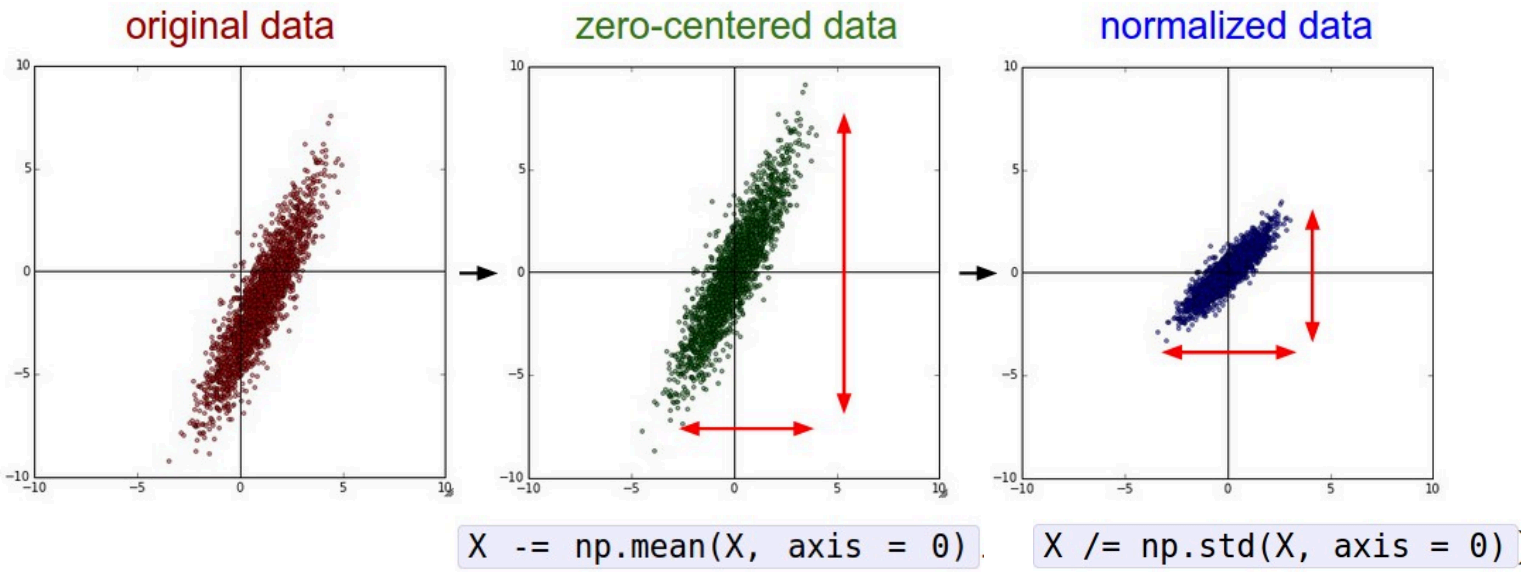
Intro to Deep Learning

Lecture 9: Convolutional Neural Networks

Data Preprocessing

Data Preprocessing



(Assume X [NxD] is data matrix,
each example in a row)

Batch Normalization

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want zero-mean unit-variance hidden features? just make them so."

consider a batch of hidden features at some layer. To make each dimension zero-mean unit-variance, apply:

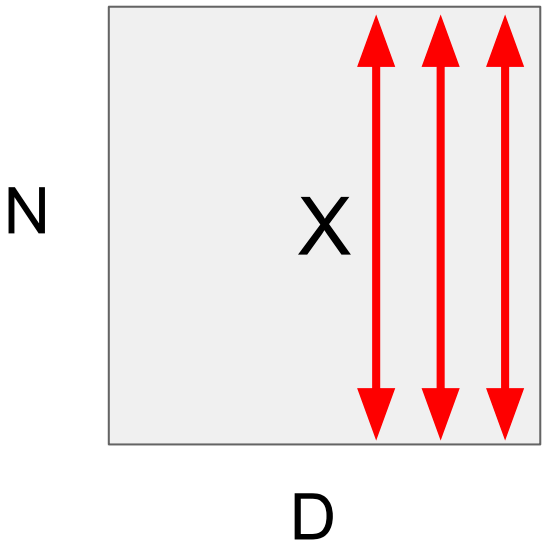
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want zero-mean unit-variance hidden features? just make them so."



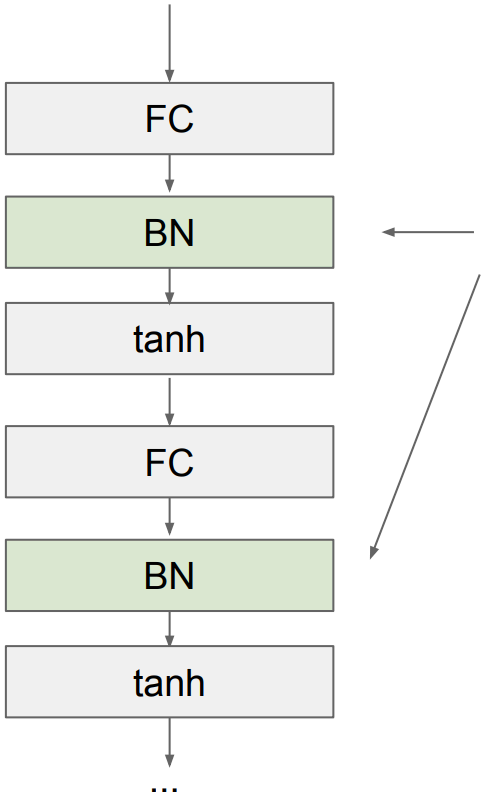
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

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

Batch Normalization

[Ioffe and Szegedy, 2015]



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

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

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

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

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

A linear transformation on $\hat{x}$ with learnable parameters γ, β

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

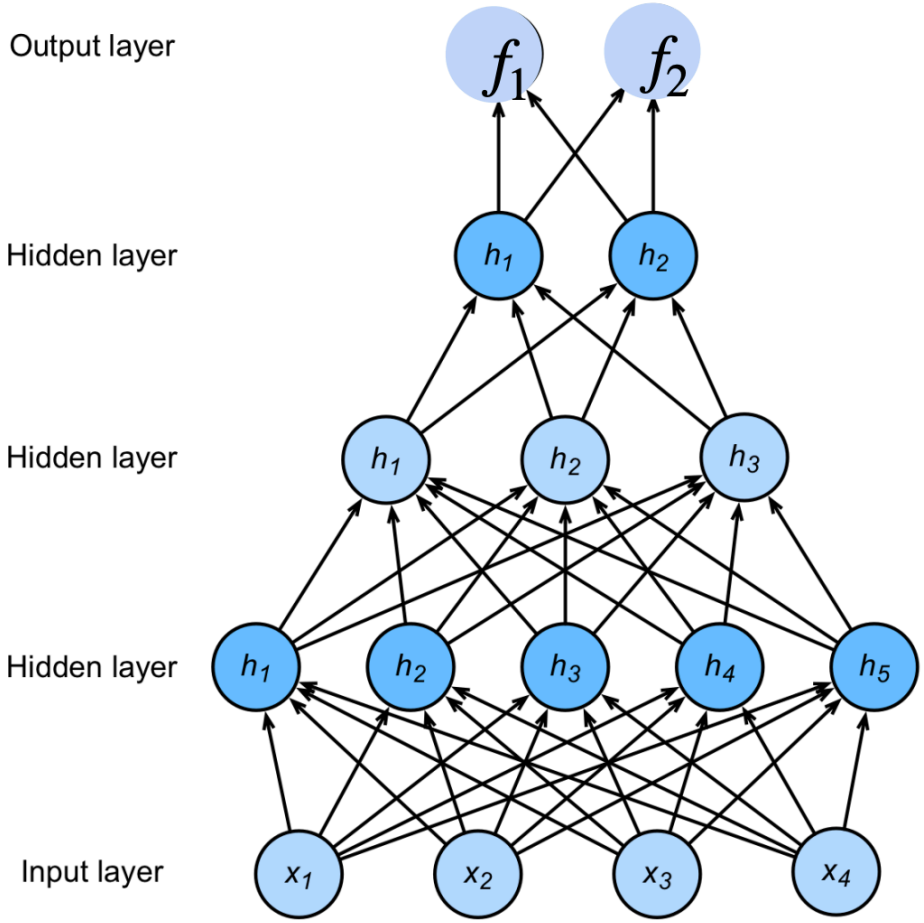
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

At inference time, batch statistics (save running avg. mean/std) are used

Review: Deep neural networks (DNNs)

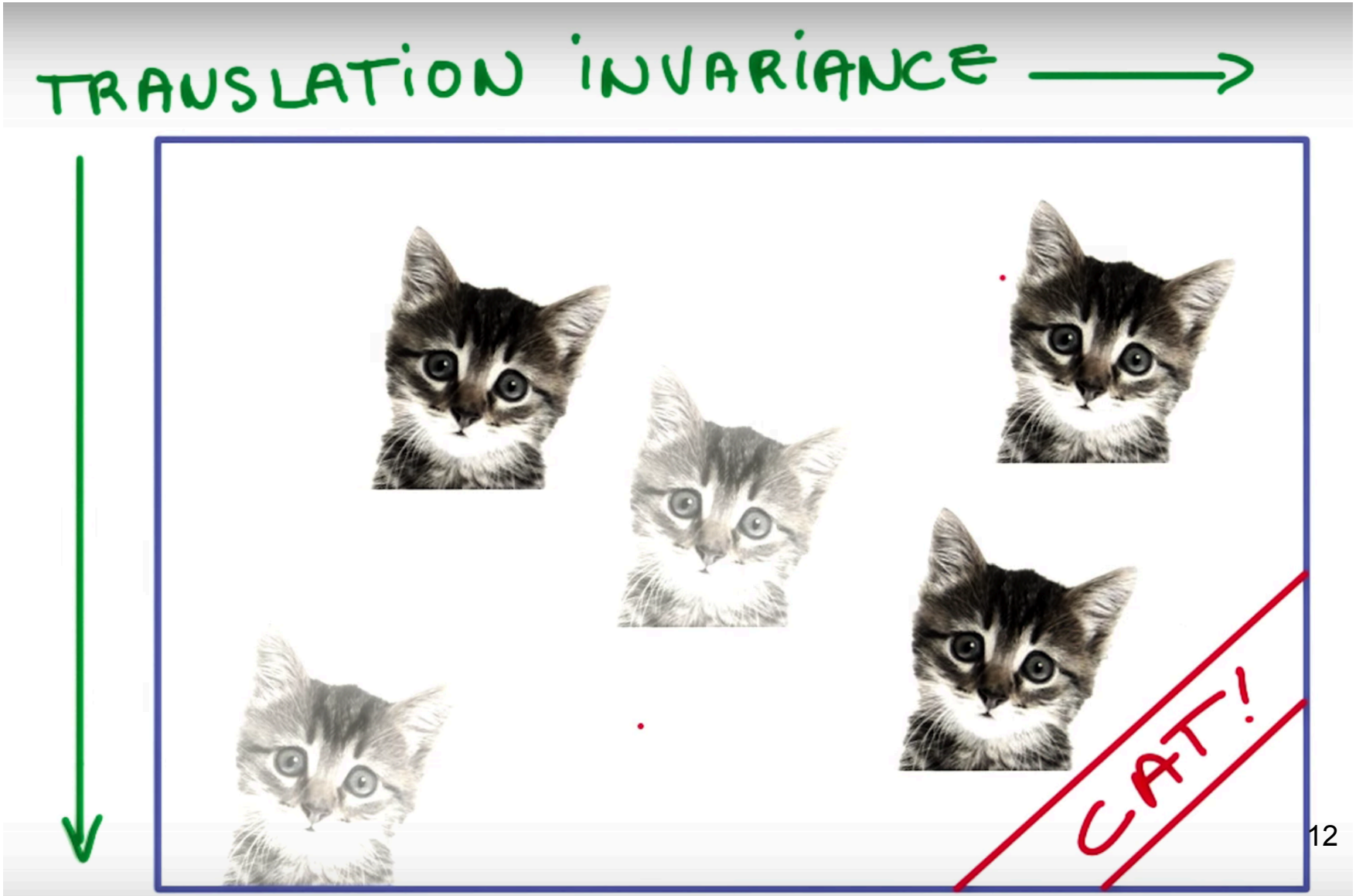


$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

What are the problems with the standard DNNs (MLPs)?

Drawbacks of MLPs



Drawbacks of MLPs

How to classify

Cats vs. dogs?

Dual

12MP

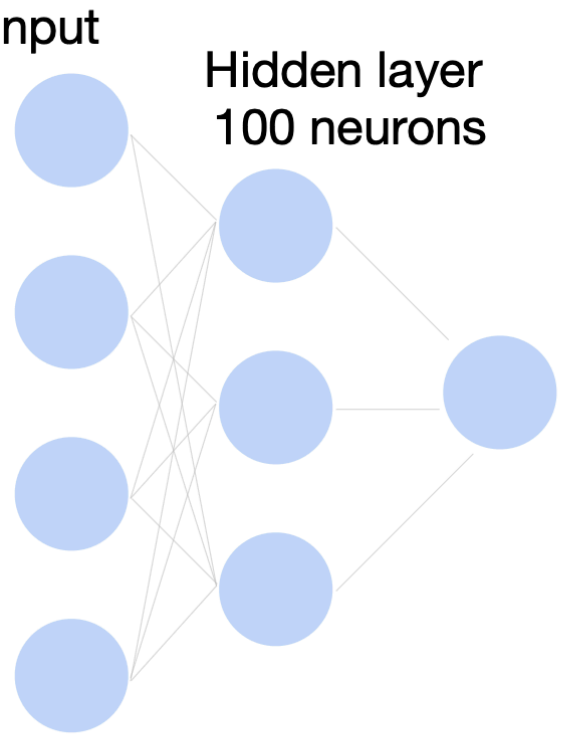
wide-angle and
telephoto cameras

36M floats in a RGB image!

Drawbacks of MLPs

Fully Connected Networks

Cats vs. dogs?



~ 36M elements x 100 = ~**3.6B** parameters!

More than half the size of LLaMA 7B

Convolutions Neural Networks come to rescue!

Outline

- Intro to convolution neural networks
  - Definitions
  - 2D convolution
  - Padding and stride
  - Multiple input and output channels
  - Poolings

Convolution

- Let $f, g : \mathbb{R}^d \to \mathbb{R}$ be two functions
- The convolution of f, g is defined as

$$[f * g](z) = \int_{\mathbb{R}^d} f(u)g(z-u)\,du$$

Convolution

- Suppose f is defined on a large grid $\{0,1,\ldots,L\}$ and denote $u_i = f(i)$.
- Suppose g is defined on a much larger grid and denote $w_i = g(i)$.

  $[f * g](z) = u_0 w_z + u_1 w_{z-1} + \ldots + u_L w_{z-L}$
- $$= \sum_{i=0}^{L} u_i w_{z-i}$$

"Flip" w; do elementwise multiplication and then a summation

Cross Correlation

- Let $f, g : \mathbb{R}^d \to \mathbb{R}$ be two functions
- The cross correlation of f, g is defined as

$$[f * g](z) = \int_{\mathbb{R}^d} f(u)g(z+u)\,du$$

Cross Correlation

- Suppose f is defined on a large grid $\{0,1,\ldots,L\}$ and denote $u_i = f(i)$.
- Suppose g is defined on a much larger grid and denote $w_i = g(i)$.

$$[f * g](z) = u_0 w_z + u_1 w_{z+1} + \ldots + u_L w_{z+L}$$

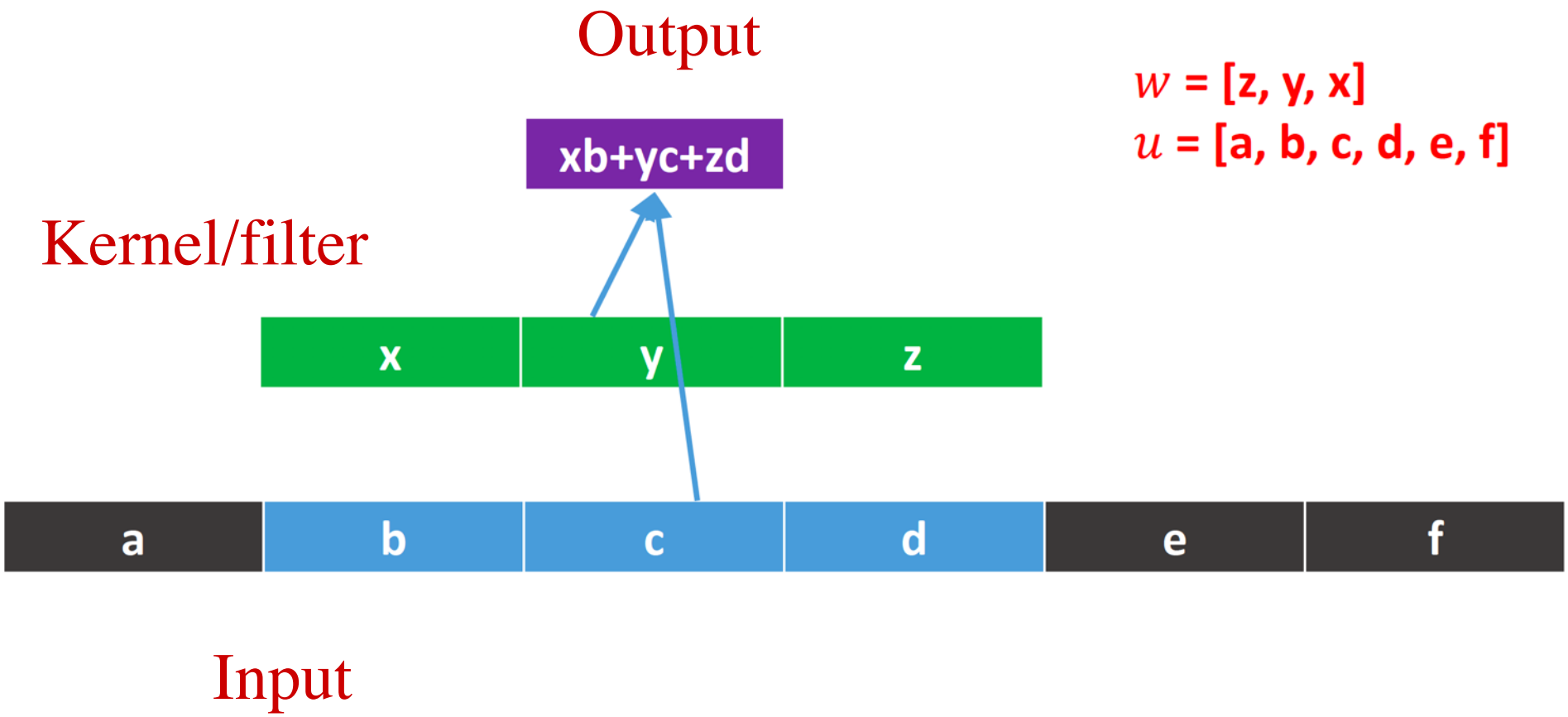
- $$= \sum_{i=0}^{L} u_i w_{z+i}$$

Do elementwise multiplication and then a summation

It’s really Cross-Correlation

Convolutions Neural Networks come to rescue!

