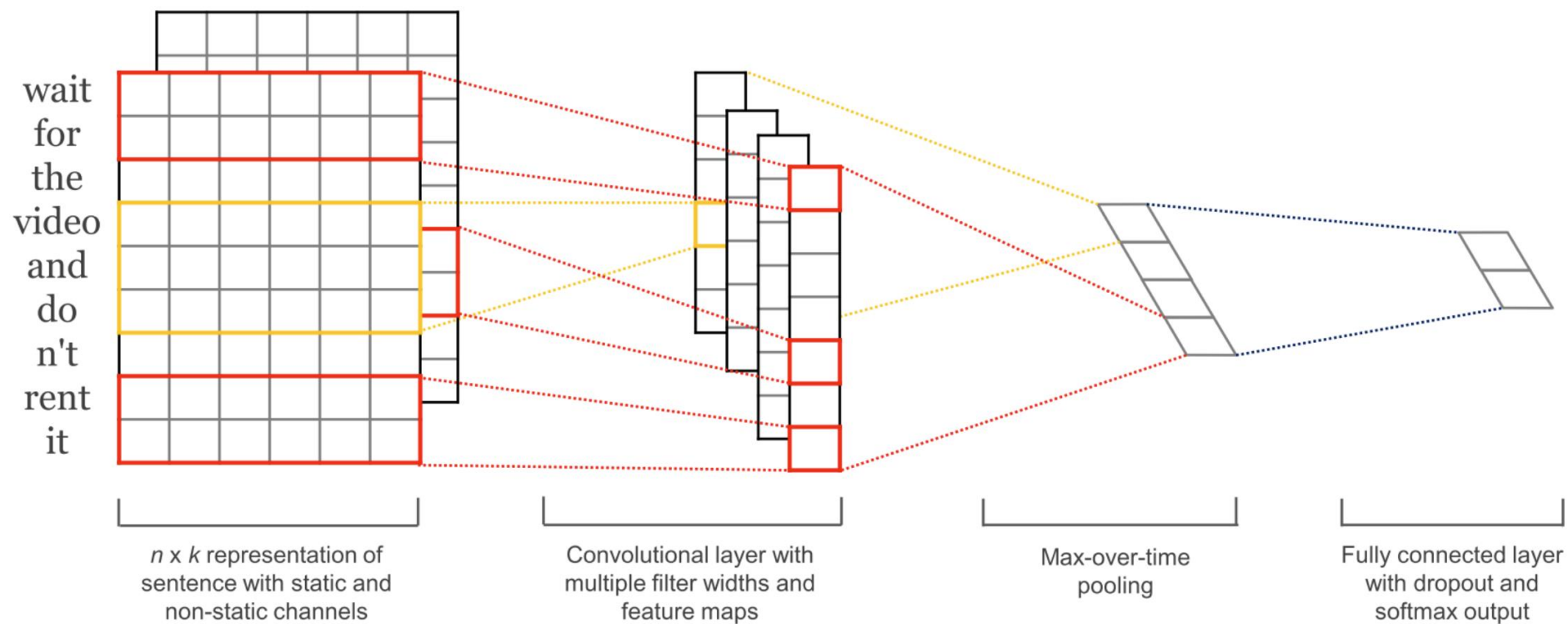
Filter size: 3, Stride: 1



Filter size: 3, Stride: 2

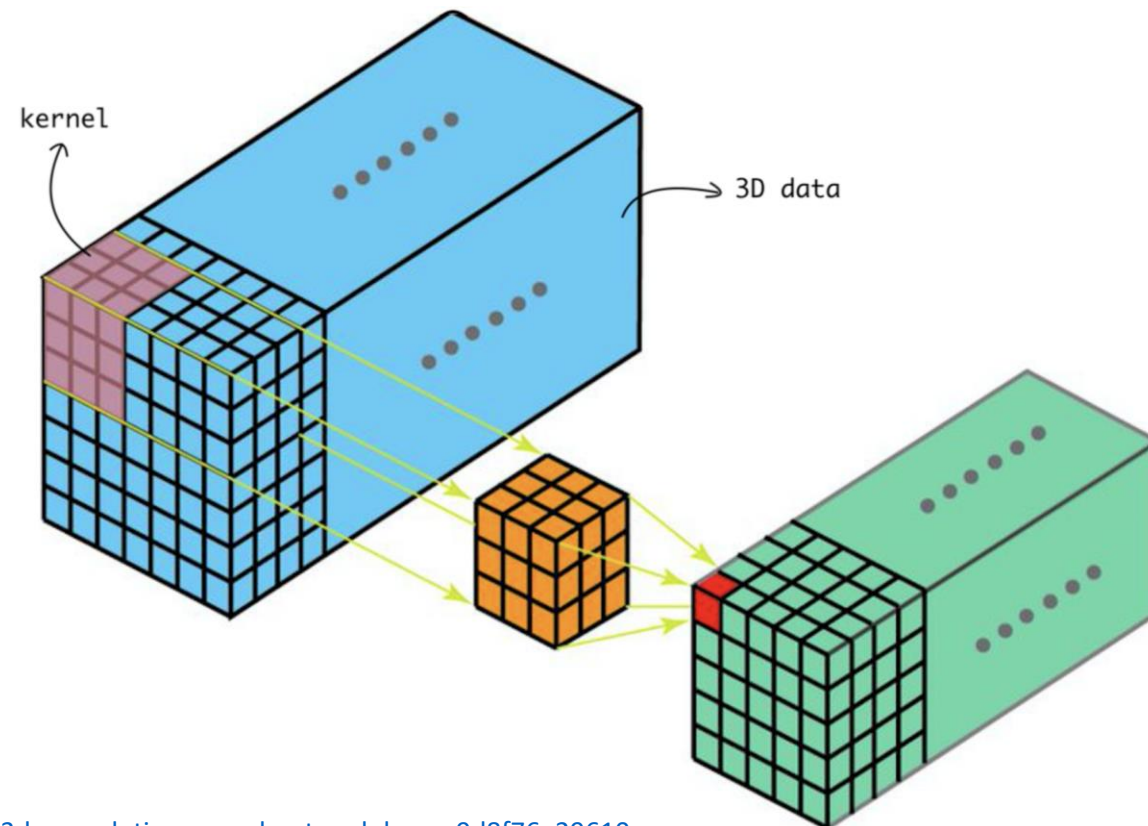
1D ConvNet

- Processing a sequence of word embeddings

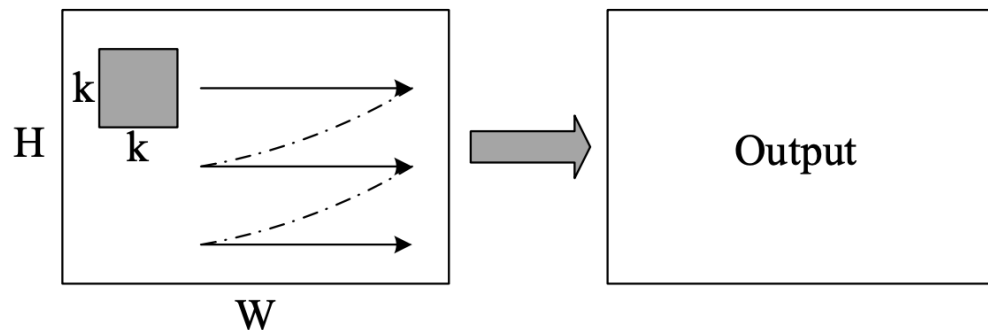


3D ConvNet

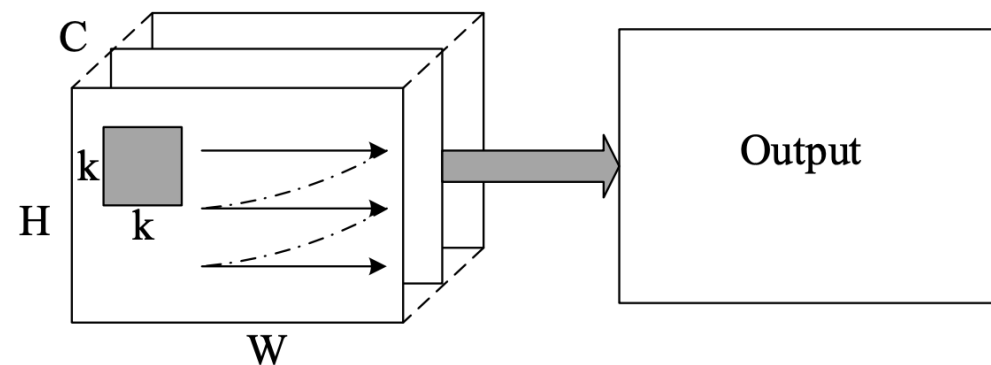
- Used for processing 3D tensors
 - Sequence of images (CT scans), videos



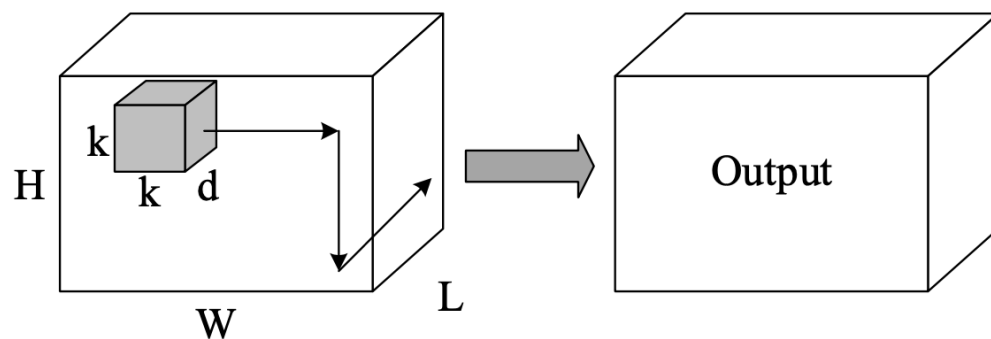
2D Conv VS 3D Conv



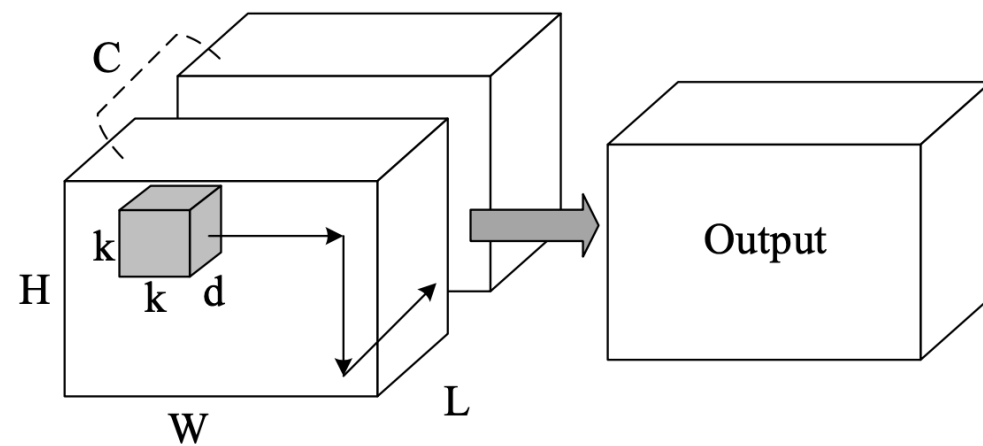
(a) 2D convolution on an image



(b) 2D convolution on multiple channels



(c) 3D convolution on a volume



(d) 3D convolution on multiple channels

Training Technique

Normalization

- It is usually a good idea to normalize your input to $N(0, 1)$
 - 0 mean, unit variance
 - Assumes normally distributed data though

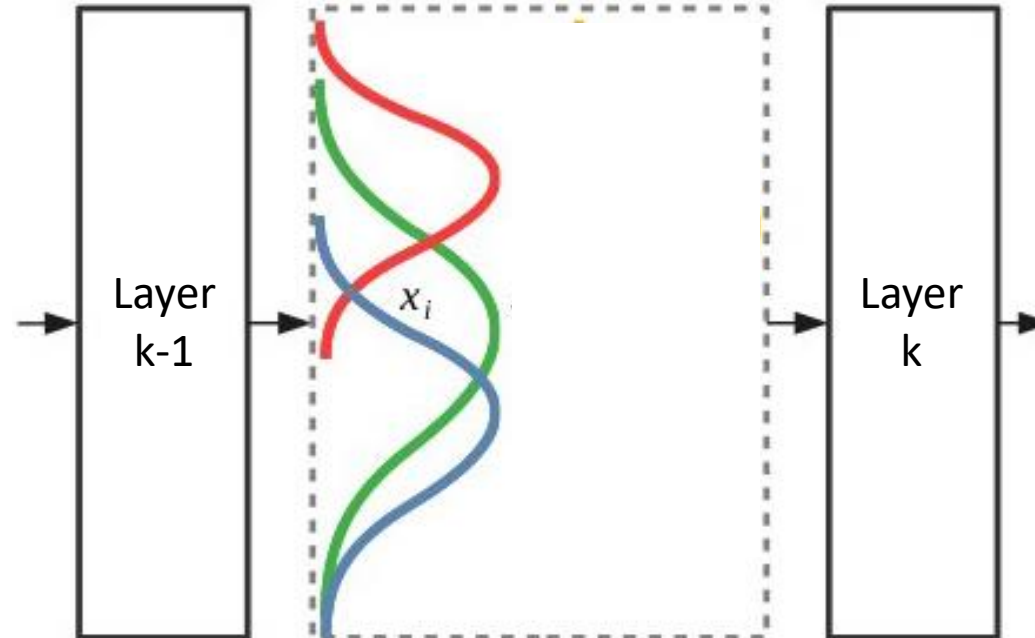
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Distribution Shift

- If your test data are wildly different from the training data...
 - Training data: MNIST
 - Test data: CIFAR-10
- Machine learning models are usually vulnerable to distribution shift

Internal Distribution Shift

- Layer k-1's activation is layer k's input.
- Parameter update changes layer k-1's activation.
 - Forces layer k to adapt to new distribution after every update.



