ML: Lecture 5

Convolutional Neural Networks

Panagiotis Papapetrou, PhD
Professor, Stockholm University

Syllabus

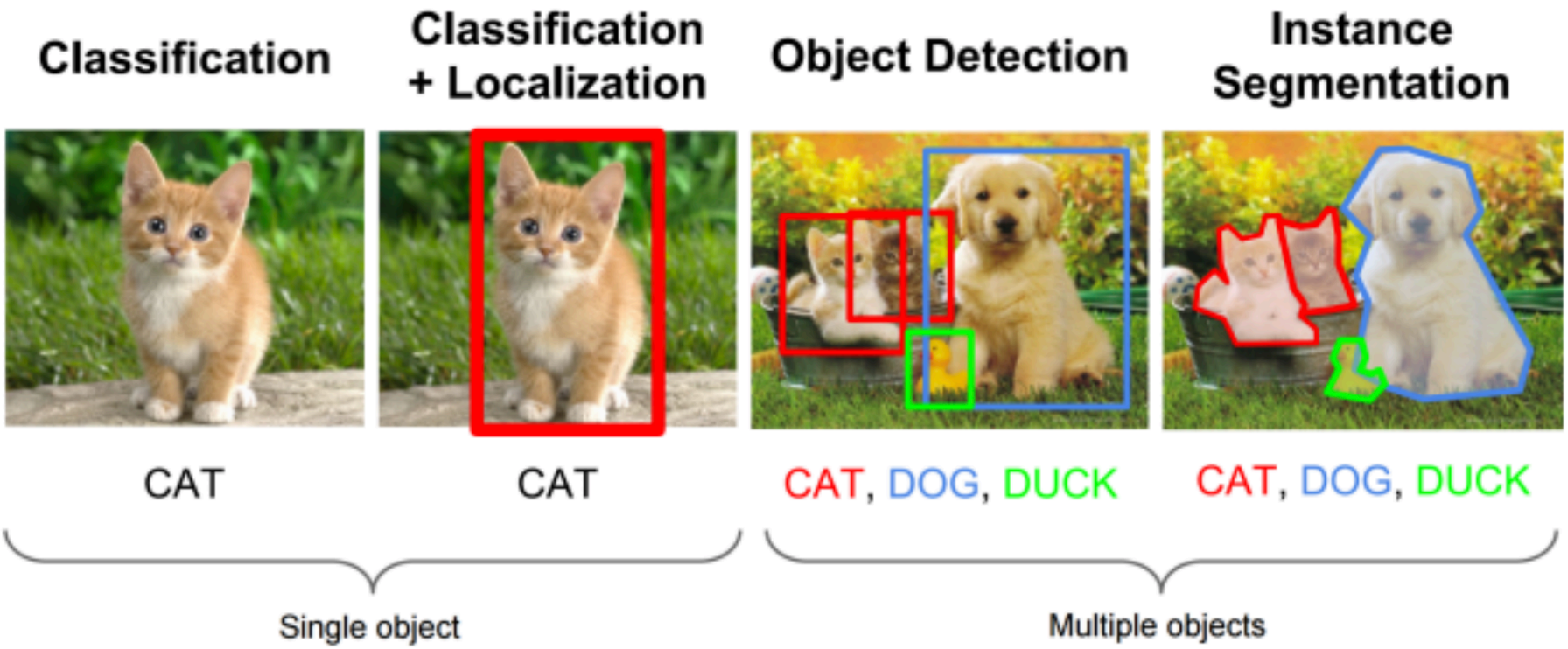
Jan 16	Introduction to machine learning
Jan 18	Regression analysis
Jan 19	Laboratory session 1: numpy and linear regression
Jan 23	Ensemble learning
Jan 25	Deep learning I: Training neural networks
Jan 26	Laboratory session 2: ML pipelines, ensemble learning
Jan 30	**Deep learning II: Convolutional neural networks**
Feb 1	Laboratory session 3: training NNs and tensorflow
Feb 6	Deep learning III: Recurrent neural networks
Feb 8	Laboratory session 4: CNNs and RNs
Feb 13	Deep learning IV: Autoencoders, transformers, and attention
Feb 20	Time series classification

Today

- Why CNNs?
- What are the main applications they are used for?
- What is convolution and what is pooling?
- Common CNN achitectures
- Training and optimization techniques for CNNs models

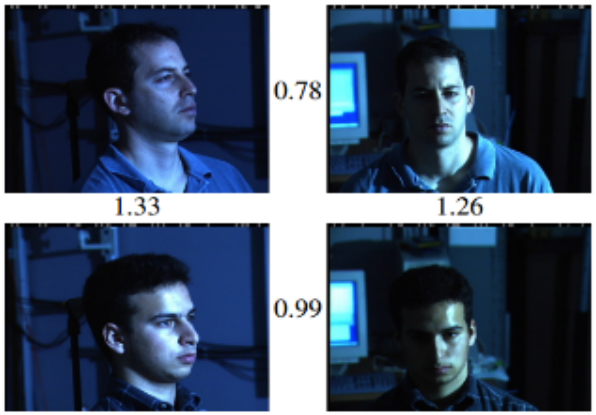
Why CNNs?

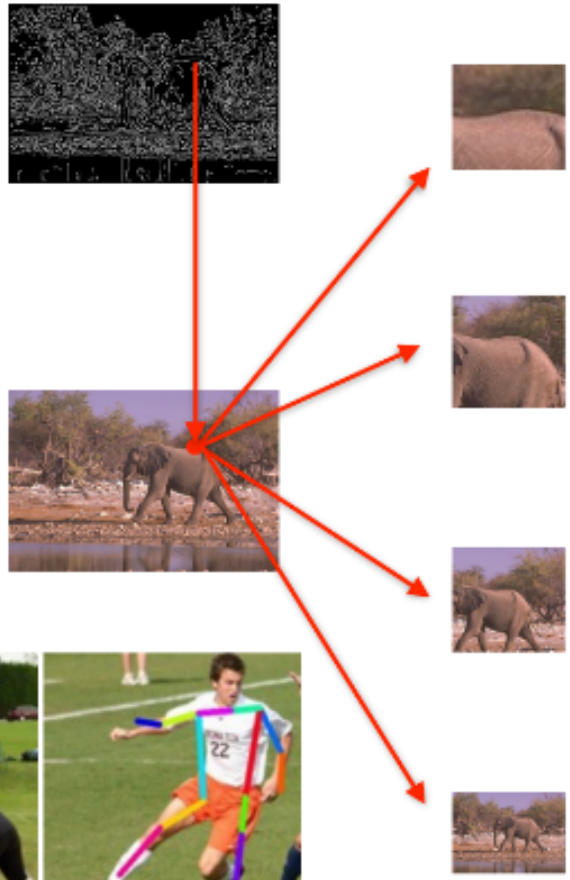
CNN: Convolutional Neural Network



Why CNNs?

- Edge detection
- Face recognition
- Image regression
- Pose estimation
- etc…





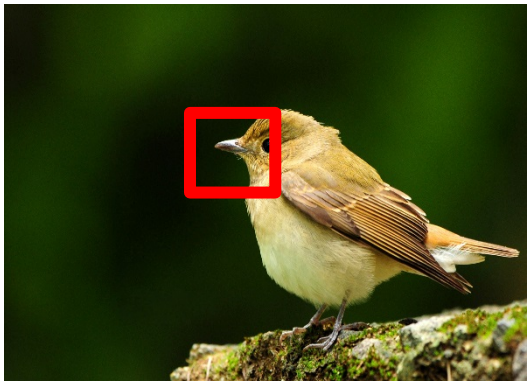
Small pattern detection

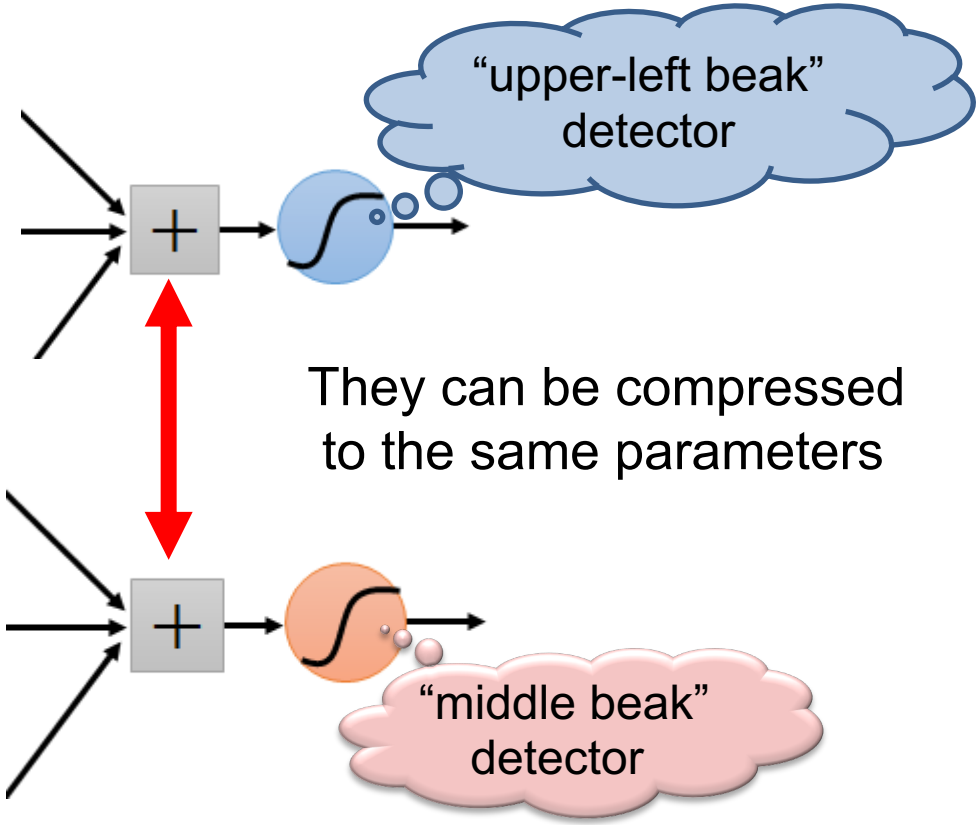
- Some patterns are much smaller than the whole image
- We want to be able to detect them

How can we represent a small region with fewer parameters?

Small pattern compression

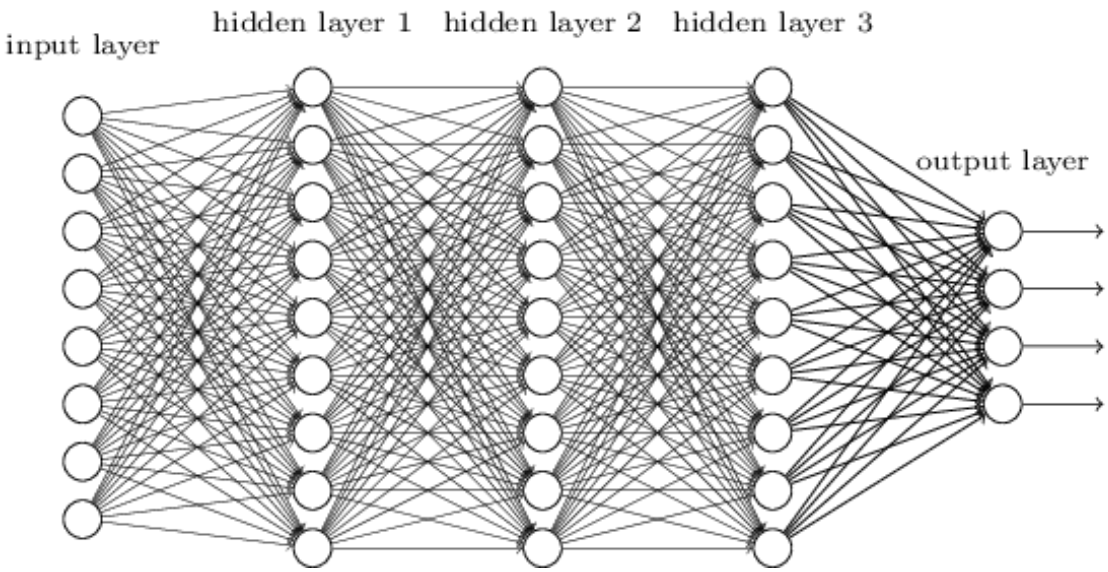
- The same patterns may appear in different places
- They can be compressed!
- What about training a lot of such “small” detectors with each detector “moving around”?