Illustration 1



w = [z, y, x]

u = [a, b, c, d, e, f]

Illustration 1

Slide kernel to get another output

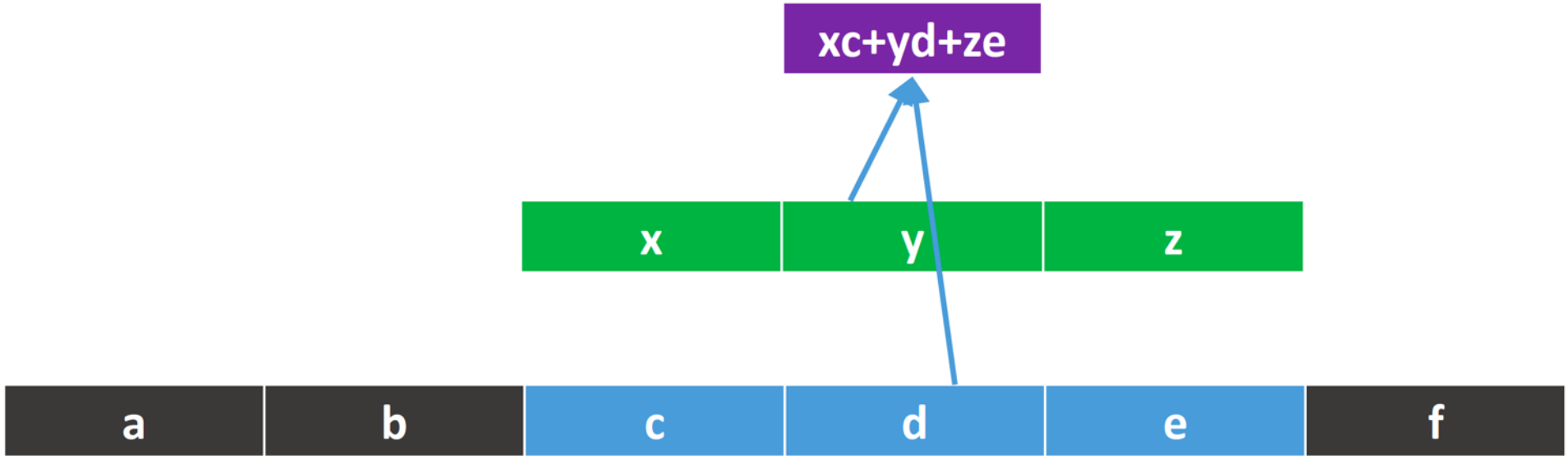


Illustration 1

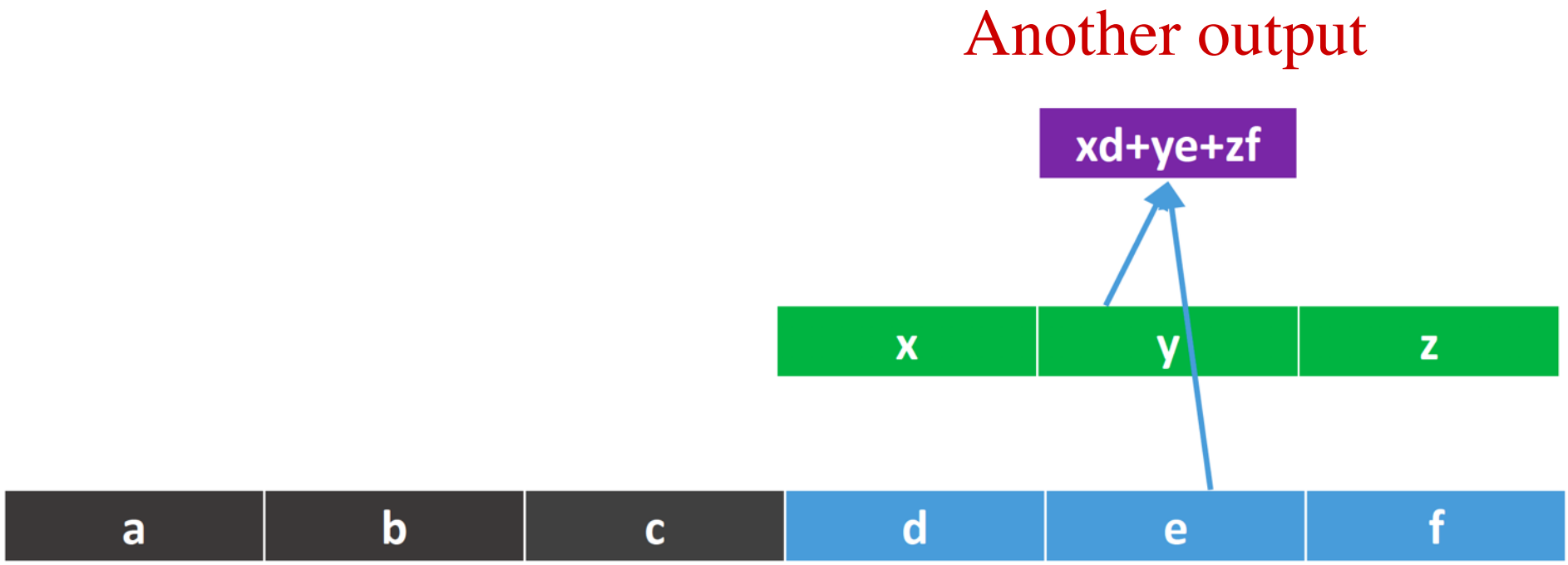


Illustration 1: boundary case

Ignore the overflow

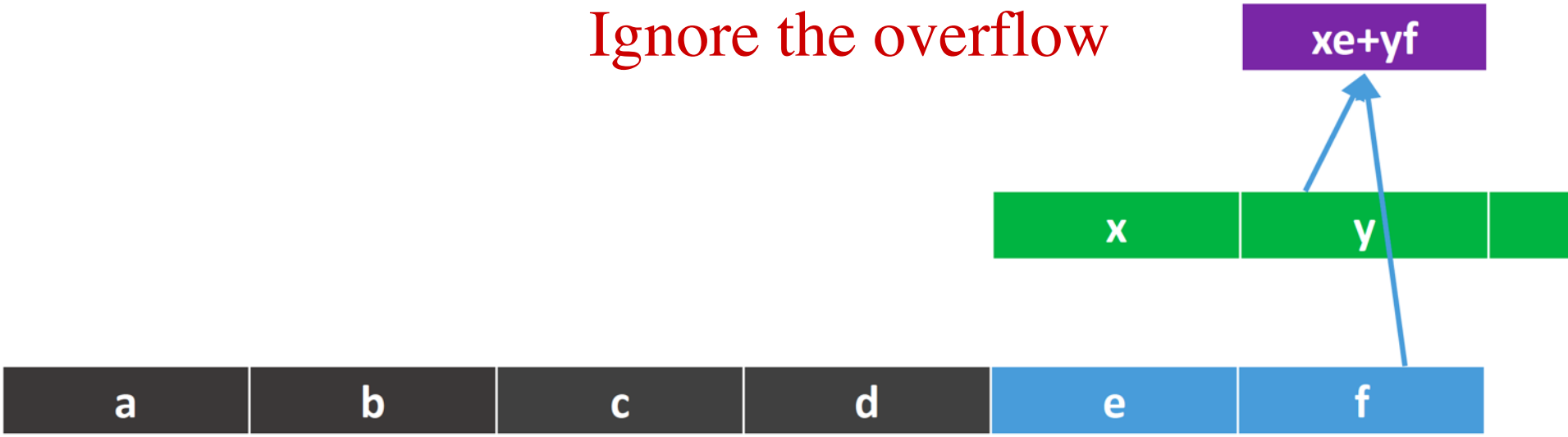
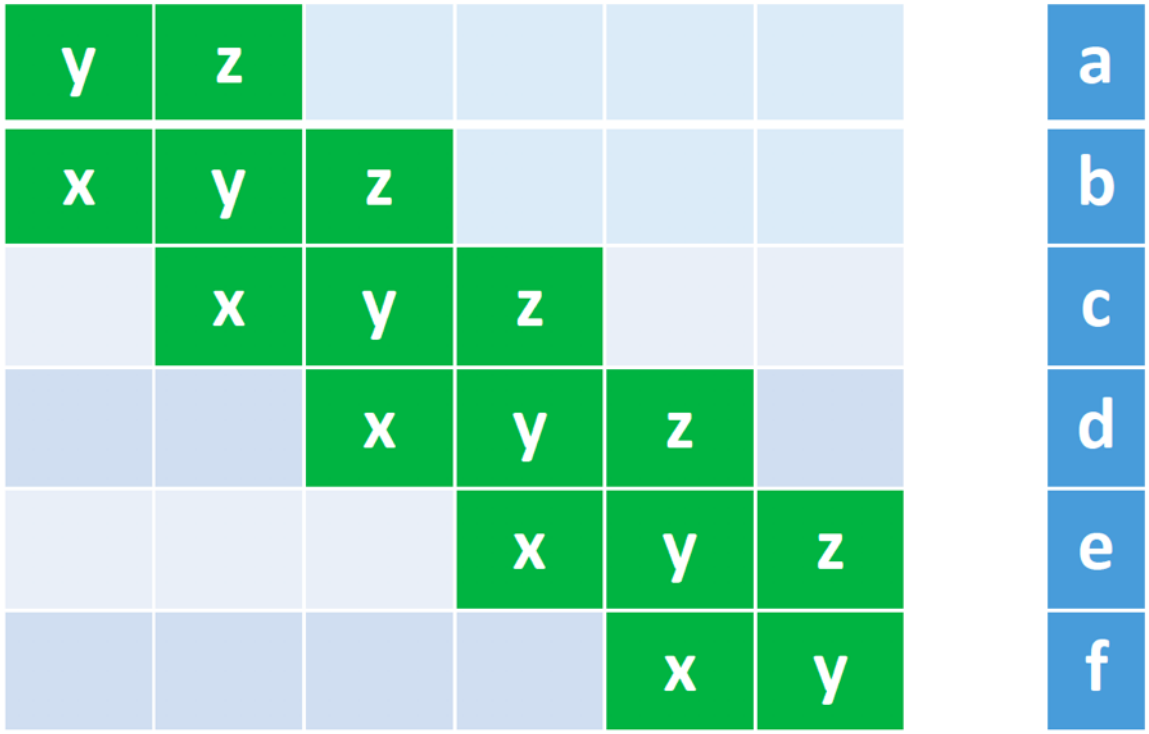
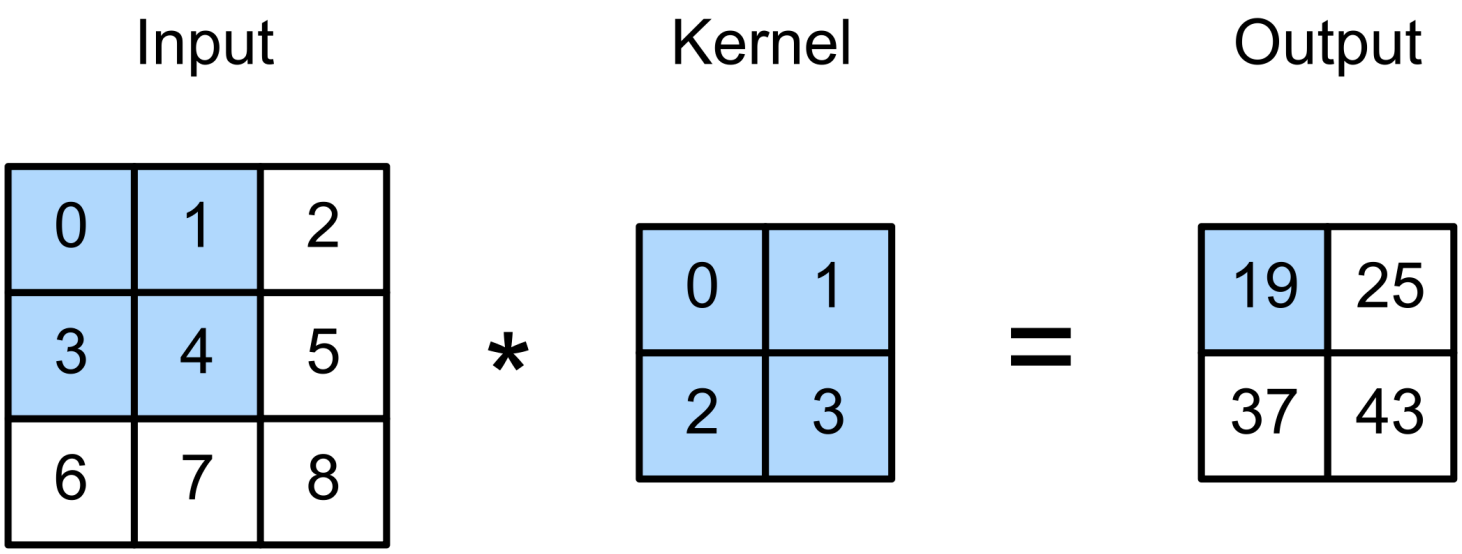


Illustration 1 as matrix multiplication

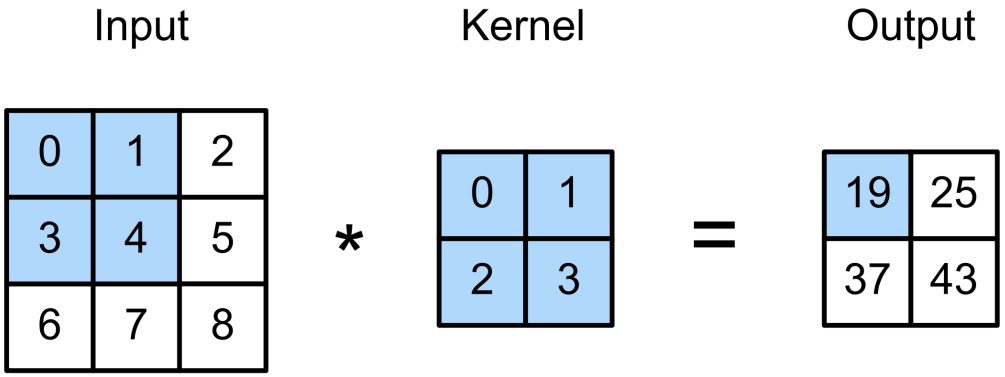


2D Convolution



- $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$
- Same calculation for other entries

2D Convolution



- $X : n_h \times n_w$ input matrix
- $W : k_h \times k_w$ kernel matrix

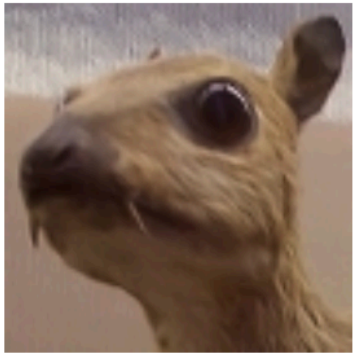
W is learnable

- $Y : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$Y = X * W$$

$+b$ if bias is added

Examples

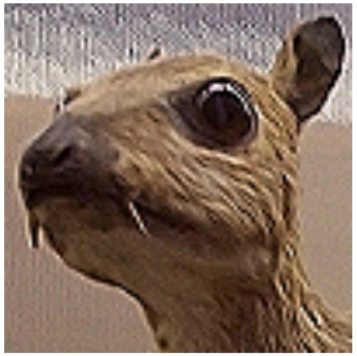


(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

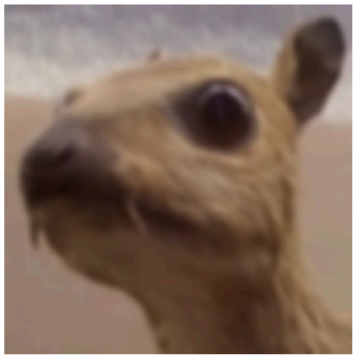
Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Gaussian Blur

Examples

(Rob Fergus)

Convolutional Neural Networks

- CNN: neural networks that use convolution in place of general matrix multiplication in at least one of their layers

- Strong empirical performance

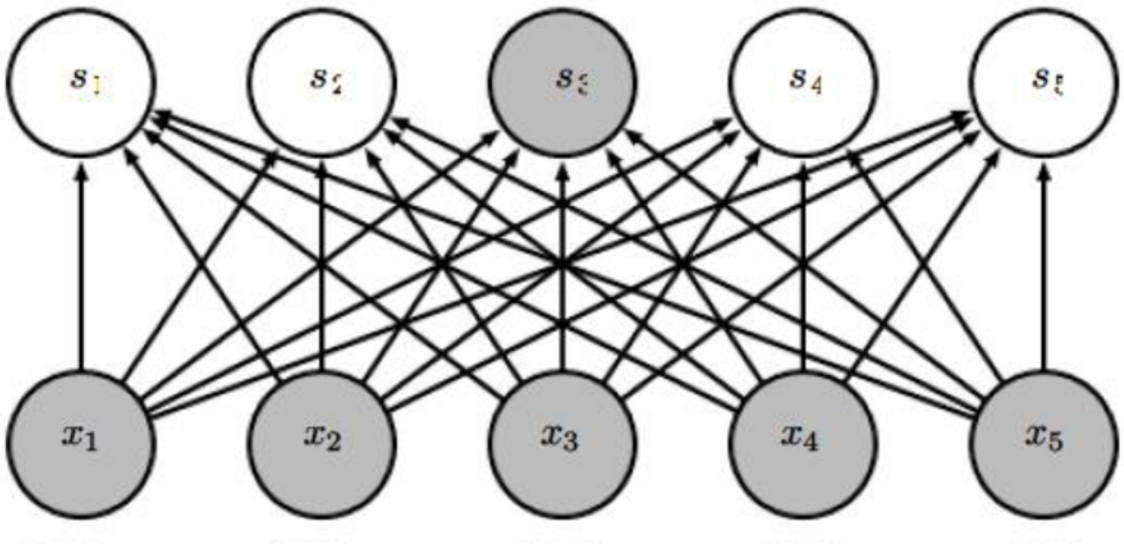
This repository contains CIFAR-10 classifier using

1. MLP
2. CNN

MLP achives a 53% accuracy while CNN achieves a 85% accurcay. Implemented in Keras with a Tensorflow backend. There are two sets of pre-trained weights for each.