Batch Normalization

- Let's remove internal distribution shift.
 - BatchNorm paper call is “Internal Covariate Shift”.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

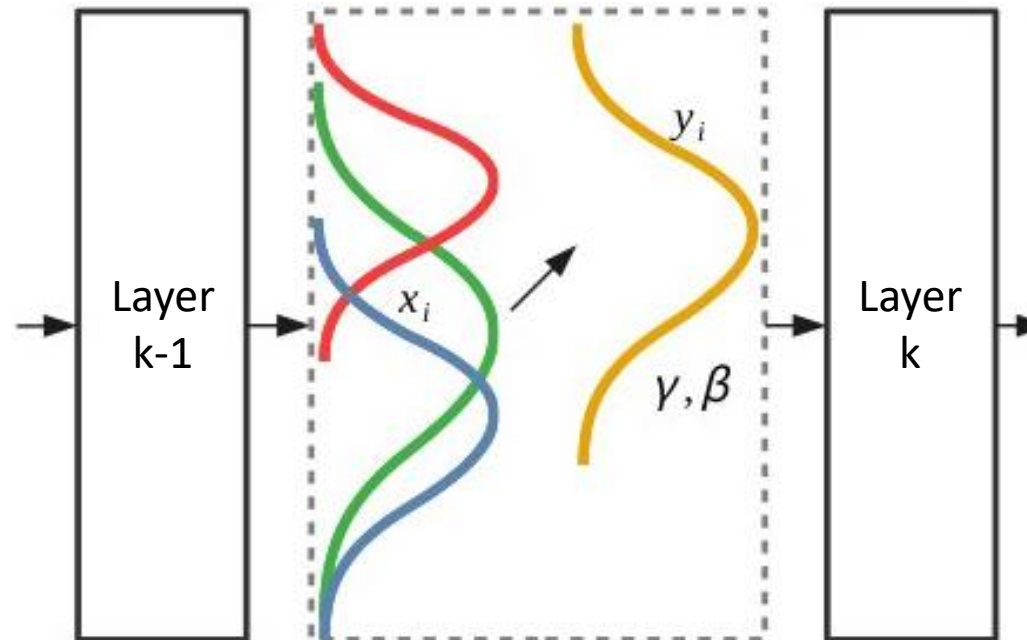
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

Internal Distribution Shift

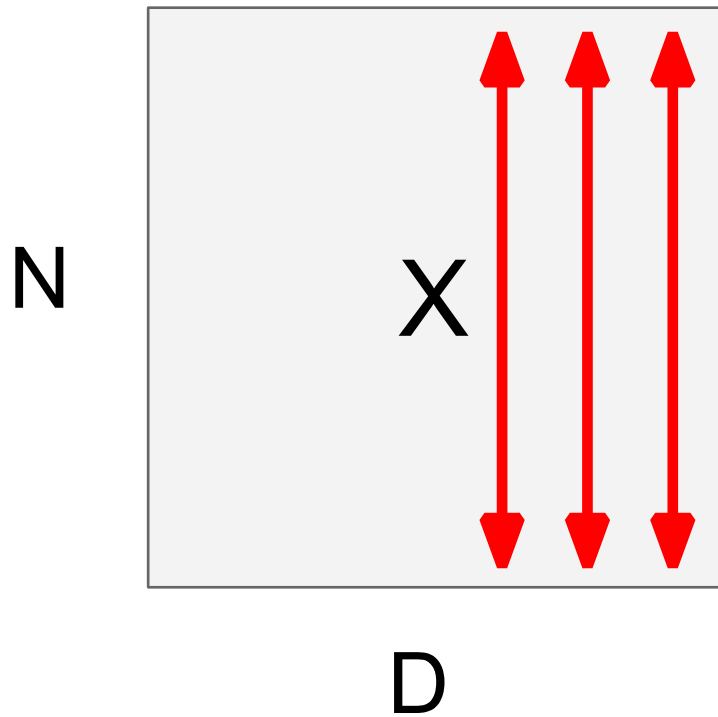
- Layer k-1's activation is layer k's input.
- Parameter update changes layer k-1's activation.
- Fix the distribution of the internal layer's activation!



Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



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

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

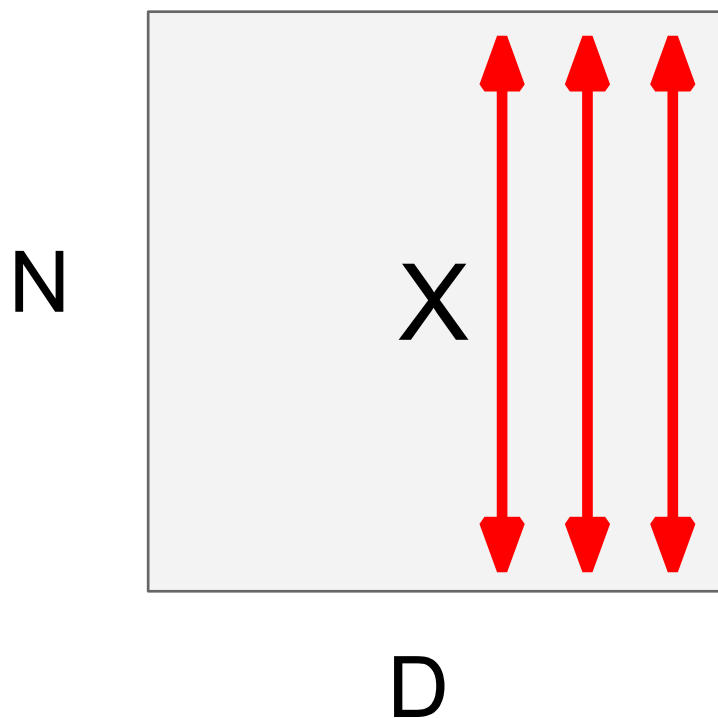
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x ,
Shape is $N \times D$

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



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

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

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

Normalized x ,
Shape is $N \times D$

Problem: What if zero-mean, unit
variance is too hard of a constraint?

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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: Test-Time

Estimates depend on minibatch;
can't do this at test-time!

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: Test-Time

Input: $x : N \times D$

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

Per-channel mean,
shape is D

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

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

Per-channel var,
shape is D

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

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

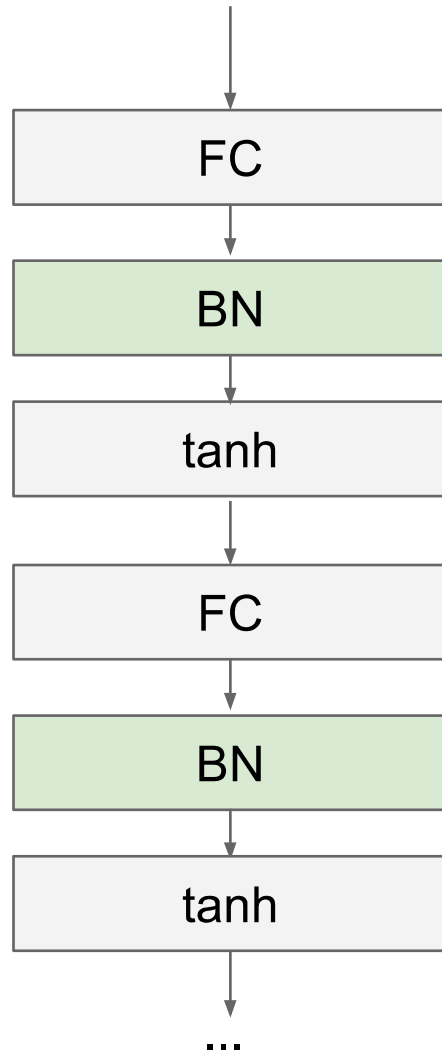
Normalized x,
Shape is N x D

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

Output,
Shape is N x D

Batch Normalization

[Ioffe and Szegedy, 2015]



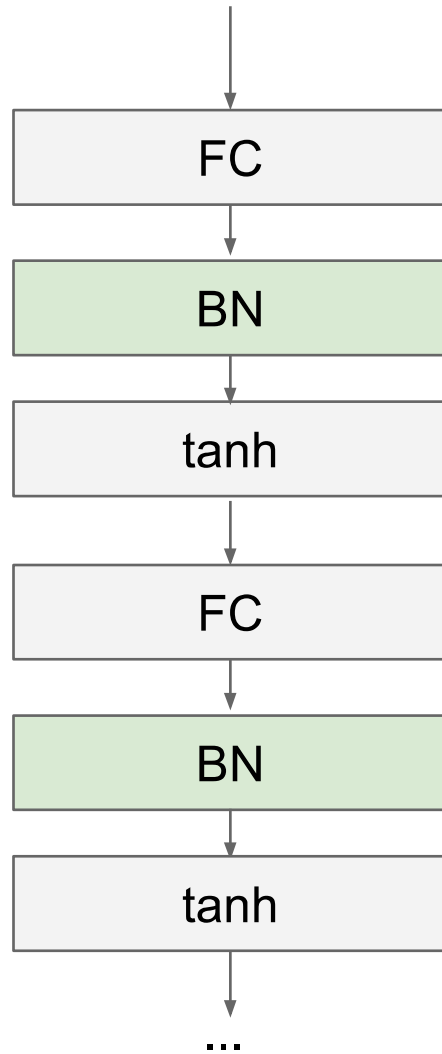
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

A topic of debate

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Batch Normalization

[Ioffe and Szegedy, 2015]



- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$$\mathbf{x} : \mathbf{N} \times \mathbf{D}$$

Normalize 

$$\boldsymbol{\mu}, \boldsymbol{\sigma} : \mathbf{1} \times \mathbf{D}$$

$$\gamma, \beta : \mathbf{1} \times \mathbf{D}$$

$$\mathbf{y} = \gamma (\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x} : \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W}$$

Normalize   

$$\boldsymbol{\mu}, \boldsymbol{\sigma} : \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1}$$

$$\gamma, \beta : \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1}$$

$$\mathbf{y} = \gamma (\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta$$

Layer Normalization

Batch Normalization for
fully-connected networks

$$\mathbf{x} : \mathbf{N} \times \mathbf{D}$$

Normalize



$$\boldsymbol{\mu}, \boldsymbol{\sigma} : \mathbf{1} \times \mathbf{D}$$

$$\boldsymbol{\gamma}, \boldsymbol{\beta} : \mathbf{1} \times \mathbf{D}$$

$$\mathbf{y} = \boldsymbol{\gamma} (\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$$\mathbf{x} : \mathbf{N} \times \mathbf{D}$$

Normalize



$$\boldsymbol{\mu}, \boldsymbol{\sigma} : \mathbf{N} \times \mathbf{1}$$

$$\boldsymbol{\gamma}, \boldsymbol{\beta} : \mathbf{1} \times \mathbf{D}$$

$$\mathbf{y} = \boldsymbol{\gamma} (\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Ba, Kiros, and Hinton, "Layer Normalization", arXiv 2016

Instance Normalization

Batch Normalization for
convolutional networks

$\mathbf{x} : \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W}$

Normalize



$\boldsymbol{\mu}, \boldsymbol{\sigma} : \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1}$

$\boldsymbol{\gamma}, \boldsymbol{\beta} : \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1}$

$$\mathbf{y} = \boldsymbol{\gamma} (\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Instance Normalization for
convolutional networks
Same behavior at train / test!

$\mathbf{x} : \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W}$

Normalize



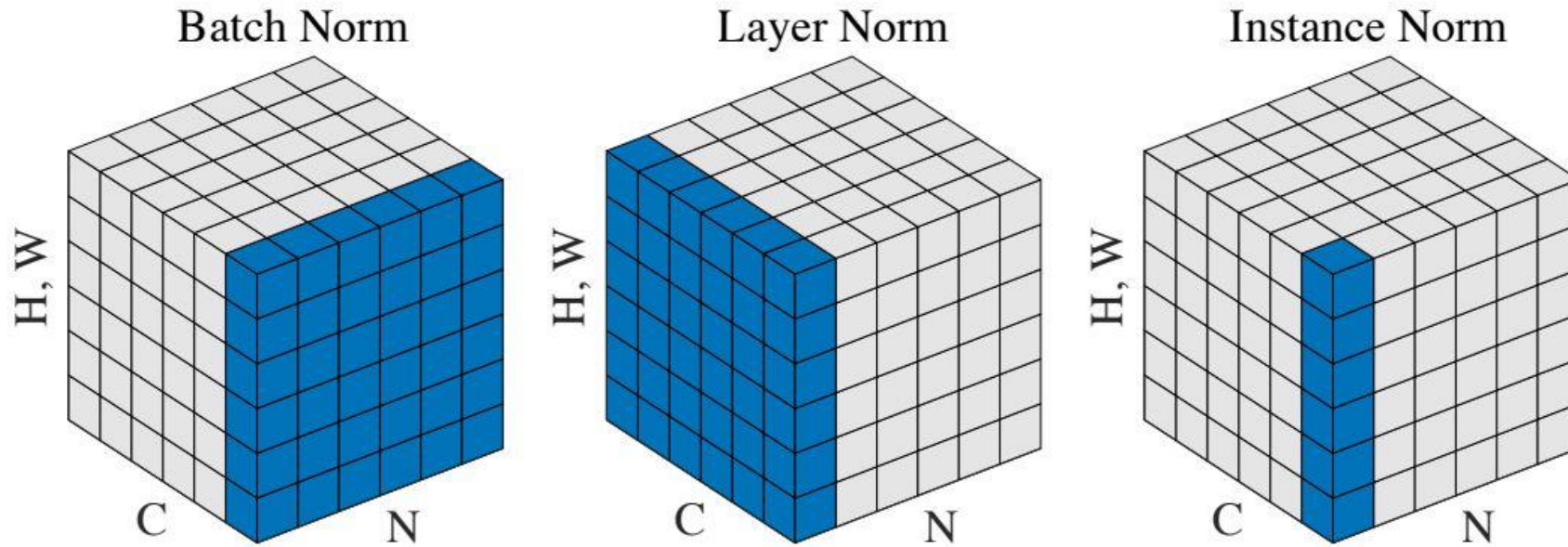
$\boldsymbol{\mu}, \boldsymbol{\sigma} : \mathbf{N} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1}$