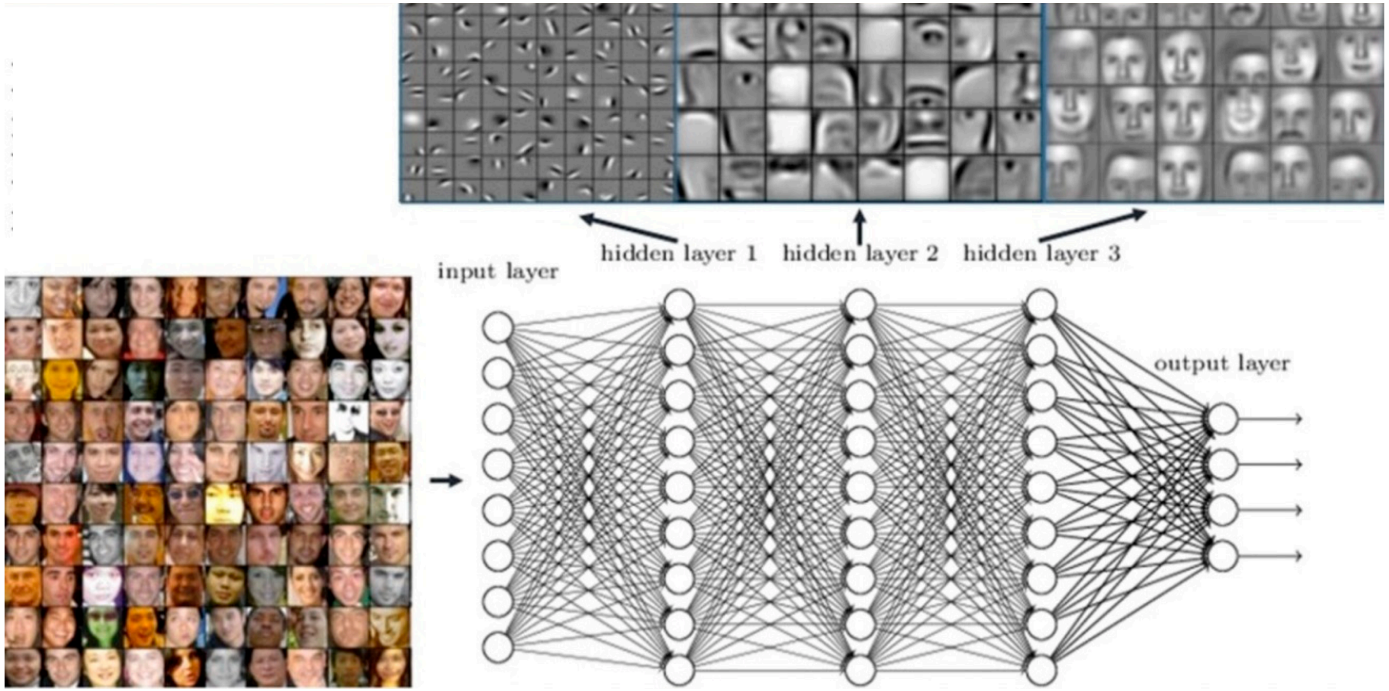
Conceptually

- **CNNs:** regularized versions of multilayer perceptrons
- **Multilayer perceptrons:**
  - *usually* fully-connected networks, each neuron in one layer is connected to all neurons of the next layer
  - "full connectivity" makes them prone to overfitting



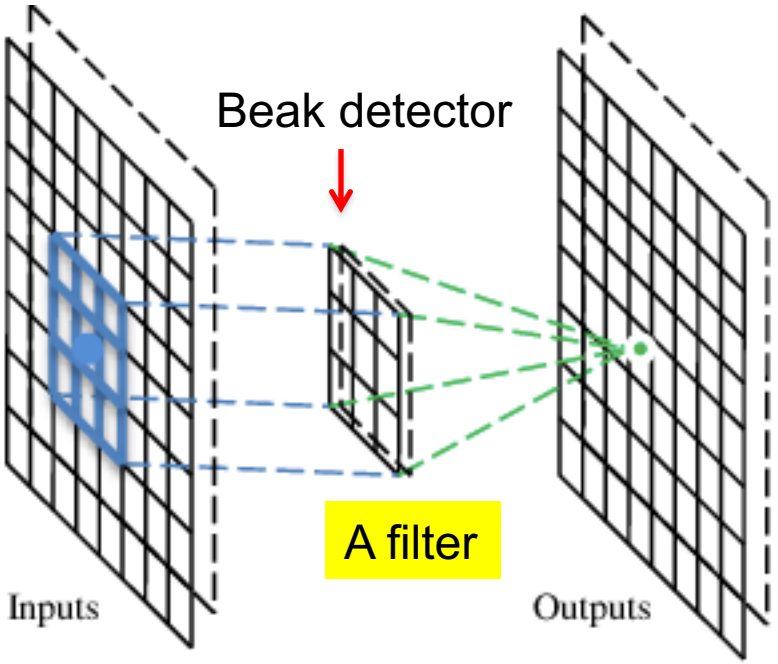
Hierarchical patterns

- They take advantage of local and hierarchical patterns in the data
- Assemble patterns of increasing complexity using smaller and simpler patterns captured by filters



A convolutional layer

- A CNN is a neural network with convolutional layers (and some other layers)
- A convolutional layer has a number of filters that perform the operation of convolution



Convolution (mathematically)

- A mathematical operation between two functions ***f*** and ***g***:
  - describes how the shape of one is modified by the other
  - defined as the integral of the product of the two functions after one is reversed and shifted

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

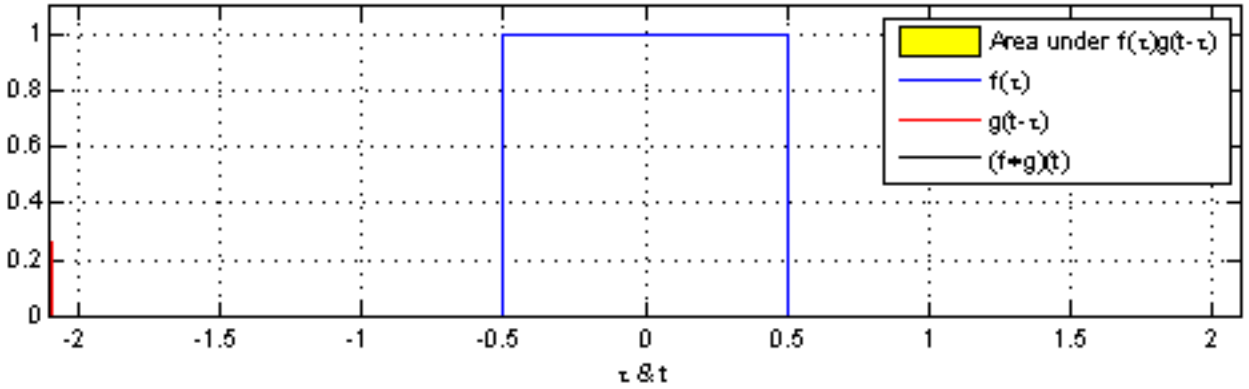
  - the integral is evaluated for all values of shift, producing the convolution function

Convolution (mathematically)

- The area under function $f(\tau)$ weighted by $g(-\tau)$ shifted by t time points

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

- As t changes, the weighting function $g(t - \tau)$ **emphasizes** different parts of the input function $f(\tau)$



The **sum** of an infinite number of copies of ***g()*** (**impulse response** *or* **input**) each shifted by a slightly different time delay (***τ***) and scaled according to the value of the input signal at the value of ***t*** that corresponds to delay ***f(τ)***

Example of a convolution

- Virus outbreak
- Treatment plan: 3 dosages in 1 day
- # of incoming patients per day: **M:**1, **T:**2, **W:**3, **T:**4, **F:**5

```
Plan  *  Patients        = Daily Usage
[3]   *  [1 2 3 4 5]     = [3 6 9 12 15]
```

- Suppose the disease mutates and requires a multi-day treatment, i.e., **day 1:** 3 dosages, **day 2:** 2 dosages, **day 3:** 1 dosage

```
Plan: [3 2 1]
```

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

5	4	3	2	1
		3	2	1

Monday: 1x3 = **3**

Daily total:

3

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

5	4	3	2	1	
			3	2	1

Monday: 1x3 = **3**

Tuesday: 2x3 + 1x2 = **8**

Daily total:

3	8

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

5	4	3	2	1
		3	2	1

Monday: 1x3 = **3**

Tuesday: 2x3 + 1x2 = **8**

Wednesday: 3x3 + 2x2 + 1x1 = **14**

Daily total:

3	8	14

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

5	4	3	2	1
		3	2	1

Monday: 1x3 = **3**

Tuesday: 2x3 + 1x2 = **8**

Wednesday: 3x3 + 2x2 + 1x1 = **14**

Thursday: 4x3 + 3x2 + 2x1 = **20**

Daily total:

3	8	14	20

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

5	4	3	2	1
3	2	1		

Monday: 1x3 = **3**

Tuesday: 2x3 + 1x2 = **8**

Wednesday: 3x3 + 2x2 + 1x1 = **14**

Thursday: 4x3 + 3x2 + 2x1 = **20**

Friday: 5x3 + 4x2 + 3x1 = **26**

Daily total:

3	8	14	20	26

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

		5	4	3	2	1
3	2	1				

Monday: 1x3 = **3**

Tuesday: 2x3 + 1x2 = **8**

Wednesday: 3x3 + 2x2 + 1x1 = **14**

Thursday: 4x3 + 3x2 + 2x1 = **20**

Friday: 5x3 + 4x2 + 3x1 = **26**

Satursday: 5x2 + 4x1 = **14**

Daily total:

3	8	14	20	26	14

Example of a convolution

M	T	W	T	F
1	2	3	4	5

reverse

$g(-\tau)$

5	4	3	2	1

D1	D2	D3
3	2	1

$f(\tau)$

3	2	1

		5	4	3	2	1
3	2	1				

Monday: 1x3 = **3**

Tuesday: 2x3 + 1x2 = **8**

Wednesday: 3x3 + 2x2 + 1x1 = **14**

Thursday: 4x3 + 3x2 + 2x1 = **20**

Friday: 5x3 + 4x2 + 3x1 = **26**

Satursday: 5x2 + 4x1 = **14**

Sunday: 5x1 = **5**

Daily total:

3	8	14	20	26	14	5

Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

These are the network parameters to be learned

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

Filter 1

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

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3)

Convolution

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

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product

6 x 6 image

Convolution

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

Filter 1

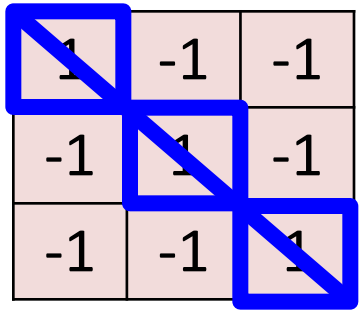
If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

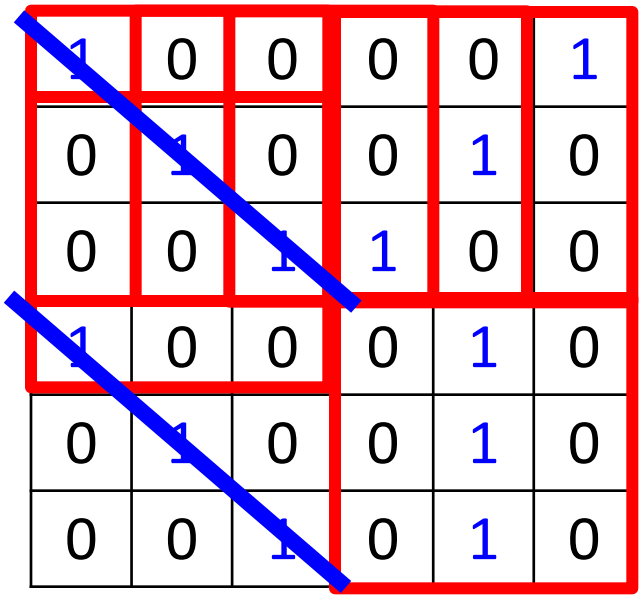
3 -3

Convolution

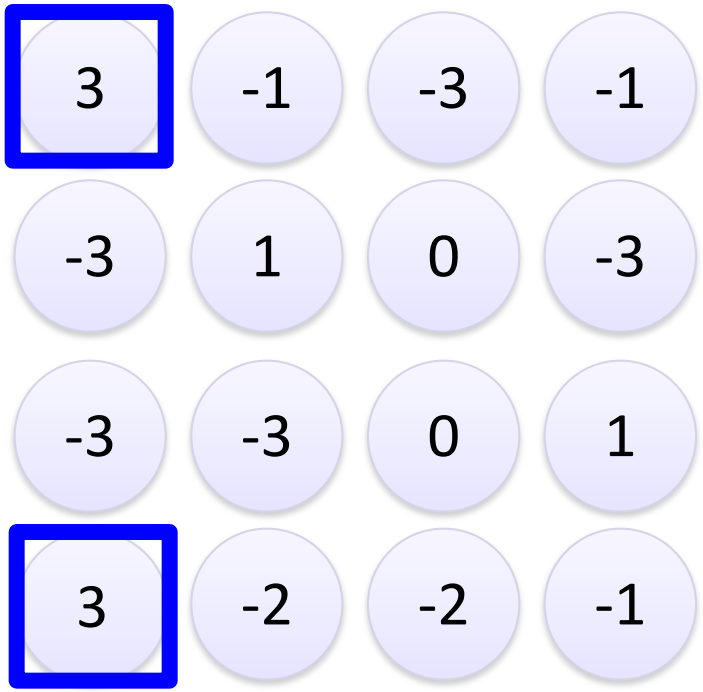


Filter 1

stride=1



6 x 6 image



Convolution

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

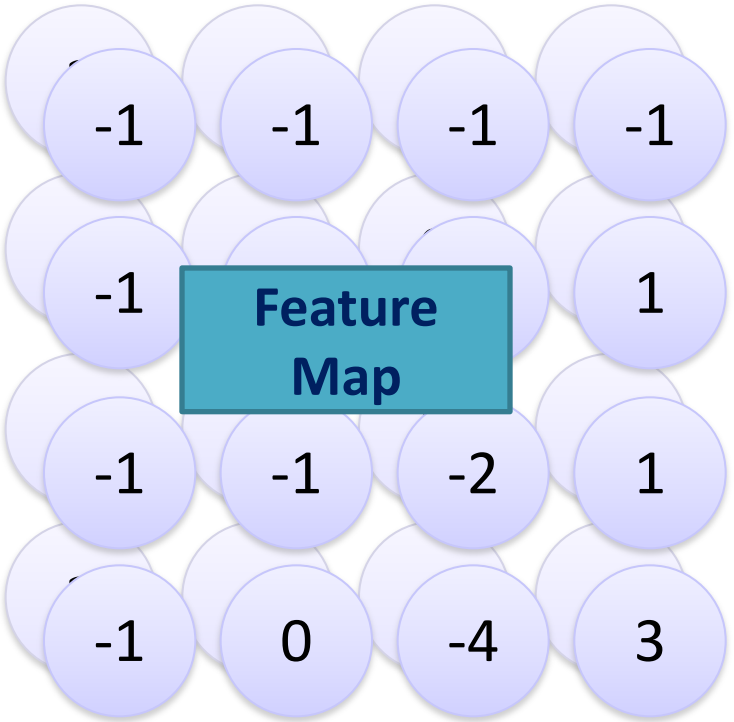
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

