



# Deep Learning for Life Sciences

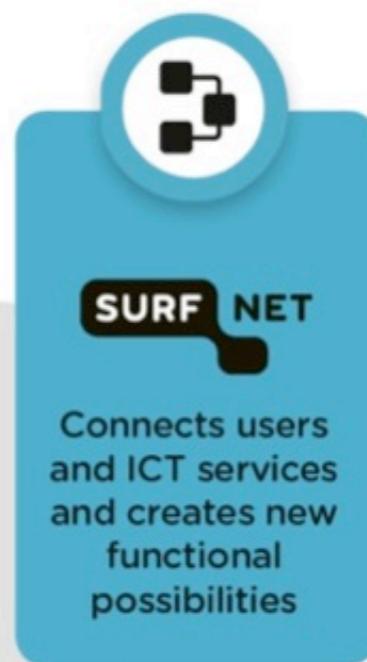
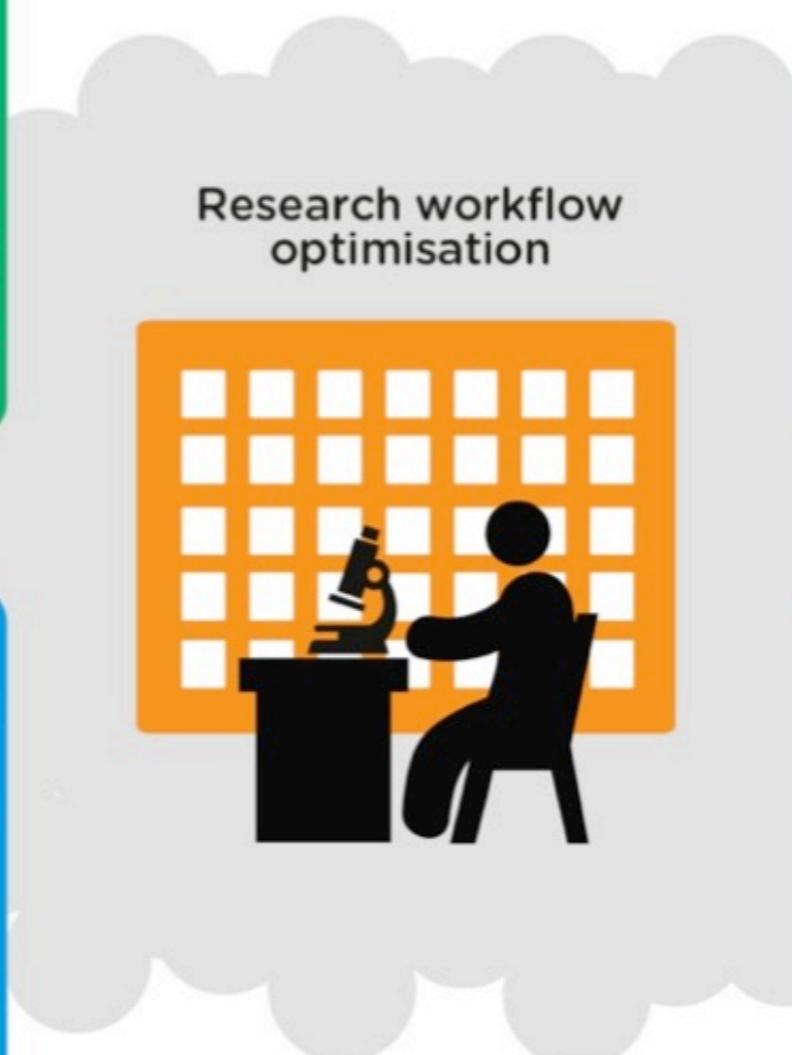
Matthijs Moed  
Haukur Páll Jónsson  
Carlos Teijeiro Barjas



# Today's schedule

9:00 - 9:30	Deep learning (in the life sciences)
9:30 - 10:15	 Your first neural network
<b>10:15 - 10:30</b>	<b>Break</b>
10:30 - 11:00	Deep learning's knobs and dials
11:00 - 12:00	 Improving your network - hyperparameter tuning
<b>12:00 - 13:00</b>	<b>Lunch break</b>
13:00 - 13:30	Convolutional neural networks
13:30 - 14:30	 Your first convolutional neural network
<b>14:30 - 14:45</b>	<b>Break</b>
14:45 - 15:15	Regularisation and advanced techniques
15:15 - 16:15	 Improving neural networks
16:15 - 17:00	Consultancy time: discuss your own problem!

# SURF

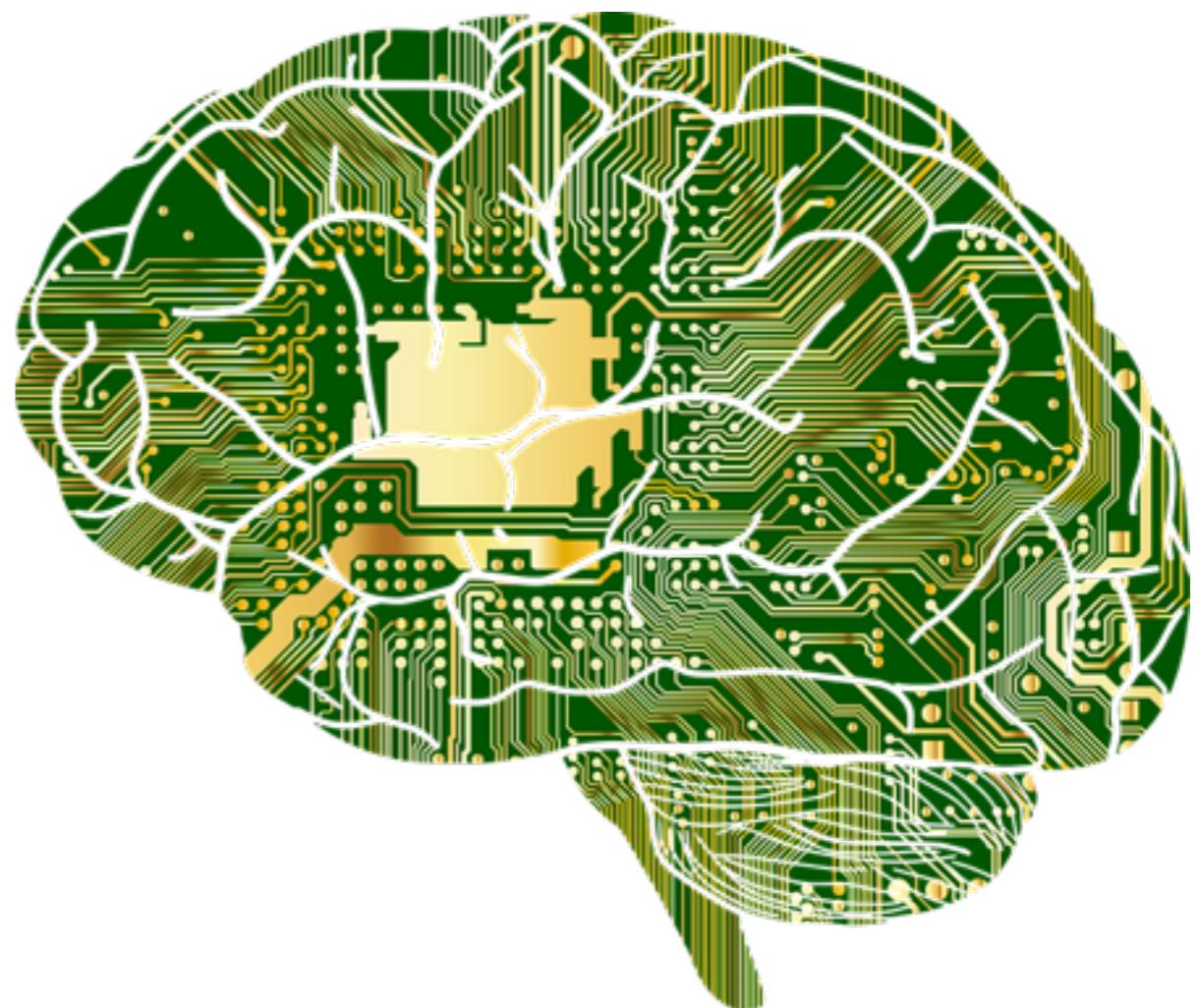


# SURF services

-  High-end computing in different flavours
-  Data storage and management solutions
-  Fast networks for data transfer and scaling out
-  Federated access management solutions
-  Facilities to share data and to collaborate online
-  Visualisation services

# Deep learning @ SURF

- GPUs on different infrastructures
- HPC Cloud
- LISA cluster
- Cartesius supercomputer
- Innovation projects
- Upscaling of deep learning models
- Deep learning for simulation
- Workshops & courses



# Goals

- Find out what deep learning can do in the life sciences
- Understand the components of a neural network
- Solve a real-life image classification problem

# Deep learning

**airplane**



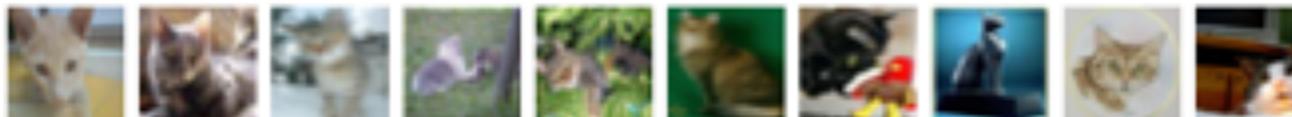
**automobile**



**bird**



**cat**



**deer**



• ;

**dog**



**frog**



**horse**



**ship**



**truck**



# Style transfer



Source: [Deep Dream Generator](#)

# Text generation



**DeepDrumpf**

@DeepDrumpf

Follow



Replying to @jimjefferies

If I win, I'm going to instruct my attorney general to put racists in charge. We are going to do great crimes with my programs. @jimjefferies

6:31 PM - 9 Oct 2016

157 Retweets 266 Likes



3



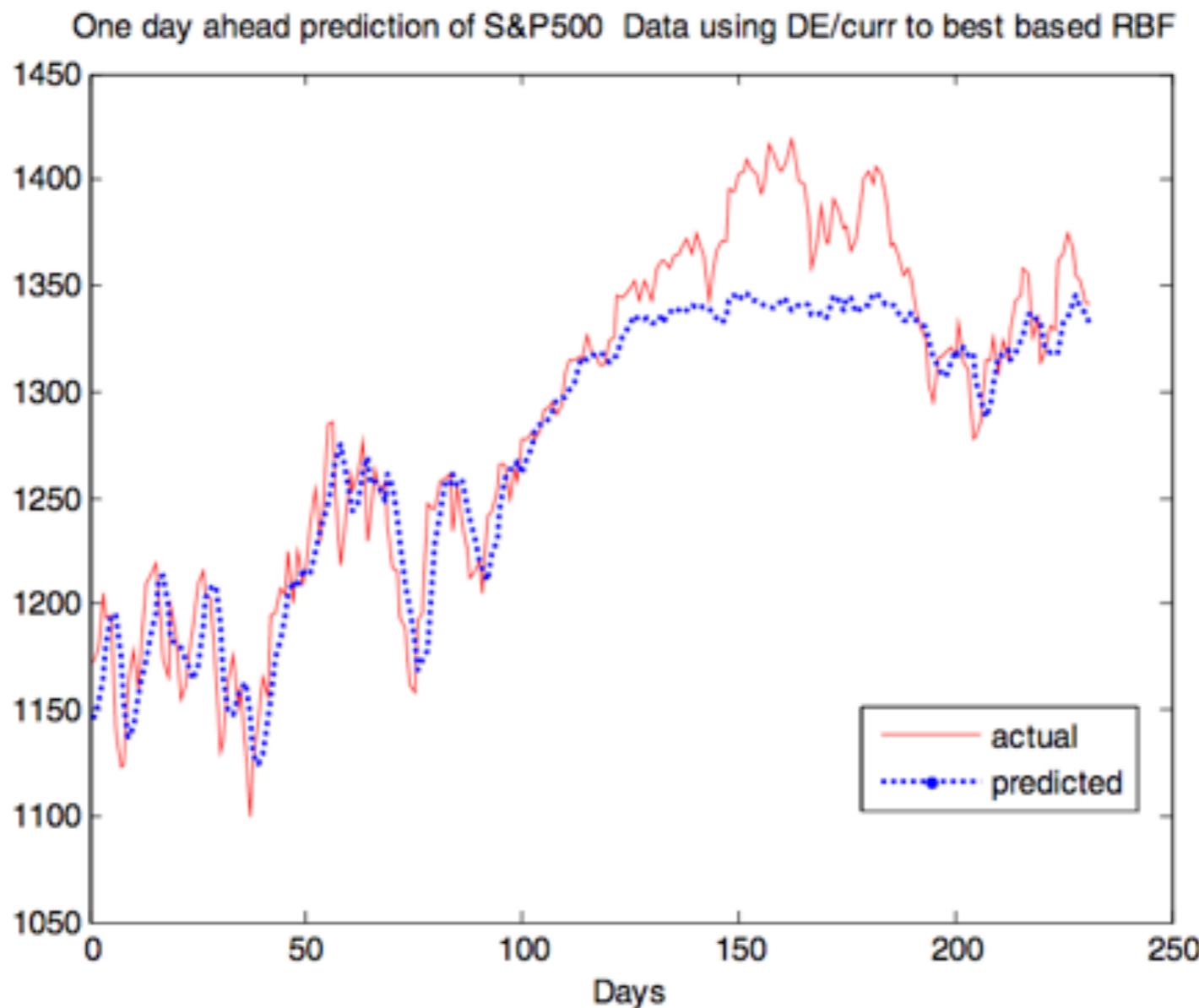
157



266

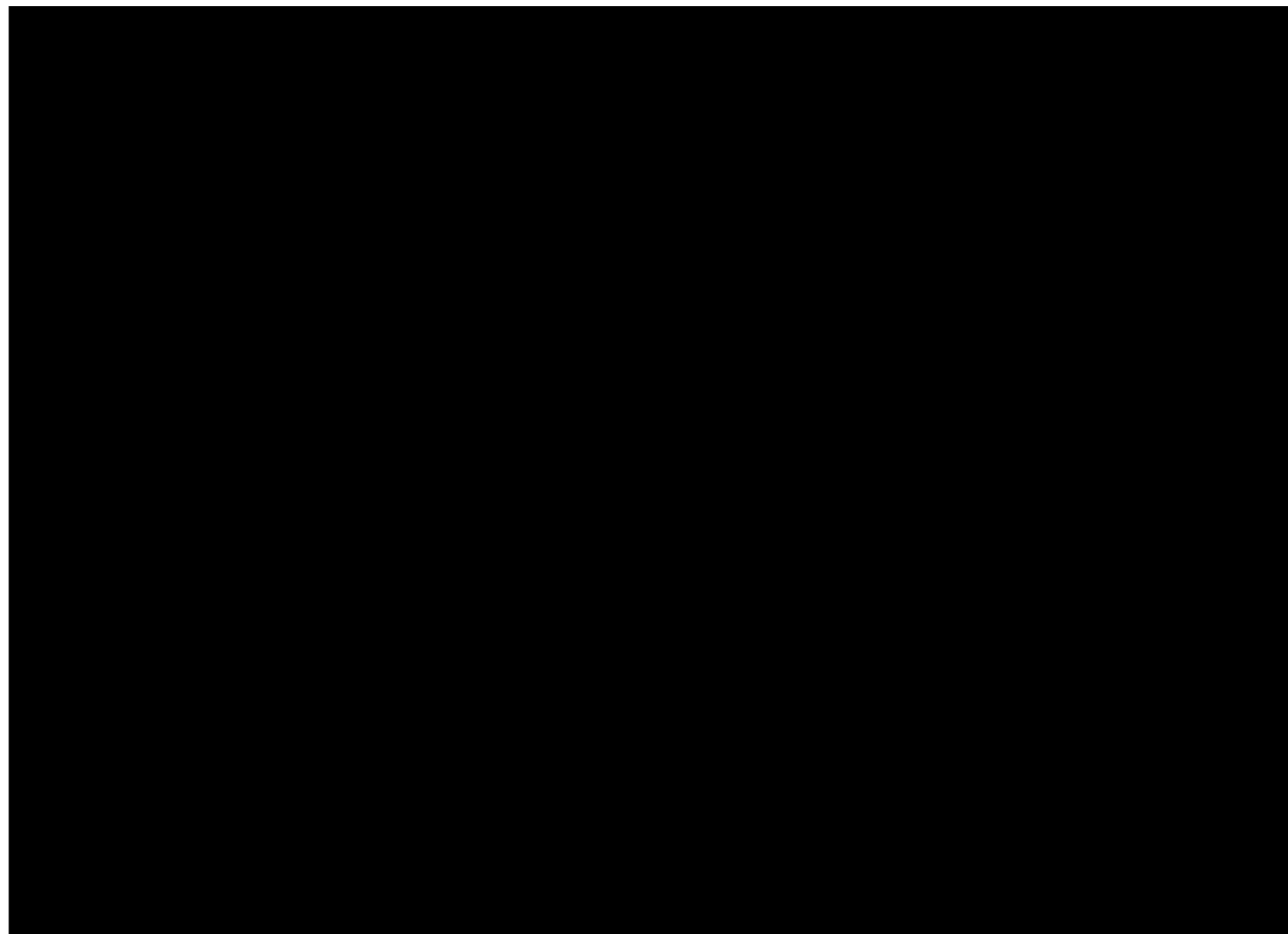
Source: [Deep Drumpf](#)

# Time series prediction



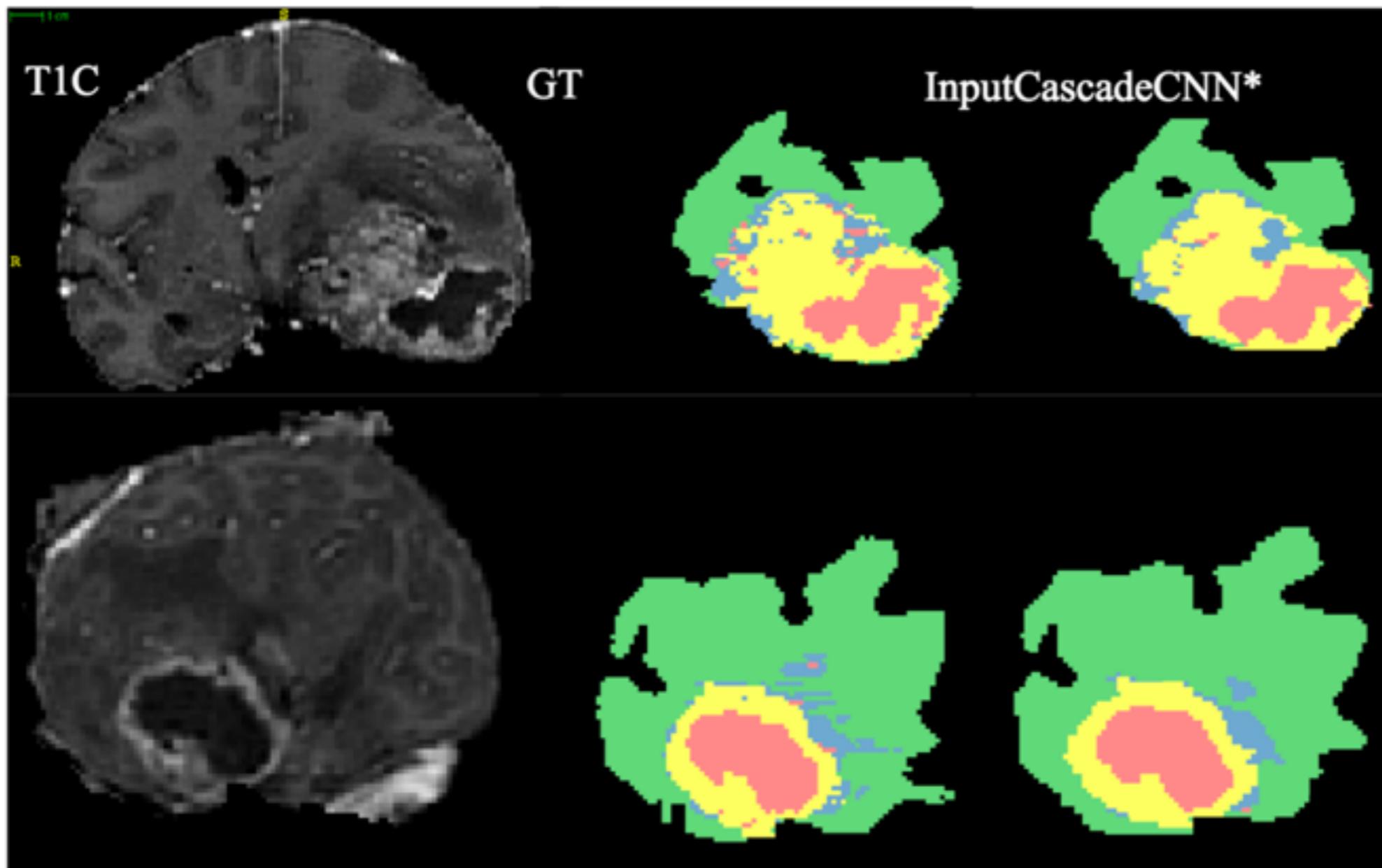
Source: [Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach](#),  
Rout et al., 2015

# Reinforcement learning



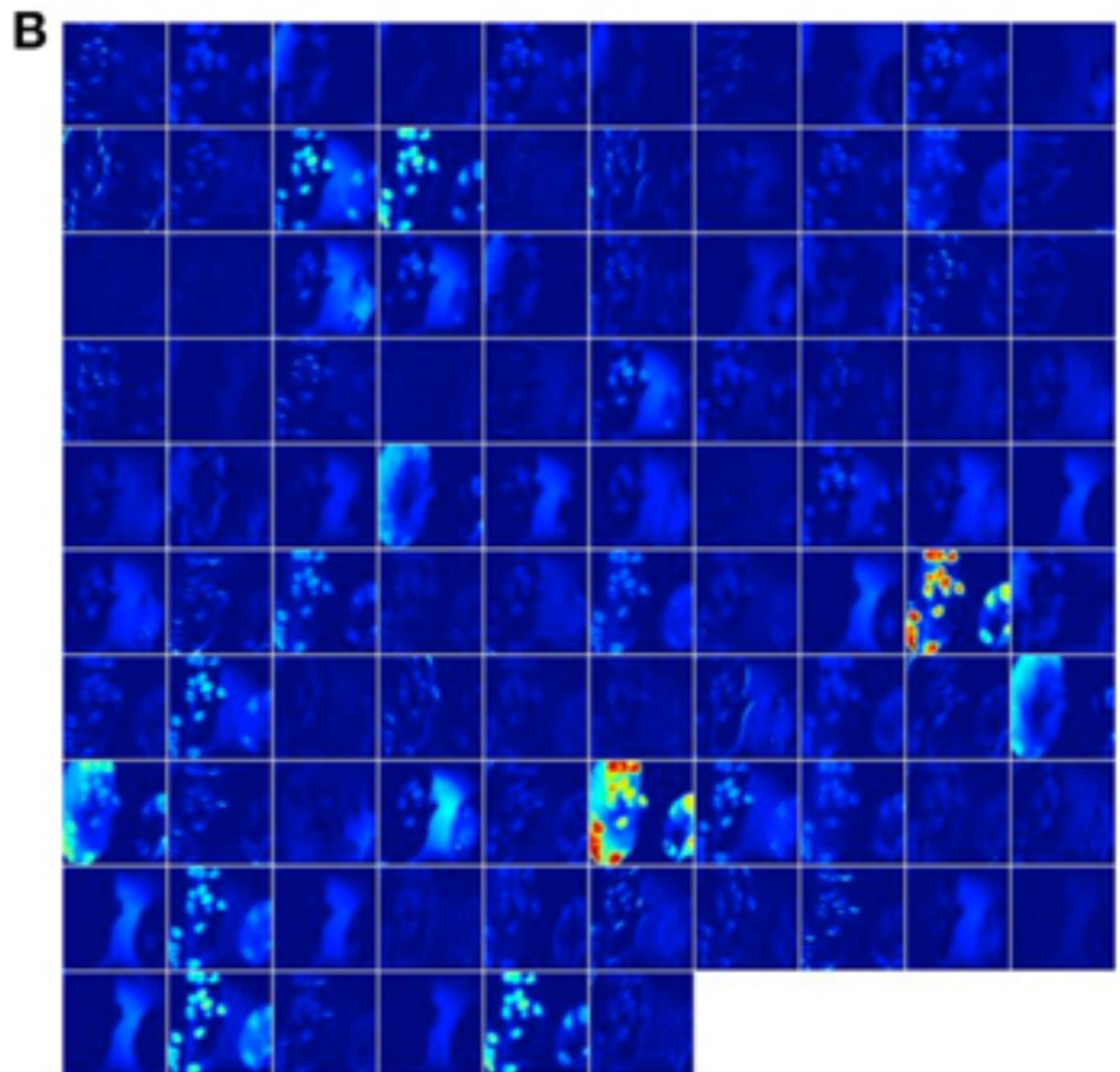
Source: [Emergence of Locomotion Behaviours in Rich Environments](#), DeepMind, 2017

# Medical image segmentation



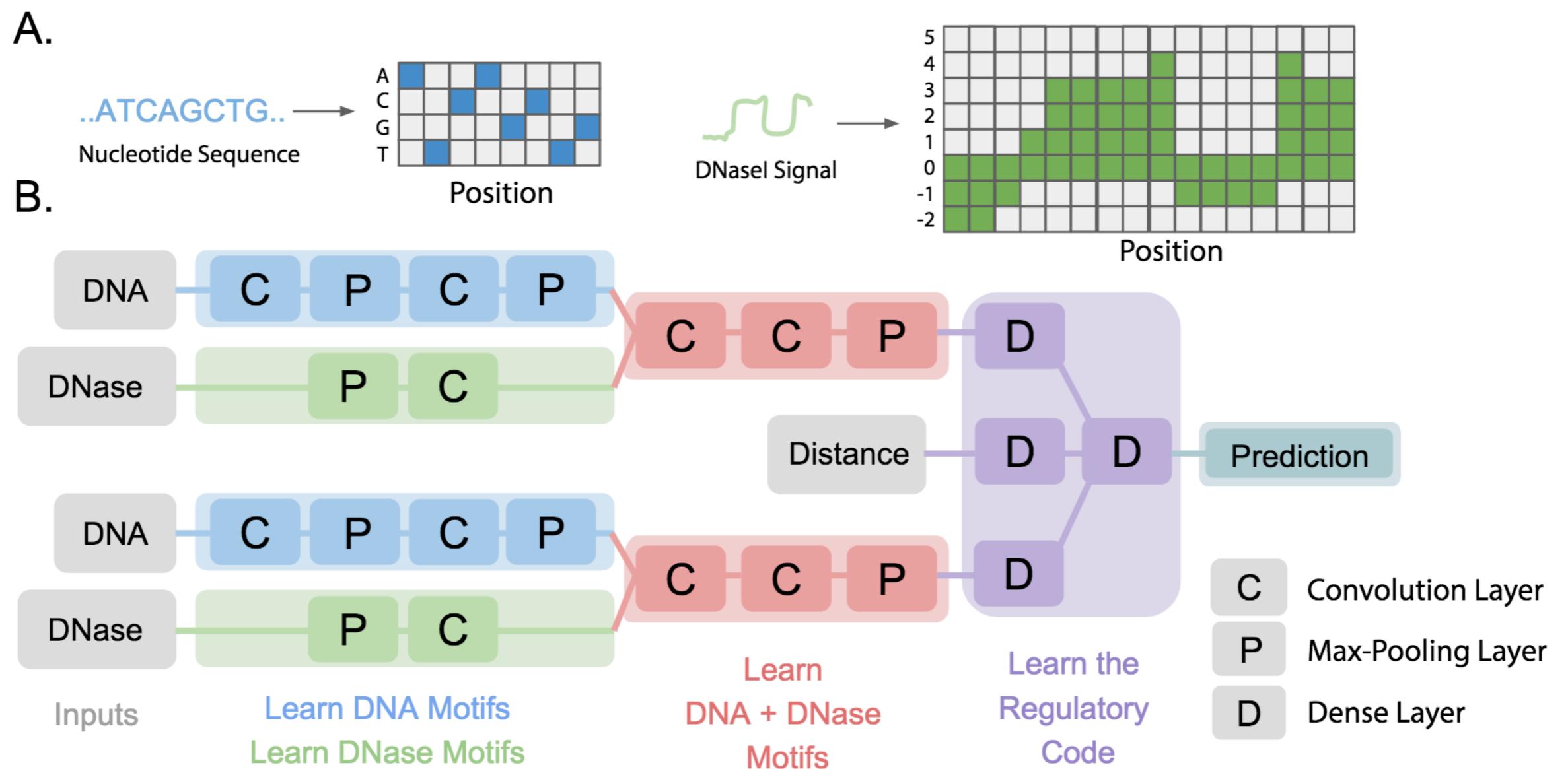
Source: [Brain Tumor Segmentation with Deep Neural Networks](#), Havaei et al., 2016