$\boldsymbol{\gamma}, \boldsymbol{\beta} : \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1}$

$$\mathbf{y} = \boldsymbol{\gamma} (\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

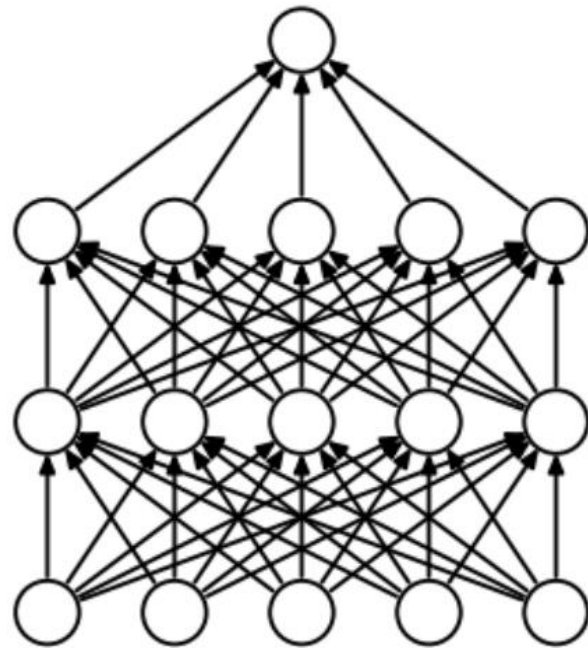
Comparison of Normalization Layers



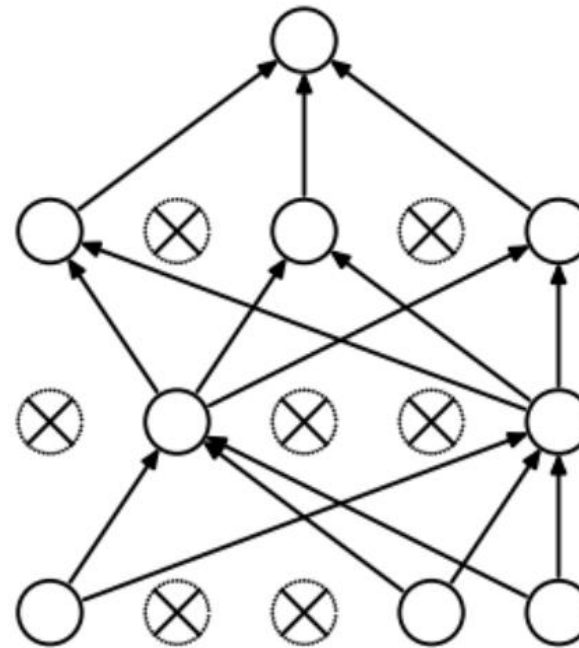
Wu and He, "Group Normalization", ECCV 2018

Dropout

- Set p (e.g. 10%) hidden nodes to zero.
 - p is a hyperparameter.
 - Type of regularization.

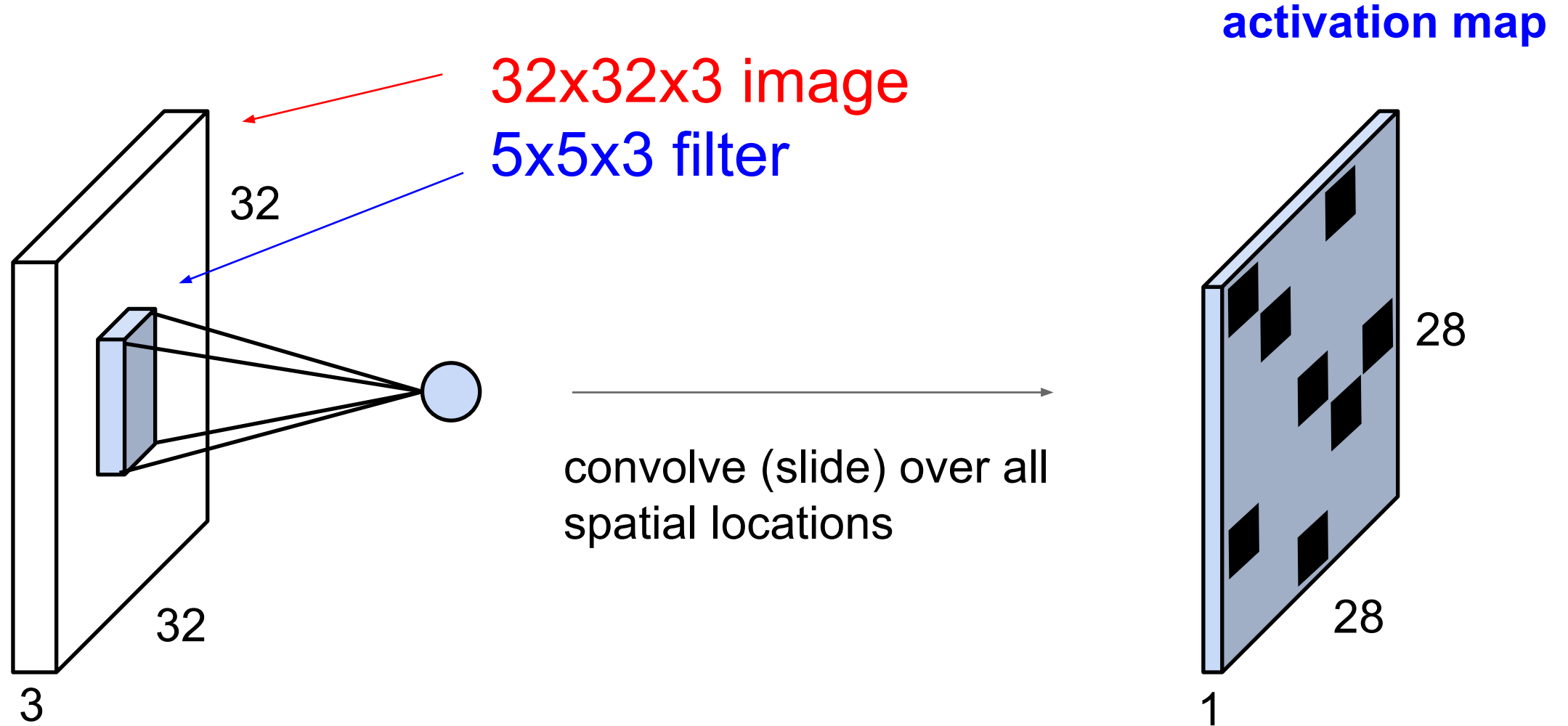


(a) Standard Neural Net

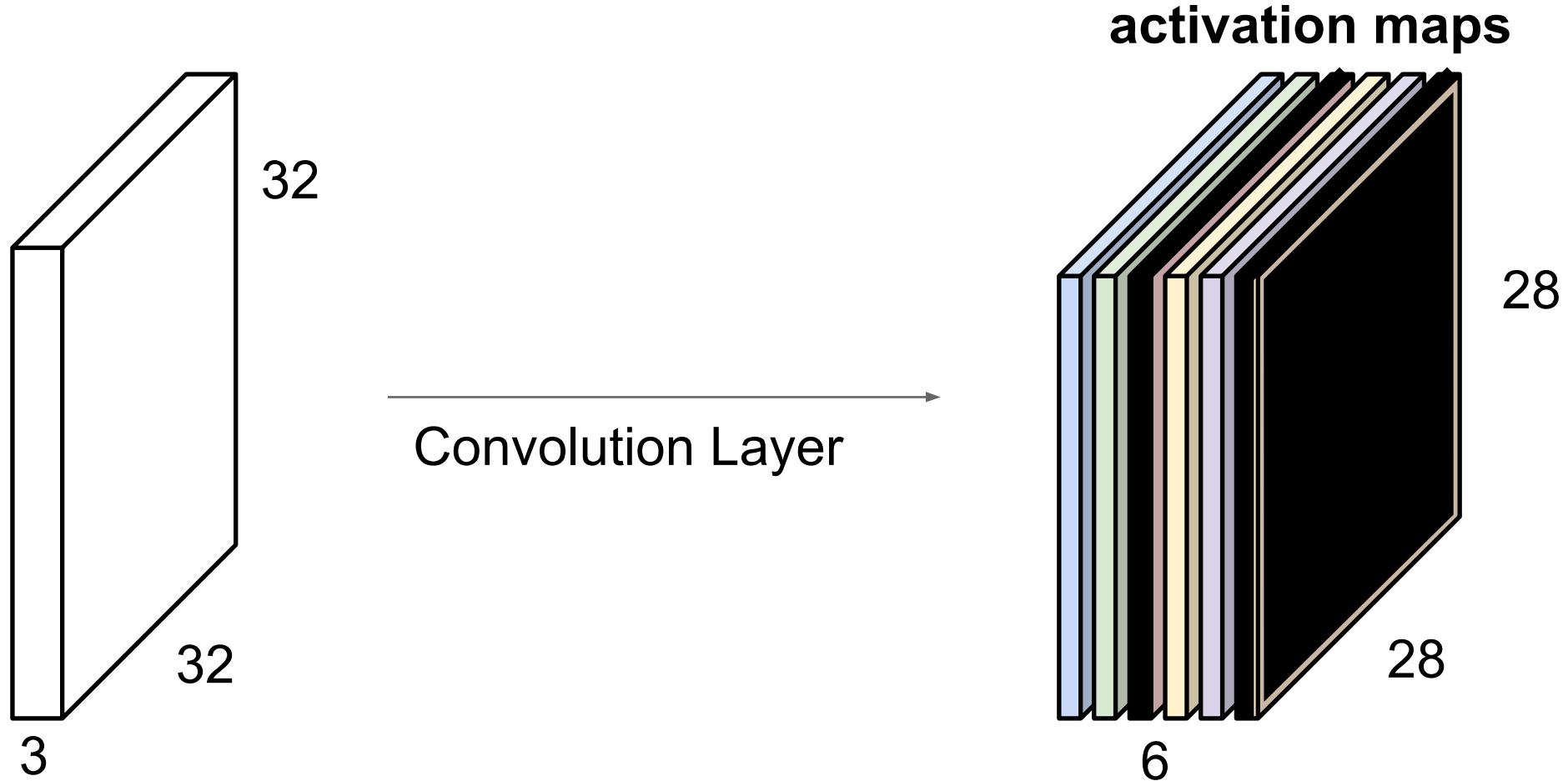


(b) After applying dropout.

Dropout in ConvNet



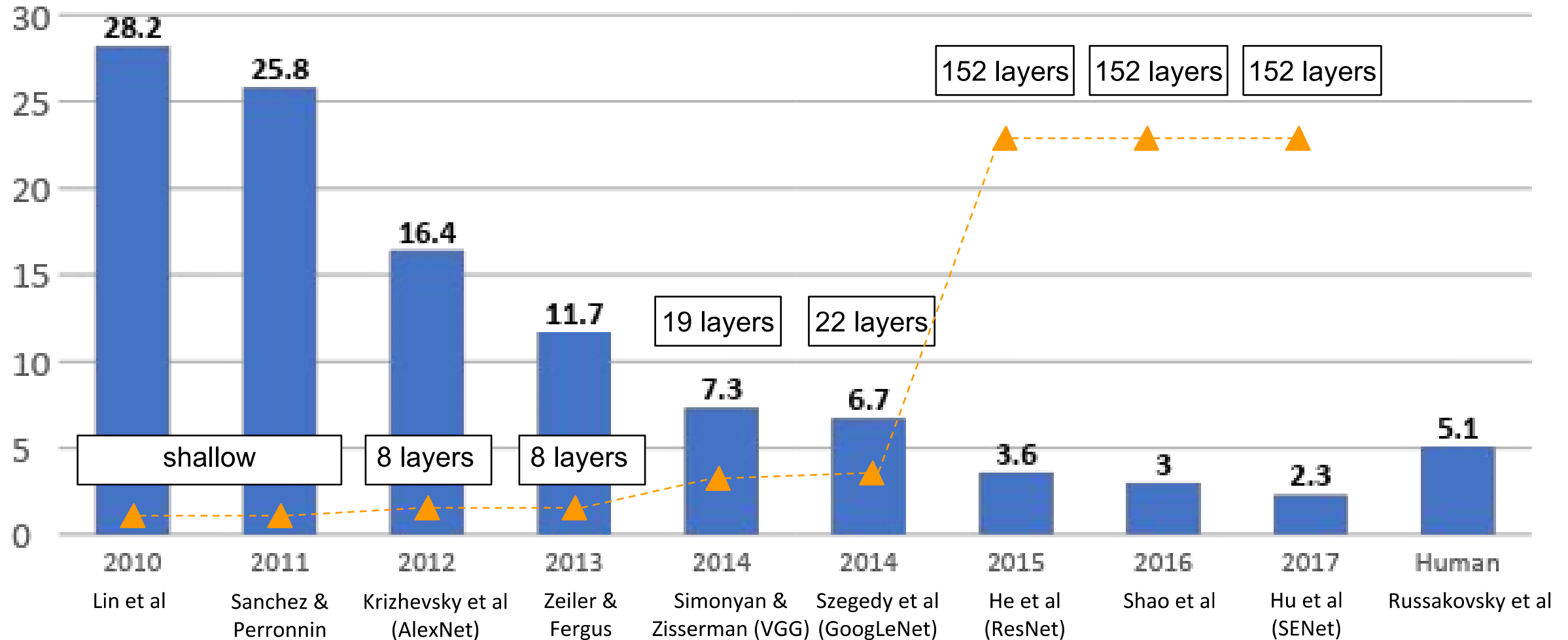
Dropout in ConvNet



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

CNN Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

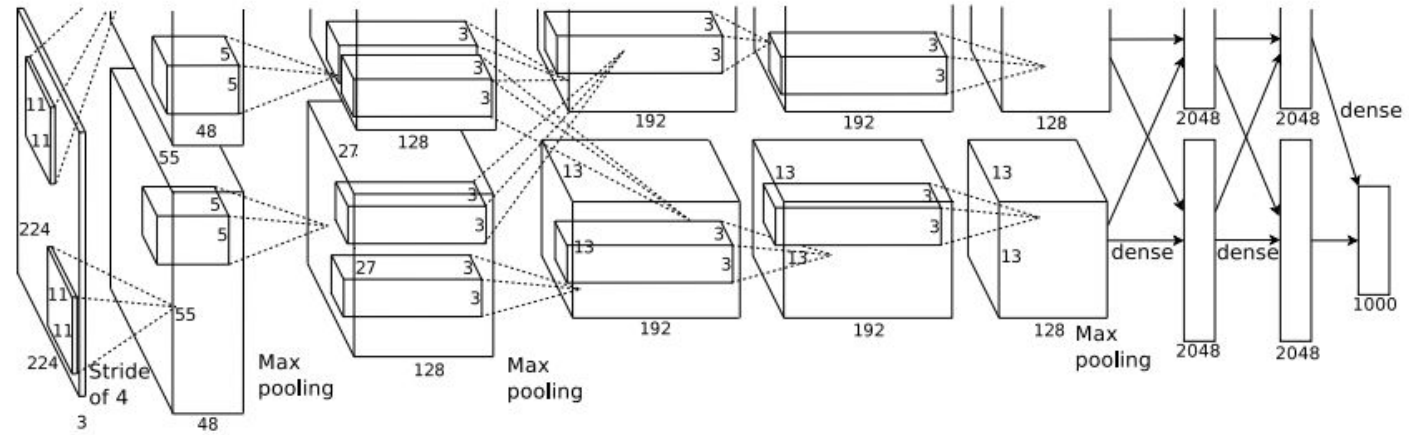
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