Advantage: sparse interaction

Fully connected layer, $m \times n$ edges



m output nodes

n input nodes

Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

$O(nm)$ paras. & computation

Advantage: sparse interaction

Convolutional layer, $\leq m \times k$ edges

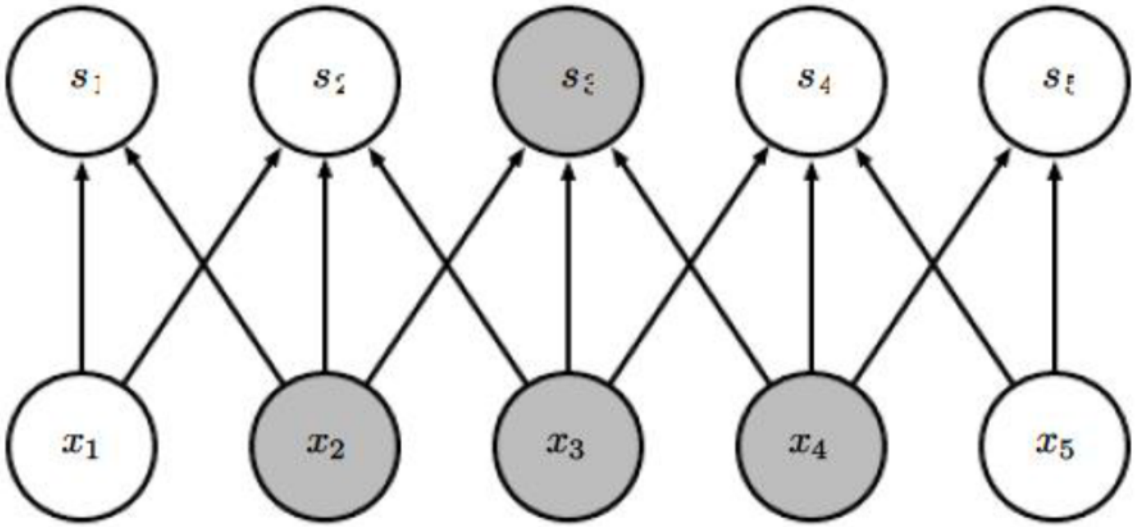
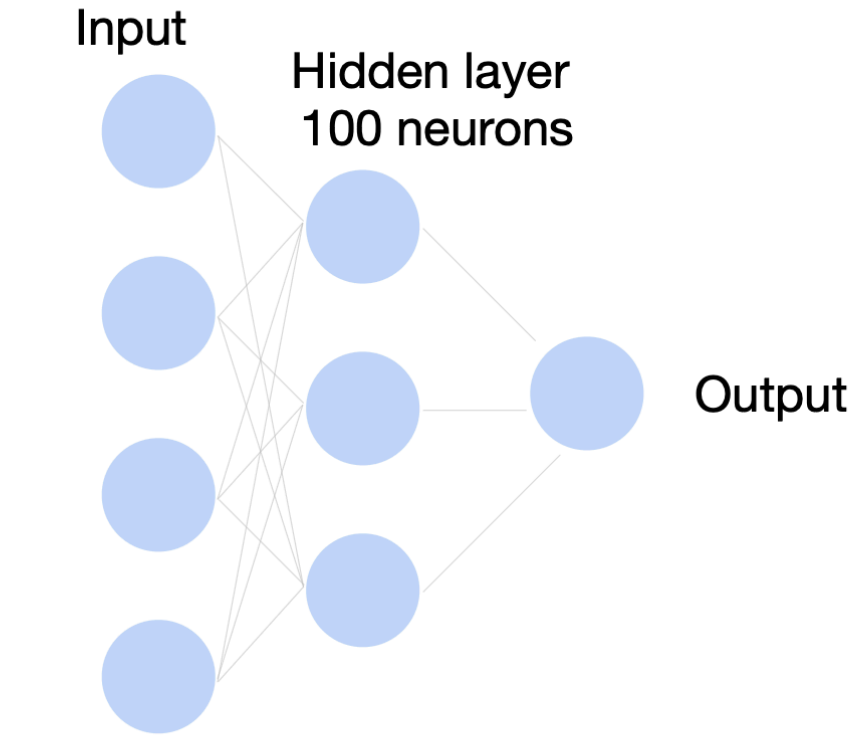


Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

$O(k)$ paras. & $O(km)$ computation

Fully Connected Networks

Cats vs. dogs?



~ 36M elements x 100 = ~**3.6B** parameters!

CNN: a couple hundreds paras.

Advantage: equivariant representations

- Equivariant: transforming the input = transforming the output

- Example: input is an image, transformation is shifting
- Convolution(shift(input)) = shift(Convolution(input))

- Useful when care only about the existence of a pattern, rather than the location

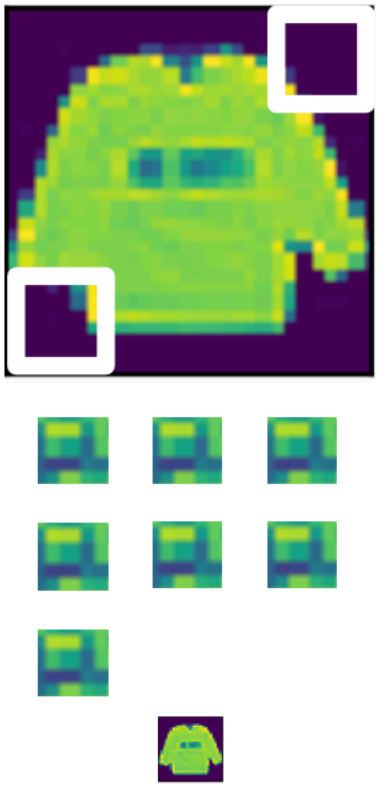

Padding and Stride

Padding

- Given a 32×32 input image
- Apply Convolution with 5×5 kernel
  - Get a 28×28 output
  - Shape reduces from $n_h \times n_w$ to

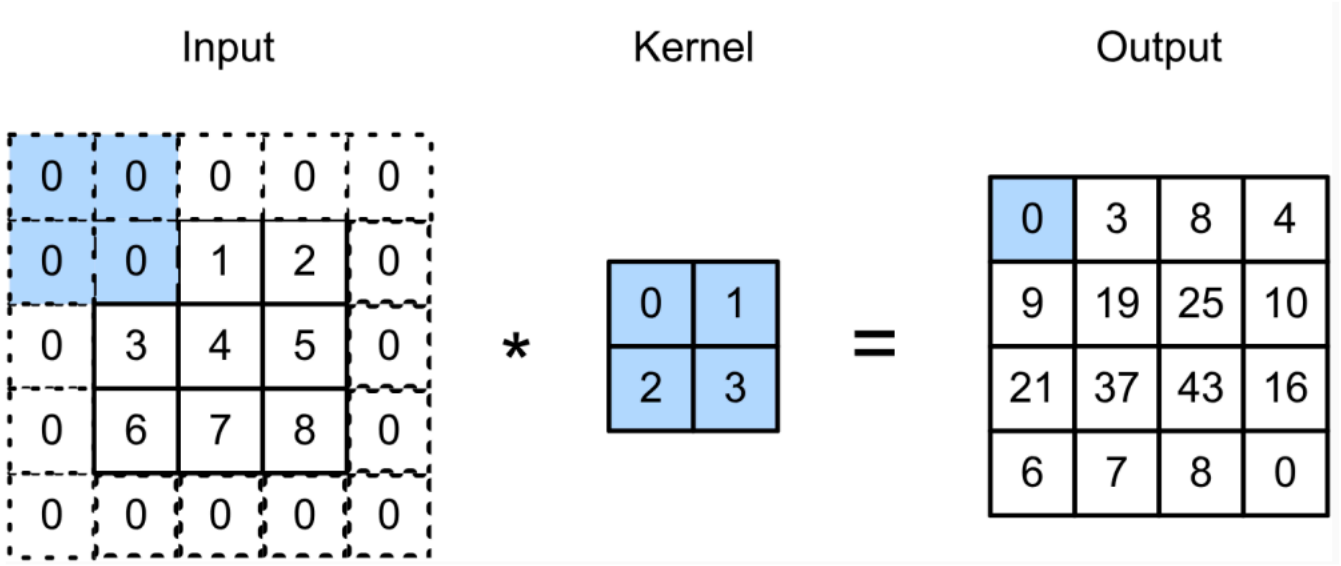
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

- Apply another 5×5 kernel
  - Get a 24×24 output
  - Image get reduced again …



Padding

- A method to prevent size reduction
  - Simply add rows/columns around input



$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

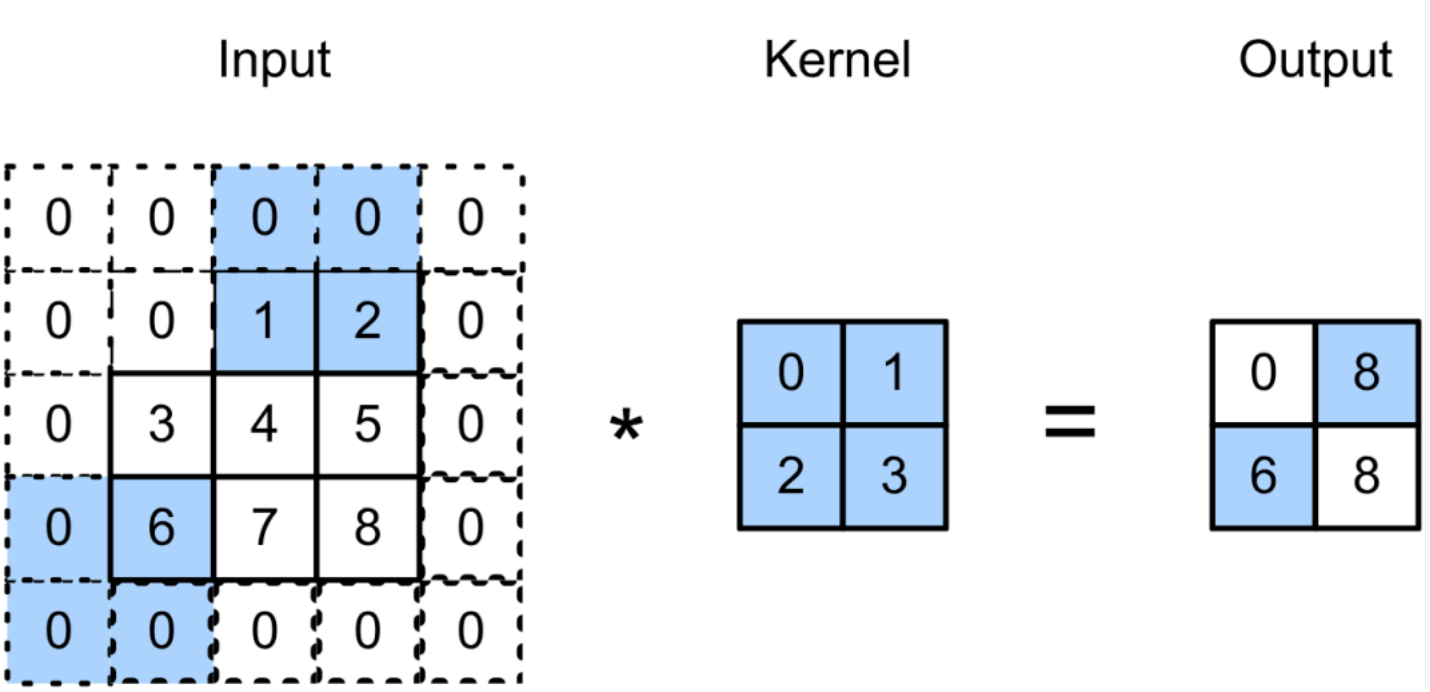
- Padding p_h rows (on both sides) and p_w columns (on both sides), output shape is

$$(n_h + 2p_h - k_h + 1) \times (n_w + 2p_w - k_2 + 1)$$

- To keep the original image size:
  - Choose $2p_h = k_h - 1$ and $2p_w = k_w - 1$

Stride

- Stride is the #rows/#columns per slide
  - Strides of 3 for height and 2 for width



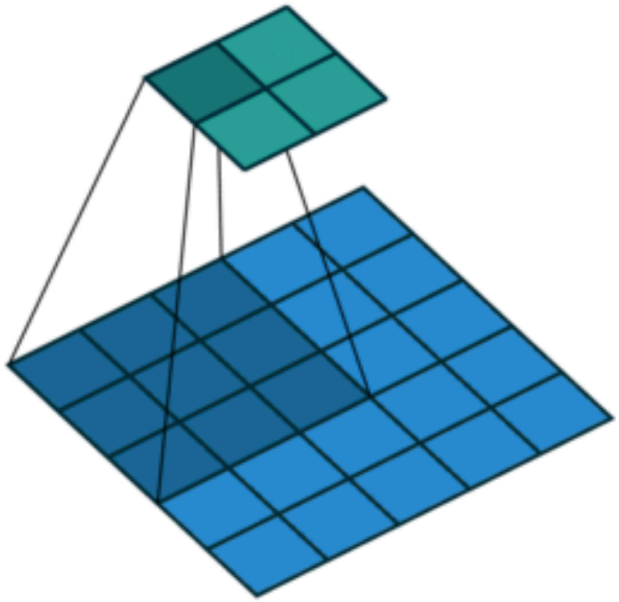
$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$

$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

Stride

- Given stride s_h for height and s_w for width, the output shape is

$$\lfloor (n_h + 2p_h - k_h)/s_h + 1 \rfloor \times \lfloor (n_w + 2p_w - k_w)/s_w + 1 \rfloor$$
$$= \lfloor (n_h + 2p_h - k_h + s_h)/s_h \rfloor \times \lfloor (n_w + 2p_w - k_w + s_w)/s_w \rfloor$$



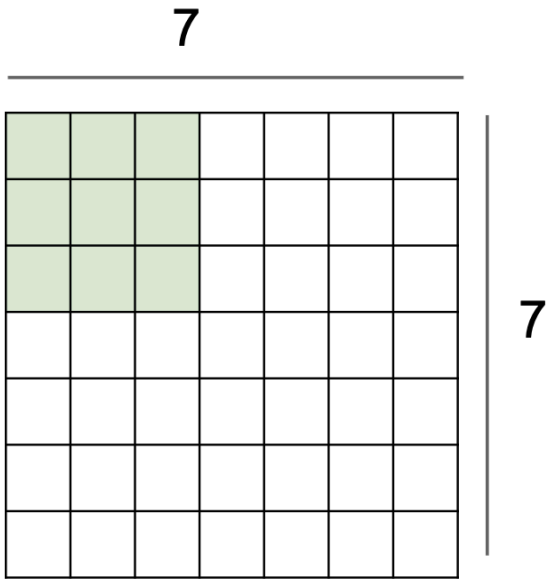
Q1. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

A. 3x3

B. 7x7

C. 5x5

D. 2x2



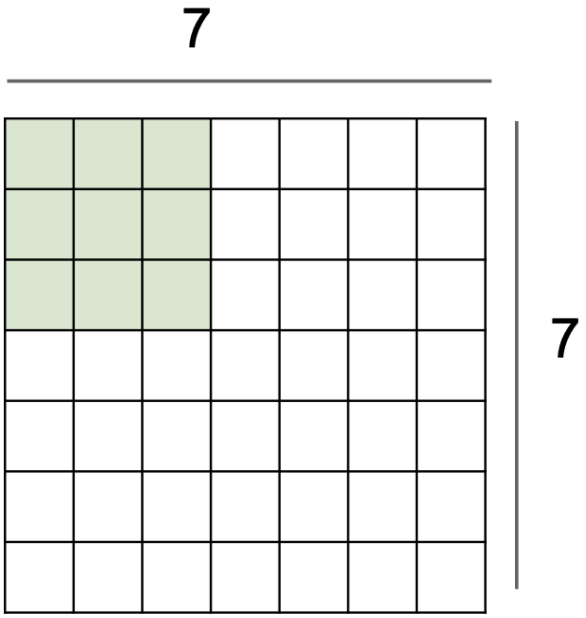
Q1. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

A. 3x3

B. 7x7

C. 5x5

D. 2x2



$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$


Multiple Input and
Output Channels

Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

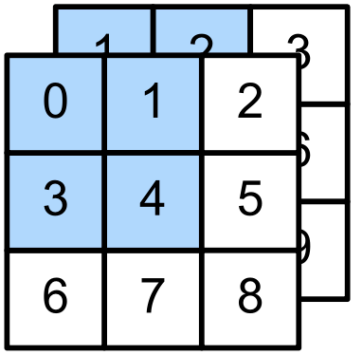
Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Have a kernel for each channel and sum the results together

Input



Multiple Input Channels

- Input $X : n_h \times n_w \times C$
- Kernel: $W : k_h \times k_w \times C$
- Output: Y

$$Y = \sum_{i=0}^{C-1} X_{:,:,i} * W_{:,:,i}$$

Multiple Output Channels

- We have discuss the case with a single output
- We have multiple kernels, each generates an output channel

Each 3-D kernel may recognize a particular pattern

(Gabor filters)

Multiple Output Channels

- Input $X : n_h \times n_w \times C$
- Kernel: $W : k_h \times k_w \times C \times D$
- Output: $Y : m_h \times m_w \times D$
- For any $d \in \{0, \ldots, D-1\}$, we have

$$Y_{:,:,d} = \sum_{i=1}^{C-1} X_{:,:,i} * W_{:,:,i,d}$$

Q3-1. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of scalar multiplications involved in this operation (without considering any optimization in matrix multiplication)?