# Disease detection



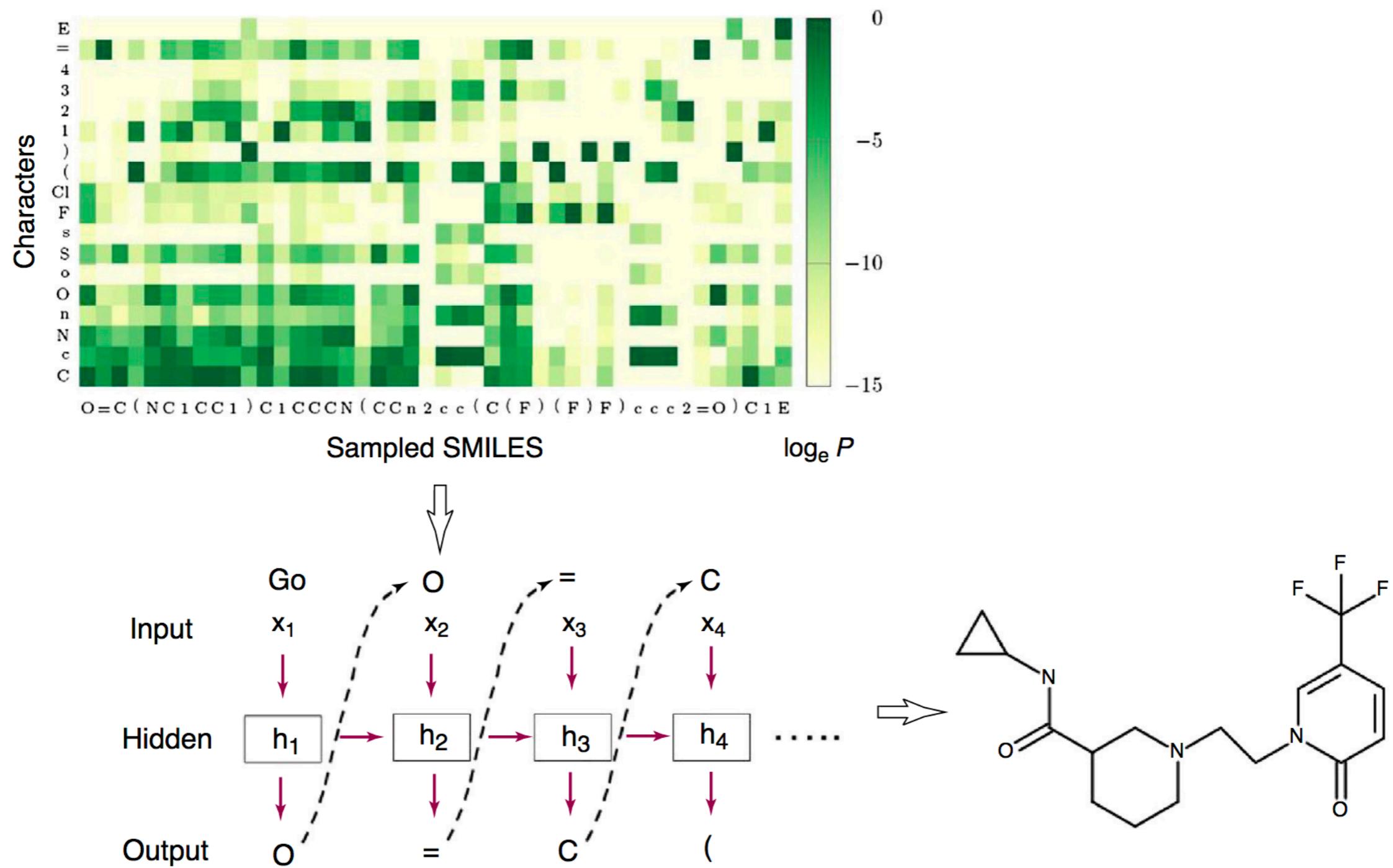
Source: [Using Deep Learning for Image-Based Plant Disease Detection](#), Mohanty et al., 2016

# Omics



Source: [Nucleotide sequence and DNaseI sensitivity are predictive of 3D chromatin architecture](#),  
Schreiber et al., 2017

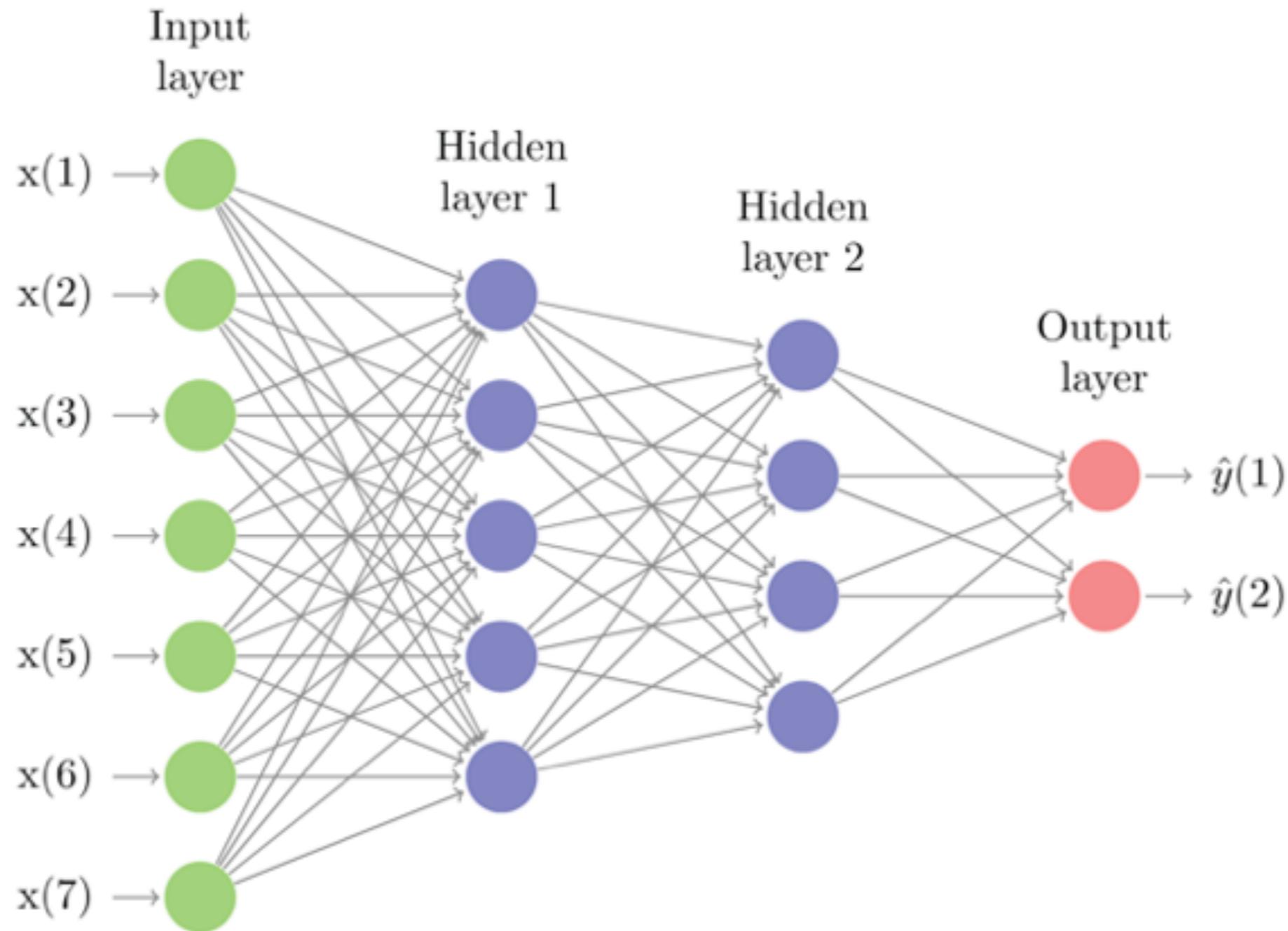
# Drug design



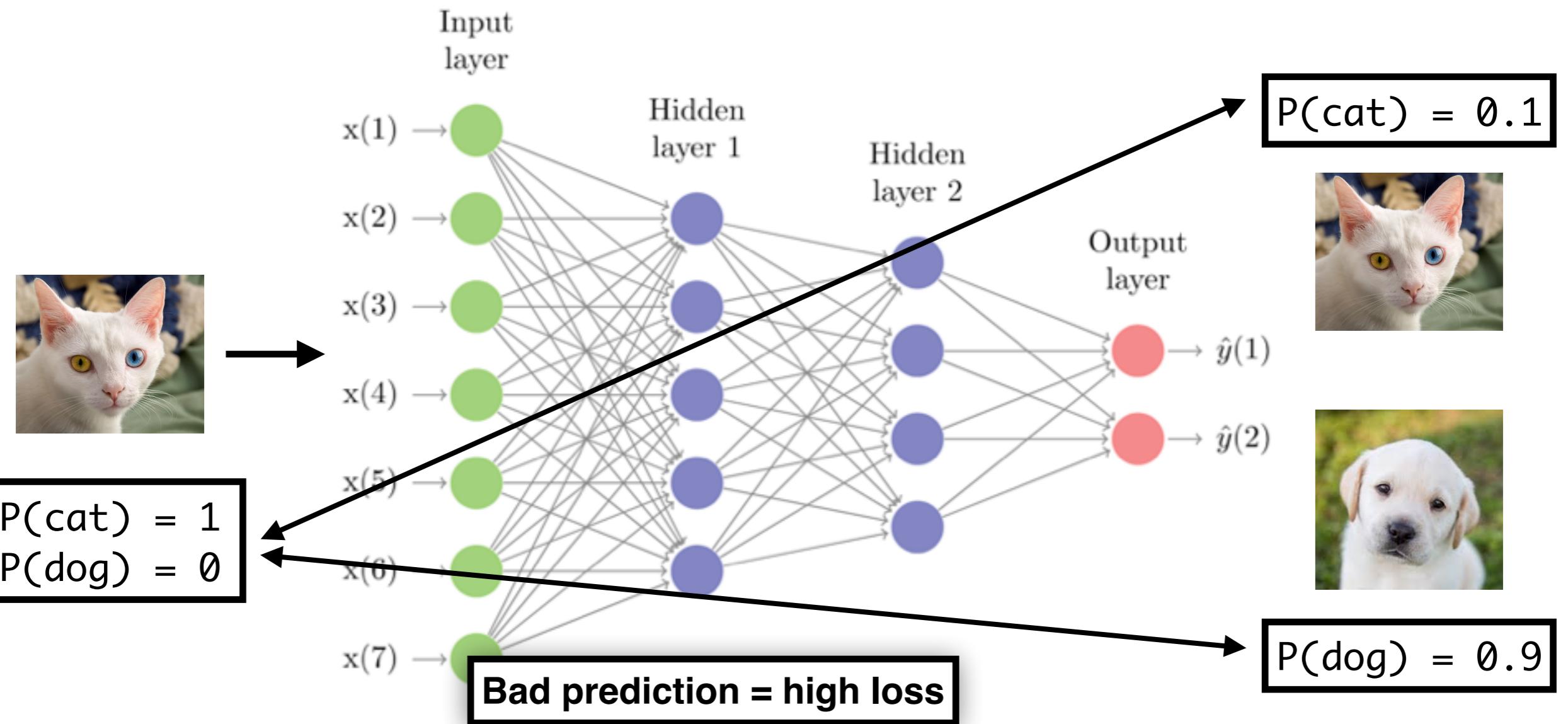
Drug Discovery Today

Source: [The rise of deep learning in drug discovery](#). Chen et al., 2018

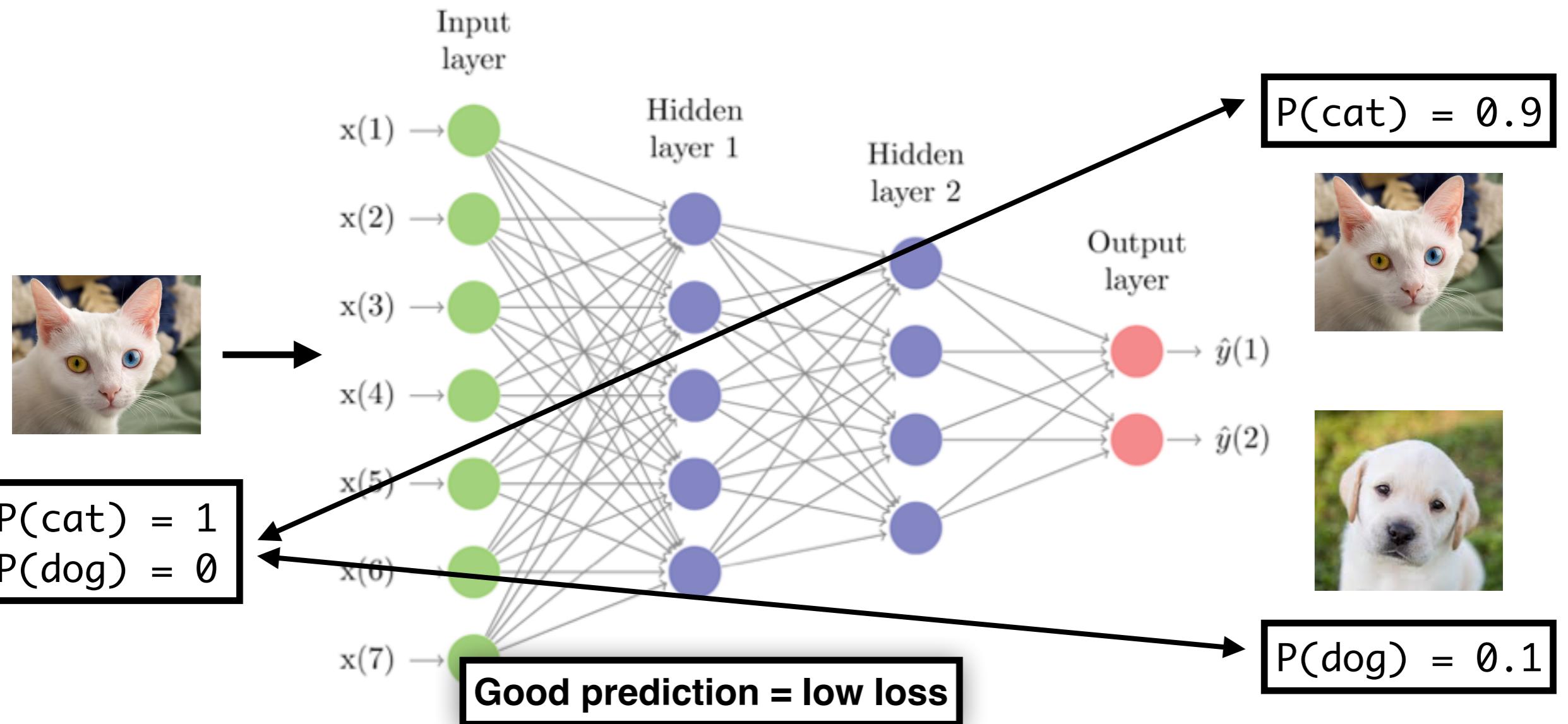
# Neural networks



# Example



# Example

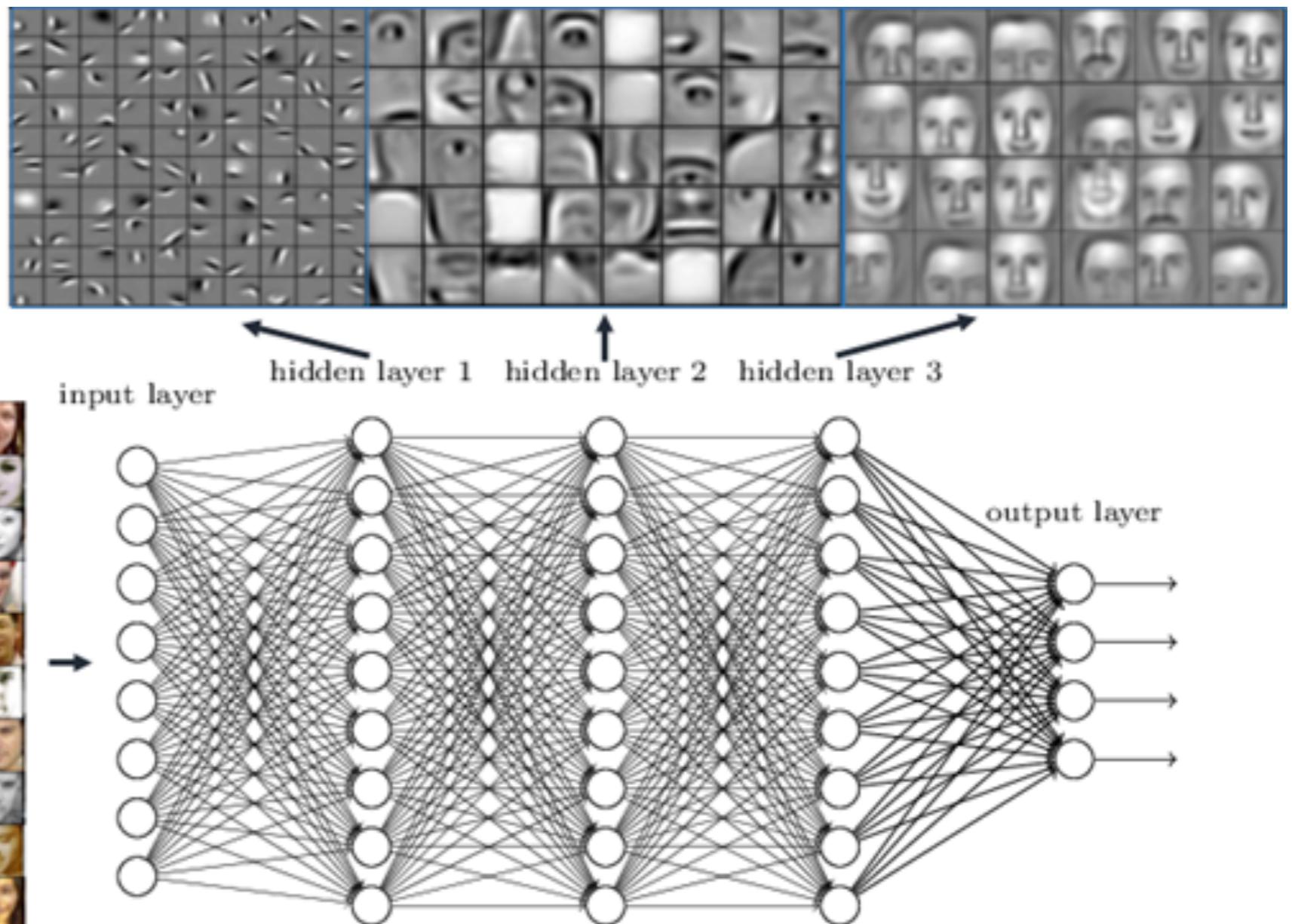


# Learners and loss functions

- A neural network can learn to solve a problem
- Learn representations from data
- Every layer applies a simple mathematical function
- Adapt parameters of network to reduce loss

# Feature hierarchy

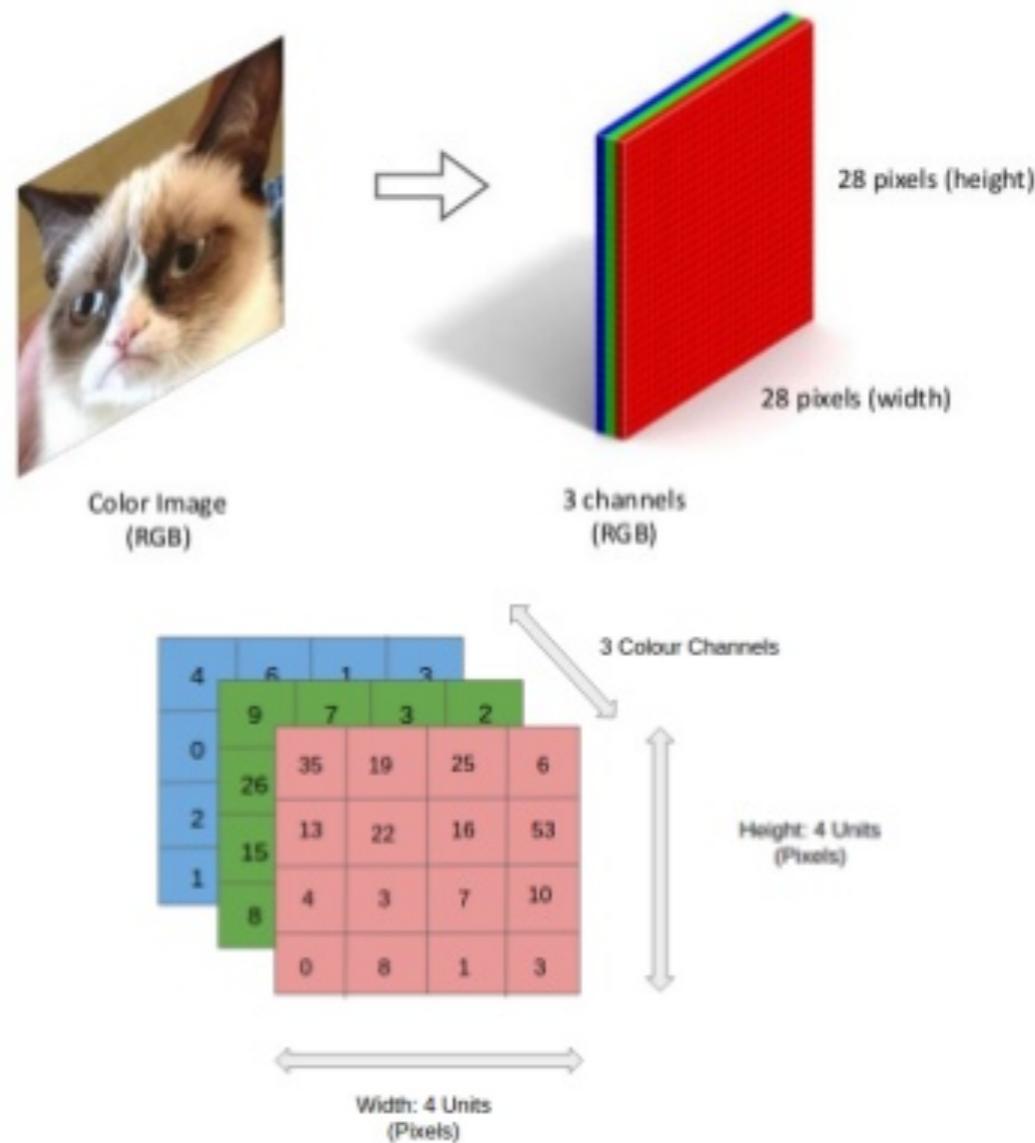
Deep neural networks learn hierarchical feature representations



# Image data

color image is 3rd-order tensor

- With colours = 3D
- 3 color channels
- Instead of just storing single value between 0-255 we store 3 such values.



Source: <https://www.slideshare.net/pabitramani/digital-image-processing-and-interpretation>

# Jupyter notebooks

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** The title bar shows the notebook name "03-your-first-cnn".
- Toolbar:** Includes standard browser-style buttons (back, forward, search) and a URL bar pointing to "https://elixir-dl.sda-projects.nl/user/user40/notebooks/material/03-your-first-cnn.ipynb".
- Menu Bar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- Logos:** "accelerate" logo on the left and "SURF SARA" logo on the right.
- Section Header:** **Your first convolutional neural network**.
- Text:** A brief introduction: "In this notebook you will learn how to build and train a convolutional neural network (CNN) yourself, and to figure out what the network is learning."
- Text:** "We will continue with the leaf classification problem from before, and make it a bit harder. Instead of the ten classes from before, we now have 15, with and build a first CNN to evaluate its performance against a typical dense network."
- Text:** "Let's load the data first. It consists of a training and validation set. Let's print the class names as well:

```
In [ ]: import lib
import keras

%config InlineBackend.figure_format = 'retina'

(X_train, Y_train), (X_val, Y_val), labels = lib.dataset_plant_village_small()

for index, name in enumerate(labels):
    print('{}: {}'.format(str(index).rjust(2), name))
```

- Text:** "Let's have a look at some of the examples in the data set:

```
In [2]: lib.plot_examples(X_train[:32], Y_train[:32], labels);
```

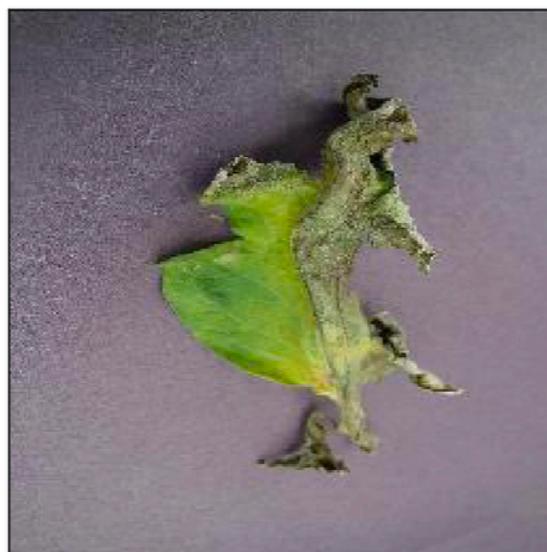
- Image:** Four leaf images labeled: potato.early-blight, potato.healthy, potato.late-blight, and tomato.bacterial-spot.