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

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

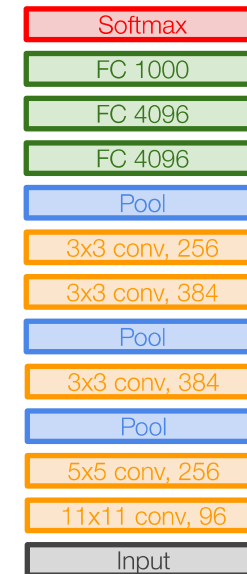
8 layers (AlexNet)

-> 16 - 19 layers (VGG16Net)

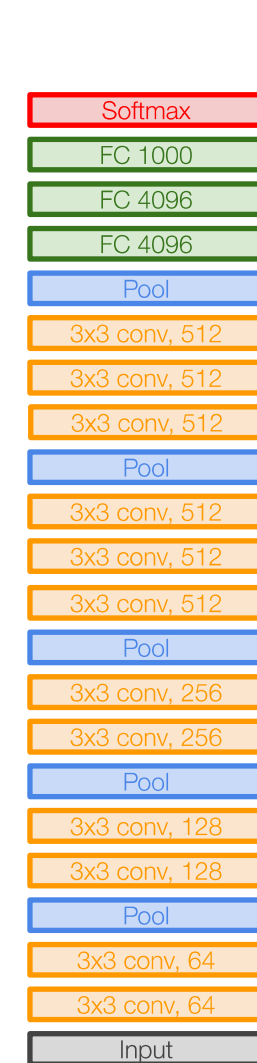
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)

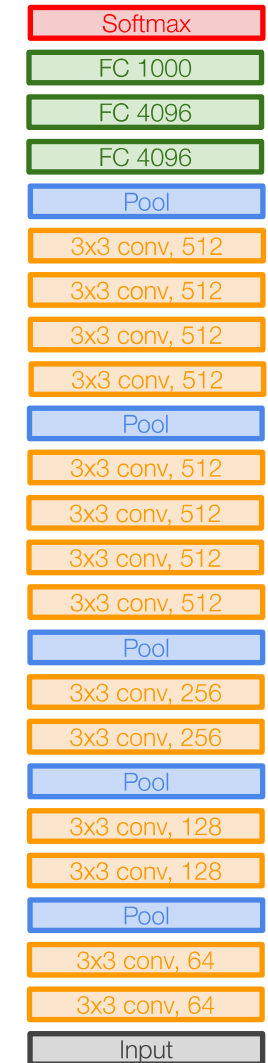
-> 7.3% top 5 error in ILSVRC'14



AlexNet



VGG16



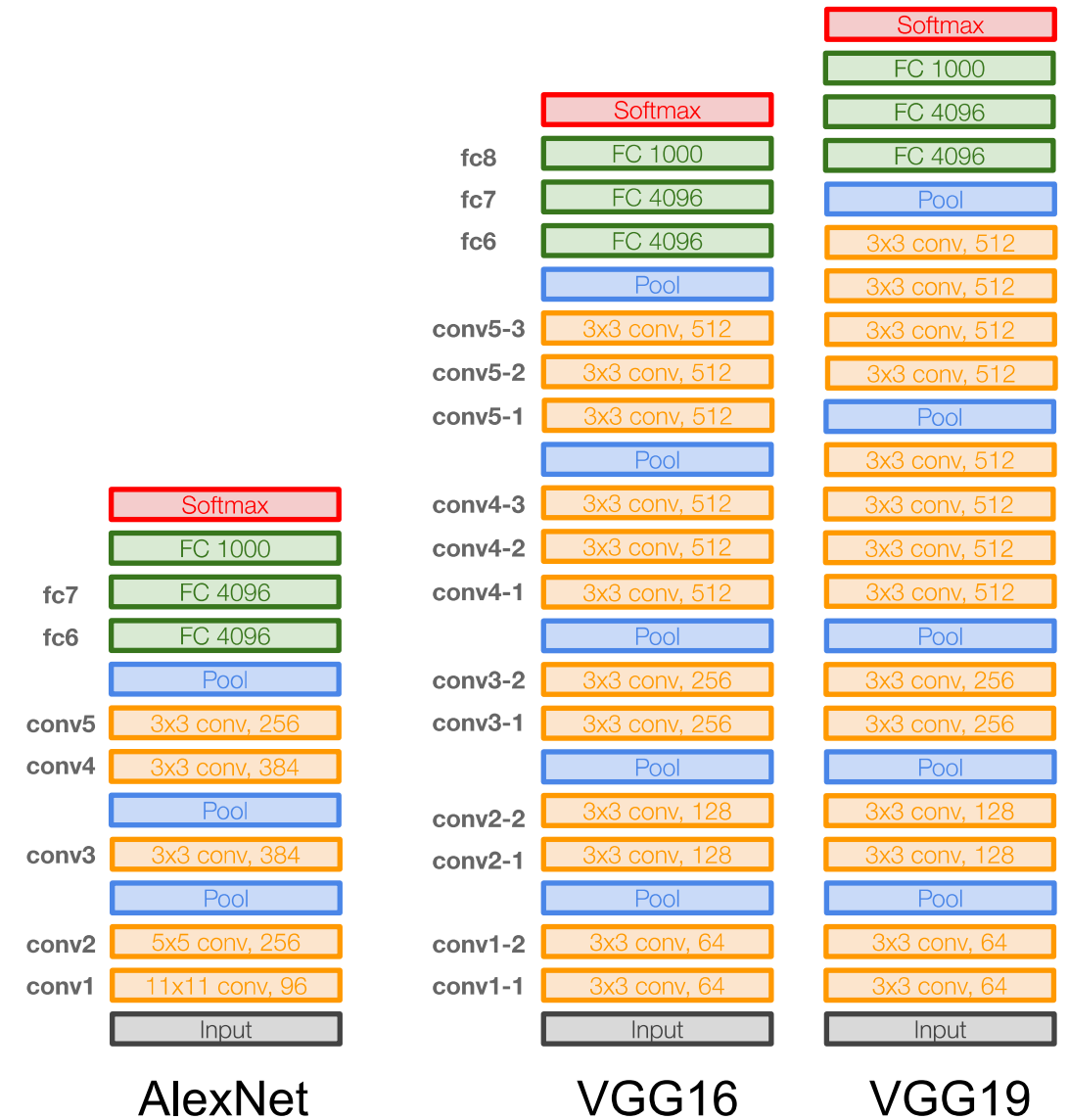
VGG19

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

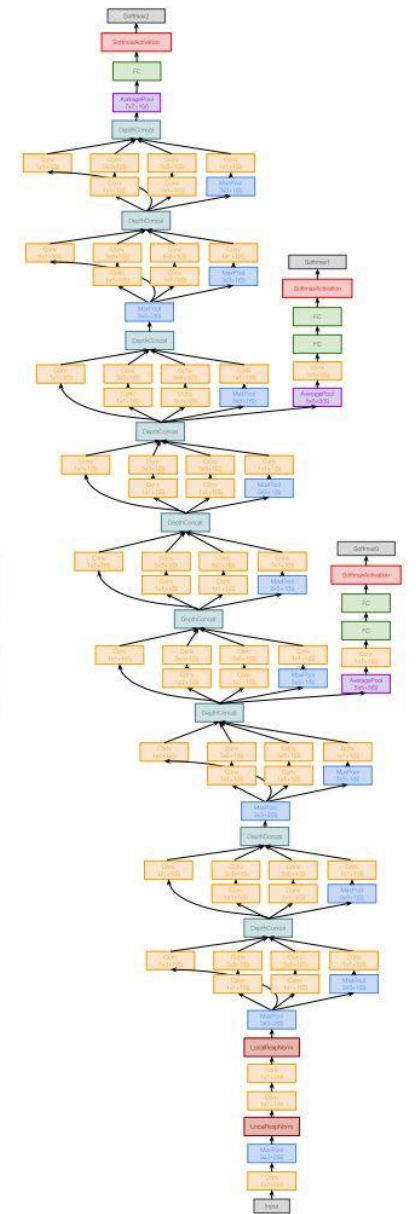
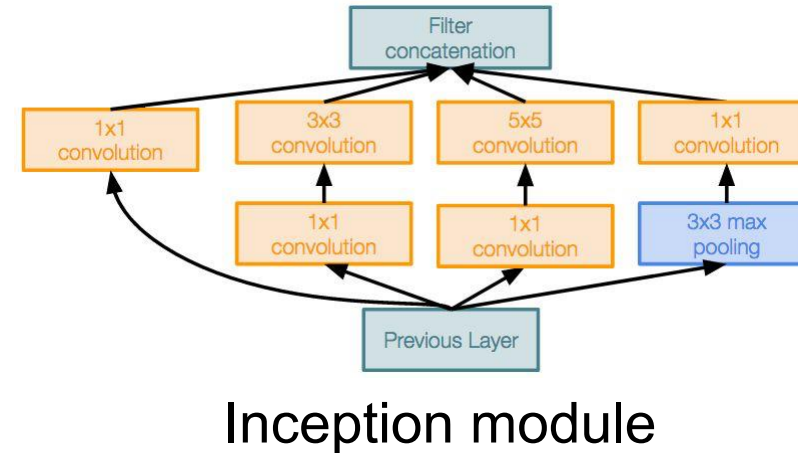


Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

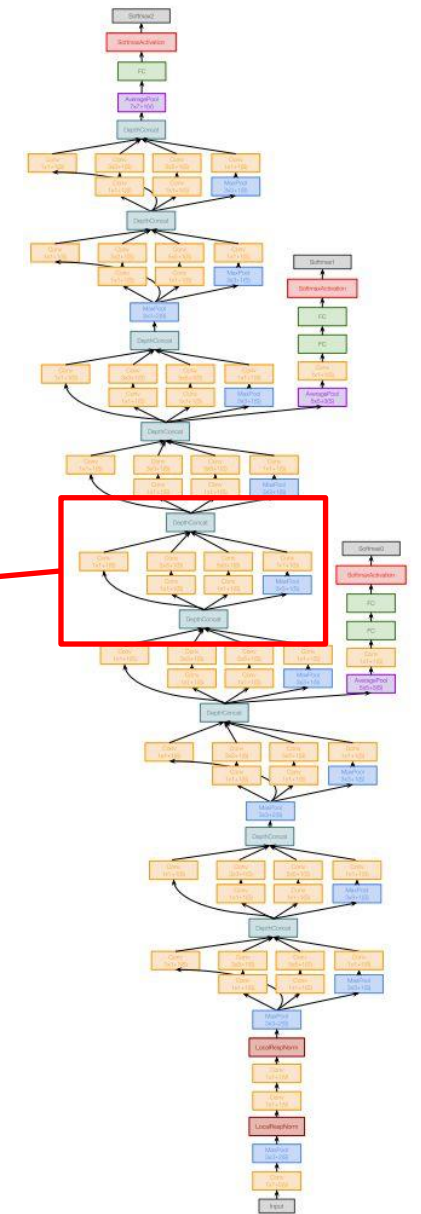
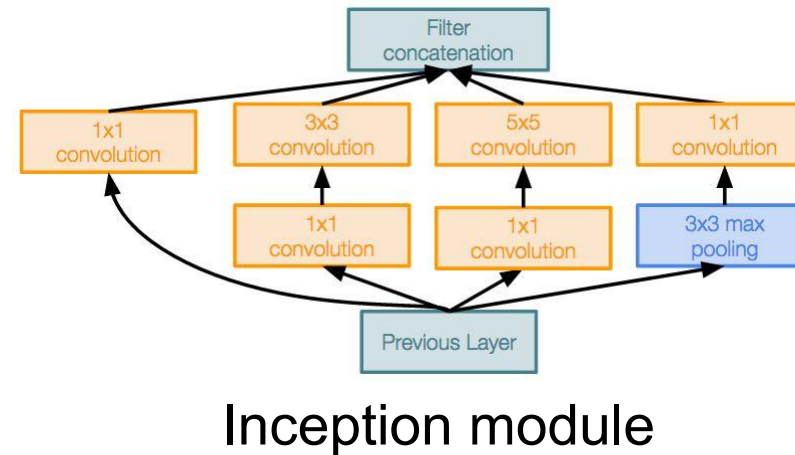
- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
12x less than AlexNet
27x less than VGG-16
- Efficient “Inception” module
- No FC layers