A. 64x3x3x222x222

B. 64x3x3x222

C. 3x3x222x222

D. 64x3x3x3x222x222

Q3-1. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of scalar multiplications involved in this operation (without considering any optimization in matrix multiplication)?

A. 64x3x3x222x222

B. 64x3x3x222

C. 3x3x222x222

D. 64x3x3x3x222x222

Q 3-2. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of learnable parameters?

A. 64x222x222

B. 64x3x3x222

C. 3x3x3x64

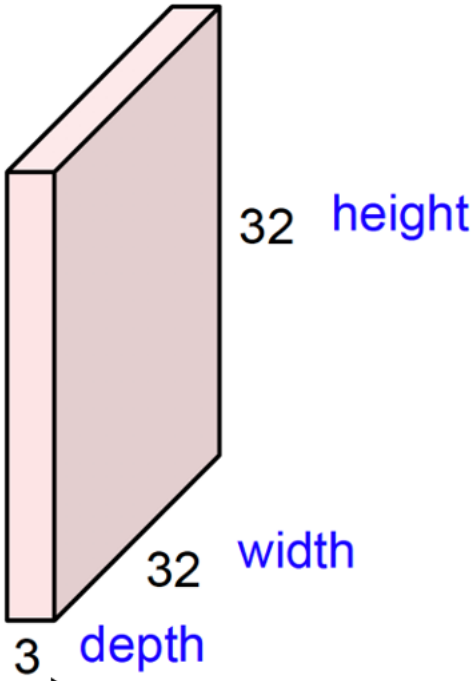
D. (3x3x3+1)x64

C if no bias, D if there is bias

Break

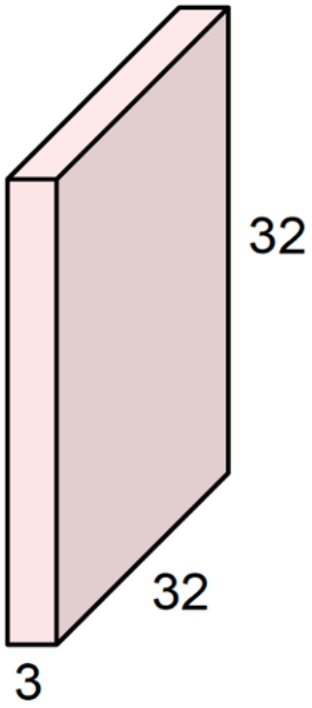
Convolutional Layer

32x32x3 image -> preserve spatial structure

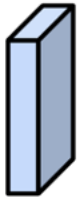


Convolutional Layer

32x32x3 image



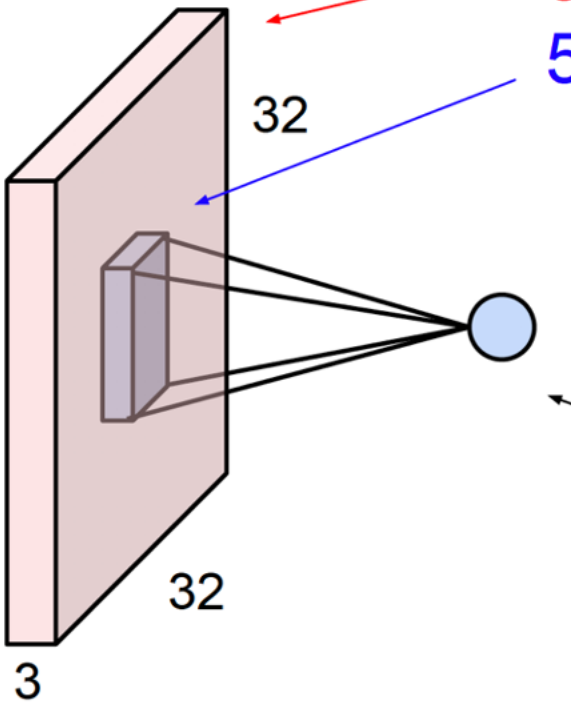
5x5x3 filter



Kernel has the full depths as input

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

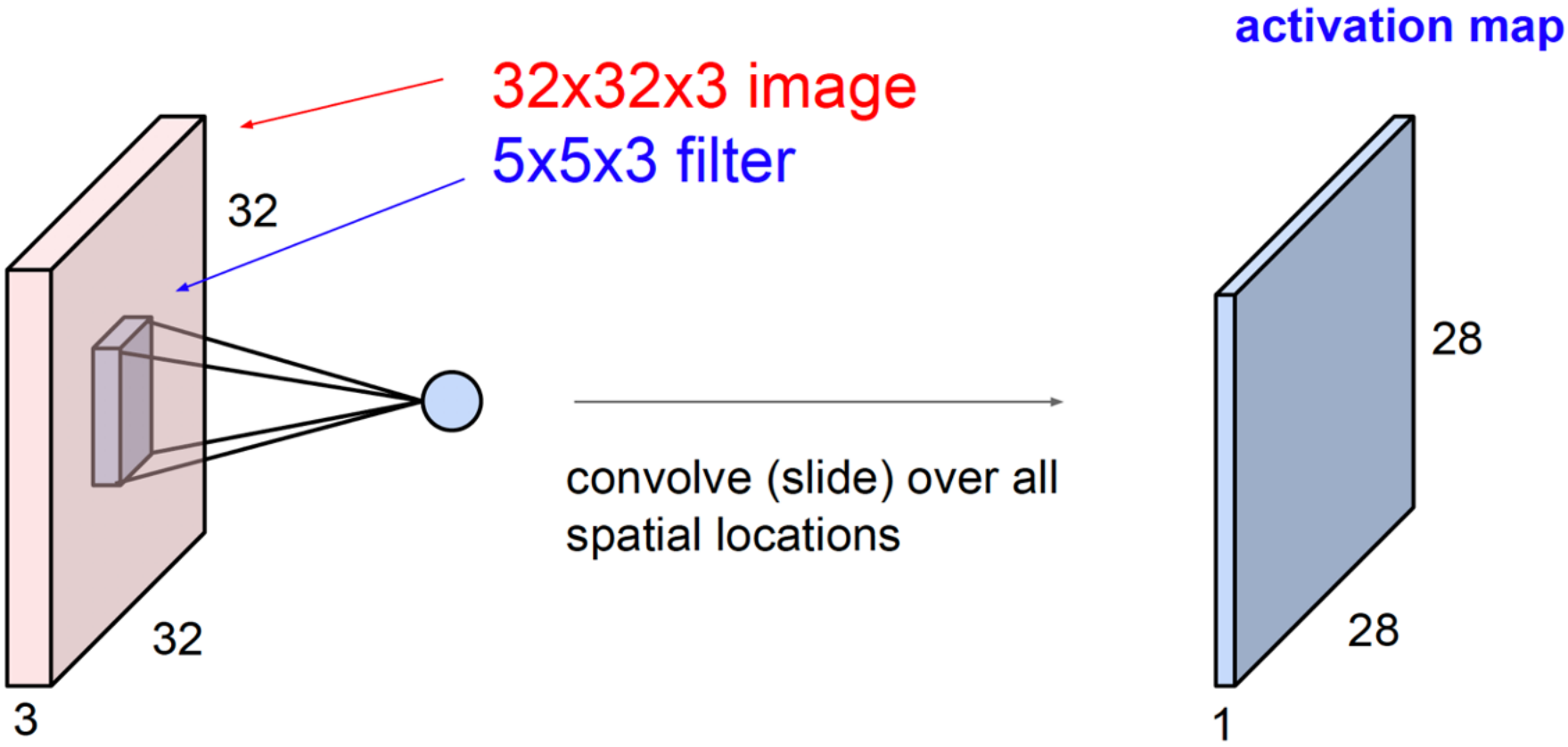