# The problem

diseased



diseased



diseased



diseased



diseased



diseased



healthy



healthy



# Hands-on

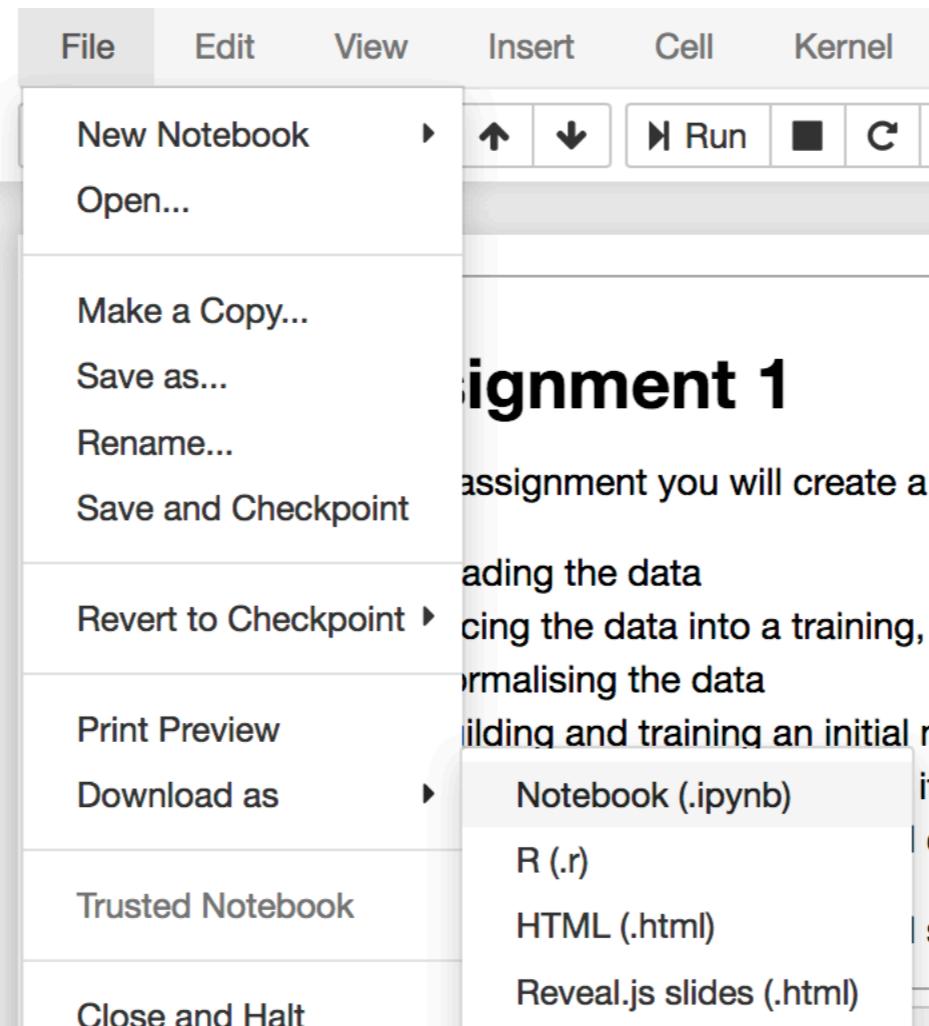


Go to <https://elixir-dl.sda-projects.nl>

Notebook: 01-your-first-neural-network.ipynb

**Break at 10:15**

# Workshop material



<https://github.com/sara-nl/elixir-dl>

# Creating neural networks



- Many frameworks exist;  
**TensorFlow**, **CNTK**, **Torch**,  
**Keras**, **Theano**, **Caffe**, ...
- Tensorflow backend
- Keras R interface



# Unpacking the notebook



128 pixels

128 pixels



3 channels

# Unpacking the notebook

```
from keras.models import Sequential  
from keras.layers import Flatten, Dense  
  
model = Sequential()  
model.add(Flatten(input_shape=X.shape[1:]))  
model.add(Dense(2, activation='softmax'))  
model.summary()
```

**128 \* 128 \* 3 = 49.152 pixels / image**

Layer (+type)	Output Shape	Param #
<b>Probability distribution over two classes, e.g. [0.3, 0.7]</b>		
flatten_1 (Flatten)	(None, 49152)	0

dense_1 (Dense)	(None, 2)	98306
-----------------	-----------	-------

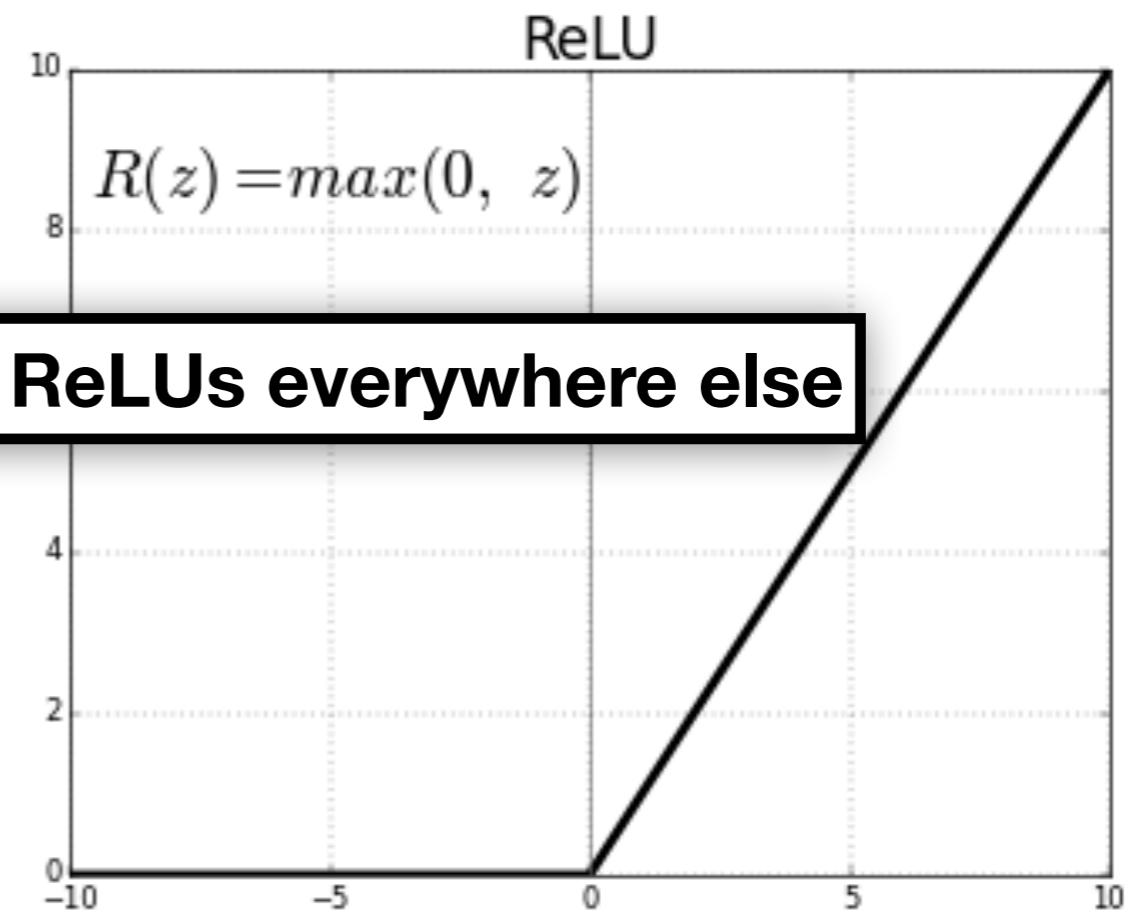
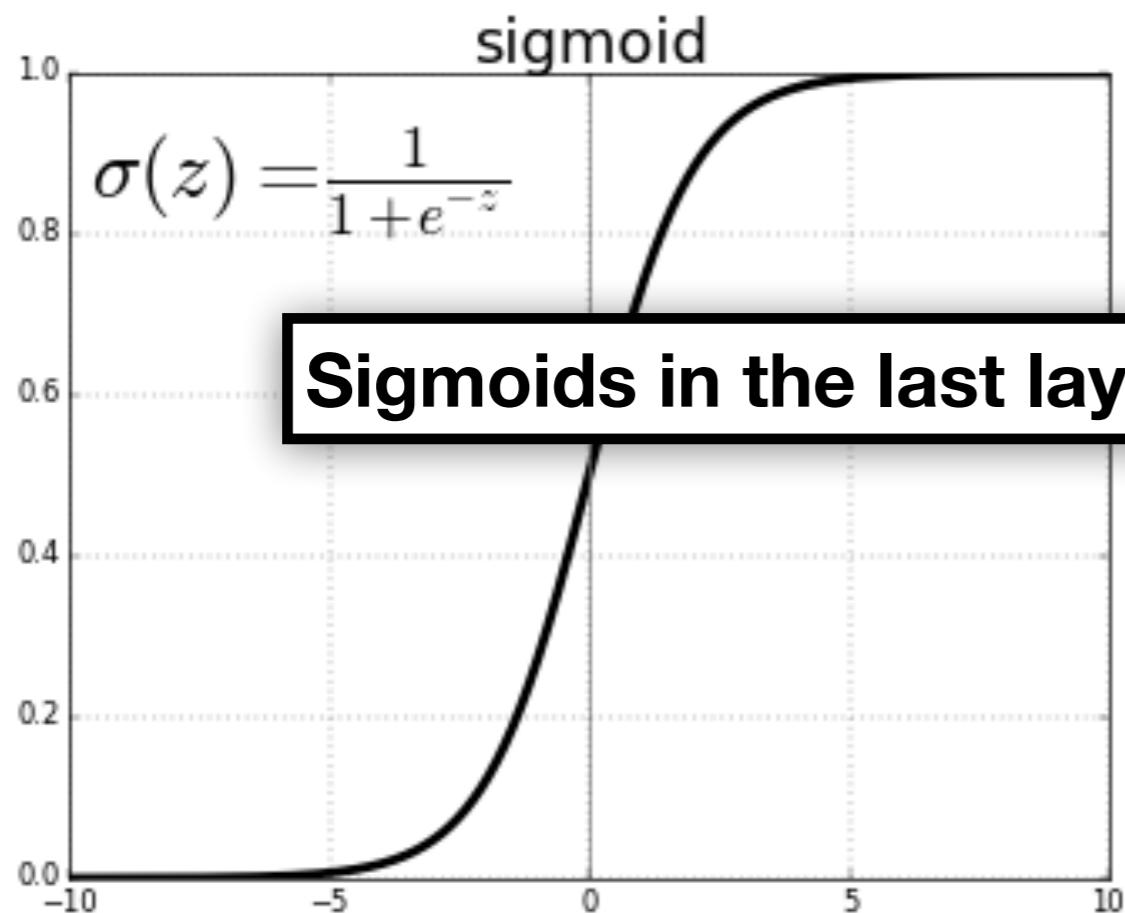
=====

Total params: 98,306

Trainable params: 98,306

Non-trainable params: 0

# Activation functions



Sigmoids in the last layer, ReLUs everywhere else

Source: [Activation Functions in Neural Networks](#)

# Unpacking the notebook

```
from keras.optimizers import Adam

model.compile(
    Adam(lr=0.0001),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
history = model.fit(X, Y, epochs=25, batch_size=32)
```

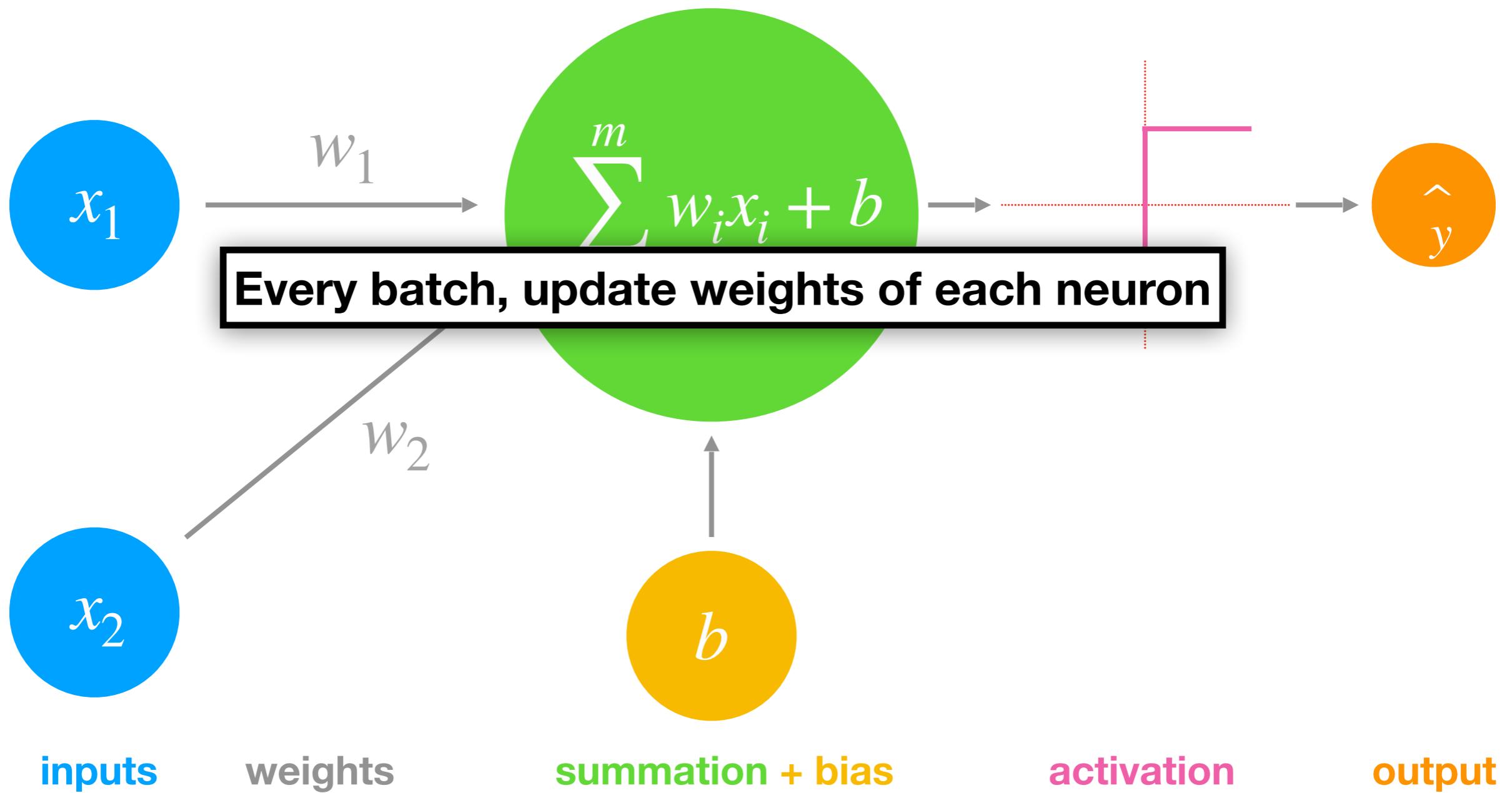
## Hyperparameters

```
Epoch 1/25
3500/3500 [=====] - 1s 306us/step - loss: 0.6972
- acc: 0.6417
```

# Hyperparameters

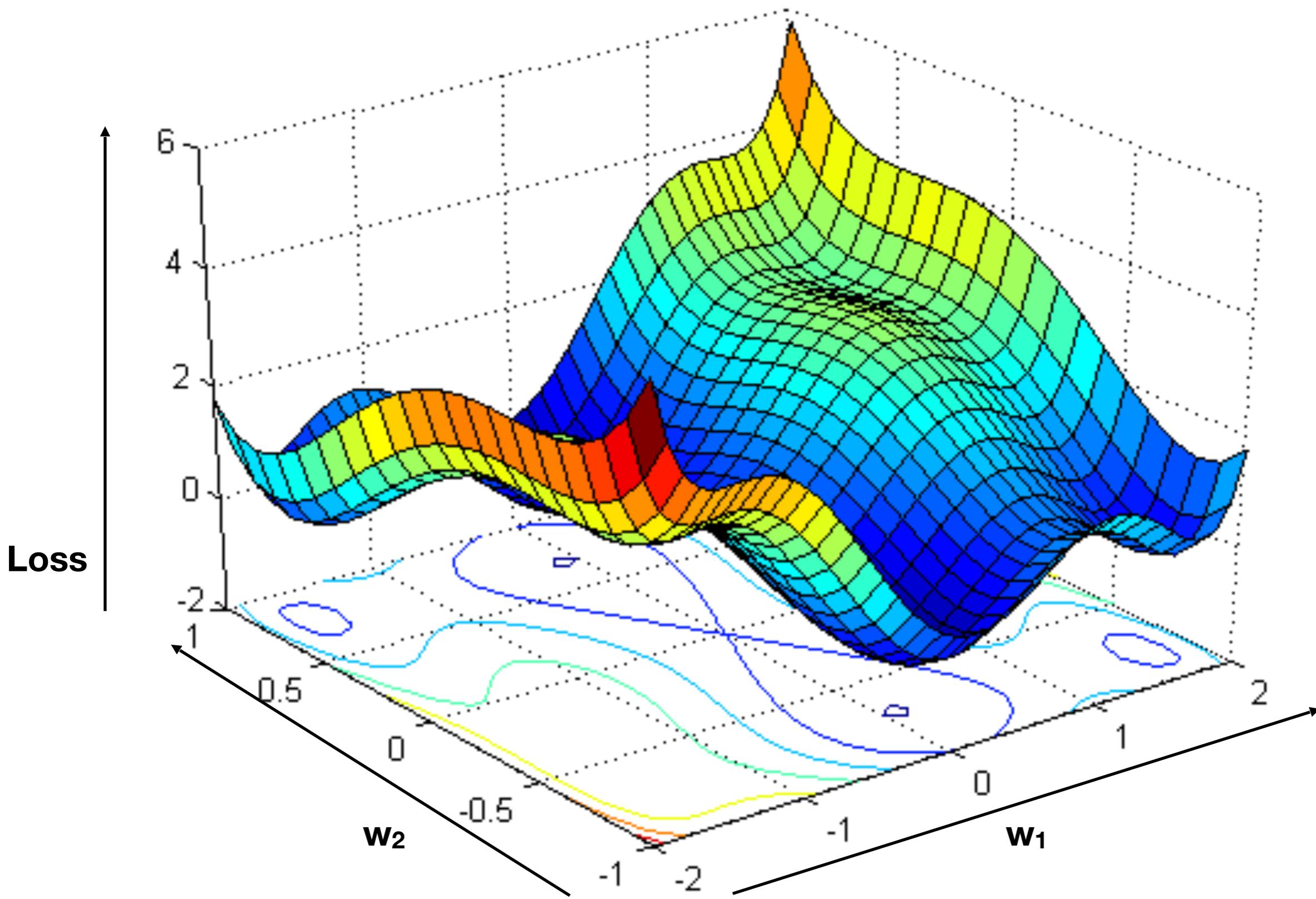
- ‘Ordinary’ parameters: weight and bias parameters
- **Hyperparameters** for network structure and training:
  - Number of layers
  - Number of hidden units
  - Learning rate
  - Batch size

# A single neuron



# Optimisation

- We are optimising a high-dimensional problem
- Network weights are parameters
- Usually millions/dozens of millions of parameters
- **Loss function** quantifies network performance



Source: [Introduction to loss functions](#)

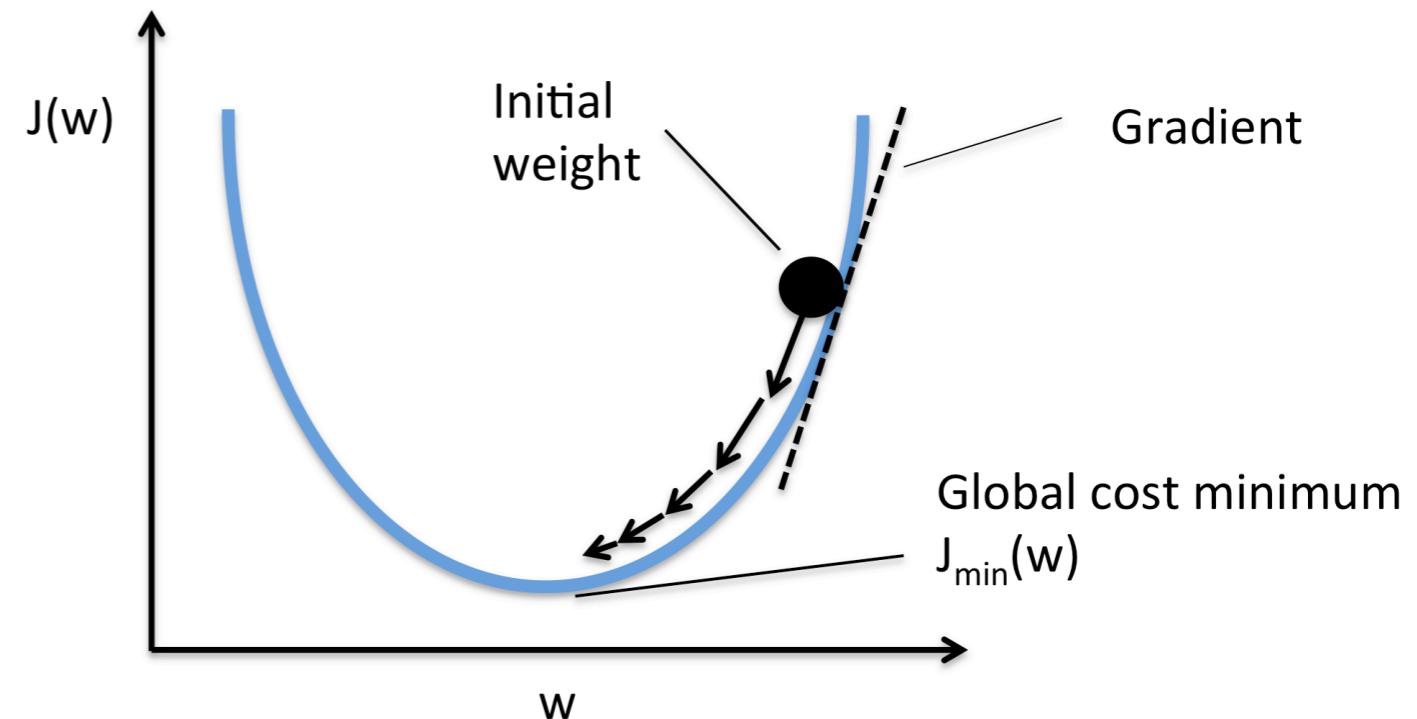
# Updating weights

(batch) Gradient descent

- We apply this procedure a few times, in each iteration we update the weights s.t.

$$w_i := w_i - \mu \frac{\partial J(w)}{\partial w_i}$$

where  $\mu$  is a value in  $(0;1]$  called the **learning rate**

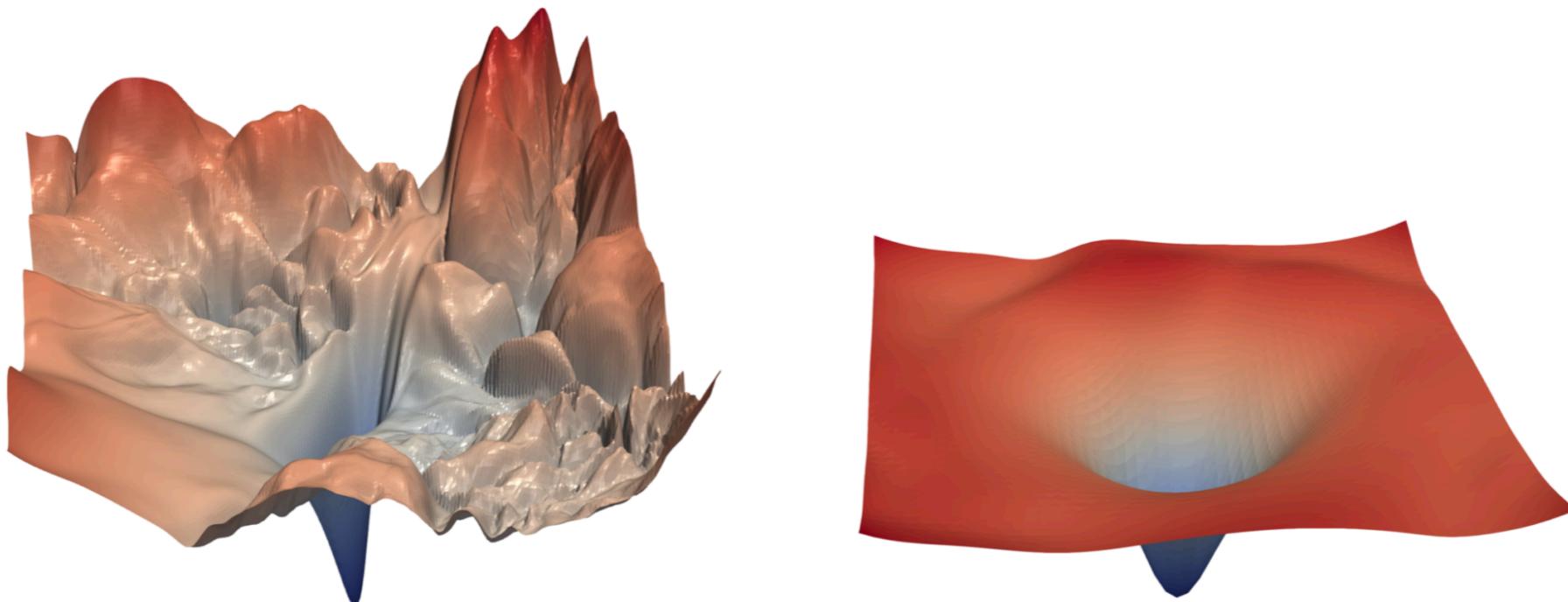


- This algorithm is called **(batch) gradient descent (GD)**.
- The loss (and therefore the gradient) is computed over **all elements** in the dataset.