Case Study: GoogLeNet

[Szegedy et al., 2014]

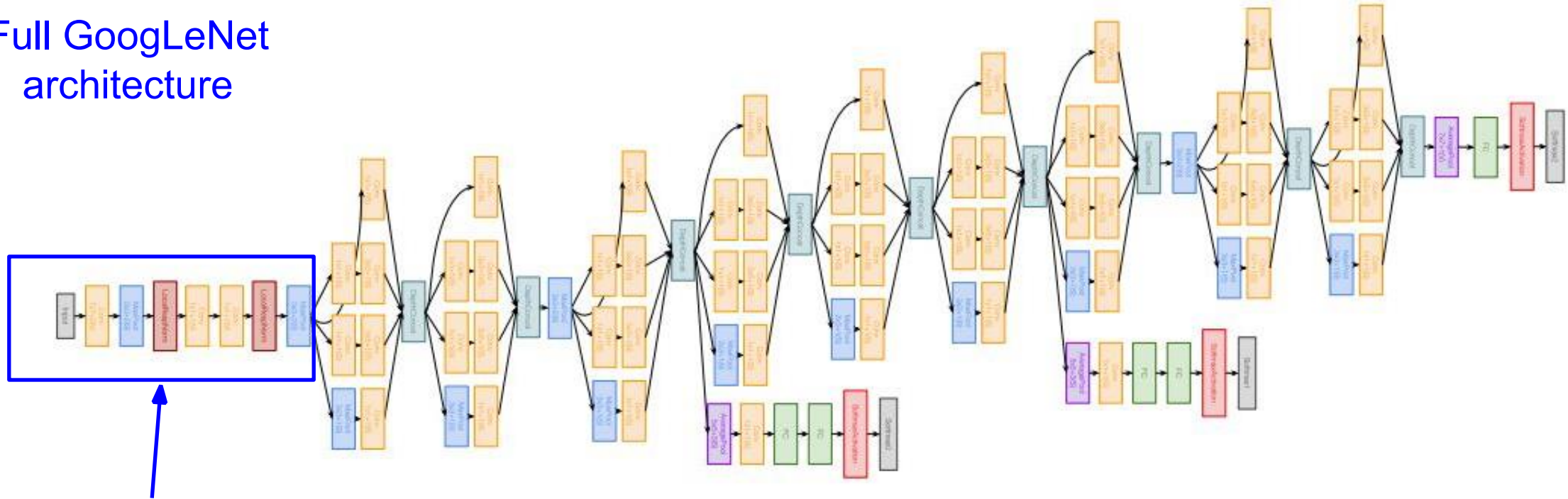
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

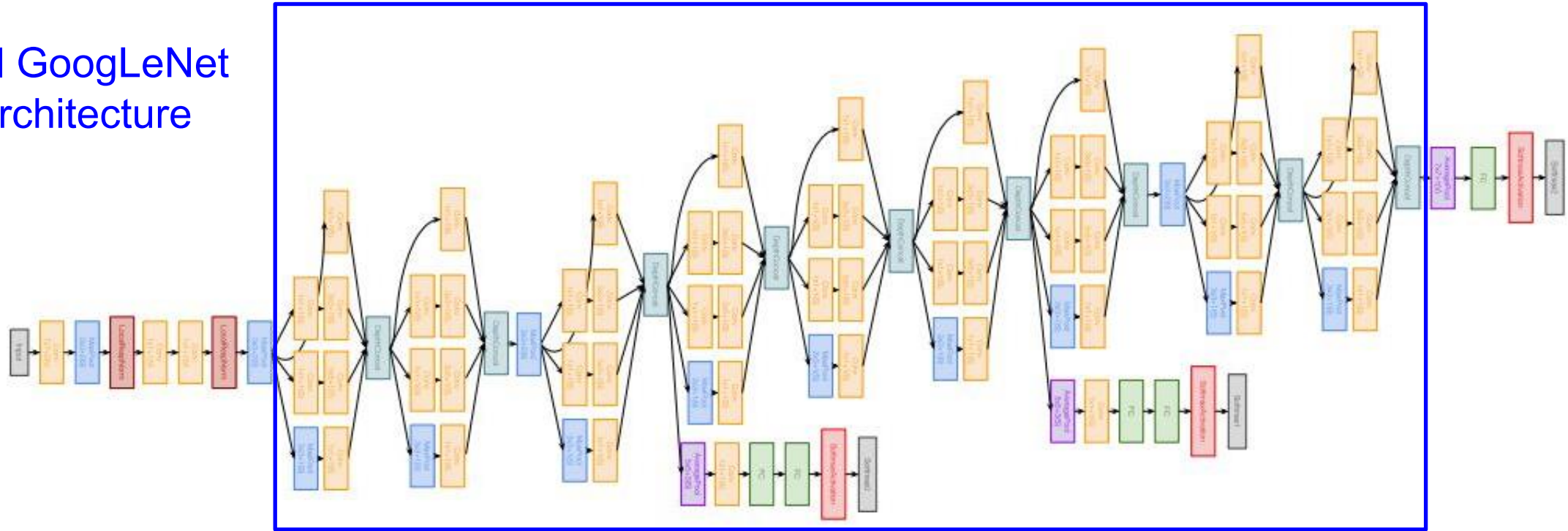


Stem Network:
Conv-Pool-
2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

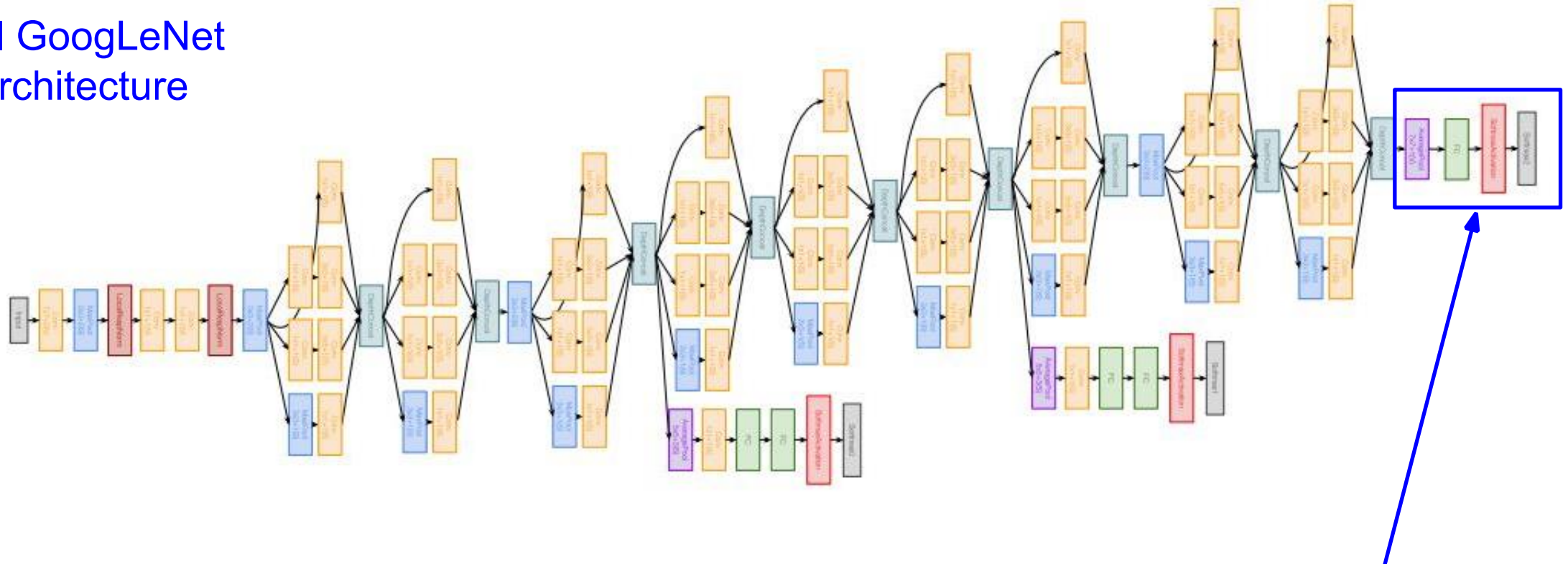


Stacked Inception
Modules

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

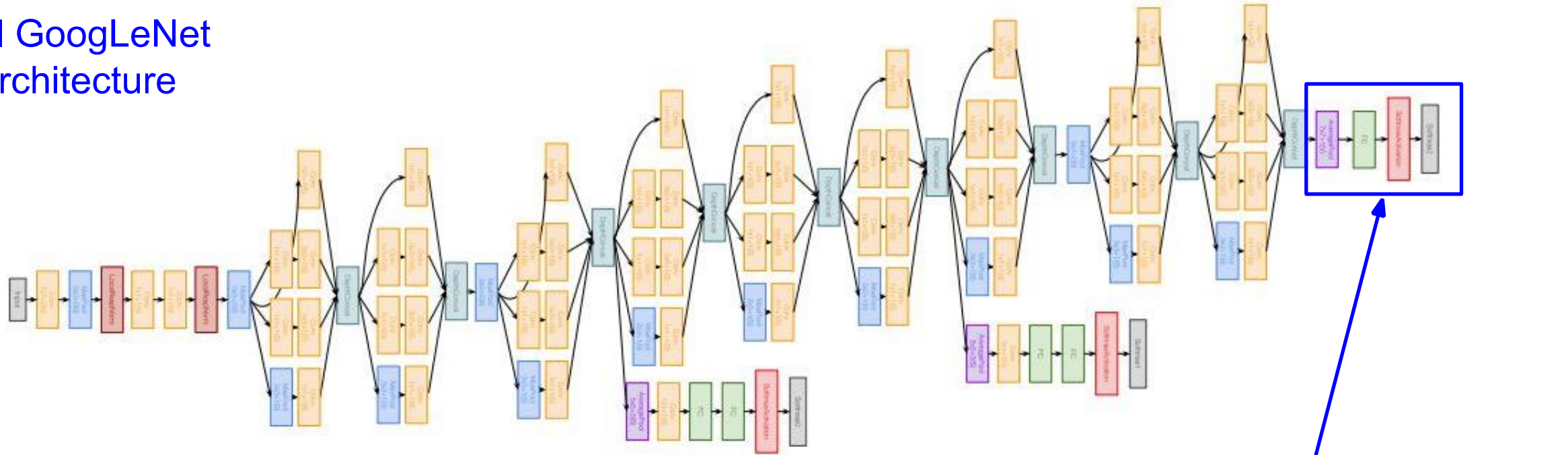


Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



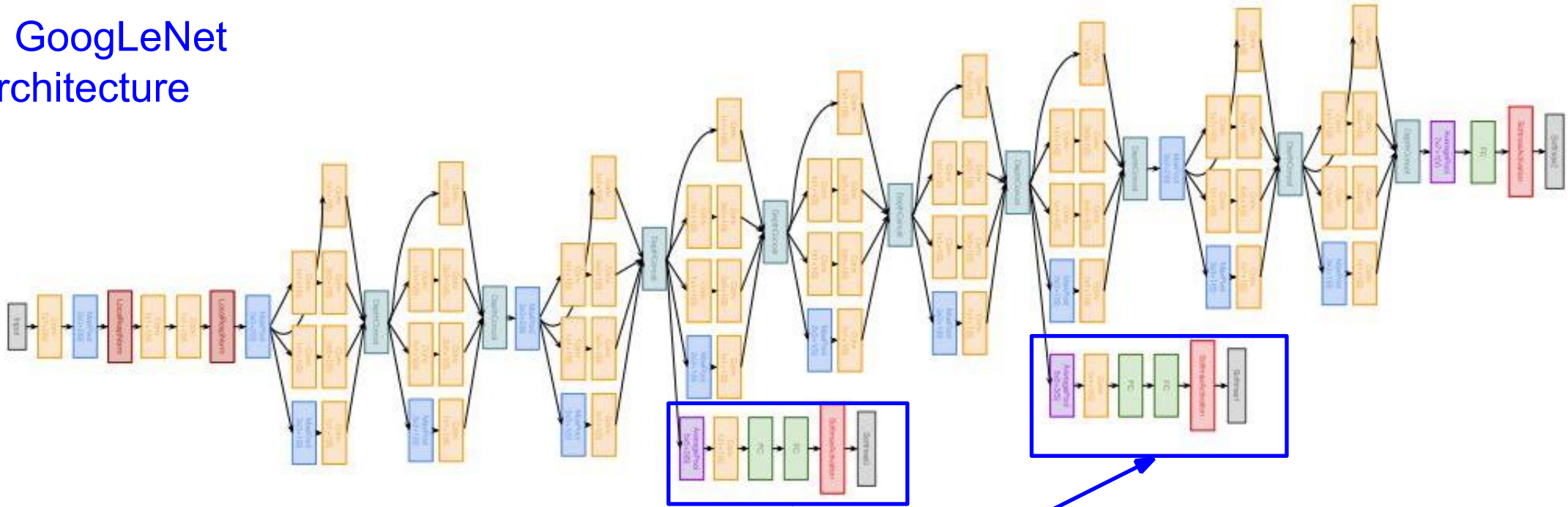
Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