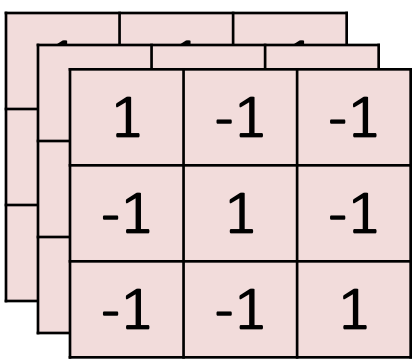
6 x 6 image

Repeat this for each filter

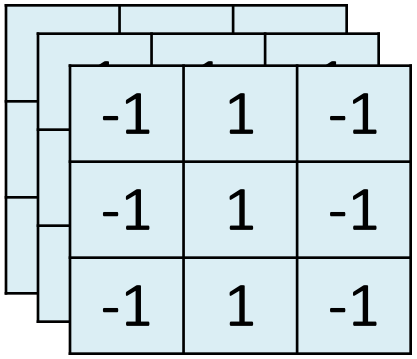


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: 3 **RGB** channels

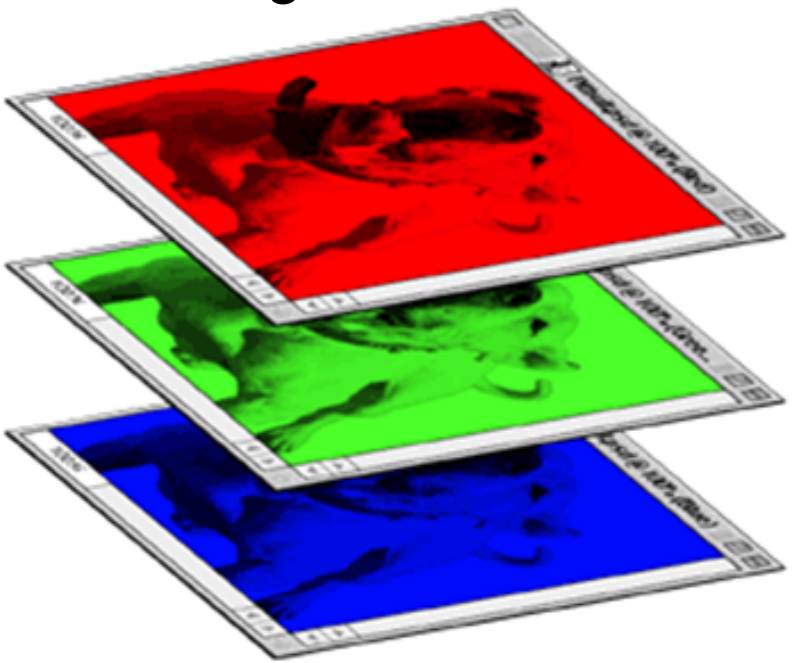


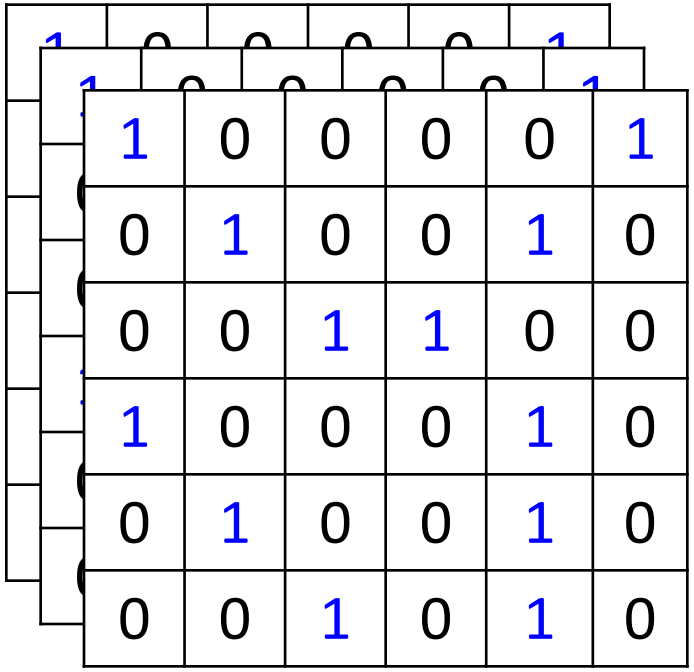
Filter 1



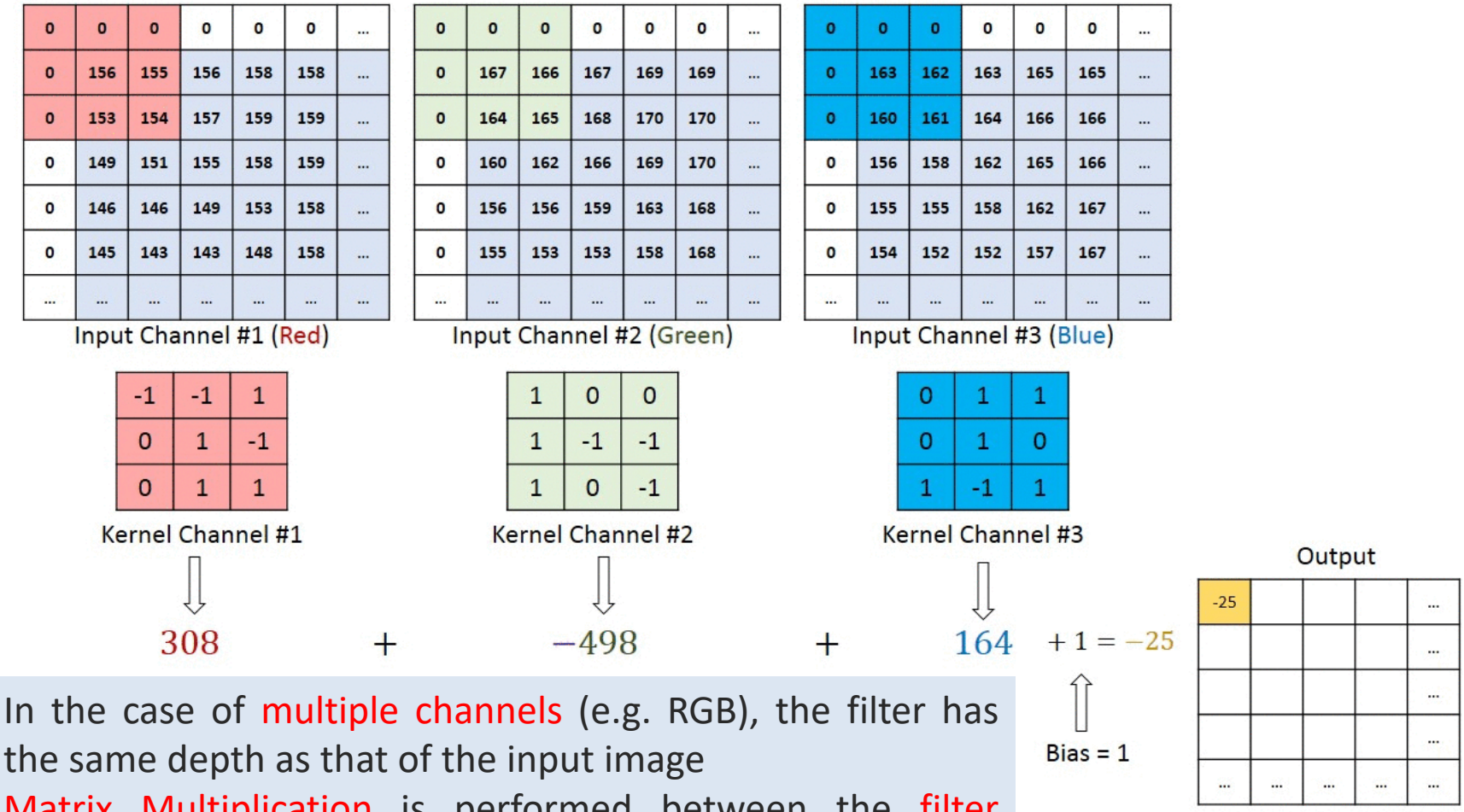
Filter 2

Color image





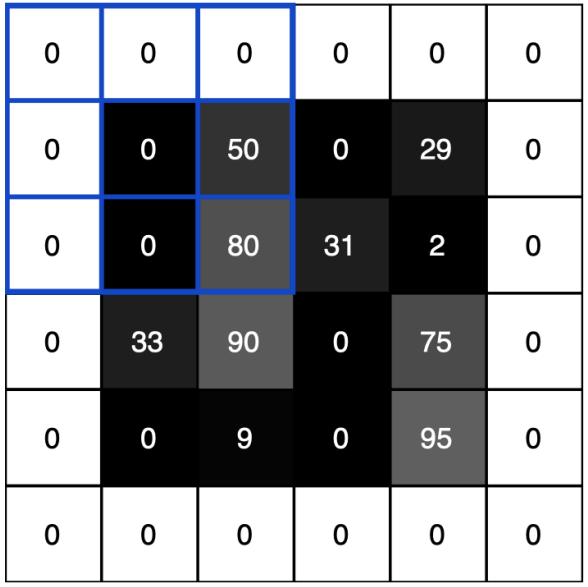
Merging the 3 channels to 1 convolution layer

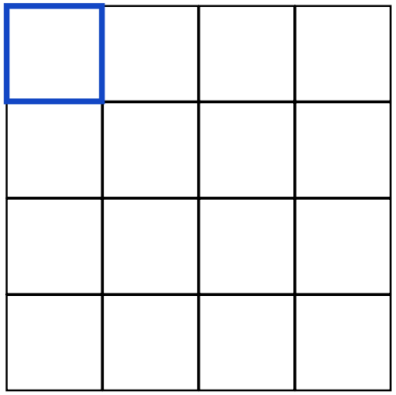


- In the case of multiple channels (e.g. RGB), the filter has the same depth as that of the input image
- Matrix Multiplication is performed between the filter (Kernel) channel and the input channel
- The results are summed with the bias to give us a “squashed” single channel

Padding

- Padding: adding zeros to the original image
- **Input:** 4x4 image, filter: 3x3
- **Padding:** 1 pixel
- **Output:** 4x4 image