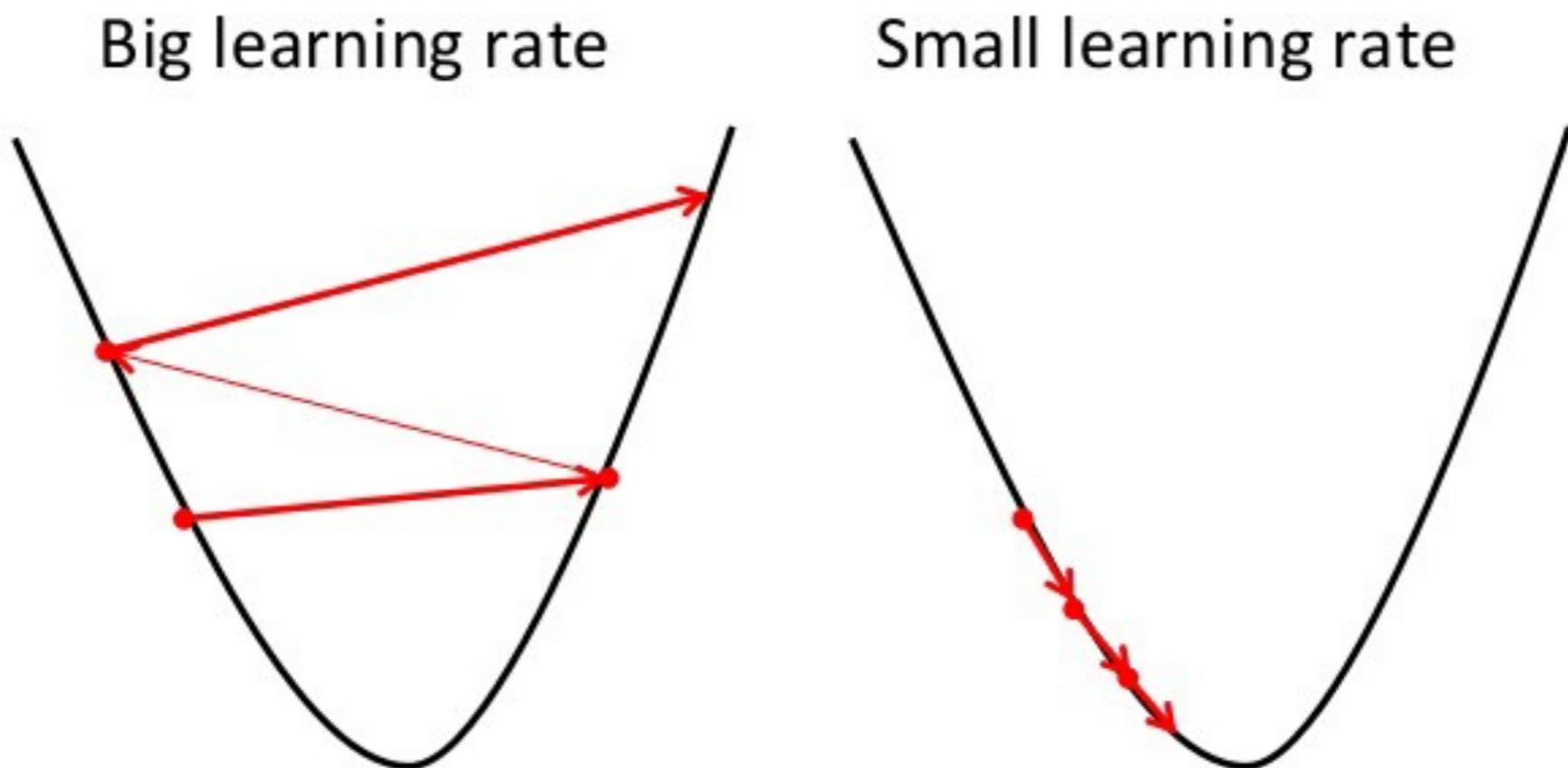
Source: [Gradient optimization](#)

# Updating weights

- Batch gradient descent is time-consuming
- Batch gradient descent expects convex loss surfaces
- Use **mini-batch gradient descent** instead
- Batch size = 1 is **stochastic gradient descent**

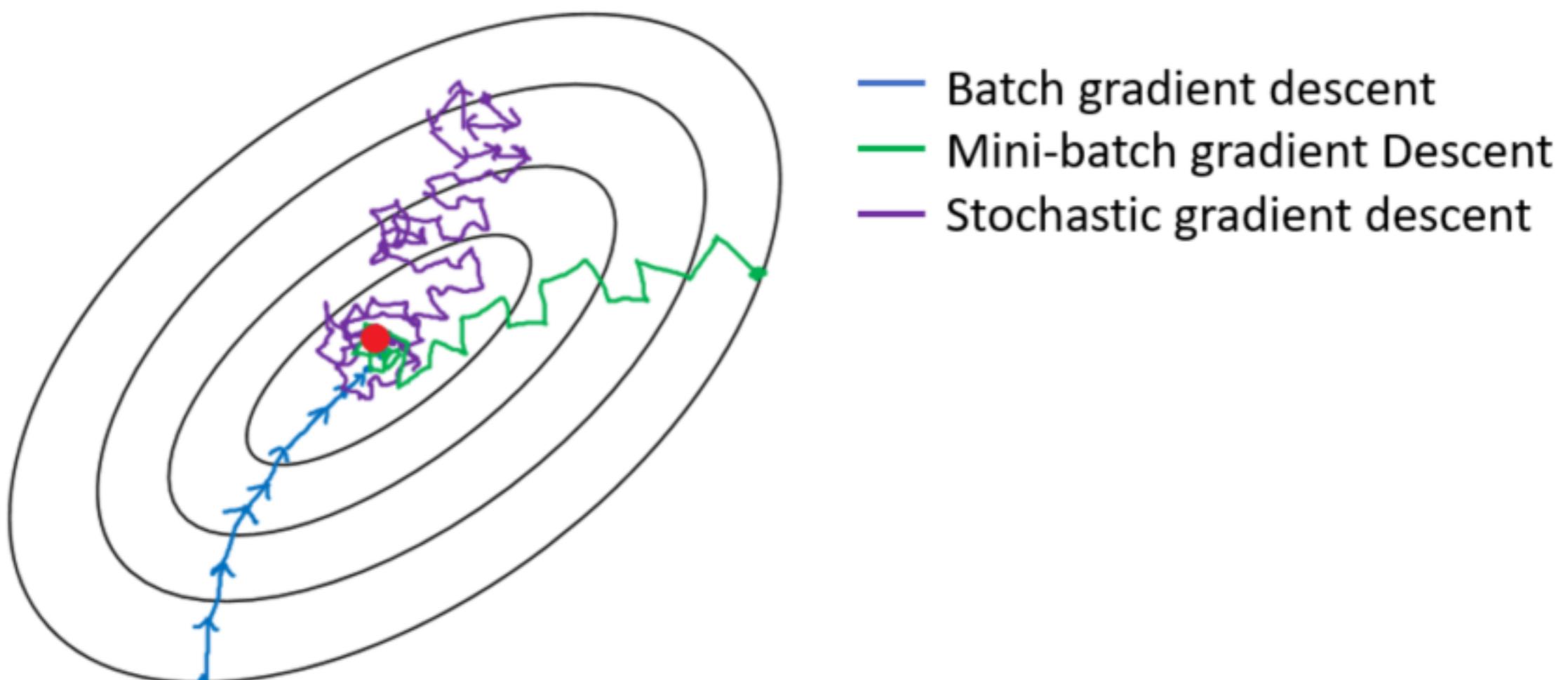


# Learning rate



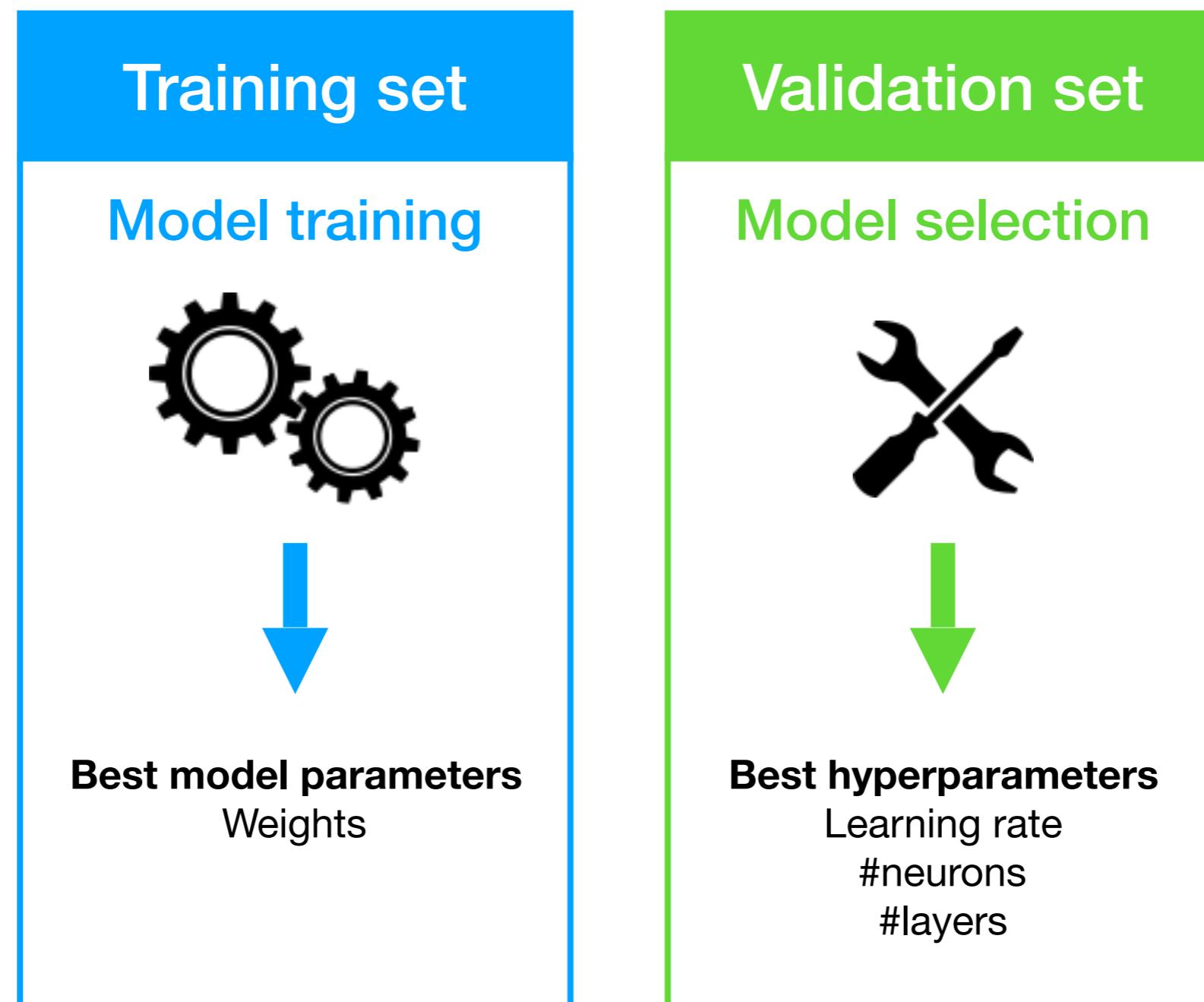
Source: [What are Hyperparameters?](#)

# Batch size

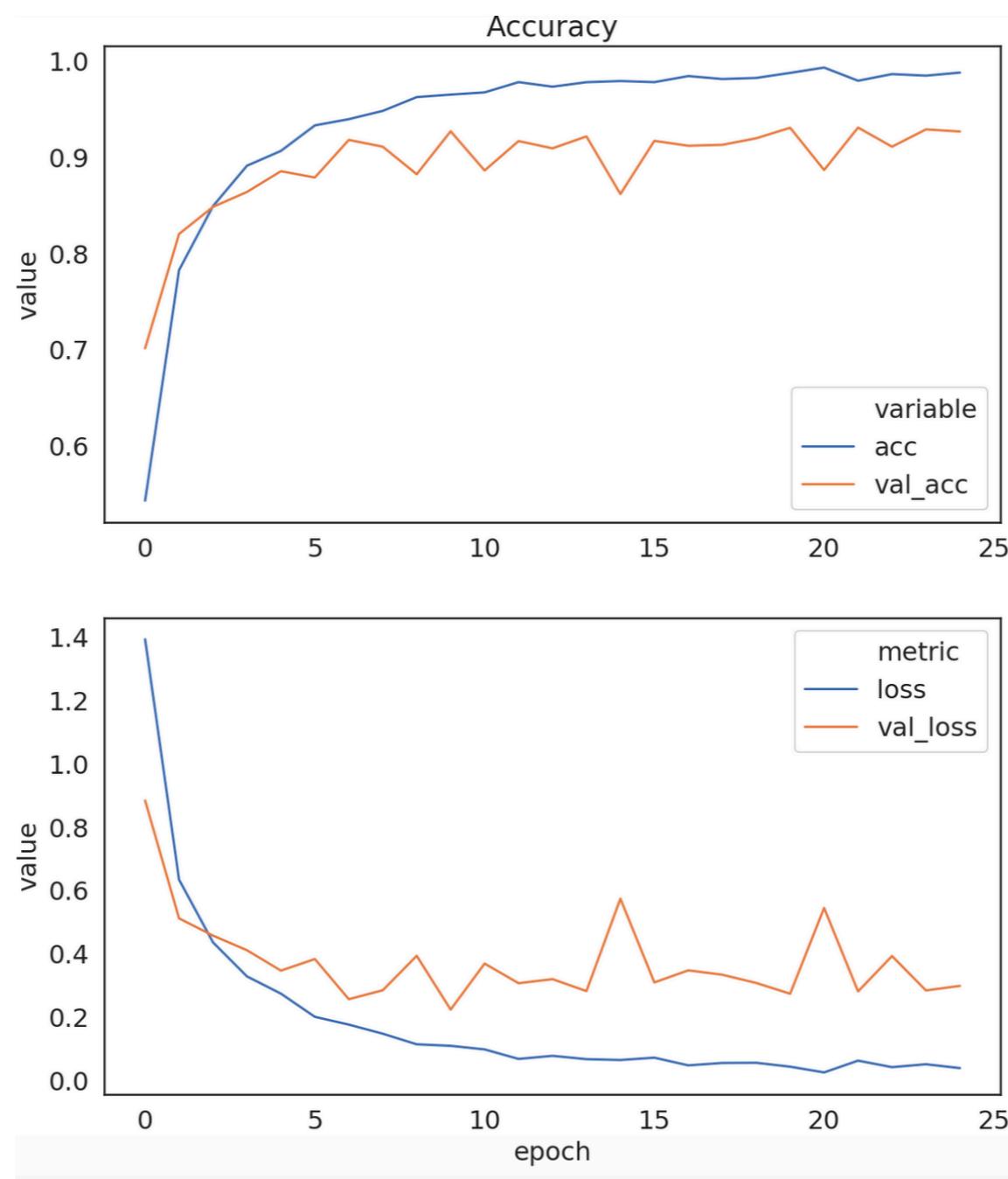


Source: Andrew Ng, deep learning course, Coursera

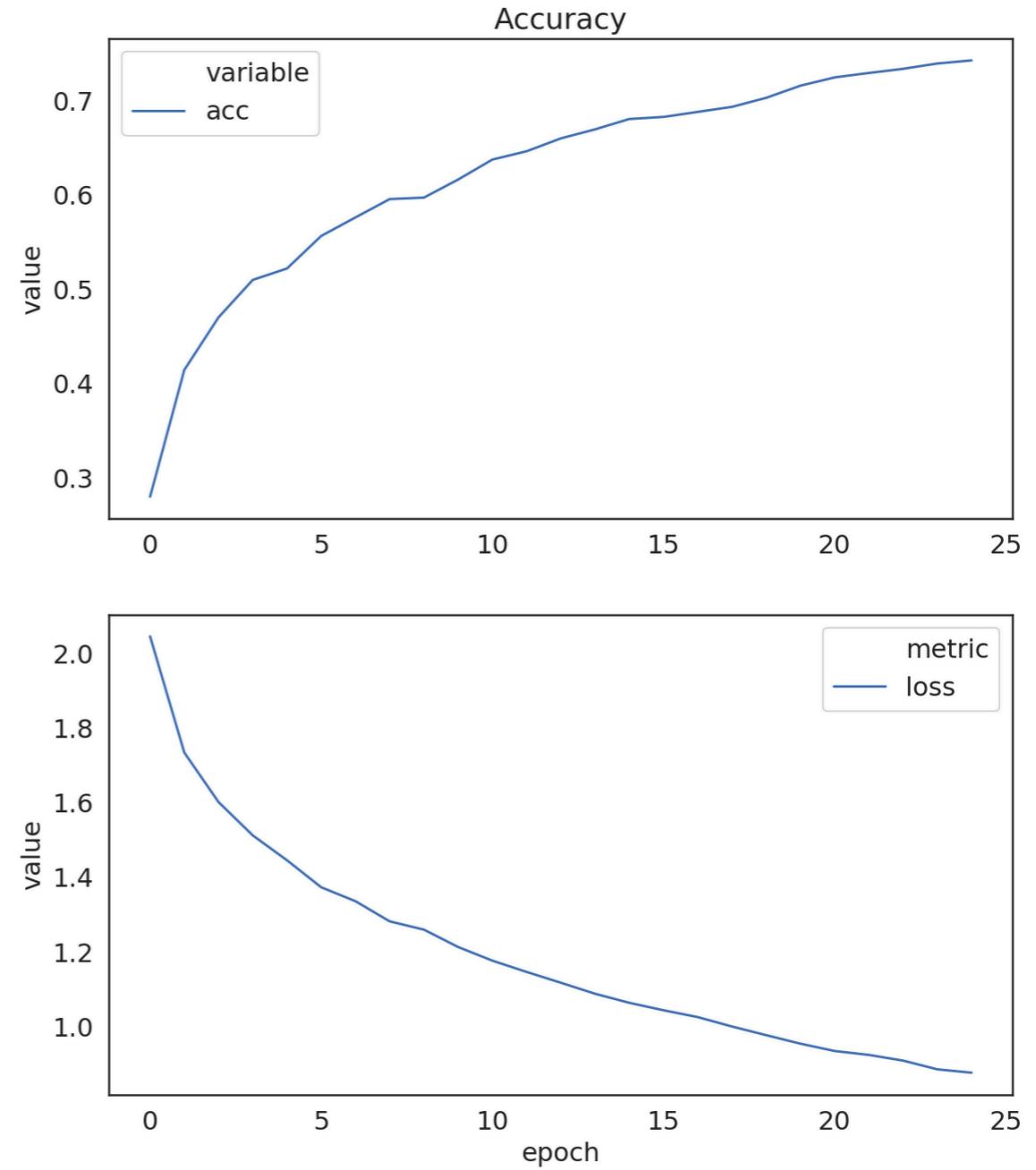
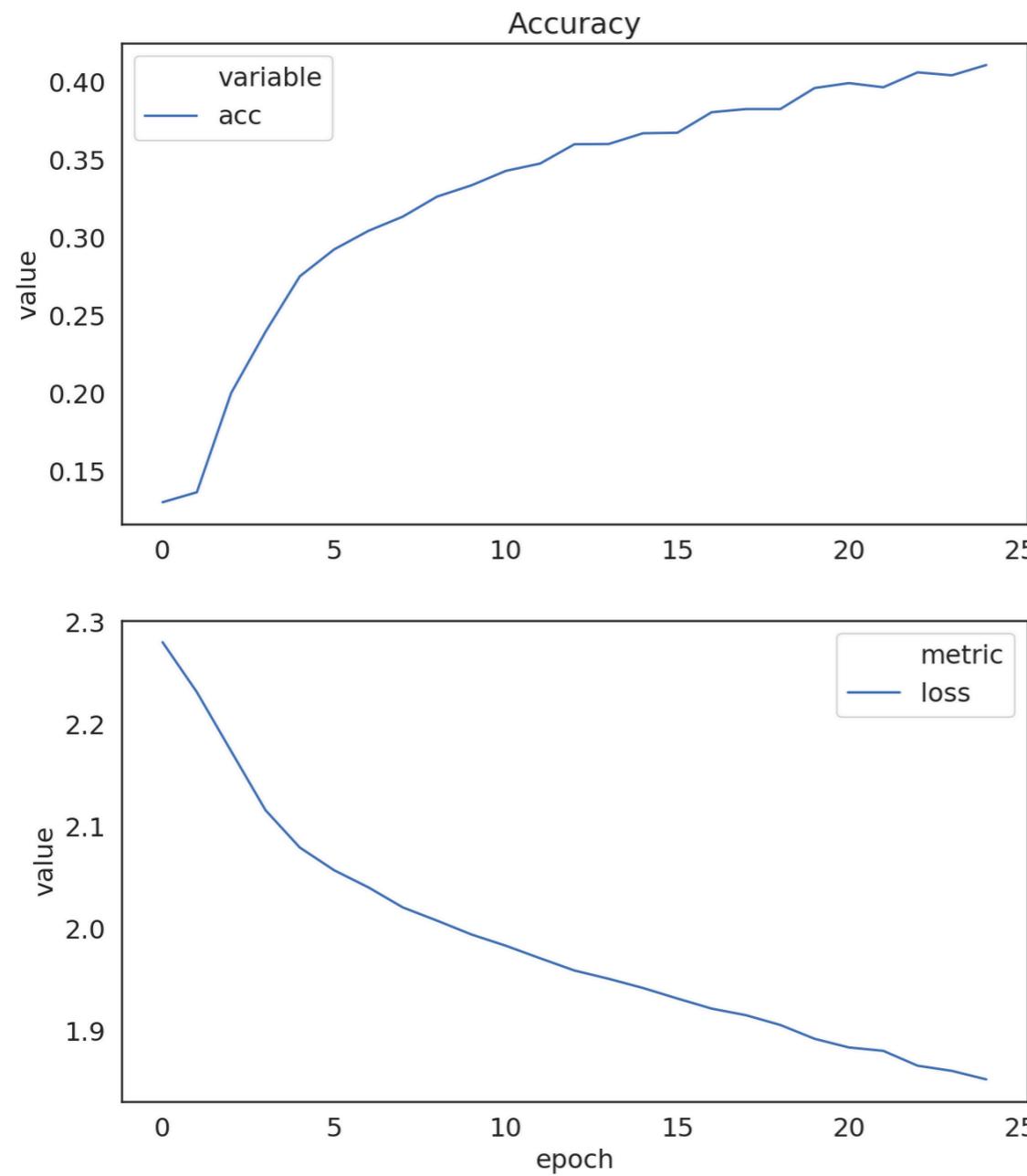
# Training & validation