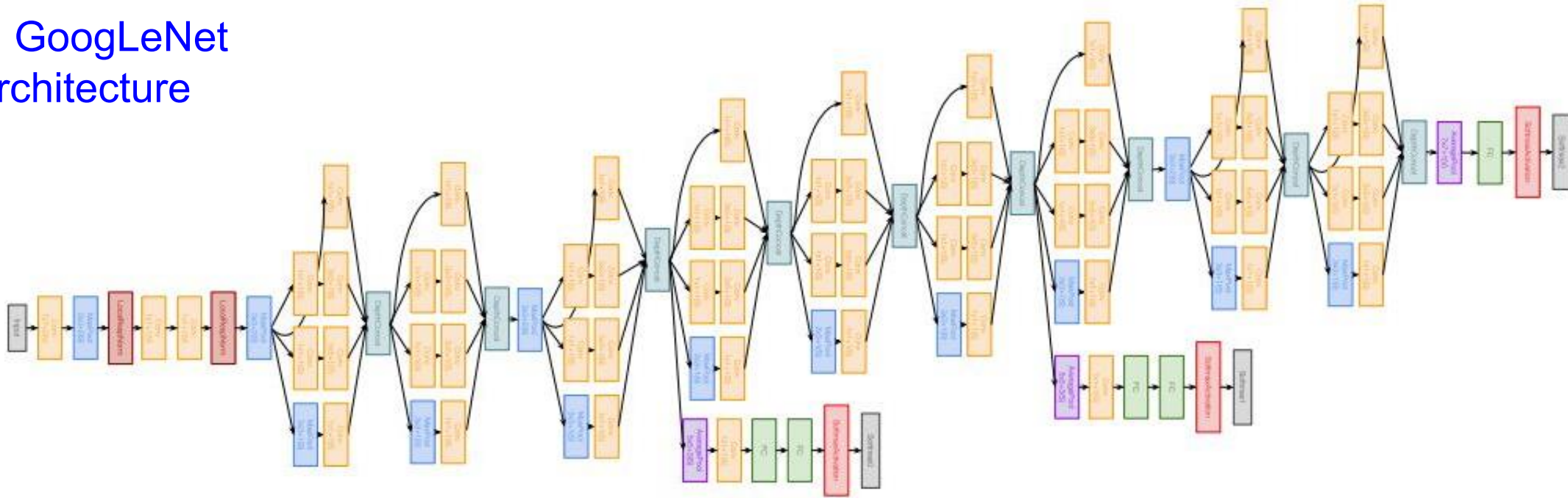


Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



22 total layers with weights

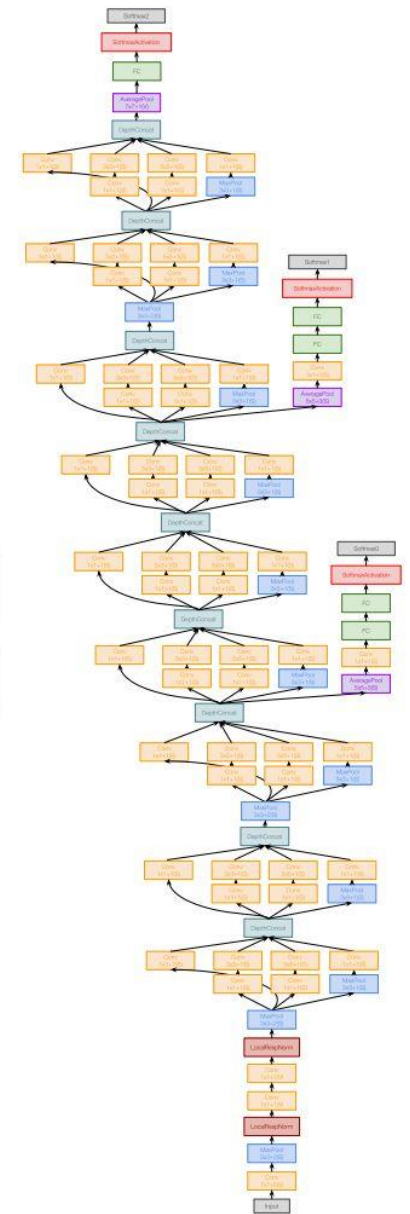
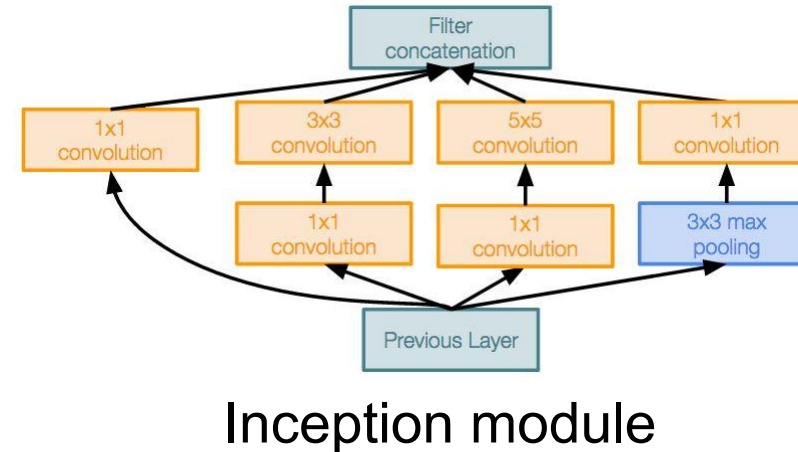
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)

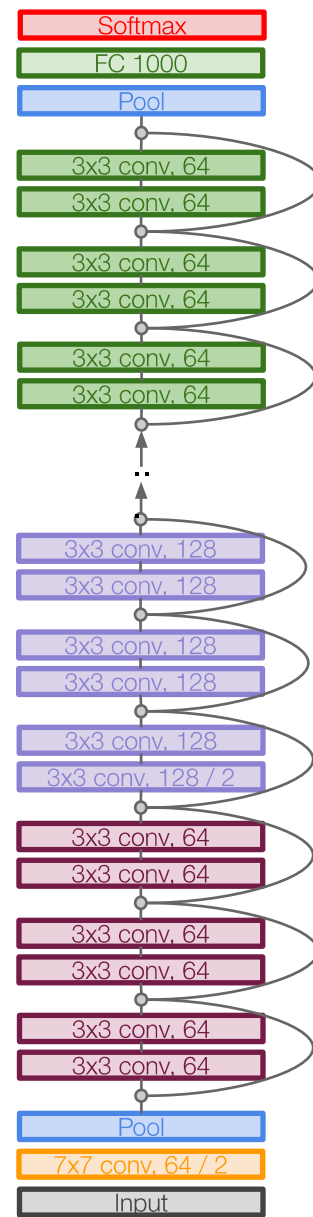
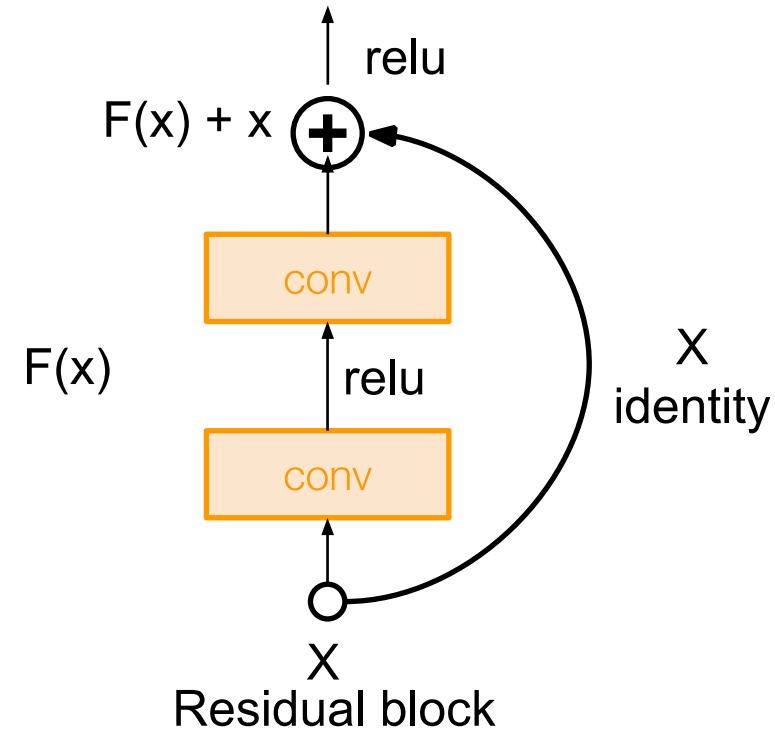


Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

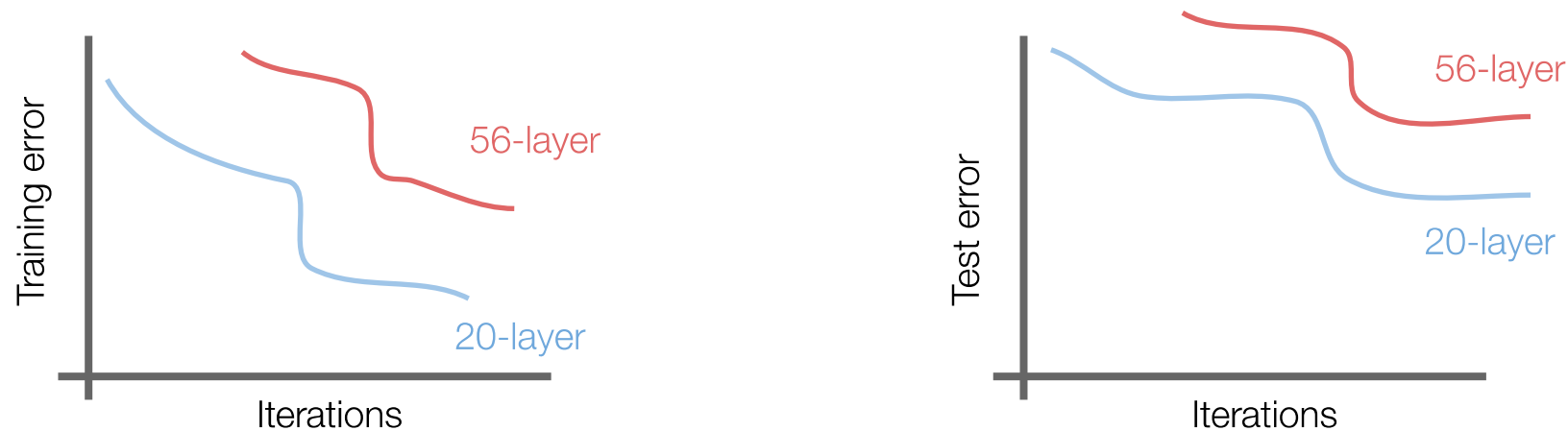
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



Case Study: ResNet

[He et al., 2015]

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



56-layer model performs worse on both training and test error

-> The deeper model performs worse, but it's **not caused by overfitting!**

Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem,
deeper models are harder to optimize

Case Study: ResNet

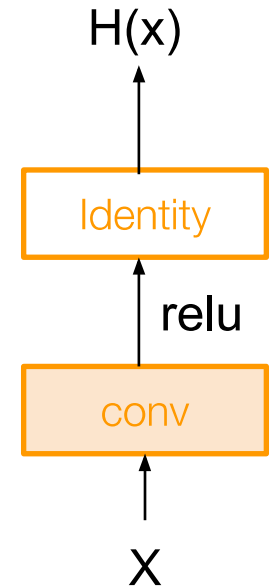
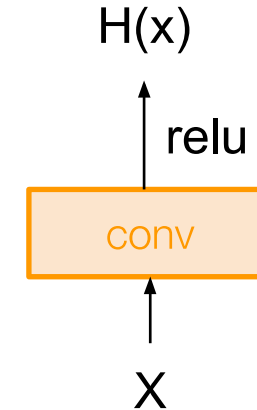
[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

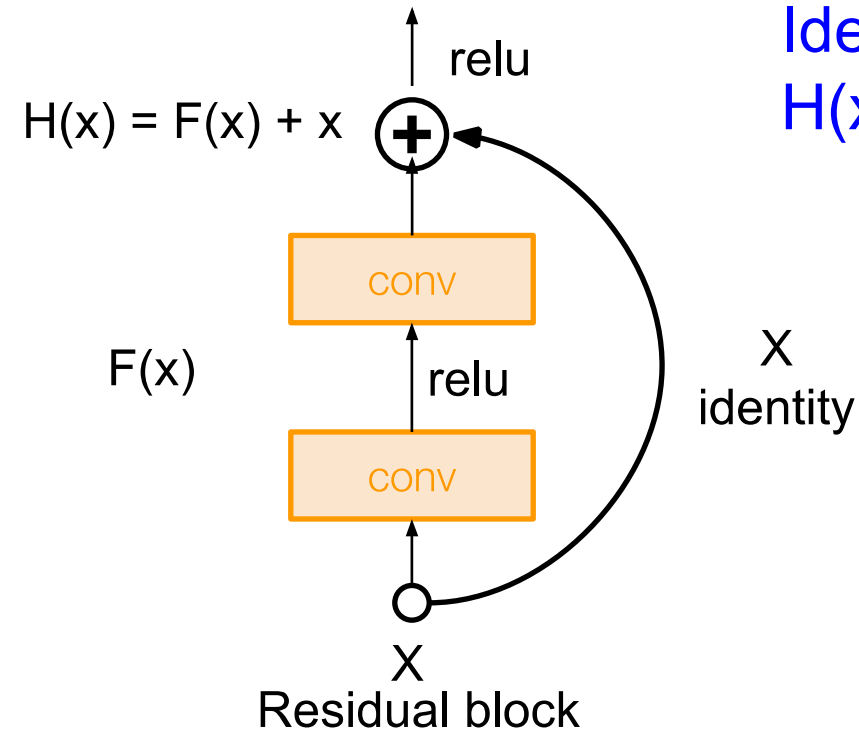
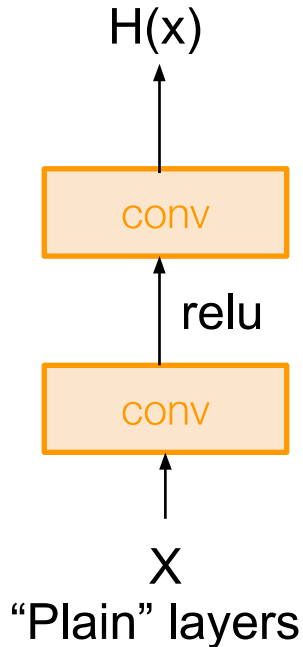
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



Case Study: ResNet

[He et al., 2015]