Convolutional Layer



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

$$w^T x + b$$

Convolutional Layer

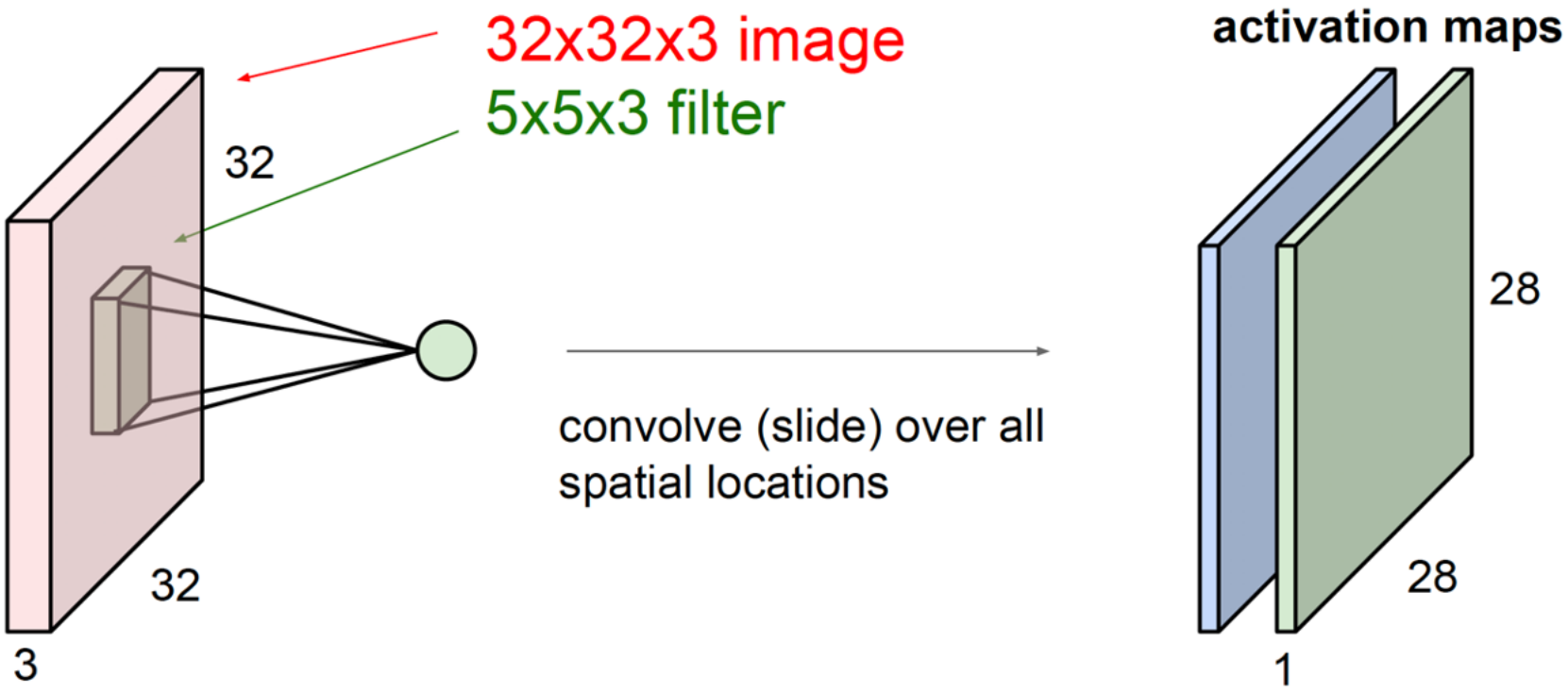


Note: If no zero padding, dimension goes 32->28

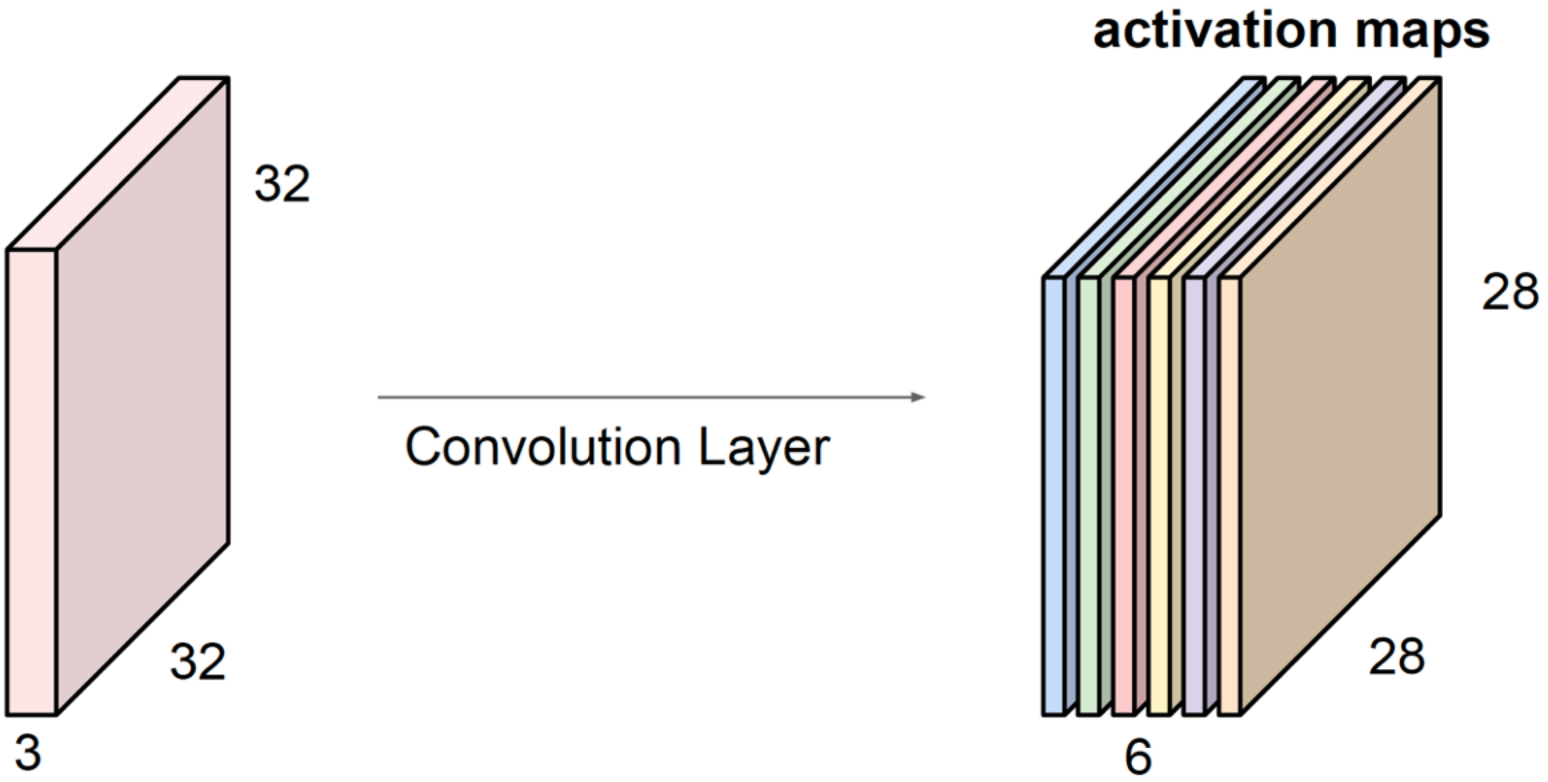
Convolutional Layer

Convolution Layer

consider a second, green filter

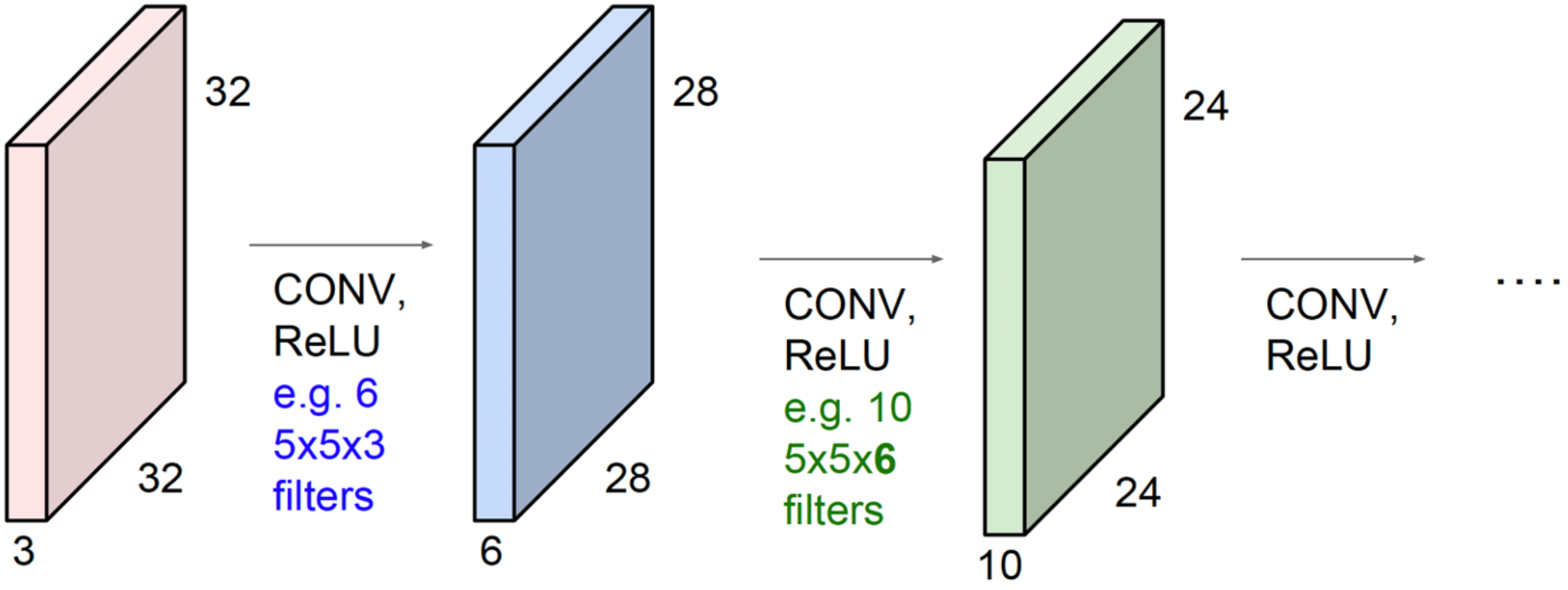


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

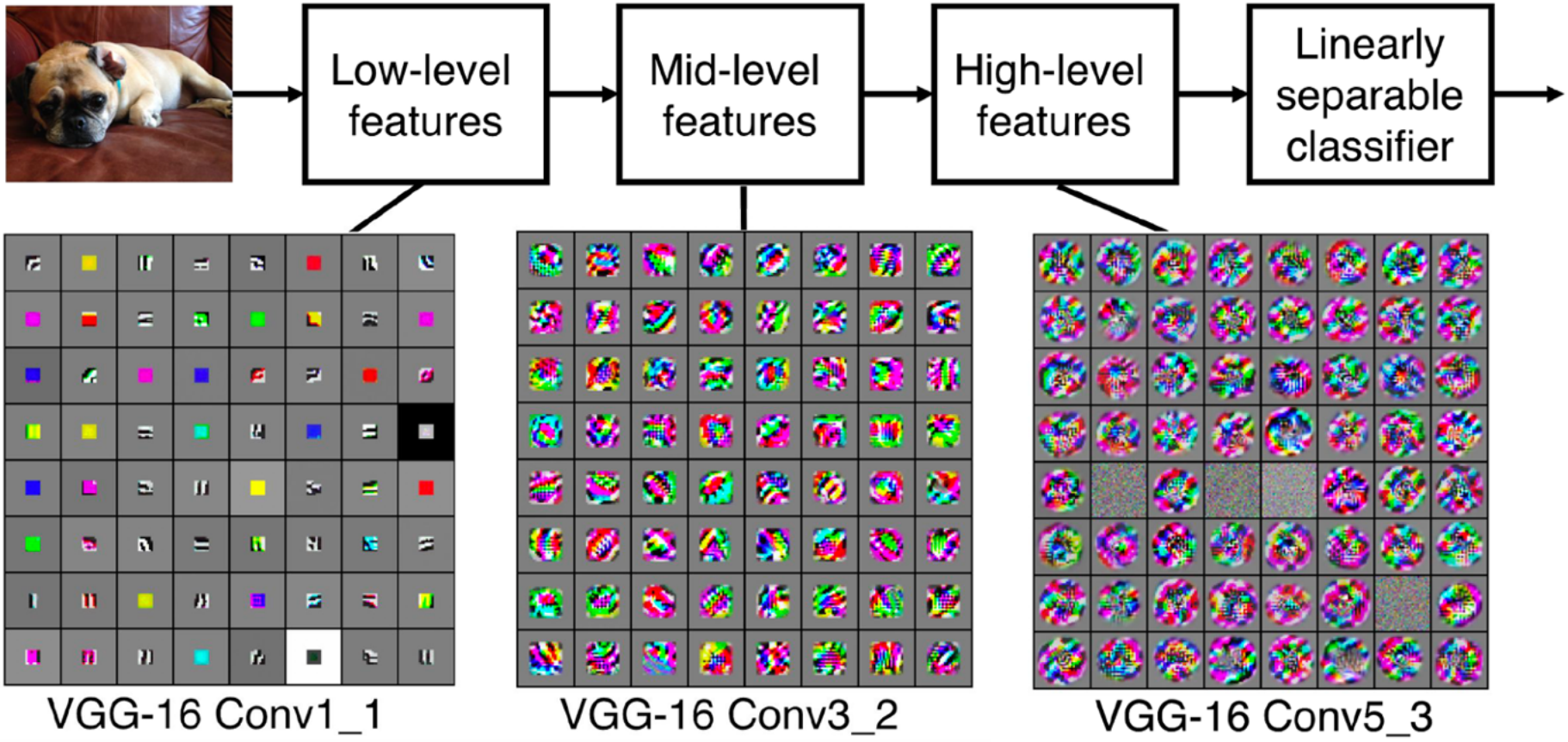
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

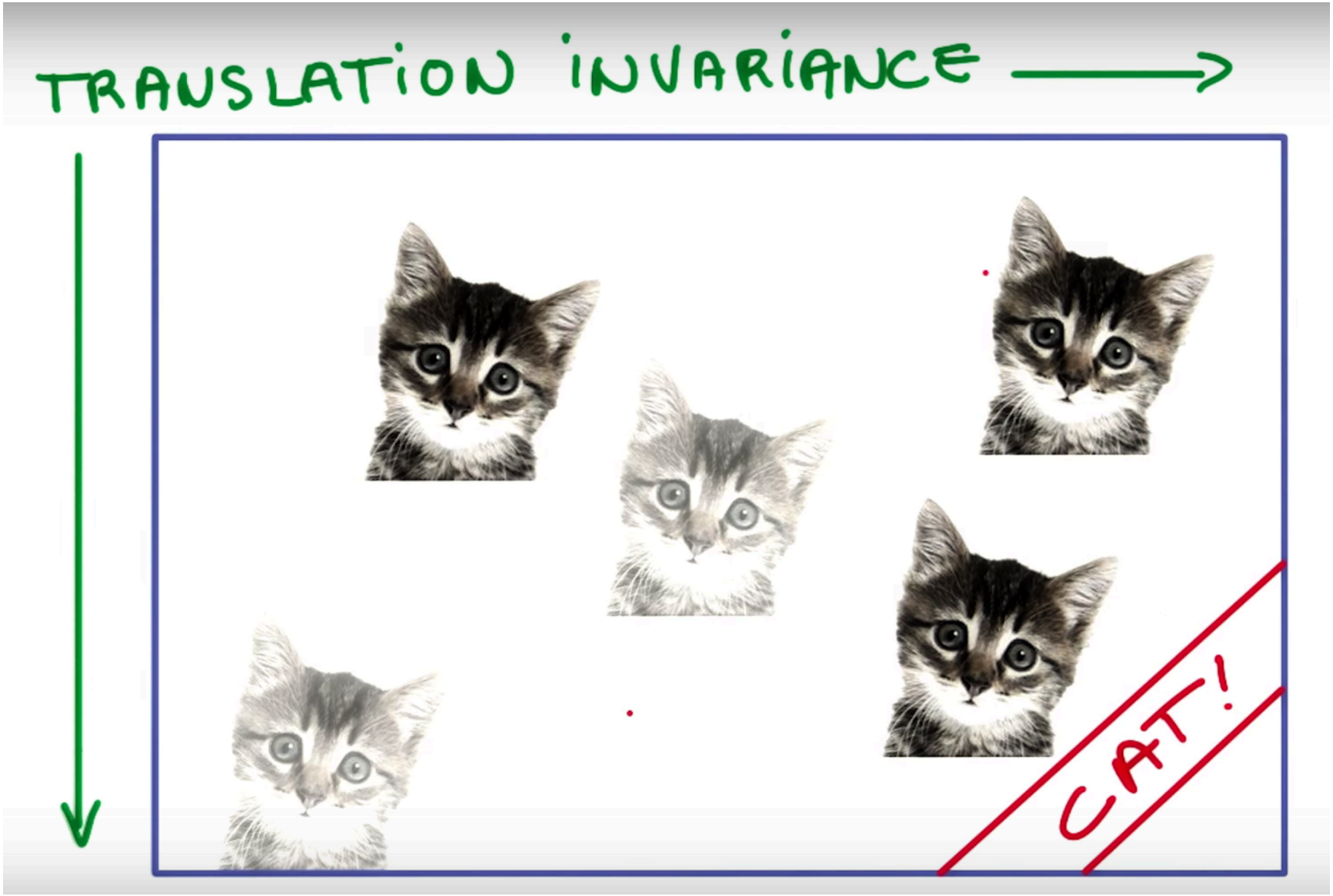


Preview

[Zeiler and Fergus 2013]

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

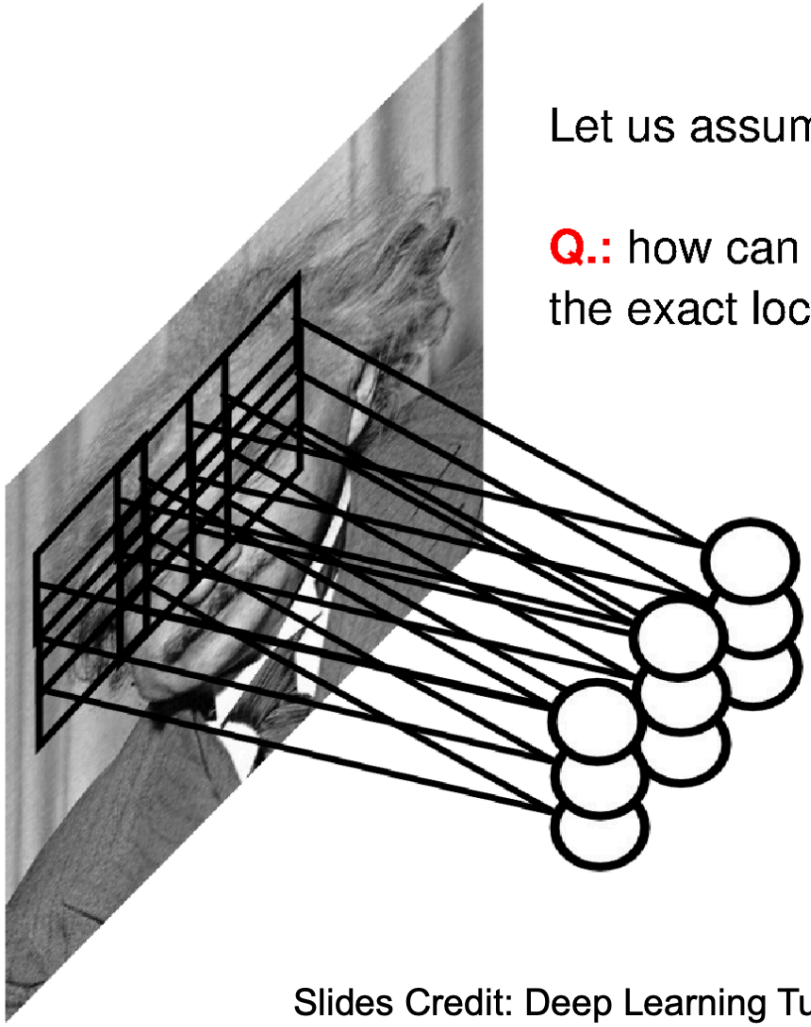




How to be invariant?

Pooling

Pooling

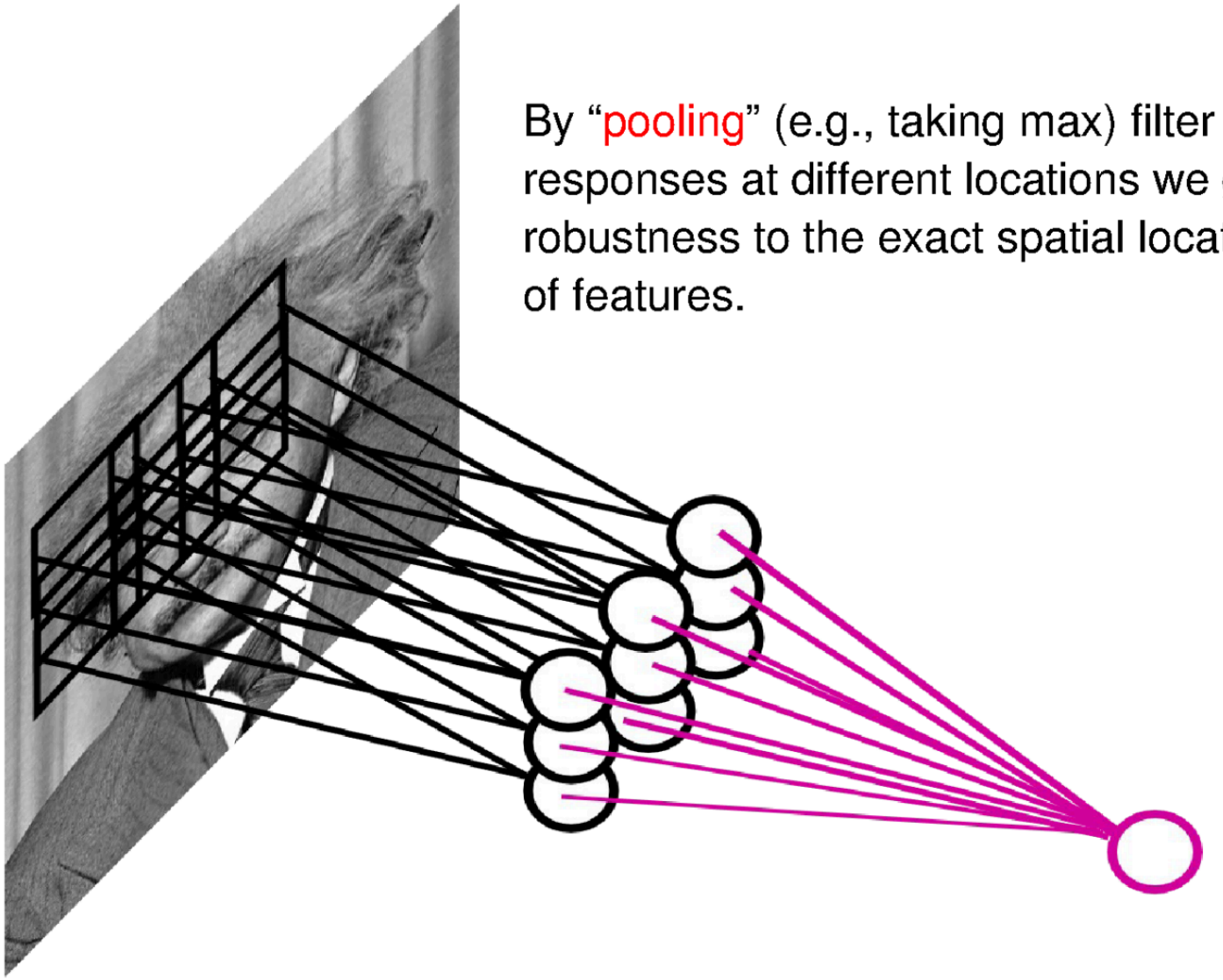


Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Slides Credit: Deep Learning Tutorial by Marc’Aurelio Ranzato