# Training & validation



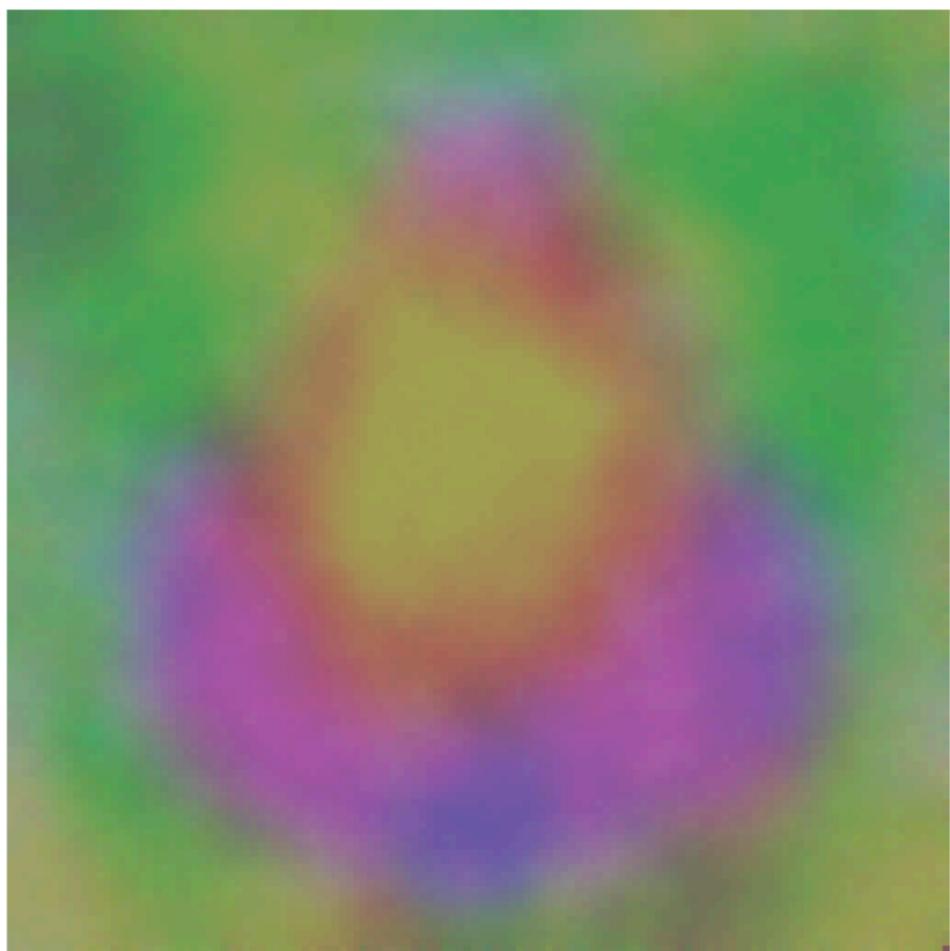
# Model capacity



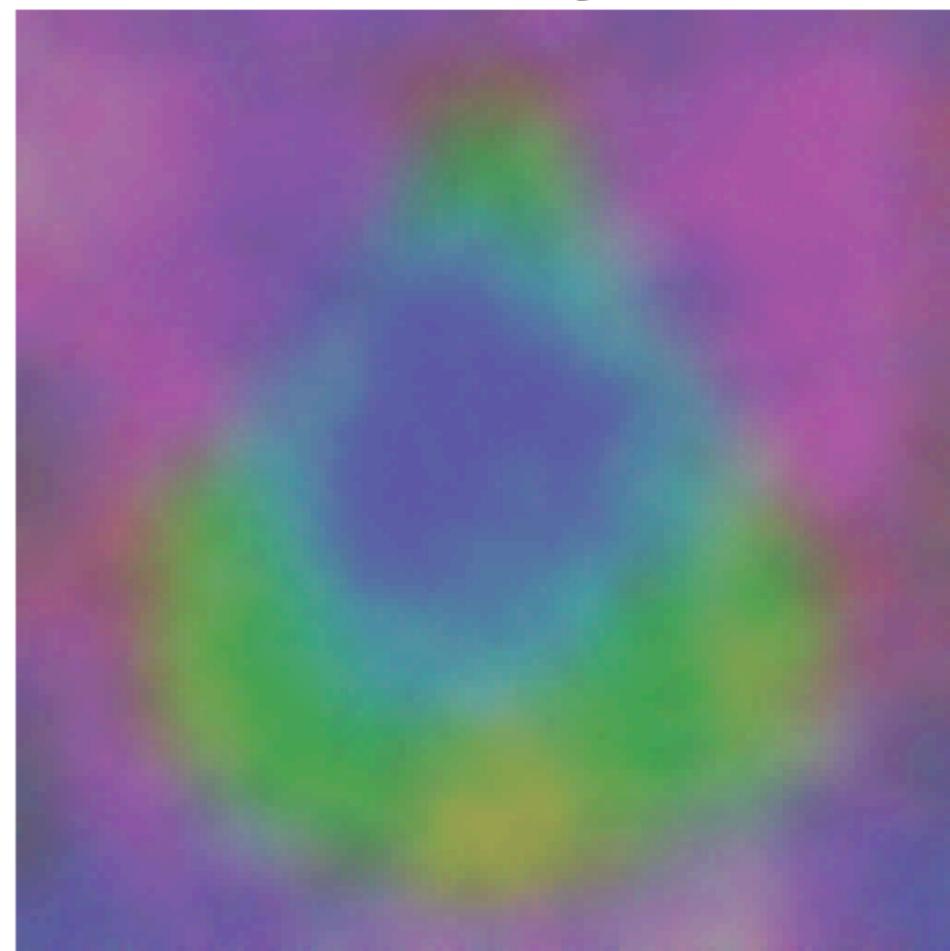
# Activations

Each image is a feature detector

diseased

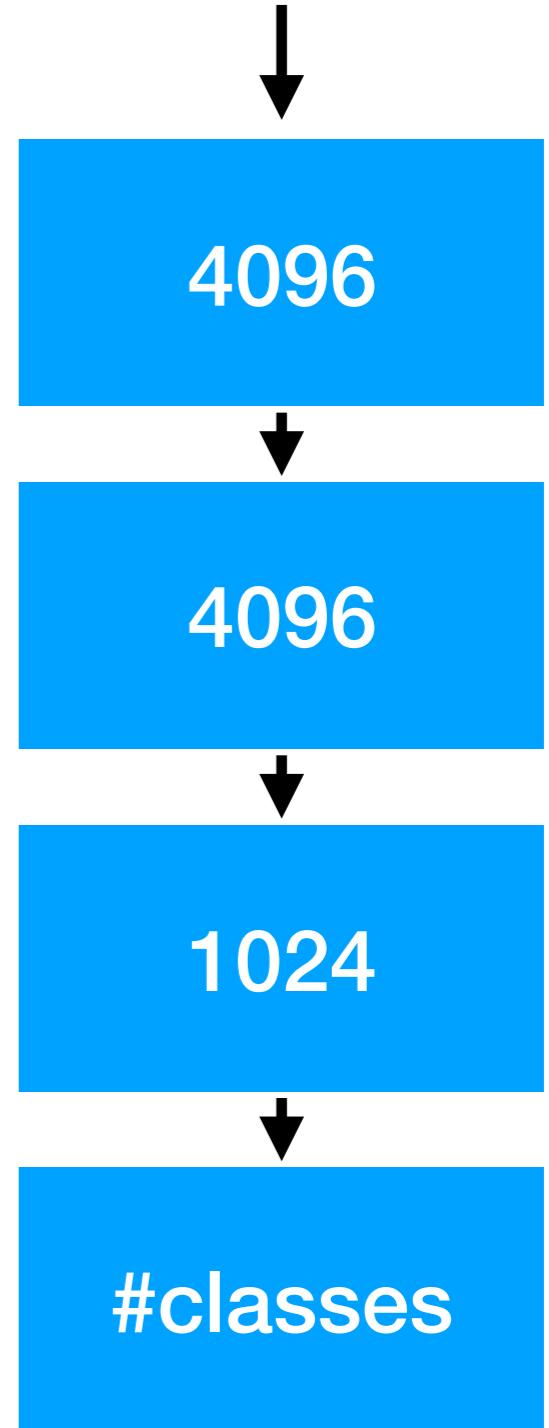


healthy

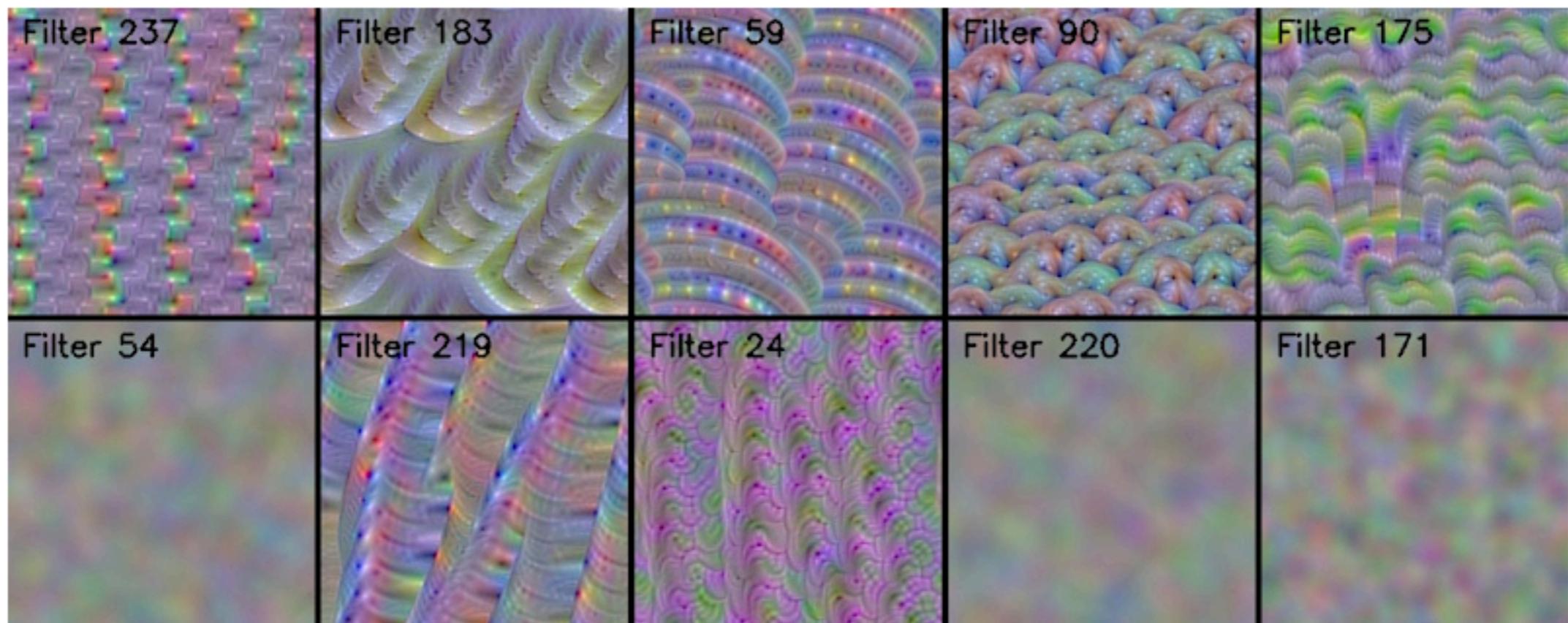


# Rules of thumb

- Three hidden layers should be sufficient
- Number of units as multiple of two
- Think of the network as a funnel
- Softmax in last layer
- ReLU everywhere else
- Batch size of 32
- Learning rate dependent on optimiser (0.001)



# Activations



# Hands-on



Go to <https://elixir-dl.sda-projects.nl>

Notebook: 02-hyperparameter-tuning.ipynb

**Lunch at 12:00**

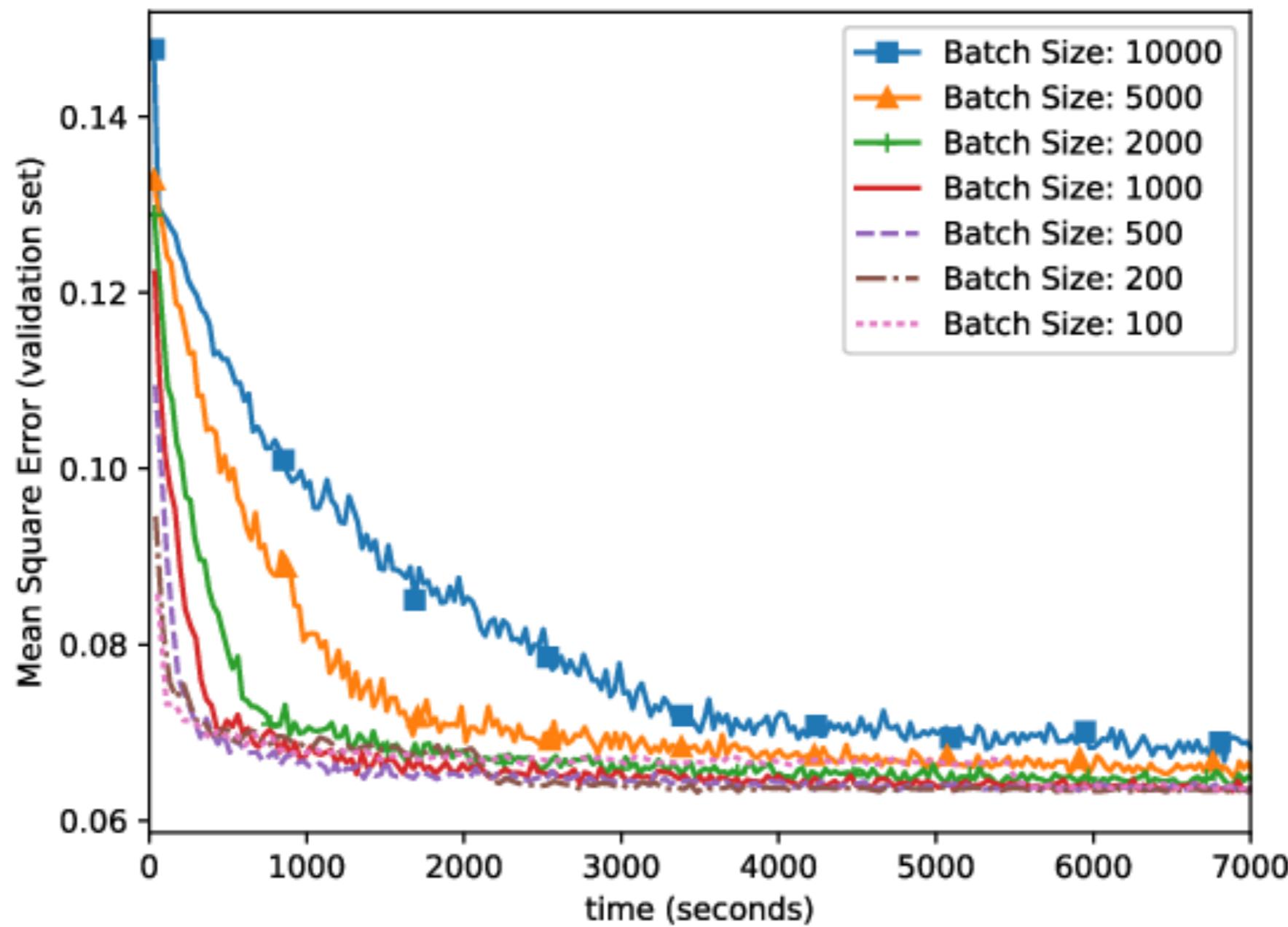
<https://github.com/sara-nl/elixir-dl>

# **LUNCH**

**Return at 13:00**

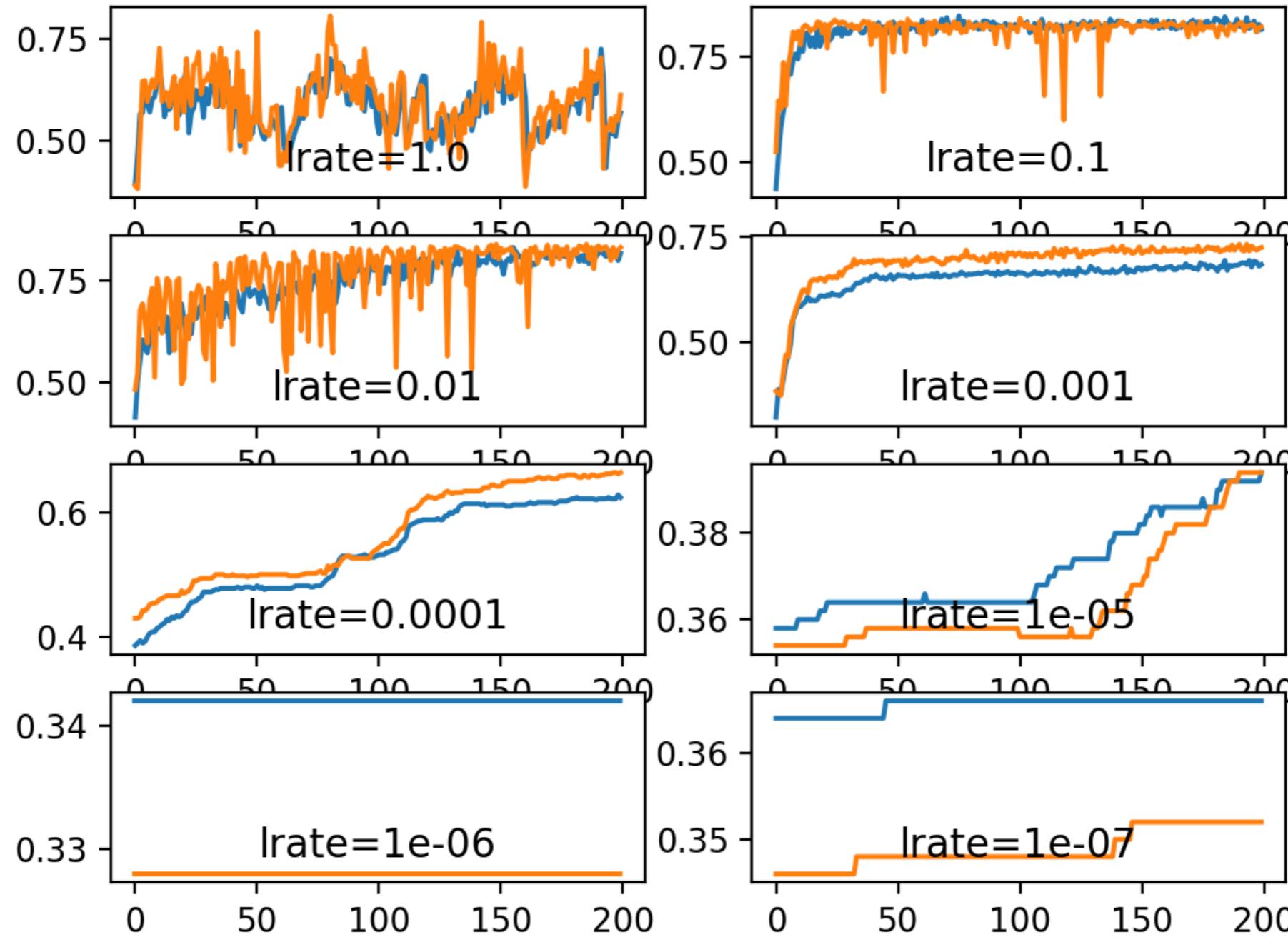
**Please sign the attendance sheet!**

# Recap

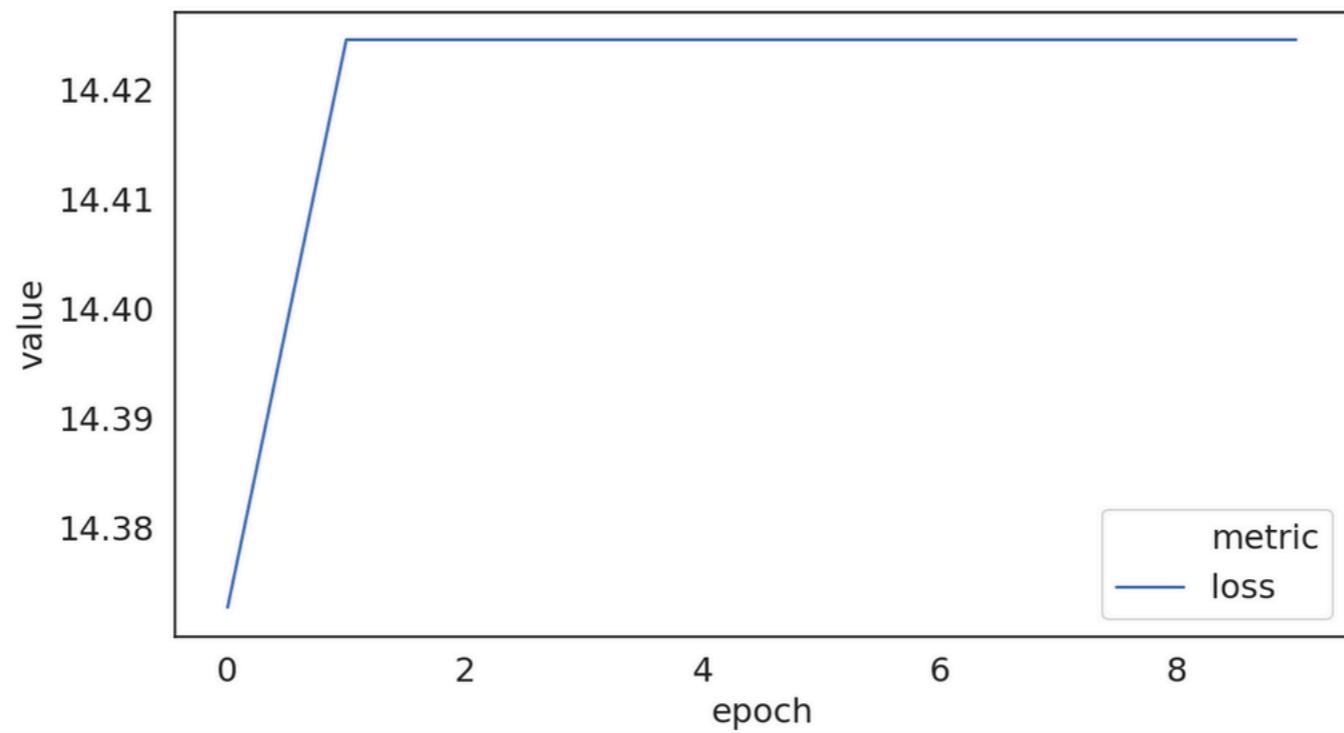
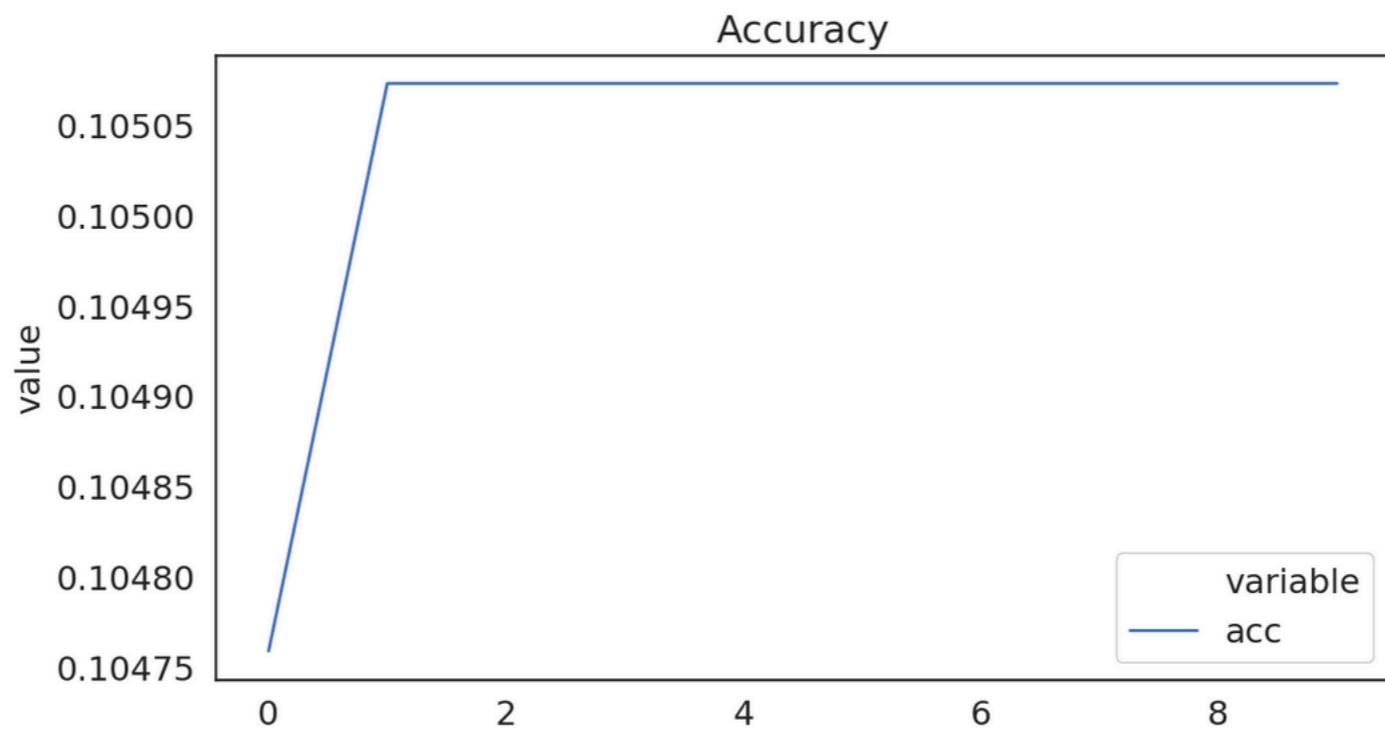


(a) Batch size selection

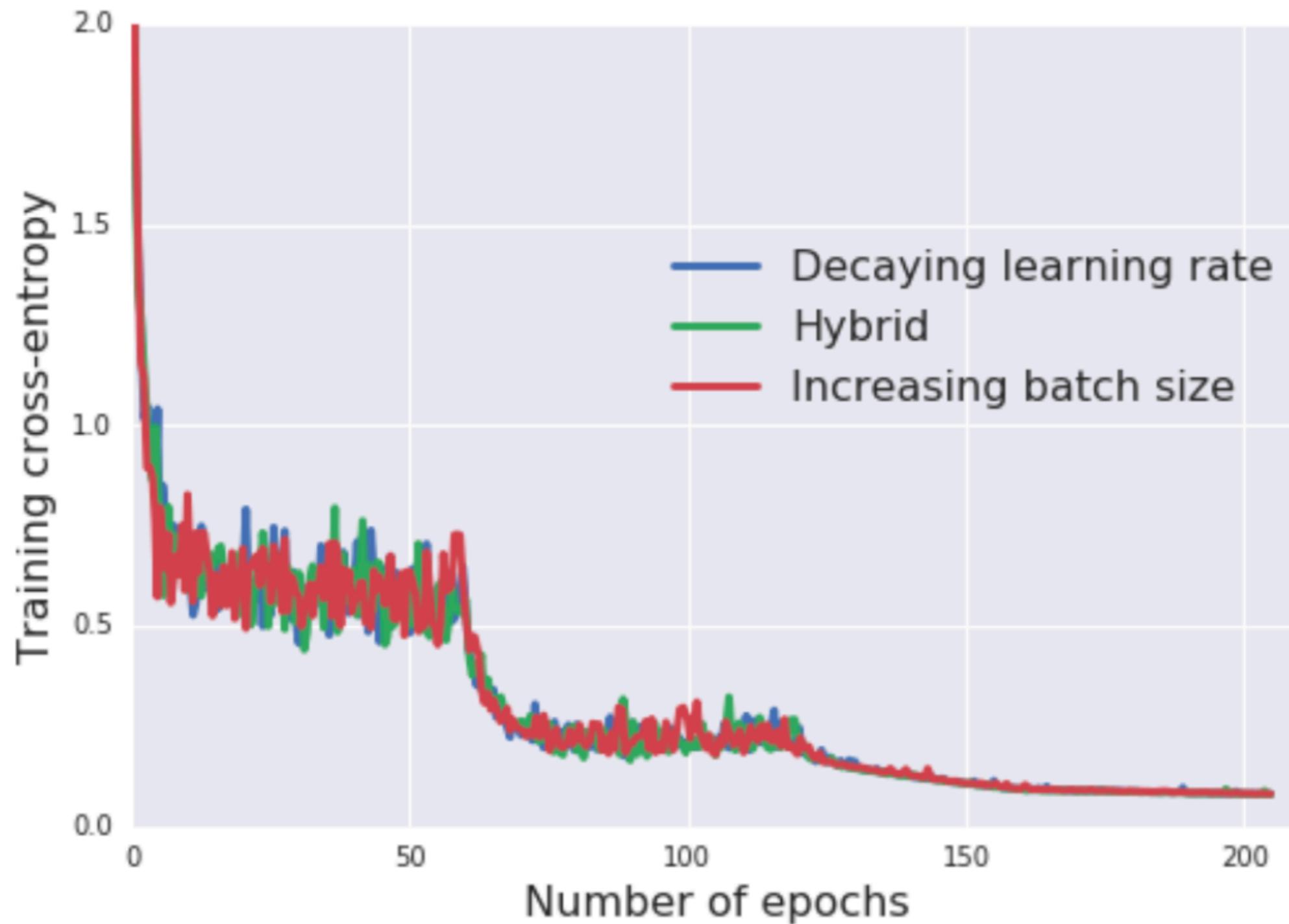
# Learning rates



# Learning rates



# Convergence is not linear

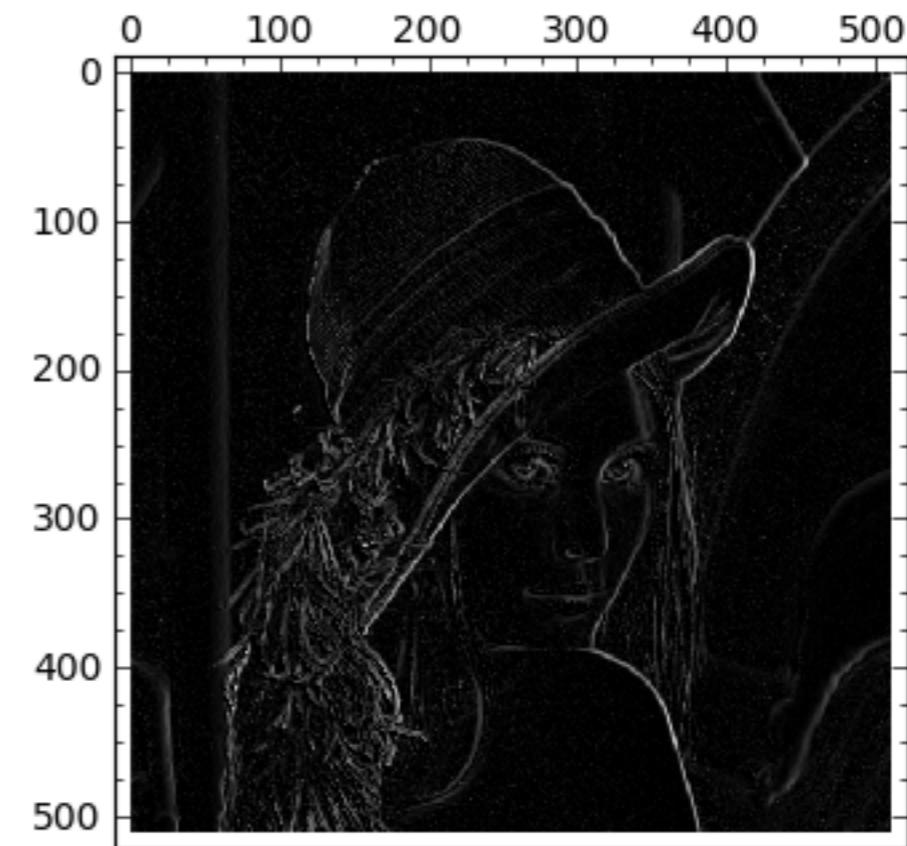
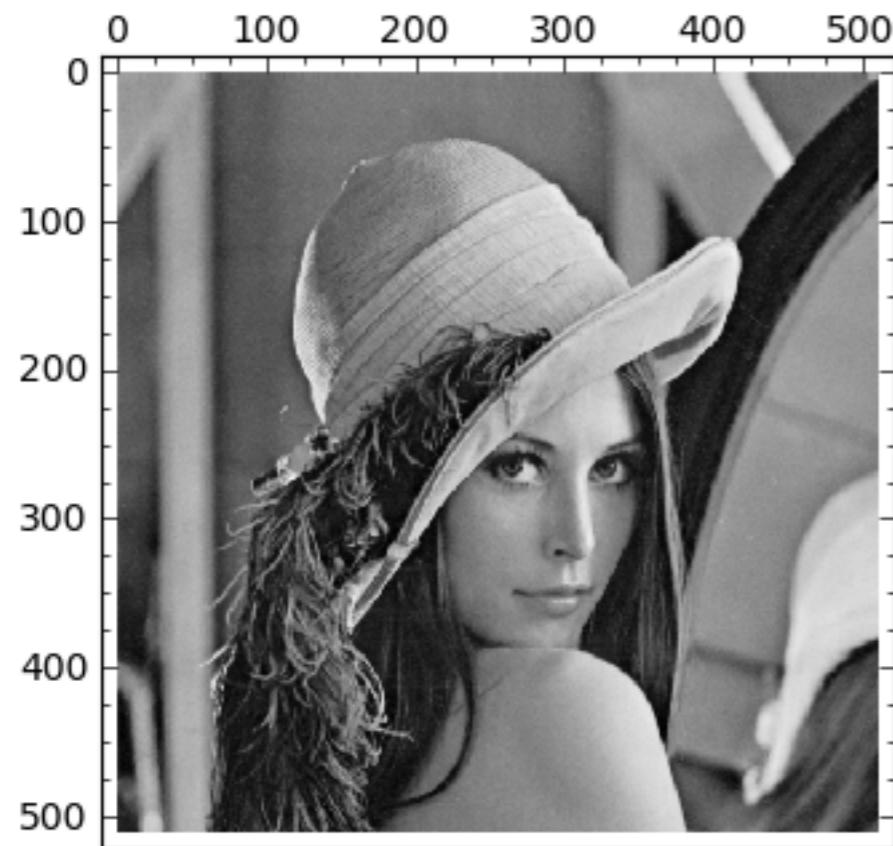


# Rules of thumb

- Change one hyperparameter at a time
- Learning rates in orders of magnitude (0.001, 0.0001, etc.)
- Initial phase of exploring hyperparameters
- Hyperparameter optimisation is active area of research
- Convergence is not linear. Be patient!
- **Be patient!**

# Convolutions

- Applying a filter to an image.



# Convolutions

- We can think of convolutions as small filters on the image.
- They process parts of the image and "fire" when they detect the filter pattern.

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

# Convolutions

## Technicalities

The diagram illustrates a convolution operation between two matrices. On the left, a 6x6 input matrix is shown with its first three columns highlighted in orange. An asterisk (\*) indicates the multiplication operator, followed by a 3x3 kernel matrix enclosed in an orange border. A curved orange arrow points from the highlighted input columns to the top-left element of the kernel matrix, with the text "we multiply these two matrice element - wise and sum them" explaining the operation. An equals sign (=) follows the kernel matrix, leading to a 4x4 output matrix where the first element is highlighted in orange.

6 x 6

3 x 3

4 x 4

we multiply these two matrice element - wise  
and sum them

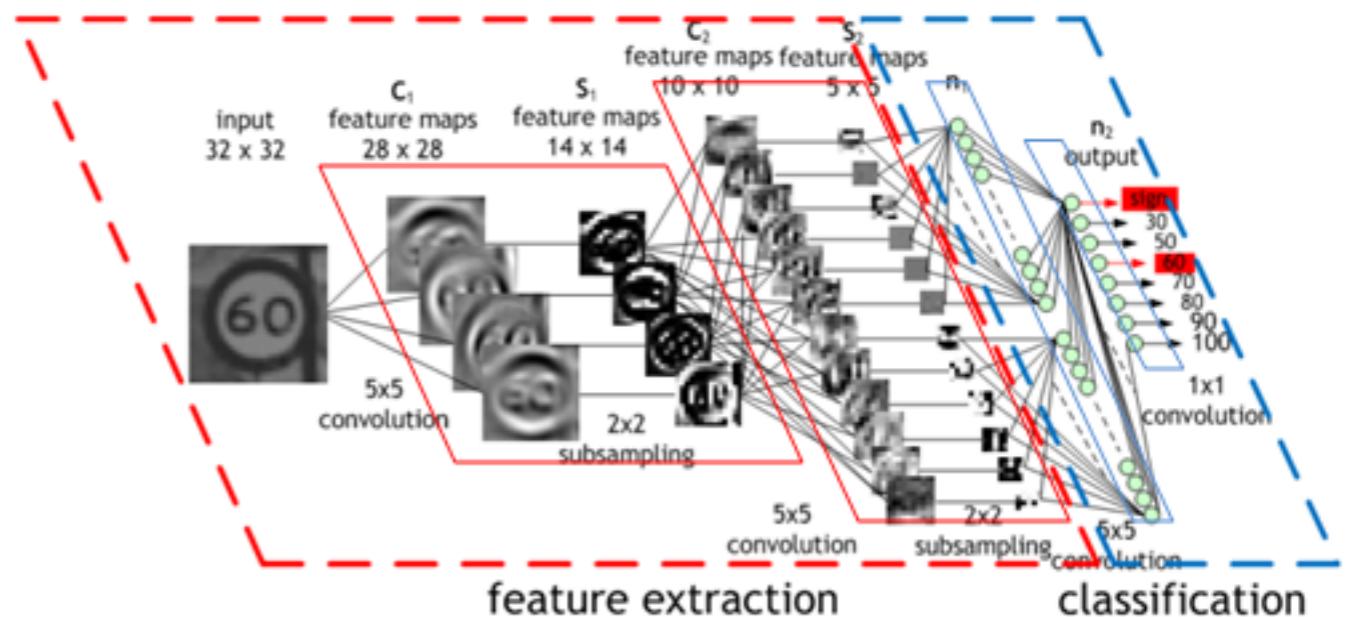
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

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

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

# Convolutions

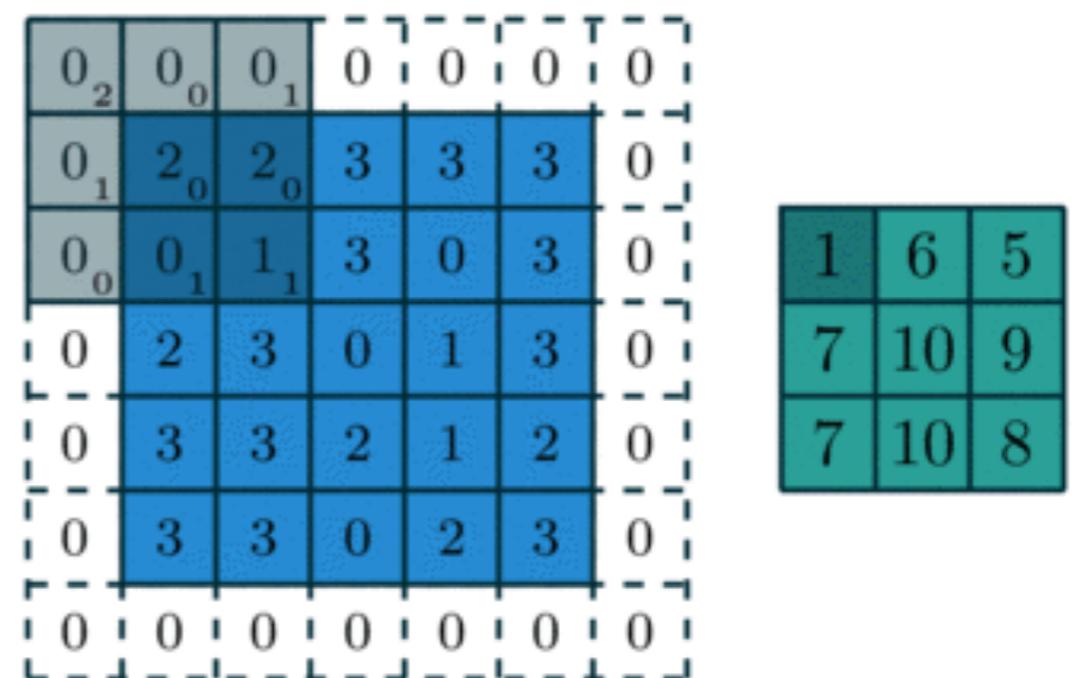
- Stack convolutions
- Create filters from other filters
- Detect more complex patterns



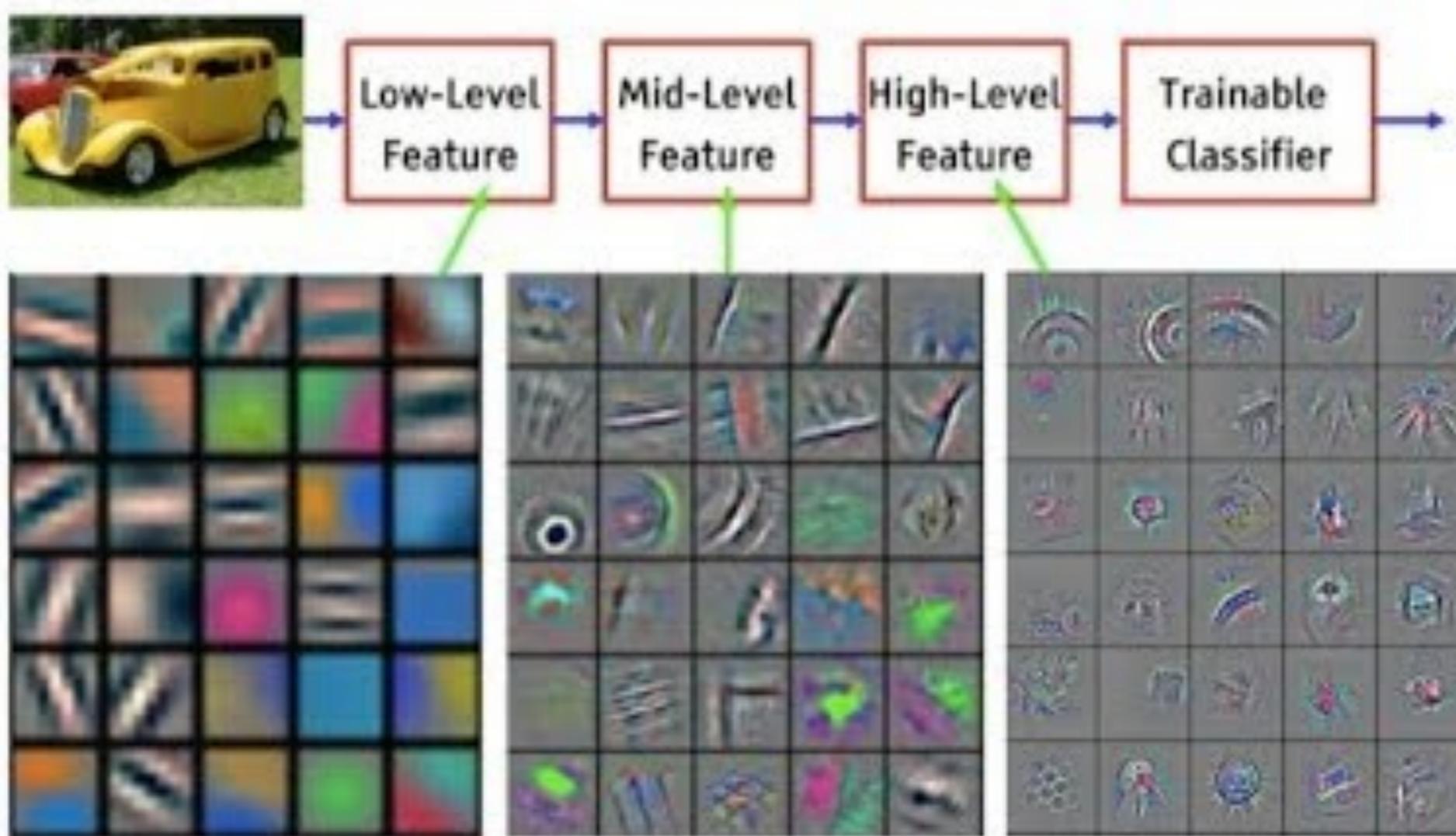
Source: <https://developer.nvidia.com/discover/convolutional-neural-network>

# Convolutions

- We apply a **kernel** with dimensions (3, 3).
- Using **stride** = 2.
- **Padding** = 1, to increase output size
- **Same-padding** = Output is same size as input
- Otherwise **valid-padding**.
- Apply activation function to output.
- We try to **learn the filters!** The filters are the parameters/weights.



# Feature hierarchy

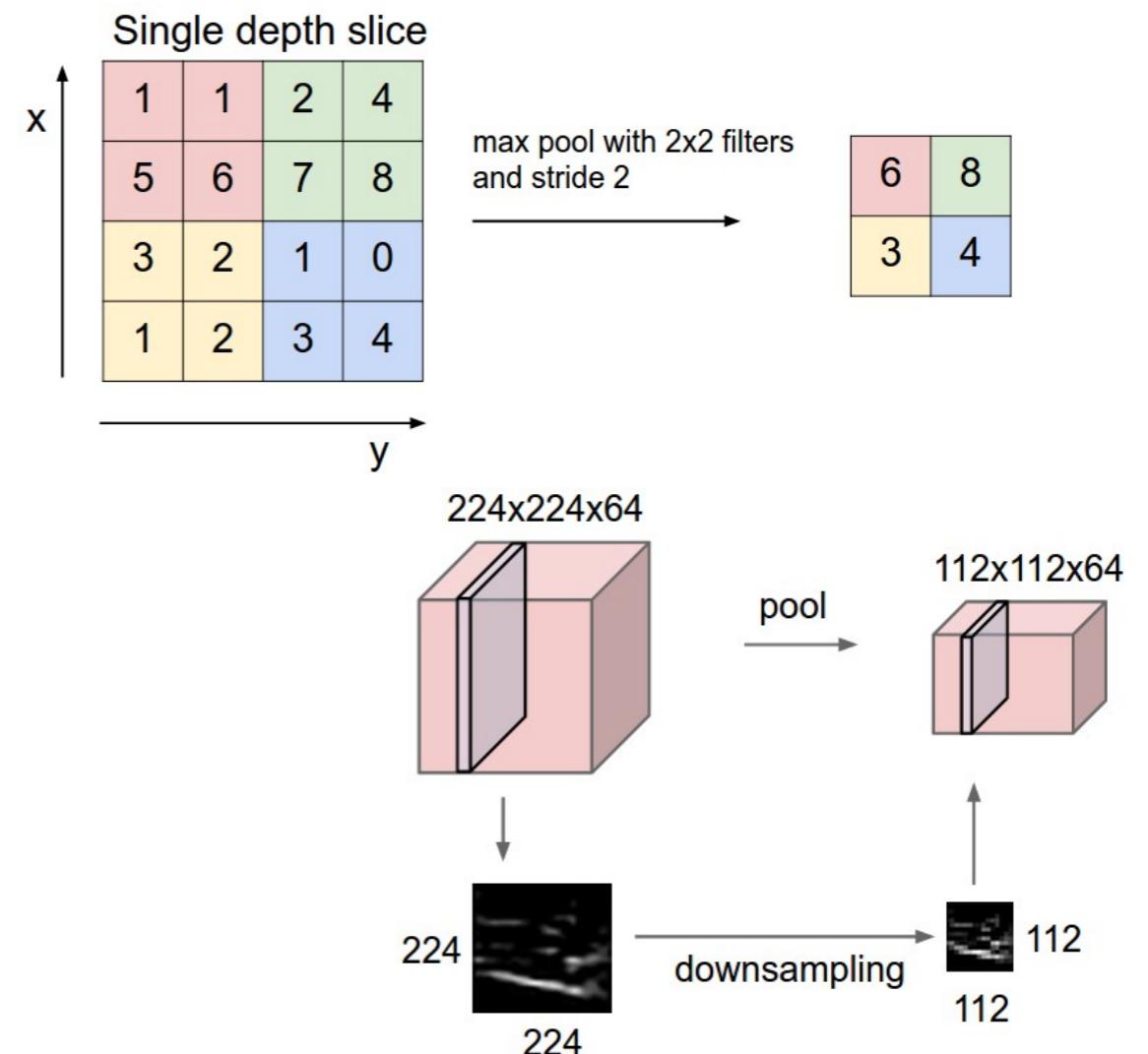


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# CNNs

## Pooling layers

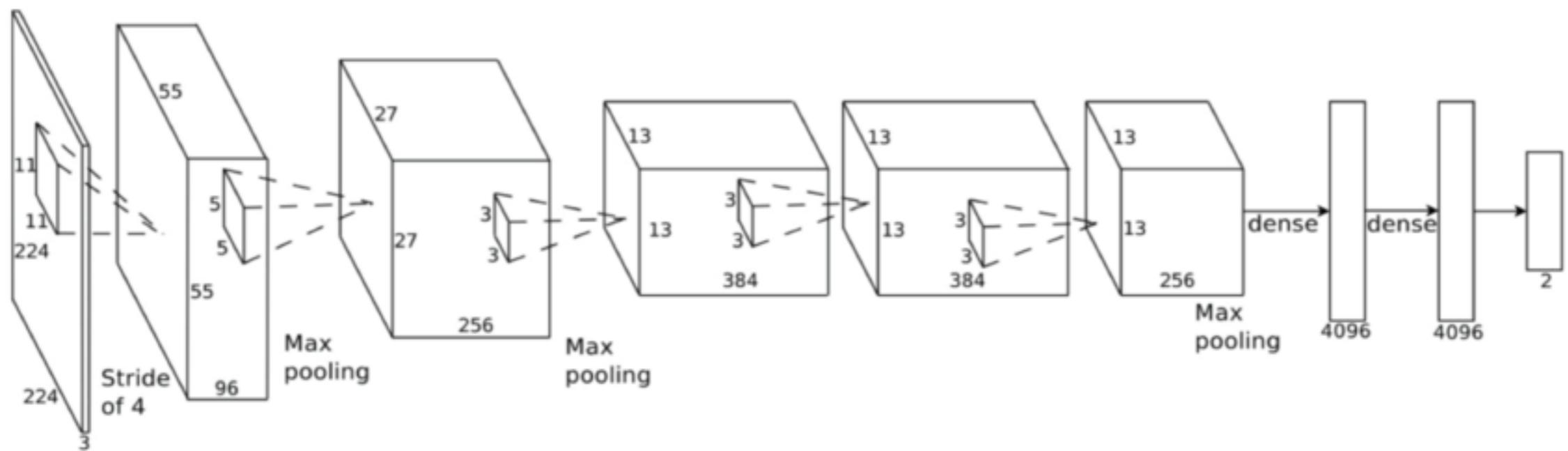
- Our filter maps are sometimes very large, so we make them **smaller using pooling**.
- **Max pooling**, take the max.
- Applied after convolutions



# CNNs

## Architectures - AlexNet

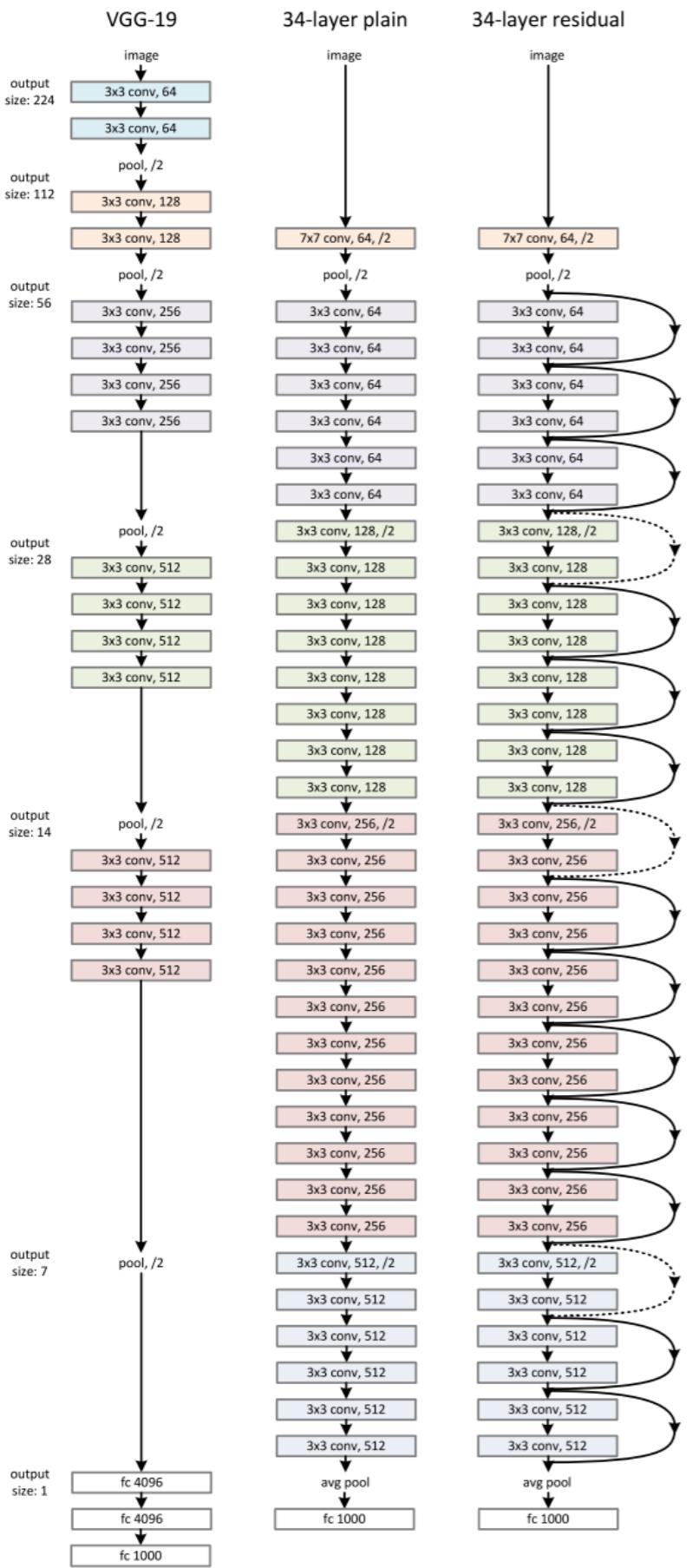
- By chaining different combinations of them we can create Convolutional Neural Networks (CNNs).



Source: <https://www.jeremyjordan.me/convnet-architectures/>

# CNNs

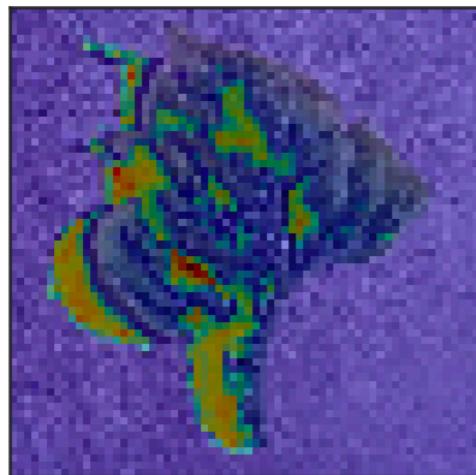
## Architectures - ResNet (2015)



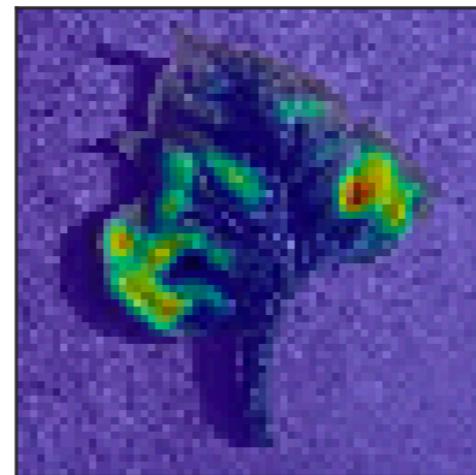
Source: <https://arxiv.org/abs/1512.03385>

# Class activation maps

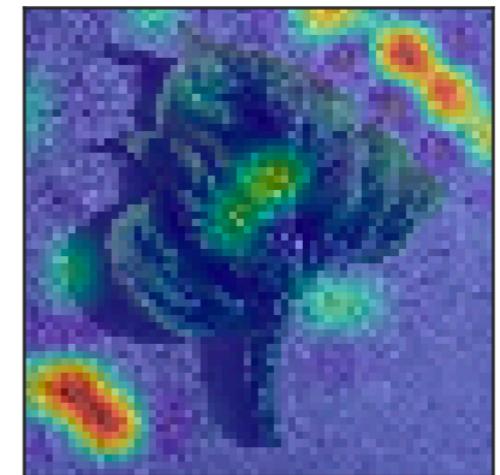
Conv. layer 1



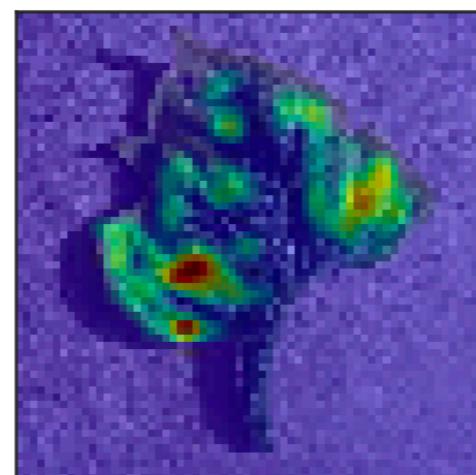
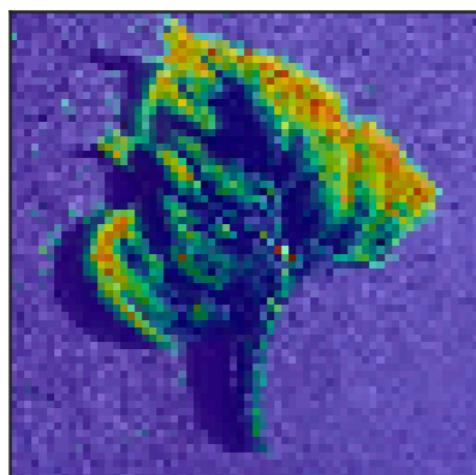
Conv. layer 2



Conv. layer 3

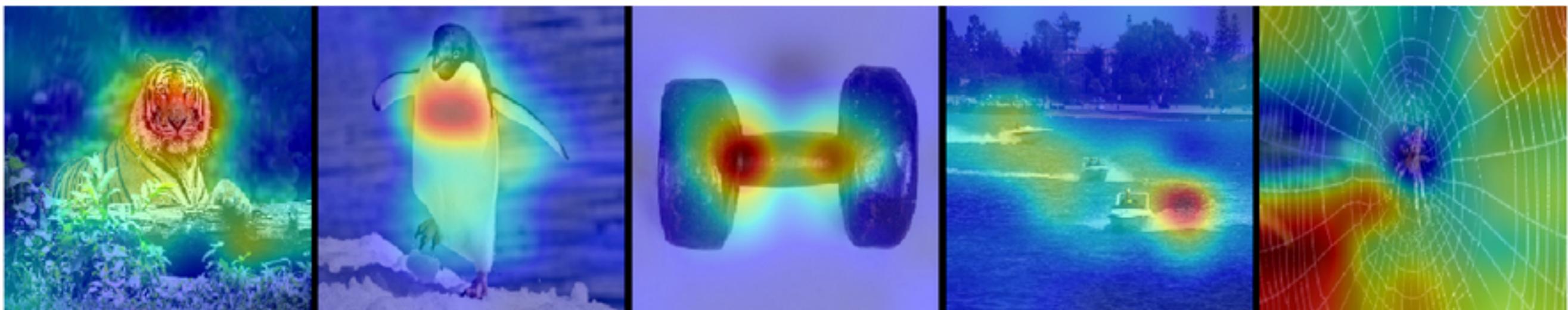


Healthy



Diseased

# Class activation maps



# Hands-on



**Kill your notebooks!**

Notebook: 03-your-first-cnn.ipynb

**Break at 14:30**

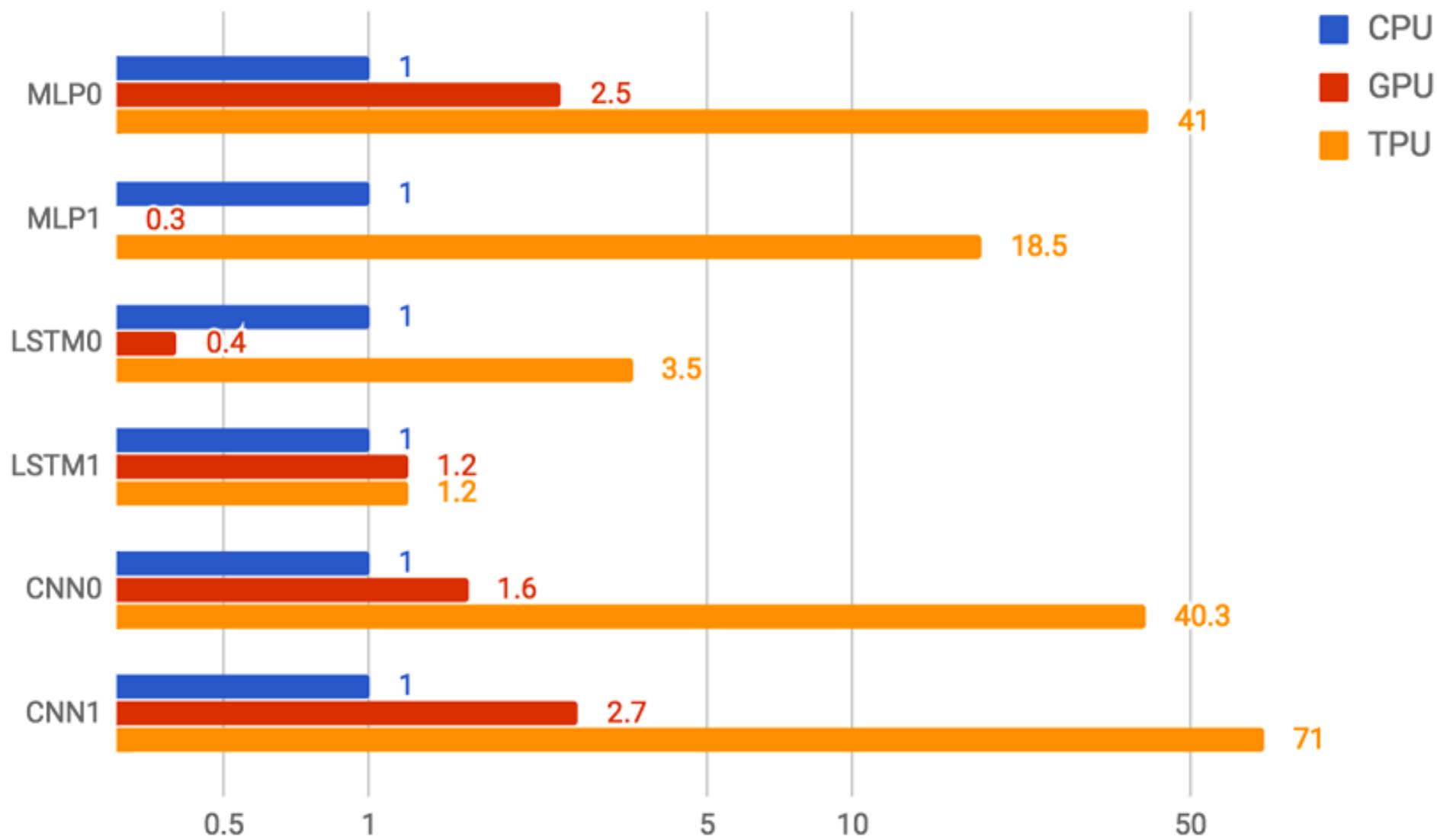
**Please fill out the survey: <https://bit.ly/2HaYGXX>**

# Recap

- Initialisation of weights is random
- Filters can be redundant or useless
- Class activation maps are not perfect
- Pixel values are normalised
- Fully-convolutional networks and dense layers

# CPUs, GPUs, TPUs

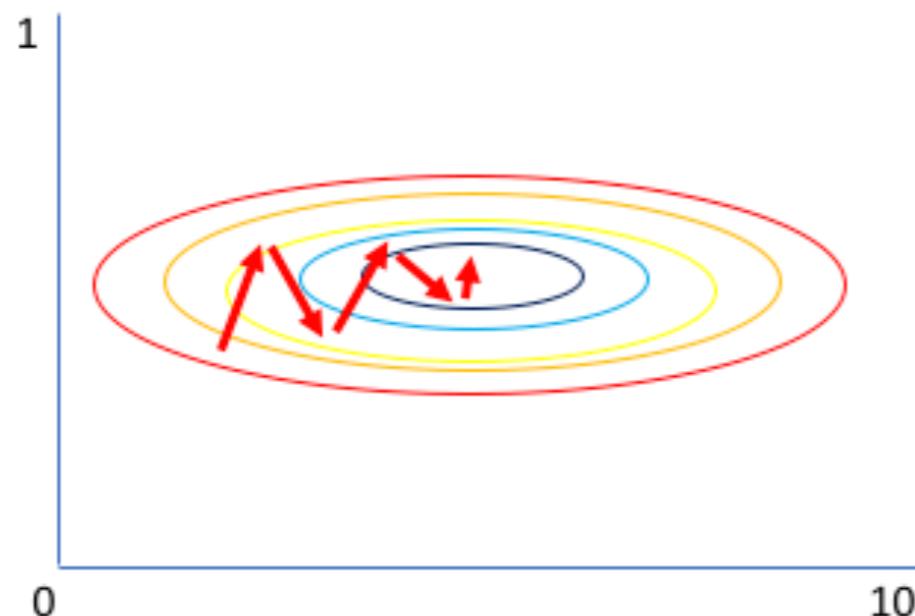
CPU vs GPU vs TPU performance comparison on six reference workloads log scale)



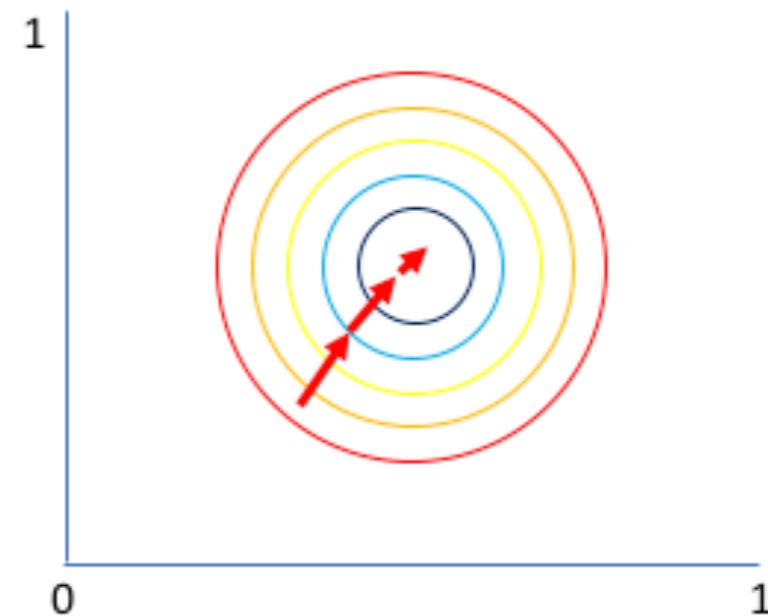
Source: <https://www.androidauthority.com/google-powering-ai-816146/>

# Input normalisation

Why normalize?



Gradient of larger parameter  
dominates the update

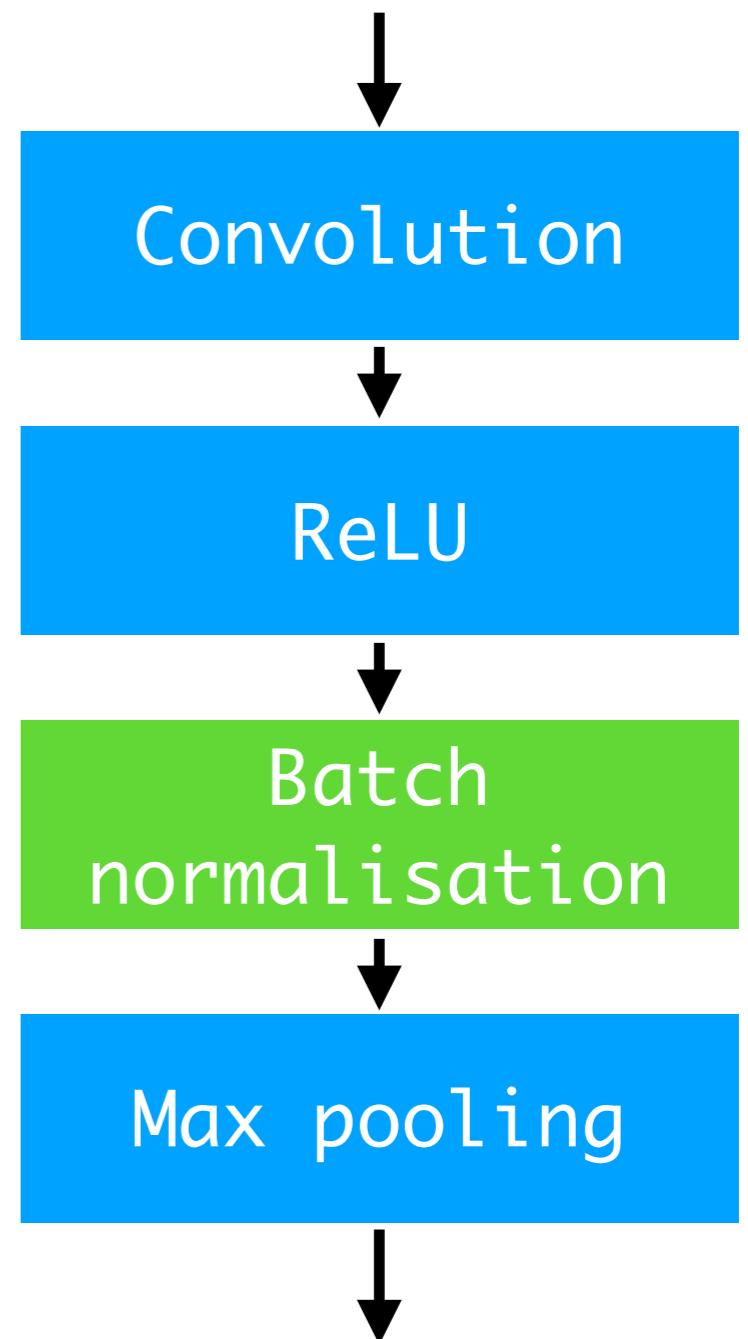


Both parameters can be  
updated in equal proportions

# Model improvement

## Batch normalisation

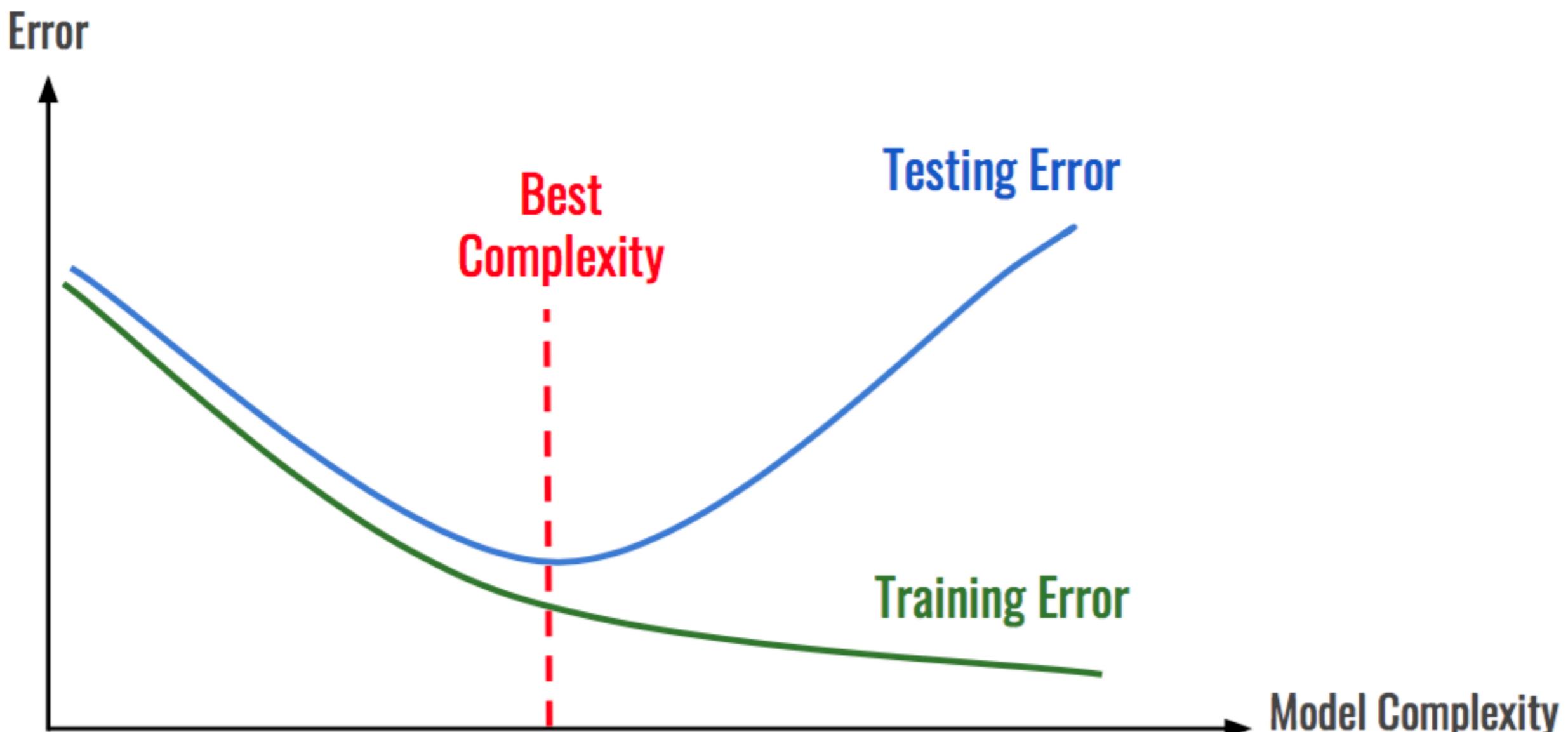
- Why only do this normalisation on the input?
- Renormalising in an intermediary layer speeds up learning.
- We compute the mean and variance per batch and uses it to normalise to **0 mean** and **variance 1**.



# Batch normalisation

- Adding BN will add more parameters to the model and extra computation.
- BN allow us to more easily train deeper networks.
- BN makes the network more robust to hyperparameter selections.
- BN allows us to train with a higher learning rate.
- We apply BN before the activations.

# Controlling overfitting



Source: [Hackernoon](#)

# Regularisation

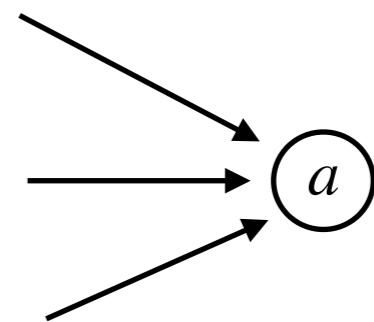
- Controlling the complexity of the model
  - Early stopping
  - Simplifying our network
  - L2 regularisation
  - Dropout

# L2 regularisation

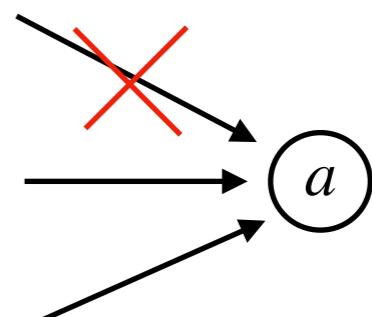
- We add a new term to the total loss function.
- This term adds additional loss to the function which takes the value of the weights into account.
- We then optimise this new loss function instead.
- A new **hyperparameter**,  $\lambda$  is added. This is usually a small value and we will need trial and error to find an acceptable value.

# L2 regularisation

- Why does L2 regularisation work?
- We add a cost to the weights, thus making a "**more complex**" model **more expensive**.
- If some learnt weight is high (say, 10) it "costs" more than a weight with value 1.
- Thus, our model becomes "simpler" by **forcing the weights down**.
- When some weights are forced to 0, we are effectively "**removing**" connections.

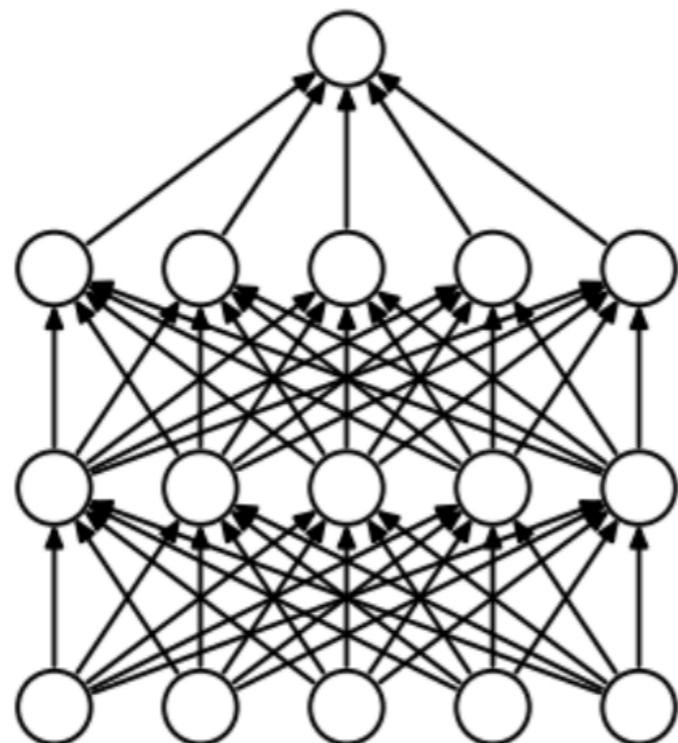


Set weight to 0

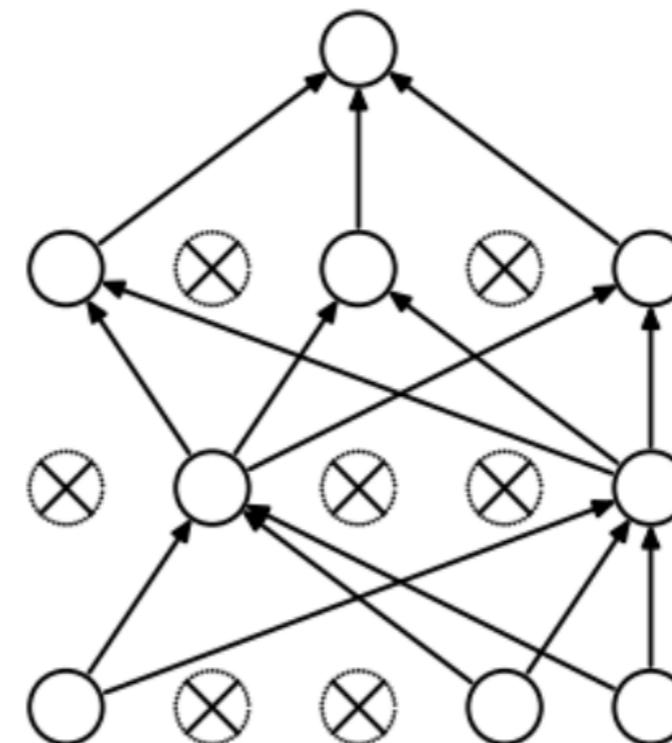


# Dropout

Dropout will **randomly set the activations** of neurons to 0.



(a) Standard Neural Net



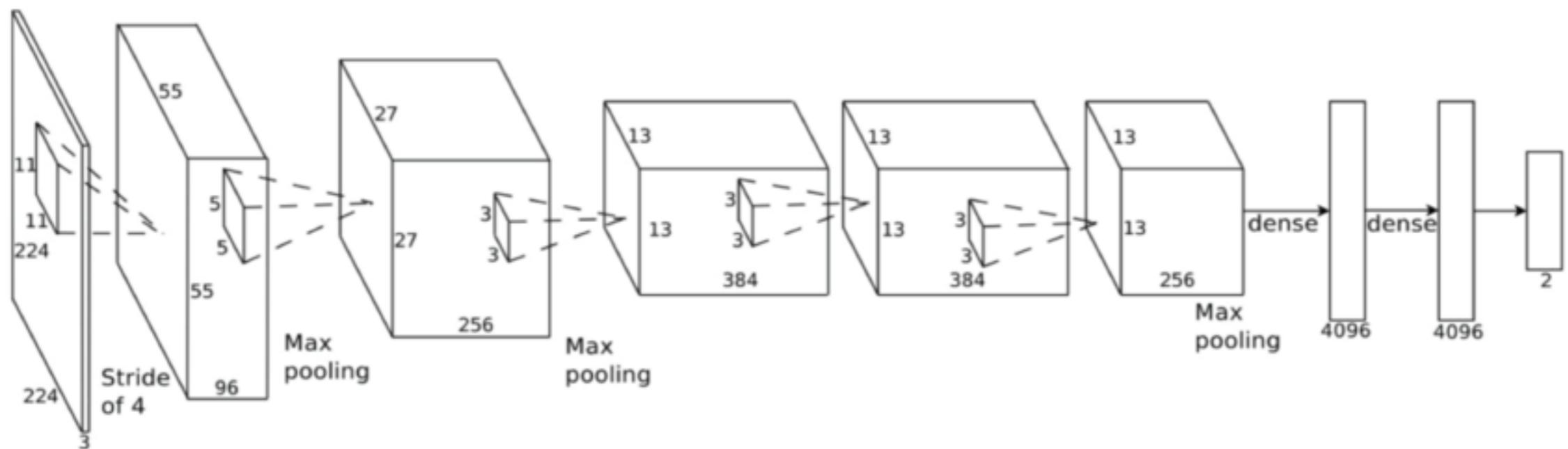
(b) After applying dropout.

Source: [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#), Srivastava et al., 2014

# Transfer learning

## Pretrained CNNs

- Should we develop CNNs ourselves?
- No, we used them, pre-trained
- This is called **transfer learning**.

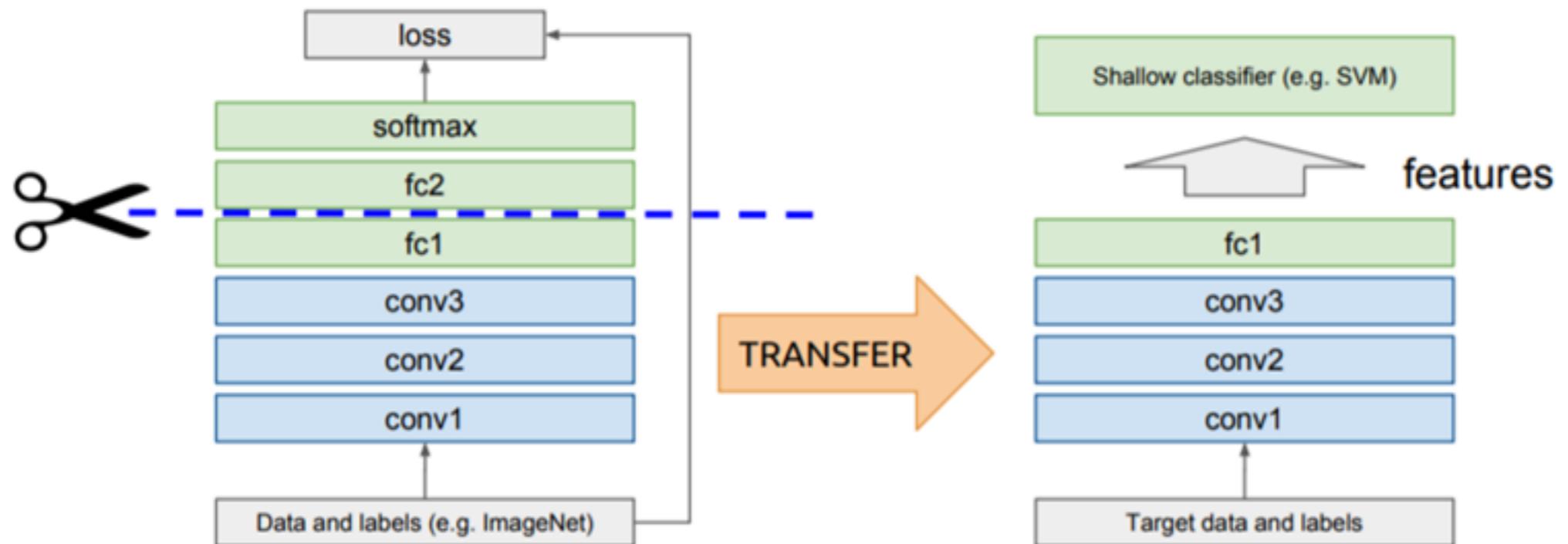


Source: <https://www.jeremyjordan.me/convnet-architectures/>

# Transfer learning

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that  $D_S = D_T$



# Hands-on

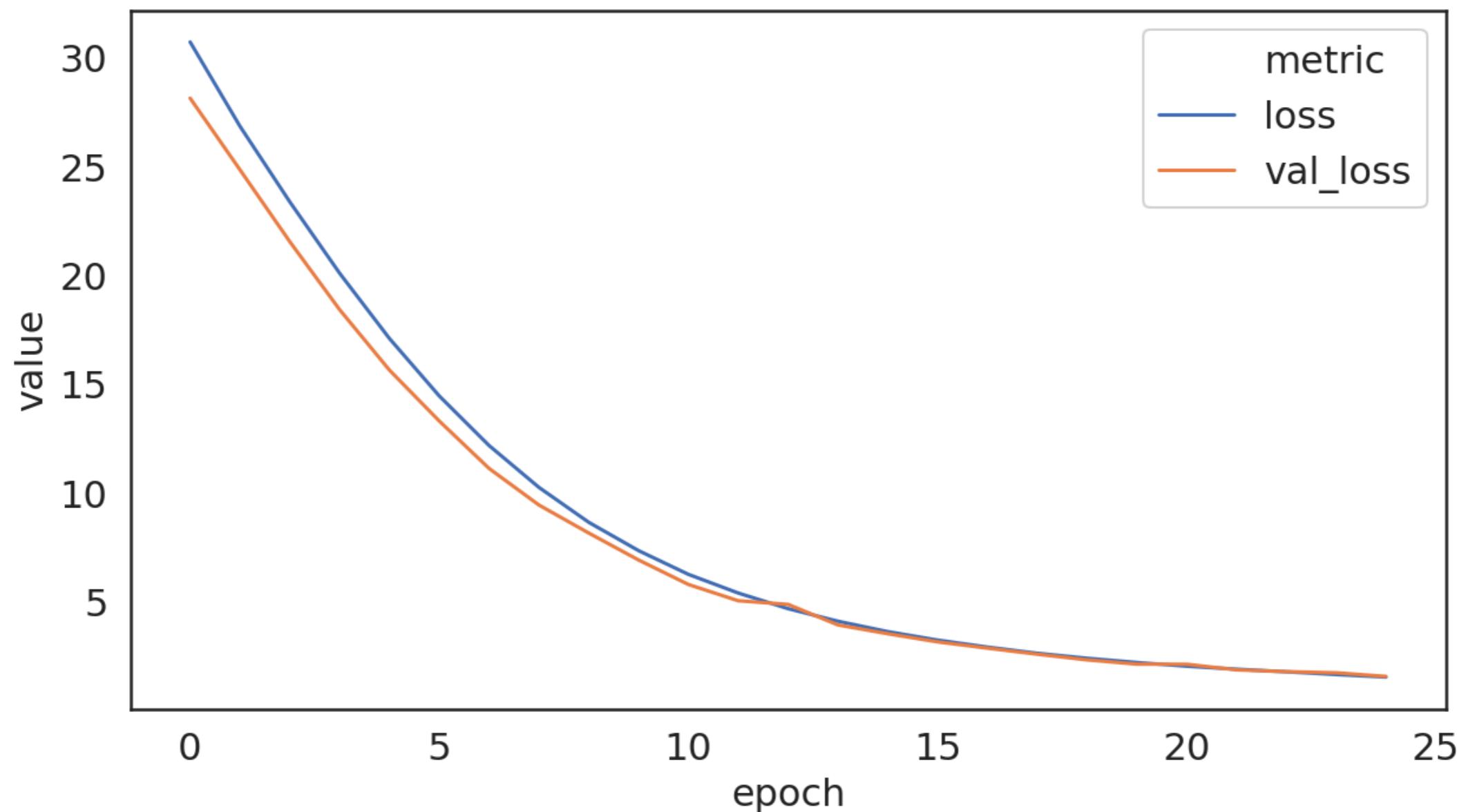


**Kill your notebooks!**

Notebook: 04-improving-neural-networks.ipynb

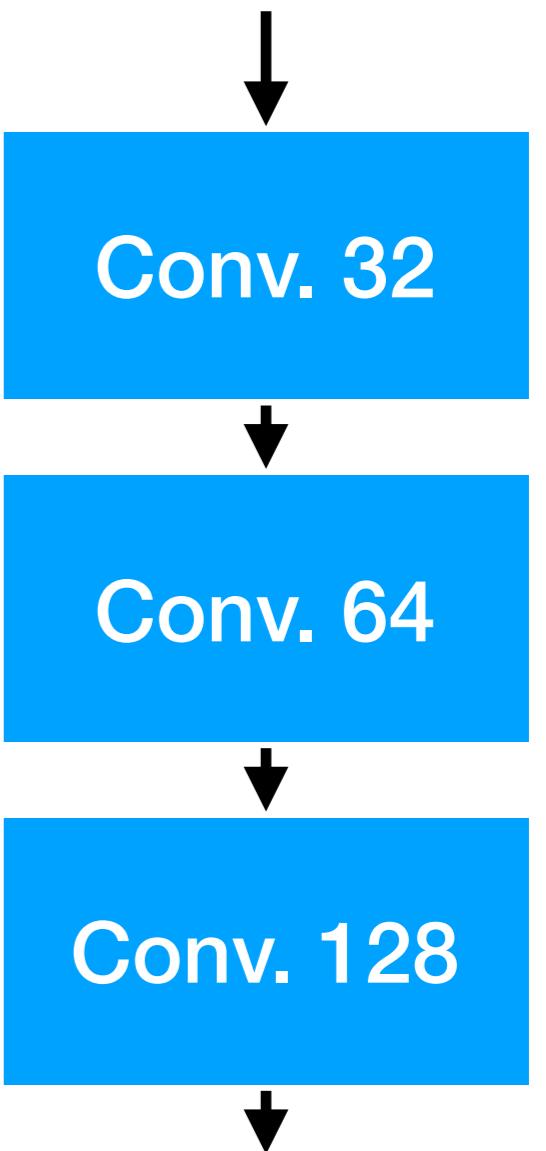
**Please fill out the survey: <https://bit.ly/2HaYGXX>**