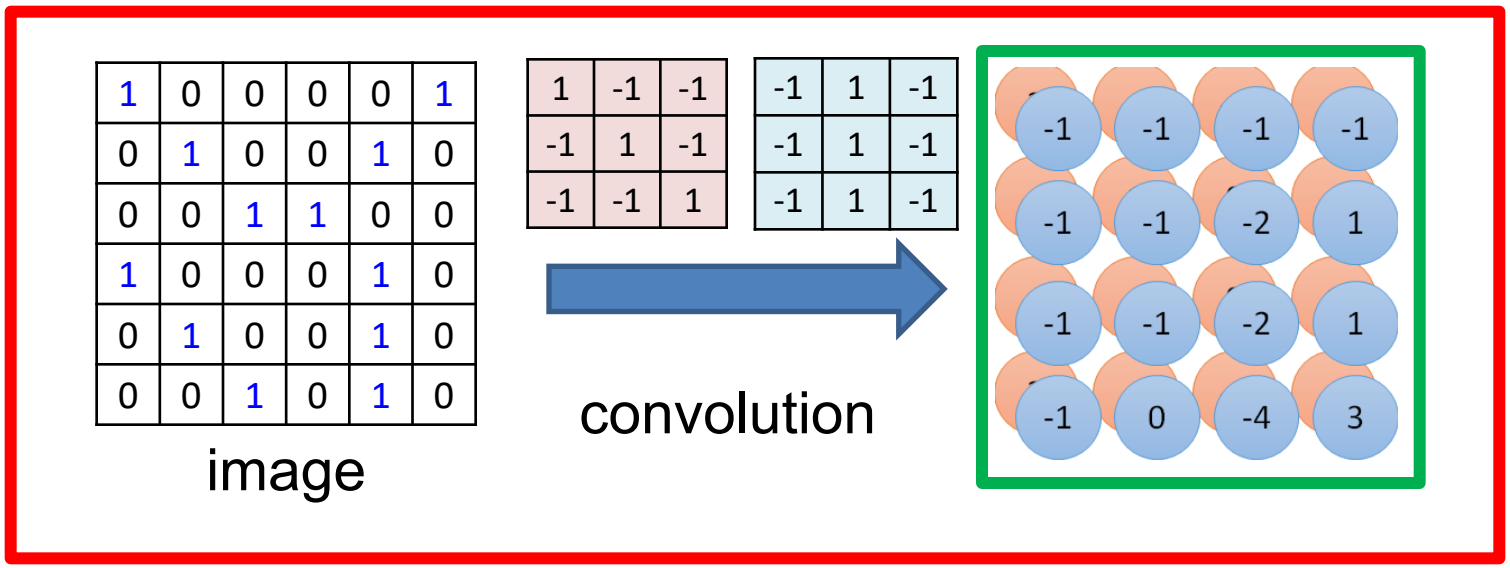




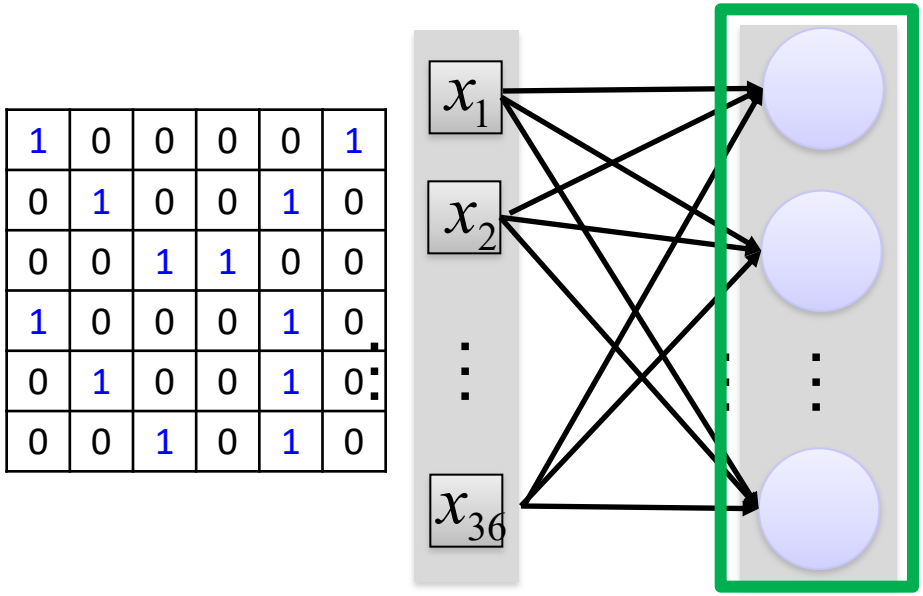
- This is done so that the border pixels are not undervalued (lost)
- *padding = filter dimension – 1*
- *padding/2* rows/columns per side

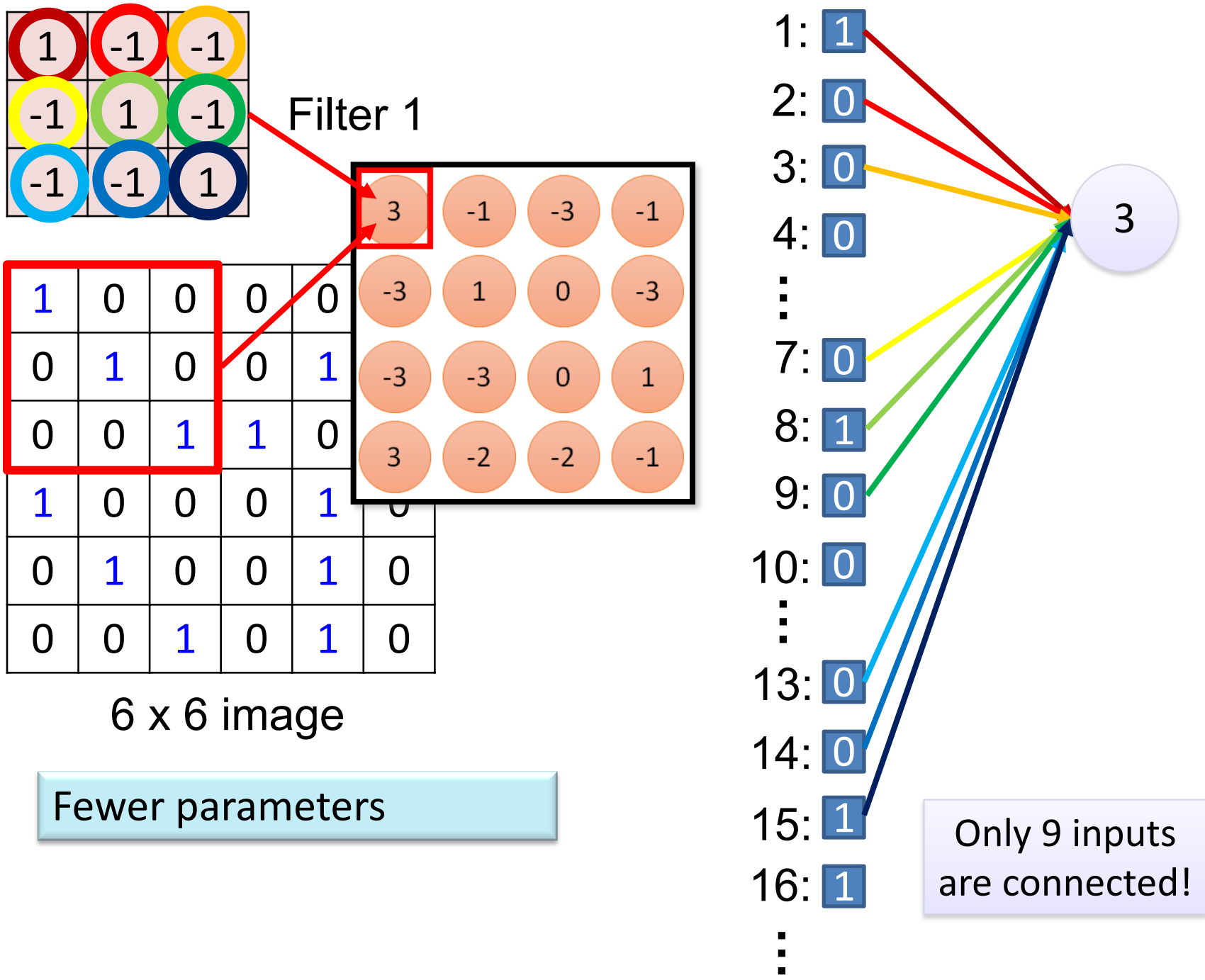
- CNN filters are typically of odd dimensionality
- Padding preserves the original dimensionality

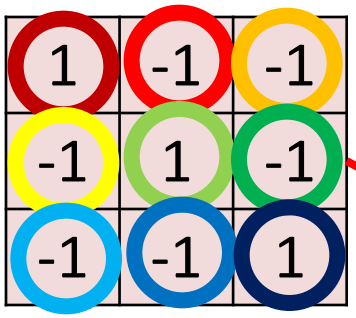
Convolution v.s. Fully Connected



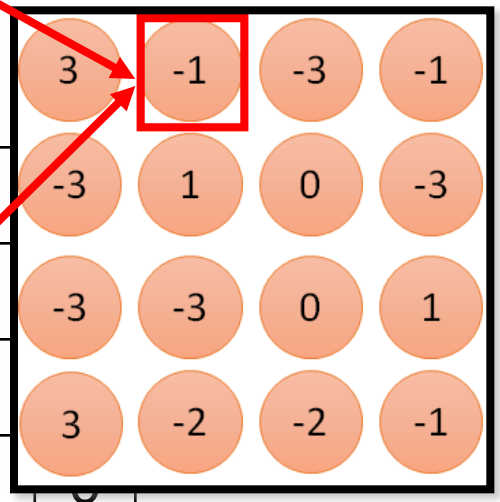
Fully-connected




1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Fewer parameters
1:
2:
3:
4:
7:
8:
9:
10:
13:
14:
15:
16:
3
Only 9 inputs are connected!



Filter 1

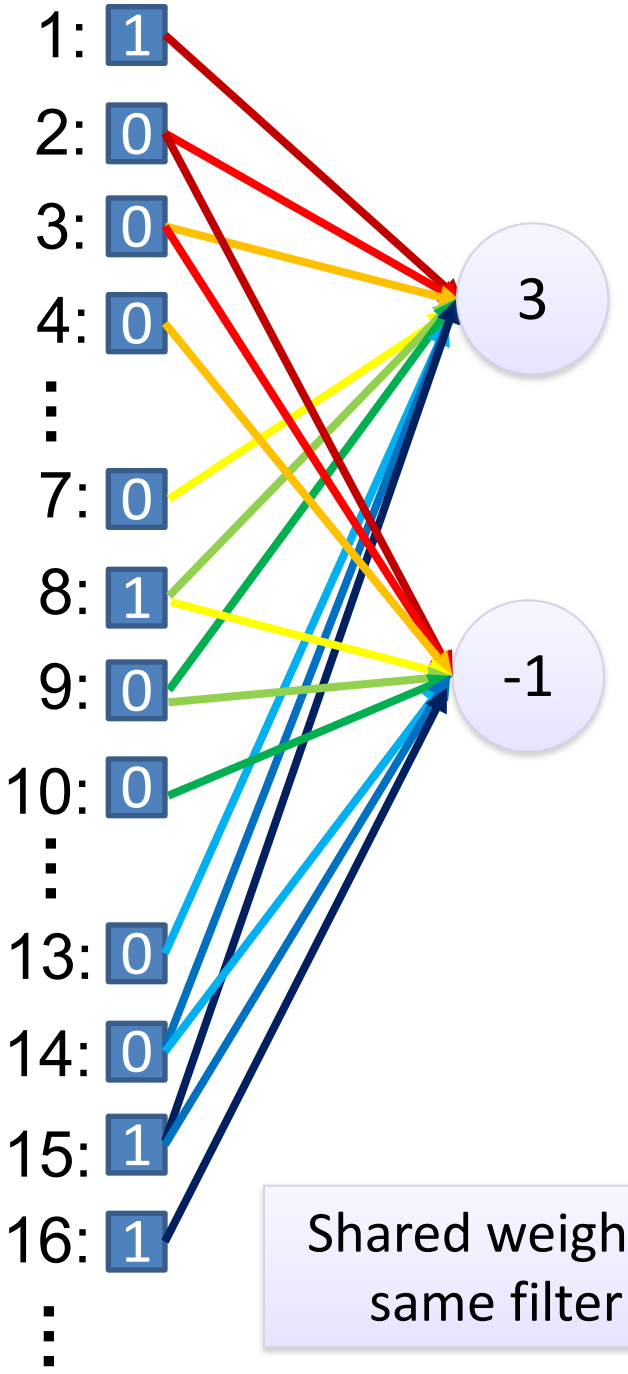


1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Fewer parameters

Even fewer parameters

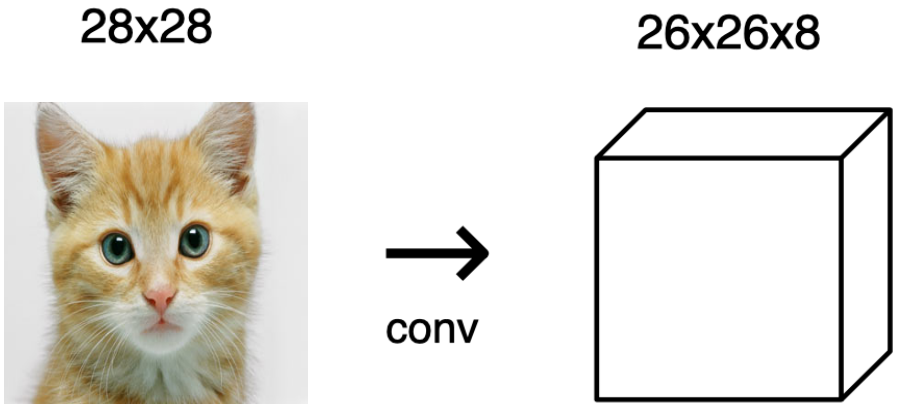


Shared weights, same filter

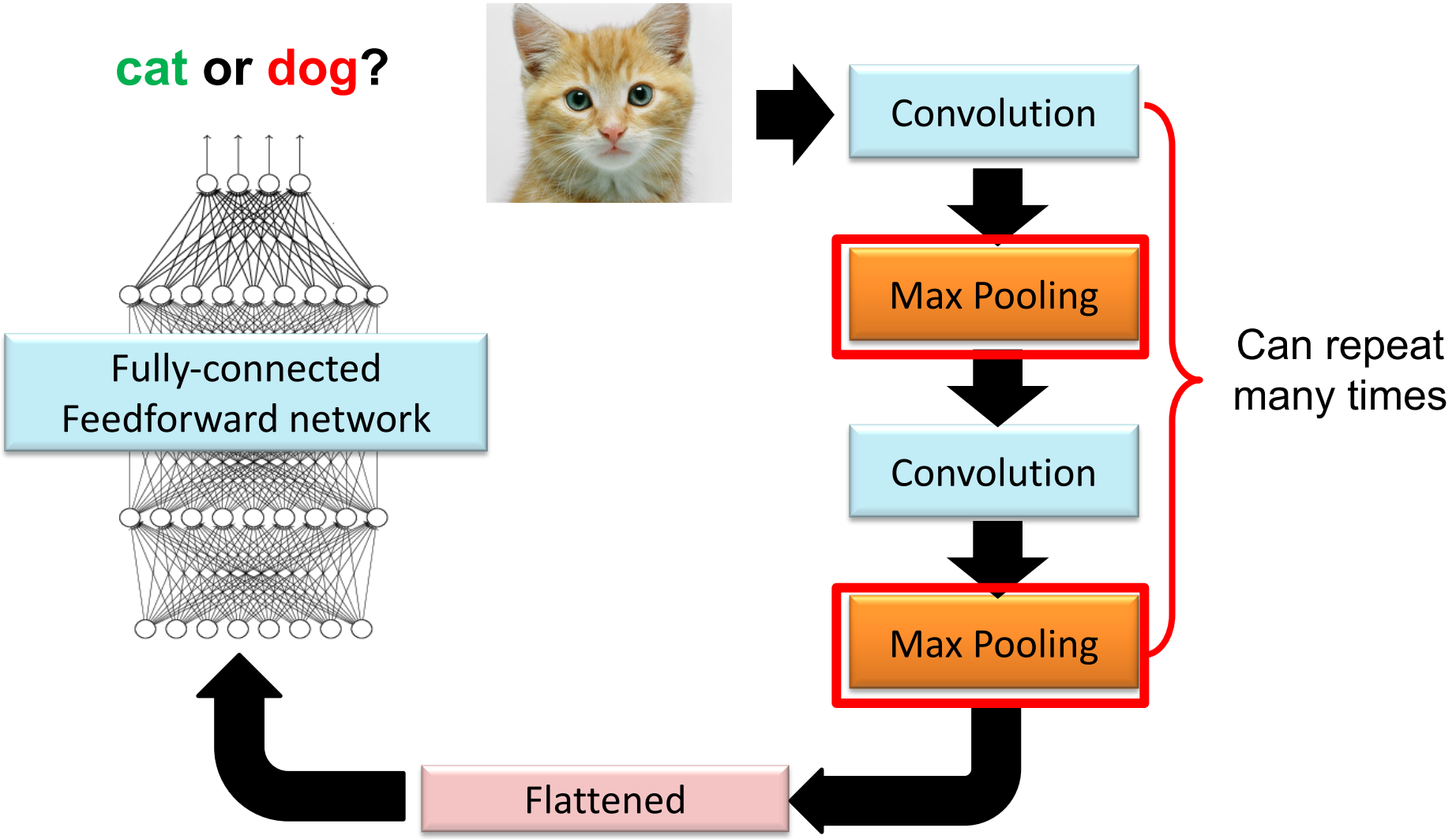
Numerical example

- Input image: 28 x 28
- Convolution layer: 8 filters, 3 x 3 each
- Output image: 26 x 26 (no padding)
- Stacked together: 26 x 26 x 8
- How many weights?

3 × 3 (filter size) × 8 (number of filters) = **only 72 weights**



The whole CNN



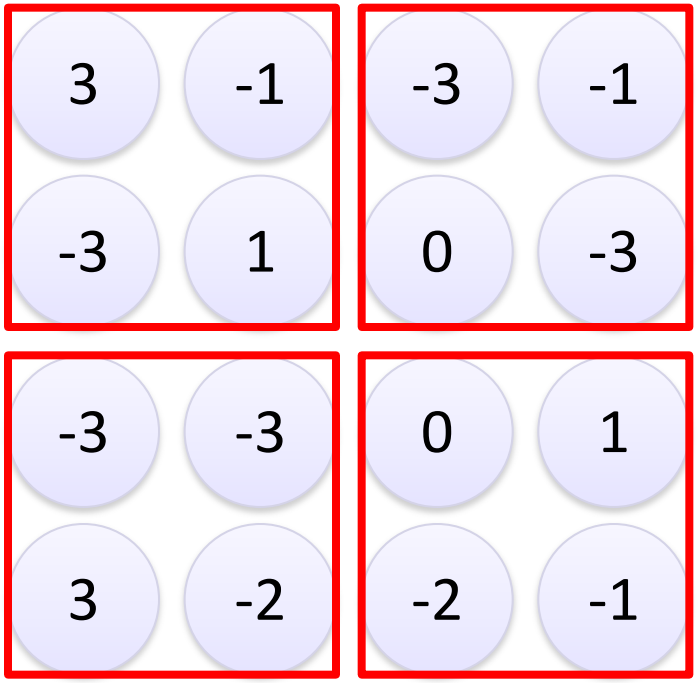
Max Pooling (2x2)

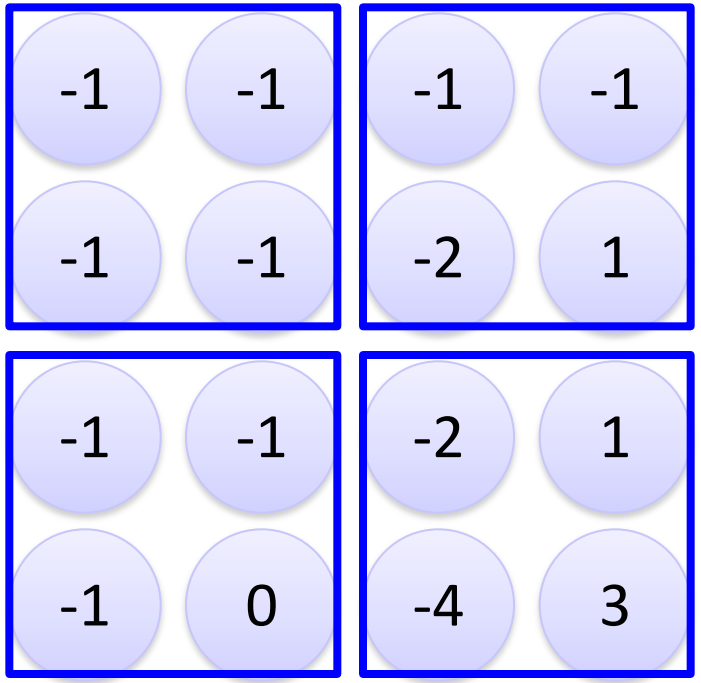
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

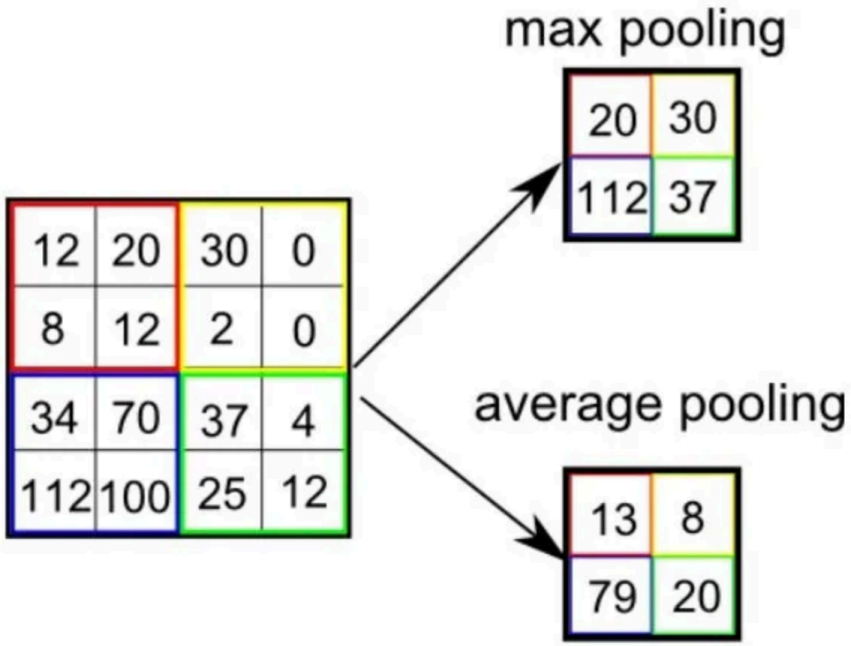
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Average Pooling (2x2)



Max pooling: takes the **max** value in a 2x2 (non-overlapping) window

Average pooling: takes the **average** in a 2x2 (non-overlapping) window

Min – max – average pooling?

- **Dark background:** max pooling selects the brightest pixels
- **Light background:** min pooling selects the darkest features
- **Image smoothing:** average pooling

Why Pooling?

- Subsampling pixels will not change the object

bird

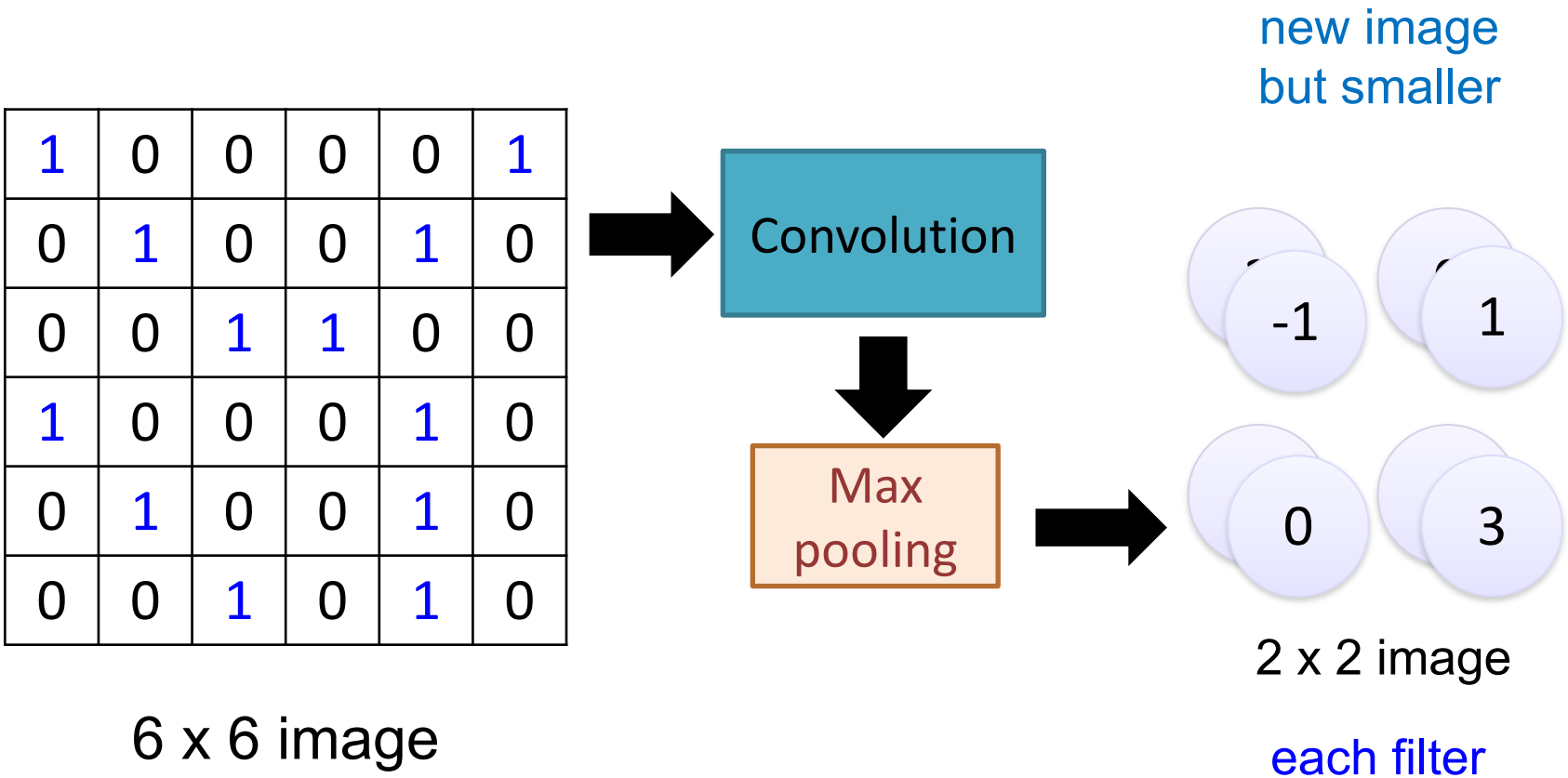
subsampling

bird

We can subsample the pixels to make image smaller

fewer parameters to characterize the image

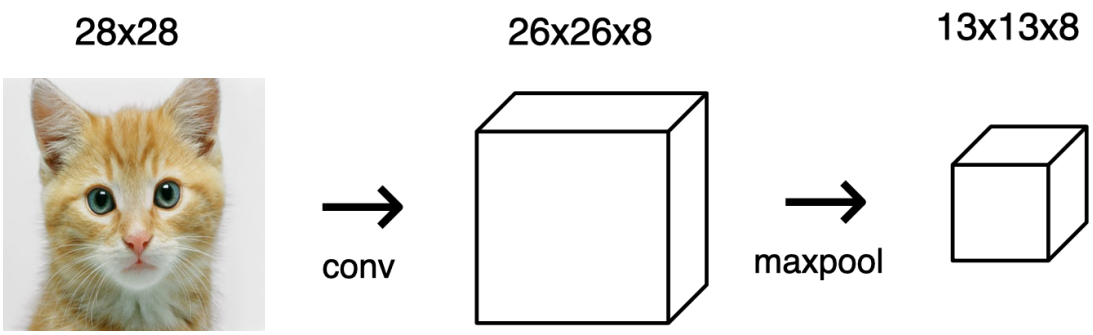
Hence…



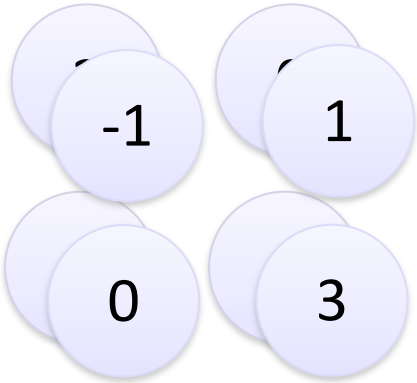
- Reduced number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

Reduction by pooling

- Input image: 28 x 28
- Convolution layer: 8 filters, 3 x 3
- Output image: 26 x 26
- Stacked together: 26 x 26 x 8
- Weights: 72
- Image reduction: 13 x 13 x 8



The whole CNN



Convolution

Max Pooling

Convolution

Max Pooling

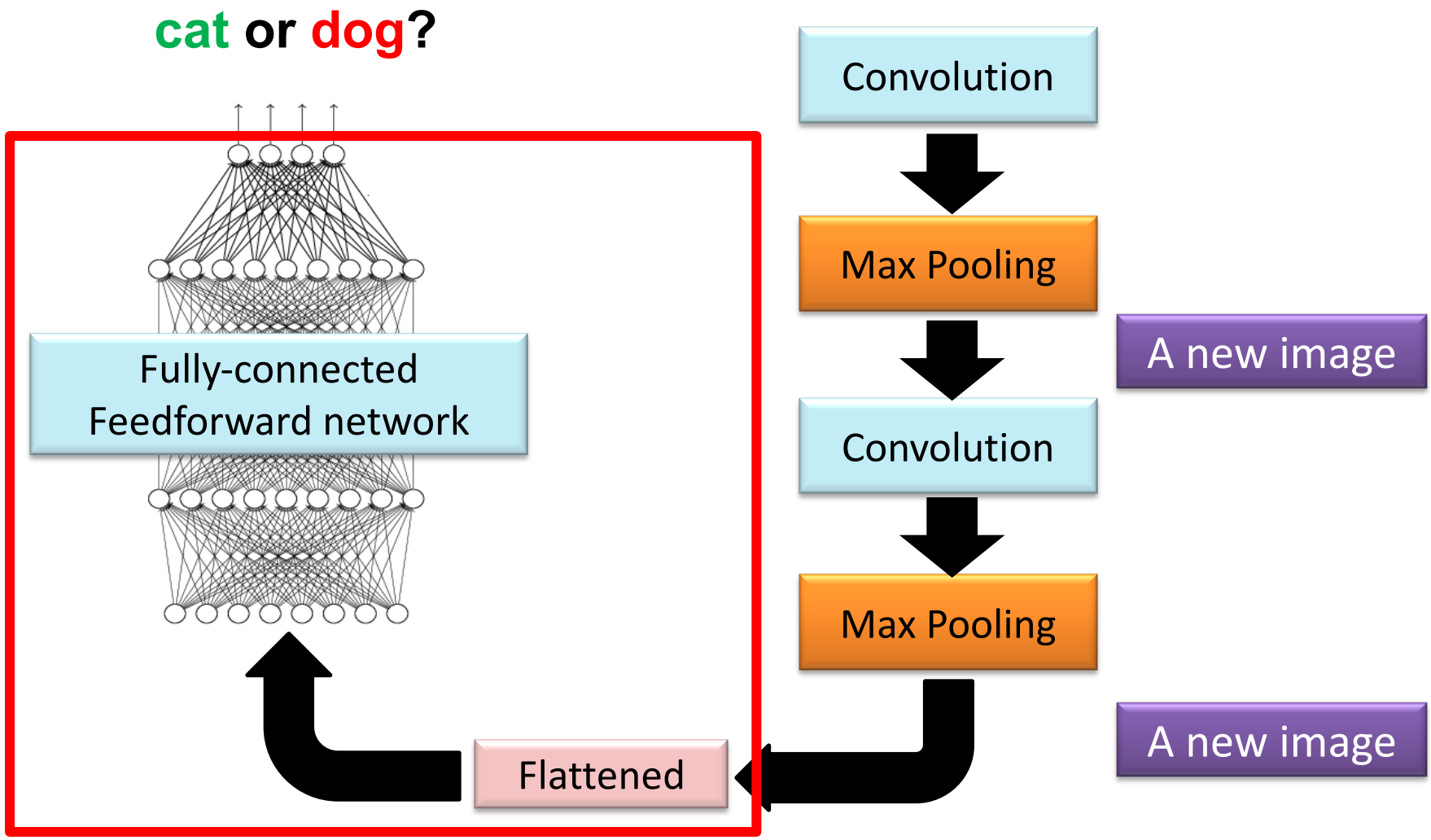
can repeat many times

A new image

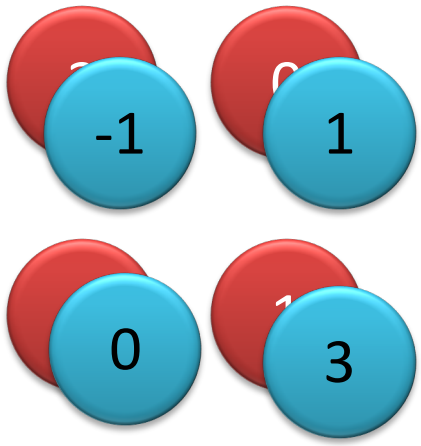
smaller than the original image

the number of channels is the number of filters

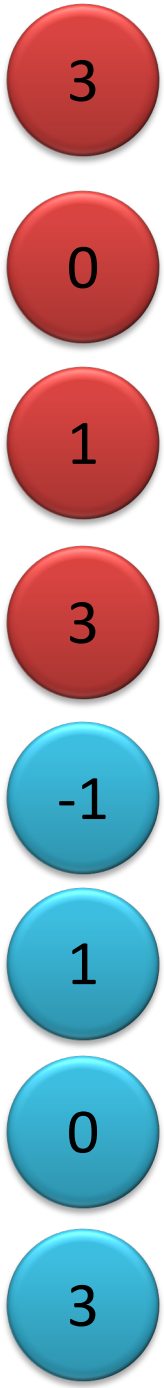
The whole CNN

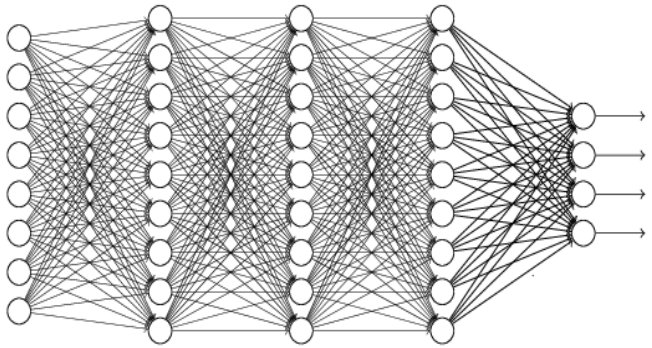


Flattening



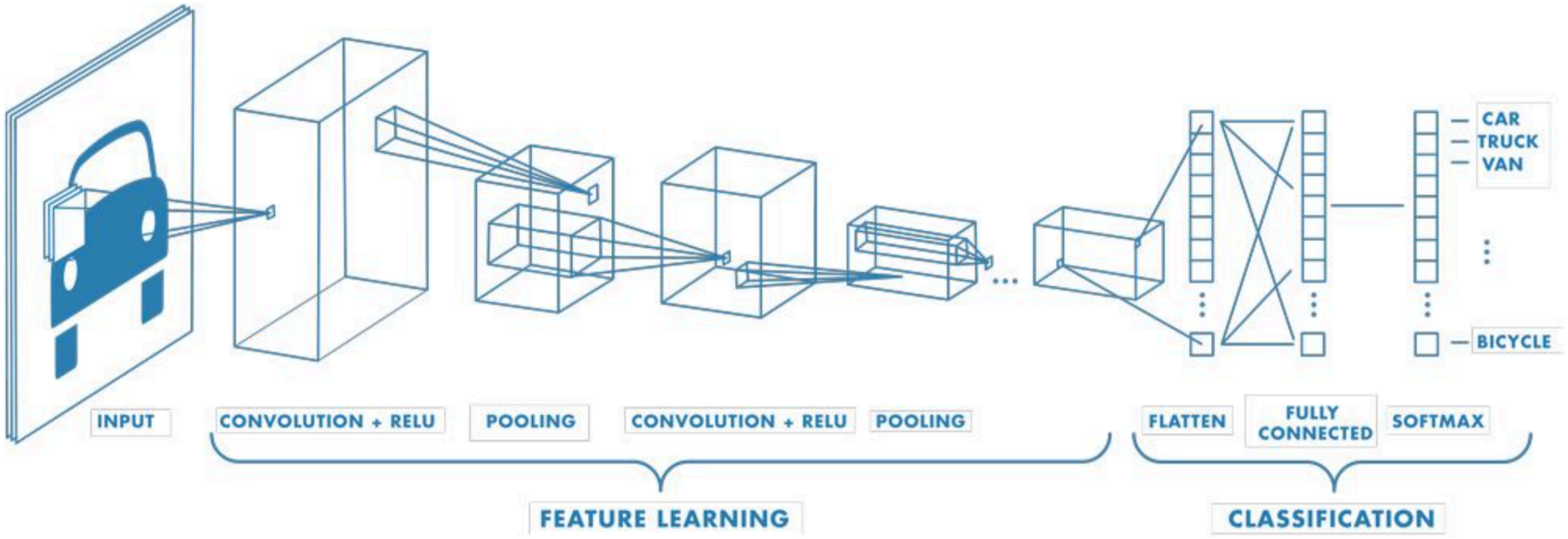
Flattened





Fully-connected
Feedforward network

Example: vehicle image classification



- ReLU is applied after each convolution: faster and more effective training by *mapping negative values to zero* and *maintaining positive values*
- Final layer: *softmax on the previous layer, yielding the* probability of the ouptut being of a particular class

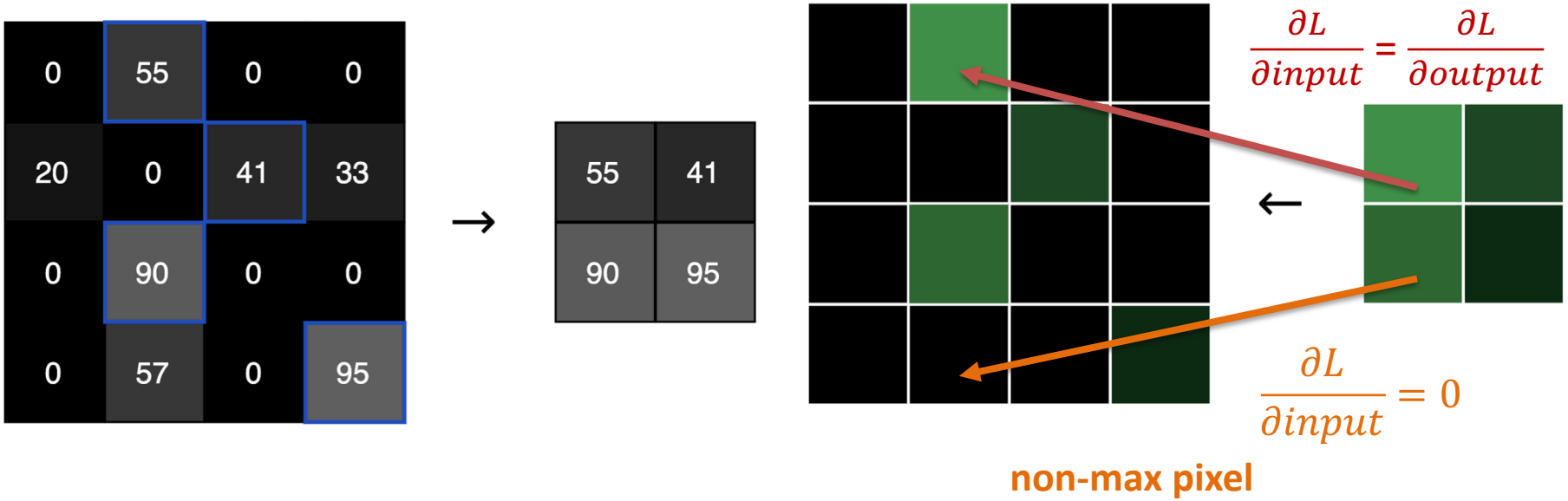
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Training a CNN: two steps

- **Forward phase:** the input is passed through the network
  - each layer will **cache** any input and intermediate values needed for the backward phase
- **Backward phase:** gradients are backpropagated and weights are updated
  - each layer will **receive a gradient** and also **return a gradient**
  - it will receive the gradient loss with respect to its *outputs* $\frac{\partial L}{\partial output}$
  - and return the gradient loss with respect to its *inputs* $\frac{\partial L}{\partial input}$
- **Cross entropy** loss is used as the loss function

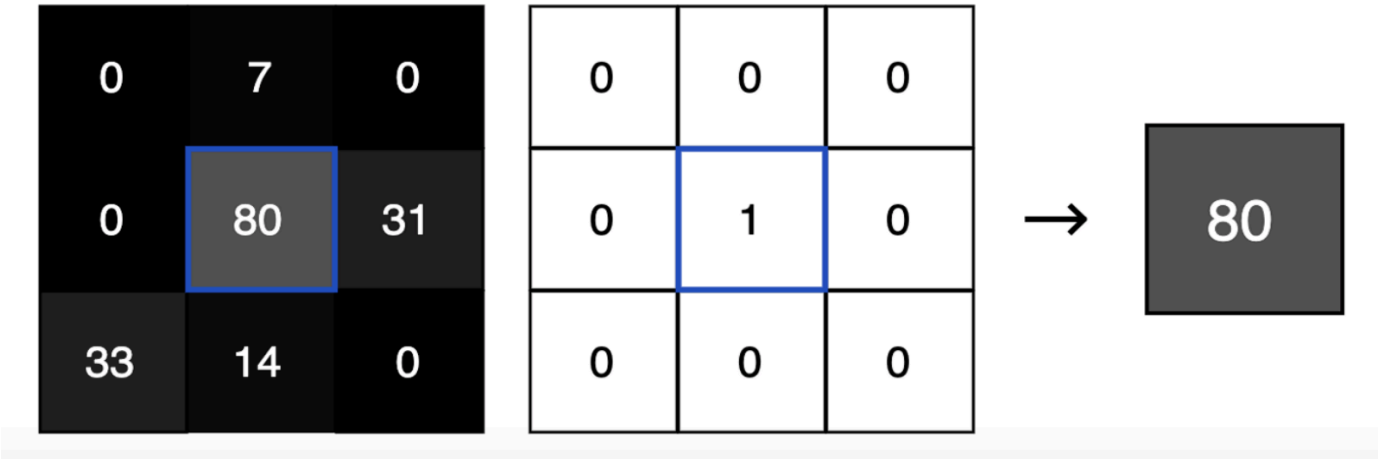
Training a CNN: max pooling

- A Max Pooling layer cannot be trained as it does not have any weights
- **How to propagate the gradients?**
  - **double the width & height** of the gradient by assigning each value to **where the original max value was** in the corresponding pooled block



Training a CNN: convolution

- **Changing filter weights affects the *entire* output image** for that filter
- *Every* output pixel uses *every* pixel weight during convolution
- Increasing any of the filter weights by 1 would increase the output by the value of the corresponding image pixel



- This suggests that the derivative of a specific output pixel (i, j) with respect to a specific filter weight is just the corresponding image pixel value

Training a CNN: convolution

$$\text{out(i, j)} = \text{convolve(image, filter)}$$

$$= \sum_{x=1}^{3} \sum_{y=1}^{3} \text{image}(i+x, j+y) * \text{filter}(x, y)$$

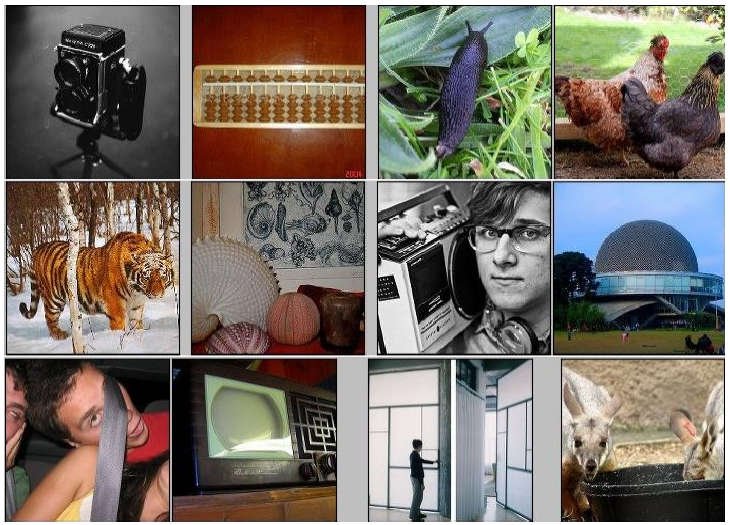
$$\frac{\partial \text{out}(i,j)}{\partial \text{filter}(x,y)} = \text{image}(i+x, j+y)$$

Hence, the **loss gradient** for specific filter weights is:

$$\frac{\partial L}{\partial \text{filter}(x,y)} = \sum_i \sum_j \frac{\partial L}{\partial \text{out}(i,j)} * \frac{\partial \text{out}(i,j)}{\partial \text{filter}(x,y)}$$

Application: ImageNet

IMAGENET

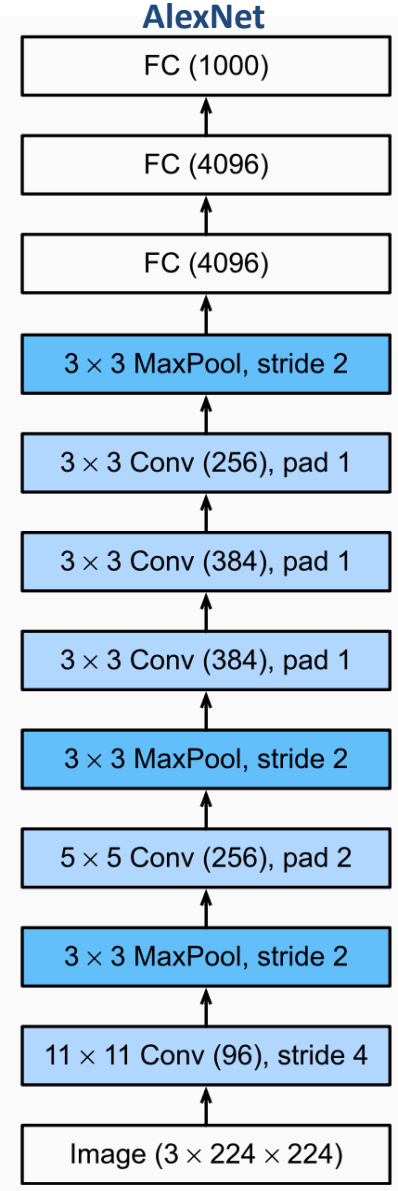


[Deng et al. CVPR 2009]

- ~14 million labeled images with 20k classes
- In at least one million of the images, bounding boxes are also provided
- Images gathered from the Internet
- Human labels via Amazon Turk
- Since 2010, an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes

AlexNet [Krizhevsky 2012]

- Seminal work that introduced CNNs to computer vision and set some standards for neural networks
  - 7 hidden layers, 650,000 units, **60 million** params ReLU activation function
  - GPU implementation: 50x speedup over CPU, trained on two GPUs
  - Local response normalization (lateral inhibition)
  - Overlapping pooling (Modified max-pooling)
  - Dropout regularization and ReLU activation



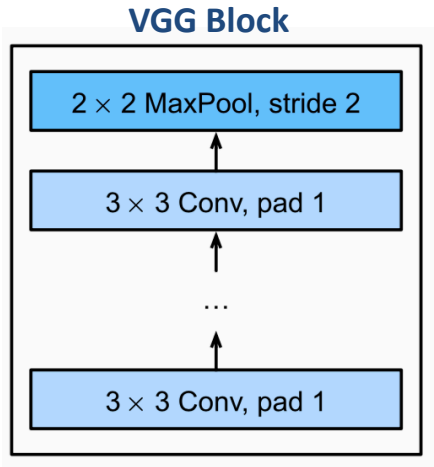
Lateral inhibition: refers to the capacity of a very active neuron to overshadow its neighbors:

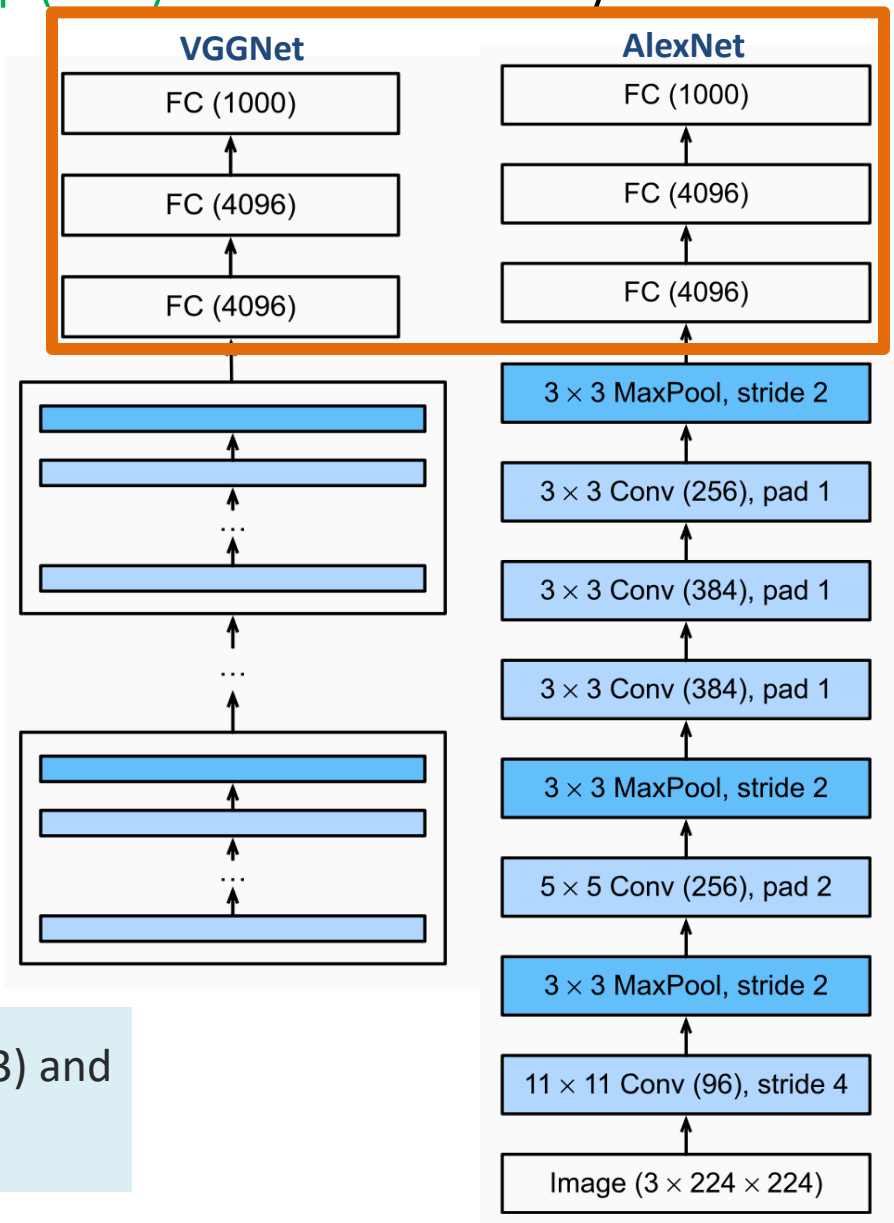
- **Inter-Channel LRN:** Normalize within the same channel: **2D neighborhood** of dimensionality **n x n (square normalization)**
- **Intra-Channel LRN:** Normalize across channels: **1D neighborhood** along the third dimension, **n x 1 x 1 (square normalization)**

Both methods amplify *excited neurons* while dampening the surrounding ones

VGGNet [Simonyan 2014]

- Introduced by the Visual Geometry Group (VGG) at Oxford University
  - Idea of using blocks
  - 11, 16 or 19 hidden layers
  - Uses stacked 3x3 convolutions (2-3 per block)
  - Fully connected layers part same as AlexNet
  - **144 million parameters**





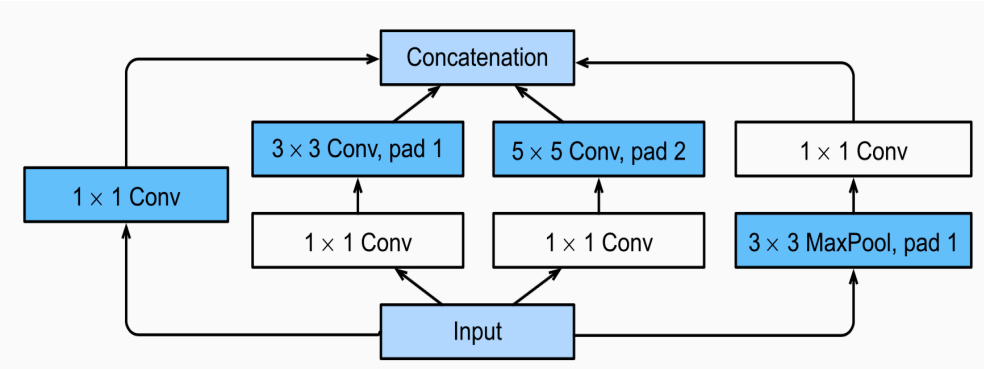
OBS: Using small convolution filter sizes (3x3) and increasing the depth improves performance

GoogleNet [Szegedy 2014]

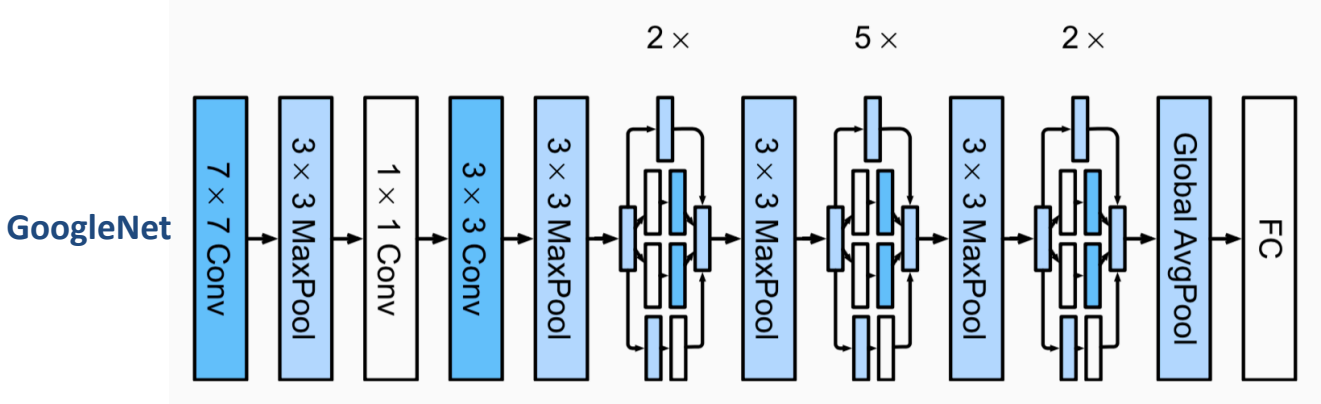
- **GoogleNet (or InceptionV1):** uses sparse structures instead of VGG-style fully connected network architectures (27+ layers, **4 million parameters**)

The inception module:

- the output of an inception module concatenates features detected by convolution filters of multiple sizes (1x1, 3x3, 5x5)
- different types of filters will be prioritized according to the relative position of the layer in the network

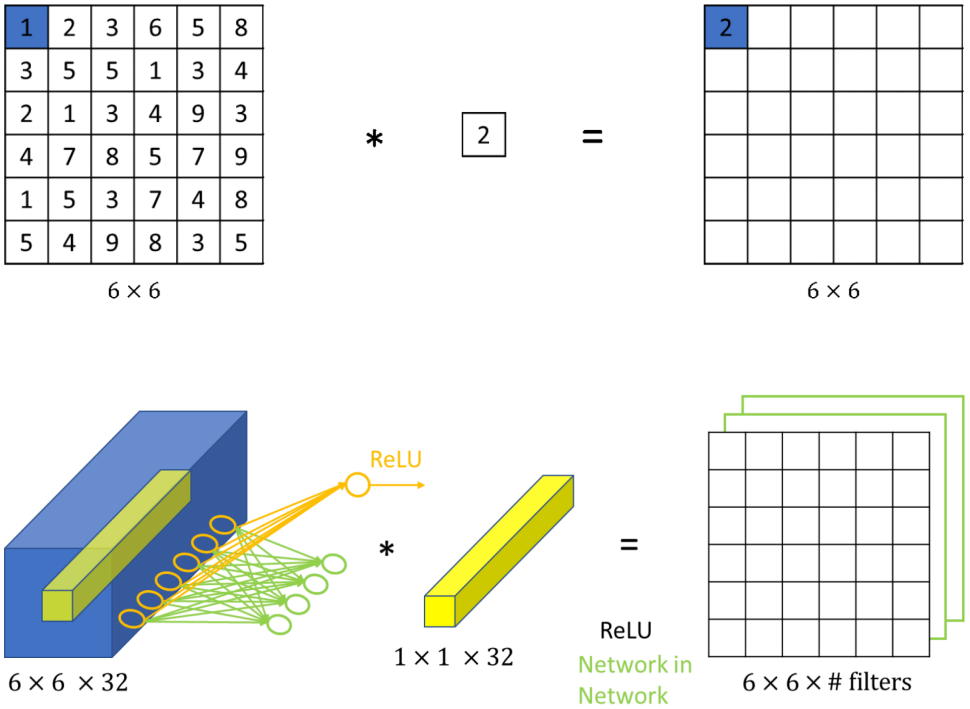


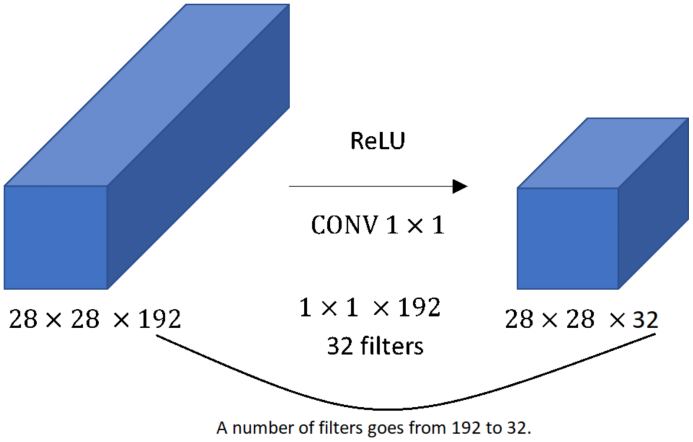
- High computation time
- Number of filters increases due the concatenation of the output of pooling
- A 1x1 convolution can be added to reduce the number of channels for each operation



1x1 Convolution Layers

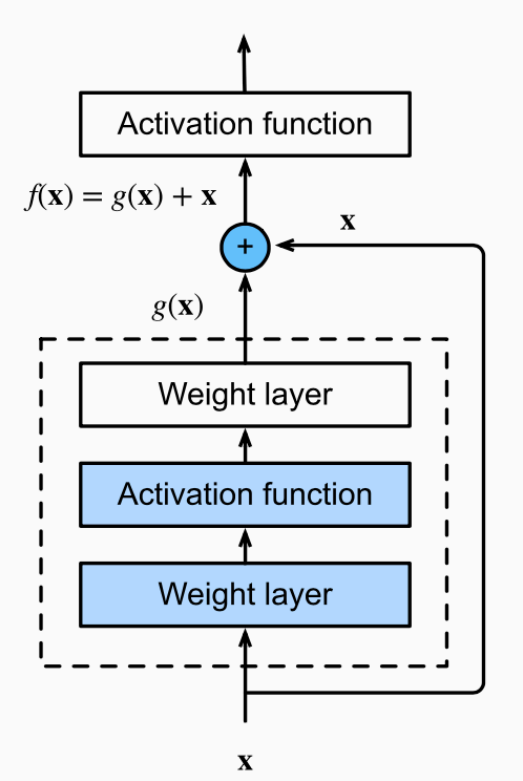
- They have an effect when the image has **multiple channels**
- **Dimensionality reduction:** reduce # of channels to # of 1x1 filters

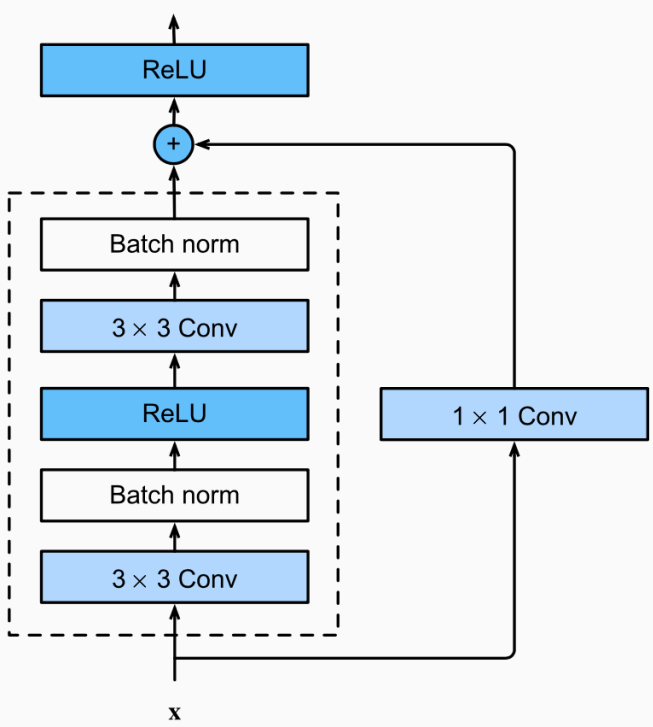


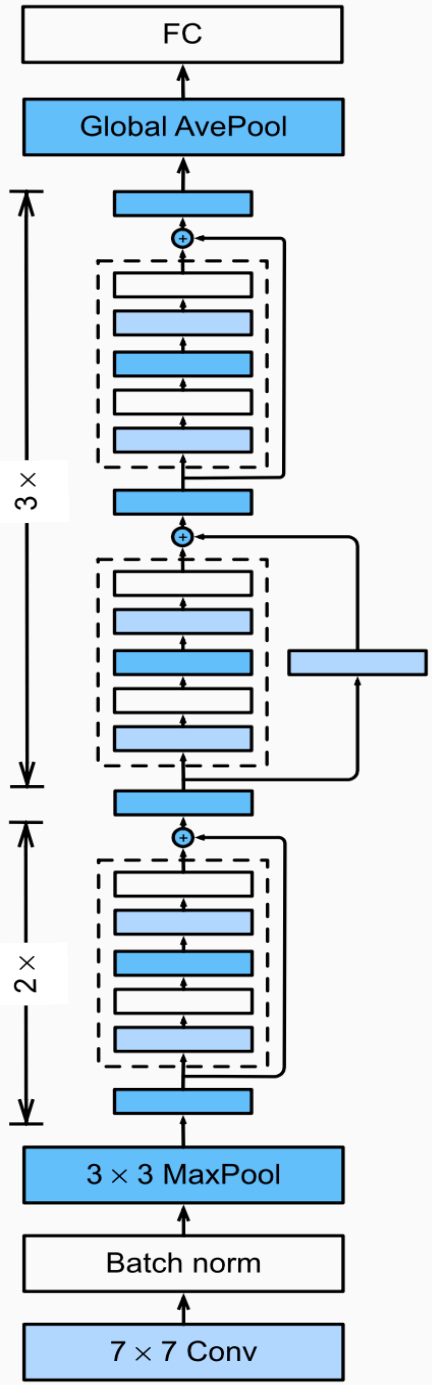


ResNet [He 2015]

- **Different versions: 34, 50, 101, 152 layers**
- **ResNet34/50/101/152**
- Final **average pooling**
- **60 million parameters**
- A skip-connection (shortcut) bypasses the non-linear transformations with an identity function

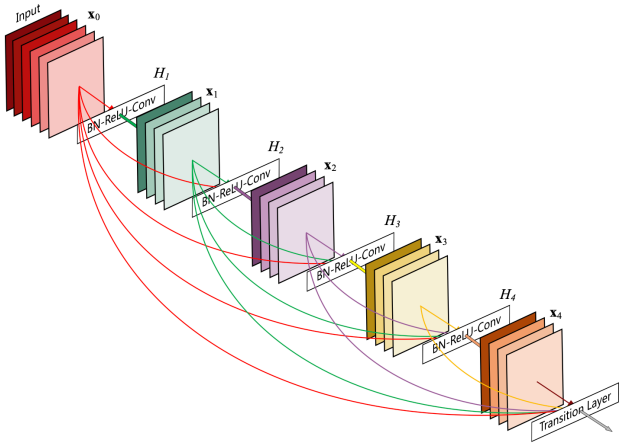


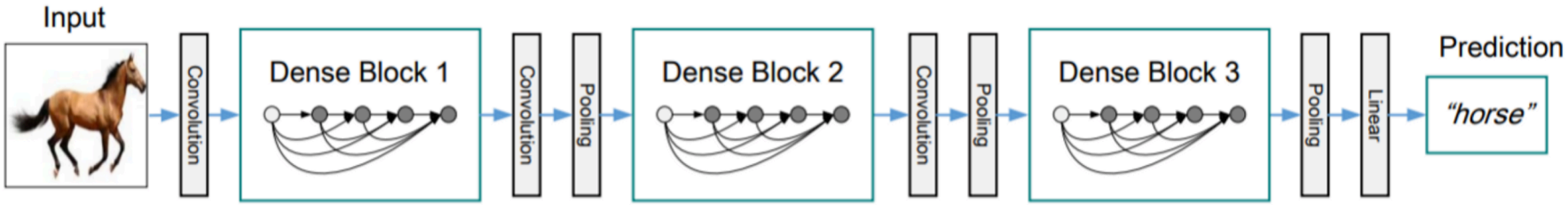




DenseNet [Huang 2016]

- Introduce direct connections from any layer to all subsequent layers
- The l^{th} layer receives the feature-maps of all preceding layers
- **DenseNets** *require fewer parameters* than an equivalent traditional CNN, as there is *no need to learn redundant feature maps*
- **DenseNets** do not sum the output feature maps of the layer with the incoming feature maps but concatenate them





- If each layer produces *k* feature-maps, then the # of inputs into the *l^{th} layer* will be

$$k_0 + k \times (\ell - 1)$$

- k_0 is the number of channels in the input layer

DenseNet [Huang 2016]

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Deploying a pre-trained CNN

- Pretrain a neural network model on a source dataset
- Create a new neural network model by:
  - copying the pretrained model
  - removing the output layer
  - adding an output layer (number of outputs = classes of new dataset)
  - initializing randomly the model parameters of the new output layer
- Train the new model on the target dataset
- The output layer will be trained from scratch, while the parameters of all other layers are fine-tuned based on the parameters of the source model

Coming up next…

Jan 17	Introduction to machine learning
Jan 19	Regression analysis
Jan 20	Laboratory session 1: numpy and regression
Jan 25	Evaluating machine learning results
Jan 26	Ensemble learning
Jan 27	Laboratory session 2: ensemble learning
Feb 2	Deep learning I: Training neural networks
Feb 7	Deep learning II: Convolutional neural networks
Feb 10	Laboratory session 3: training NNs and tensorflow
Feb 14	**Deep learning III: Recurrent neural networks**
Feb 16	Deep learning IV: Transformers, attention, and autoencoders
Feb 17	Laboratory session 4: RNNs

TODOs

- **Reading:**
  - The Deep Learning Book: Chapter 9
- **Homework 1**
  - due: Feb 5