Pooling

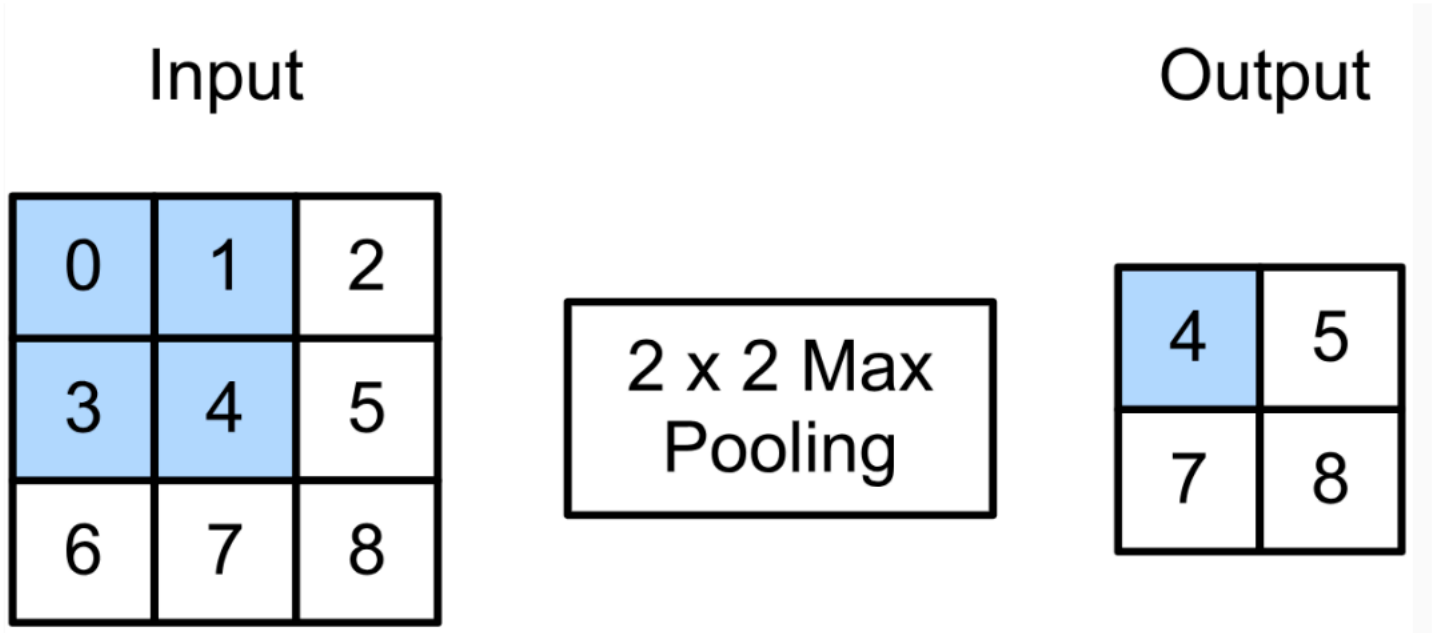


By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Max Pooling

- Return the maximum value in the sliding window



$$\max(0,1,3,4) = 4$$

Padding, Stride, and Multiple Channels

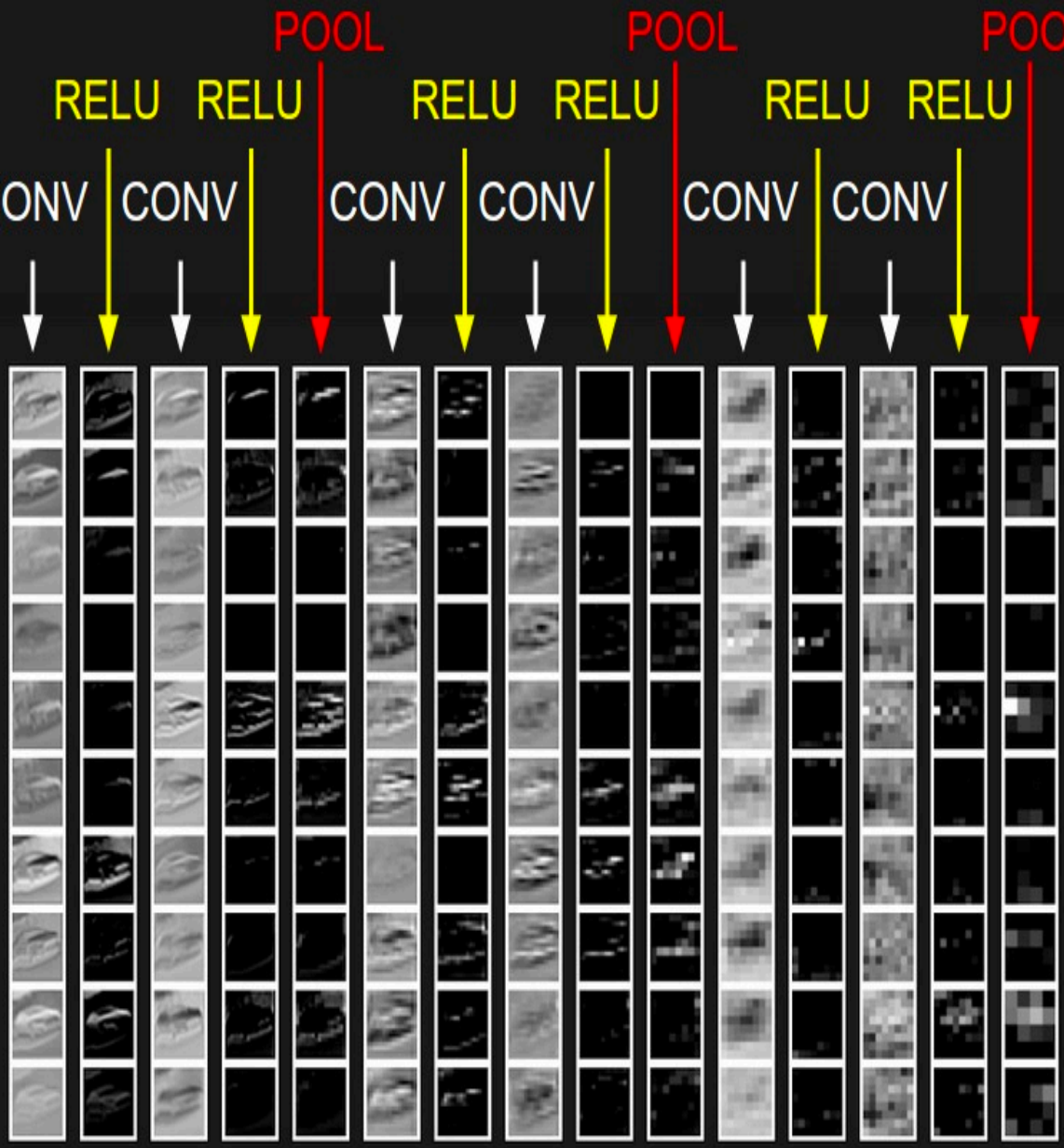
- Pooling layers have similar padding and stride as convolutional layers
- Number of parameters: 0
- Apply pooling for each input channel to obtain the corresponding output channel

output channels = # input channels

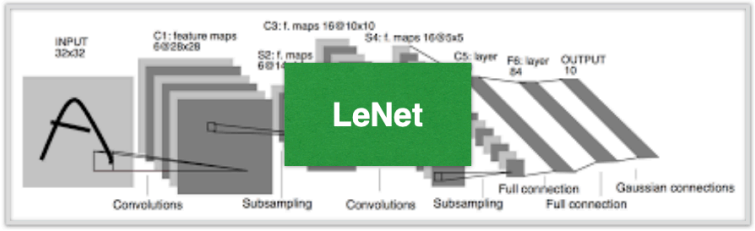
Average Pooling

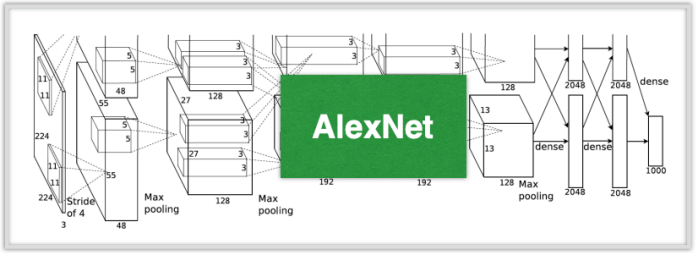
- Max Pooling: Extract the strongest singal in a window
- Average pooling: replace “Max” with “Mean” in max pooling
  - Extract the average final in a window

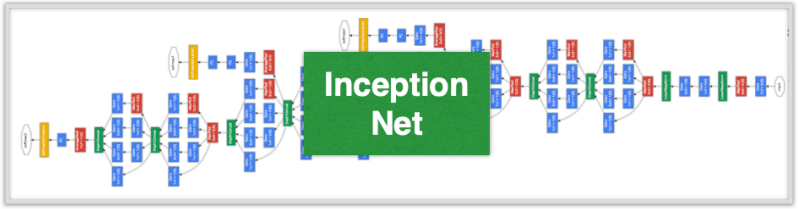
Convolutional Neural Networks

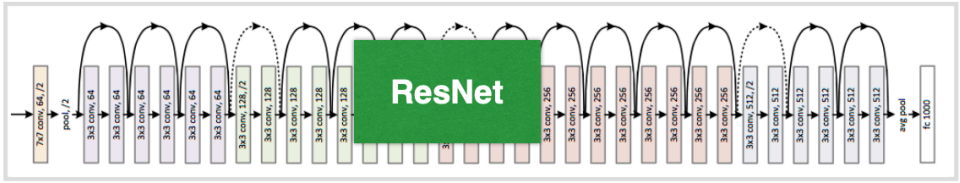


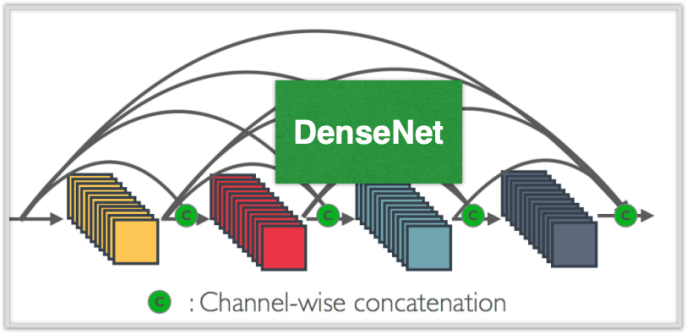
Evolution of neural net architectures







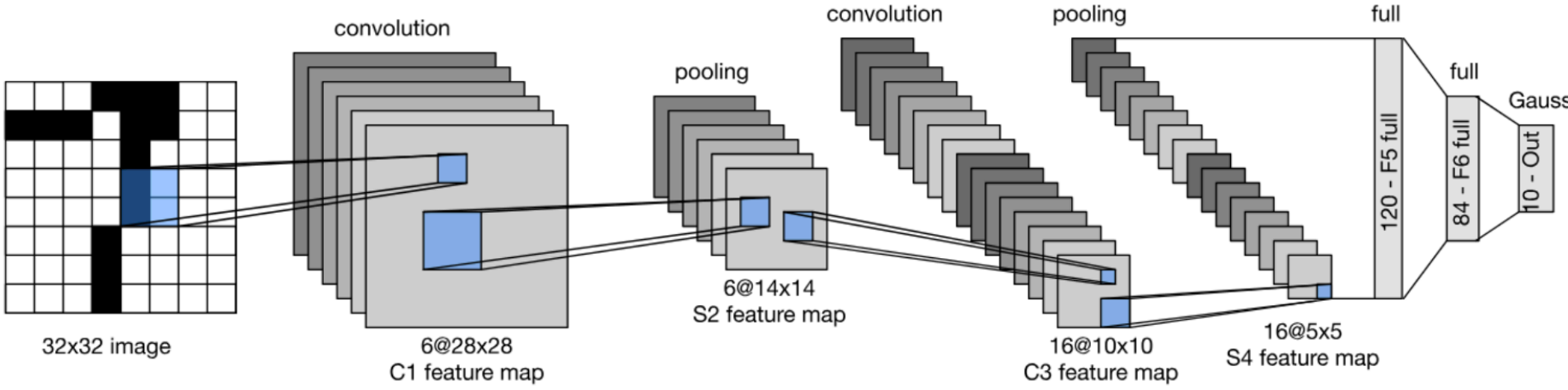




LeNet

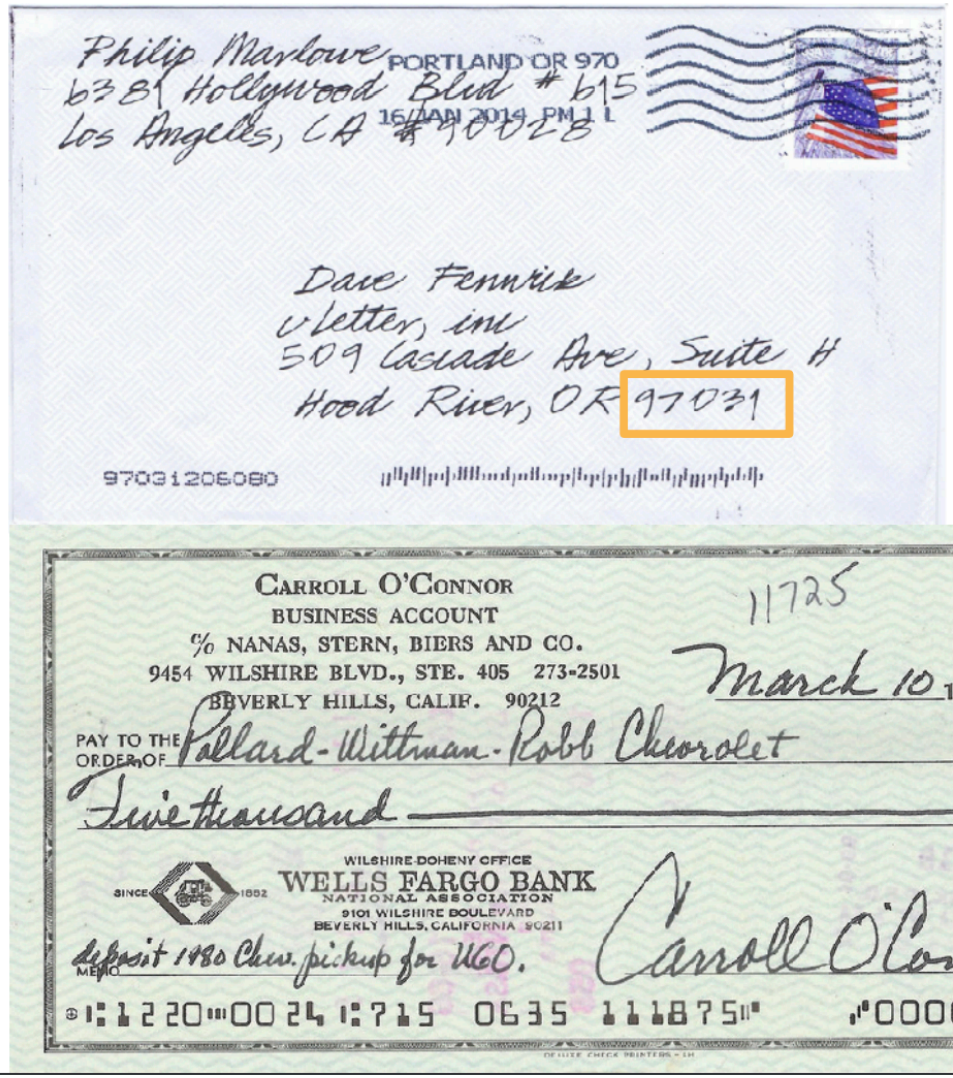
- One of the earliest convolutional neural networks
- Developed by Turing Award Laureate Yann LeCun in 1980s

LeNet Architecture



gluon-cv.mxnet.io

Handwritten Digit Recognition

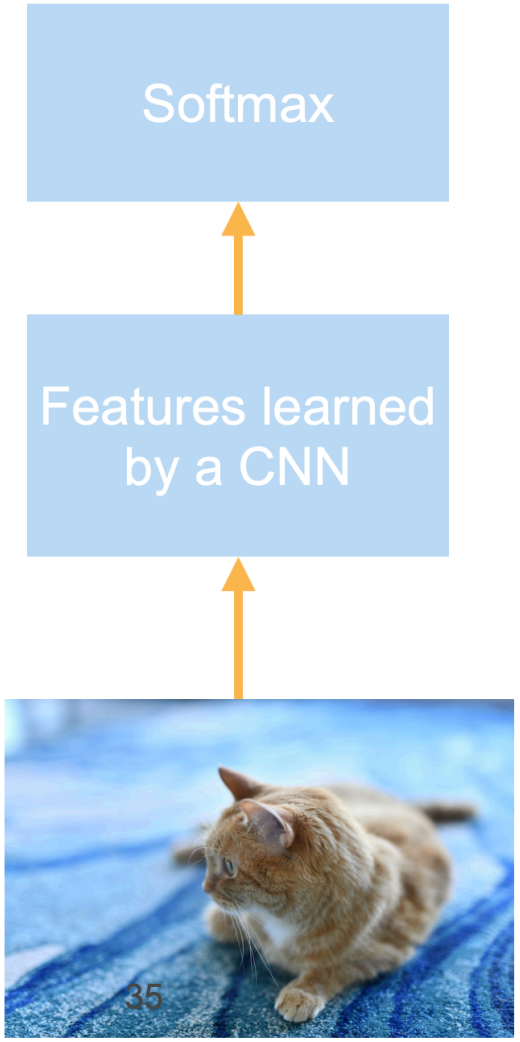


LeNet in Pytorch

```
def __init__(self):
    super(LeNet5, self).__init__()
    # Convolution (In LeNet-5, 32x32 images are given as input. Hence padding of 2 is done below)
    self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2, bias=True)
    # Max-pooling
    self.max_pool_1 = torch.nn.MaxPool2d(kernel_size=2)
    # Convolution
    self.conv2 = torch.nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0, bias=True)
    # Max-pooling
    self.max_pool_2 = torch.nn.MaxPool2d(kernel_size=2)
    # Fully connected layer
    self.fc1 = torch.nn.Linear(16*5*5, 120)   # convert matrix with 16*5*5 (= 400) features to a matrix of 120 features (col
    self.fc2 = torch.nn.Linear(120, 84)       # convert matrix with 120 features to a matrix of 84 features (columns)
    self.fc3 = torch.nn.Linear(84, 10)        # convert matrix with 84 features to a matrix of 10 features (columns)
```

AlexNet

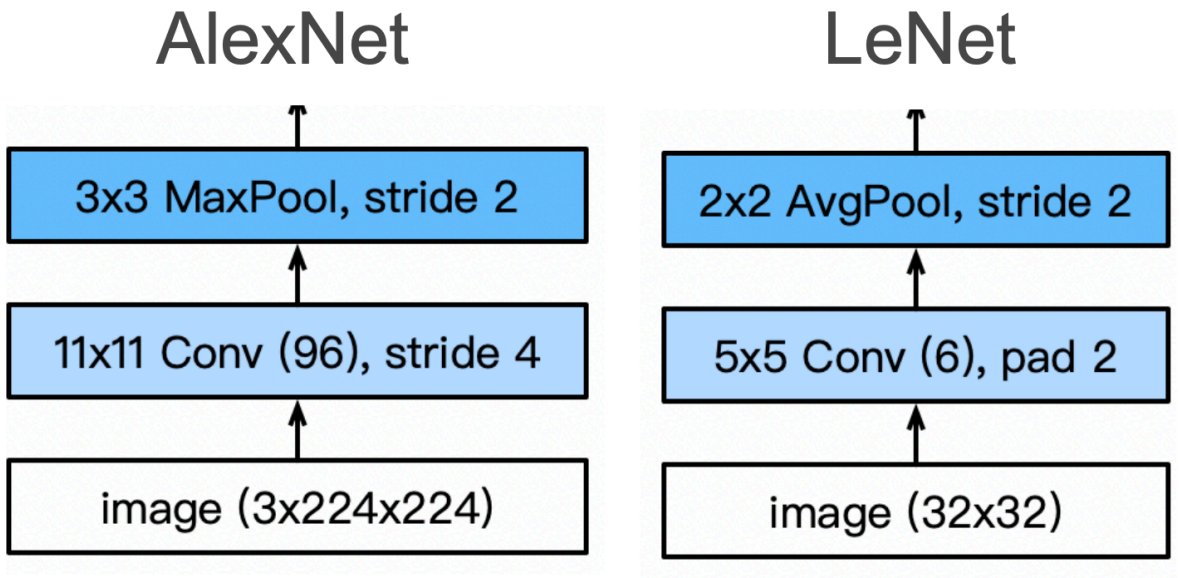
- AlexNet won ImageNet competition in 2012
- Can be thought as a deeper and bigger version of LeNet
- Paradigm shift for computer vision



AlexNet Architecture

Larger, overlapping pool size

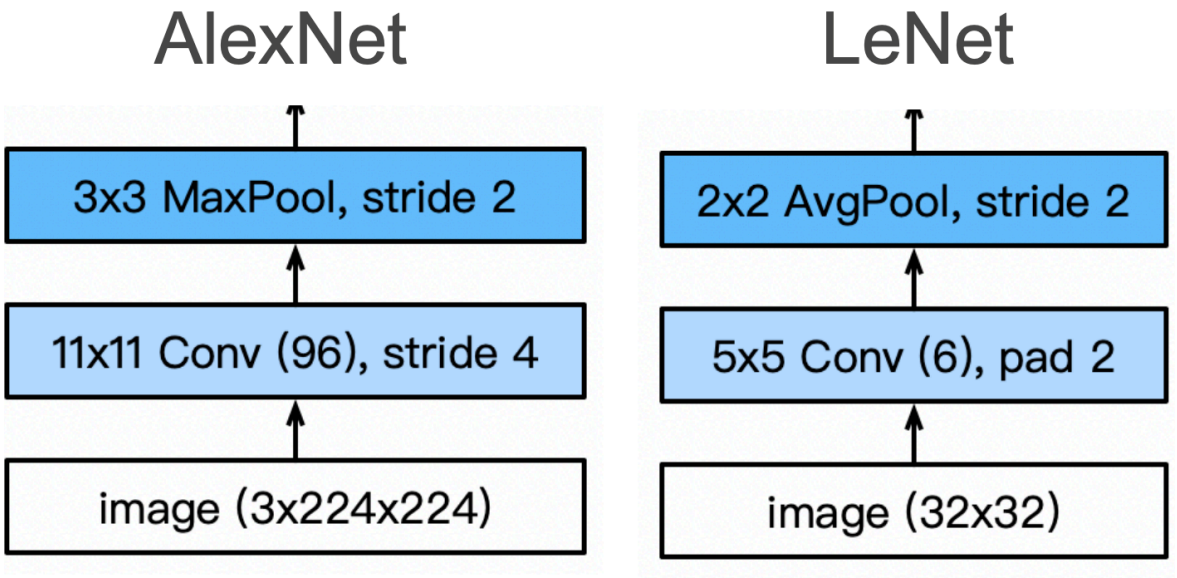
Larger kernel and stride for larger images, more output channels



Larger input size, 3 channels

$$\lfloor (n_h + 2p_h - k_h)/s_h + 1 \rfloor \times \lfloor (n_w + 2p_w - k_w)/s_w + 1 \rfloor$$

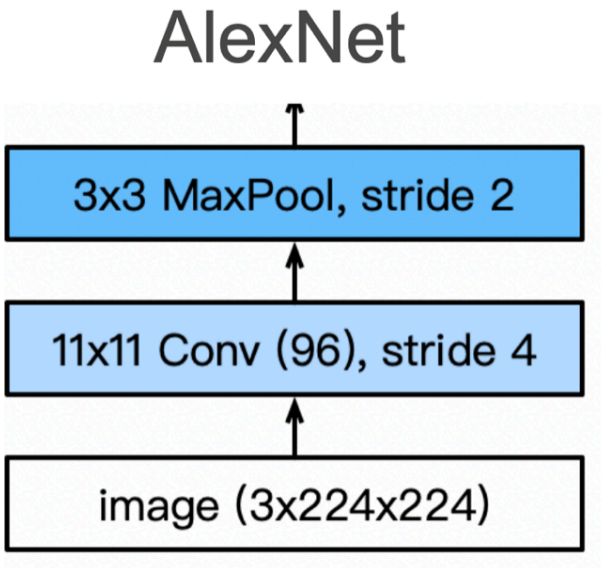
AlexNet Architecture



Q1: How many parameters in the CONV1 layer?

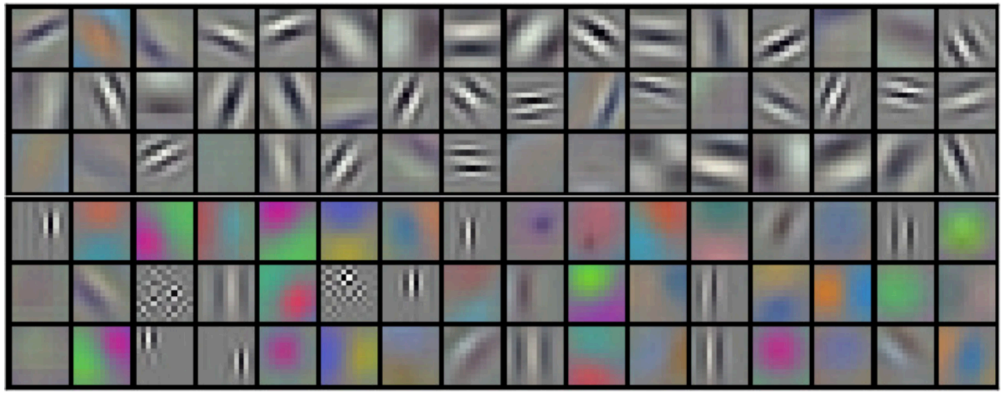
A1: $11 \times 11 \times 3 \times 96 \approx 35k$

AlexNet Architecture

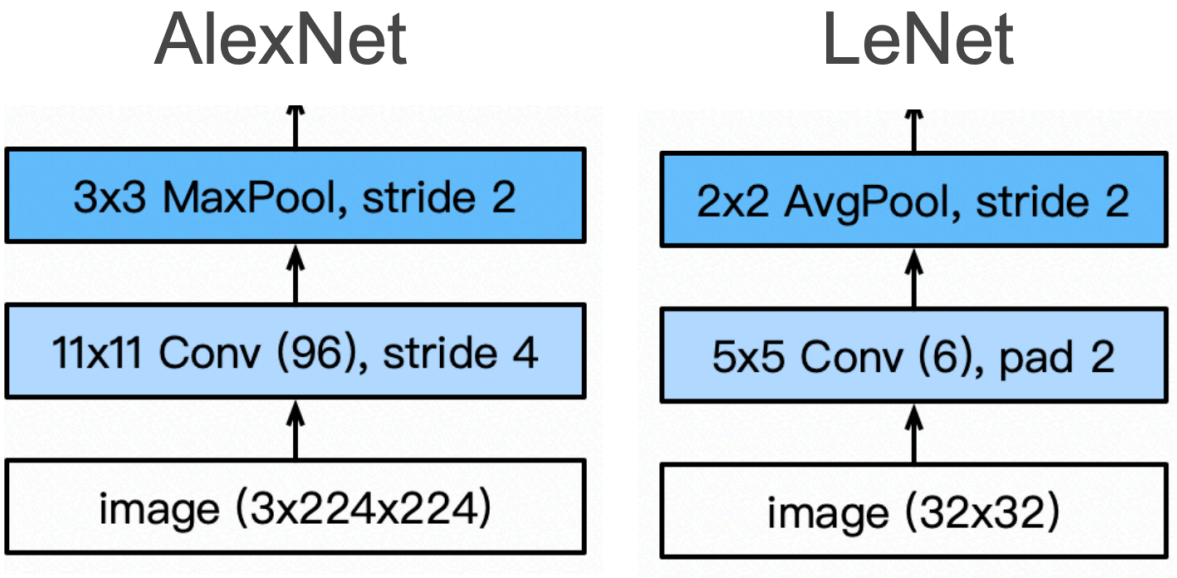


Visualization of 96 kernels of size

$11 \times 11 \times 3$



AlexNet Architecture



Q2: How many parameters in the MaxPool layer?

A2: 0.

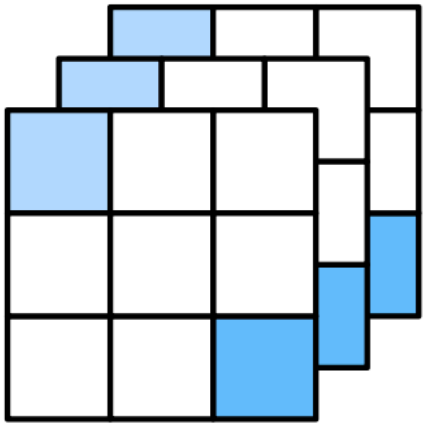
1×1 Convolution

- A kernel of size $1 \times 1 \times C \times D$, where C is the input depth and D is the output depth
- For input $X : n_h \times n_w \times C$, we have

$$Y_{i,j,d} = \sum_{k=0}^{C-1} X_{i,j,k} * W_{0,0,k,d}$$

1×1 Convolution

Input



1×1 Convolution

- Benefits:
  - An efficient way to summarize information across different channels (dimension reduction)
  - Can be easily trained—viewed as a MLP across channels (Network-in-network)

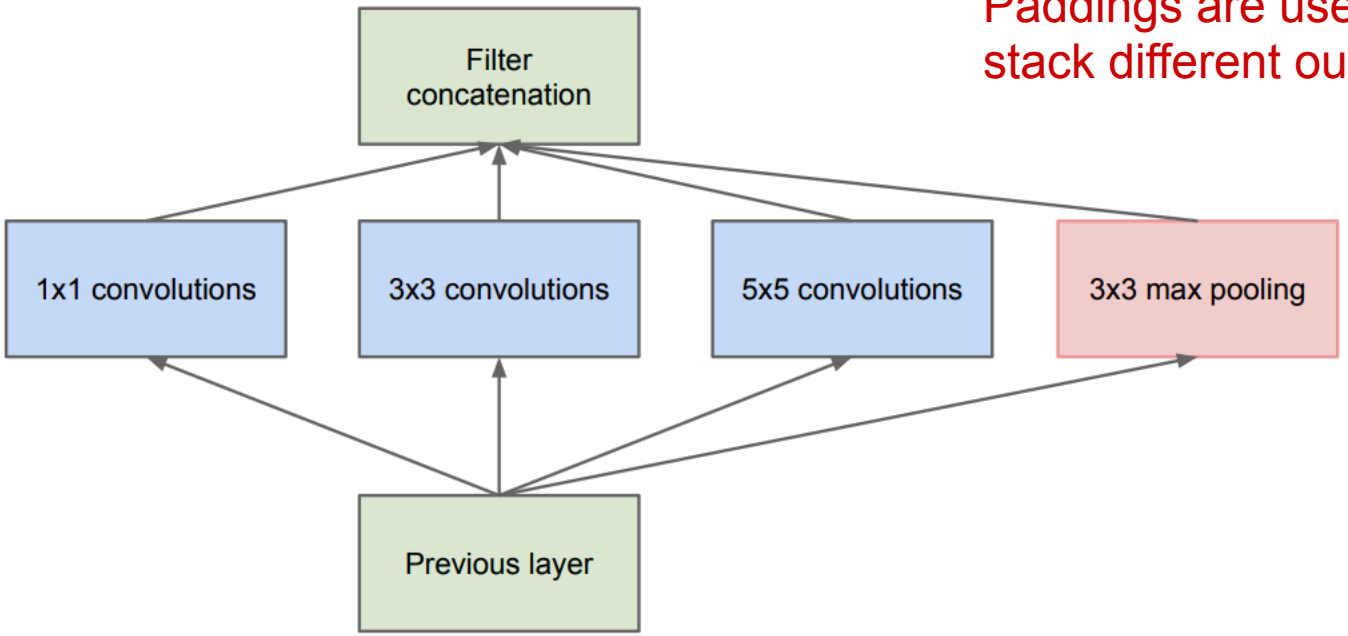
Kernels of Different Sizes

- Kernels of size $1 \times 1, 3 \times 3, 5 \times 5 \ldots$
- Each have their own functionalities

- How to choose the kernel with the “right” size??

The Inception Block in GoogLeNet

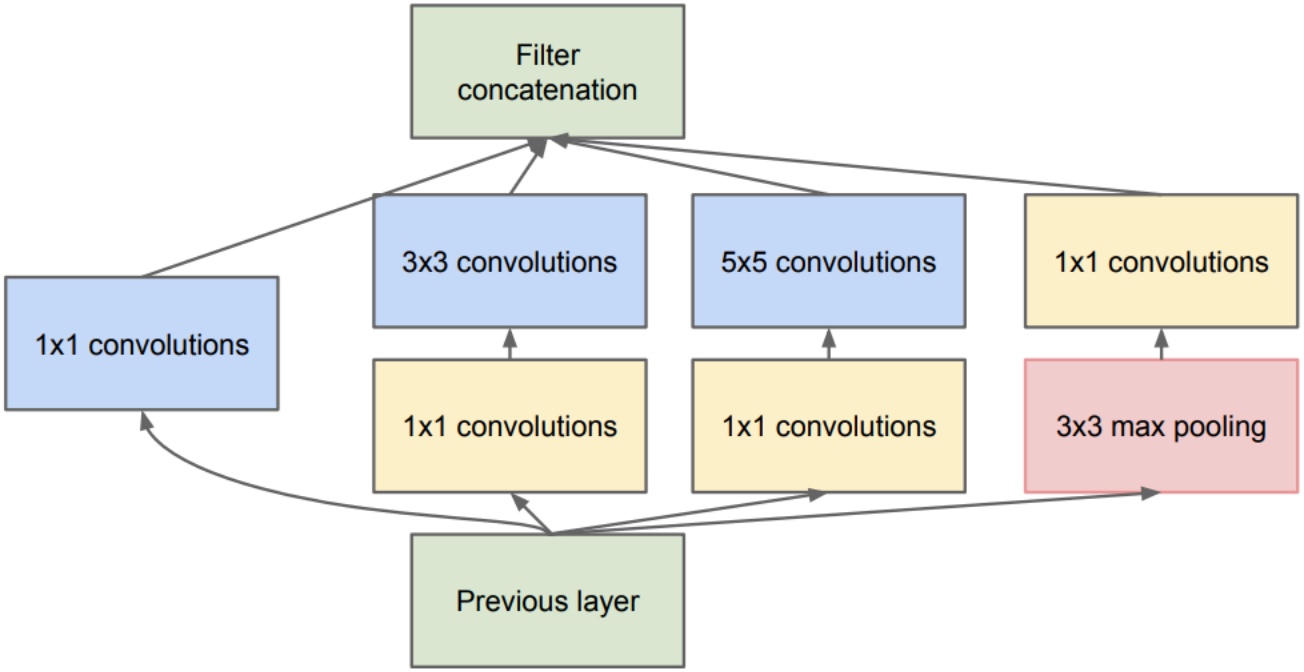
Paddings are used to stack different outputs



(a) Inception module, naïve version

paras of 5×5 Conv is $5 \times 5 \times C \times D$

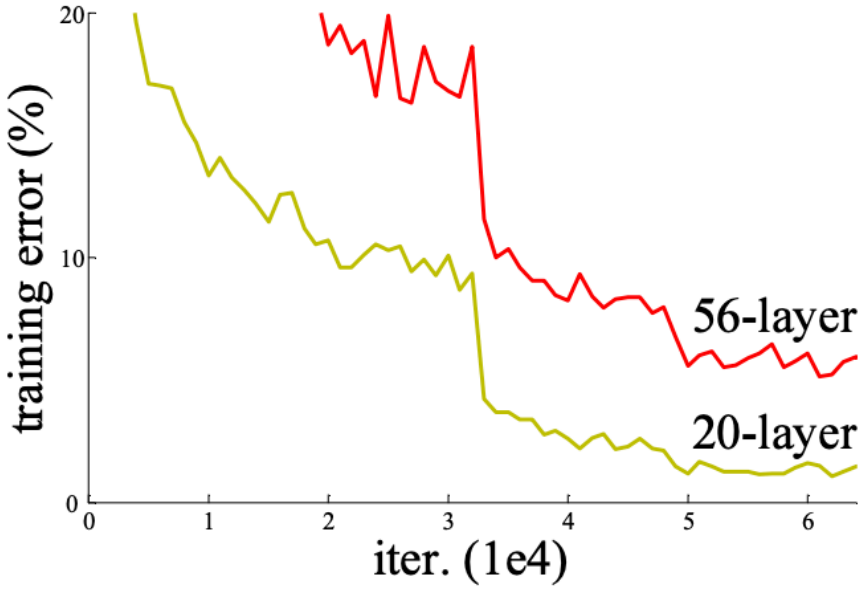
The Inception Block in GoogLeNet

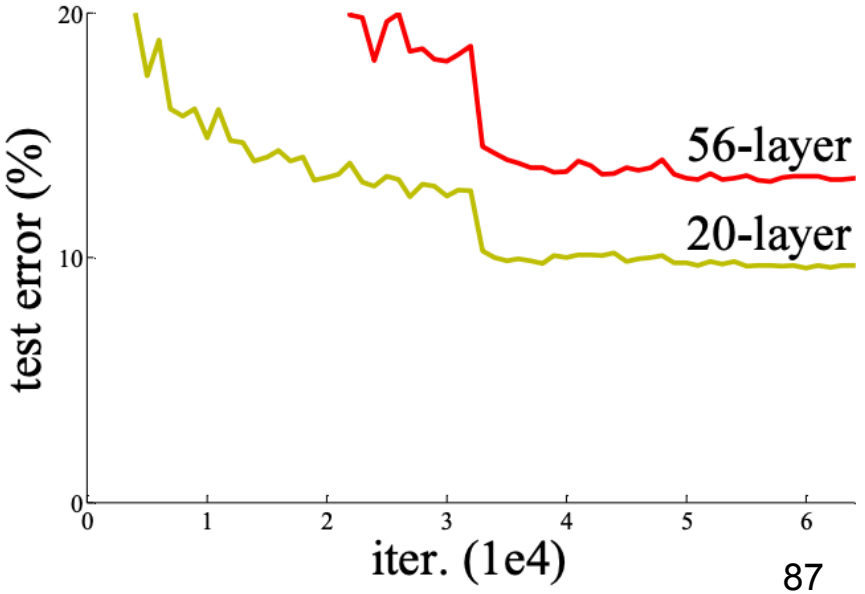


(b) Inception module with dimension reductions

Deeper Models Seems to be Better

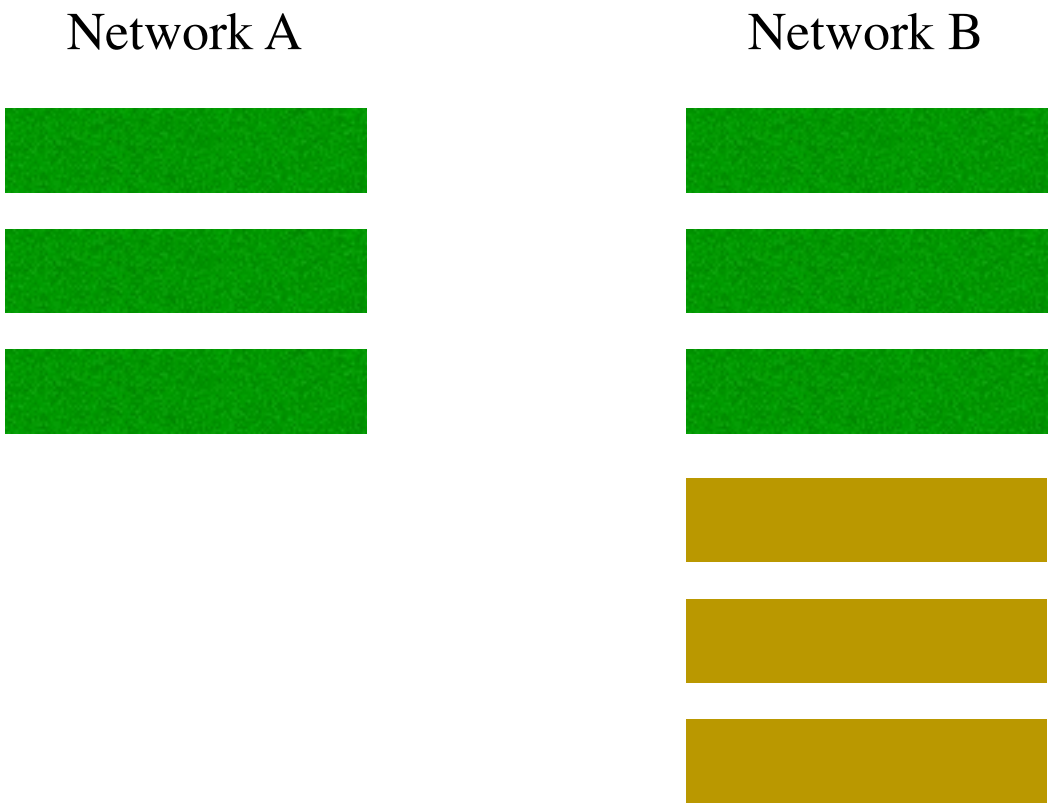
- Simple idea: Just add more layers?
- No! Deeper Neural Networks are hard to train.