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

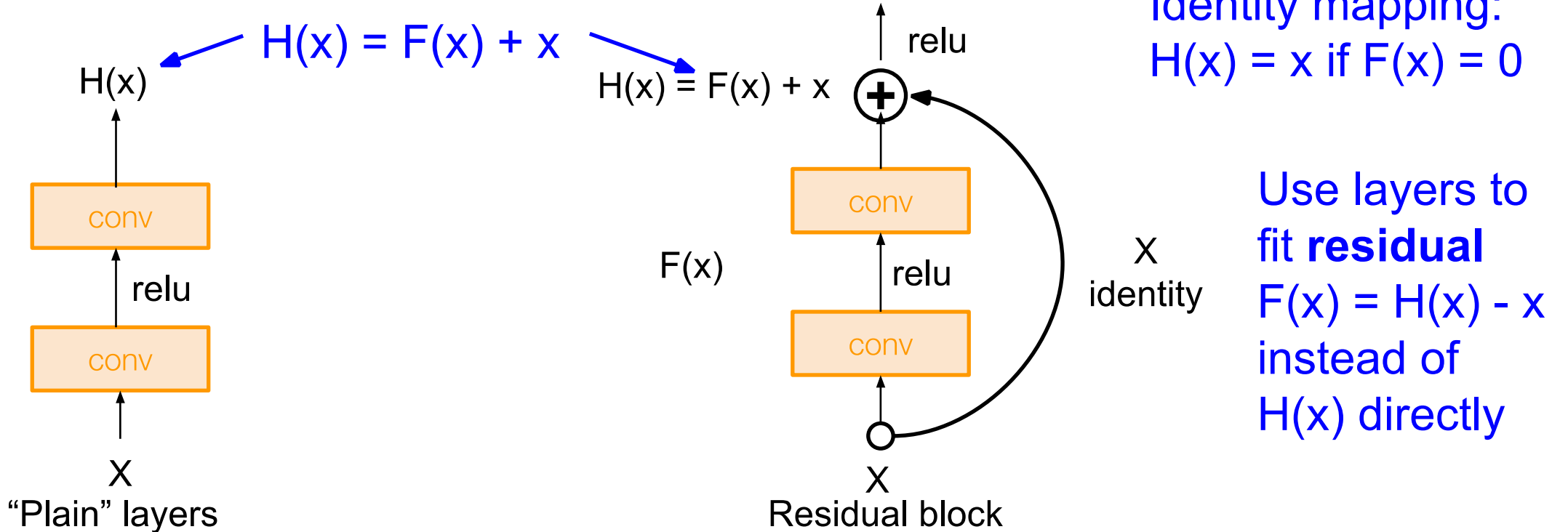


Identity mapping:
 $H(x) = x$ if $F(x) = 0$

Case Study: ResNet

[He et al., 2015]

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

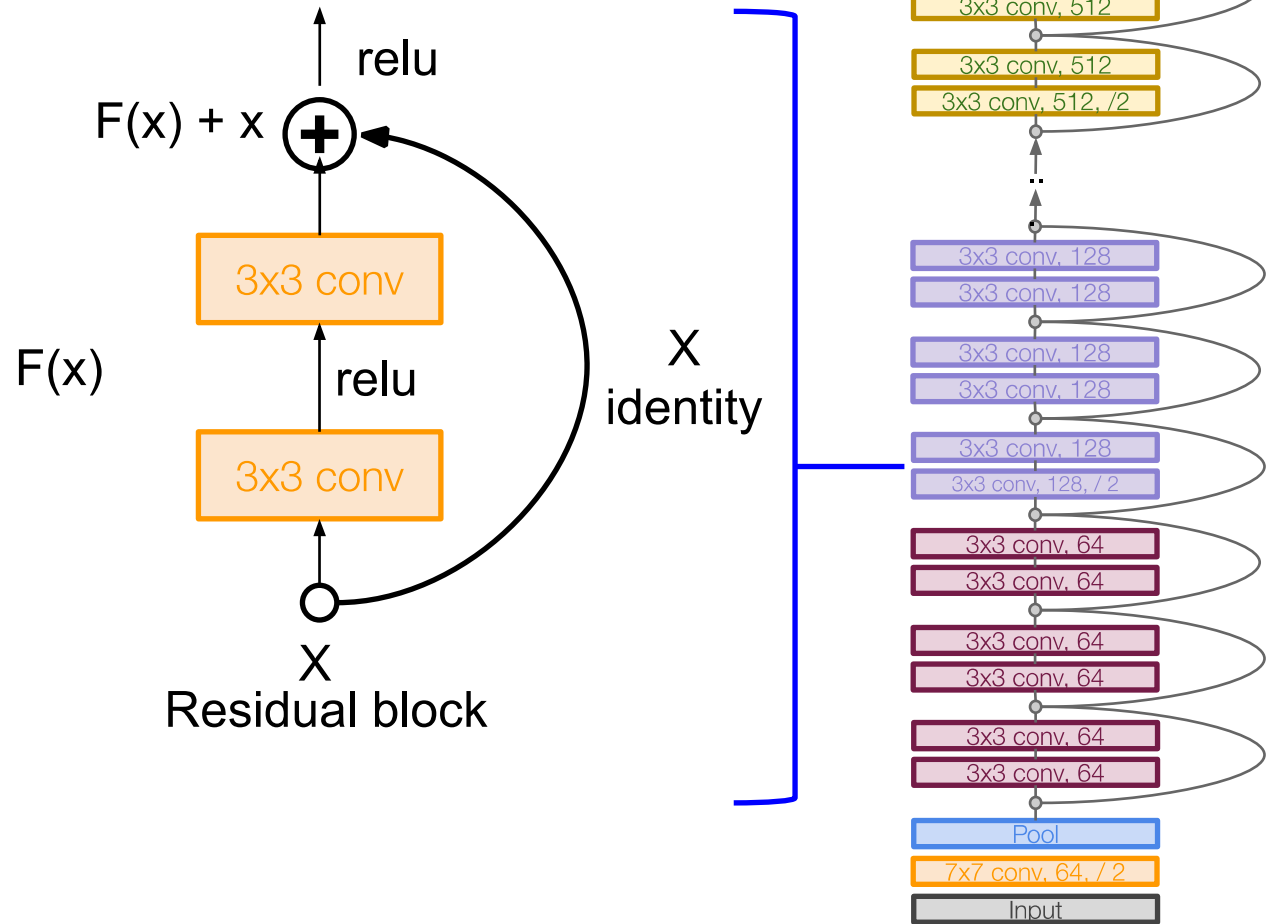


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

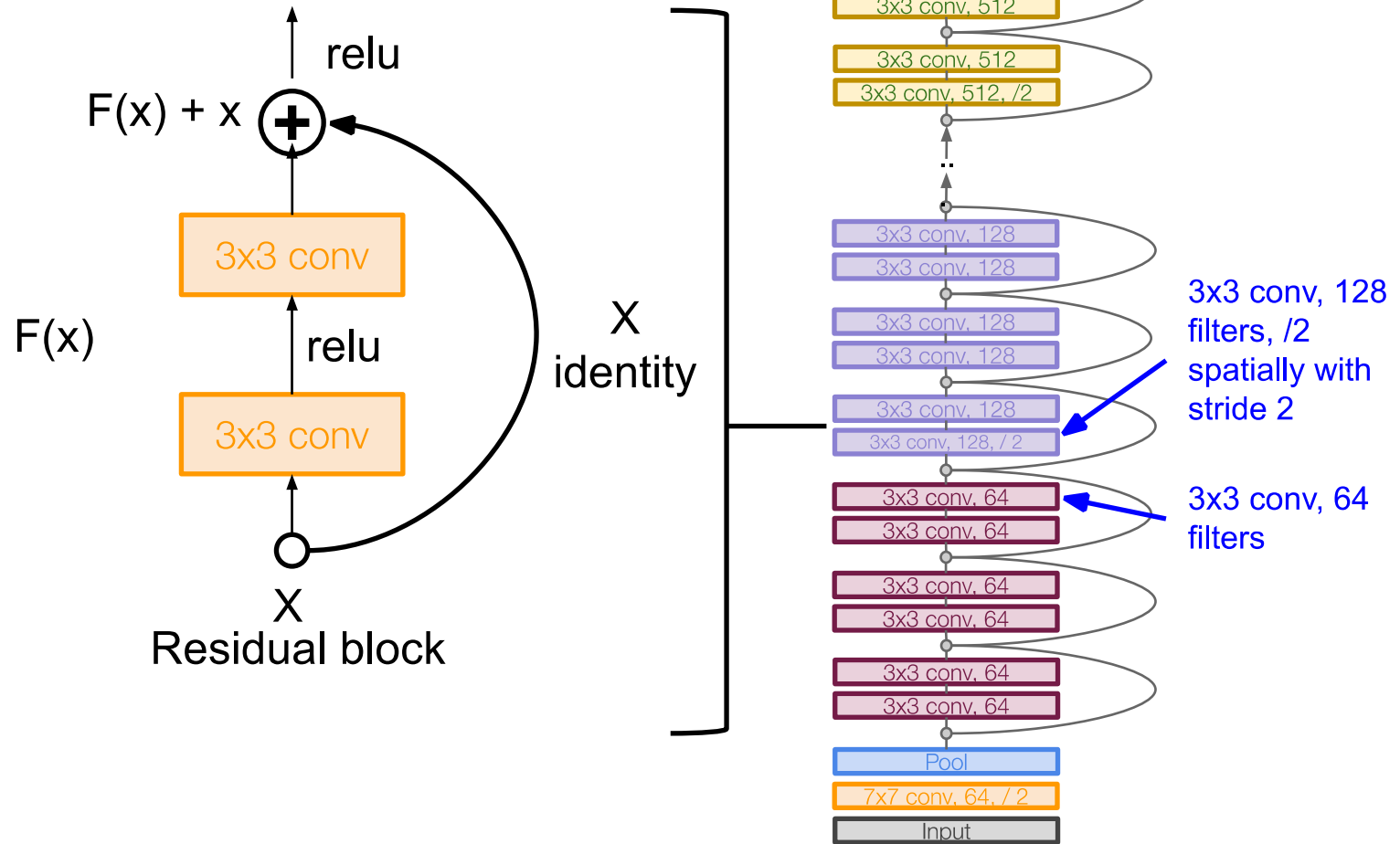


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
 - Every residual block has two 3x3 conv layers
 - Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Reduce the activation volume by half.

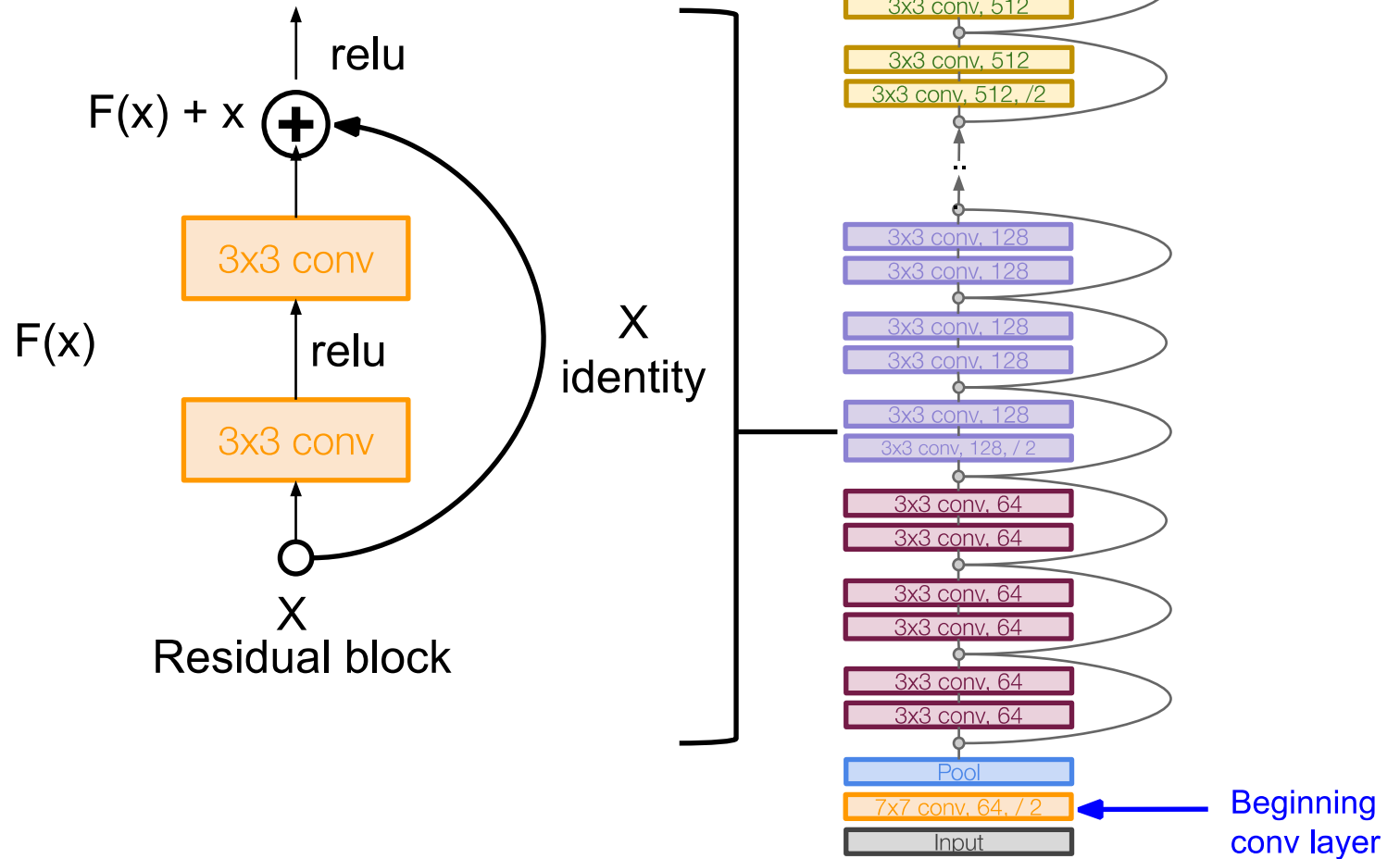


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)

