Convolutional Neural Networks

Sascha Diefenbacher

sascha.daniel.diefenbacher@uni-hamburg.de

March 1st, 2023

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

CLUSTER OF EXCELLENCE

QUANTUM UNIVERSE

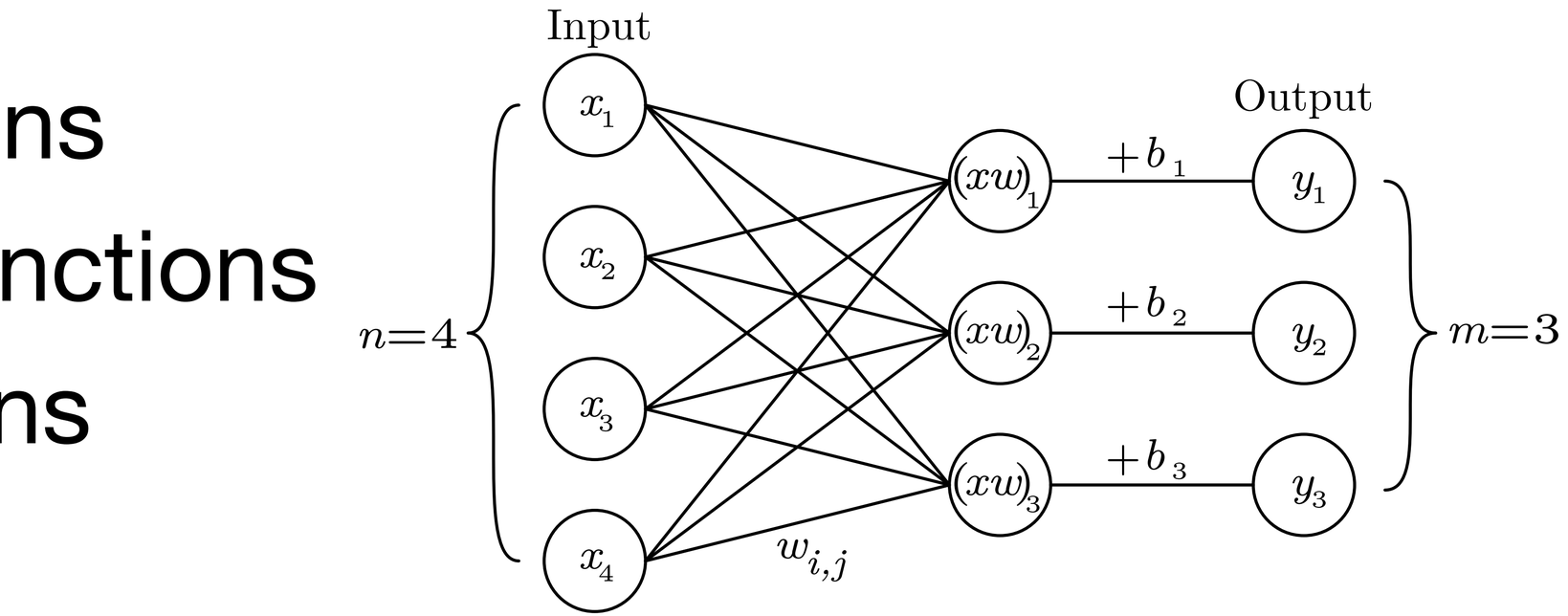
About Me

- Bachelor + Master in Heidelberg
  - Dark matter models + LHC event classification using machine learning
- PhD in Hamburg
  - Generative models for fast simulation (and other applications) in particle physics

- Slides inspired by **Stefan Funk** and **Judith Reindl**

Previous Lectures

- Gradient descent/backpropagation
- Fully connected layers and networks
- Optimisers
- Normalisations
- Activation functions
- Loss functions

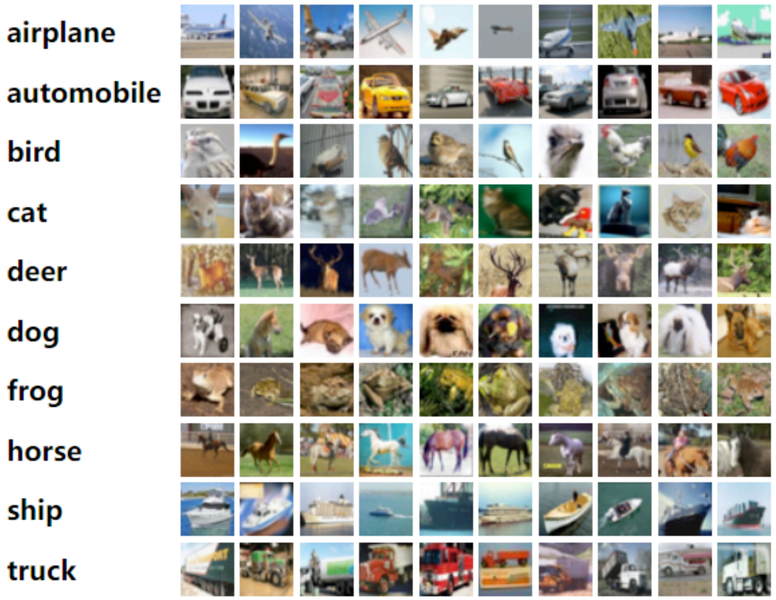


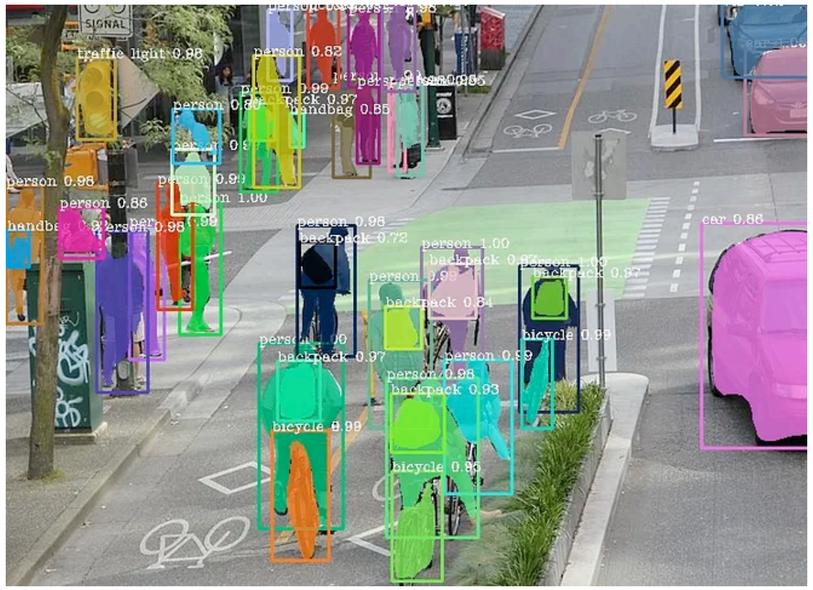
Agenda

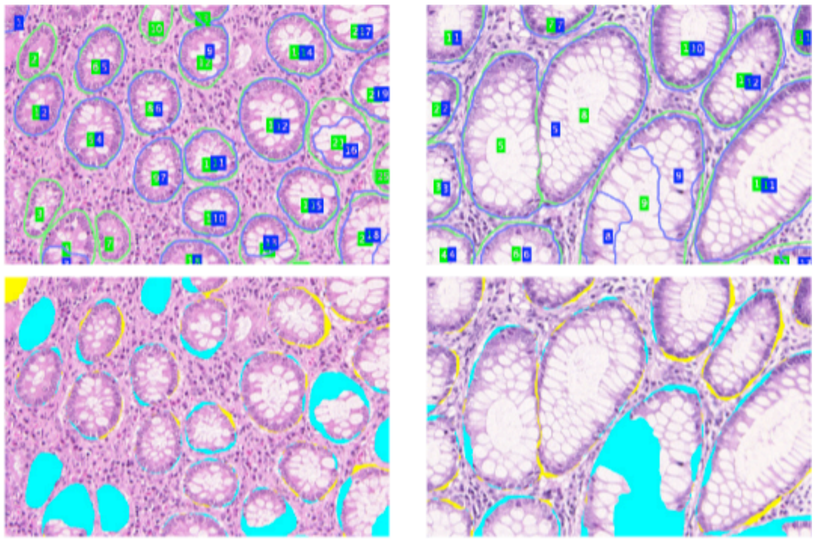
- Challenges of image classification
- Basics of convolutions
- Image convolutions
- Convolutional layers
- Convolutional neural networks
- Outlooks

Challenge: Image Classification

- Present in multitude of applications
  - Picture labelling, self driving cars, image segmentation, cancer screening …







(e) benign (f) malignant

Images (as Seen by Computers)

What we see:

How its encoded

Colors are encoded into RGB components

Images (as Seen by Computers)

What we see:

How its encoded

5	10	2	4	2	0
3	6	2	0	1	3
0	7	10	4	8	0
11	10	8	0	3	7
10	4	8	6	0	4
5	12	11	0	2	0

5	6	2	3	2	0
5	6	2	0	1	3
0	5	12	2	5	6
2	2	8	0	3	7
0	4	5	10	0	4
4	1	4	0	2	9

6	10	2	4	2	2
3	6	12	0	1	6
0	5	2	4	5	4
11	13	8	1	3	7
9	4	4	6	0	9
5	12	5	8	2	9

Color values are matrices

Challenge: Image Classification

Difficulty 1:

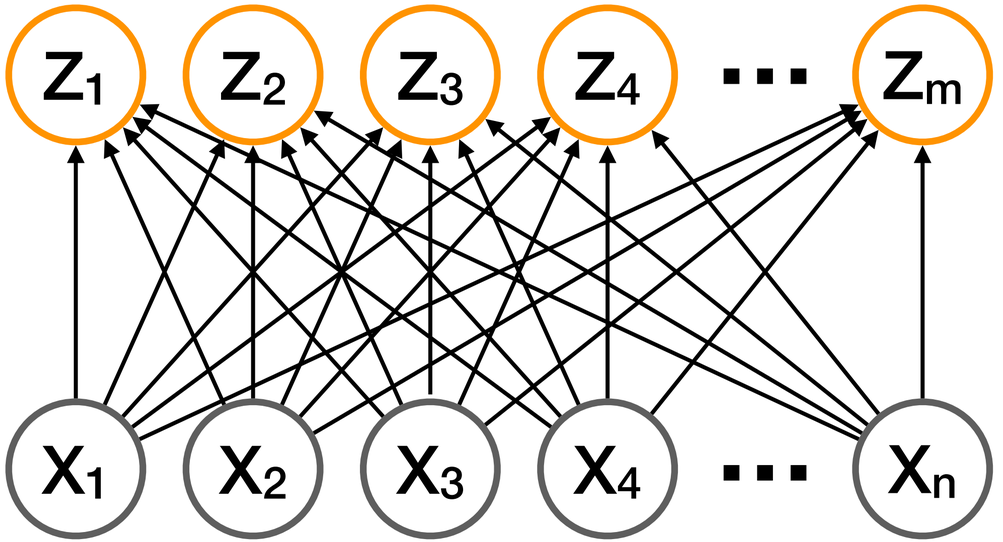
- Fully connected layers do not scale well
  - CIFAR10 RGB image

➡ 32x32x3 = 3072 values

  - 32x32x3 → 32x32x3 layer

➡ 9.440.256 parameters

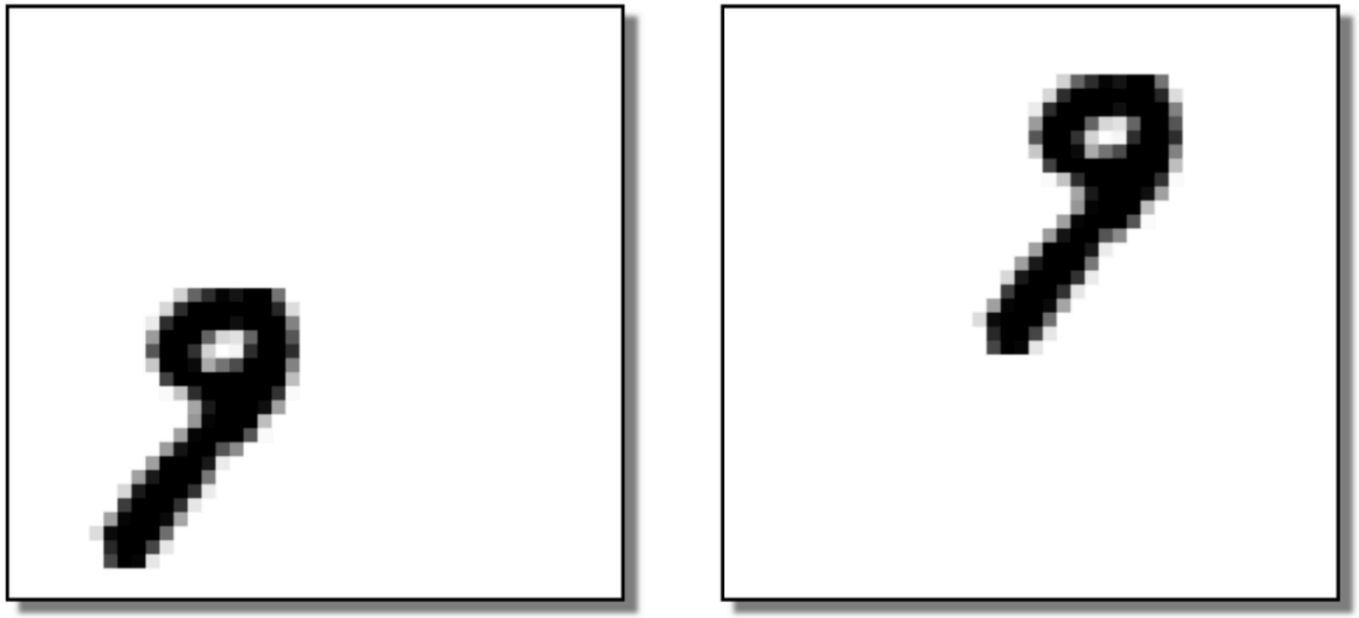
- Even more significant for high-resolution inputs



Challenge: Image Classification

Difficulty 2:

- Feature translation
  - Feature position less important than feature being present
  - FCN layers: independent weights for each input
  - Needs to relearn for every feature position



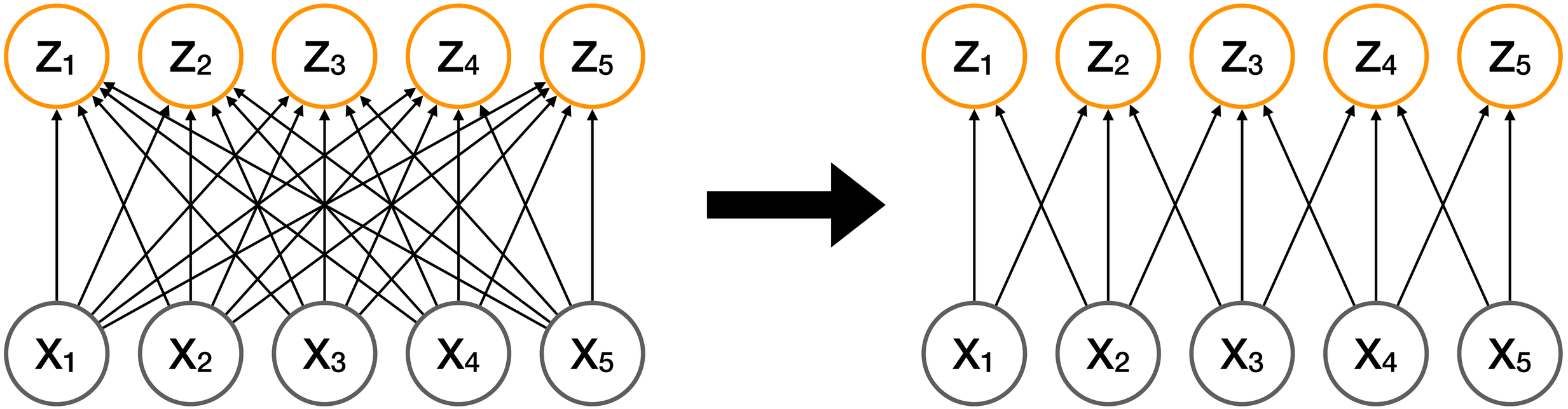
Challenge: Image Classification

What we need/want:

- **Local connections**
  - Inputs are connected to neighbouring inputs, but not everything
- **Parameter sharing**
  - One set of weights for every location
- **Translational invariance**
  - Shifted input results in same/similar output

Local Connections

- Image features have limited size
- Not entire image needs to be ‘seen’ at once

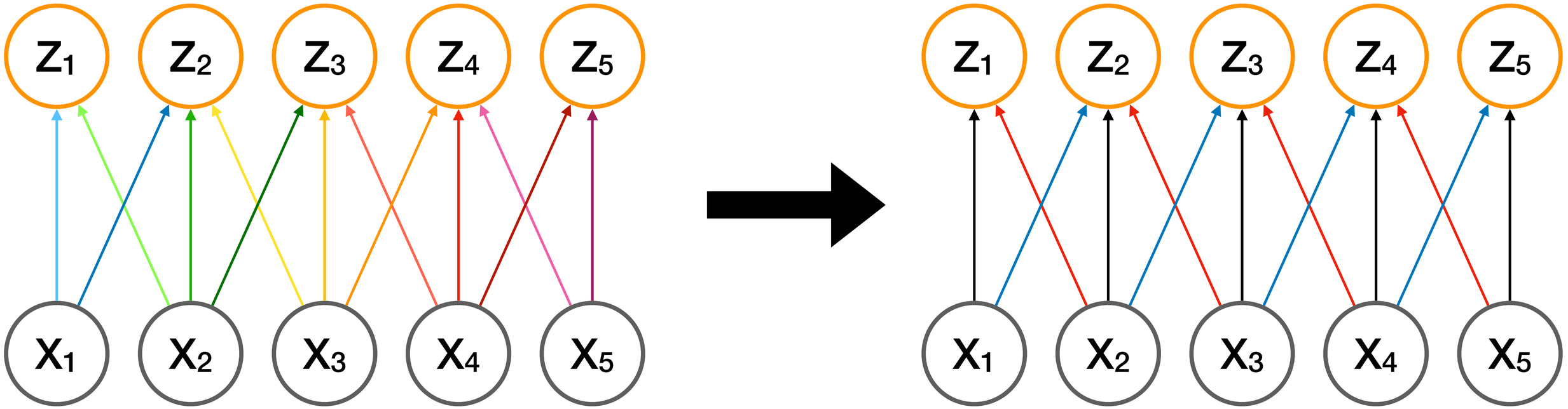


Fully Connected Layer

Locally Connected

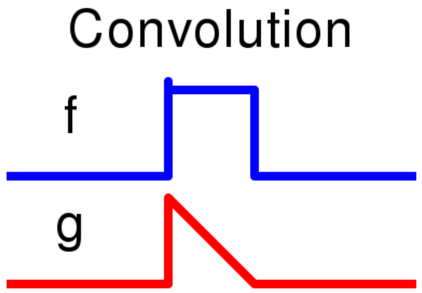
Parameter Sharing

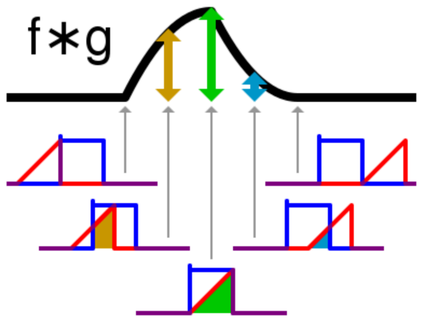
- Features can appear at various positions
- Weights that detect features can be repeatedly applied

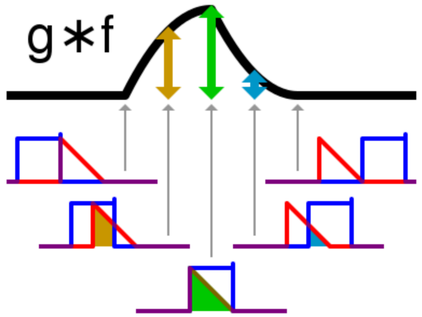


Locally Connected **Convolution**

Mathematical Convolutions







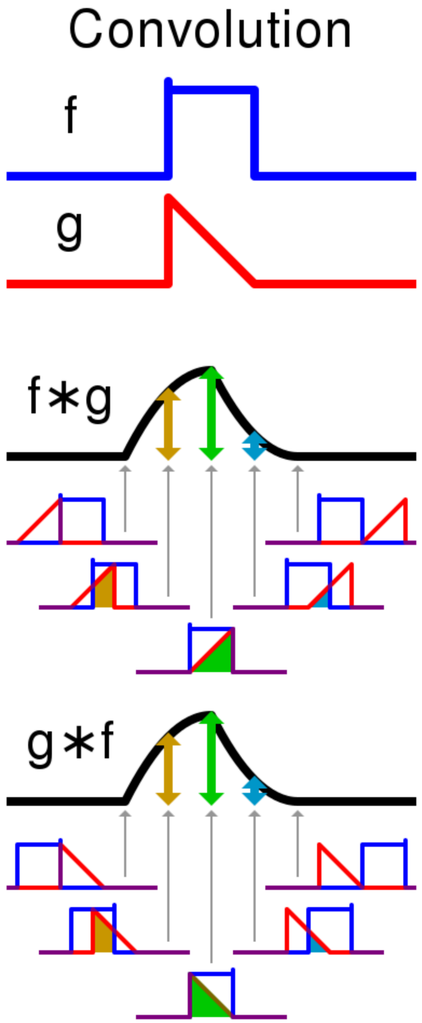
- Continuous convolution for real-valued functions f and g $(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$
- Our image inputs are discrete

➡ Discrete formulation $(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m)$

- Two dimensional case with Image I and Kernel K:

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Mathematical Convolutions



- Two dimensional case with Image I and Kernel K:

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

  - Pick point (i, j) in image
  - Multiply image pixel at position by kernel entry
  - Repeat for area around (i, j)
  - Still somewhat abstract, let's see in practice

➡ Image convolution

Image Convolutions

Image I

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

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1_0	1_1	1_0	0	0
0_1	1_2	1_1	1	0
0_0	0_1	1_0	1	1
0	0	1	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1_0	1_1	1_0	0	0
0_1	1_2	1_1	1	0
0_0	0_1	1_0	1	1
0	0	1	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4		

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1	1_0	1_1	0_0	0
0	1_1	1_2	1_1	0
0	0_0	1_1	1_0	1
0	0	1	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4		

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1	1_0	1_1	0_0	0
0	1_1	1_2	1_1	0
0	0_0	1_1	1_0	1
0	0	1	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4	6	

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1	1	1_0	0_1	0_0
0	1	1_1	1_2	0_1
0	0	1_0	1_1	1_0
0	0	1	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4	6	

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1	1	1_0	0_1	0_0
0	1	1_1	1_2	0_1
0	0	1_0	1_1	1_0
0	0	1	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4	6	4

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1	1	1	0	0
0_0	1_1	1_0	1	0
0_1	0_2	1_1	1	1
0_0	0_1	1_0	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4	6	4

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

1	1	1	0	0
0_0	1_1	1_0	1	0
0_1	0_2	1_1	1	1
0_0	0_1	1_0	1	0
0	1	1	0	0

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4	6	4
2		

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Image I

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

X

Kernel K

0	1	0
1	2	1
0	1	0

=

I * K

4	6	4
2	5	6
2	5	4

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) = (K * I)(i,j)$$

Image Convolutions

Fills our requirements:

- Local connections
- Parameter sharing
- Translational equivariance

But what does it DO?

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

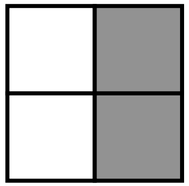
X

0	1	0
1	2	1
0	1	0

=

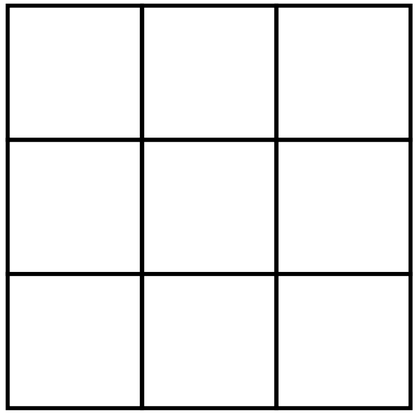
4	6	4
2	5	6
2	5	4

Image Processing: Edge Detection



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

x



=

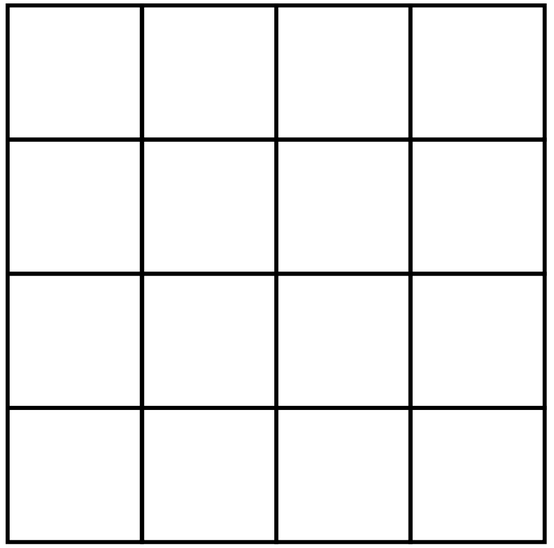
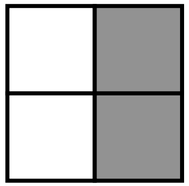
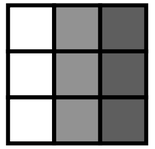


Image Processing: Edge Detection



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

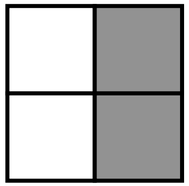
x



1	0	-1
1	0	-1
1	0	-1

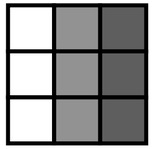
=

Image Processing: Edge Detection



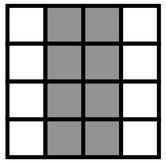
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

x



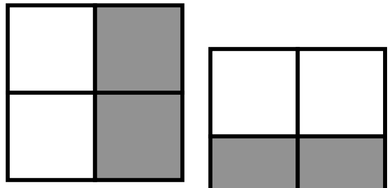
1	0	-1
1	0	-1
1	0	-1

=



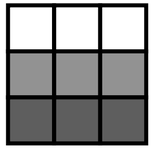
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Image Processing: Edge Detection



10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

x



1	1	1
0	0	0
-1	-1	-1

=

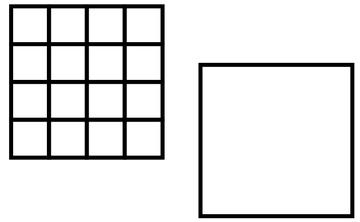
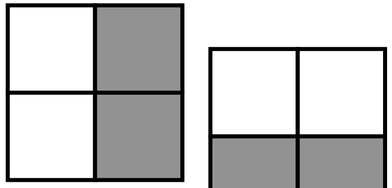
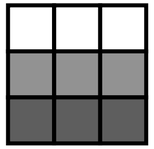


Image Processing: Edge Detection



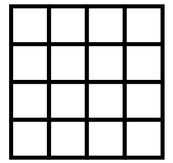
10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

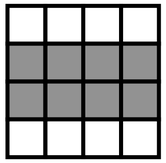
X



1	1	1
0	0	0
-1	-1	-1

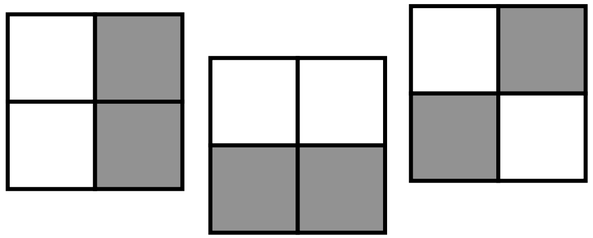
=





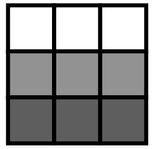
0	0	0	0
30	30	30	30
30	30	30	30
0	0	0	0

Image Processing: Edge Detection



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

x



1	1	1
0	0	0
-1	-1	-1

=

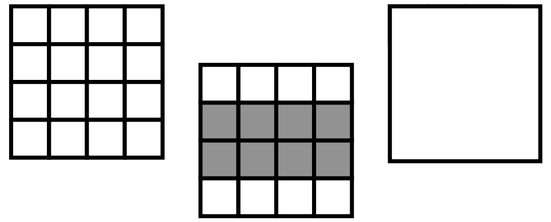
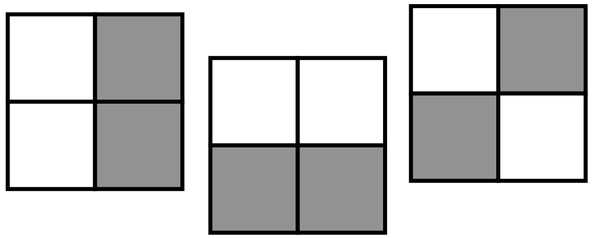
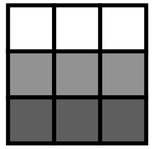


Image Processing: Edge Detection



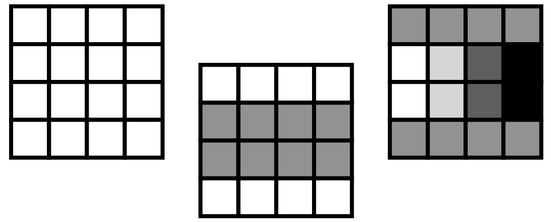
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

X



1	1	1
0	0	0
-1	-1	-1

=



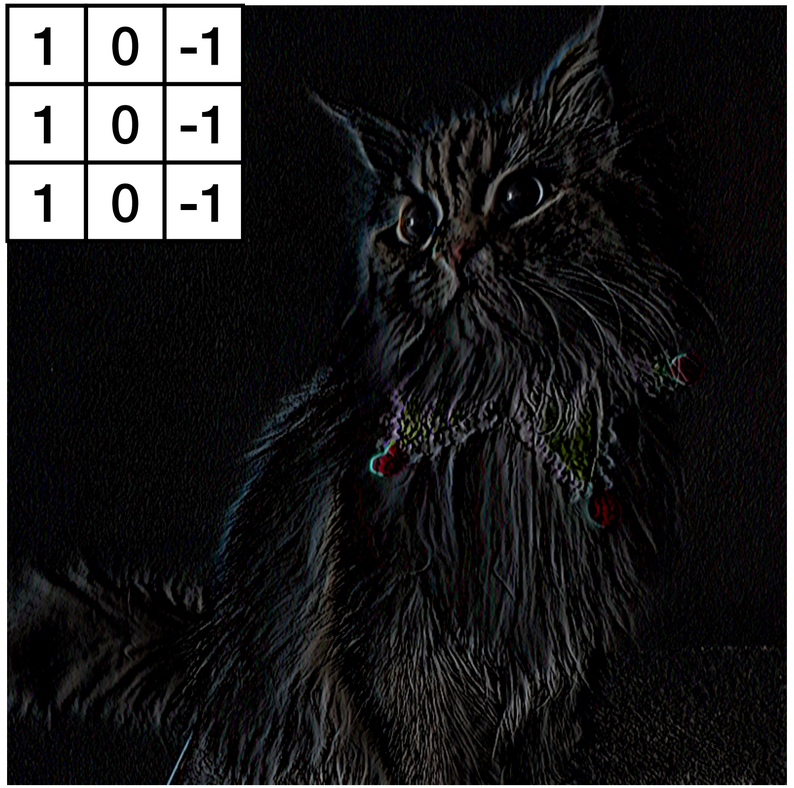
0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Image Processing: Edge Detection

Original

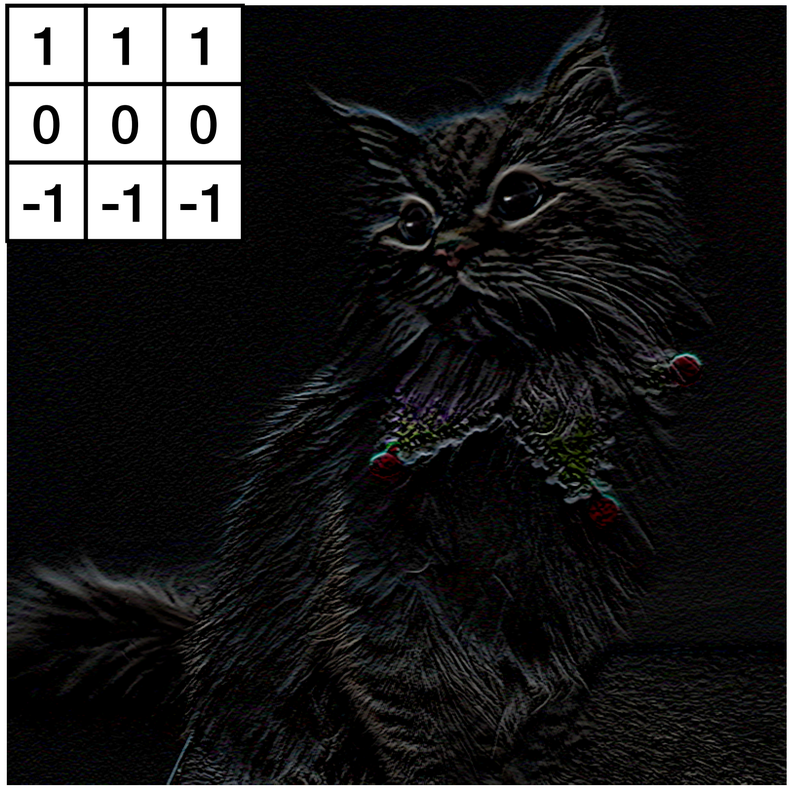
Vertical Edges

1	0	-1
1	0	-1
1	0	-1



Horizontal Edges

1	1	1
0	0	0
-1	-1	-1



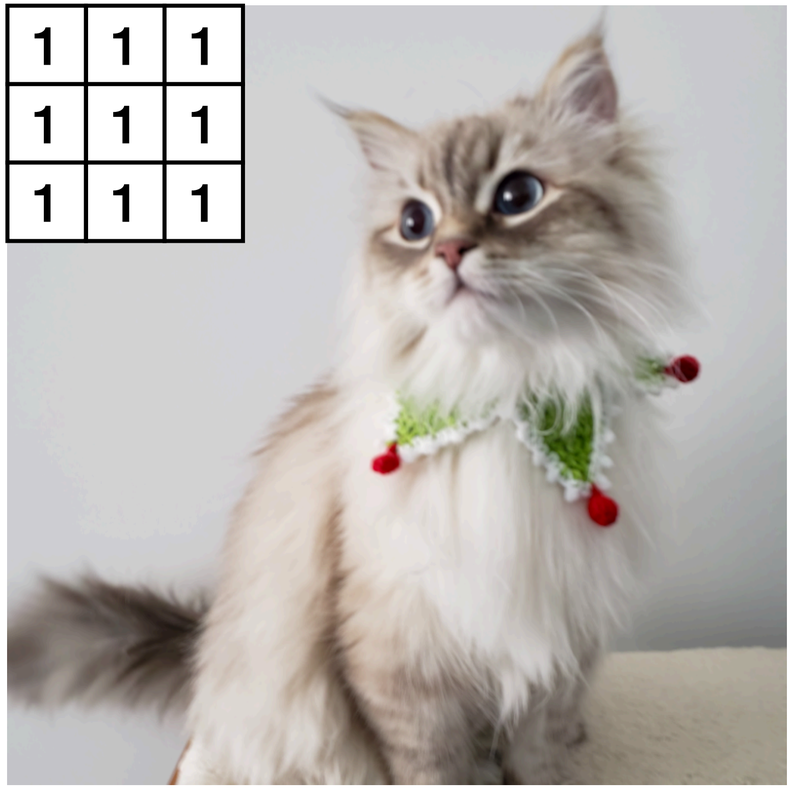
- Already allows extraction of potentially interesting features

Image Processing: Edge Detection

Original

Blur

1	1	1
1	1	1
1	1	1



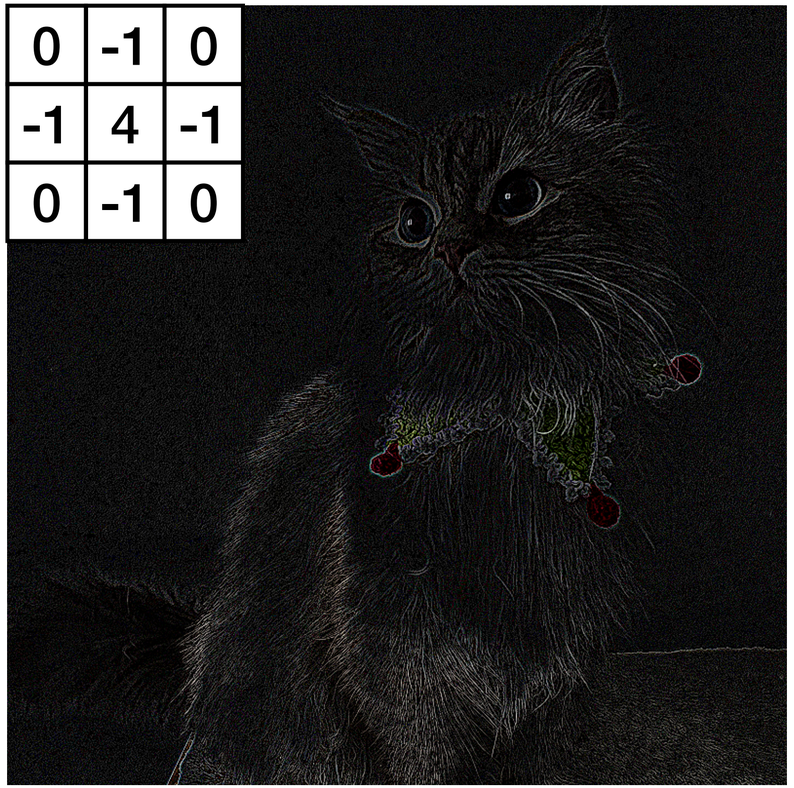
- Already allows extraction of potentially interesting features

Image Processing: Edge Detection

Original

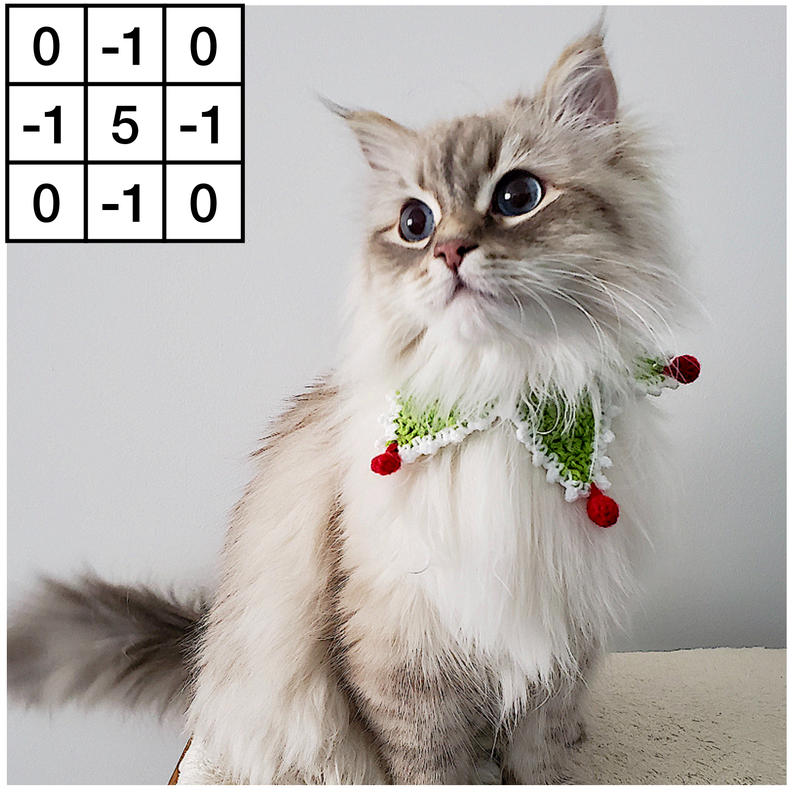
Multi Edge

0	-1	0
-1	4	-1
0	-1	0



Sharpen

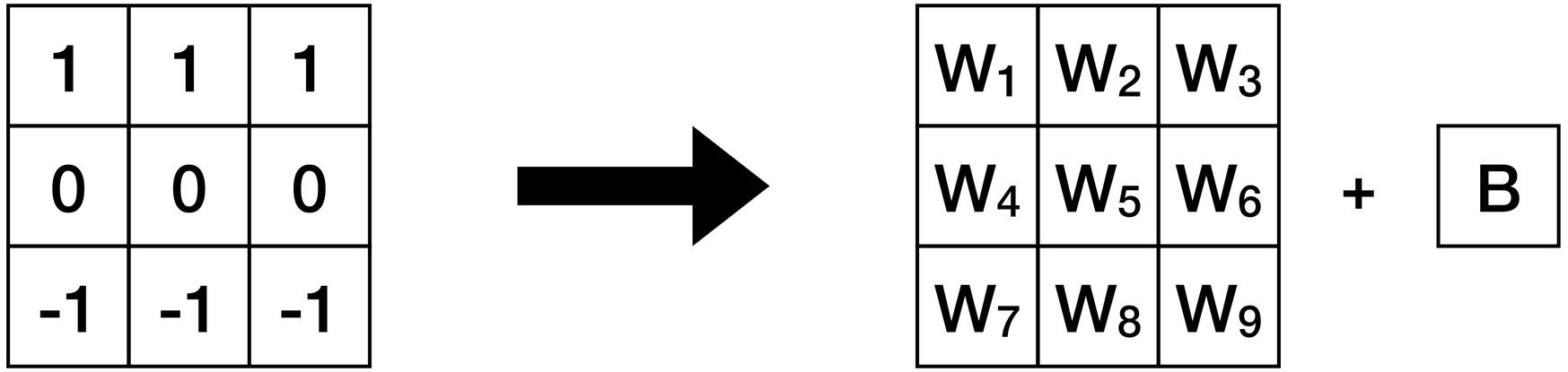
0	-1	0
-1	5	-1
0	-1	0



- Already allows extraction of potentially interesting features

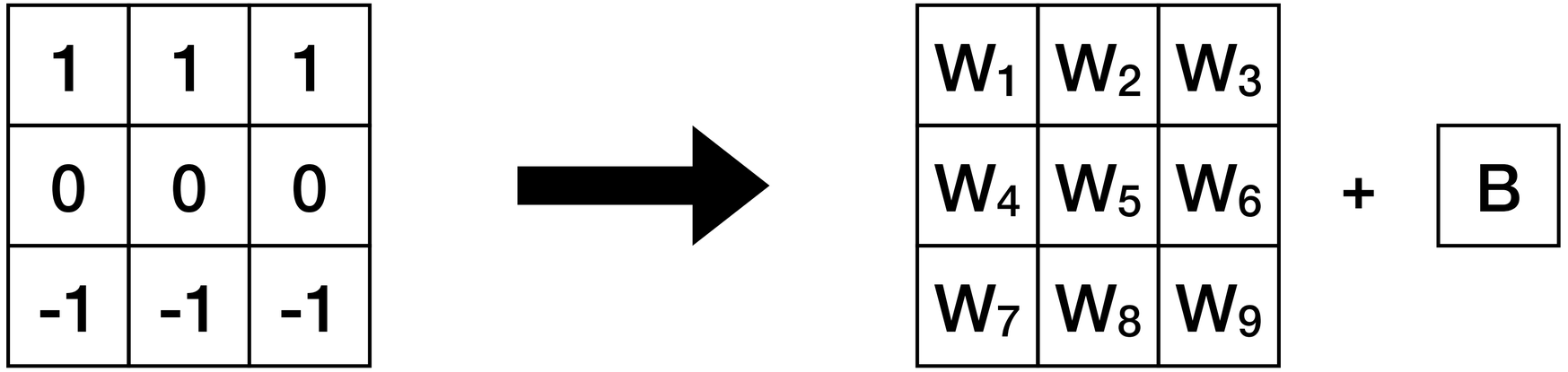
Machine Learning Convolutions

- So far: specific filter for specific purposes
- For ML: need generalised version



- Replace kernel entries with trainable weights + bias term
- Convolutional neural network layer

Machine Learning Convolutions



- Replace kernel entries with trainable weights + bias term
- Convolutional neural network layer
- Bias/weights are additive/multiplicative with inputs

➡ Simple back propagation

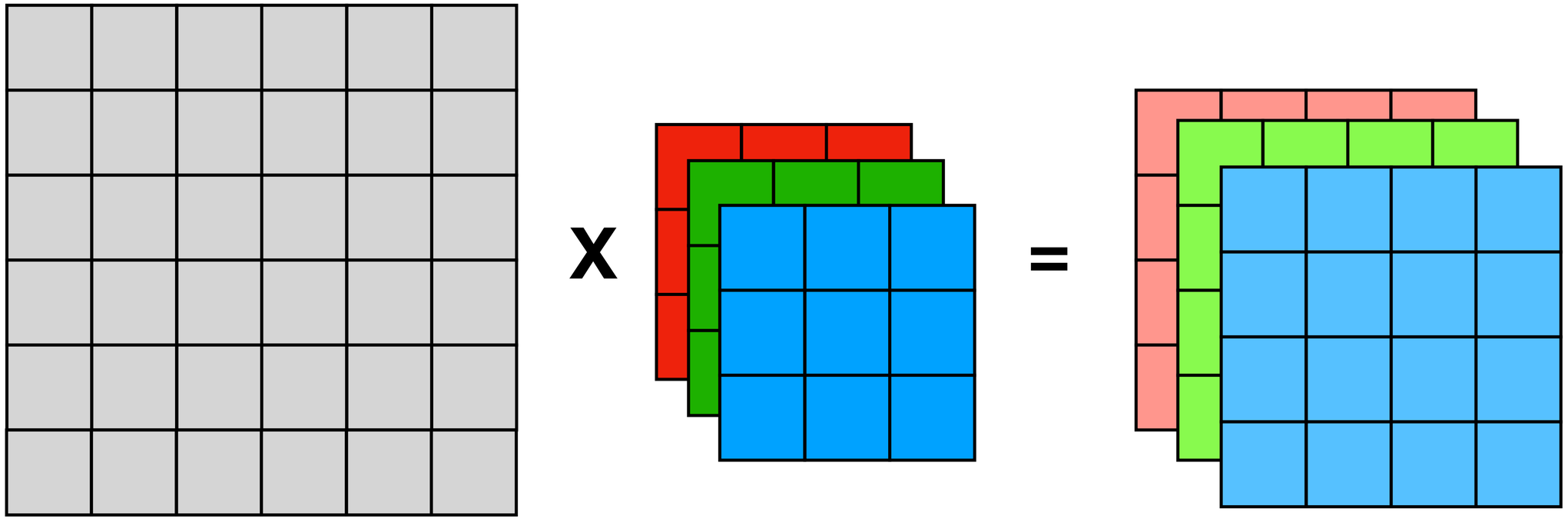
- Network can learn relevant feature extraction kernels

Convolutional Layers

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

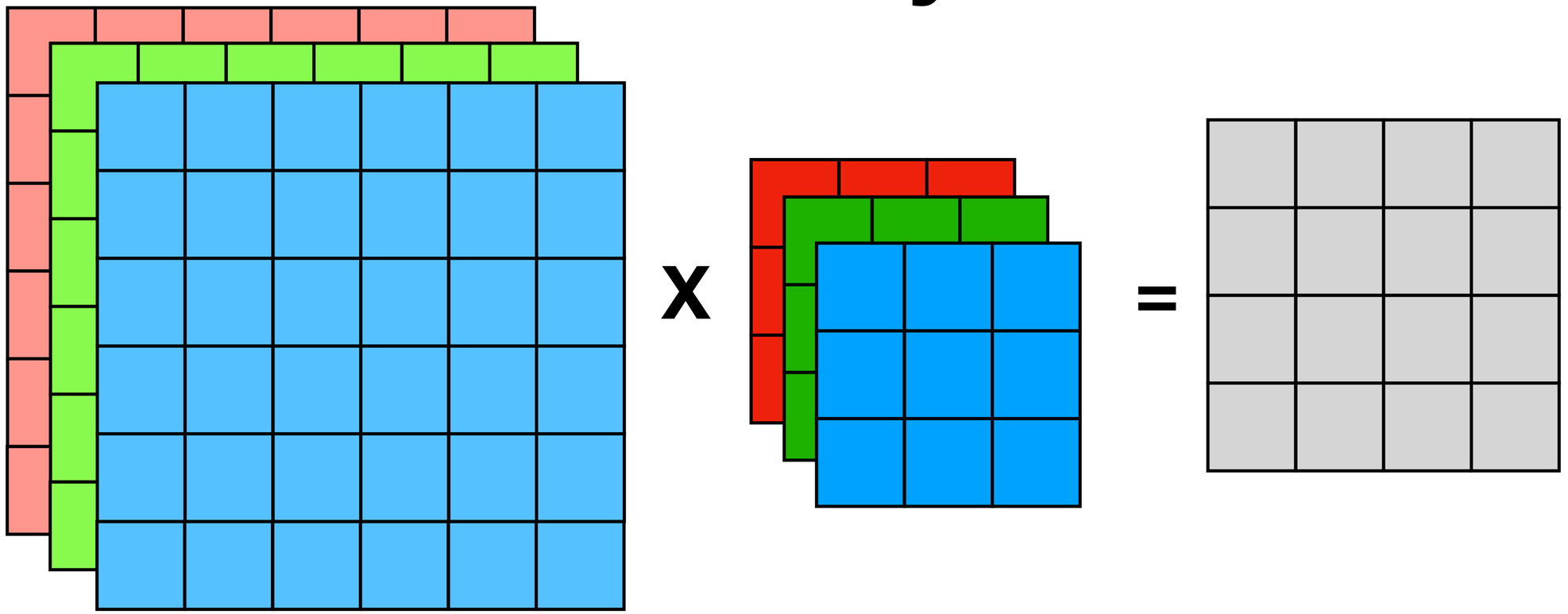
- Readily implemented in any ML library
- Important technical details:
  - Channels
  - Kernel size
  - Padding
  - Strides

Convolutional Layers: Channels



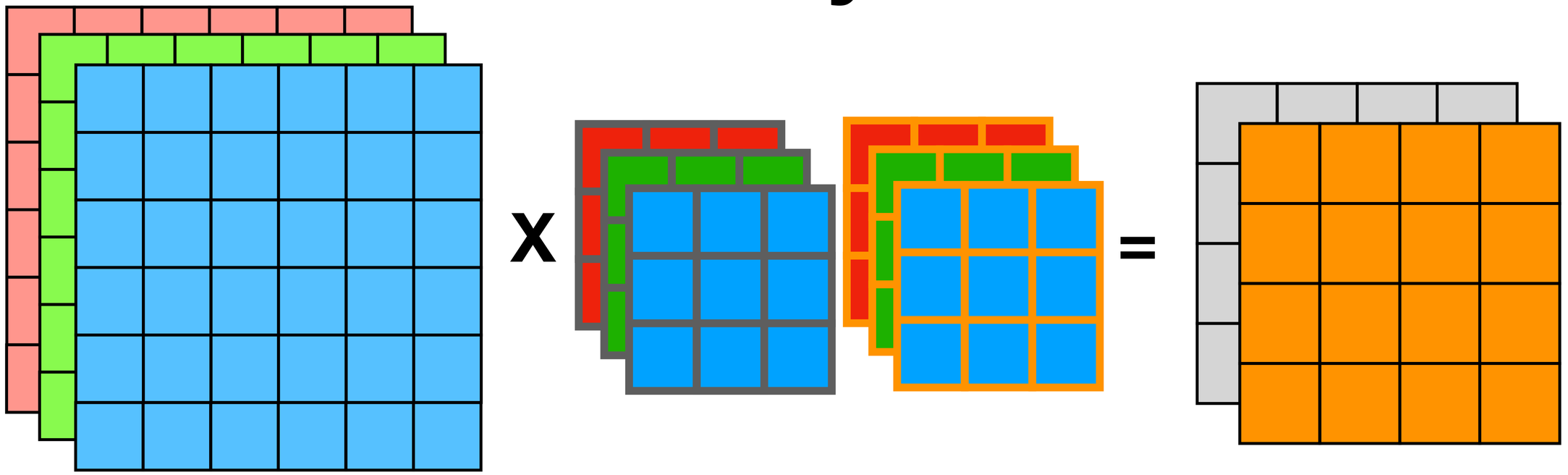
- One filter may not be sufficient to extract all important features
- Use multiple filters, giving multiple output channels/feature maps
- Each filter has independent weights

Convolutional Layers: Channels



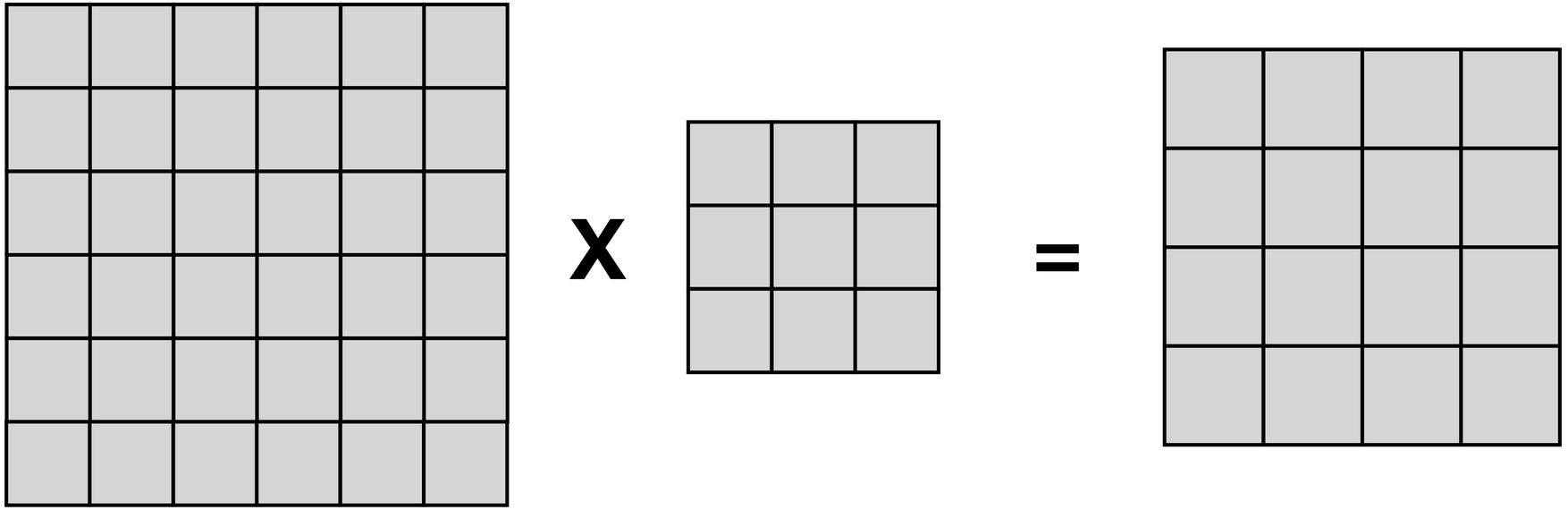
- Multiple input channels (from previous conv. layer, RGB colors…)
- One filter per input channel, sum over each convolution result
- One output channel, containing information from all inputs

Convolutional Layers: Channels



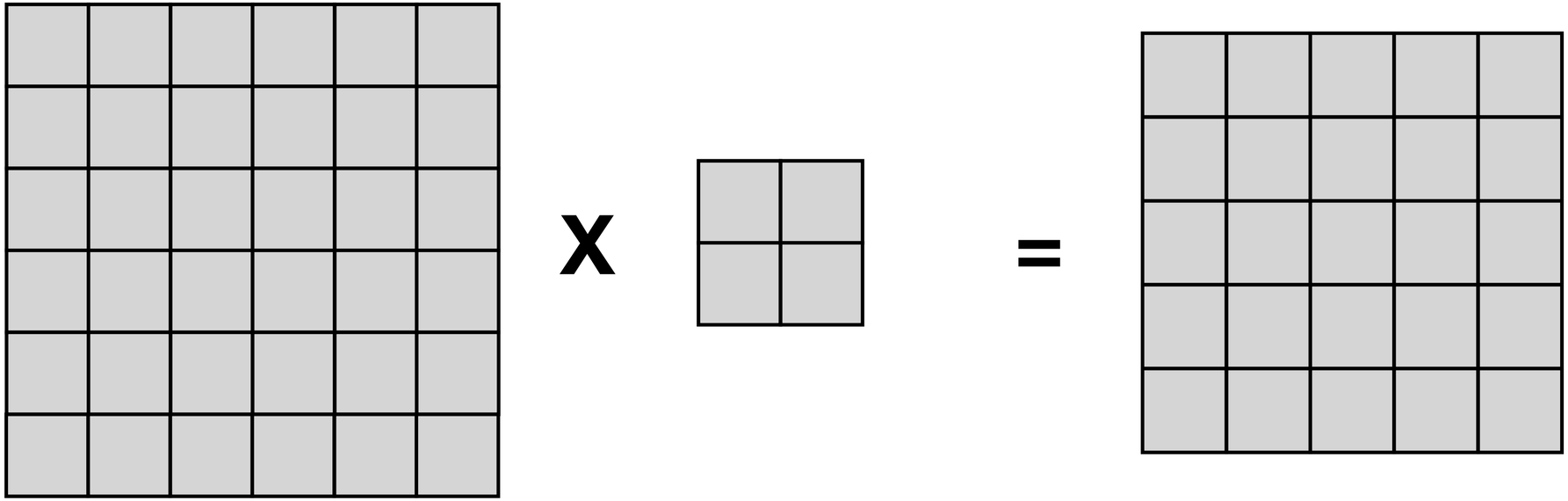
- Multiple input and output channels
- One set of filters per output channel
- One filter in set per input channel, sum over input results

Convolutional Layers: Kernel Sizes



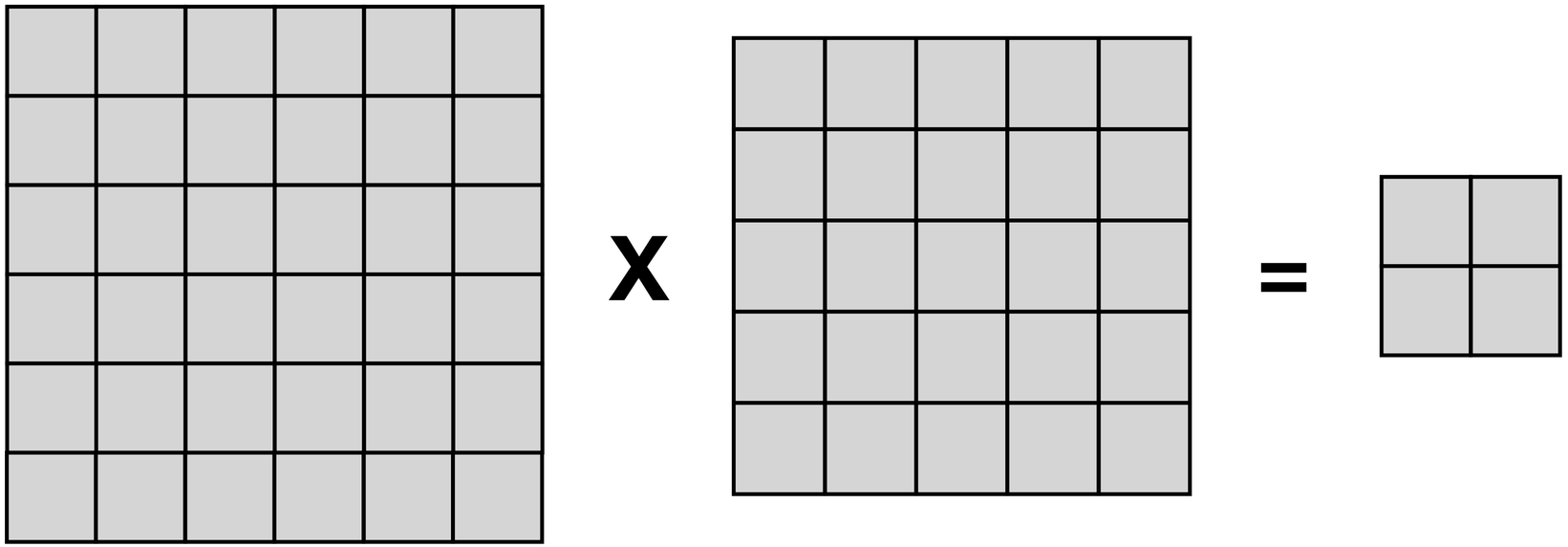
- Kernel size dictates how far pixels are connected

Convolutional Layers: Kernel Sizes



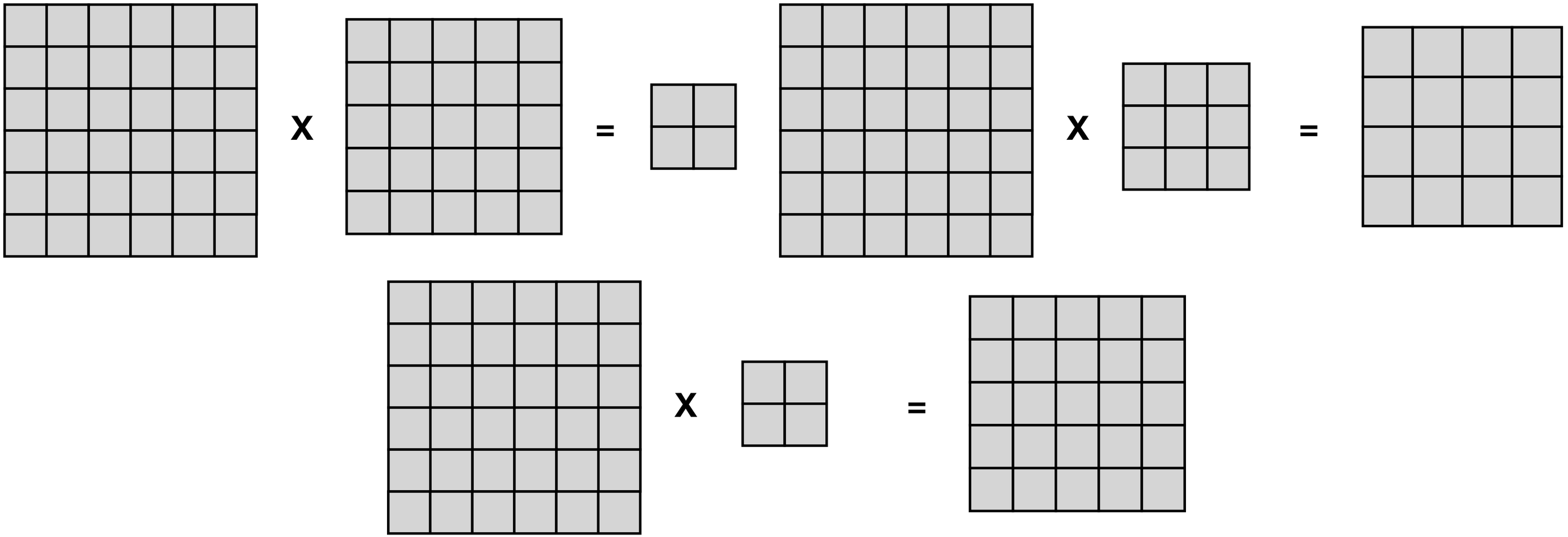
- Kernel size dictates how far pixels are connected
- Too small kernel: insufficient to cover interesting features

Convolutional Layers: Kernel Sizes



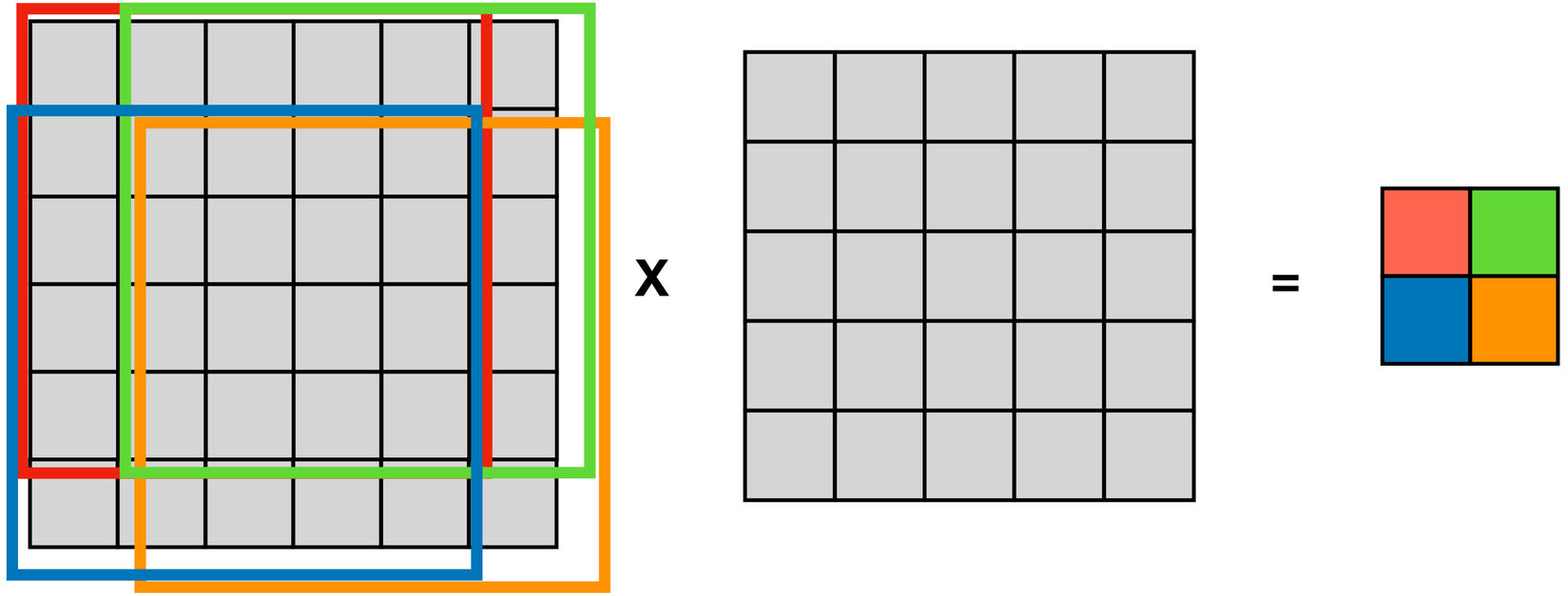
- Kernel size dictates how far pixels are connected
- Too small kernel: insufficient to cover interesting features
- Too large kernel: defeats purpose of shared weights

Convolutional Layers: Padding



- Observation: kernel size affects size of convolution output

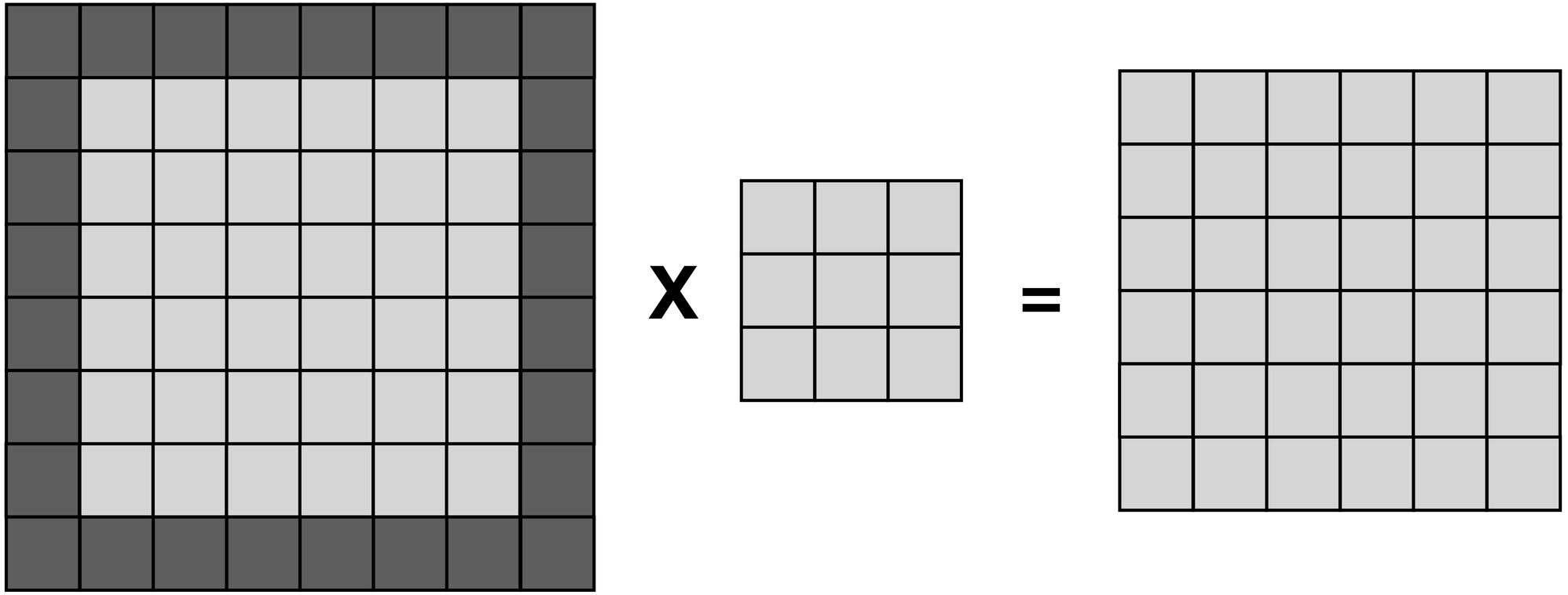
Convolutional Layers: Padding



- Kernel needs to overlap fully with input image
- For kernel size > 1: reduces the output size (compared to input)
- This is the default setting, also known as valid padding
- We can also use other padding

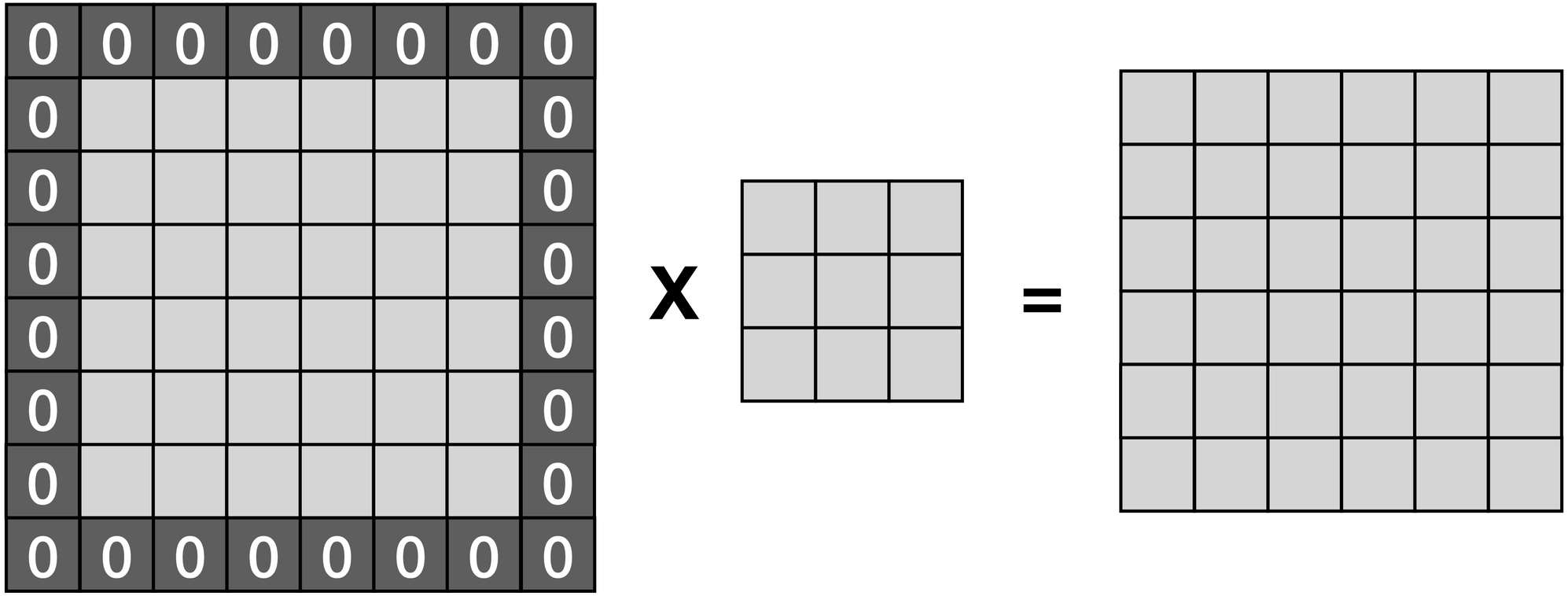
Convolutional Layers: Padding

- ‘Same’ padding: pads such that input and output are same size
- Other padding shapes are possible, e.g. fixed numbers of padded pixels (padding = 1, 2, 3 …)



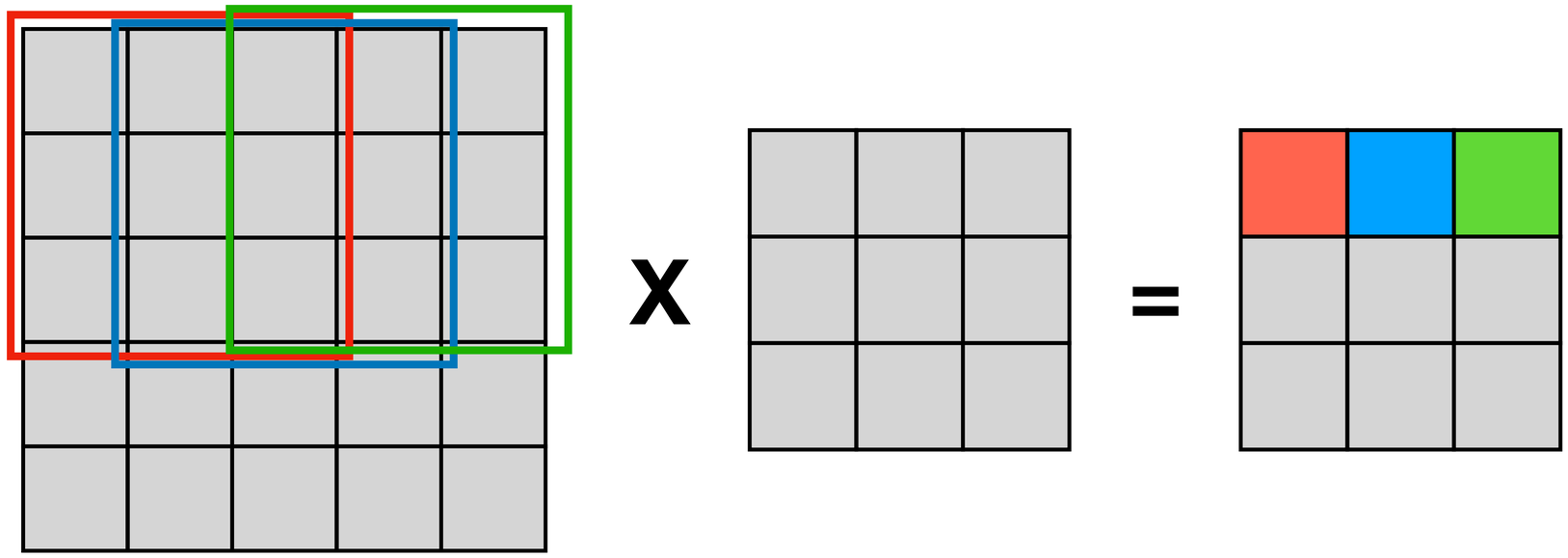
Convolutional Layers: Padding

- ‘Zero’ padding: pads given area using 0 as a fill value
- Can have ‘Zero’ (value) ’Same’ (shape) padding



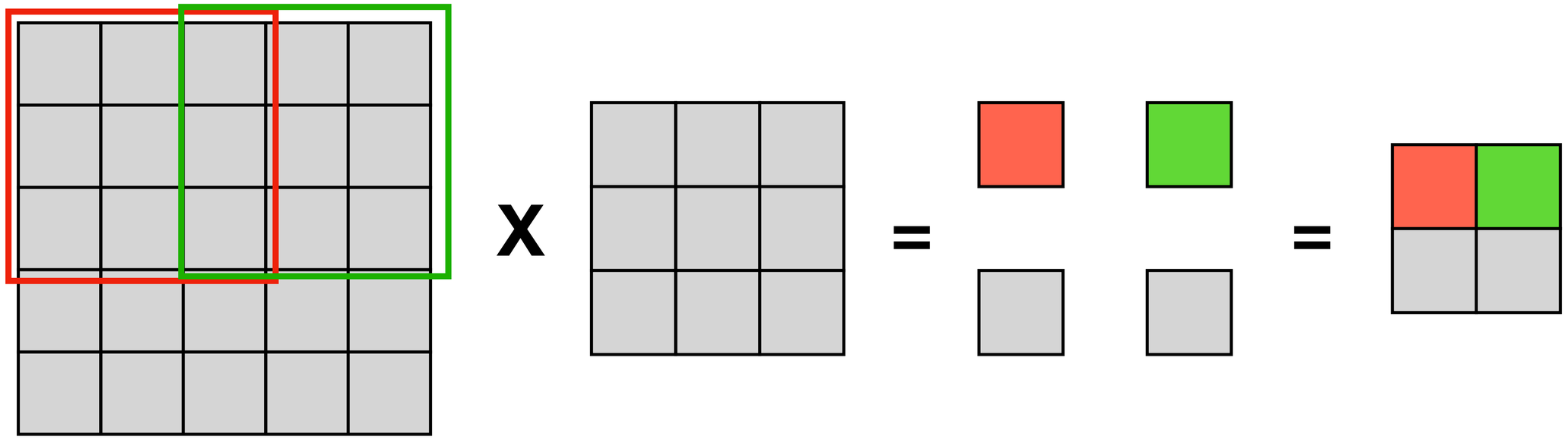
Convolutional Layers: Strides

- Strides describe how the convolutional kernel moves each step
- Stride = 1: Move kernel by 1 pixel



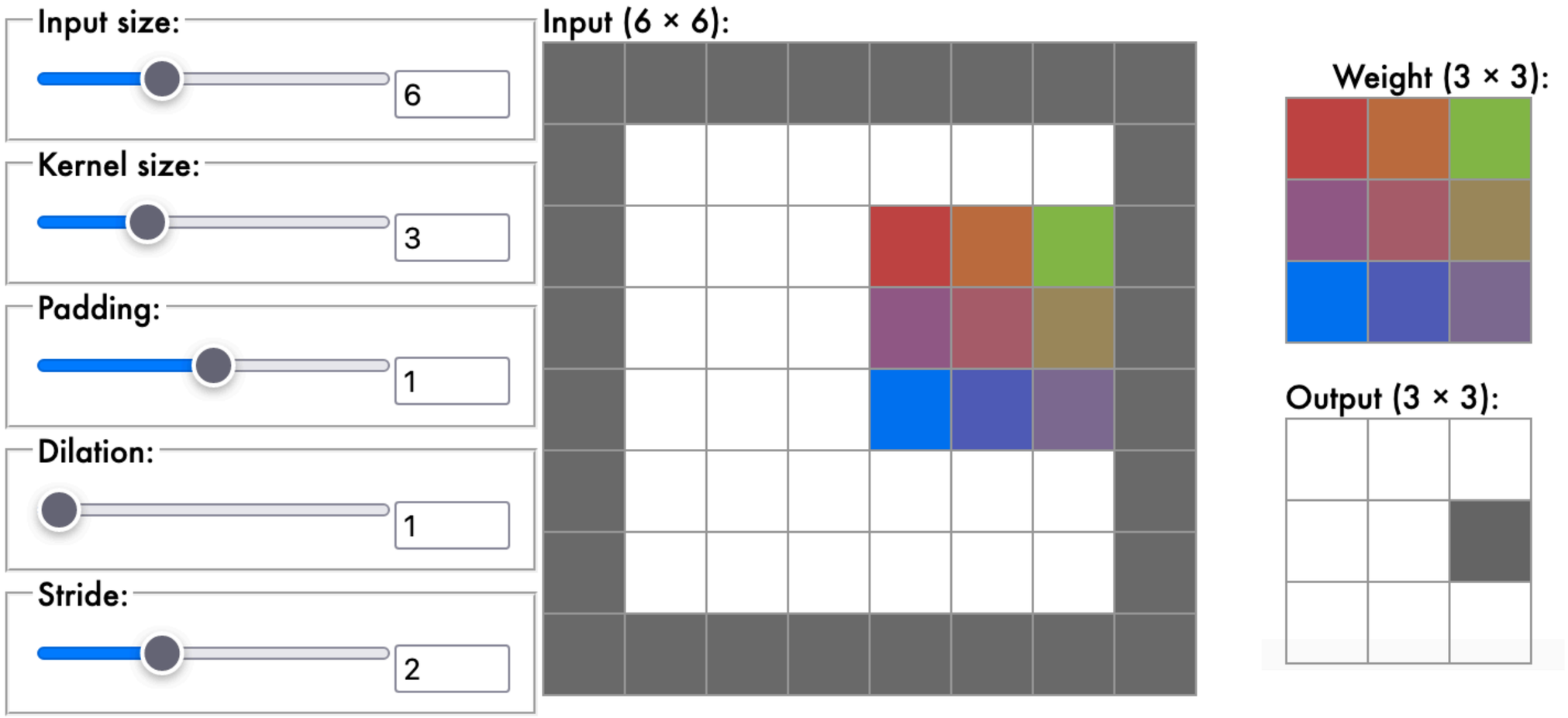
Convolutional Layers: Strides

- Strides describe how the convolutional kernel moves each step
- Stride = 2: Move kernel by 2 pixel



- Strides > 1 act as downsampling

Interactive Visualisation



https://ezyang.github.io/convolution-visualizer/

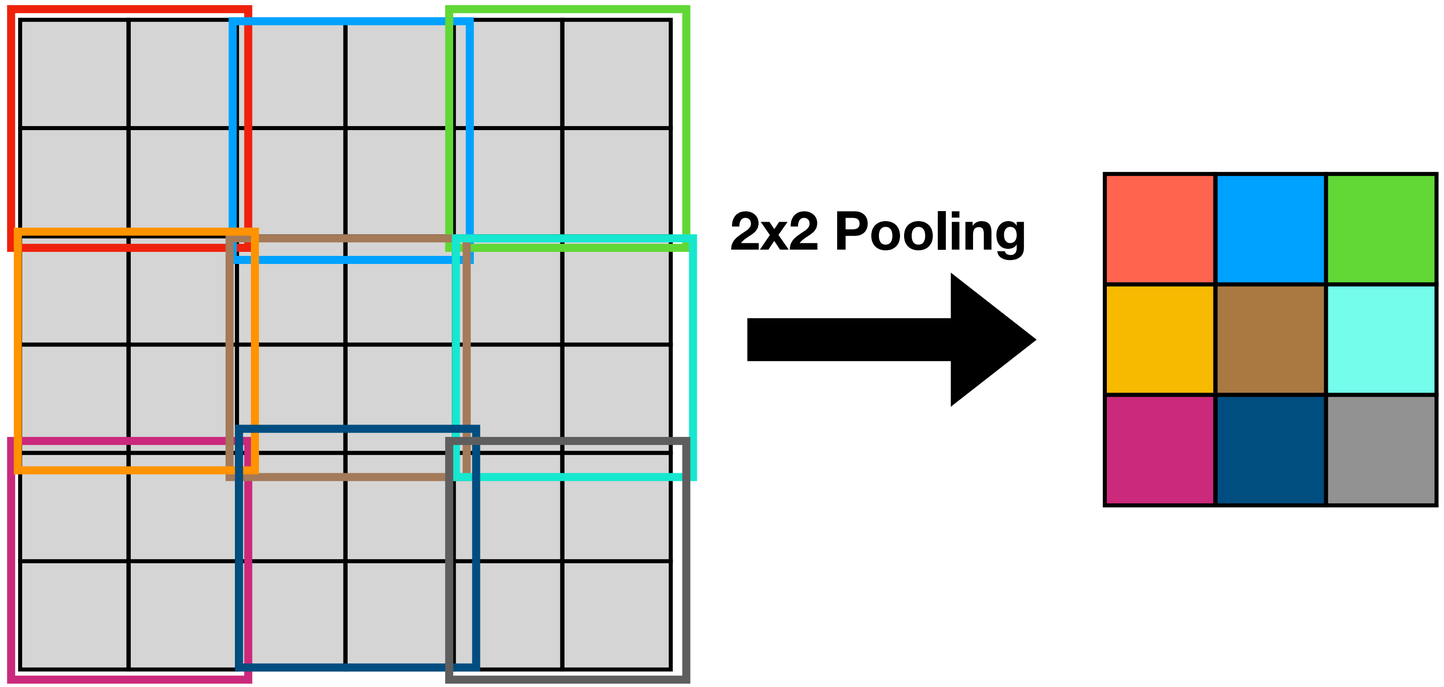
Convolutional Layers

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

- Readily implemented in any ML library
- Important technical details summary:
  - **Channels:** number of input and output feature maps
  - **Kernel size:** choose appropriate to feature size
  - **Padding:** adds ‘empty’ pixels to manipulate output size
  - **Strides:** ‘step size’, allows for downsampling

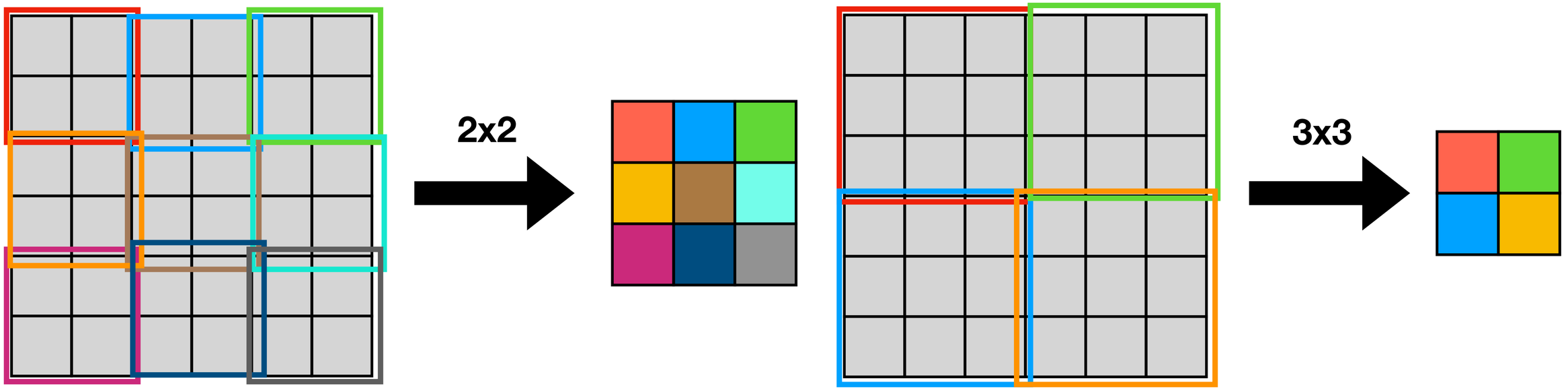
Pooling

- Reducing the image dimensionality is often required
- Stride convolutions offer one such reduction
- Pooling presents a popular alternative approach



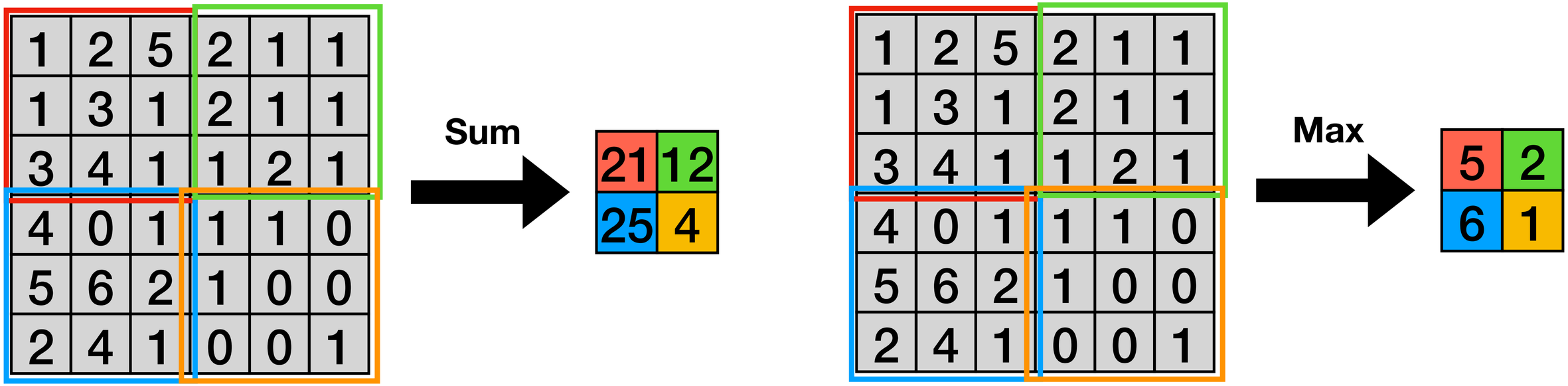
Pooling

- Pooling: kernel based operation to reduce dimensionality
- Fixed kernel function, stride = kernel size by default (other strides are possible but not common)
- Kernel size decides output size



Pooling

- Any function can be used as the kernel function, common ones:
  - Sum-pooling: preserves total intensity of pixels
  - Max-pooling: focuses on high value pixels



Convolutional Neural Networks

- We can now use convolutional layers (and pooling) to build a convolutional neural network
- Assumption: image classification

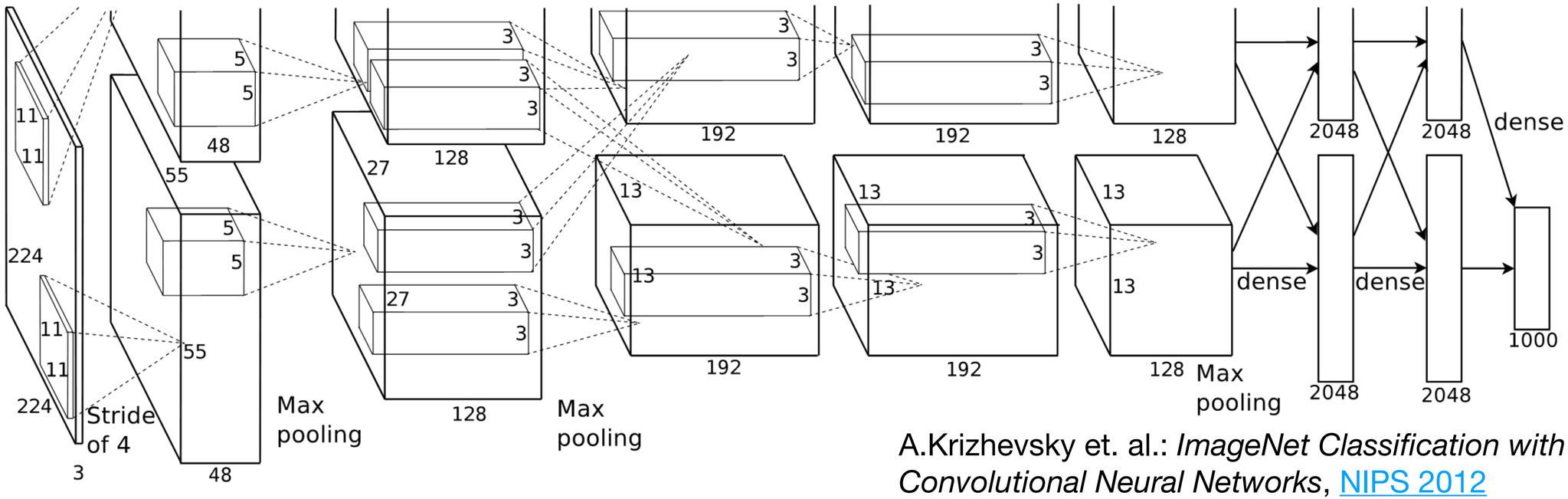
➡ Input is large image

➡ Multiple input channels if image is RGB

➡ Output is single number or vector of predictions

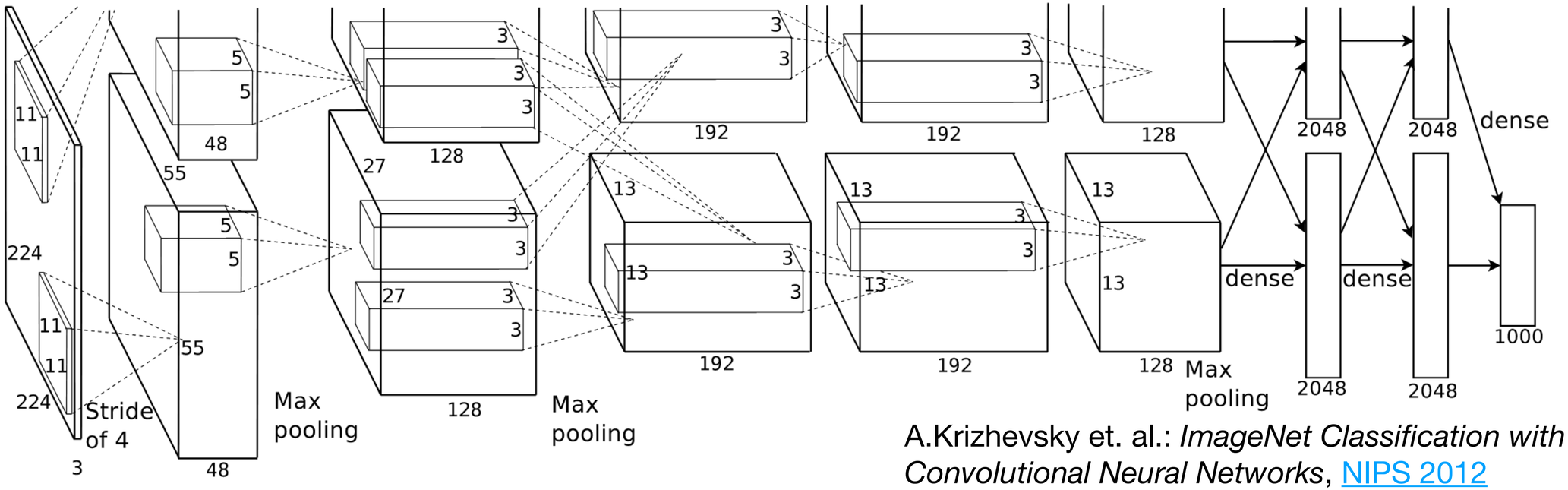
CNNs: AlexNet

- Archetypal for most convolutional networks
  - Convolutional segment
  - Dense segment



A.Krizhevsky et. al.: *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012

CNNs: AlexNet

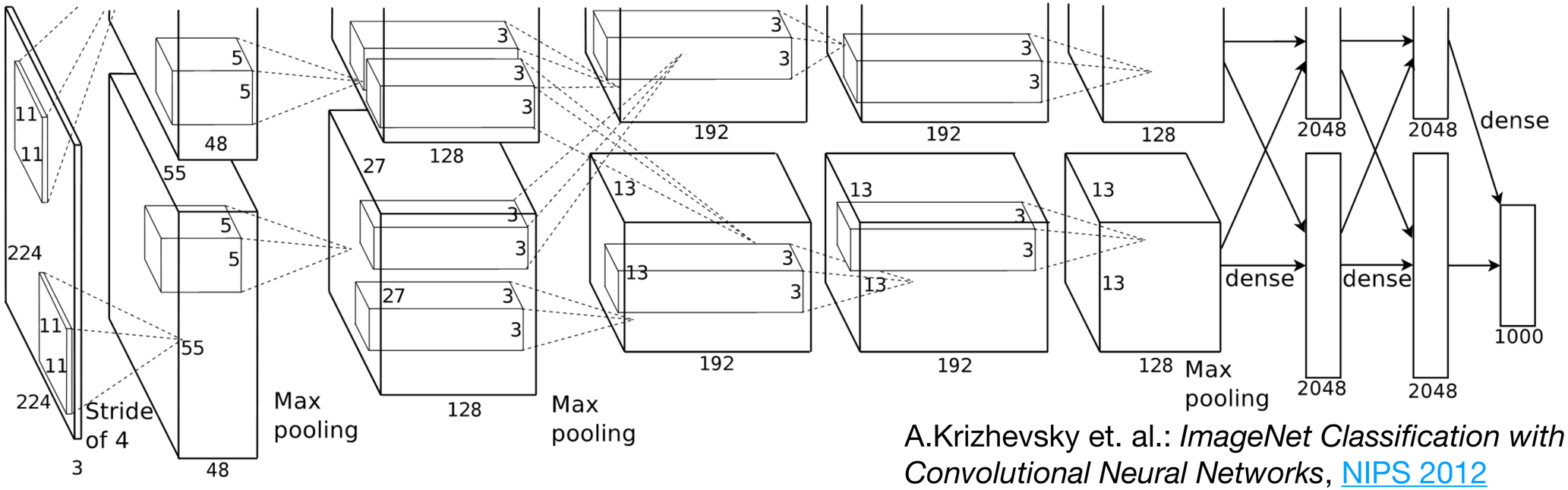


A.Krizhevsky et. al.: *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012

Convolutional segment

- First convolutional layers: extract basic features
- Later layers: recombine basic features into complex ones

CNNs: AlexNet

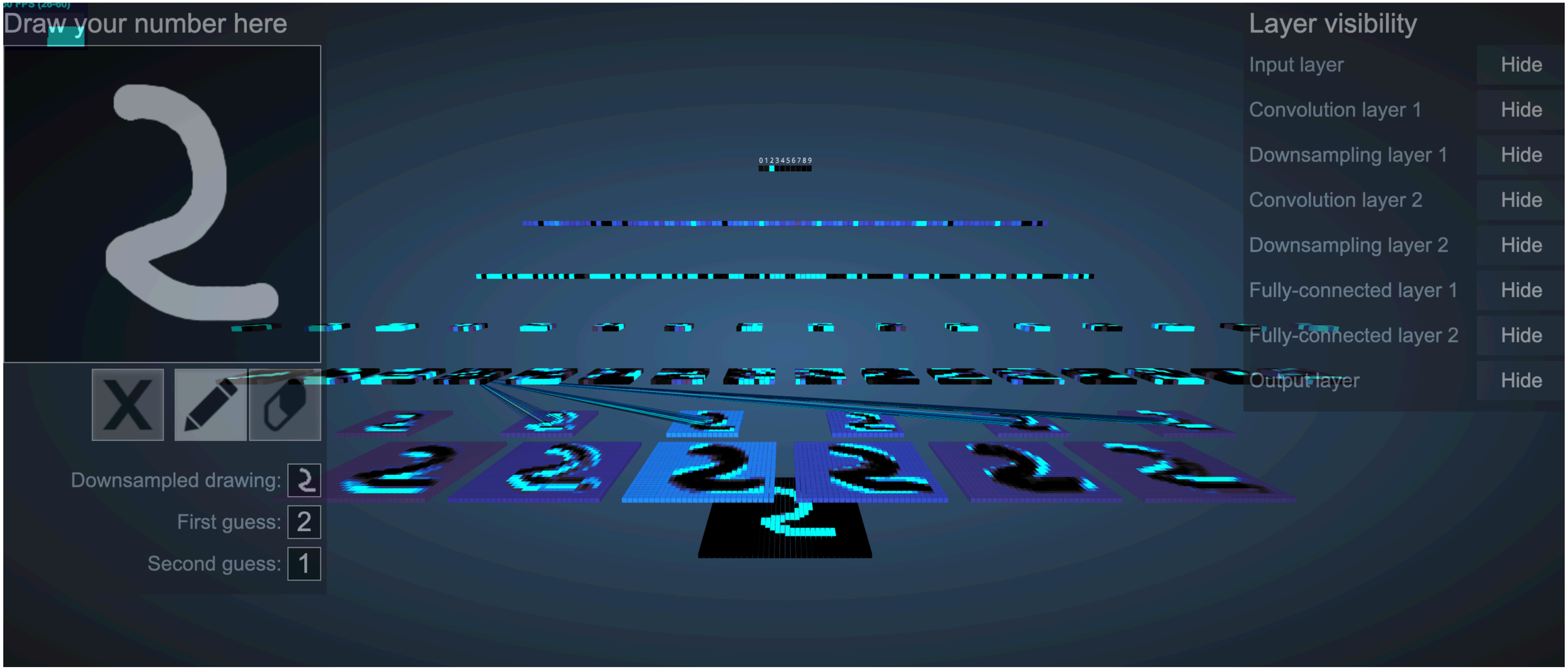


A.Krizhevsky et. al.: *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012

Dense segment

- Takes complex features as input
- Makes final prediction based on input features

Hands on Visualisation



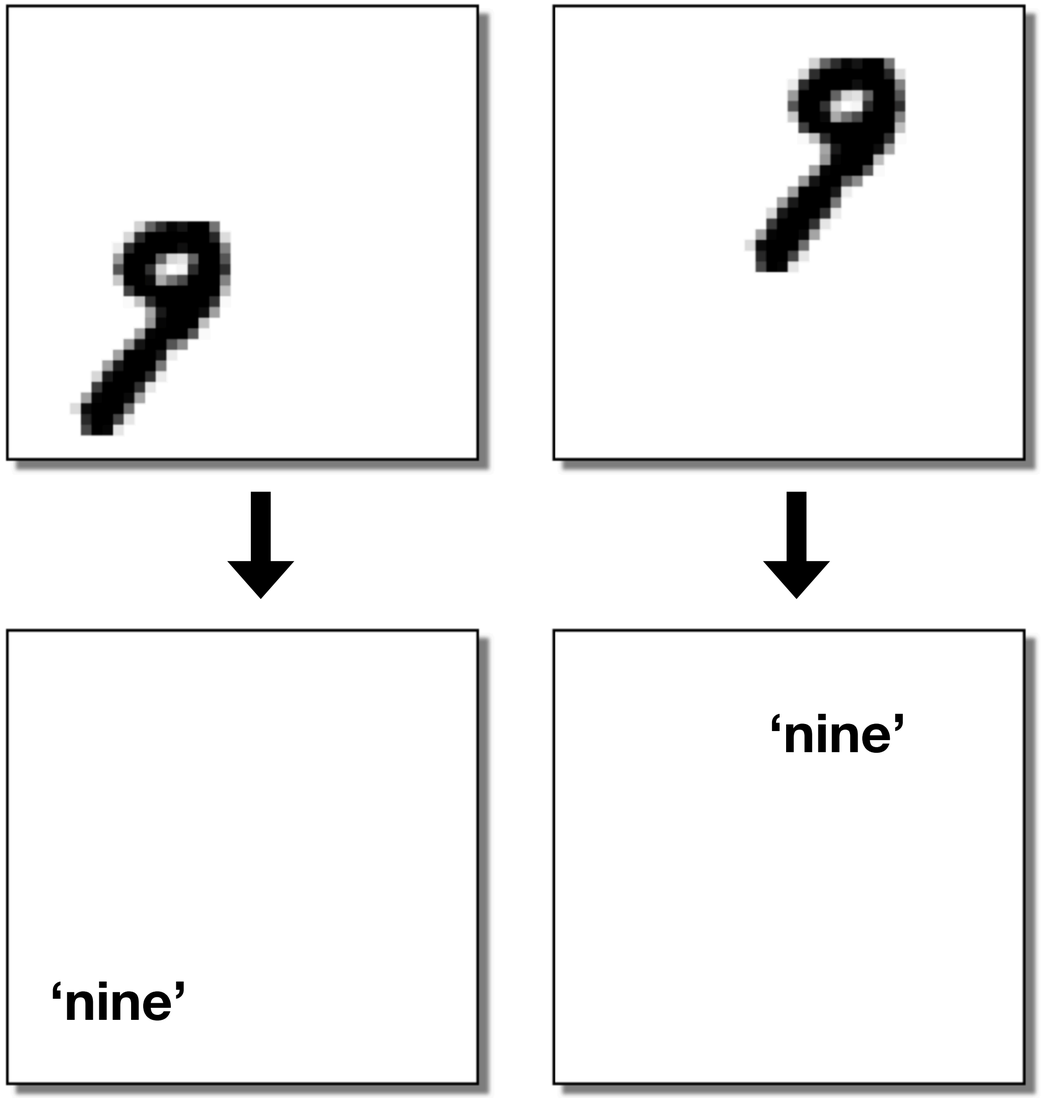
https://adamharley.com/nn_vis/cnn/3d.html

A Note on Feature Translation

- Want translation invariance
- Convolutional layer:
  - Translation **equivariant**
  - Same input at different position
  - Same output at different position

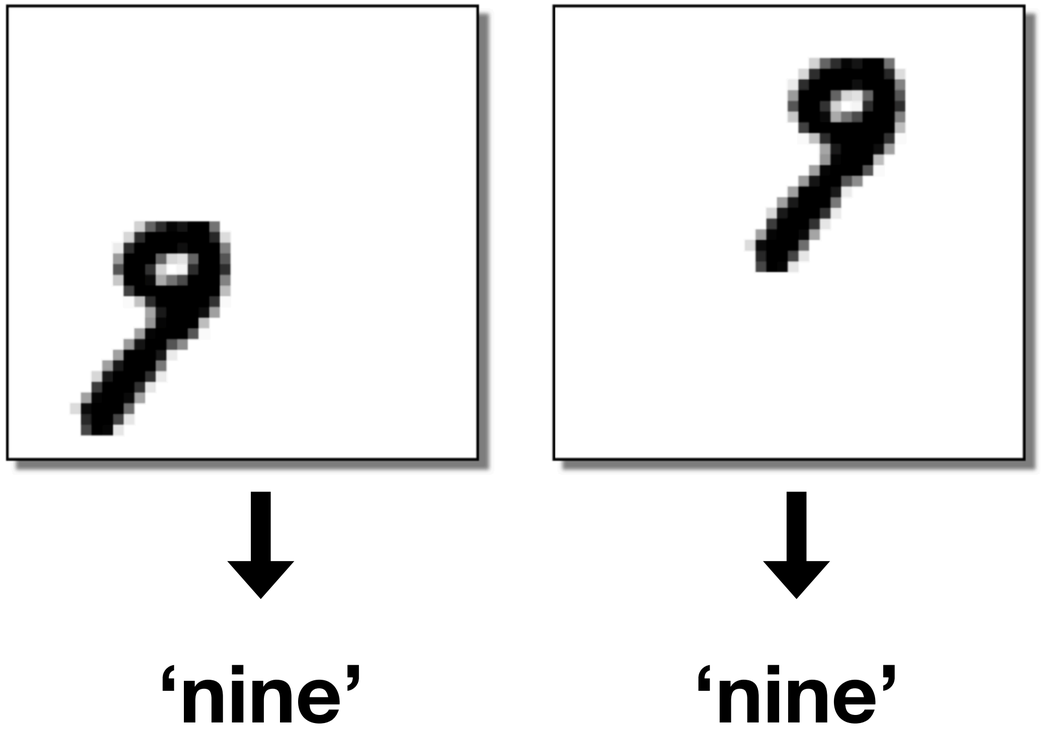
A Note on Feature Translation

- Want translation invariance
- Convolutional layer:
  - Translation **equivariant**
  - Same input at different position
  - Same output at different position

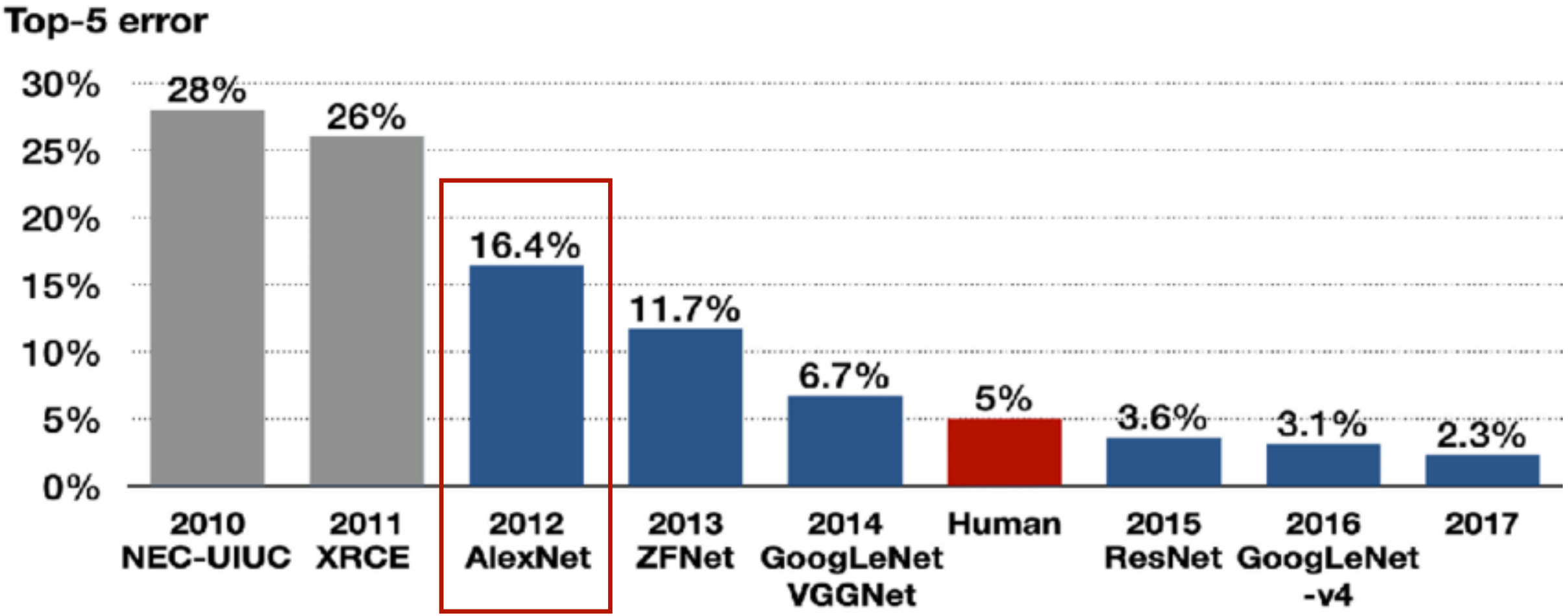


A Note on Feature Translation

- Want translation invariance
- CNN:
  - Translation **invariant**
  - Same input at different position
  - Same output

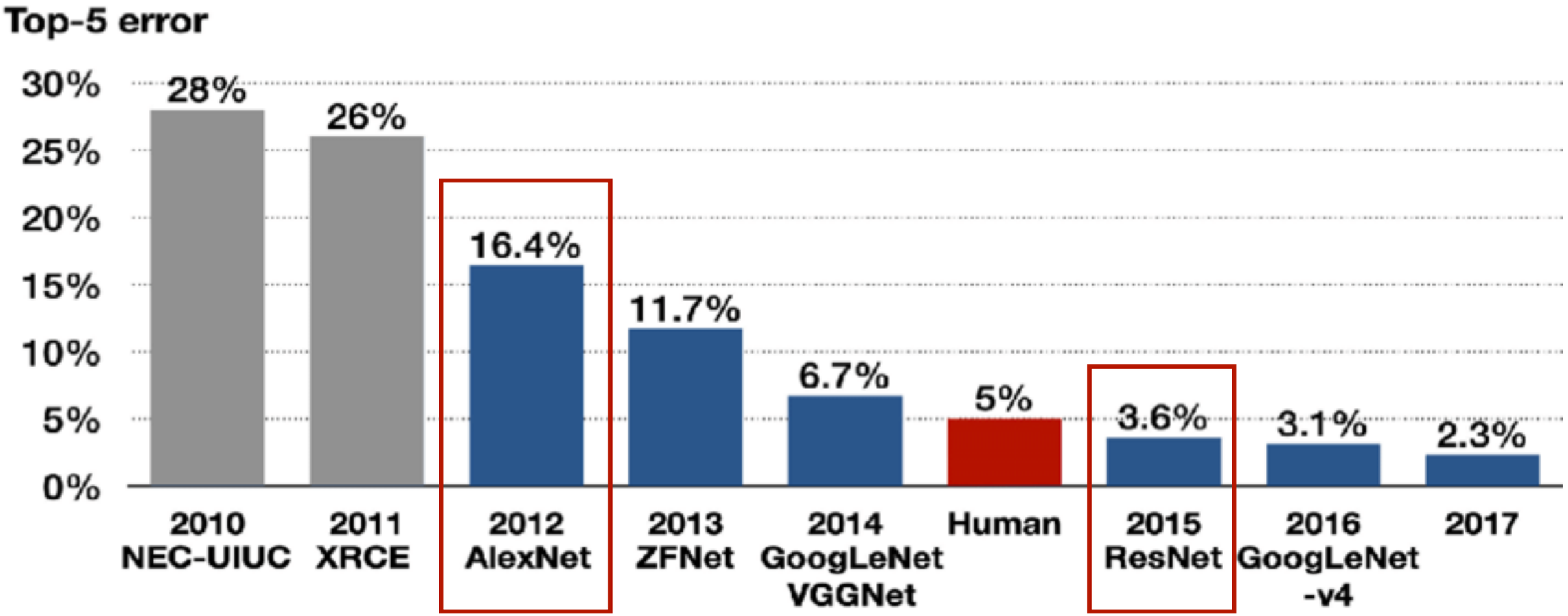


ImageNet Challenge



Dae-Young Kang et. al.: *Application of Deep Learning in Dentistry and Implantology*, 10.32542

ImageNet Challenge



Dae-Young Kang et. al.: *Application of Deep Learning in Dentistry and Implantology*, 10.32542

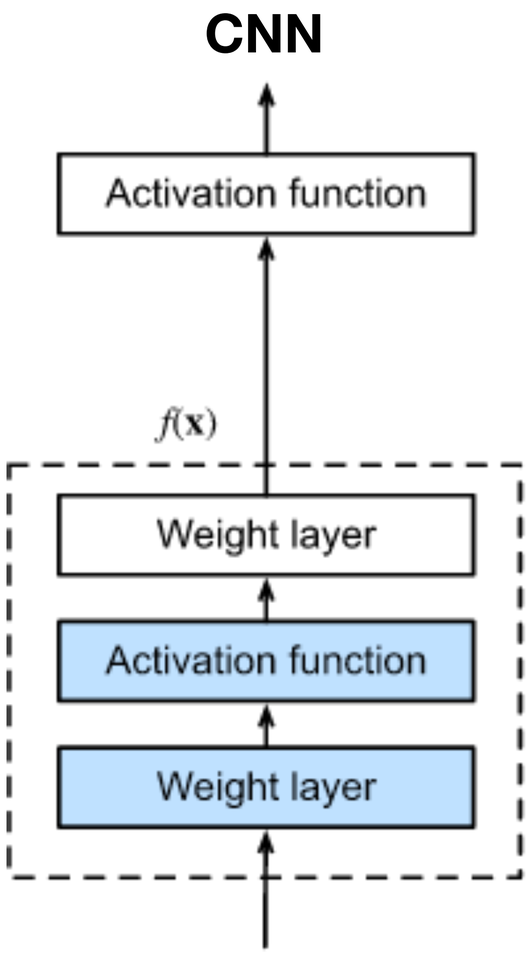
ResNet

- Standard CNNs have limited depth, too deep

➡ Vanishing gradients

  - Negative weights
  - Negative output
  - ReLU maps to zero
  - No gradient

- More likely to occur for deep networks



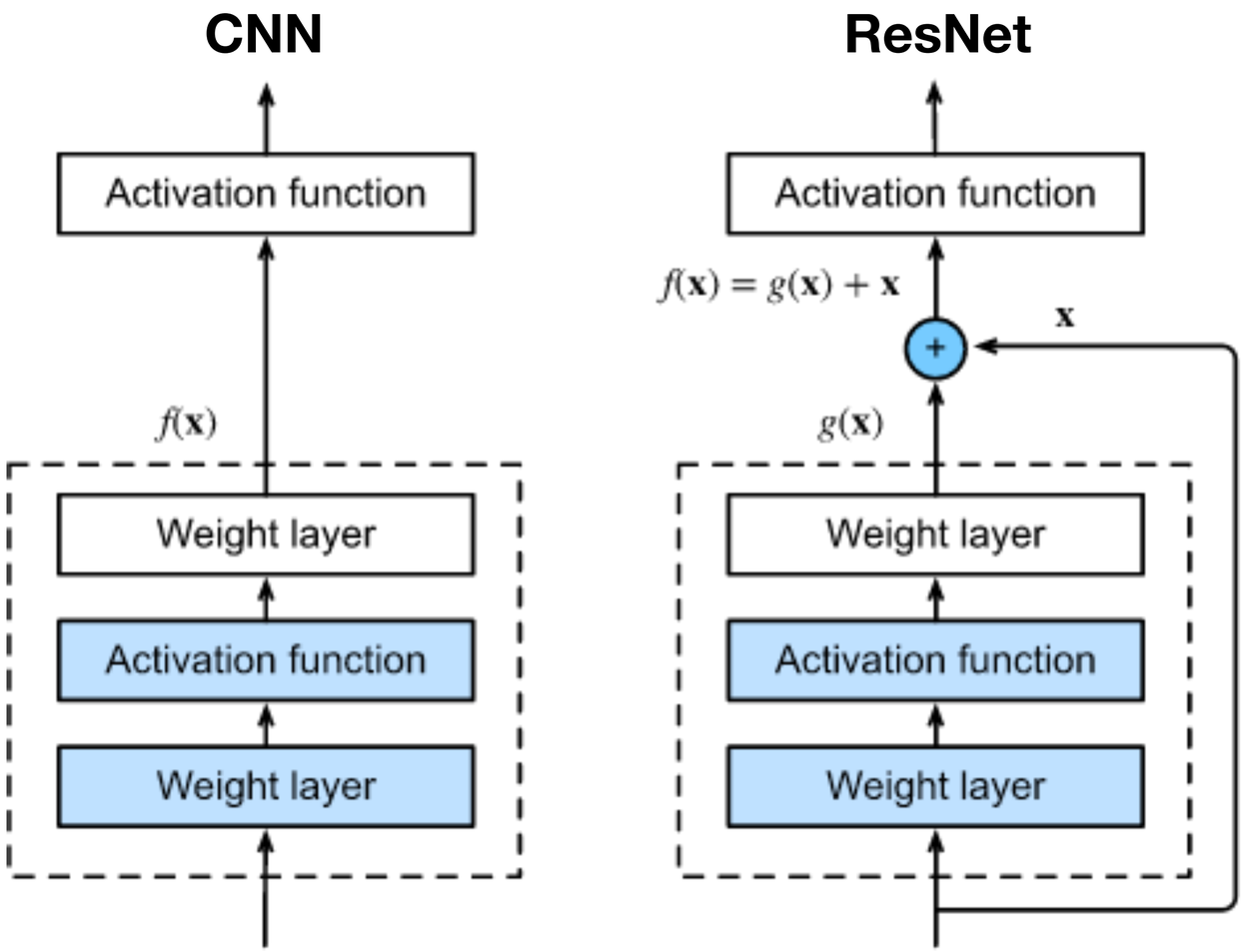
Diving into Deep Learning

ResNet

- Standard CNNs have limited depth, too deep

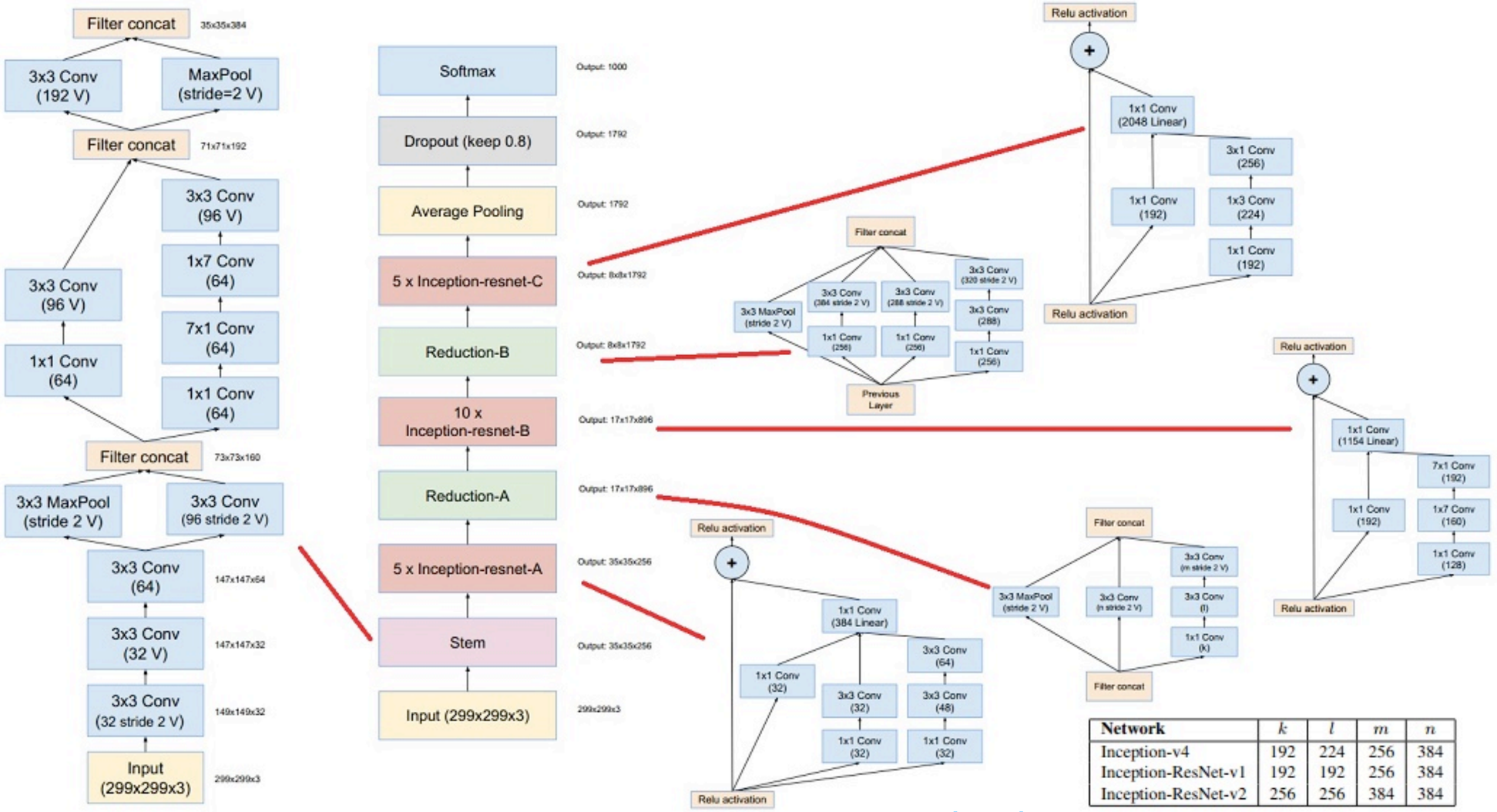
➡ Vanishing gradients

- Solution:
  - Residual/skip connections
  - Gradients can propagate through network easily
  - Conv. layers only learn modification to input



Diving into Deep Learning

Inception-ResNet



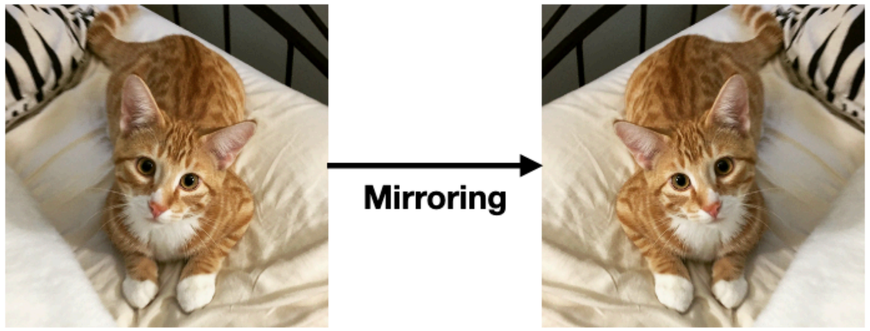
Network	k	l	m	n
Inception-v4	192	224	256	384
Inception-ResNet-v1	192	192	256	384
Inception-ResNet-v2	256	256	384	384

- ResNets can become very deep

yeephycho

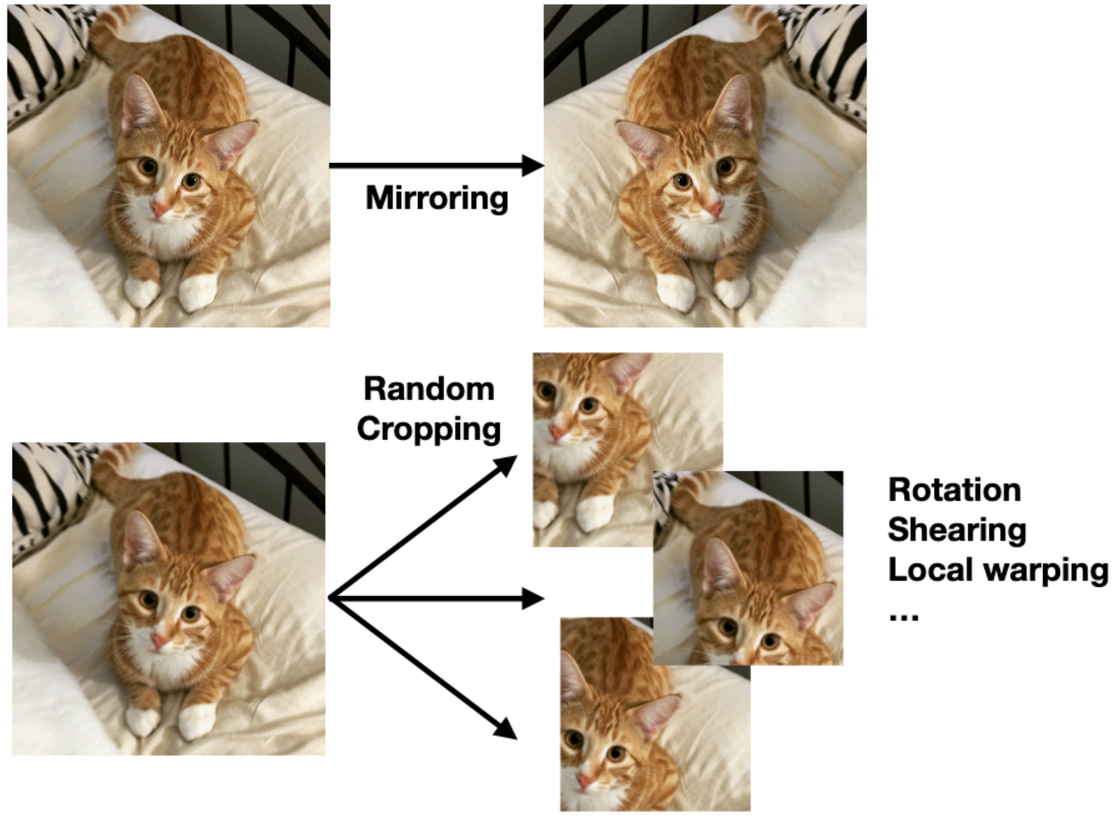
Data Augmentation

- We want network to exploit symmetries/repeated features
  - Introduce rotated/mirrored/shifted copies of data



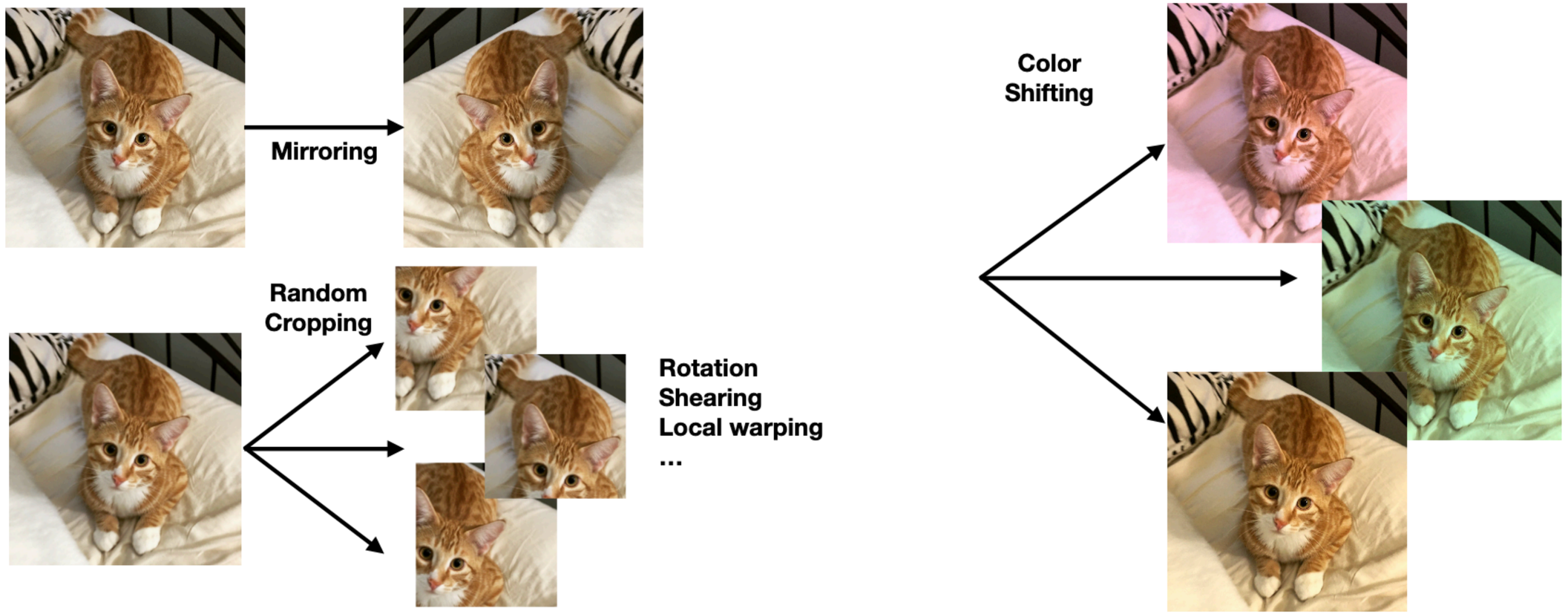
Data Augmentation

- We want network to exploit symmetries/repeated features
  - Introduce rotated/mirrored/shifted copies of data



Data Augmentation

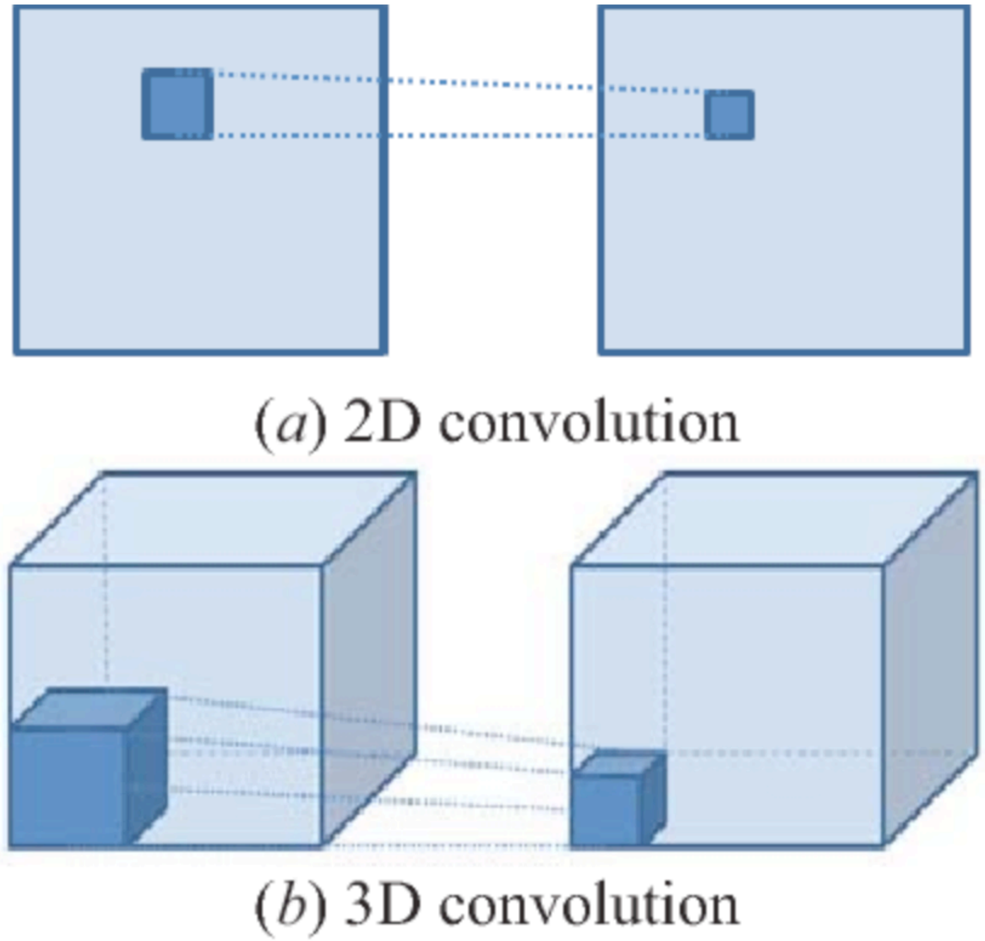
- We want network to exploit symmetries/repeated features
  - Introduce rotated/mirrored/shifted copies of data



Outlook: Beyond Images

Convolutions not limited to 2D

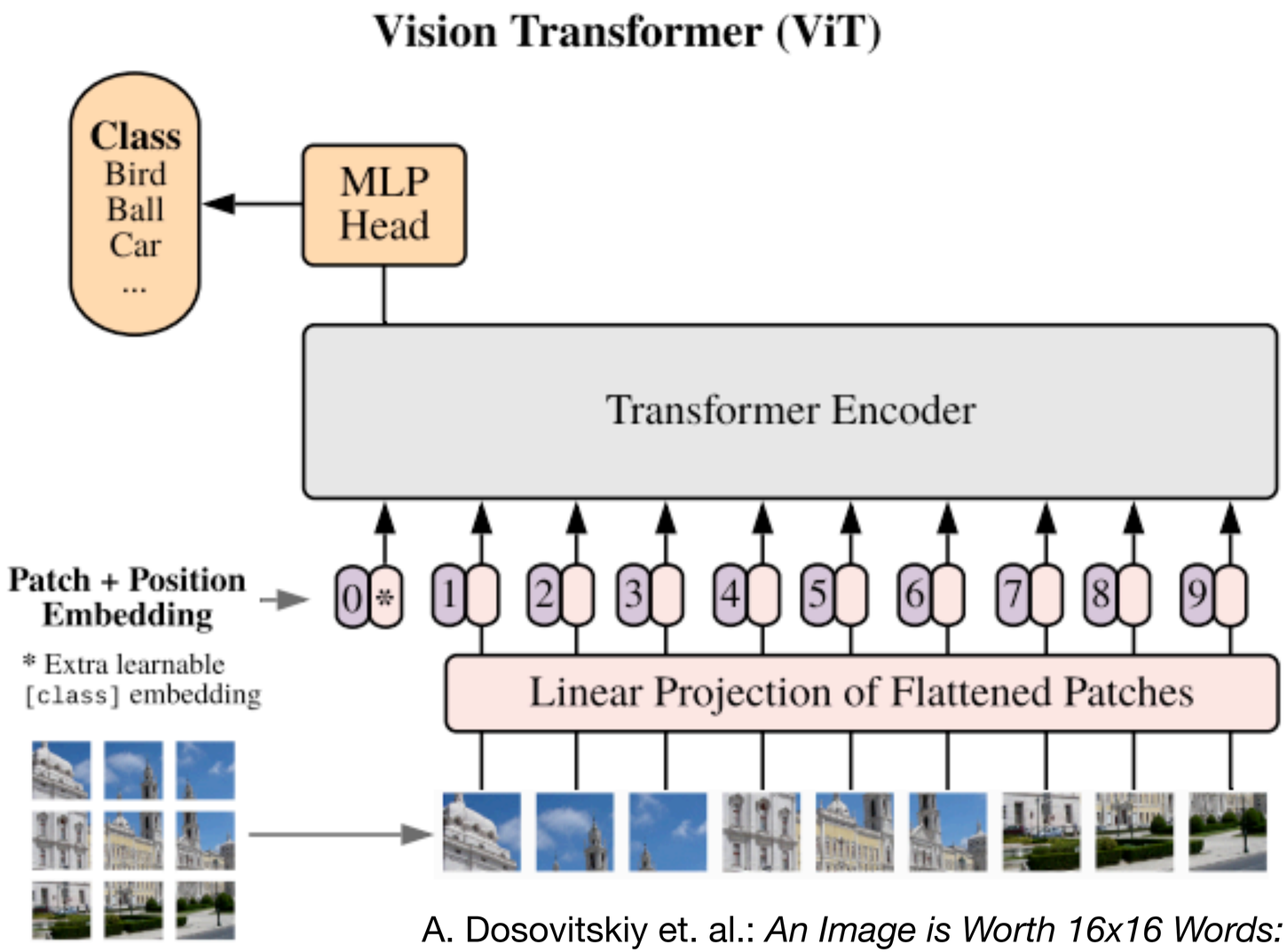
- Concept can be extended to 3D data as well
  - 3D output, 3D kernel, 3D strides, ….
  - Volumetric data processing
- Similar for 1D
  - Time sequence processing



Fan et. al.,Lung nodule detection based on 3D convolutional neural networks, document/88

Outlook: Transformers

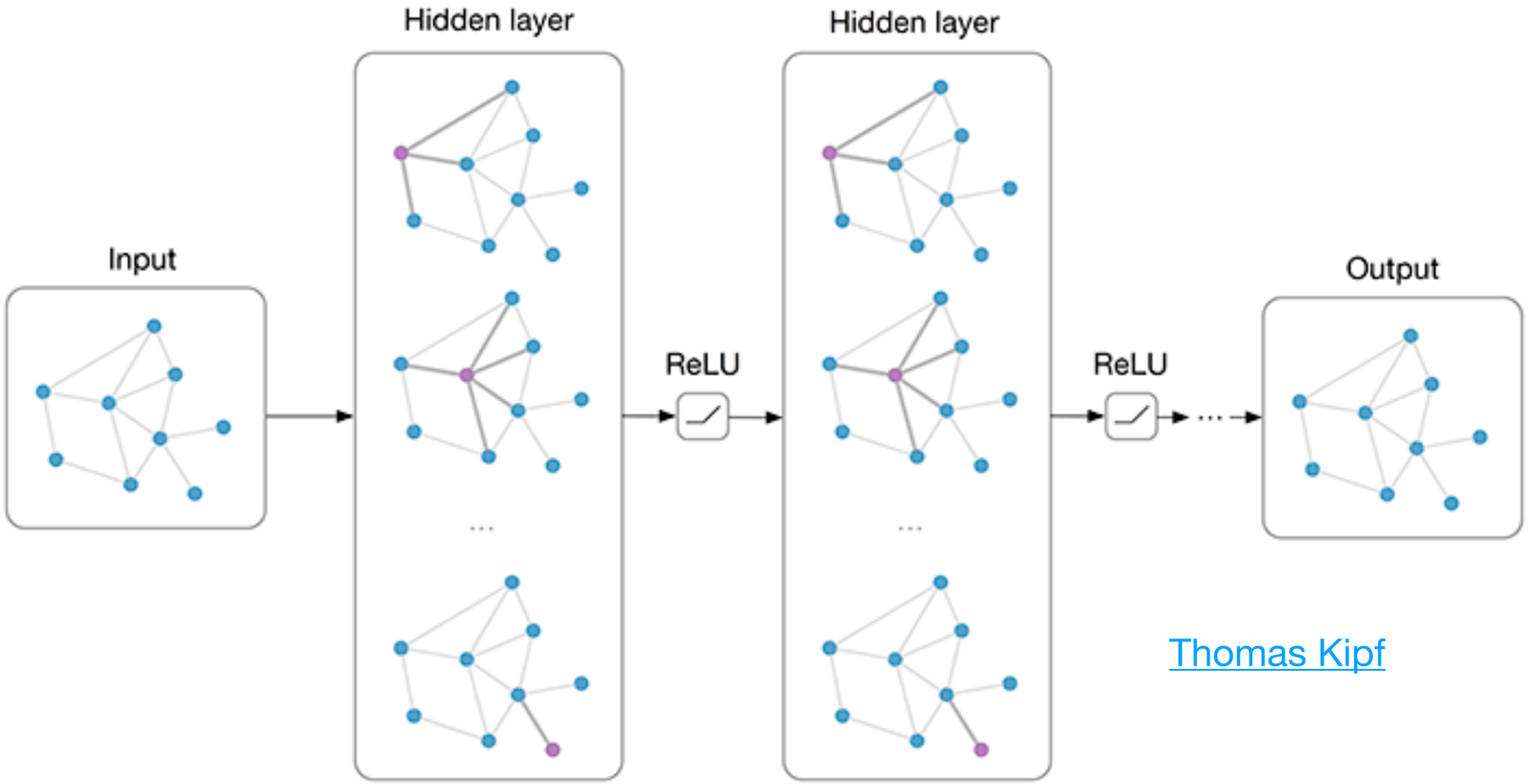
- Vision Transformers
- Can learn local connections on their by themselves
- Reach similar/better accuracy as CNNs
- More in advanced course



A. Dosovitskiy et. al.: *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 2010.11929

Outlook: Graph Convolutions

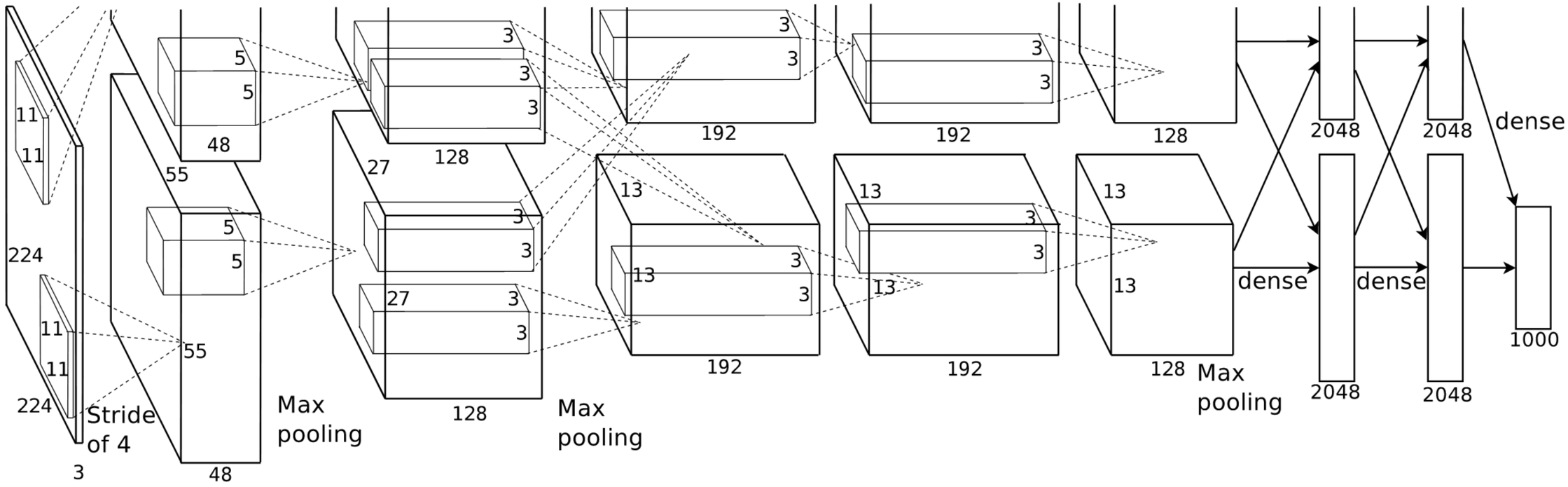
- Graph CNNs
- Operate on graphs with arbitrary node positions
- Do not require fixed structures like CNNs
- More in advanced course



Thomas Kipf

Conclusion

- CNNs are vital part of image processing networks
- Workhorse of any NN that operates on regular structures



Exercises

- Ex 1: https://colab.research.google.com/drive/17m0A8zmDrd12p0qFd06dZj9DR0EYhrr1?usp=sharing
- Solution 1: https://colab.research.google.com/drive/1ca4f5fQhr4e9_pl6gtidka5DcrGSLZBx?usp=sharing

- Ex 2: https://colab.research.google.com/drive/1Tu_clxGNv5cEWx-SL-R_XIefic091Rxw?usp=sharing
- Solution 2: https://colab.research.google.com/drive/1u_SaseBfXtVOijBmNLO6lgAhod3yKekn?usp=sharing

- Challenge: https://colab.research.google.com/drive/1U520eJ9NerQVVYcqT70Shj0LhC4eAQHk?usp=sharing