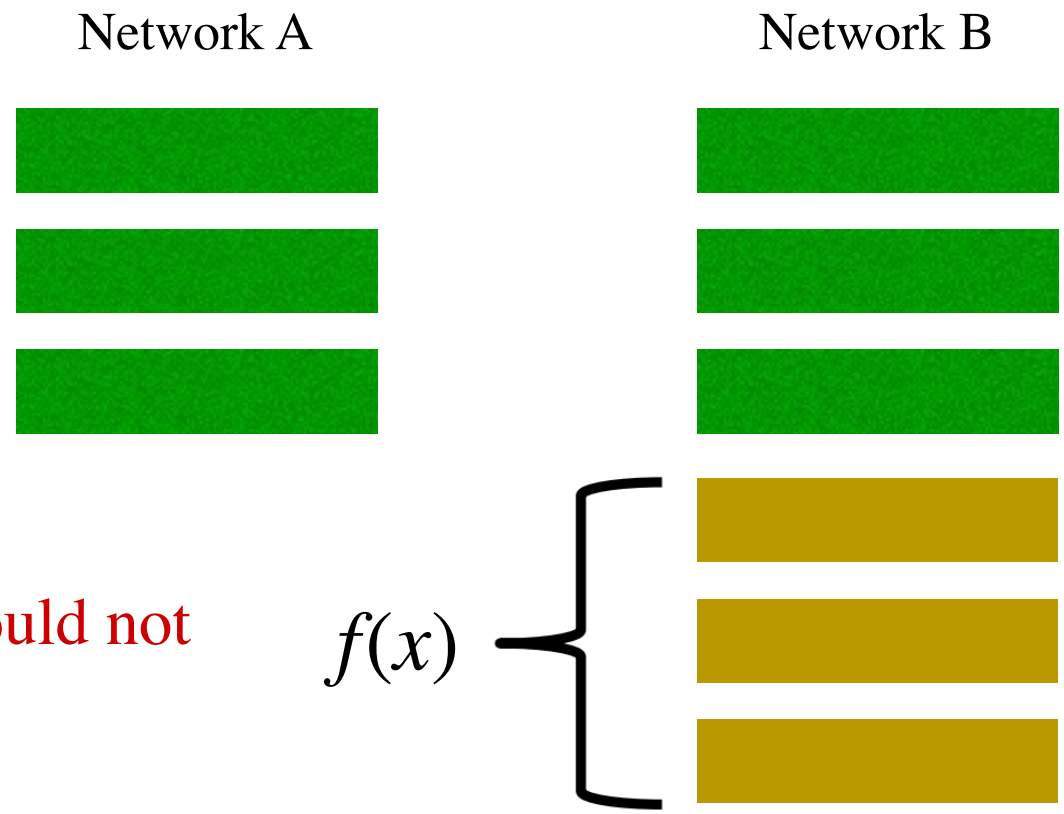
Issues with Depth

- Why would adding layers lead to worse performance?



Issues with Depth

- Why would adding layers lead to worse performance?



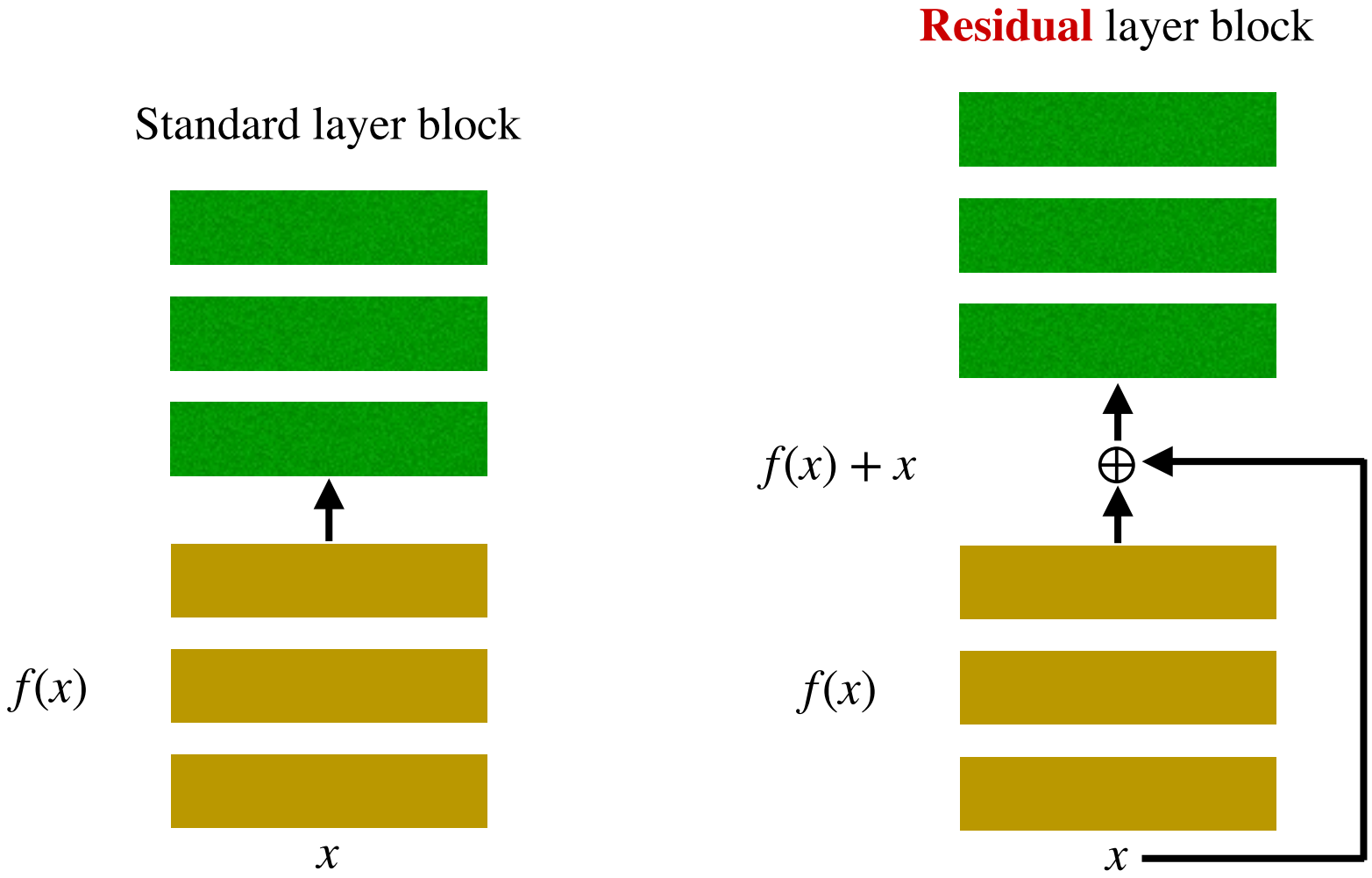
If $f(x) = x$, it should not get worse

Issues with Depth

- Conjecture: Identity mapping can be hard to learn.
- Idea: How about learning a zero mapping?

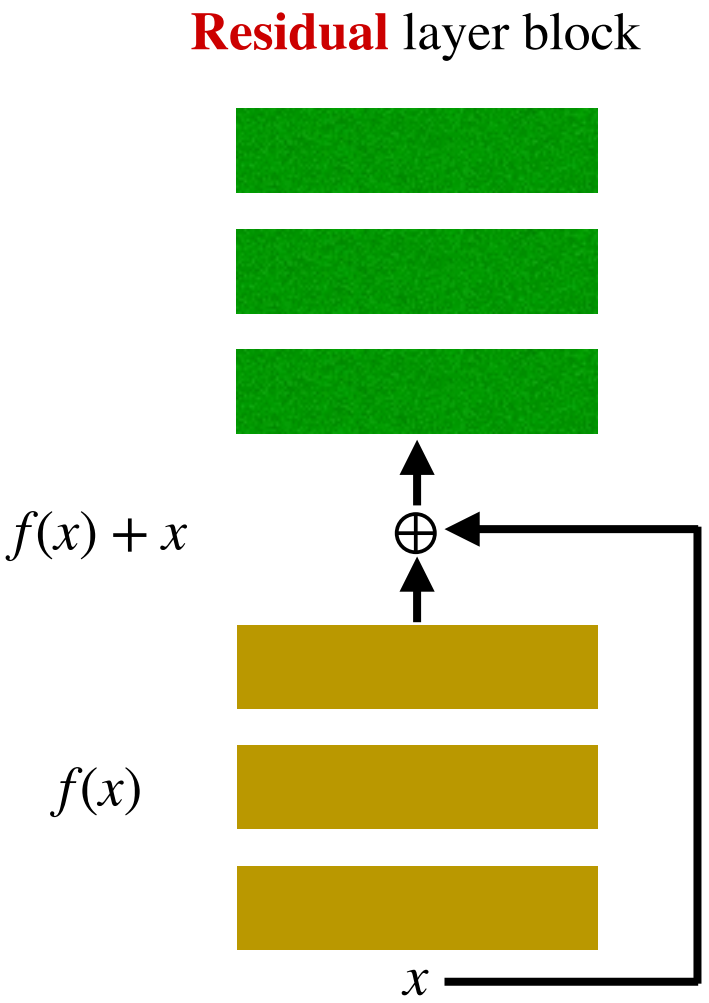
Residual Block

- Idea: How about learning a zero mapping?



Residual Block

- Instead of learning $f(x) = x$, we only need to learn $f(x) = 0$.
- This task is easier since we can simply push all weights to 0.
- No additional parameters needed.

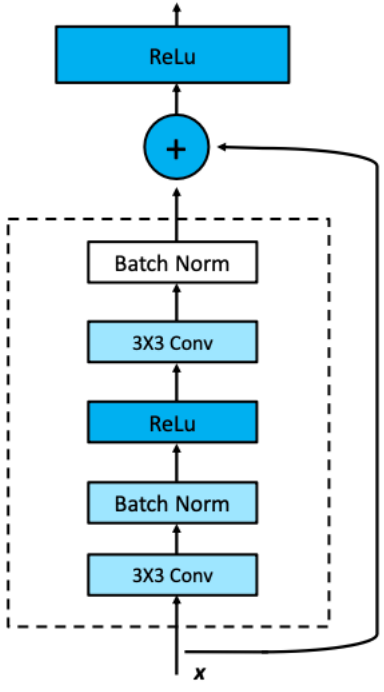


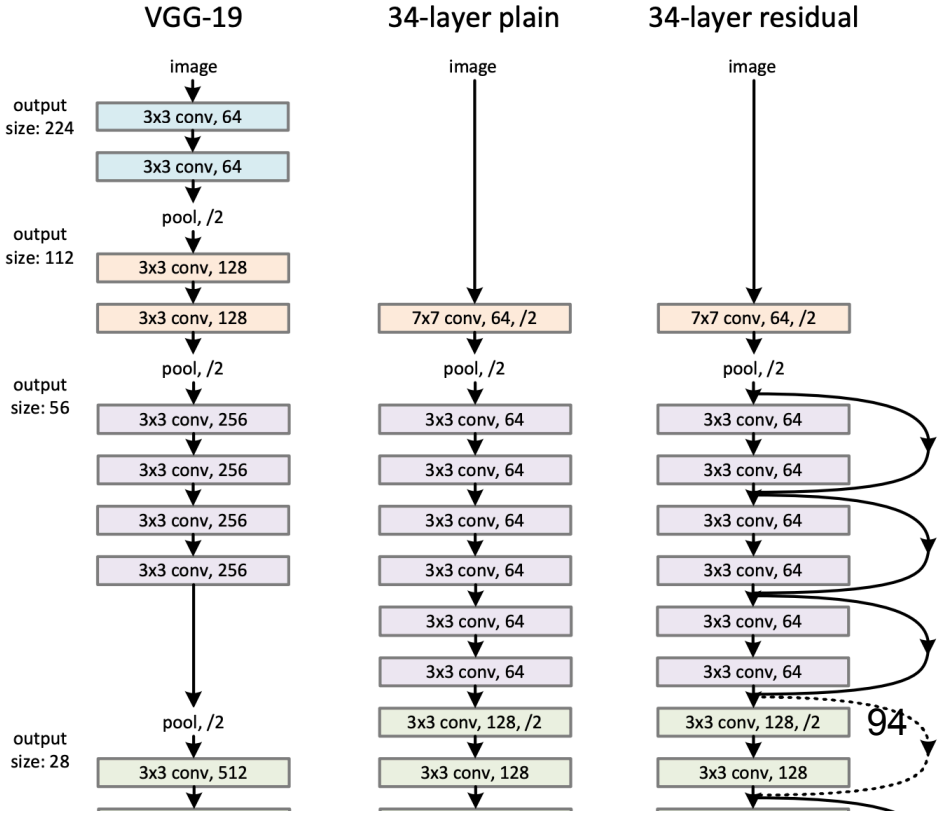
Residual Block

- What if $f(x)$ and x are of different sizes?
  - Option A: add 0 padding/use 1×1 conv
  - Option B: Using projection—$f(x) + Wx$

ResNet Architecture

- Stack of residual blocks
- Periodically double # channels and downsample spatially (e.g., using a stride of 2).





ResNet Architecture

- ResNet with various depth

Repeat 3 times

64 channels

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Training Curves on ImageNet

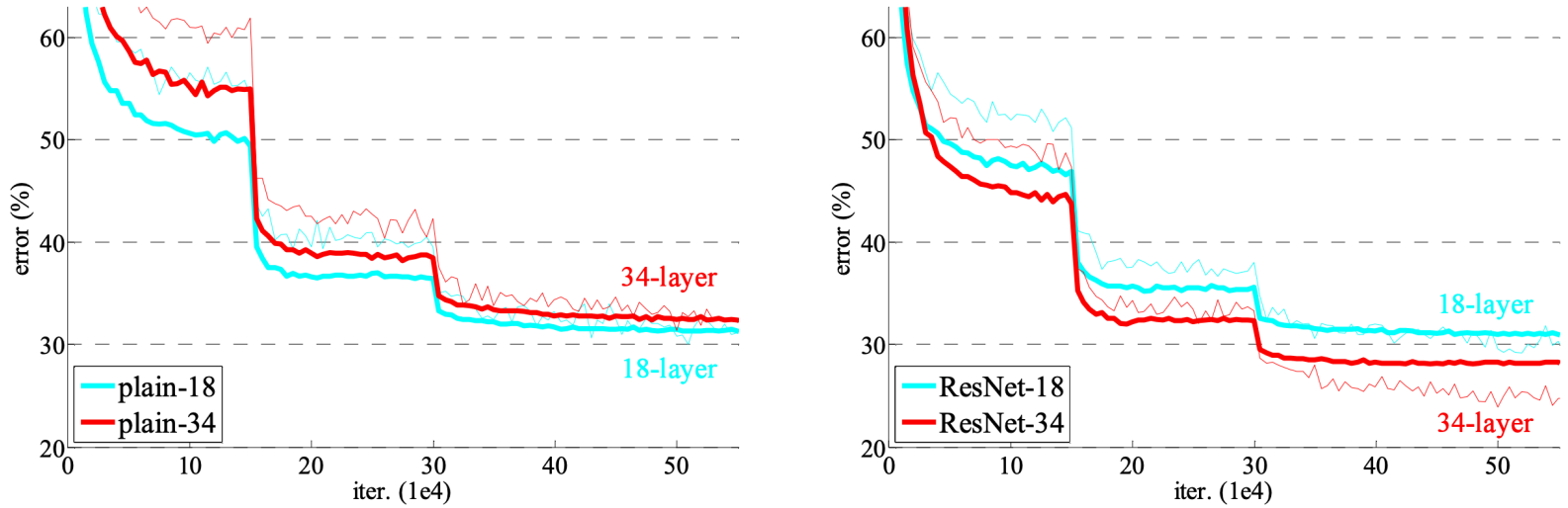


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Test Error on ImageNet

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

provement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1]*, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

CVPR 2016
Best Paper

Acknowledgements

- A lot of content from online resources
  - Yingyu Liang's slides https://www.cs.princeton.edu/courses/archive/spring16/cos495/
  - Goodfellow's slides https://www.deeplearningbook.org/
  - Kevin Murphy’s Book https://probml.github.io/pml-book/book1.html
  - Sharon Li’s slides https://pages.cs.wisc.edu/~sharonli/courses/cs540_spring2022/schedule.html