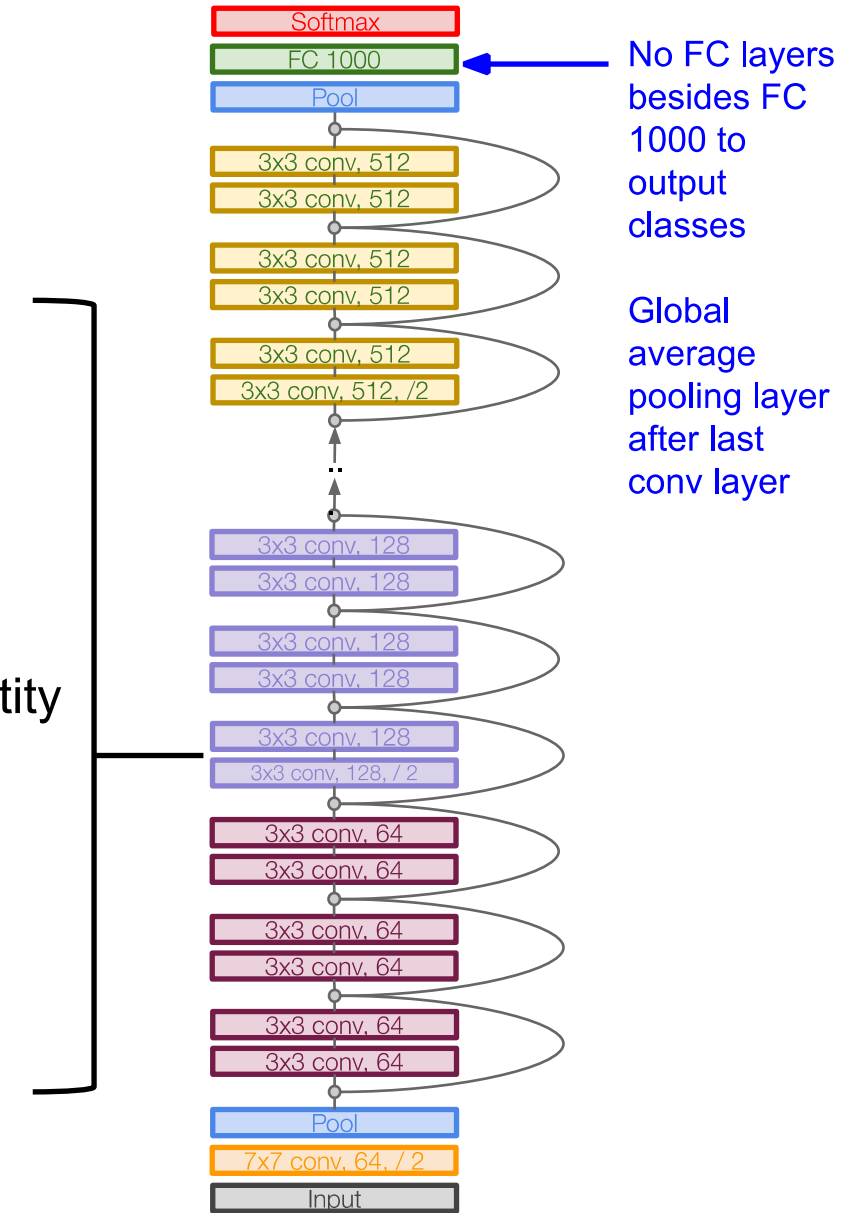
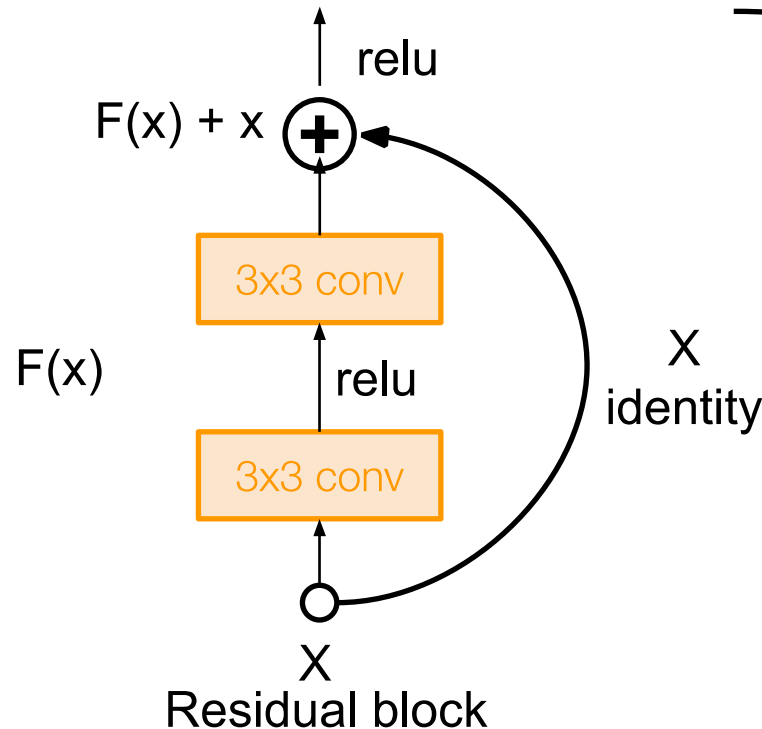


Case Study: ResNet

[He et al., 2015]

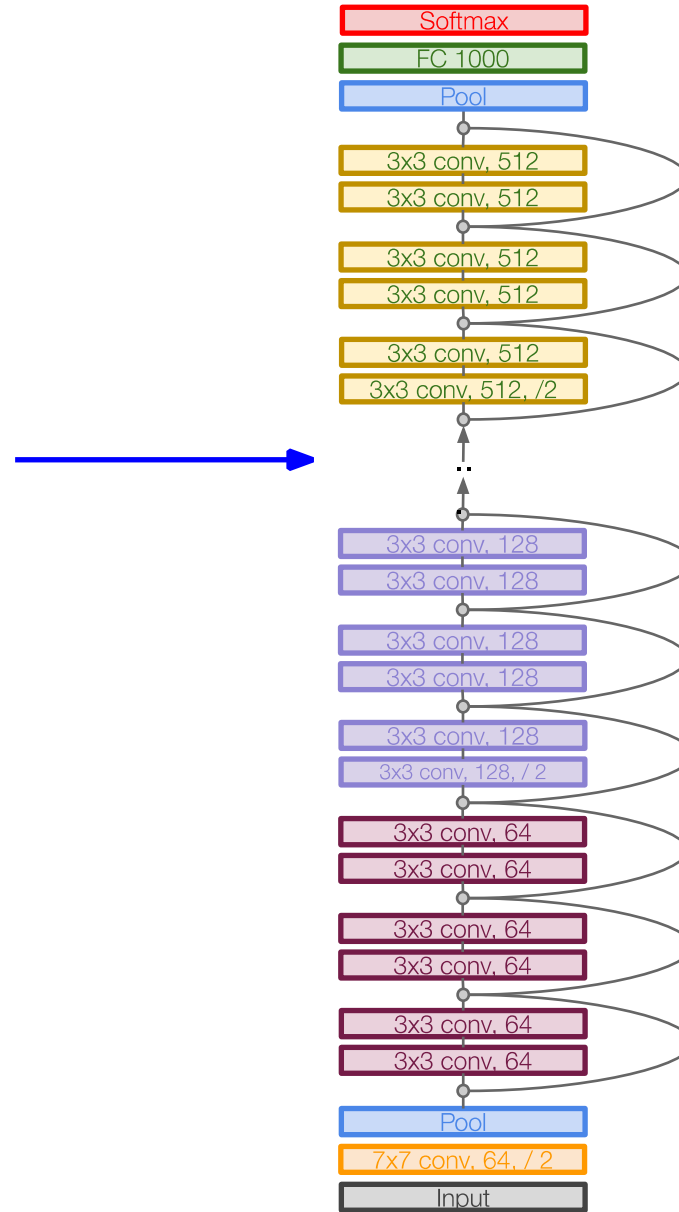
Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)



[He et al., 2015]

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



Case Study: ResNet

[He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of $1e-5$
- No dropout used

Case Study: ResNet

[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

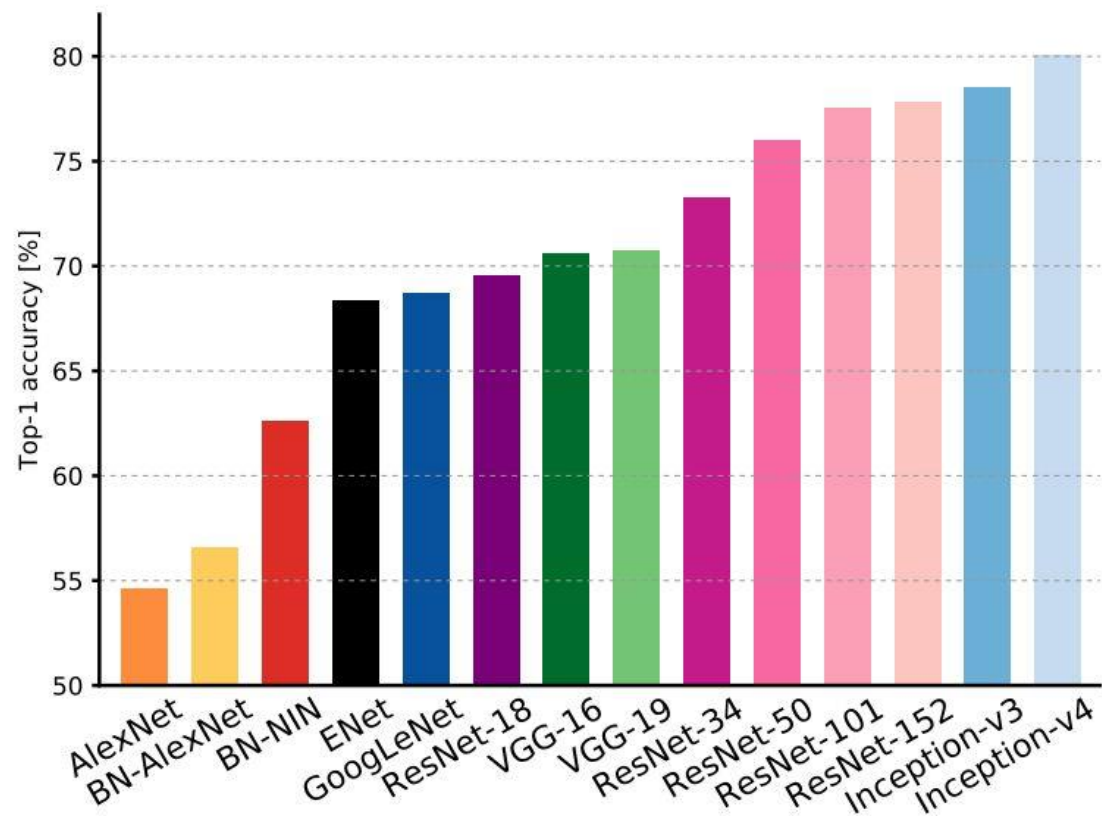
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**

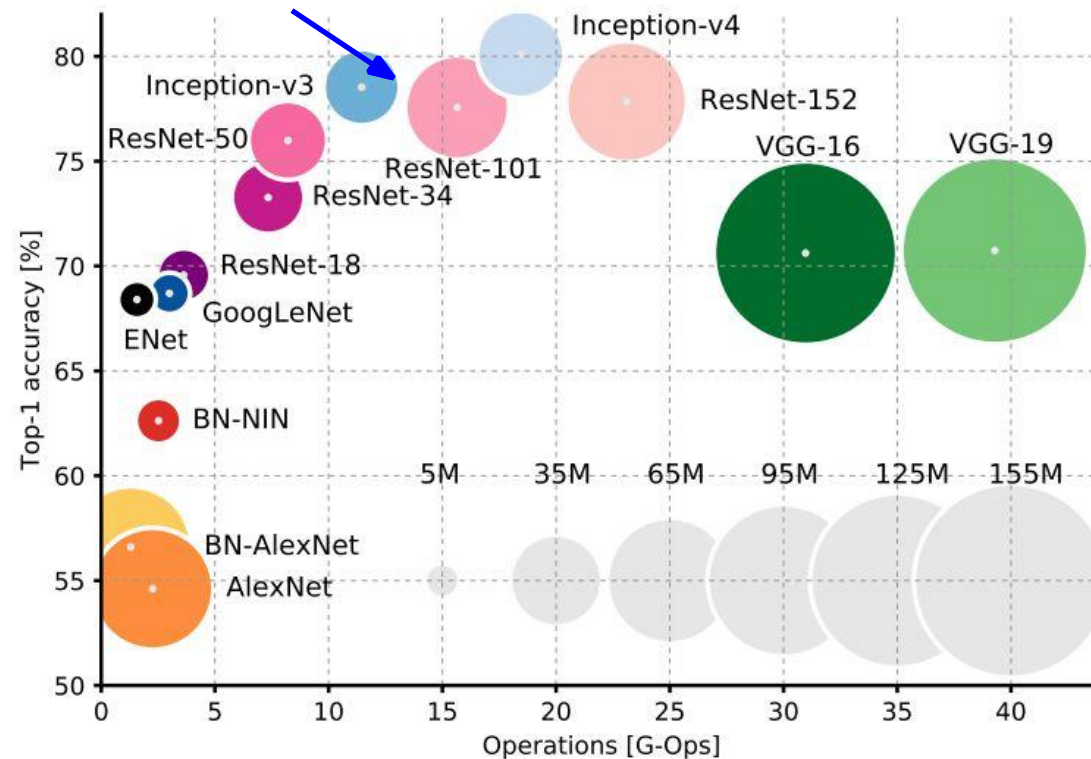
- ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)

Comparing complexity...



ResNet:
Moderate efficiency depending on
model, highest accuracy



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

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

AI504: Programming for Artificial Intelligence

Week 7: Convolutional Neural Network

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr