CNNs and key concepts

Callum Goddard

School of Elec Eng & Computer Science
Queen Mary University of London

(Thanks to Dan Stowell and Huy Phan)

Today

- Classification and regression
- Components of neural networks
- Basics of training a NN
- Convolutional neural nets (CNNs)
- Typical CNN architectures for audio work
- Receptive fields

What is machine learning?

What is machine learning?

$$y = f_\theta(x)$$

$$\mathrm{x} \xrightarrow{f_\theta} \mathrm{y}$$

What is machine learning?

$$y = f_\theta(x)$$

$$\mathrm{x} \xrightarrow{f_\theta} \mathrm{y}$$

Parameters θ to be chosen with the aid of data

What is deep learning?

What is deep learning?

$$y = f'''''_{\theta'''''}(f'''_{\theta'''}(f''_{\theta''}(f'_{\theta'}(f_\theta(x)))))$$

$$x \xrightarrow{f_\theta} \cdot \xrightarrow{f'_{\theta'}} \cdot \xrightarrow{f''_{\theta''}} \cdot \xrightarrow{f'''_{\theta'''}} \cdot \xrightarrow{f'''''_{\theta'''''}} y$$

Regression and classification

Regression

$x \xrightarrow{f_\theta} 21.7$

Classification

$x \xrightarrow{f_\theta} \boxed{\text{cat}}$ dog horse

Regression and classification

Regression

$$x \xrightarrow{f_\theta} 21.7$$

Classification (binary)

$$x \xrightarrow{f_\theta} 0 \text{ or } 1$$

Regression and classification

Regression

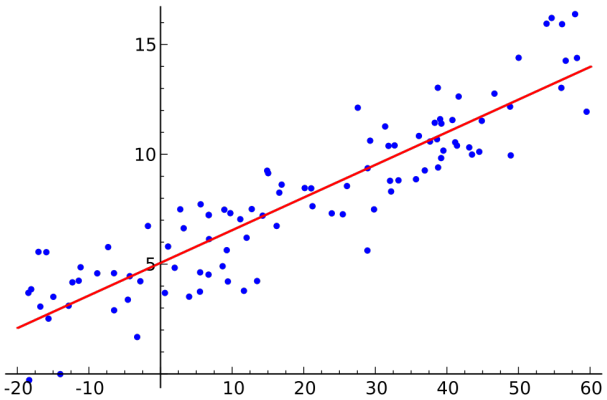
$$x \xrightarrow{f_\theta} 21.7$$

Classification (binary, prob'stic)

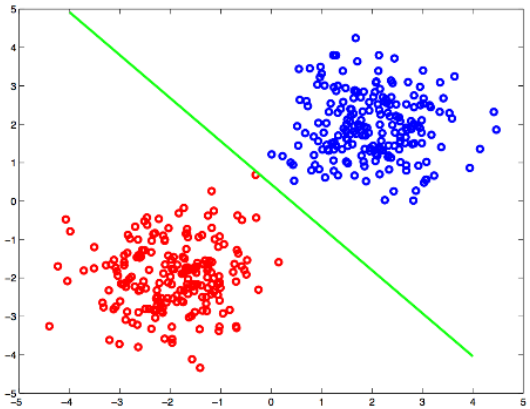
$$x \xrightarrow{f_\theta} 0.712$$

Regression and classification

Regression



(Binary) classification



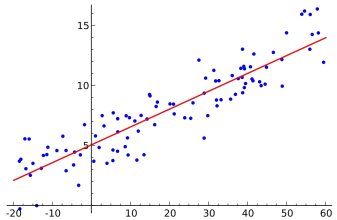
Loss functions

Also known as: cost function, objective function.
"Defining how (un)happy you will be with the result"

$L(y, y^*) \in \mathbb{R}^+$

[Python Example: `https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics`]

Linear regression

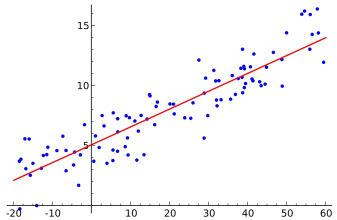


$$y = \beta_0 + \beta_1 x$$

$$Y = X\beta$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Linear regression

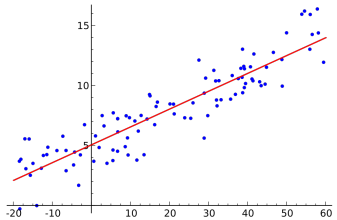


$$y = \beta_0 + \beta_1 x$$

$$Y = X\beta$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Linear regression



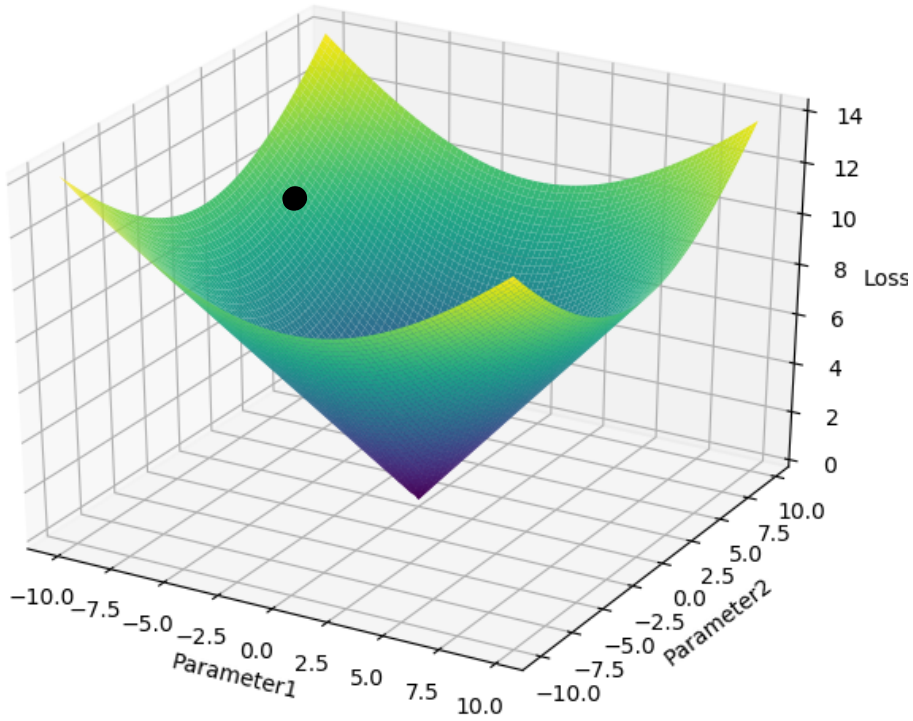
$$y = \beta_0 + \beta_1 x$$

$$Y = X\beta$$

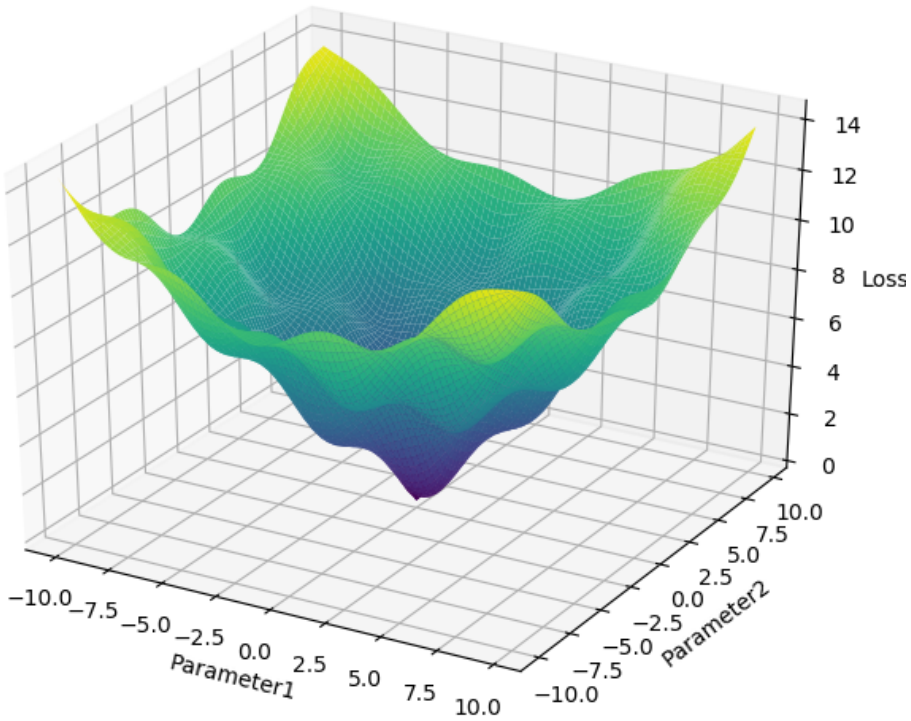
$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Loss surfaces

What about the loss as a function of the *parameters*?
$L(f_\theta(x), y^*)$

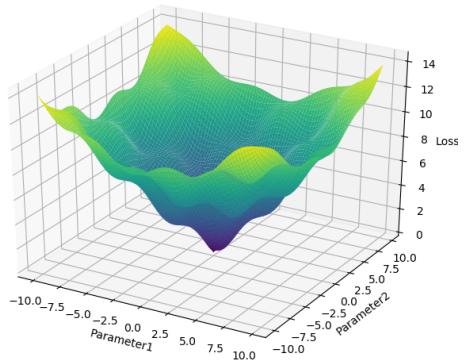


Loss surfaces

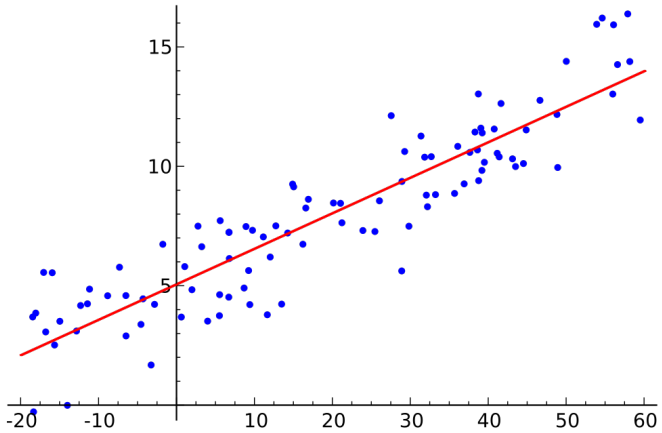


Gradient descent

1. Pick random initial parameters θ
2. Evaluate gradient $\frac{\partial L}{\partial \theta}$ at θ
3. If gradient is of size zero, stop here.
4. Update the parameters:
$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial L}{\partial \theta_i}$$
5. Go to step 2.



Linear regression



[Python example: `https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html`]

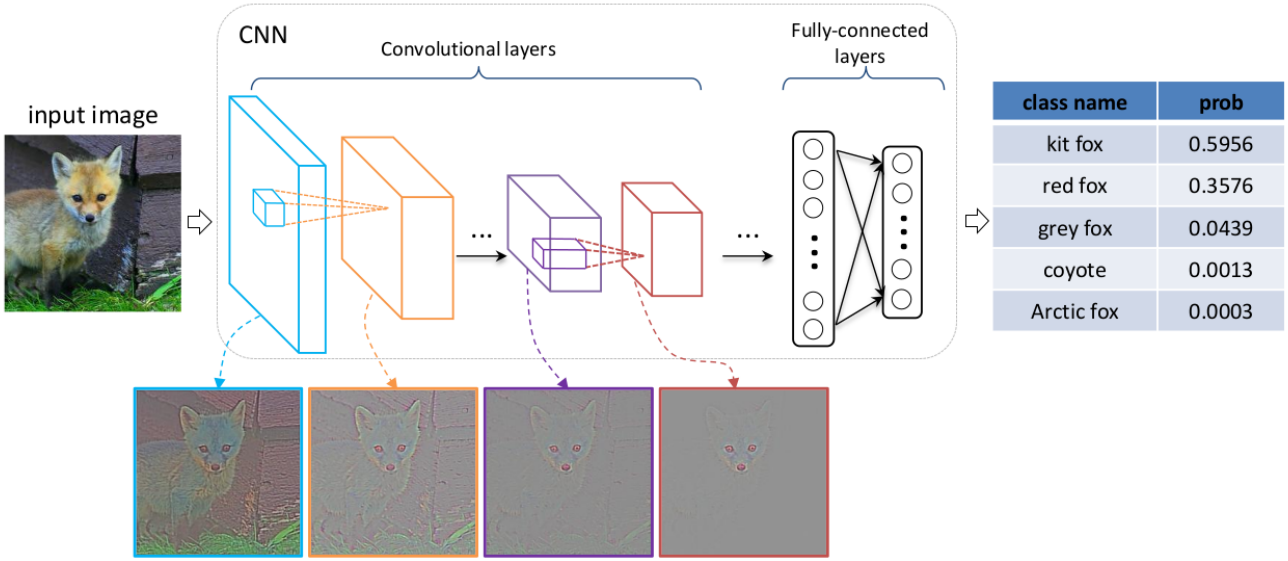
Deep learning

Deep learning

$$y = f''''_{\theta''''}(f'''_{\theta'''}(f''_{\theta''}(f'_{\theta'}(f_\theta(x)))))$$

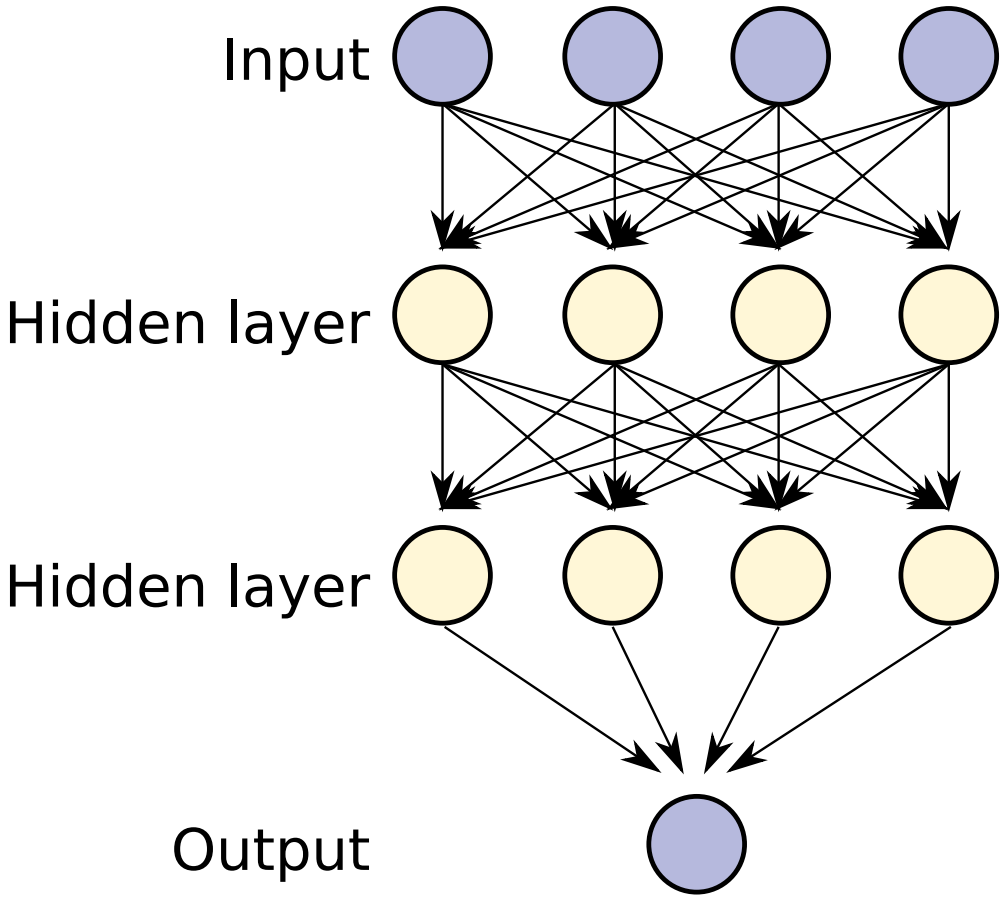
$$\mathrm{x} \xrightarrow{f_\theta} \cdot \xrightarrow{f'_{\theta'}} \cdot \xrightarrow{f''_{\theta''}} \cdot \xrightarrow{f'''_{\theta'''}} \cdot \xrightarrow{f''''_{\theta''''}} \mathrm{y}$$

You will hear a lot about "layers"



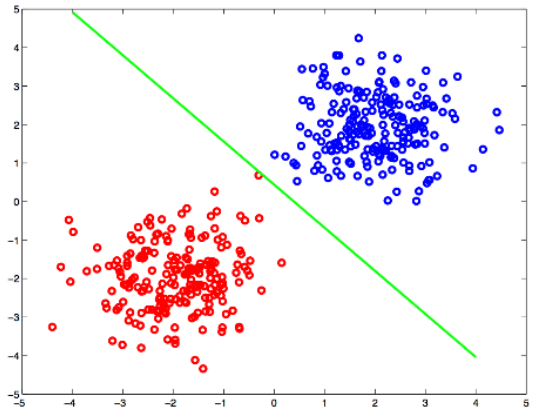
Start simple

Each *node* here is a *scalar* value

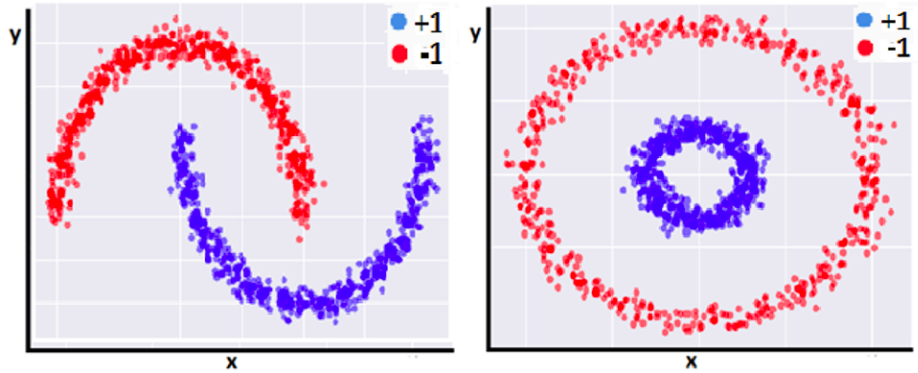


Nonlinearities

Linear decision boundary



Non-linear decision boundary



What can our NN *do* to the data values?
Rotate? Scale? Bend? Warp?

Nonlinearities

Life without nonlinearity is unfulfilling

Life without nonlinearity is unfulfilling

Why?

Every layer f_n involves multiply-and-sum of the previous layer's output.

This linear part of the layer can be written as a matrix projection:

$$\mathrm{Out}_n = F_n \cdot \mathrm{Out}_{n-1}$$

A sequence of matrix projections

$$Y = F_5 \cdot F_4 \cdot F_3 \cdot F_2 \cdot F_1 \cdot X$$

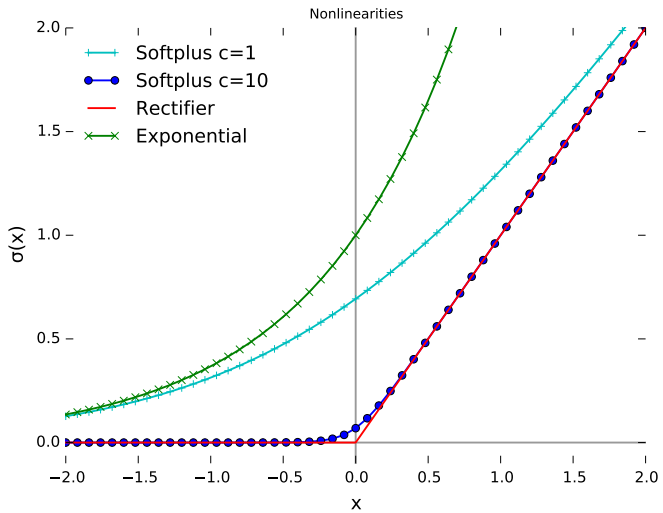
can equivalently be expressed by... (what?)

Nonlinearities

For NNs, 'Nonlinearities' means pointwise scalar functions.

Common nonlinear activation functions: *Sigmoid*, *Tanh*, *Rectified Linear Unit (ReLU)*, *Leaky ReLU*, *Parametric ReLU*, *Swish*, *Softplus*, etc.

For example: $ReLU(x) = max(0, x)$



A "layer", then

$$x \xrightarrow{f^1_{\theta^1}} \cdot \xrightarrow{f^2_{\theta^2}} \cdot \xrightarrow{f^3_{\theta^3}} \cdot \xrightarrow{f^4_{\theta^4}} y$$

A "layer", then

$$x \xrightarrow{*W_1} \xrightarrow{\sigma_1(\cdot)} \cdot \xrightarrow{*W_2} \xrightarrow{\sigma_2(\cdot)} \cdot \xrightarrow{*W_3} \xrightarrow{\sigma_3(\cdot)} \cdot \xrightarrow{*W_4} \xrightarrow{\sigma_4(\cdot)} y$$

Why deep learning?

Multi-layer learning can do more than simple logistic regression.

But still: why layers? Why not one cleverly-designed single function?

$$x \xrightarrow{f_\theta} y$$

Why deep learning? Why layers?

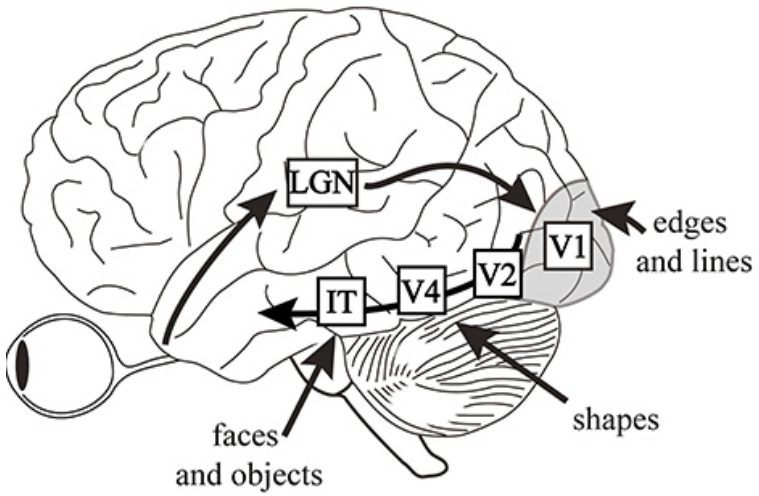
Biological motivation:

- brains seem to do it

Practical usefulness, e.g.:

- Train layer-wise
- Replace individual layers

Visual cortex



[1]

[1] Adapted from Herzog & Clarke, 2014

Why deep learning? Why layers?

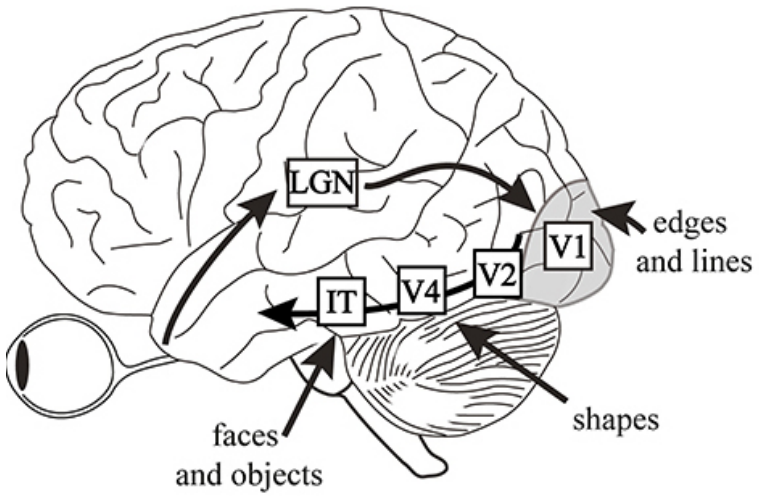
Biological motivation:

- brains seem to do it

Practical usefulness, e.g.:

- Train layer-wise
- Replace individual layers

Visual cortex



[1]

[1] Adapted from Herzog & Clarke, 2014

Why deep learning? Why layers?

Theoretical motivations:

1. **Universal approximation theorem** (Cybenko 1989, Hornik 1991):
   "a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, iff the activation function is not polynomial"

   (NB not about training)

2. "Backward feature correction", Allen-Zhu & Li (2020):
   provable that DL can learn some functions efficiently which NO known single-layer method can.
   Training: SGD (stochastic gradient descent), in poly(d) time using poly(d) samples.

Why deep learning? Why layers?

Theoretical motivations:

1. **Universal approximation theorem** (Cybenko 1989, Hornik 1991):
   "a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, iff the activation function is not polynomial"

   (NB not about training)
2. "Backward feature correction", Allen-Zhu & Li (2020): provable that DL can learn some functions efficiently which NO known single-layer method can.
   Training: SGD (stochastic gradient descent), in poly(d) time using poly(d) samples.

Why deep learning? Why layers?

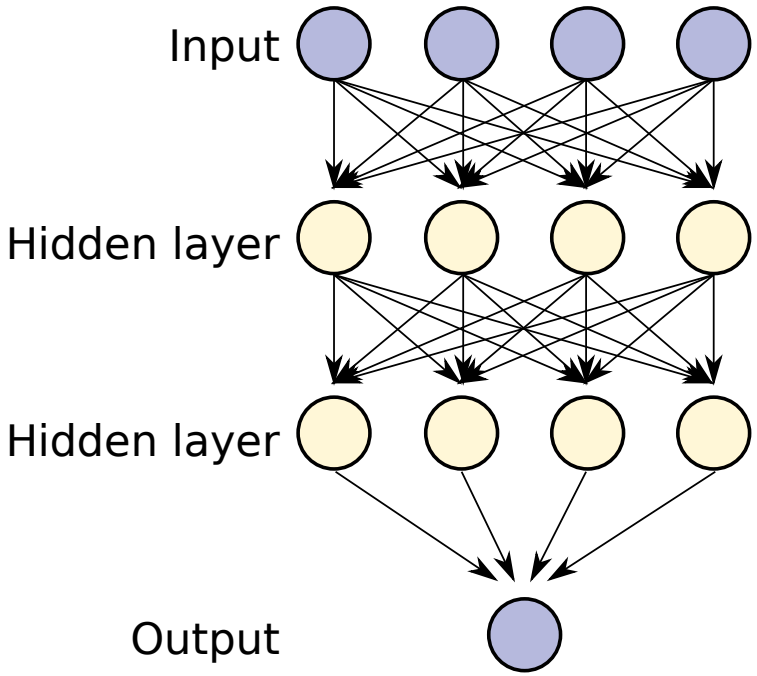
And yet:

- How many layers to choose? not clear
- What do we expect the latent features to look like? not sure

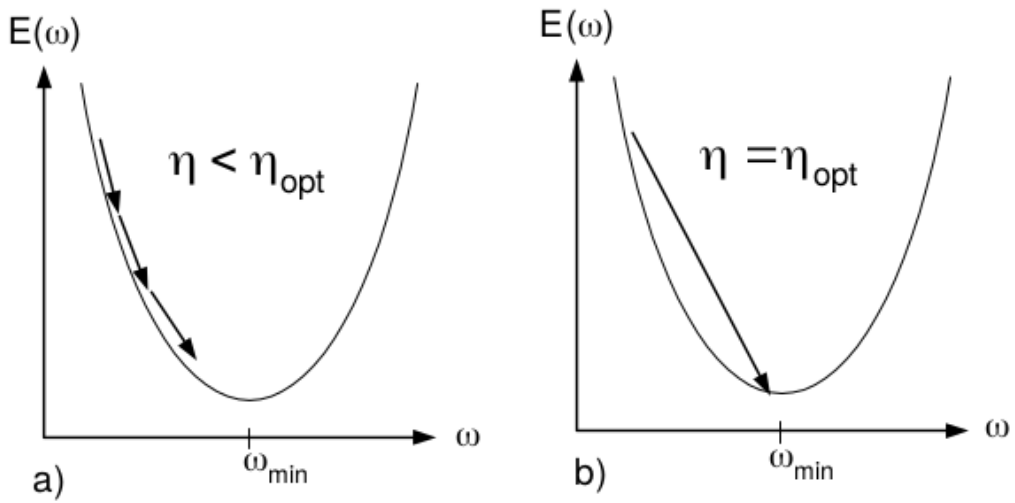
Backpropagation of errors, "Backprop"

Using chain rule and backward recurrence, we can update the parameters ('weights')

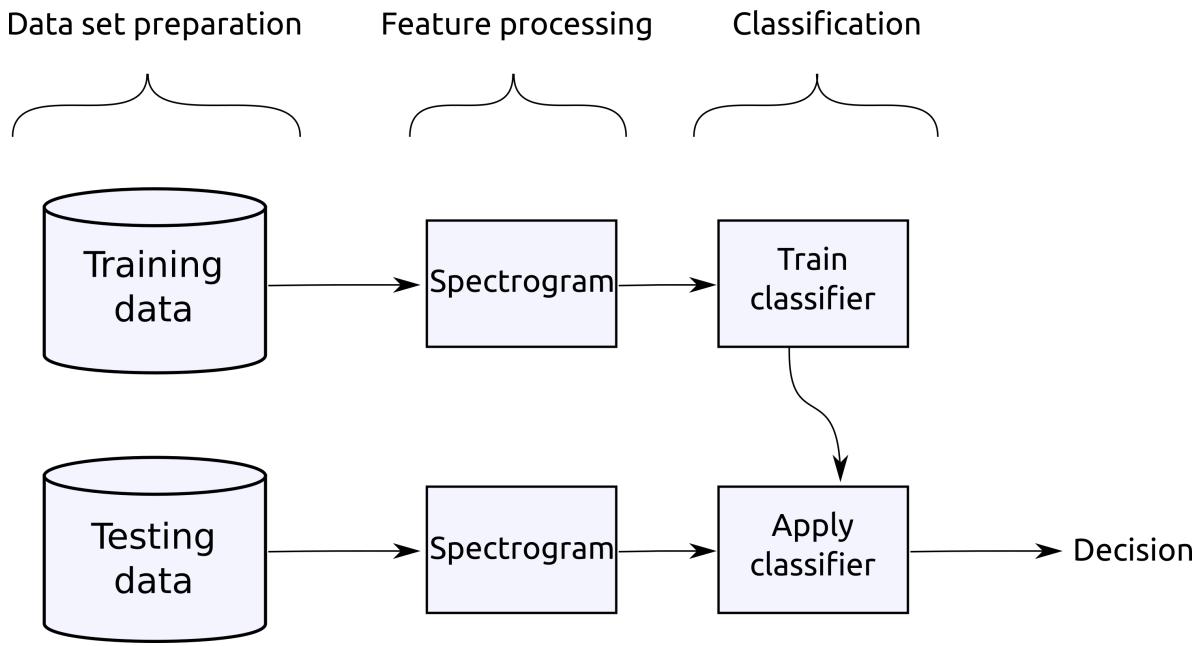
$$\frac{\partial L}{\partial \theta_n} = \frac{\partial f_n}{\partial \theta}(\text{out}_{n-1}, \theta_n)\frac{\partial L}{\partial \text{out}_n}$$



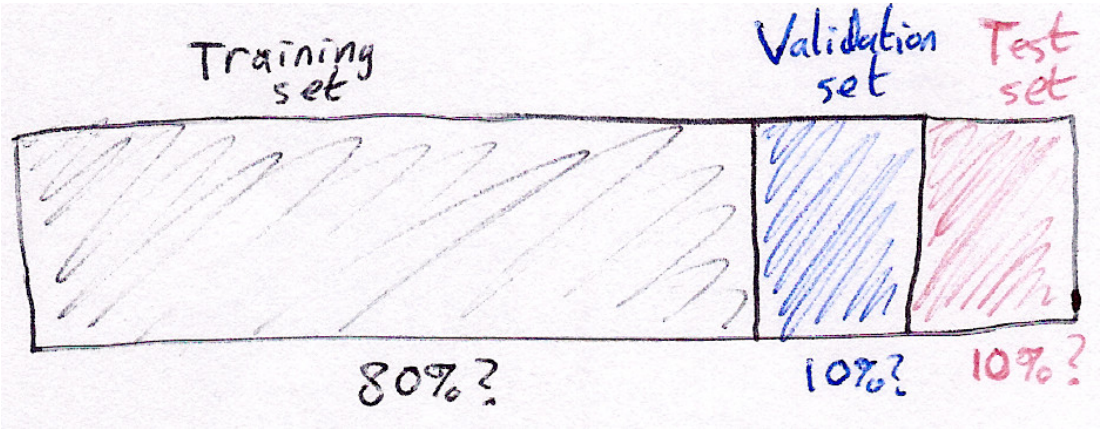
Learning rate



Divide your data: training, test



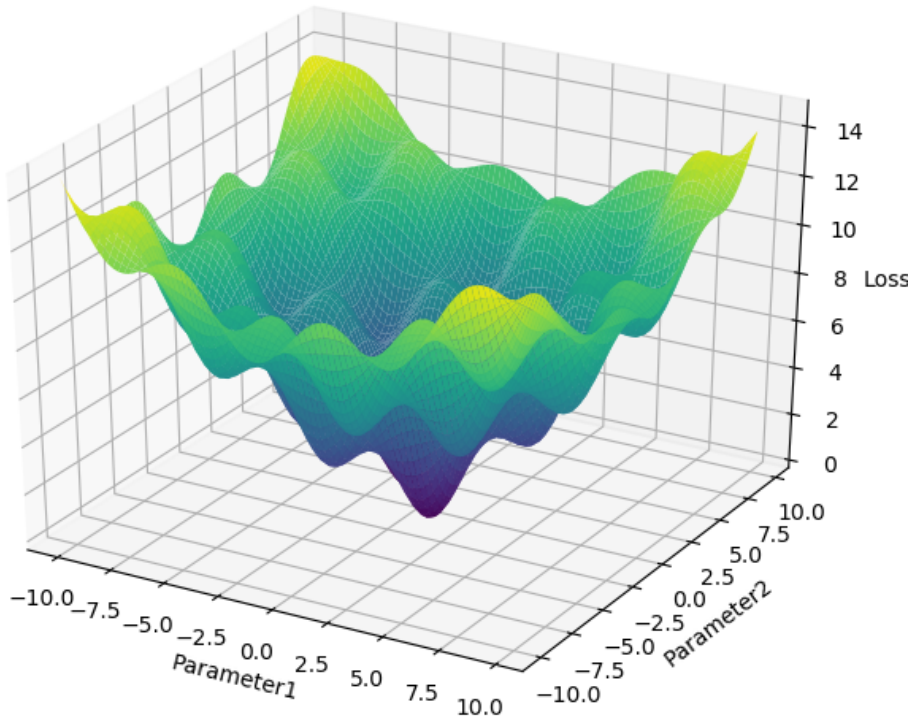
Divide your data: training, validation, test



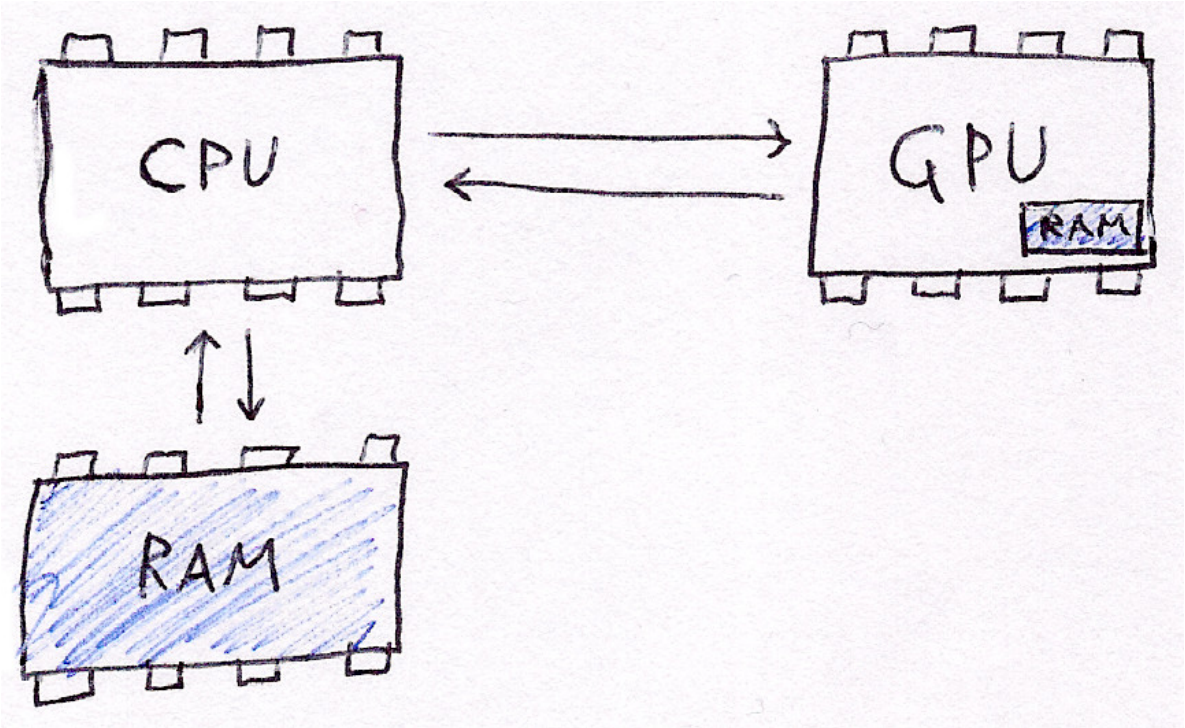
Note: it's vital that you do not use your test data at all,
until all your training, tweaking, training, tweaking, ... is finished.

Divide your data even more

Many problems are too hard for simple gradient descent.



Divide your data even more: my GPU can't hold it all

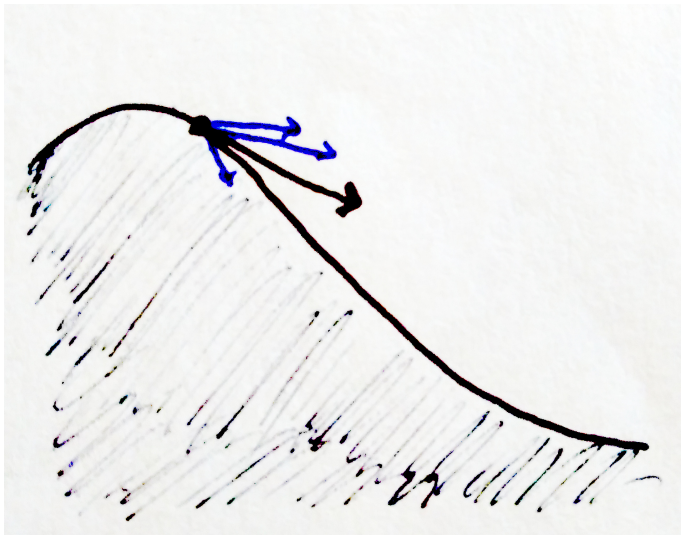


...so divide training data into 'minibatches'

Minibatches and gradient

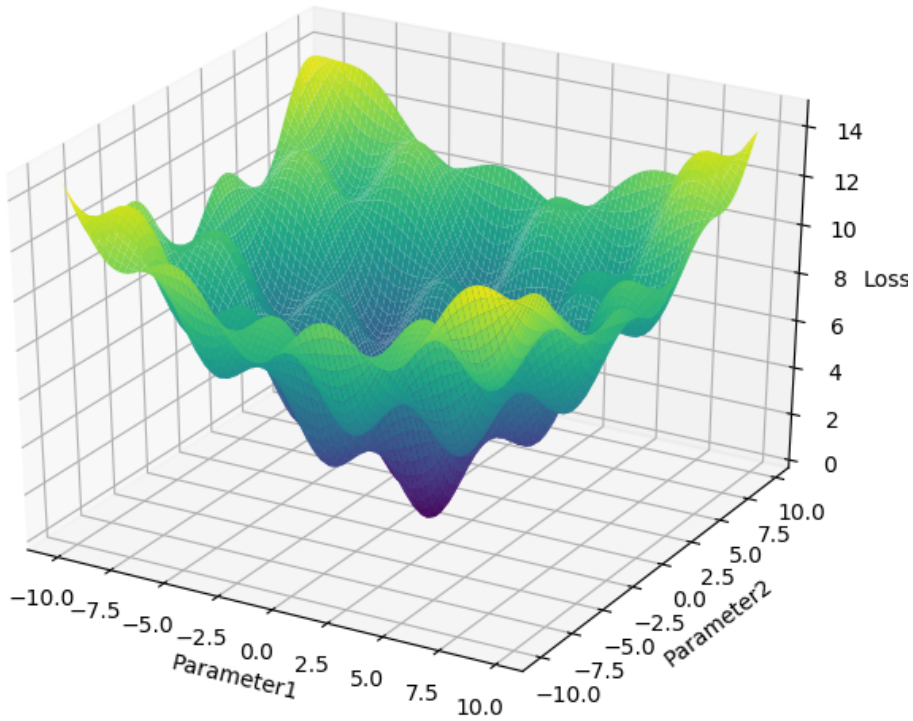
Minibatch = a subsample from the training dataset

$\partial L_{\text{minibatch}}$ = an estimate of ∂L



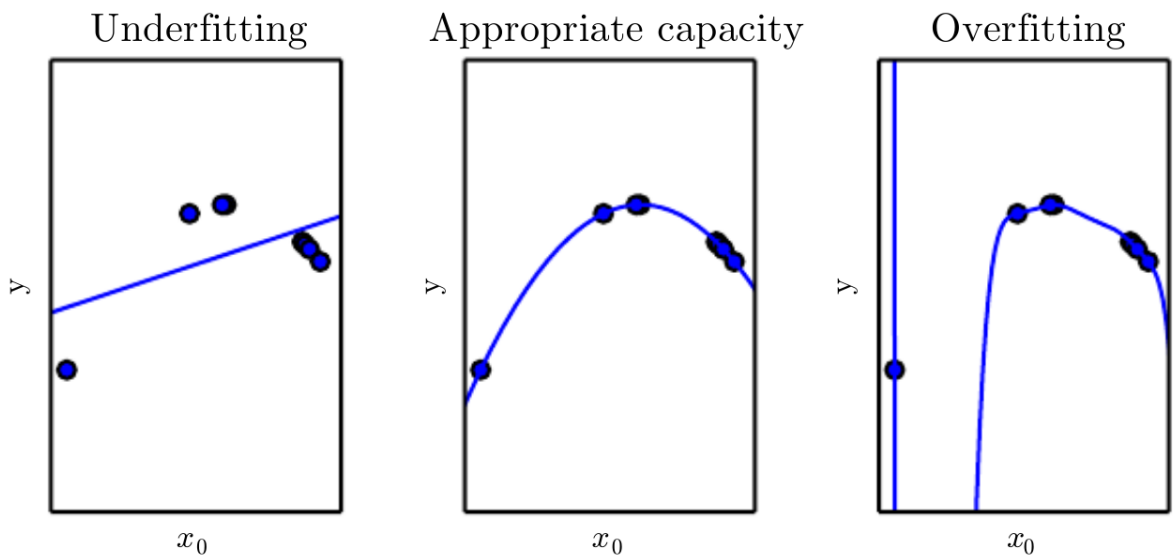
Q: Is it a good estimate? Biased? Highly variable?

Stochastic gradient descent

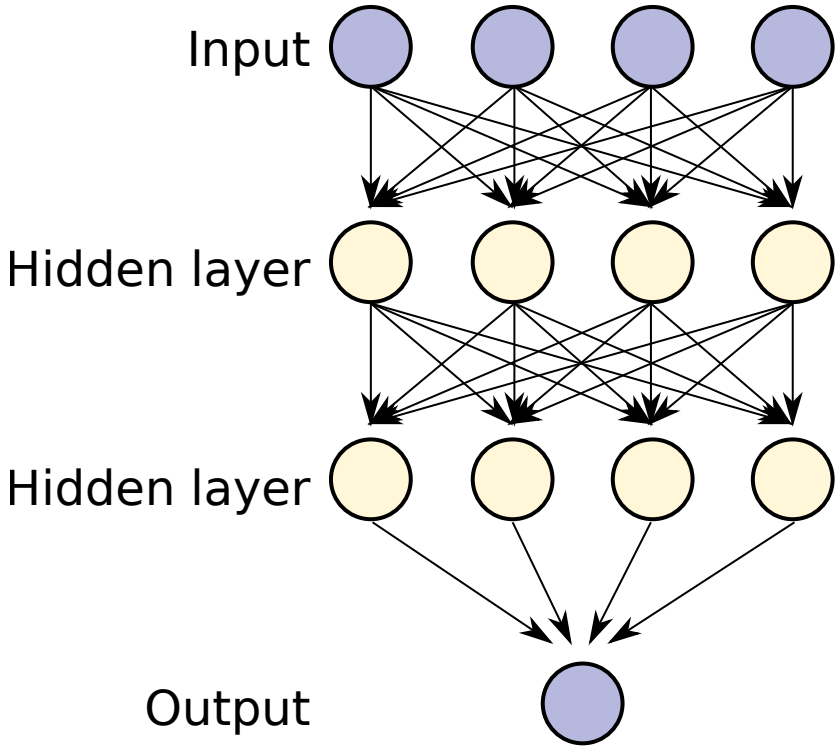


Overfitting/underfitting

A NN's 'capacity' should be appropriate to the complexity of the problem.

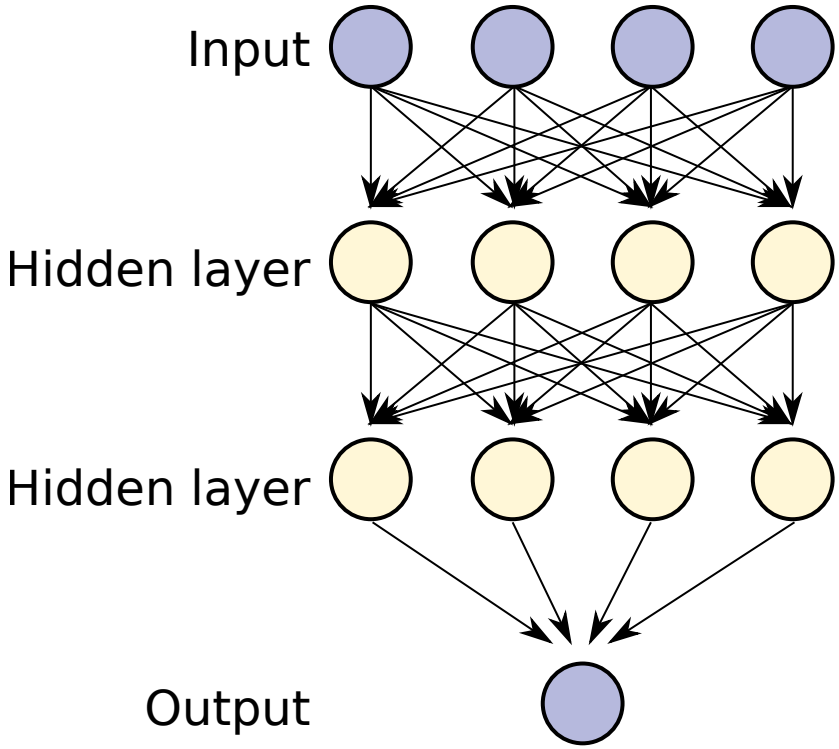


Number of parameters



AlexNet (2012): 8 layers, 62 million params
VGG16 (2015): 16 layers, 138 million params
ResNet50 (2016): 50 layers, 23 million params

Number of parameters



AlexNet (2012): 8 layers, 62 million params
VGG16 (2015): 16 layers, 138 million params
ResNet50 (2016): 50 layers, 23 million params

Training can easily fail

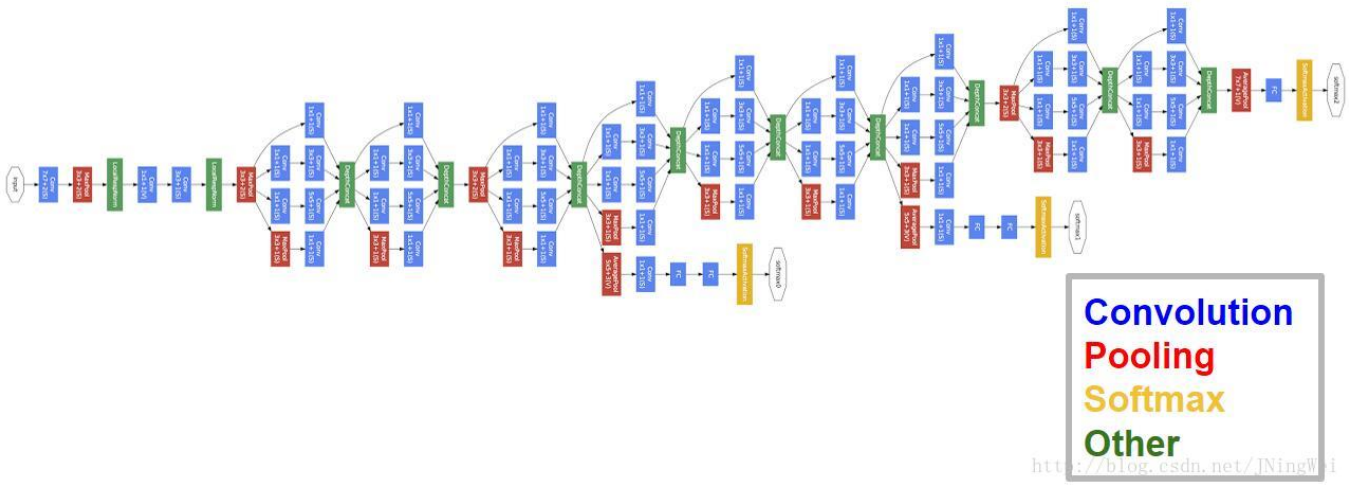
As depth and/or num params increase, we risk:

- Local minima
- Vanishing/exploding gradients:
  Backpropagating through many layers
  $\rightarrow$ deeply-nested multiplications of 'signal'
  $\rightarrow$ param update can converge to 0 or diverge to ∞

Vanishing/exploding gradients

One hack to avoid the problem...

GoogLeNet

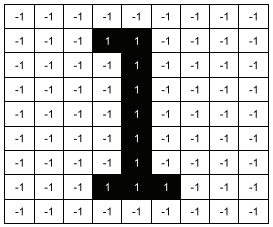


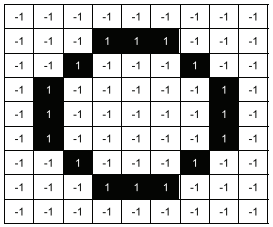
Stabilising training

Most of the important advances have been (in part) about stabilising the way NNs respond to training.

- rectifier nonlinearity (ReLU)
- batch norm
- dropout
- residual networks
- Adam
- RMSprop
- ...and CNNs?

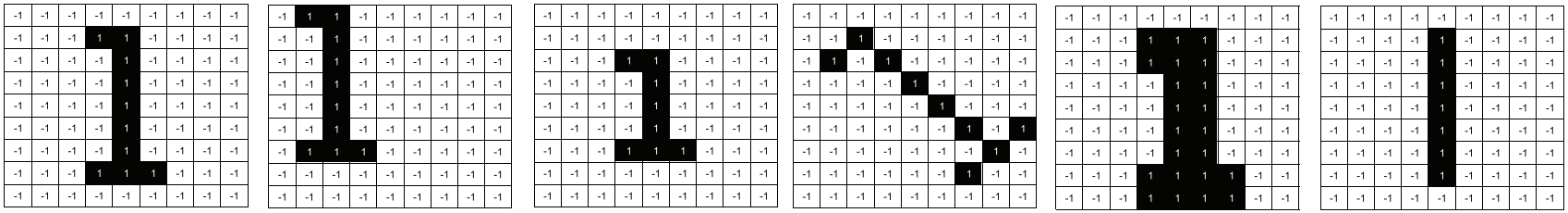
Motivational example





Inspired by B. Rohrer

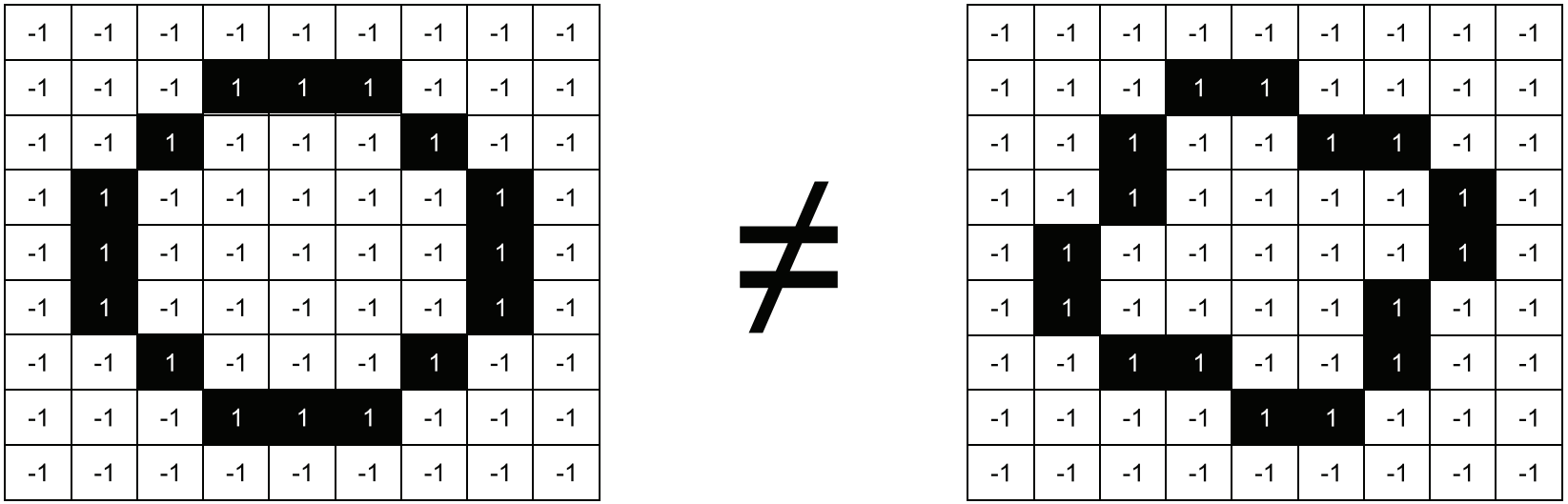
Motivational example



translated scaled rotated weighted styled

Inspired by B. Rohrer

CNNs match pieces of the images



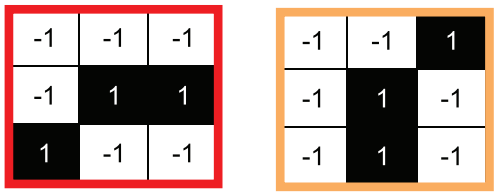
CNNs match pieces of the images

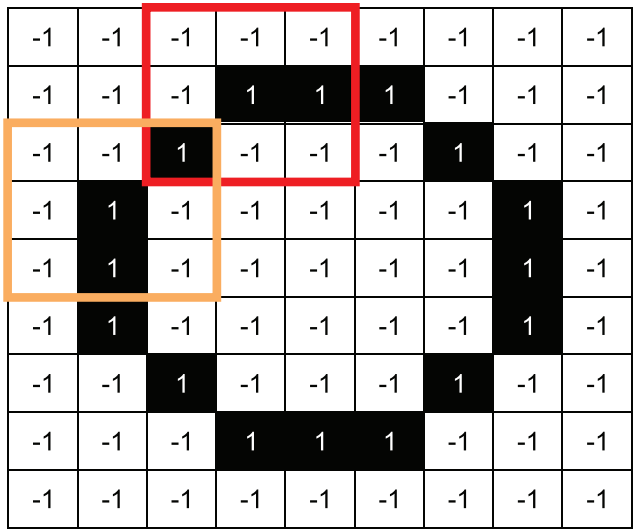
-1	-1	-1
-1	1	1
1	-1	-1

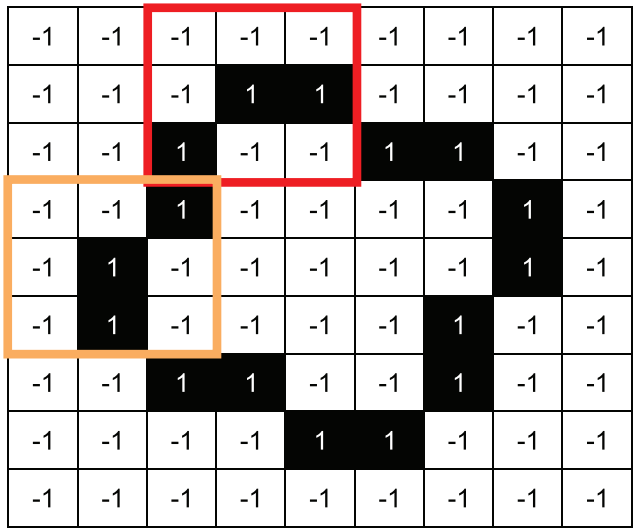
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

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

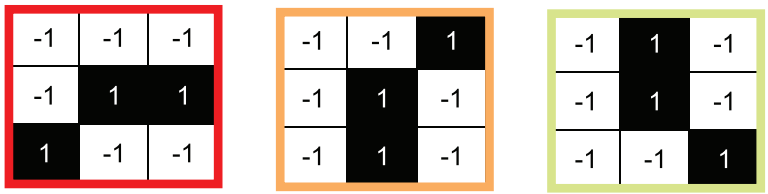
CNNs match pieces of the images

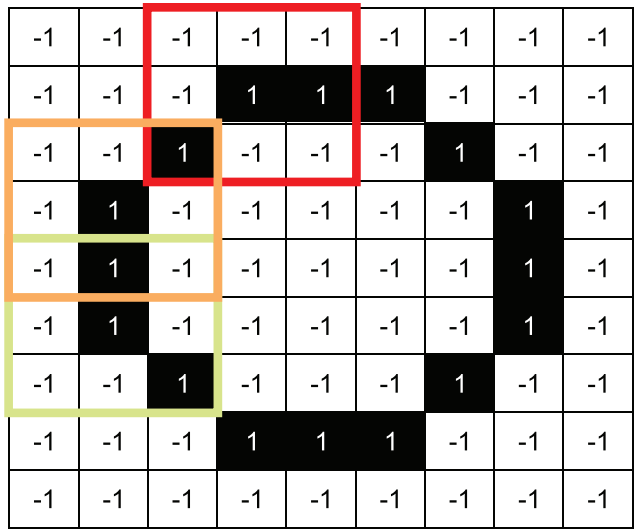


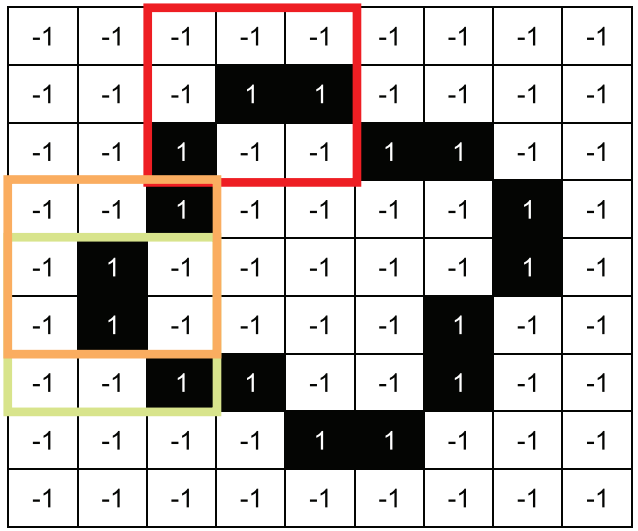




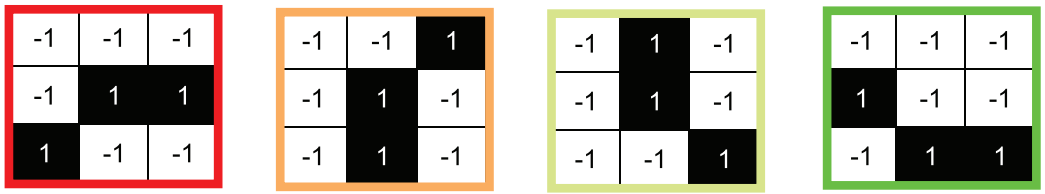
CNNs match pieces of the images

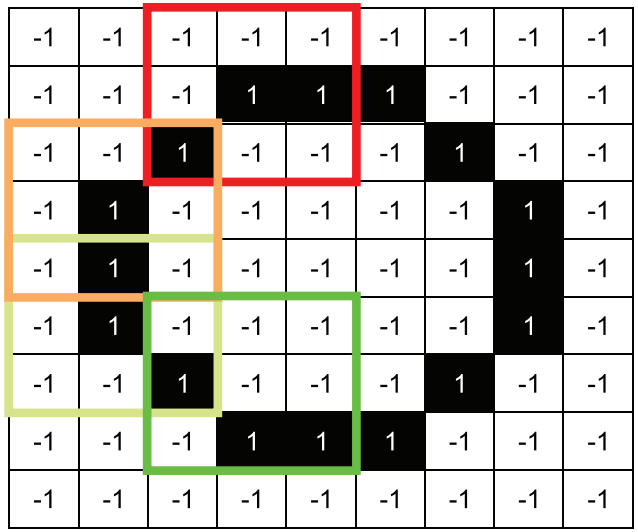


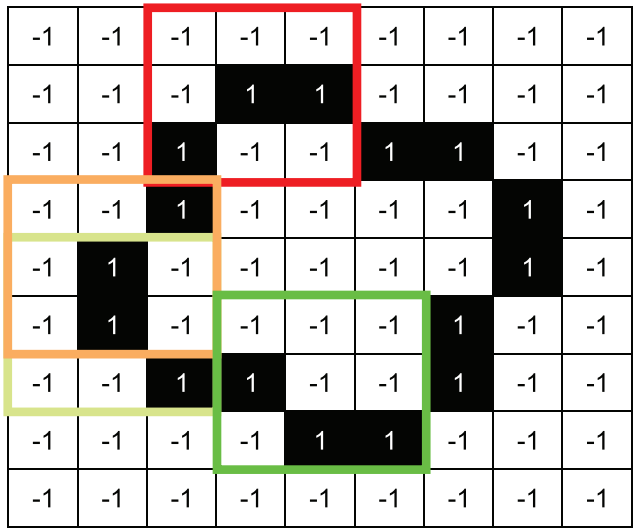




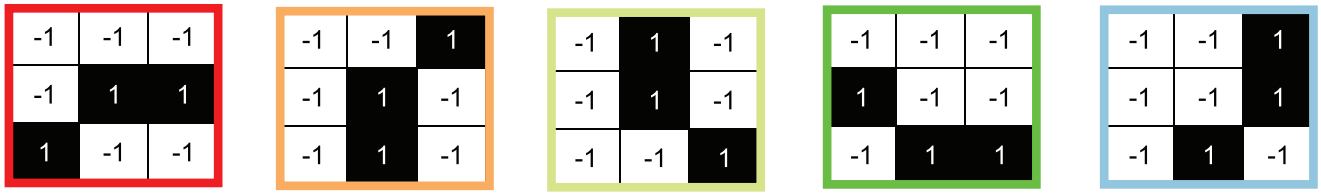
CNNs match pieces of the images

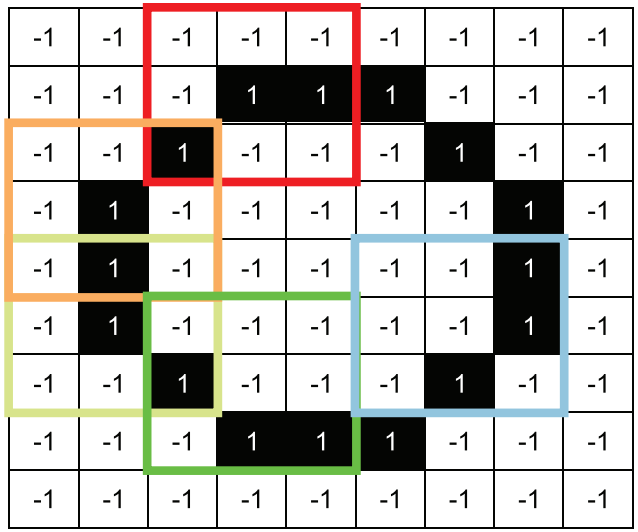


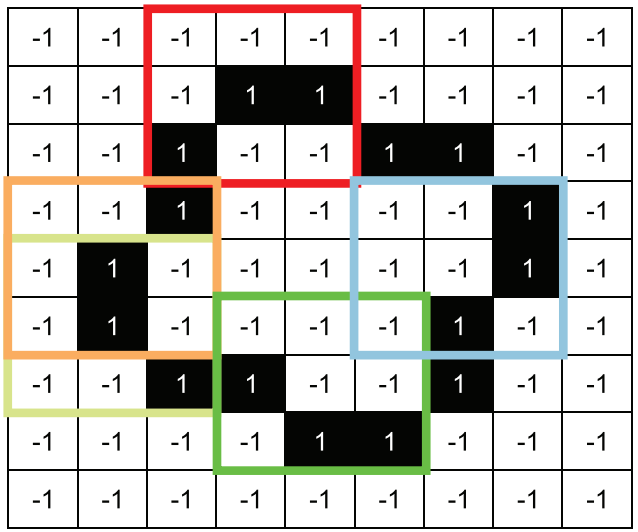




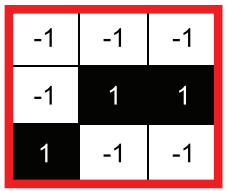
CNNs match pieces of the images

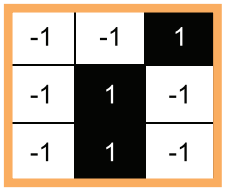


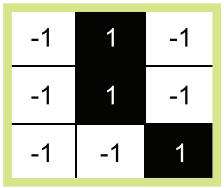


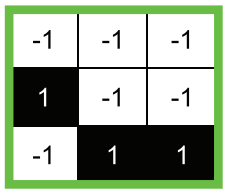


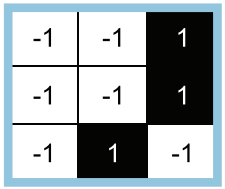
CNNs match pieces of the images

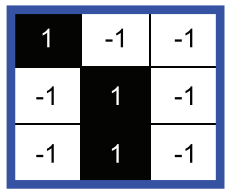


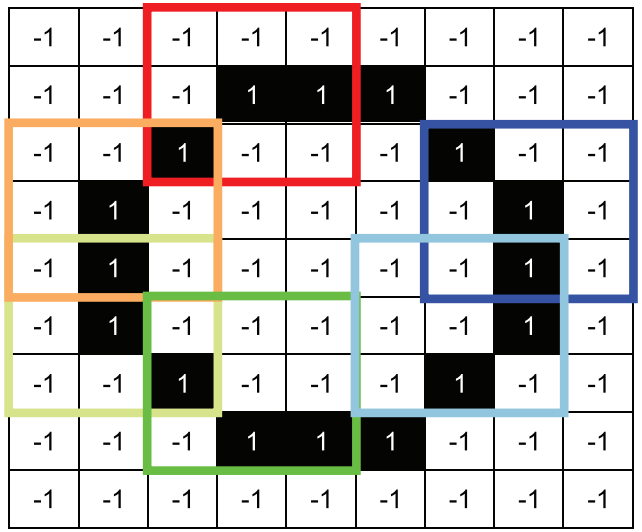


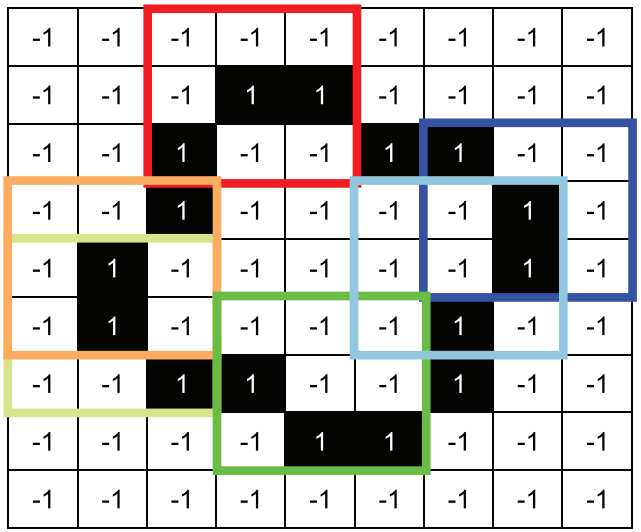




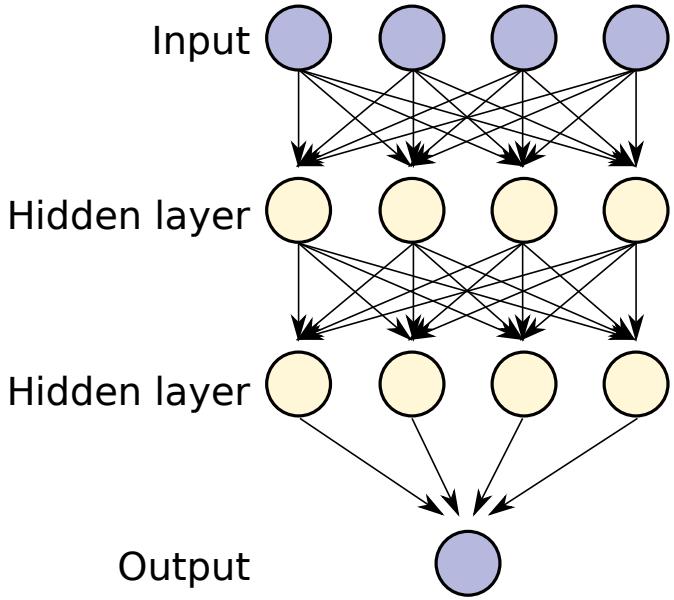




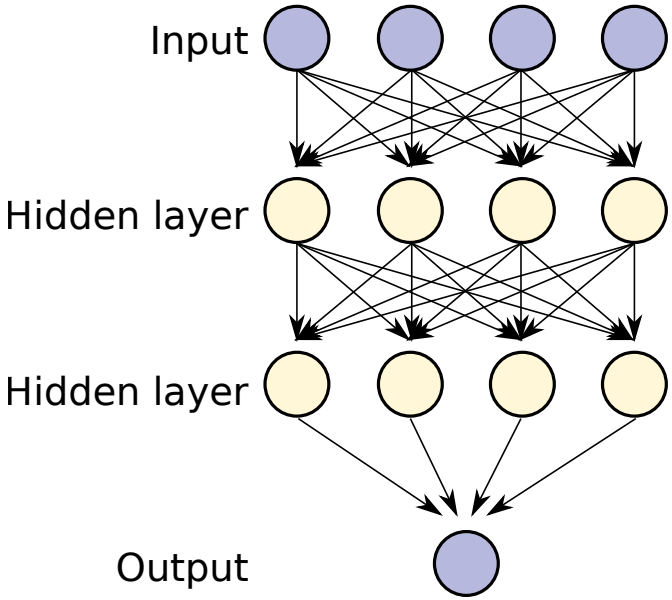


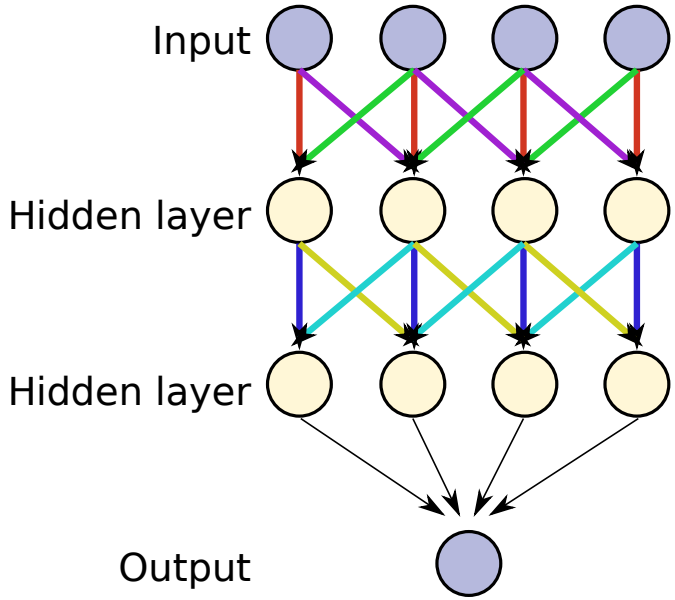


Going convolutional



Going convolutional

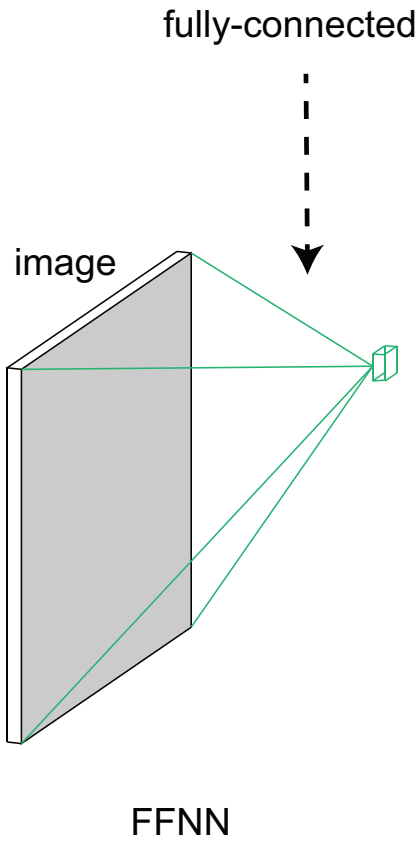




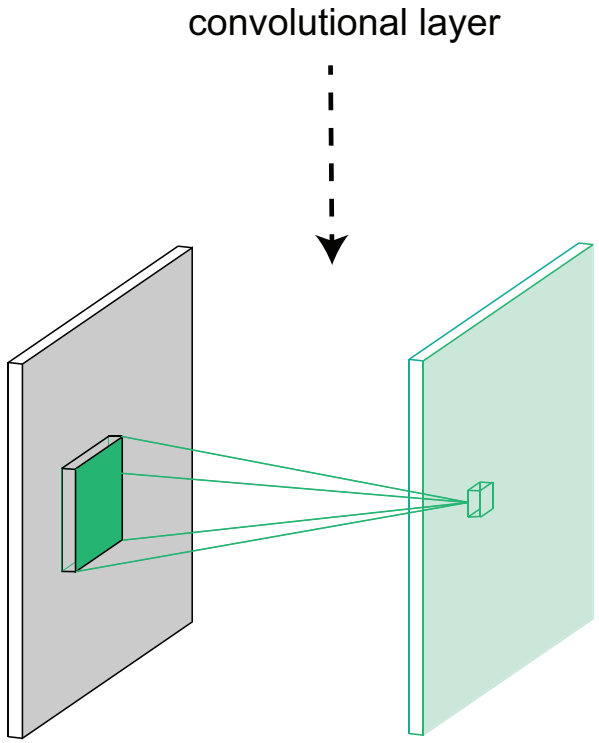
Shared weights among neurons

- Dramatically fewer free parameters $\rightarrow$ easier to train
- Shift-invariance built in
- Locality built in
- Axes preserved (e.g. frequency, time)

Convolutional layers vs. dense layers

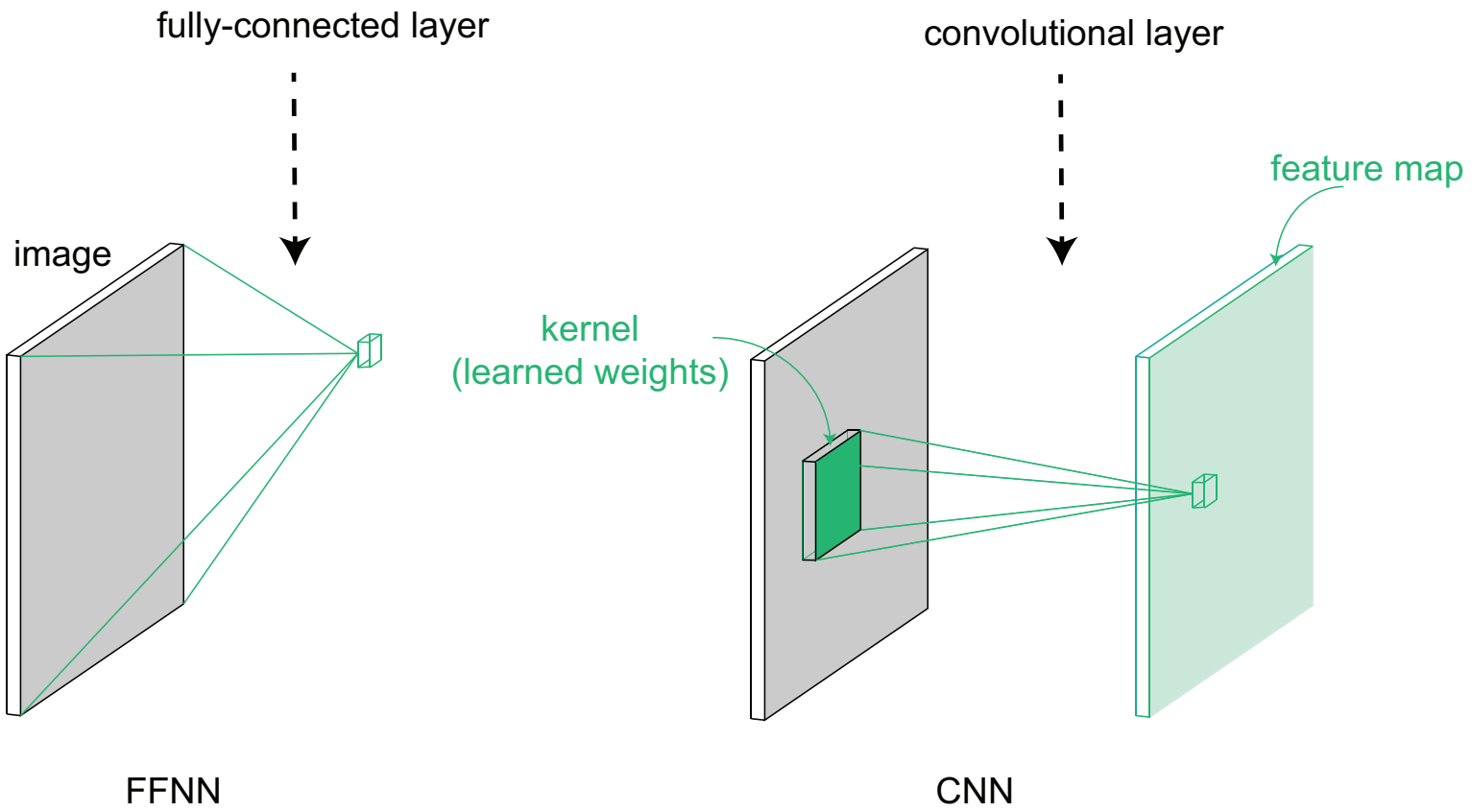


FFNN



CNN

Convolutional layers vs. dense layers



Going convolutional

Who does convolution work on 2D data?

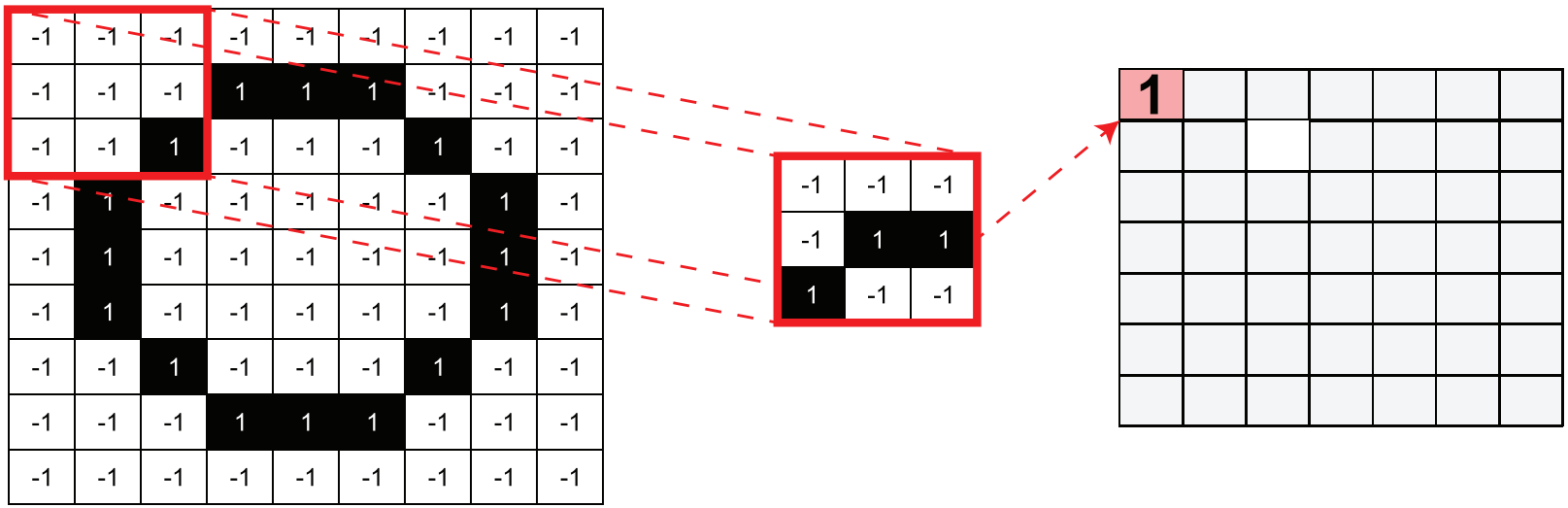
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

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

Going convolutional

Who does convolution work on 2D data?

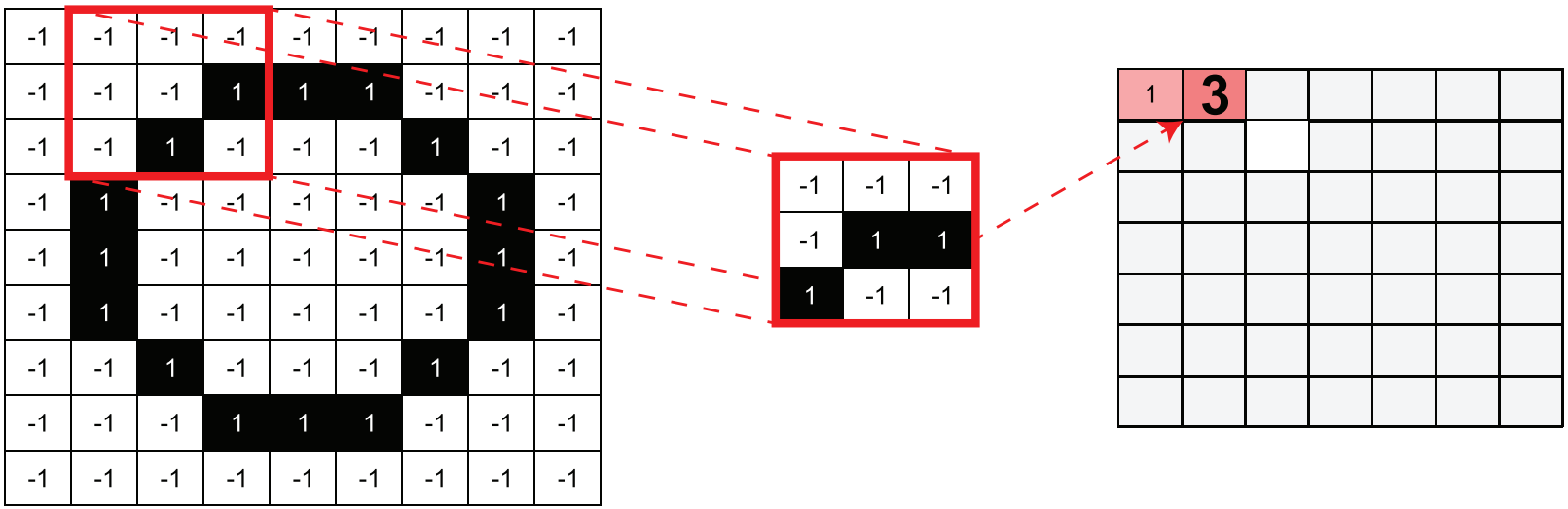
$$(-1 \times -1) + (-1 \times -1) + (-1 \times -1) + \dots$$
$$(-1 \times -1) + (-1 \times \ \ 1) + (-1 \times \ \ 1) + \dots$$
$$(-1 \times \ \ 1) + (-1 \times -1) + (\ \ 1 \times -1) = \mathbf{1}$$



Going convolutional

Who does convolution work on 2D data?

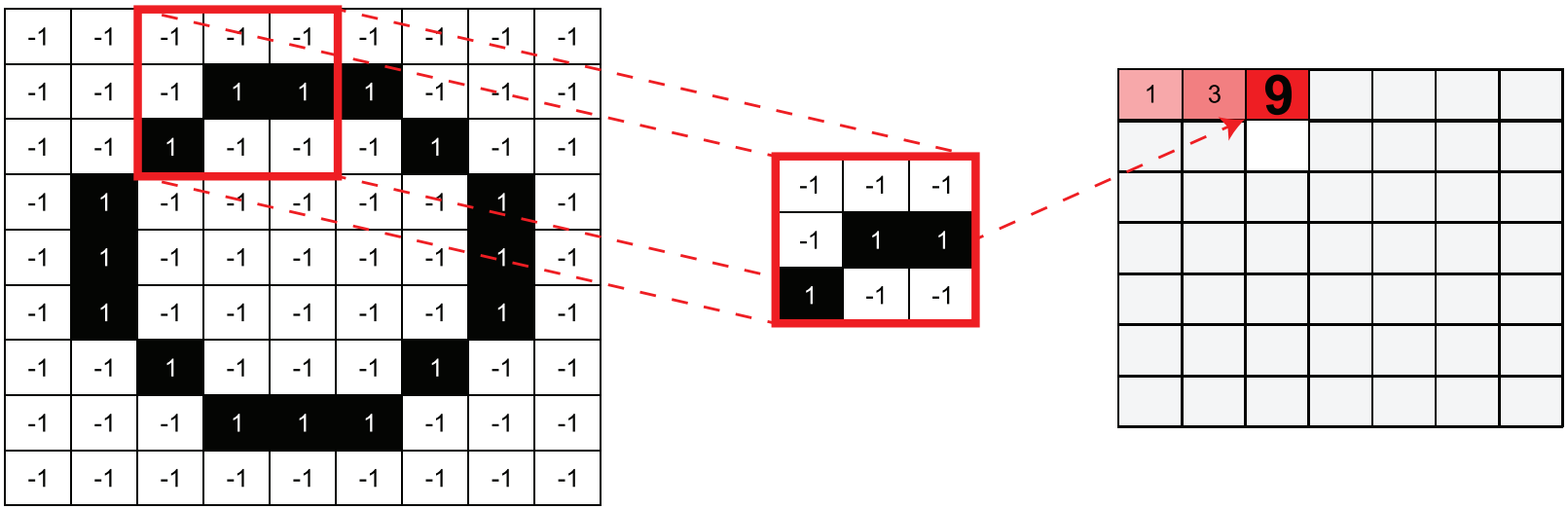
$$(-1 \times -1) + (-1 \times -1) + (-1 \times -1) + \dots$$
$$(-1 \times -1) + (-1 \times \ \ 1) + (\ \ 1 \times \ \ 1) + \dots$$
$$(-1 \times \ \ 1) + (\ \ 1 \times -1) + (-1 \times -1) = \mathbf{3}$$



Going convolutional

Who does convolution work on 2D data?

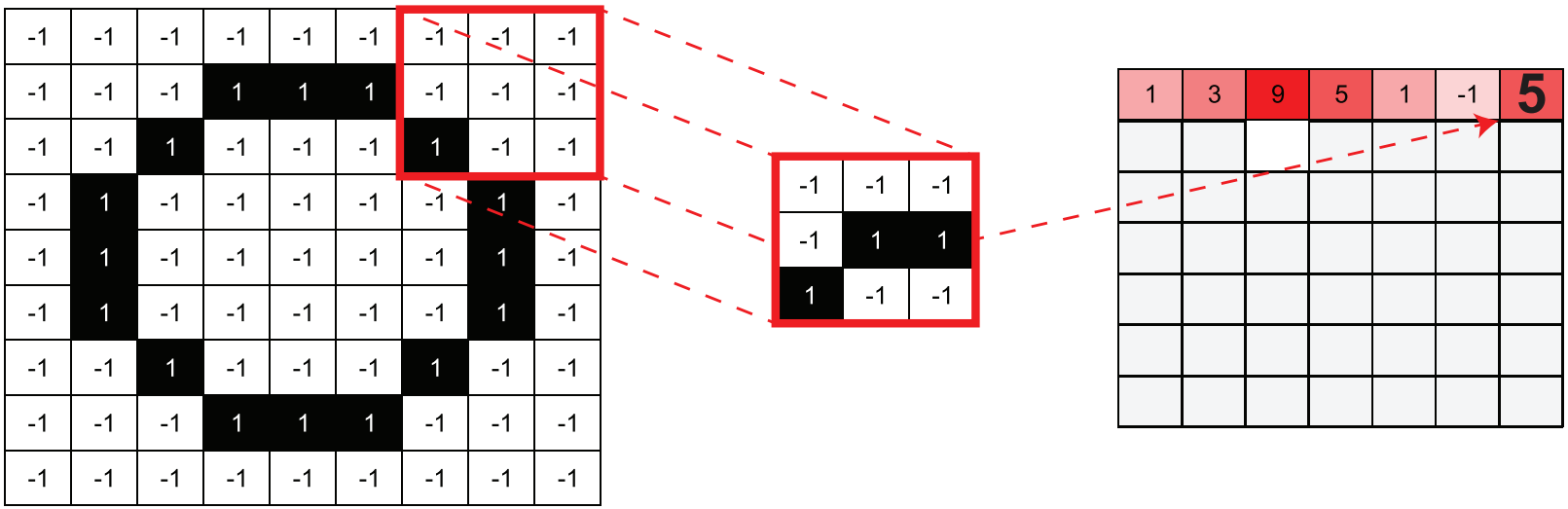
$$(-1 \times -1) + (-1 \times -1) + (-1 \times -1) + \dots$$
$$(-1 \times -1) + (\ 1 \times\ \ 1) + (\ 1 \times\ \ 1) + \dots$$
$$(\ 1 \times\ \ 1) + (-1 \times -1) + (-1 \times -1) = \mathbf{9}$$



Going convolutional

Who does convolution work on 2D data?

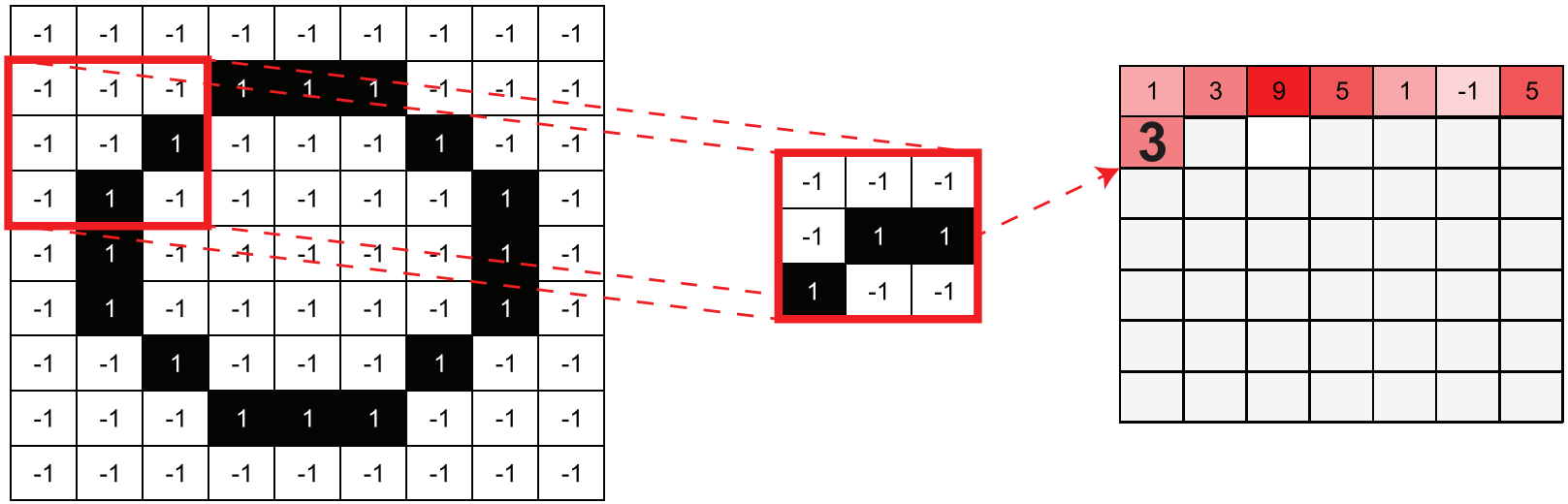
$$(-1 \times -1) + (-1 \times -1) + (-1 \times -1) + \dots$$
$$(-1 \times -1) + (-1 \times \ \ 1) + (-1 \times \ \ 1) + \dots$$
$$(\ \ 1 \times \ \ 1) + (-1 \times -1) + (-1 \times -1) = \mathbf{5}$$



Going convolutional

Who does convolution work on 2D data?

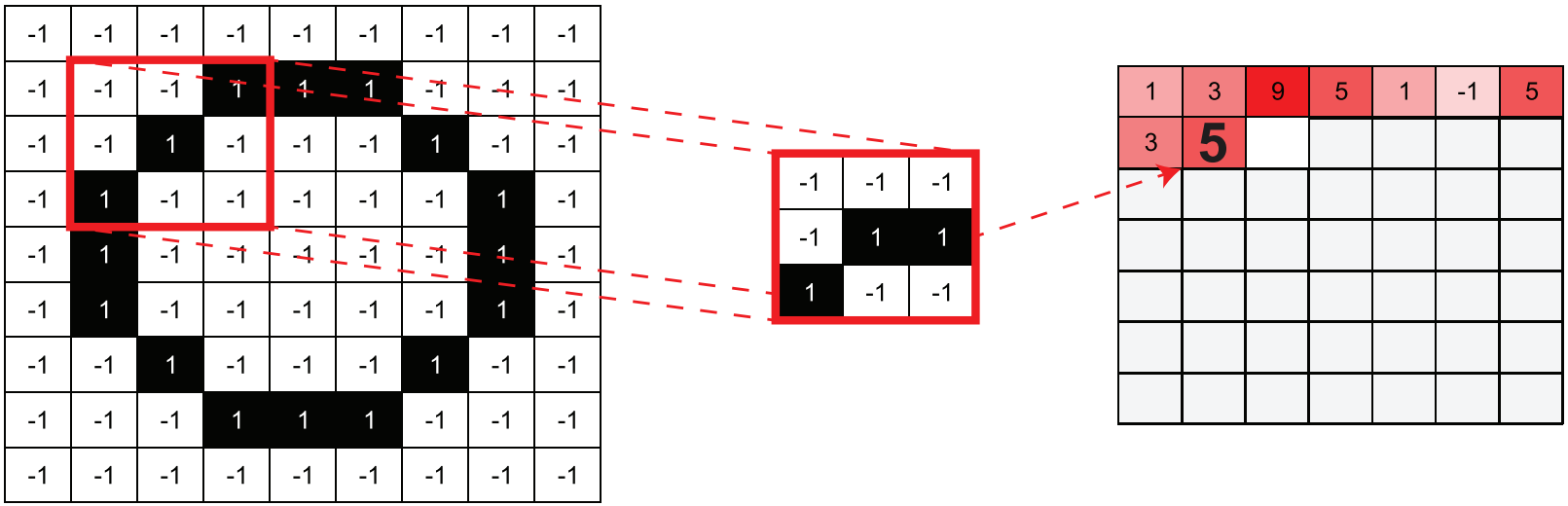
$$(-1 \times -1) + (-1 \times -1) + (-1 \times -1) + \ldots$$
$$(-1 \times -1) + (-1 \times \ \ 1) + (\ \ 1 \times \ \ 1) + \ldots$$
$$(-1 \times \ \ 1) + (\ \ 1 \times -1) + (-1 \times -1) = \mathbf{3}$$



Going convolutional

Who does convolution work on 2D data?

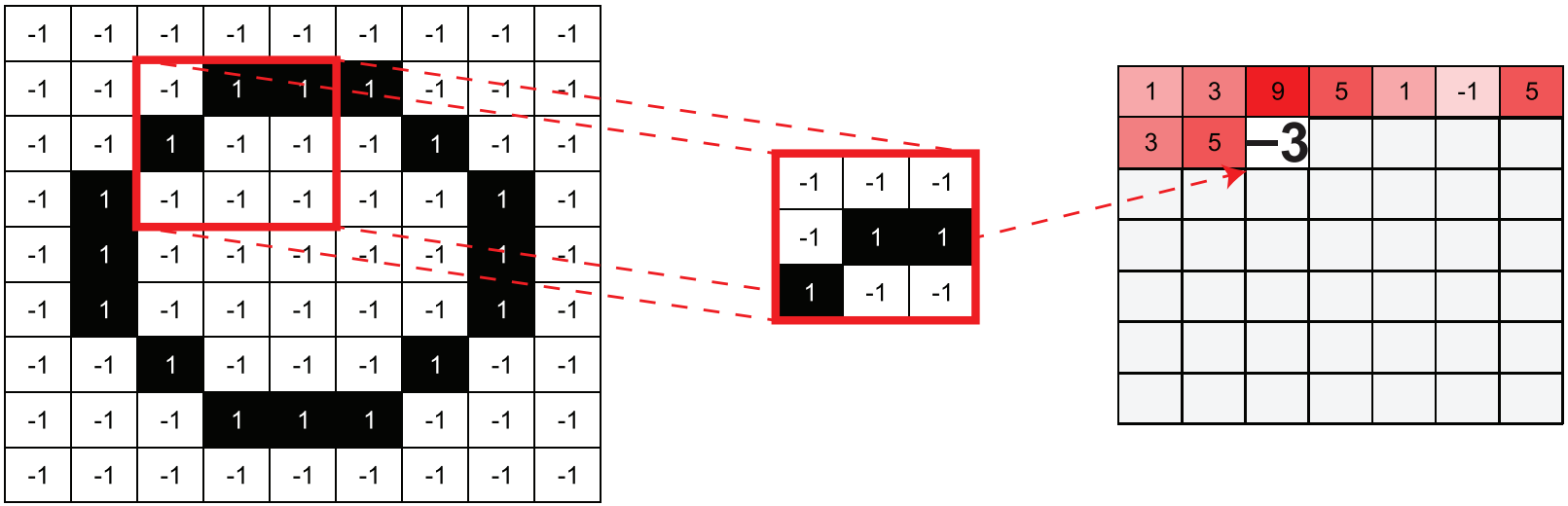
$$(-1 \times -1) + (-1 \times -1) + (\ \ 1 \times -1) + \ldots$$
$$(-1 \times -1) + (\ \ 1 \times \ \ 1) + (-1 \times \ \ 1) + \ldots$$
$$(\ \ 1 \times \ \ 1) + (-1 \times -1) + (-1 \times -1) = \mathbf{5}$$



Going convolutional

Who does convolution work on 2D data?

$$(-1 \times -1) + (\ \ 1 \times -1) + (\ \ 1 \times -1) + \ldots$$
$$(\ \ 1 \times -1) + (-1 \times \ \ 1) + (-1 \times \ \ 1) + \ldots$$
$$(-1 \times \ \ 1) + (-1 \times -1) + (-1 \times -1) = \mathbf{-3}$$



Going convolutional

Who does convolution work on 2D data?

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

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

1	3	9	5	1	-1	5
3	5	-3	-3	1	1	-1
1	1	1	3	1	1	1
1	1	3	3	3	1	1
1	-3	5	3	1	1	5
3	1	-3	1	5	5	-1
1	3	5	5	1	-1	1

Going convolutional

Who does convolution work on 2D data?

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

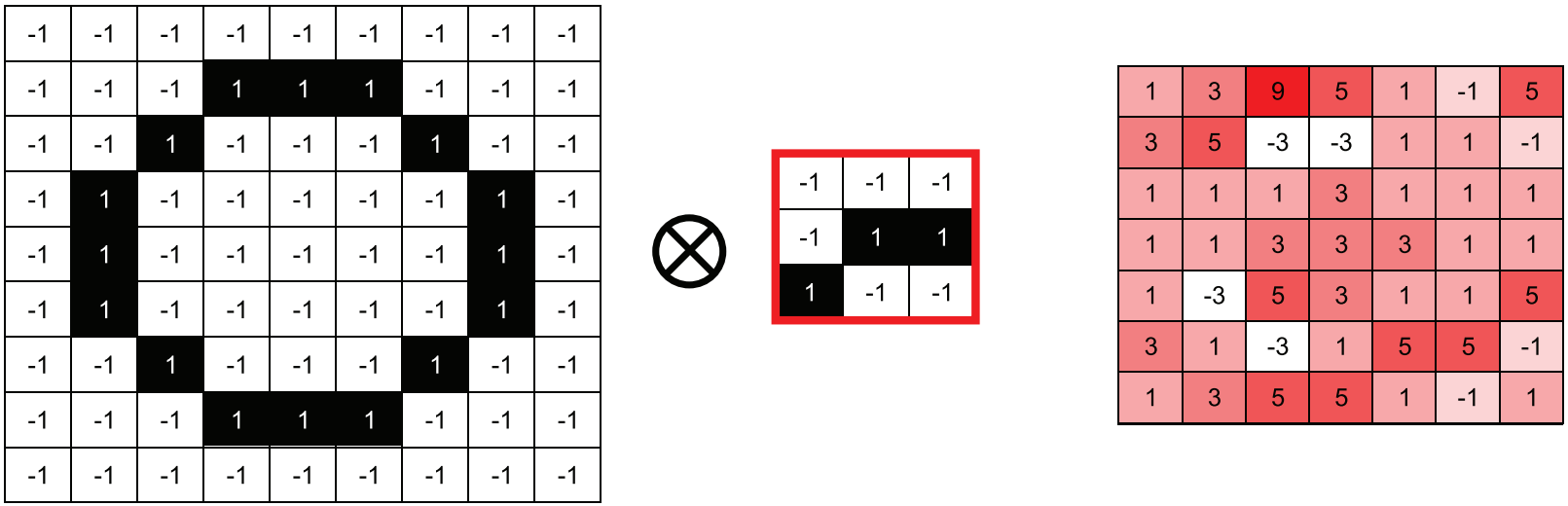
-1	-1	-1
-1	1	1
1	-1	-1

Feature map

1	3	9	5	1	-1	5
3	5	-3	-3	1	1	-1
1	1	1	3	1	1	1
1	1	3	3	3	1	1
1	-3	5	3	1	1	5
3	1	-3	1	5	5	-1
1	3	5	5	1	-1	1

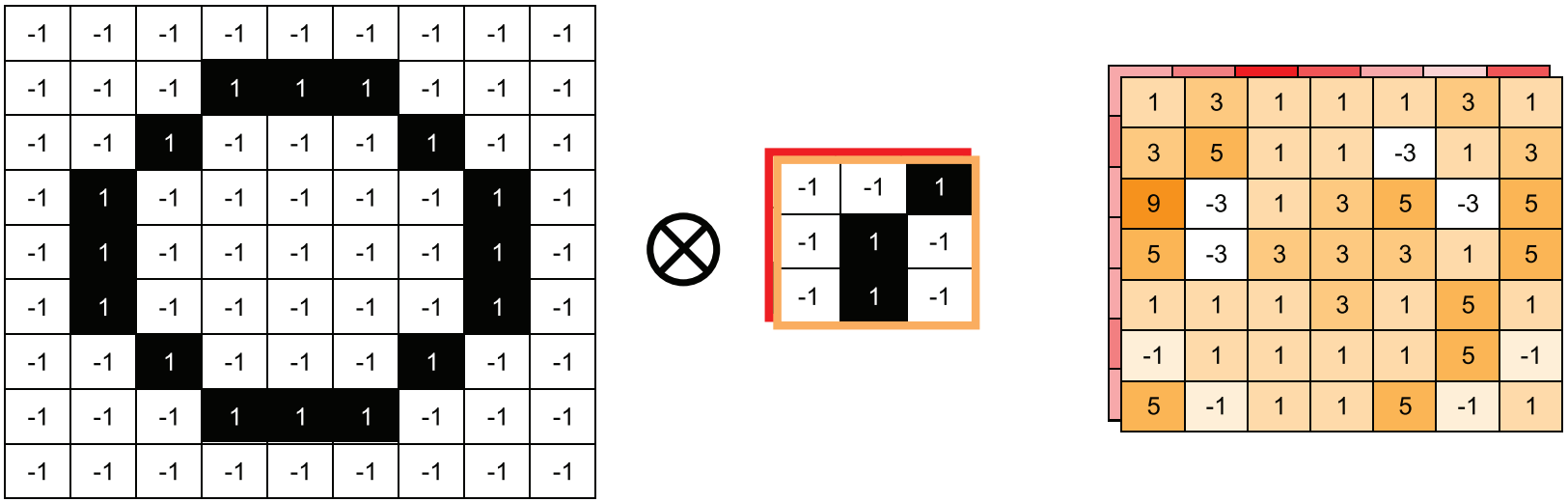
Convolution Layer

Convolution layer on 2D data:



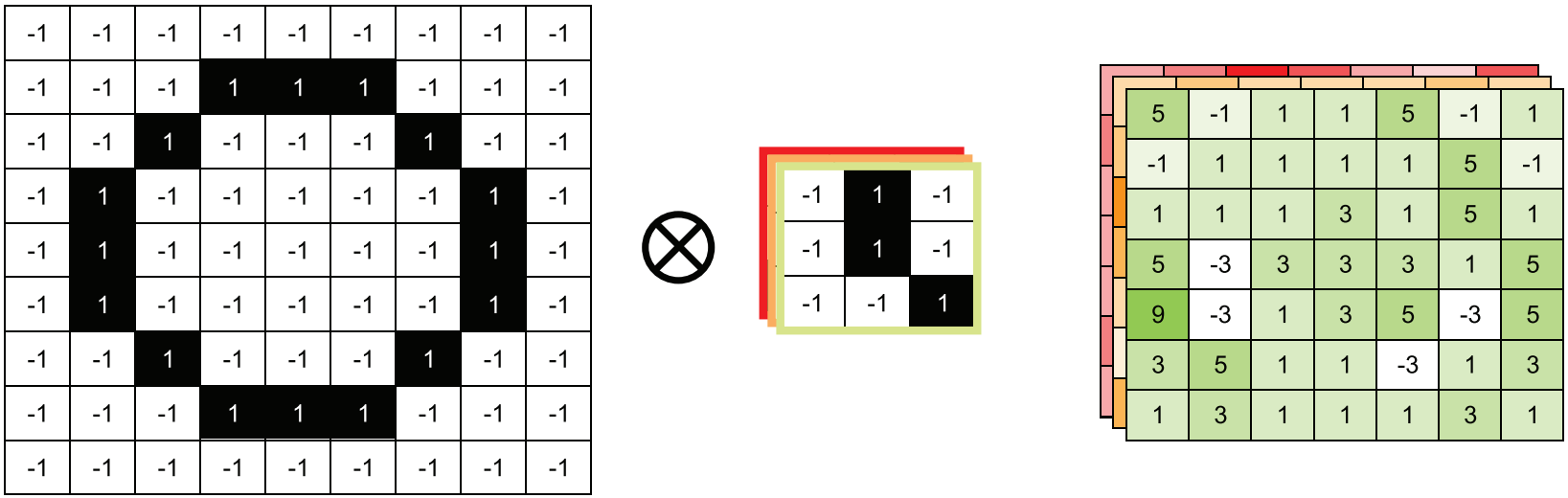
Convolution Layer

Convolution layer on 2D data:



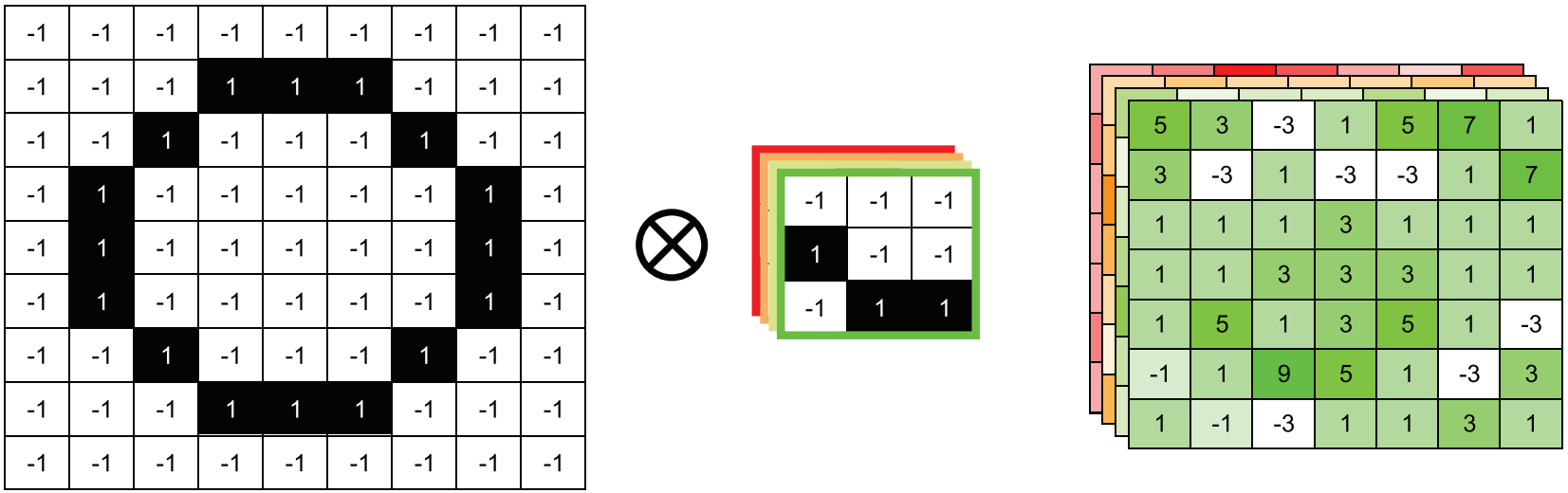
Convolution Layer

Convolution layer on 2D data:



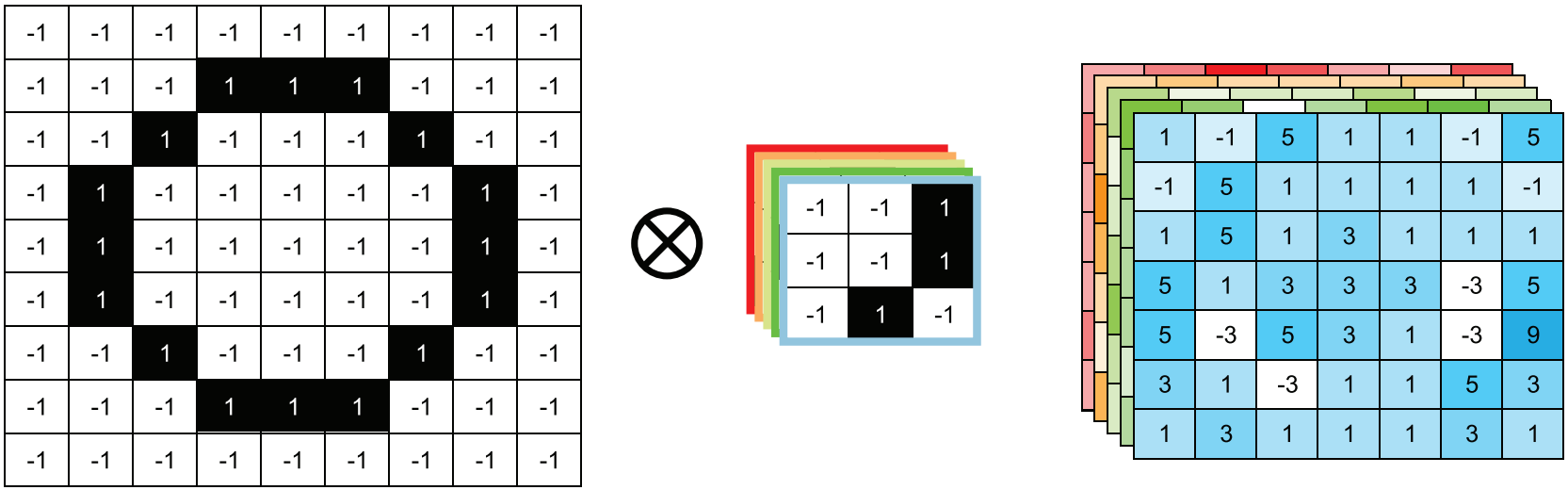
Convolution Layer

Convolution layer on 2D data:



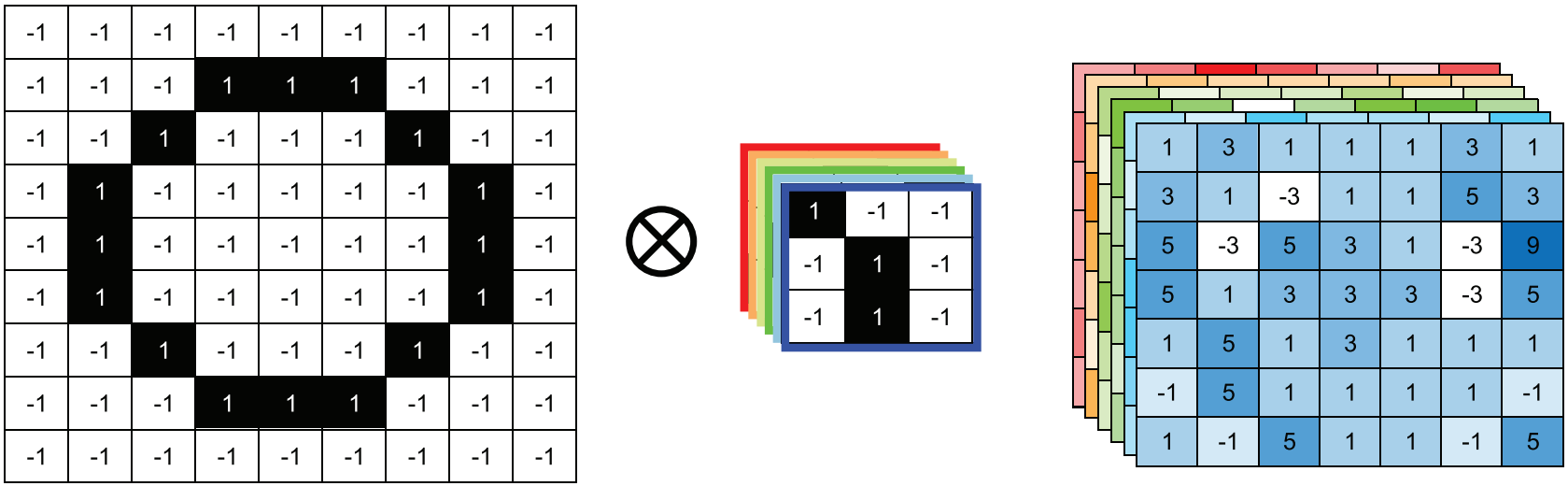
Convolution Layer

Convolution layer on 2D data:



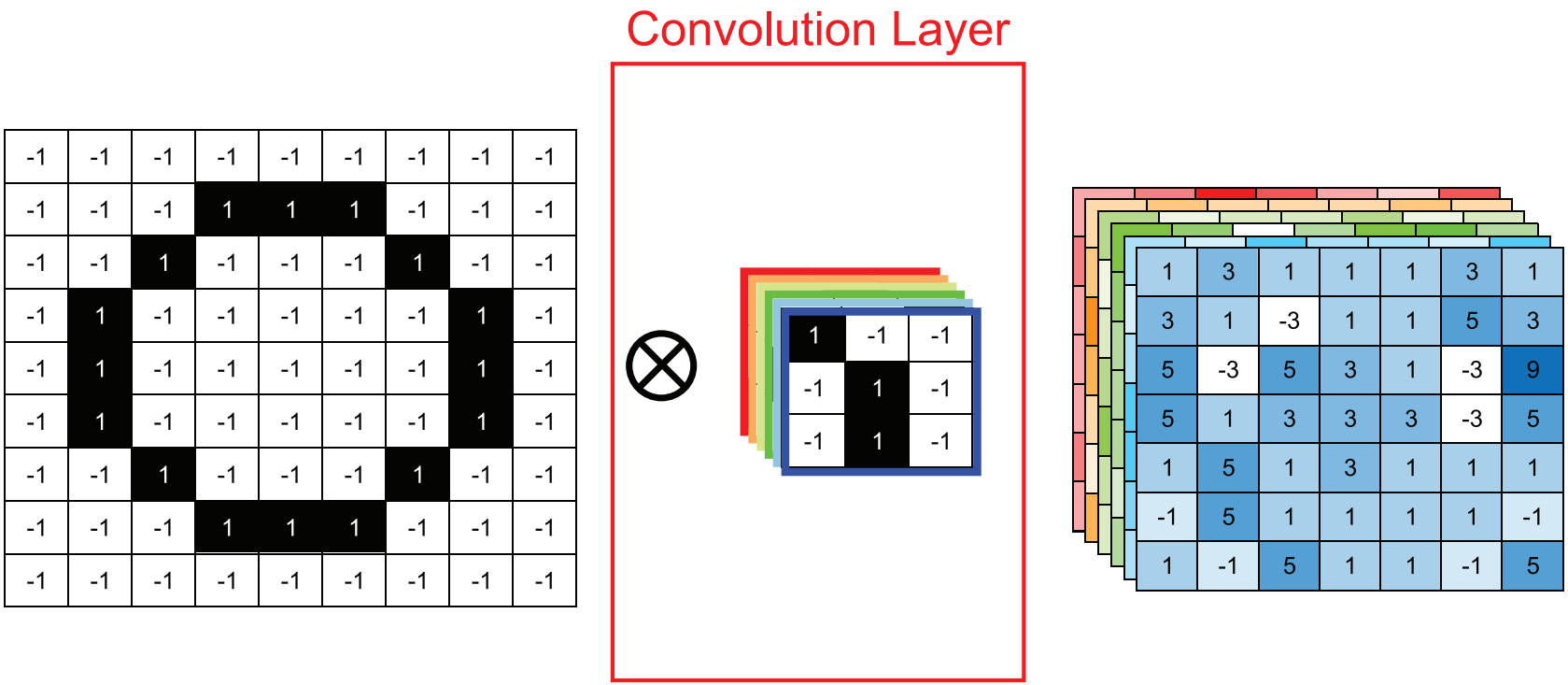
Convolution Layer

Convolution layer on 2D data:



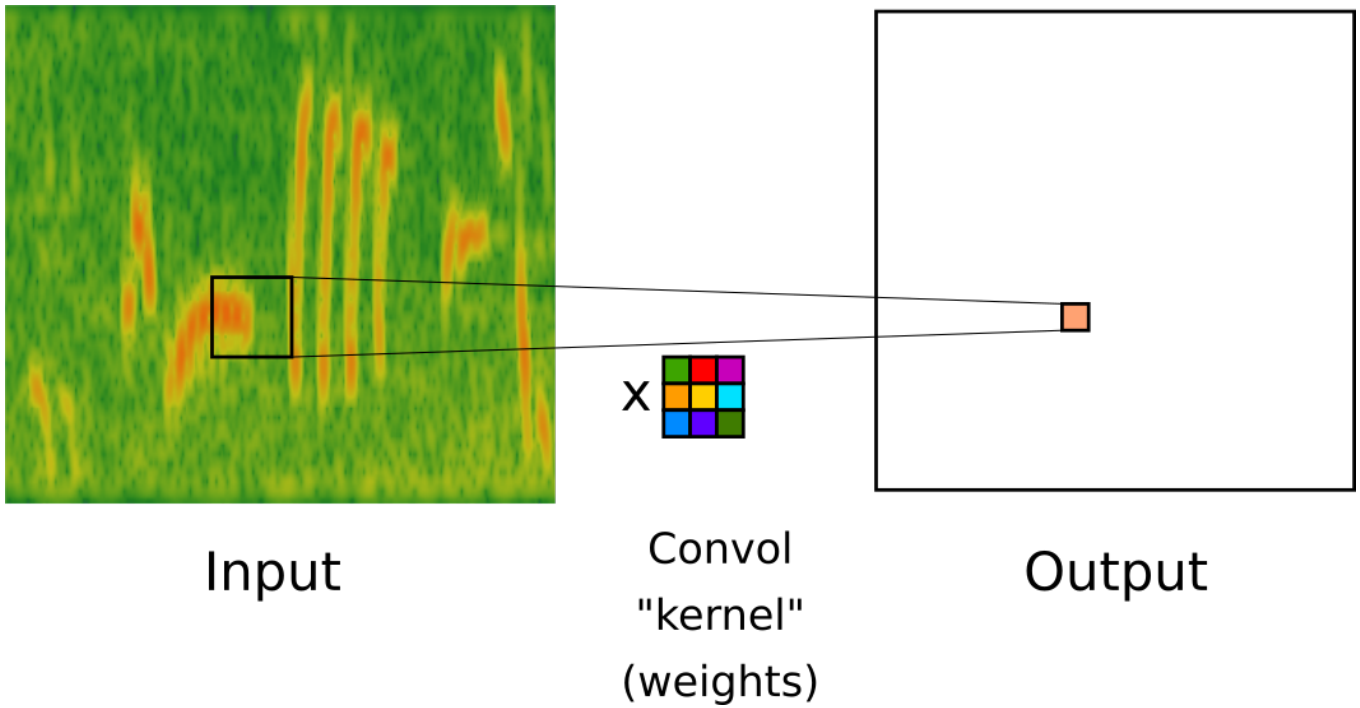
Convolution Layer

Convolution layer on 2D data:



Going convolutional

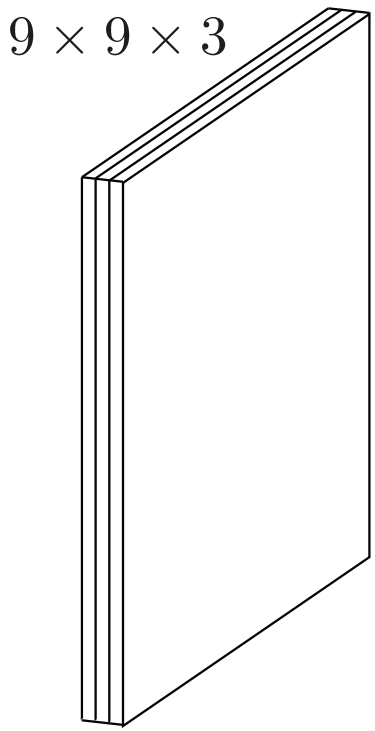
2D spectrogram data:



Channels

- Convolutions usually computed for each channel and summed

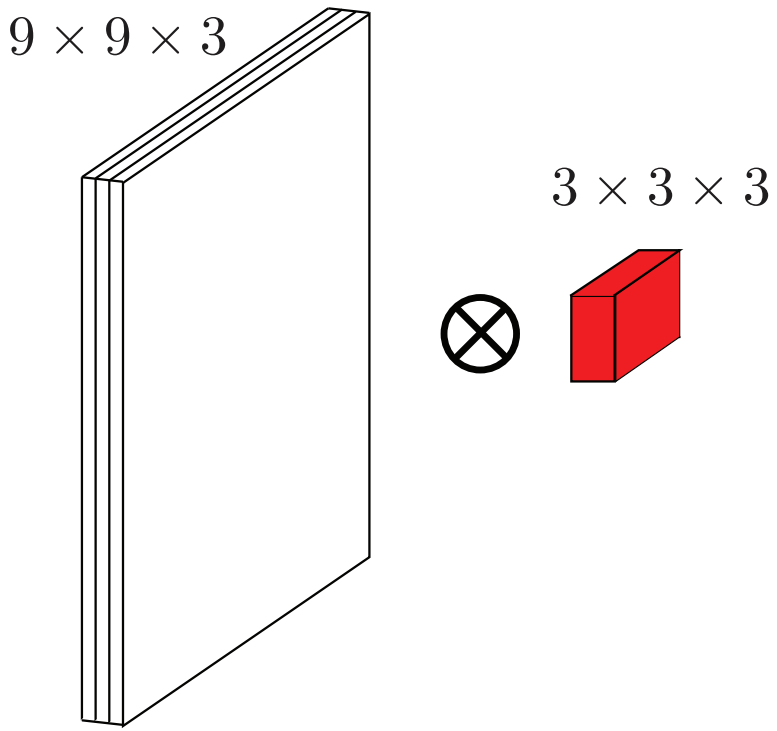
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

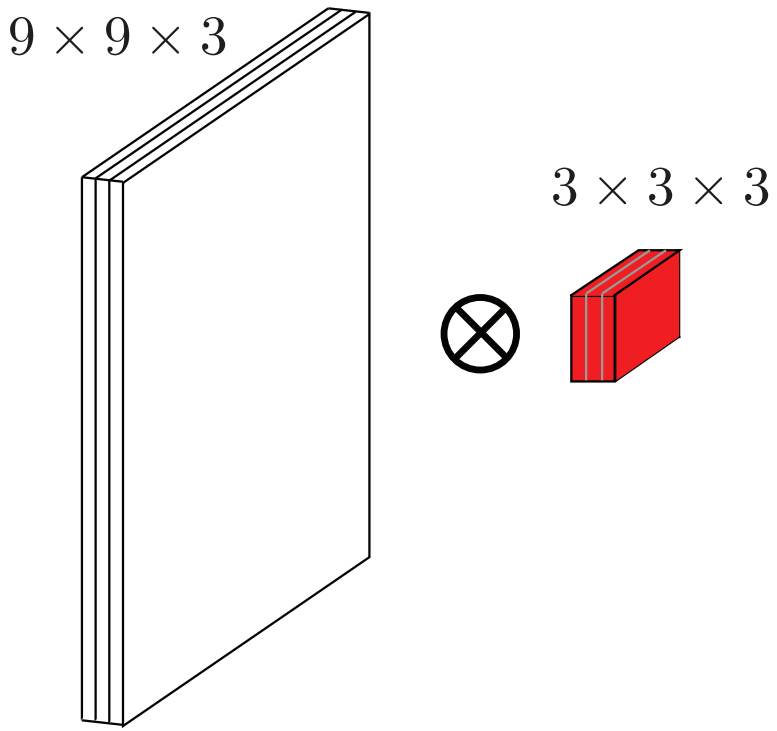
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

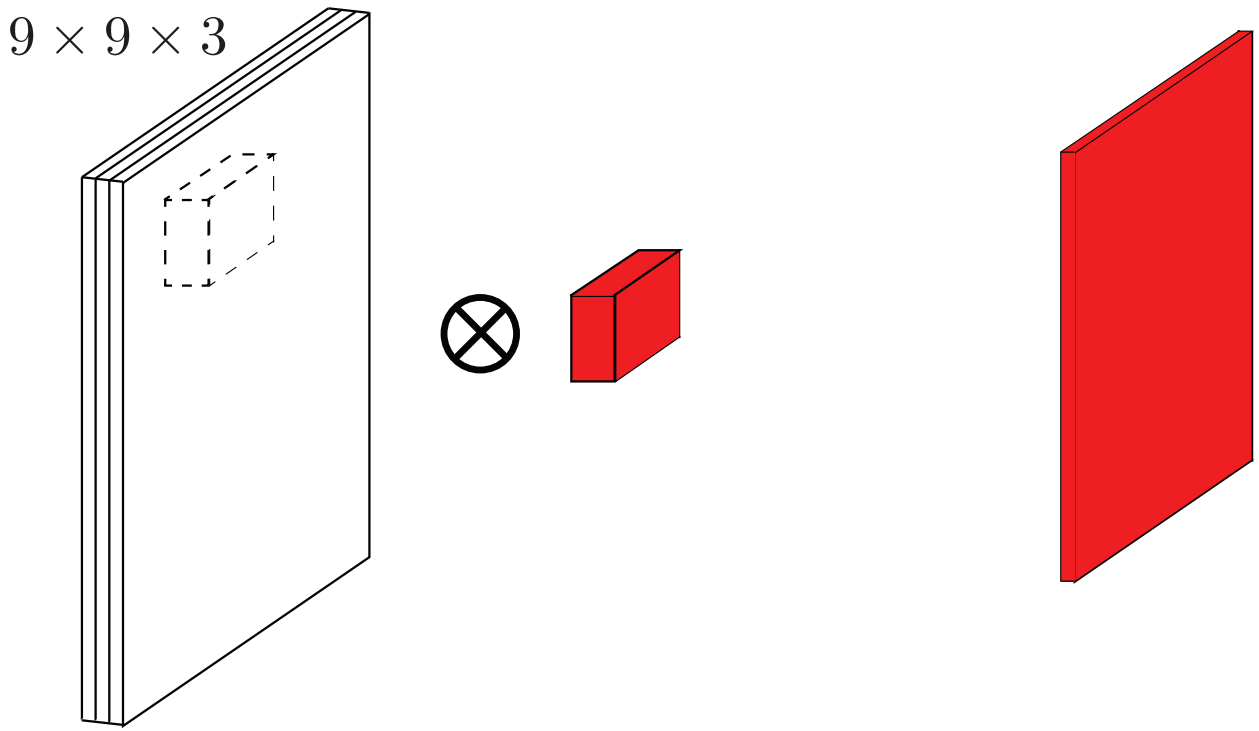
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

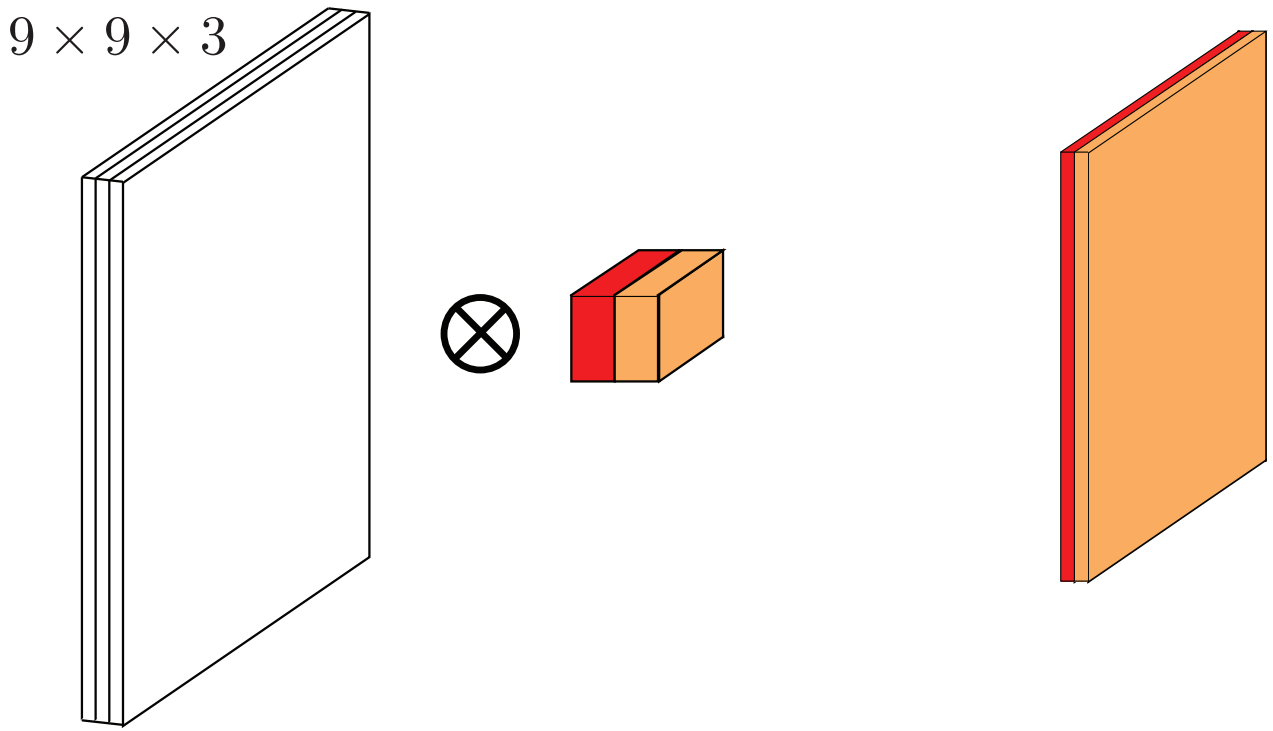
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

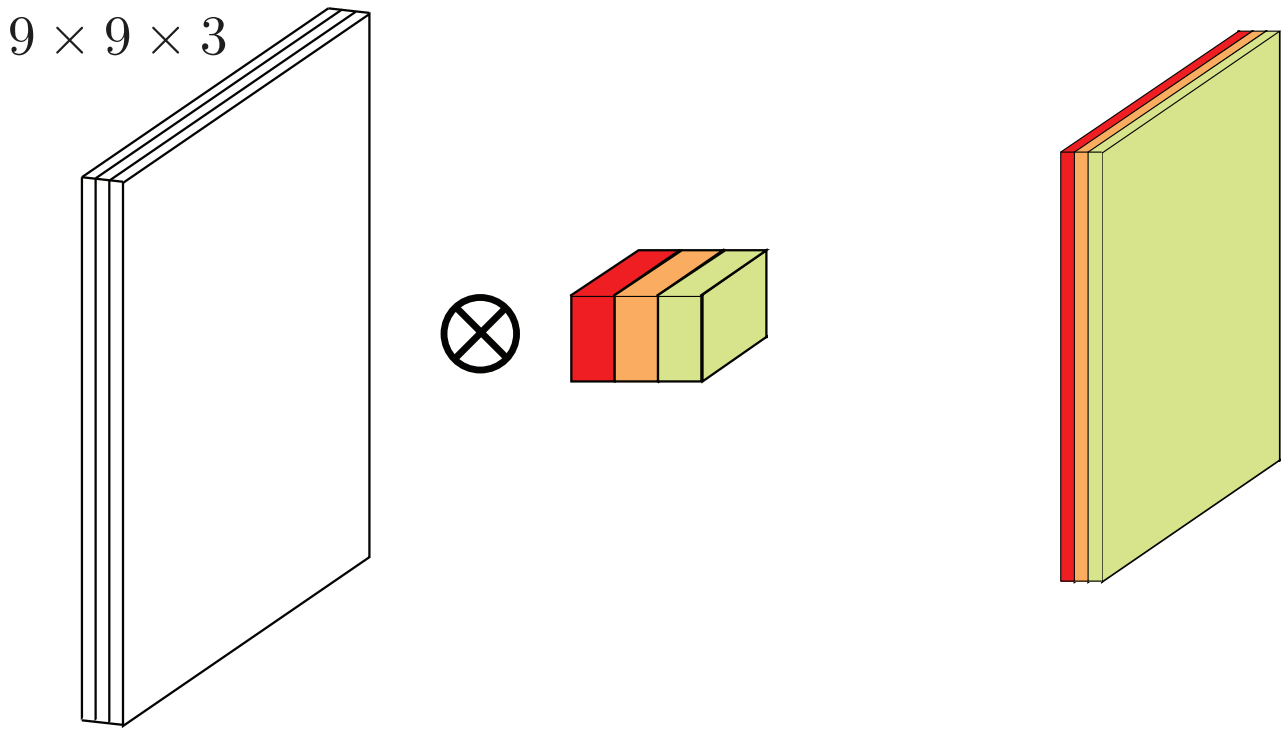
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

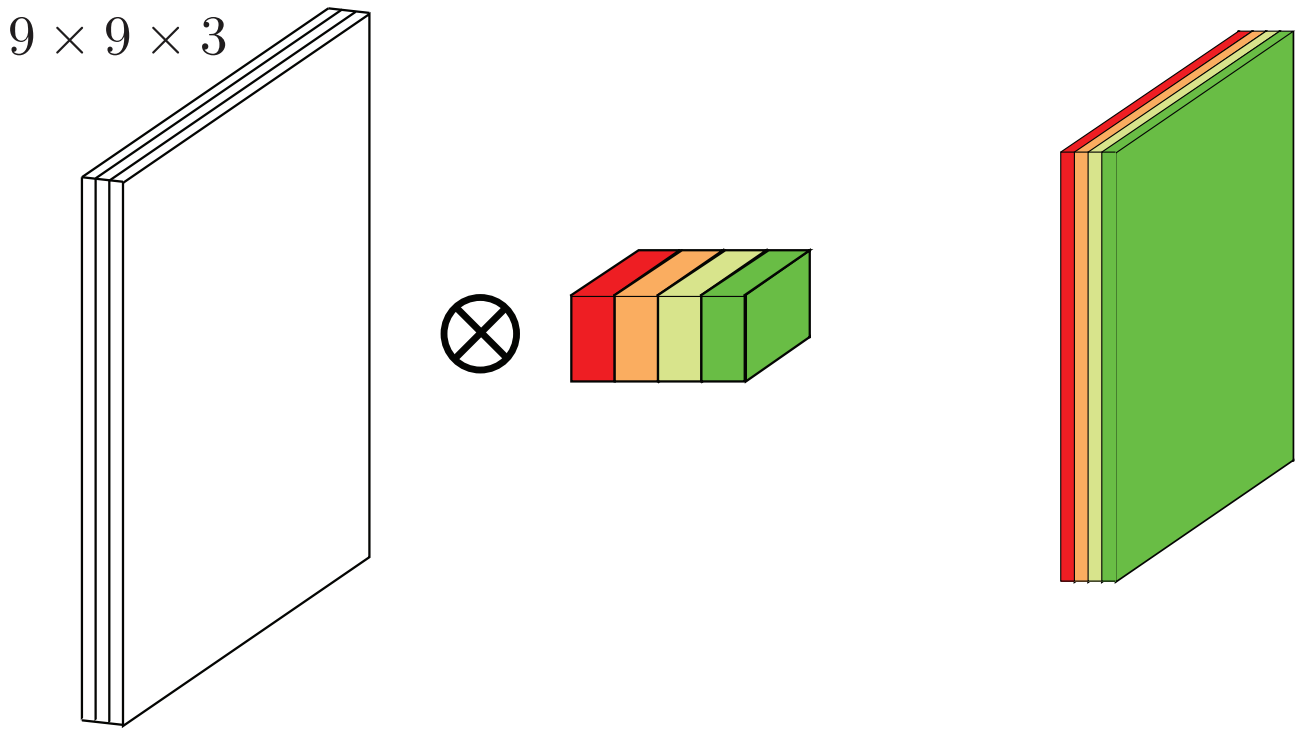
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

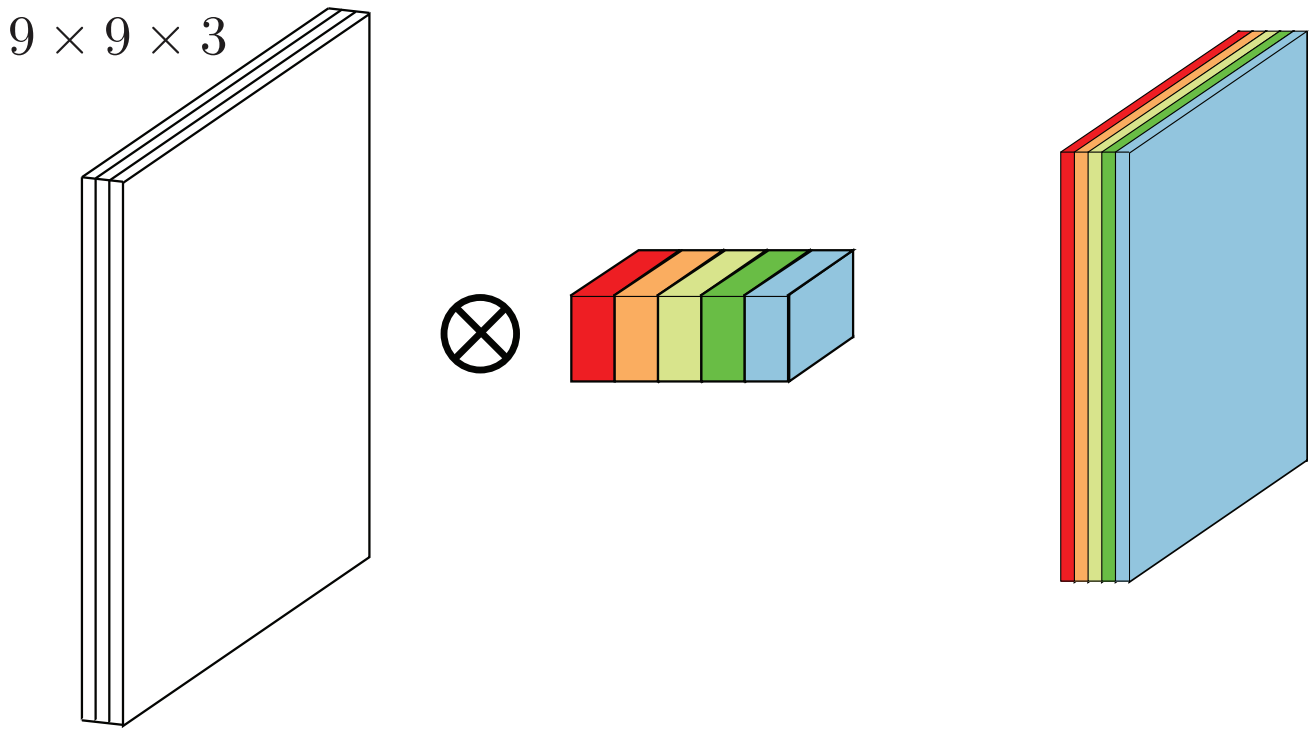
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

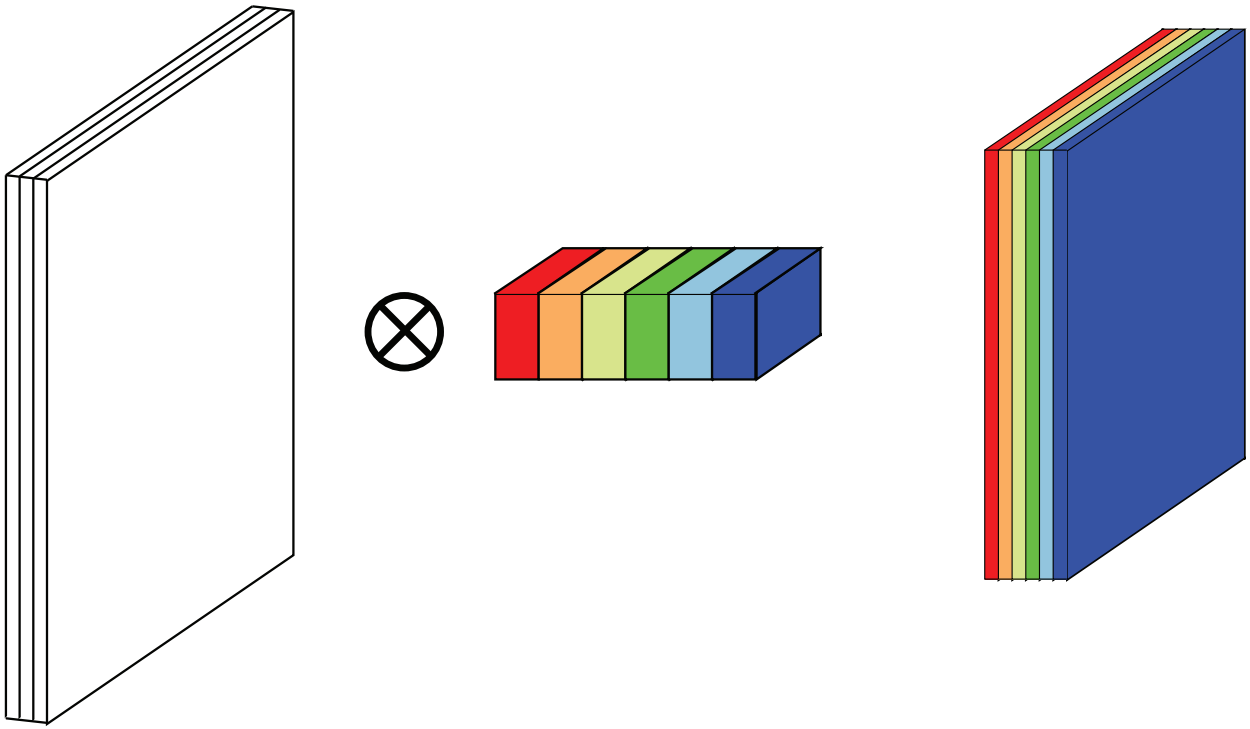
$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Channels

- Convolutions usually computed for each channel and summed

$$(k \star g^c) = \sum_{c=1}^{3} k^c \star g^c$$



Convolutional layer parameters

- An image input $128 \times 128 \times 3$
- 100 kernels of size $5 \times 5 \times 3$
- Number of parameters of this layer?

  For every filter: $5 \times 5 \times 3 + 1 = 76$ parameters.

  Total parameters: $100 \times 76 = 7600$ parameters.
- Think about how many parameters of a fully-connected layer with 100 and 1000 hidden units in this case?

  100 hidden units: $(128 \times 128 \times 3) \times 100 + 100 = 4,915,300$ parameters.

  1000 hidden units: $49,153,000$ parameters.

Convolutional layer parameters

- An image input $128 \times 128 \times 3$
- 100 kernels of size $5 \times 5 \times 3$
- Number of parameters of this layer?

  For every filter: $5 \times 5 \times 3 + 1 = 76$ parameters.

  Total parameters: $100 \times 76 = 7600$ parameters.

- Think about how many parameters of a fully-connected layer with 100 and 1000 hidden units in this case?

  100 hidden units: $(128 \times 128 \times 3) \times 100 + 100 = 4,915,300$ parameters.

  1000 hidden units: $49,153,000$ parameters.

Convolutional layer parameters

- An image input $128 \times 128 \times 3$
- 100 kernels of size $5 \times 5 \times 3$
- Number of parameters of this layer?

  For every filter: $5 \times 5 \times 3 + 1 = 76$ parameters.

  Total parameters: $100 \times 76 = 7600$ parameters.
- Think about how many parameters of a fully-connected layer with 100 and 1000 hidden units in this case?

  100 hidden units: $(128 \times 128 \times 3) \times 100 + 100 = 4,915,300$ parameters.

  1000 hidden units: $49,153,000$ parameters.

Convolutional layer parameters

- An image input $128 \times 128 \times 3$
- 100 kernels of size $5 \times 5 \times 3$
- Number of parameters of this layer?

  For every filter: $5 \times 5 \times 3 + 1 = 76$ parameters.

  Total parameters: $100 \times 76 = 7600$ parameters.
- Think about how many parameters of a fully-connected layer with 100 and 1000 hidden units in this case?

  100 hidden units: $(128 \times 128 \times 3) \times 100 + 100 = 4,915,300$ parameters.

  1000 hidden units: $49,153,000$ parameters.

Convolutional layer parameters

- An image input $128 \times 128 \times 3$
- 100 kernels of size $5 \times 5 \times 3$
- Number of parameters of this layer?

  For every filter: $5 \times 5 \times 3 + 1 = 76$ parameters.

  Total parameters: $100 \times 76 = 7600$ parameters.
- Think about how many parameters of a fully-connected layer with 100 and 1000 hidden units in this case?

  100 hidden units: $(128 \times 128 \times 3) \times 100 + 100 = 4,915,300$ parameters.

  1000 hidden units: $49,153,000$ parameters.

Learned feature hierarchy

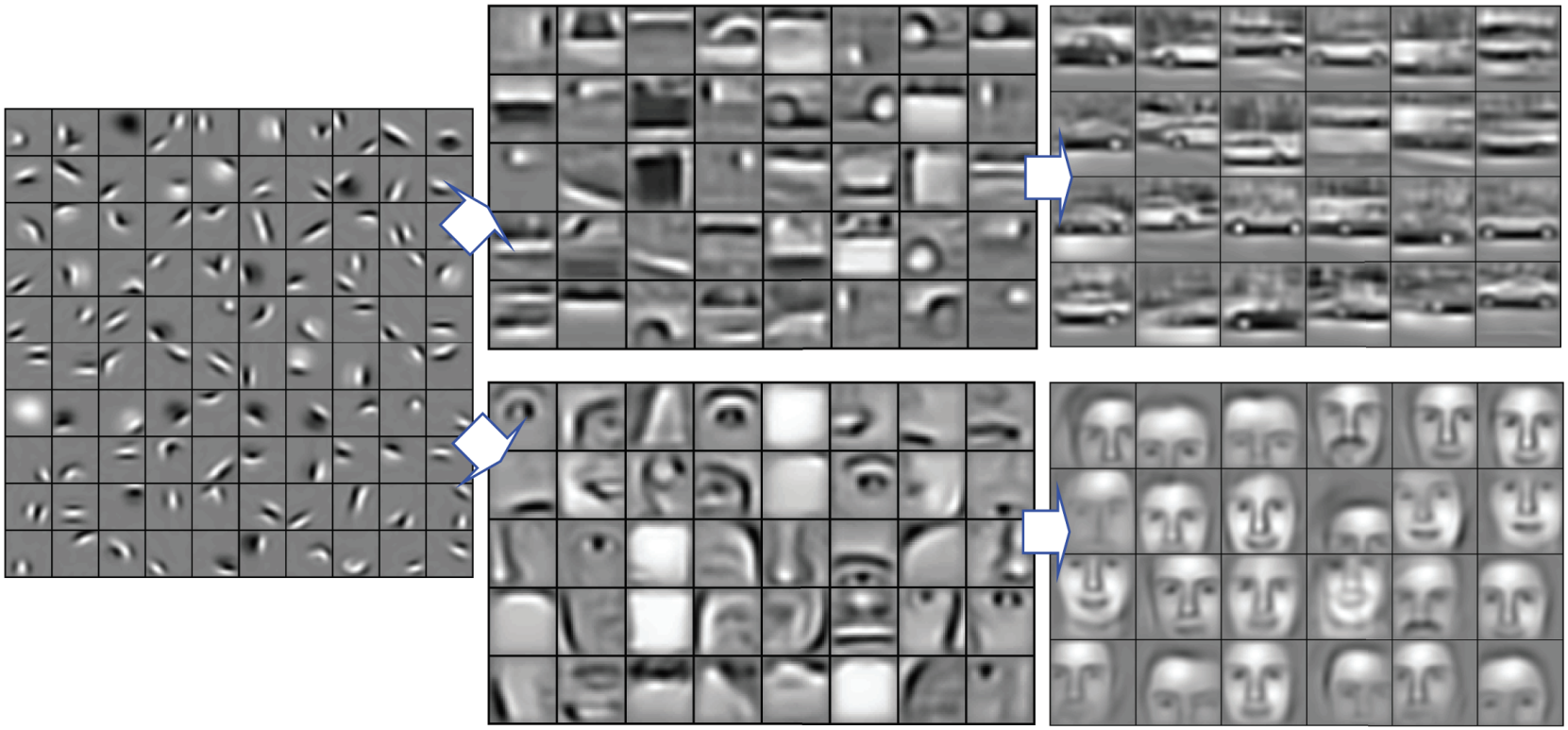
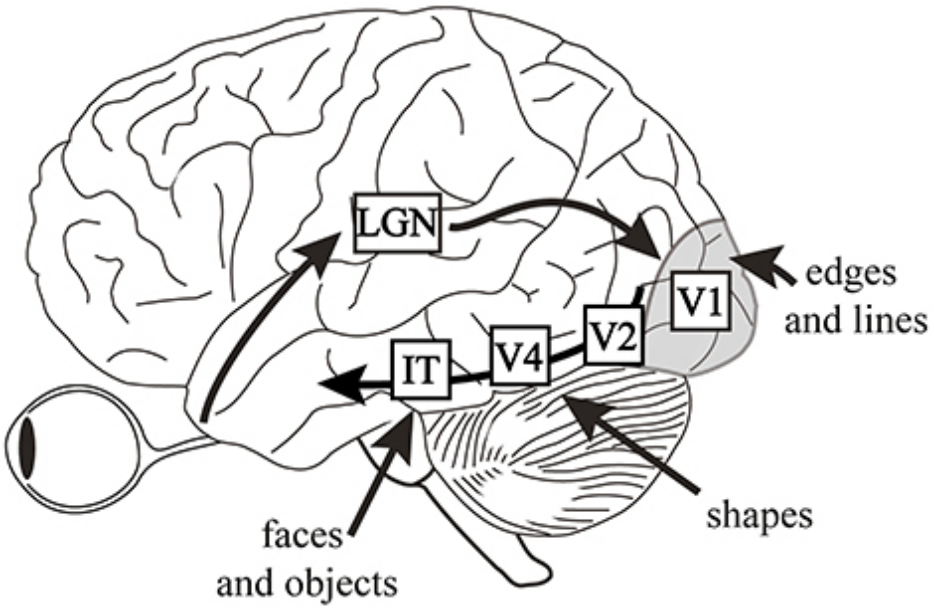
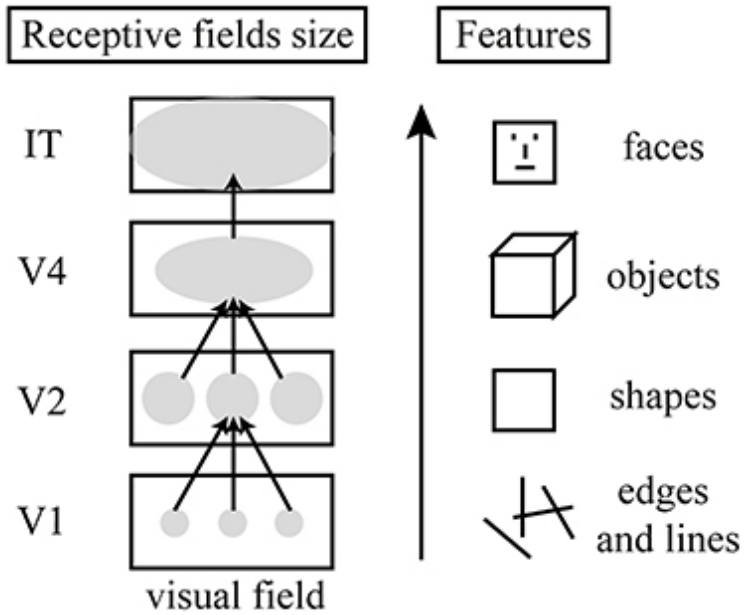


Figure adapted from [Lee, 2009]

Hierarchical processing of visual cotex

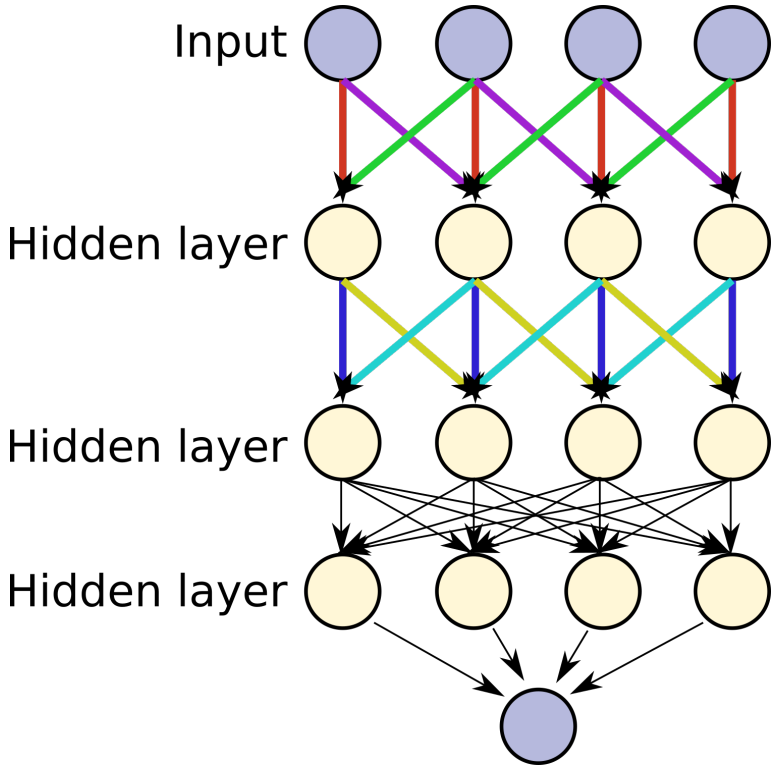




Adapted from Herzog & Clarke, 2014

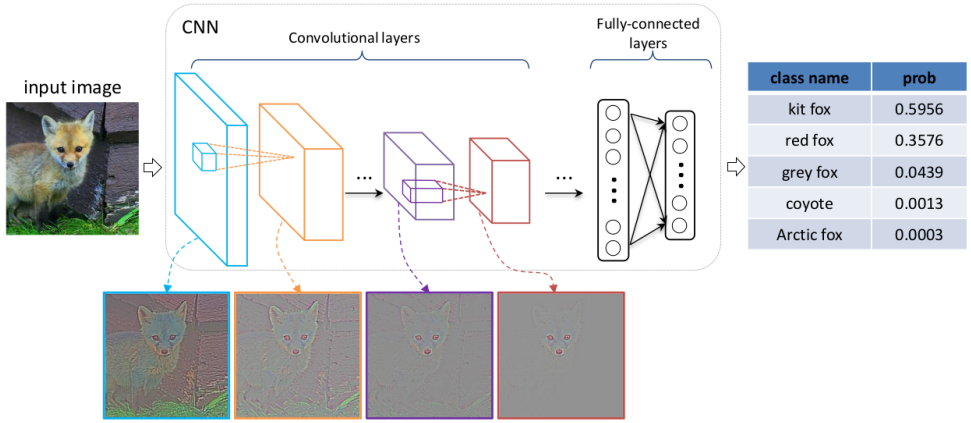
Dense layers

Also known as: fully-connected (FC) layers



Dense layers

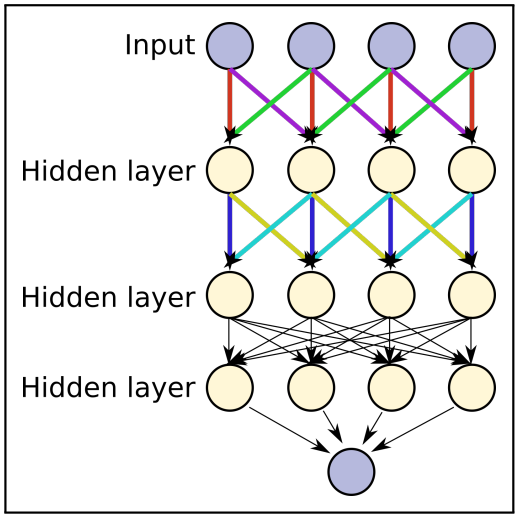
Also known as: fully-connected (FC) layers

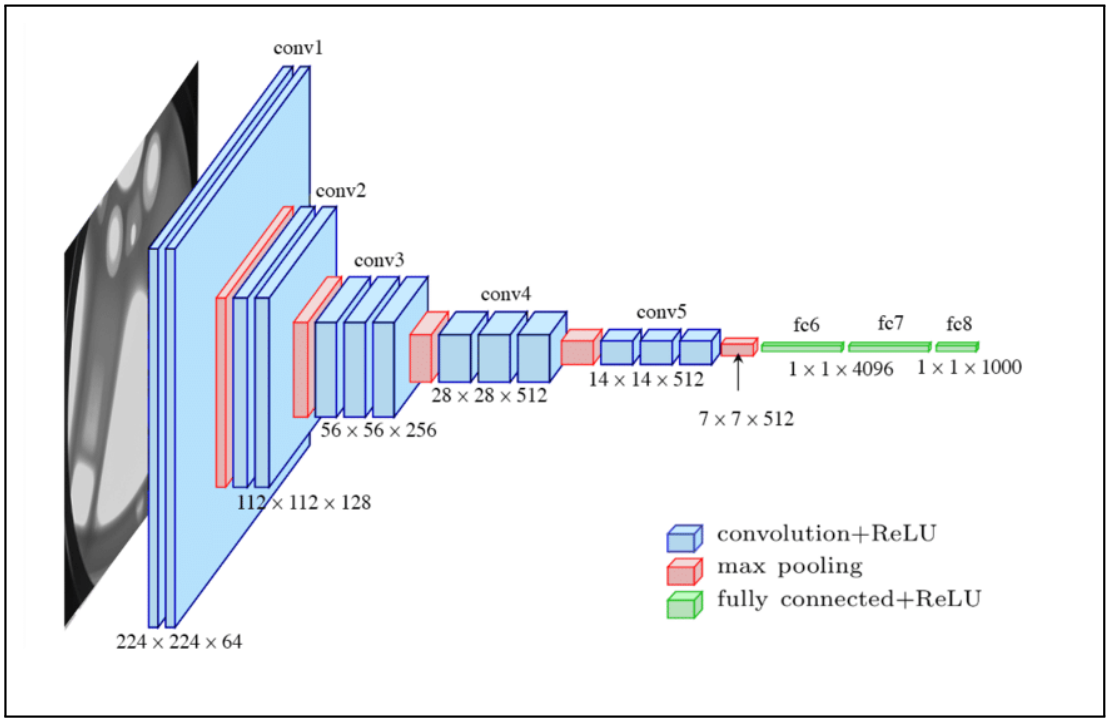


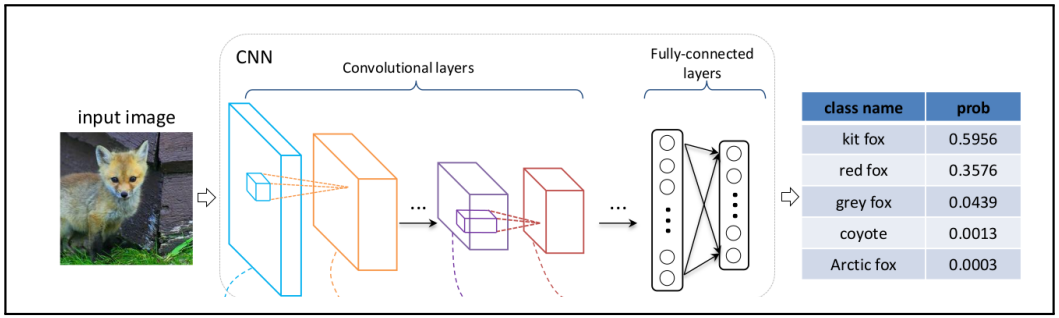
Typically 2 or 3 as end of CNN classifier. Why?

CNN diagrams...!

CNN diagrams...!

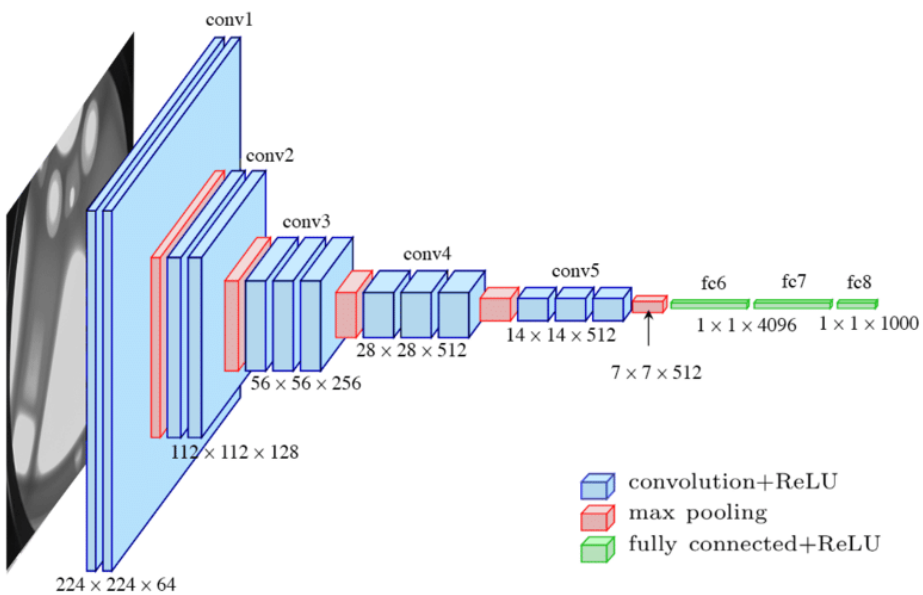






Max-pooling

Downsampling. Shift invariance.



alternatives: mean-pooling

Question: why max?

Invariance and equivariance

For some operator $a(\cdot)$,
Invariance:

$$f(a(x)) = f(x)$$

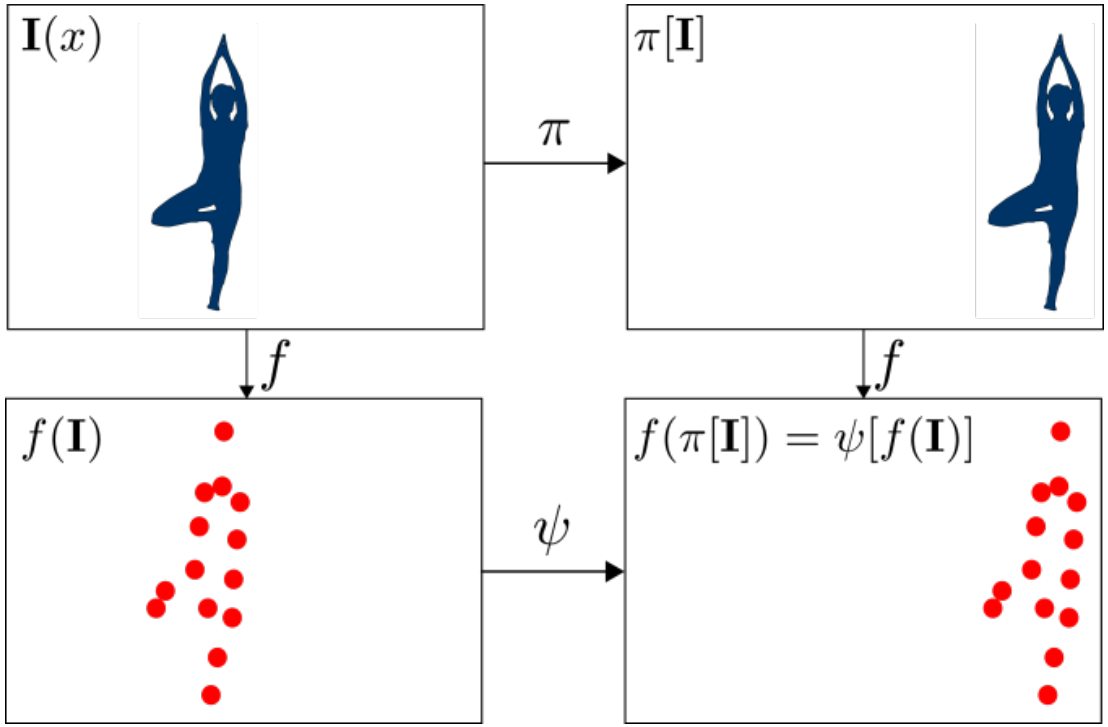
Equivariance:

$$f(a(x)) = a(f(x))$$

See also: Group theory

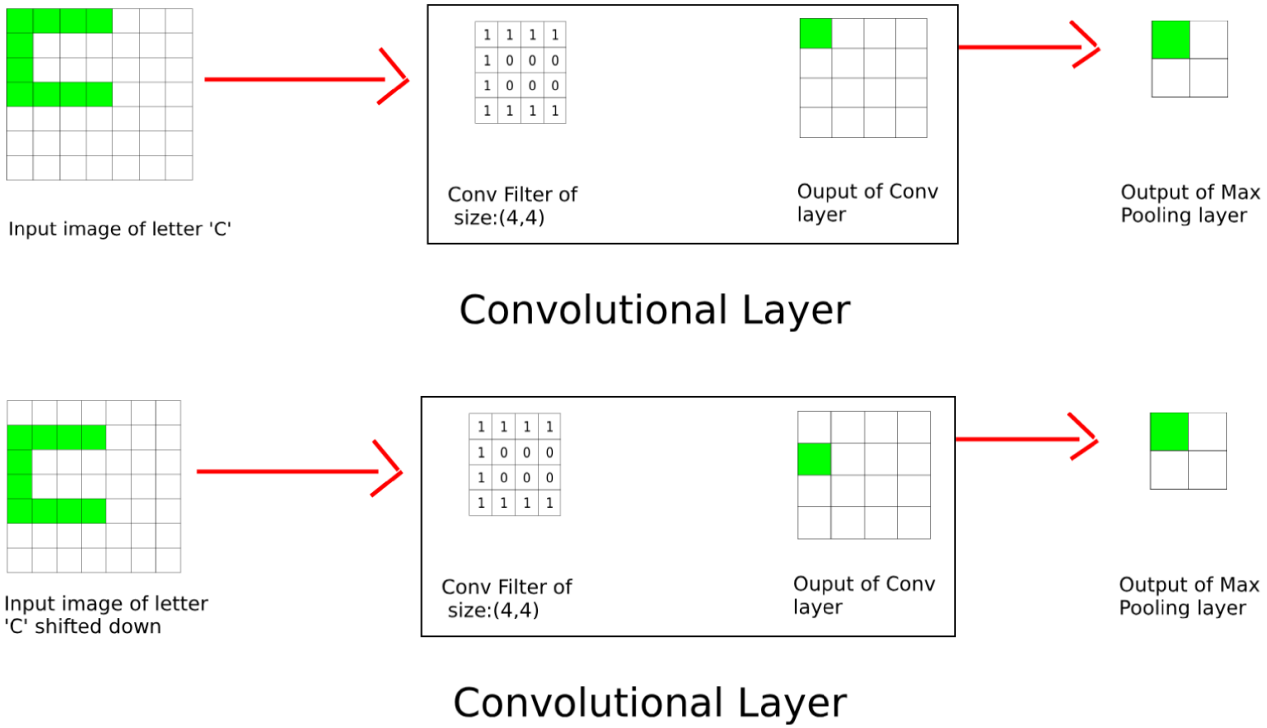
Equivariance

... because of weight sharing



Invariance

... because of max pooling



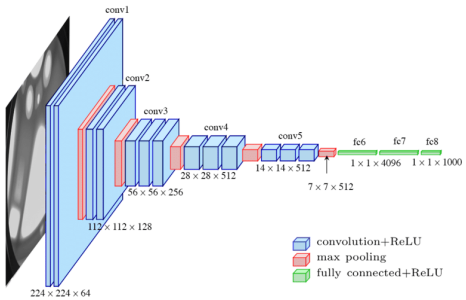
Typical CNN architectures for audio

A quick tour...

(1) Classify a spectrogram

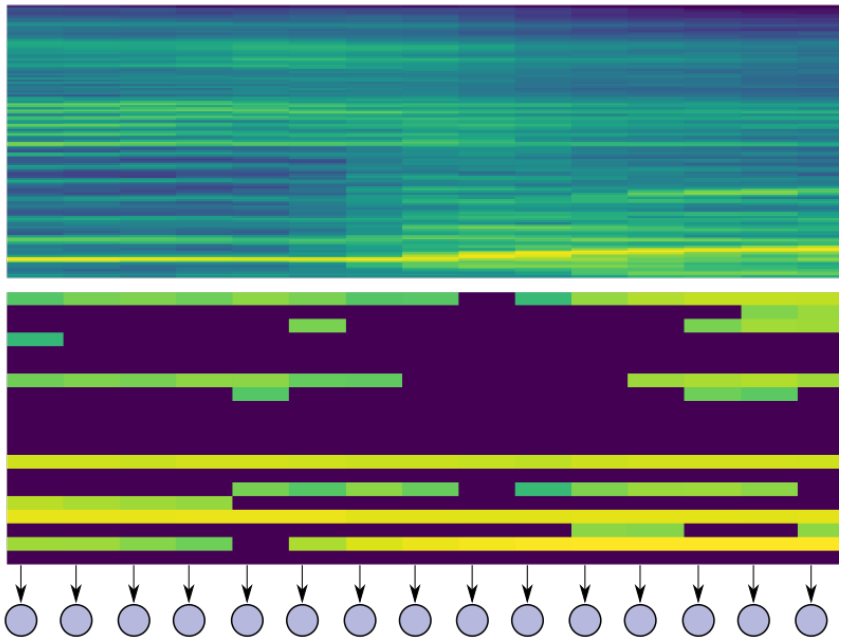
- Spectrogram as input
- 2D real-valued matrix
- “as if it was an image”
- Many alternatives to spectrogram as input—to be discussed later—but this is overwhelmingly common and successful
- Many tasks “treated as classification”

(of genre, speaker, species, ...)



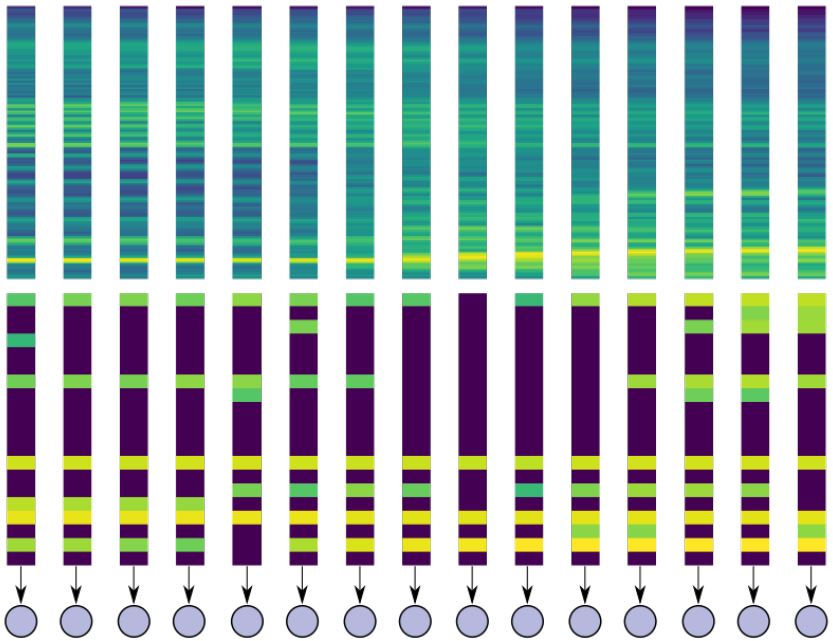
(2) Classify each column of a spectrogram

- Automatic speech recognition (ASR)
- Tracking the 'note' (pitch) of musical instruments



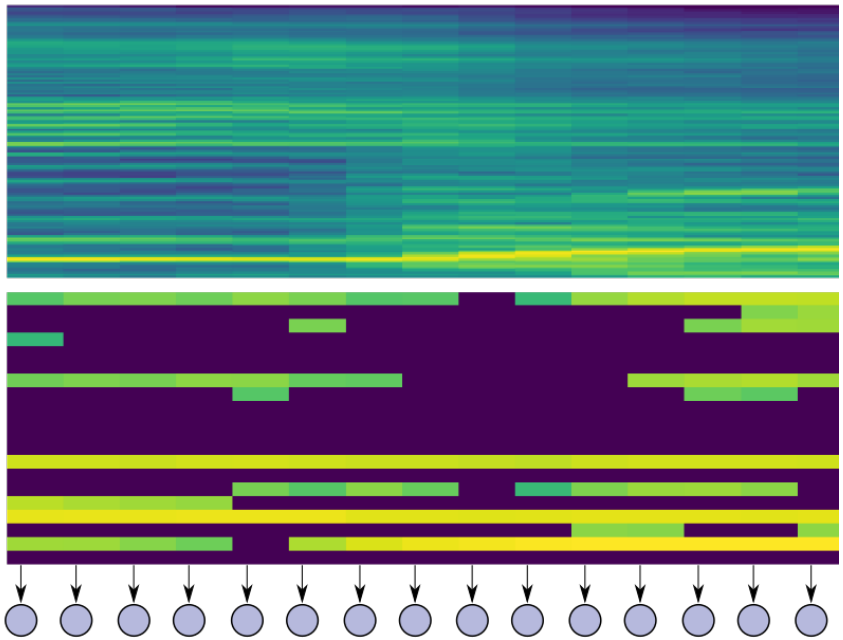
(2) Classify each column of a spectrogram

- Automatic speech recognition (ASR)
- Tracking the 'note' (pitch) of musical instruments



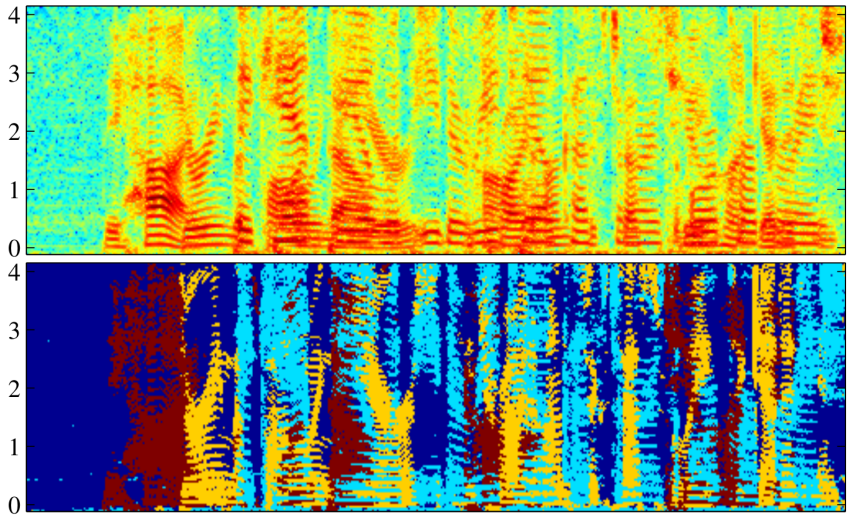
(2) Classify each column of a spectrogram

- Automatic speech recognition (ASR)
- Tracking the 'note' (pitch) of musical instruments

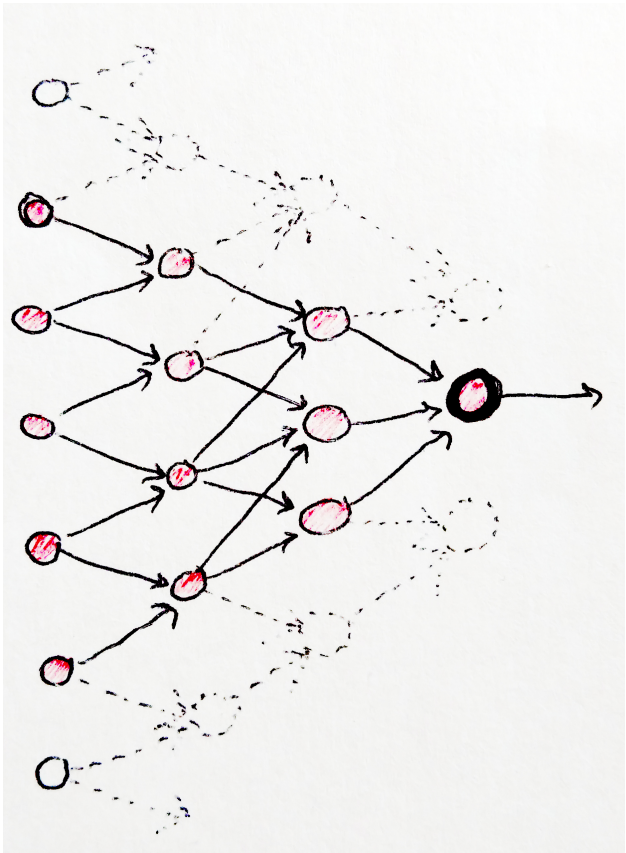


(3) Classify each pixel of a spectrogram

- Detecting acoustic events ('blobs')
- Source separation (by estimating which pixels should be active)



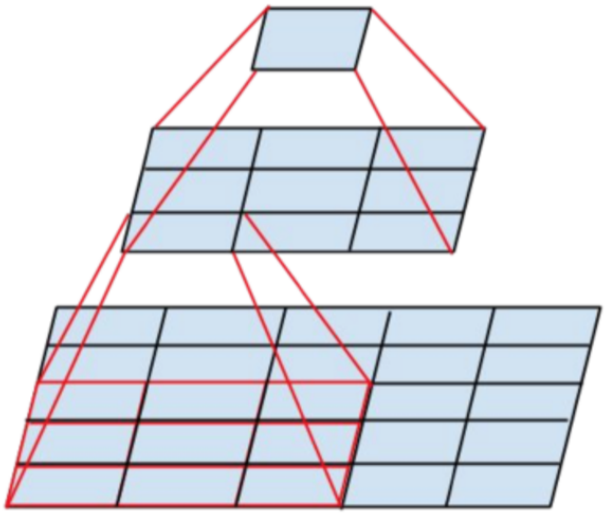
Receptive fields



"Receptive field"
for a selected neuron:

The subset of input features
whose values affect
that neuron's value.

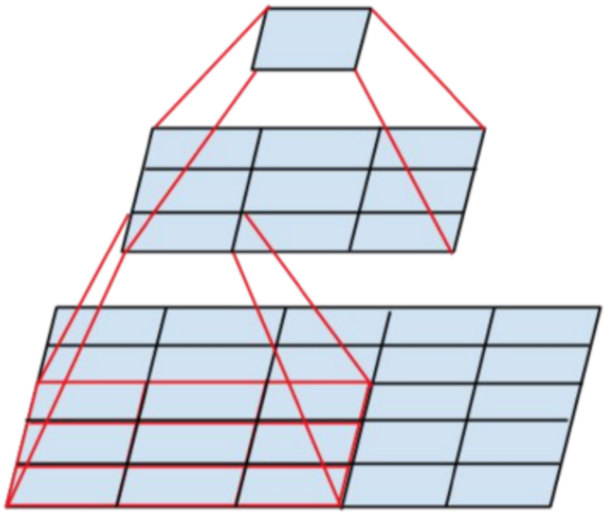
Receptive fields



"Receptive field"
for a selected neuron:

The subset of input features
whose values affect
that neuron's value.

Receptive fields

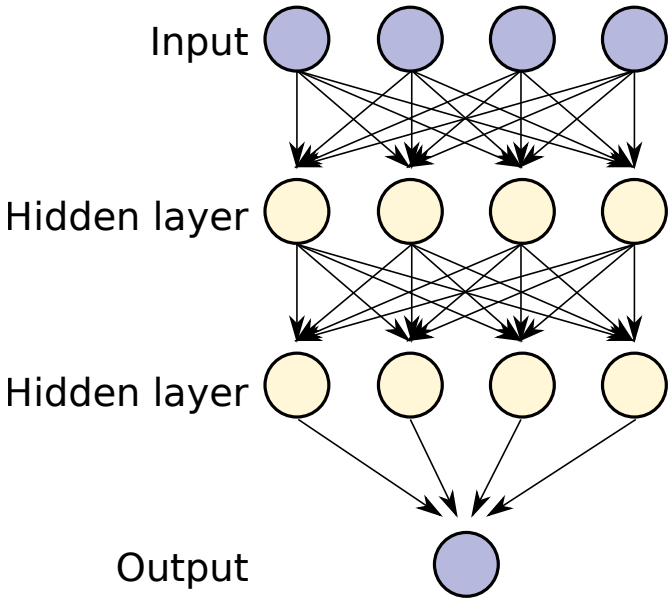


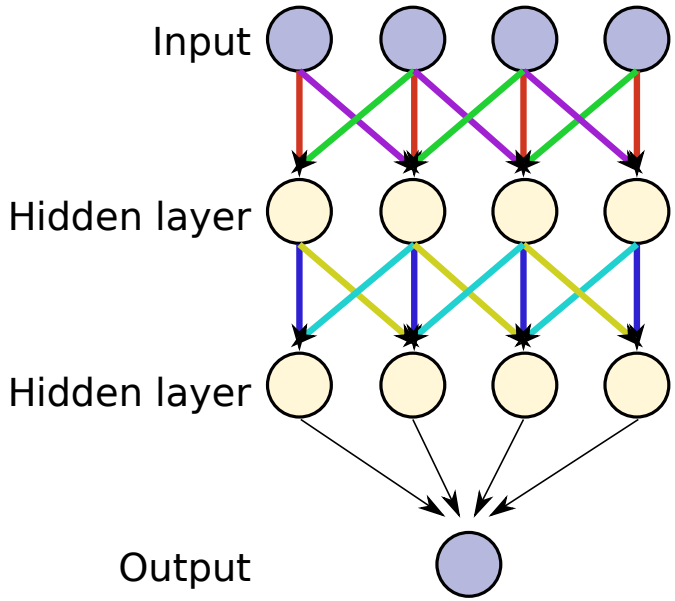
"Receptive field"
for a selected neuron:

The subset of input features
whose values affect
that neuron's value.

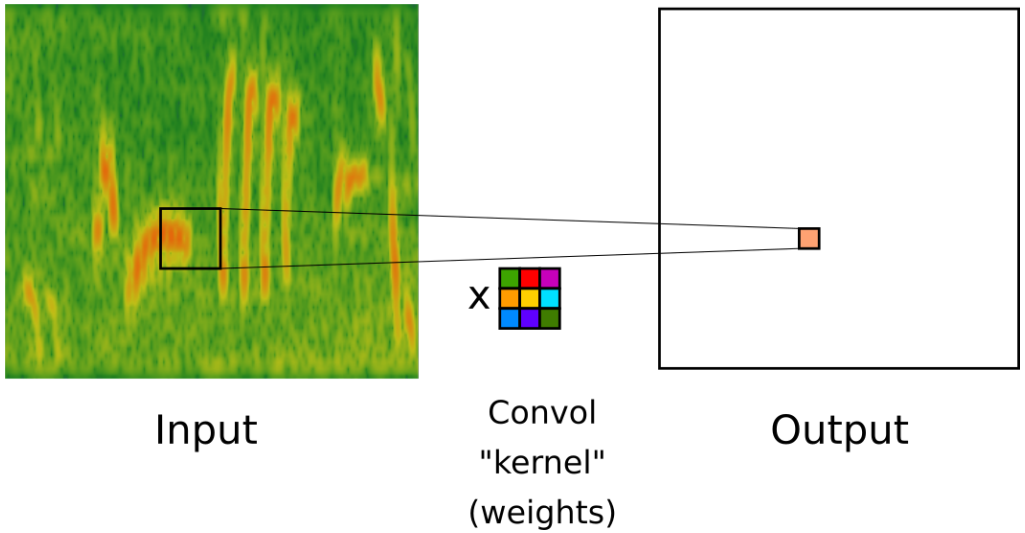
Consider the previous examples

Receptive fields





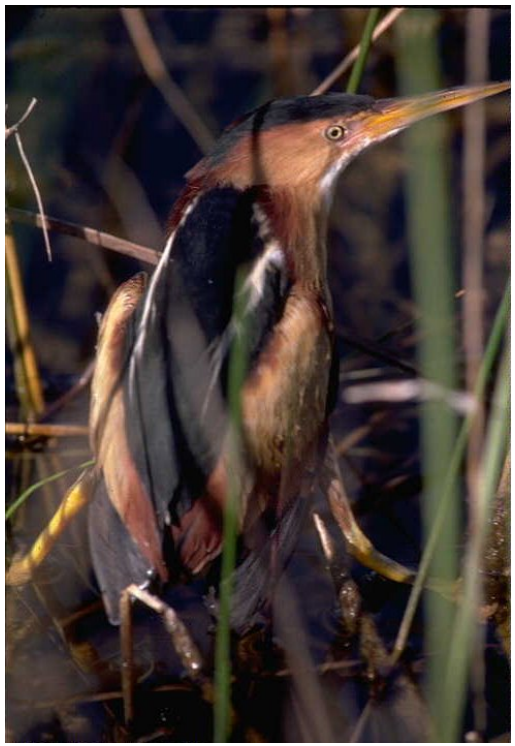
Receptive fields – in time-frequency



The spectrogram settings (sample rate, frame length, hop) also affect the receptive field

Receptive fields – the least bittern

The Least Bittern



- 'coo coo coo' breeding call: 0.5 seconds, 500–600 Hz

Receptive fields – the least bittern

- ‘coo coo coo’ breeding call:
  - 0.5 seconds, 500–600 Hz
  - Recorded at 44.1 kHz
  - Spectrogram: frame size 1024, 512 hop
  - 3x3 conv →2x2 maxpool →3x3 conv

Calculate: bandwidth; frame rate; receptive field needed...

What's more?

- Strides
- Padding
- Output size
- Backpropagation with convolution layers

To recap:

To recap:

We started with an extremely simple network

- Layers
- Nonlinearities, loss function, gradient descent

Made it convolutional

- Constrains its flexibility
- Reduces num trained parameters
- Enforces some general principles
  - Locality
  - Shift invariance/equivariance
- Maxpooling

Applied it to spectrogram data

- Typical architectures
- Receptive fields

To recap:

We started with an extremely simple network

- Layers
- Nonlinearities, loss function, gradient descent

Made it convolutional

- Constrains its flexibility
- Reduces num trained parameters
- Enforces some general principles
  - Locality
  - Shift invariance/equivariance
- Maxpooling

Applied it to spectrogram data

- Typical architectures
- Receptive fields

To recap:

We started with an extremely simple network

- Layers
- Nonlinearities, loss function, gradient descent

Made it convolutional

- Constrains its flexibility
- Reduces num trained parameters
- Enforces some general principles
  - Locality
  - Shift invariance/equivariance
- Maxpooling

Applied it to spectrogram data

- Typical architectures
- Receptive fields

To recap:

We started with an extremely simple network

- Layers
- Nonlinearities, loss function, gradient descent

Made it convolutional

- Constrains its flexibility
- Reduces num trained parameters
- Enforces some general principles
  - Locality
  - Shift invariance/equivariance
- Maxpooling

Applied it to spectrogram data

- Typical architectures
- Receptive fields

Next time...

Audio representations