# Recap



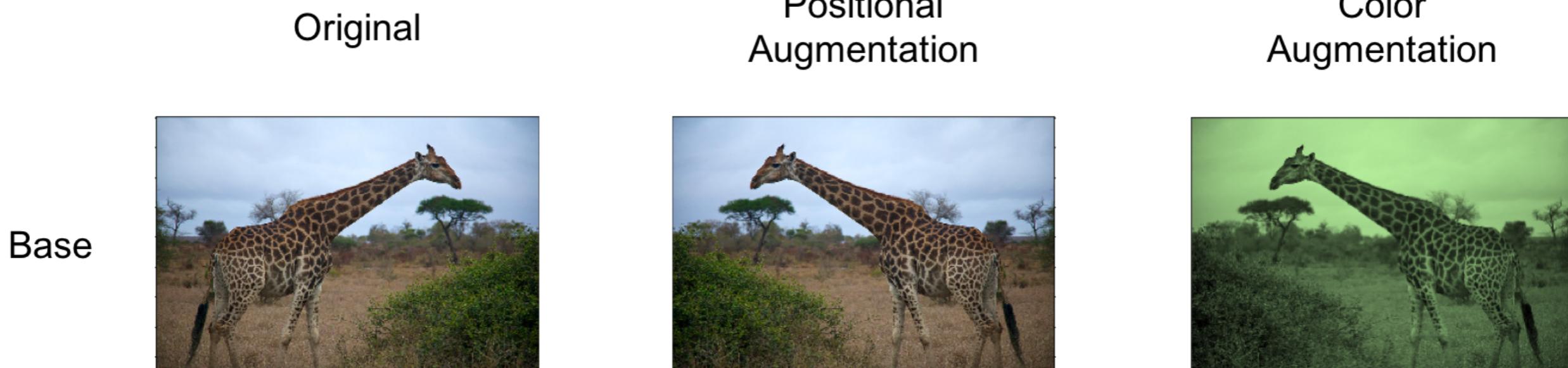
# Rules of thumb

- Number of filters as multiple of two
- Context with convolution
- Classification with dense networks
- First overfit, then regularise



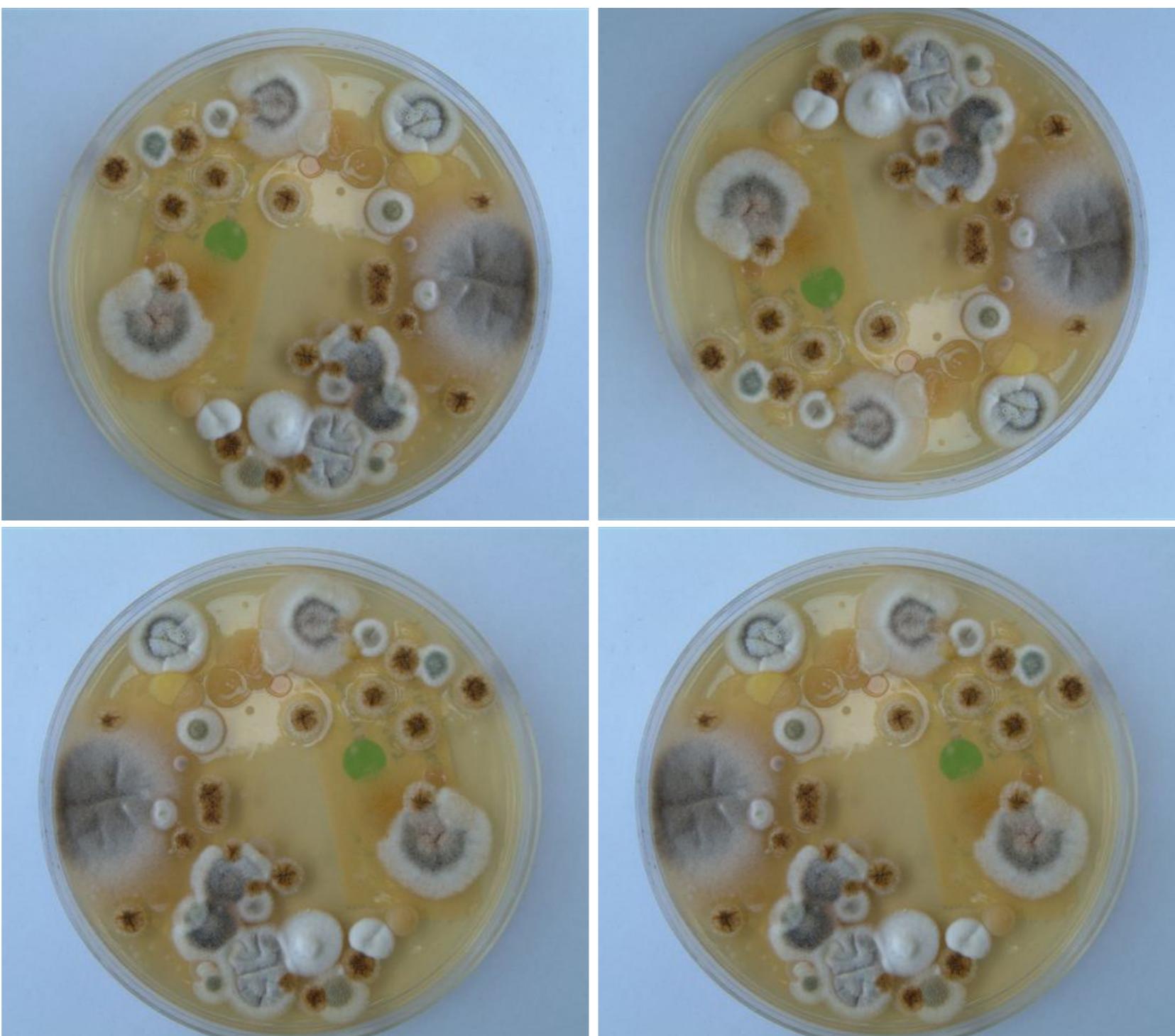
# Life science & deep learning

Augmentation for small data sets

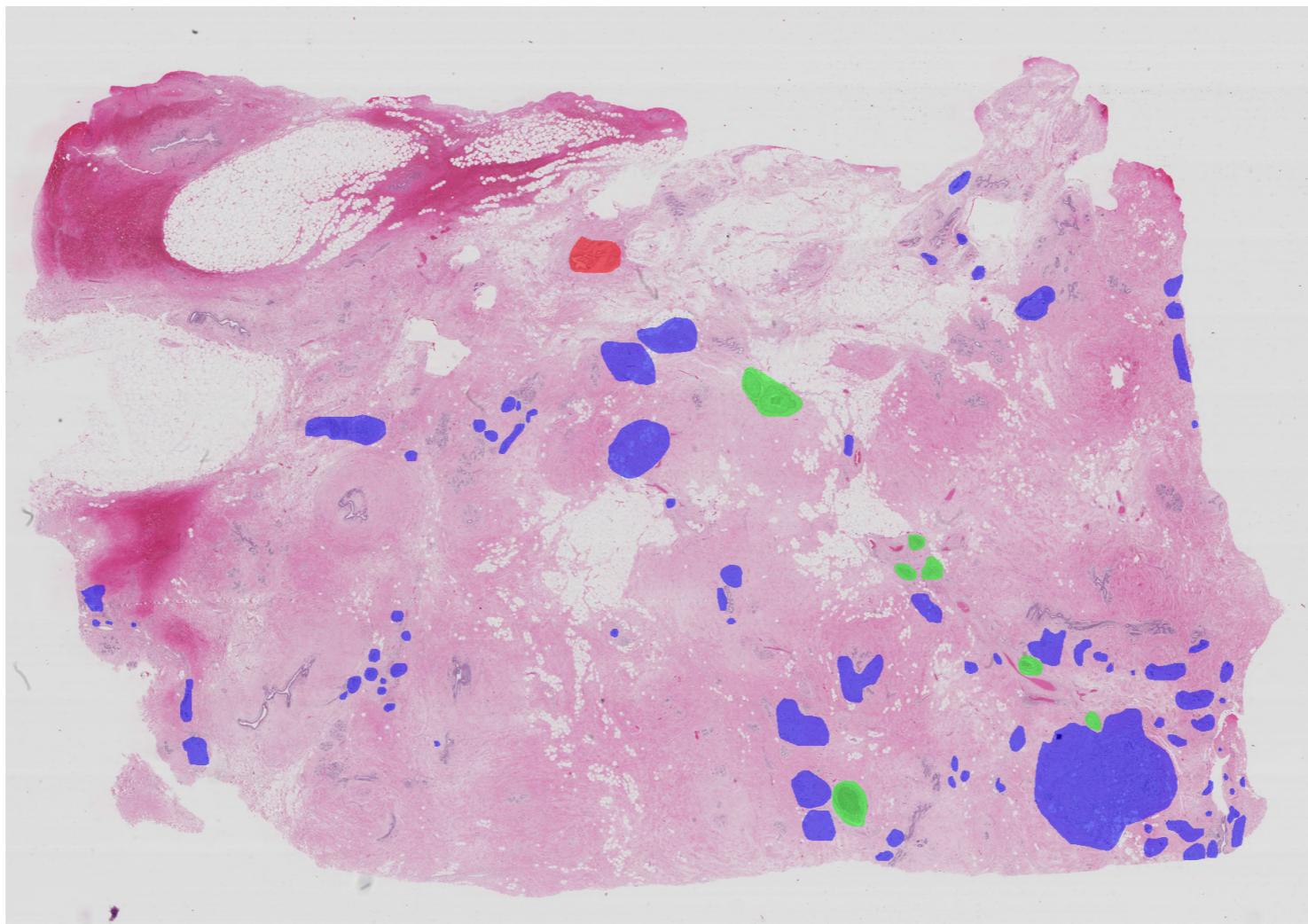


Source: [Data Augmentation with Masks](#)

# Augmentation



# Large images

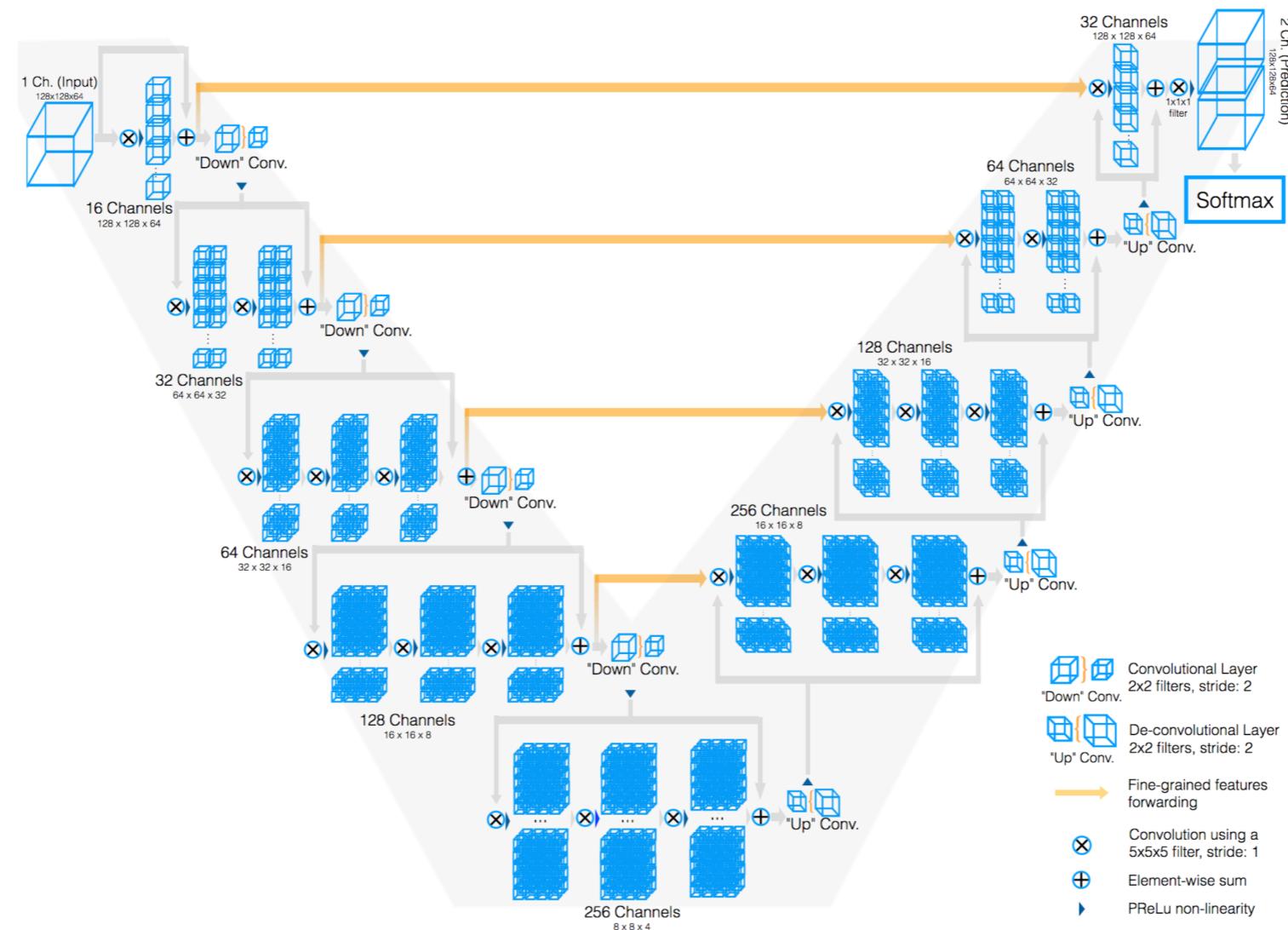


42113 x 62625 pixels

Source: [BACH Grand Challenge](#)

# Life science & deep learning

## Large (volumetric) images

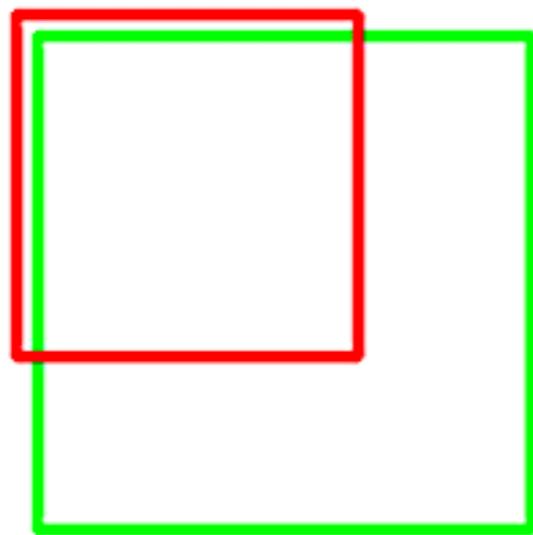


Source: [V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation](#)

# Life science & deep learning

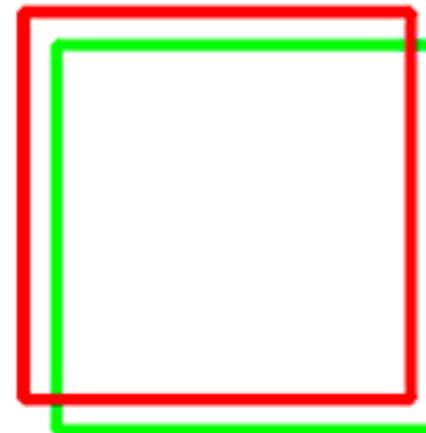
Custom metrics or optimisation

IoU: 0.4034



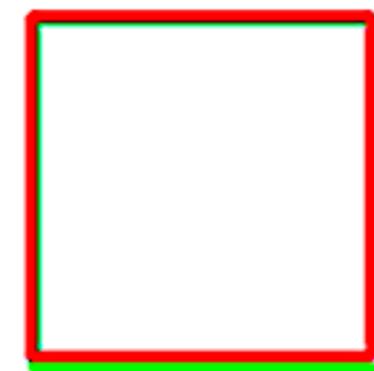
Poor

IoU: 0.7330



Good

IoU: 0.9264

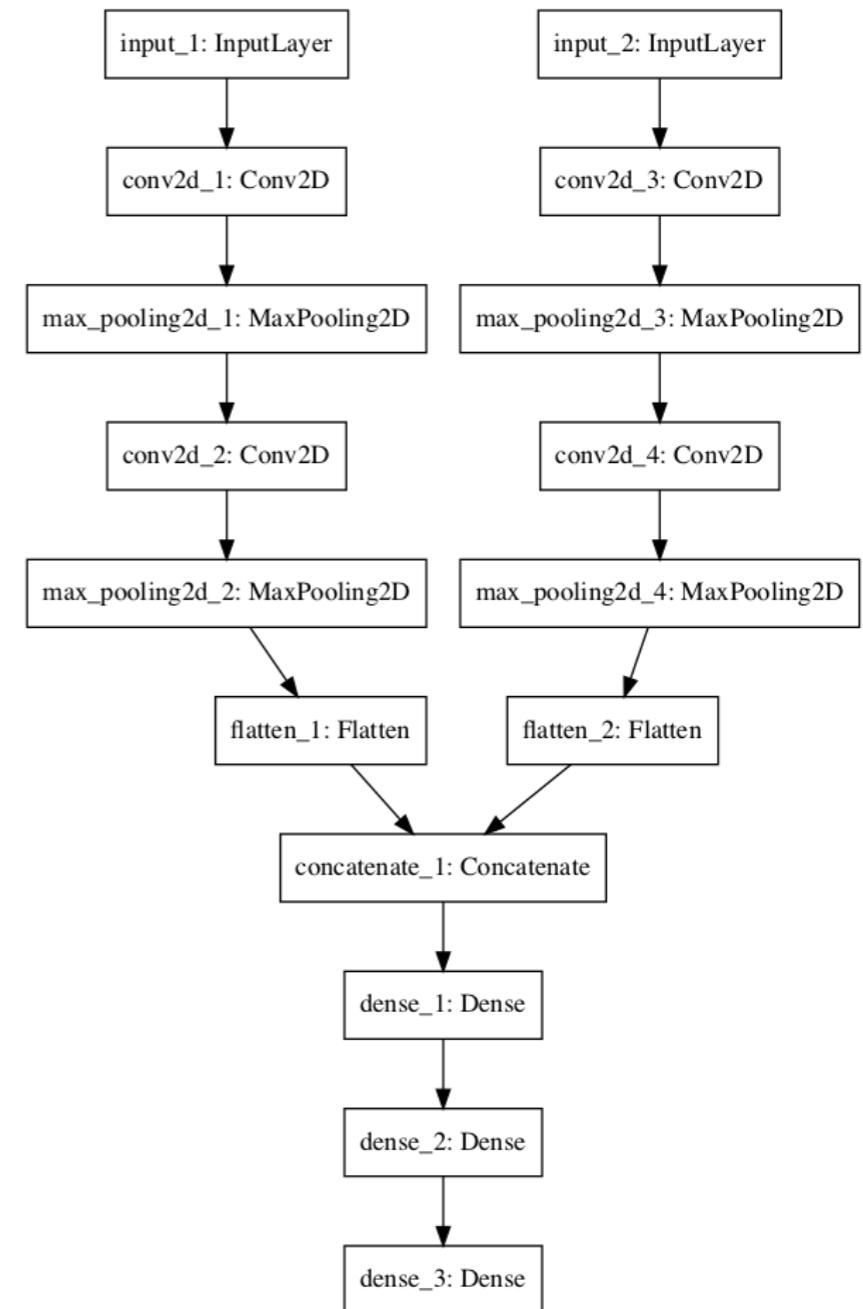


Excellent

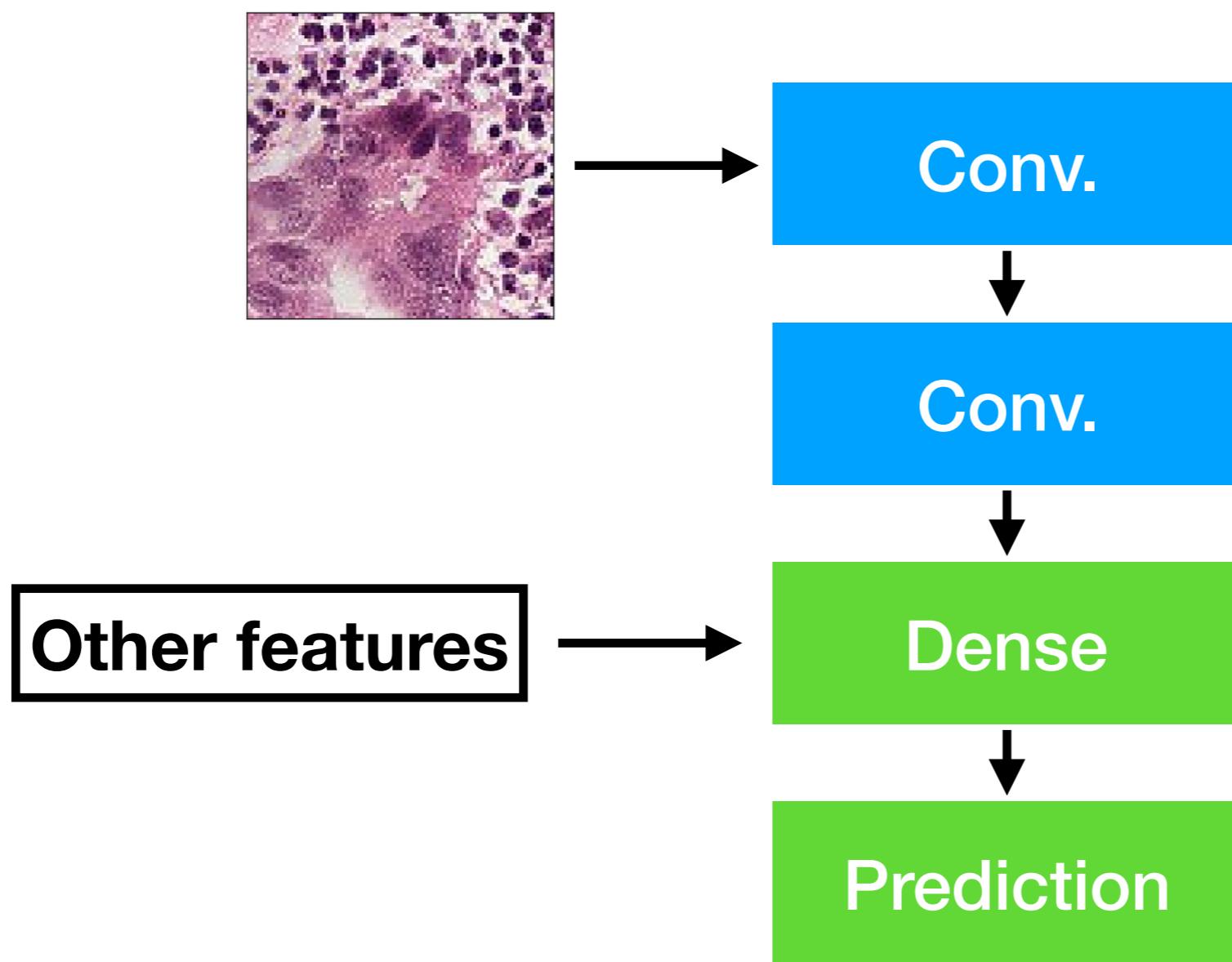
Source: [Deep Learning in Medical Imaging V](#)

# Domain-specificity

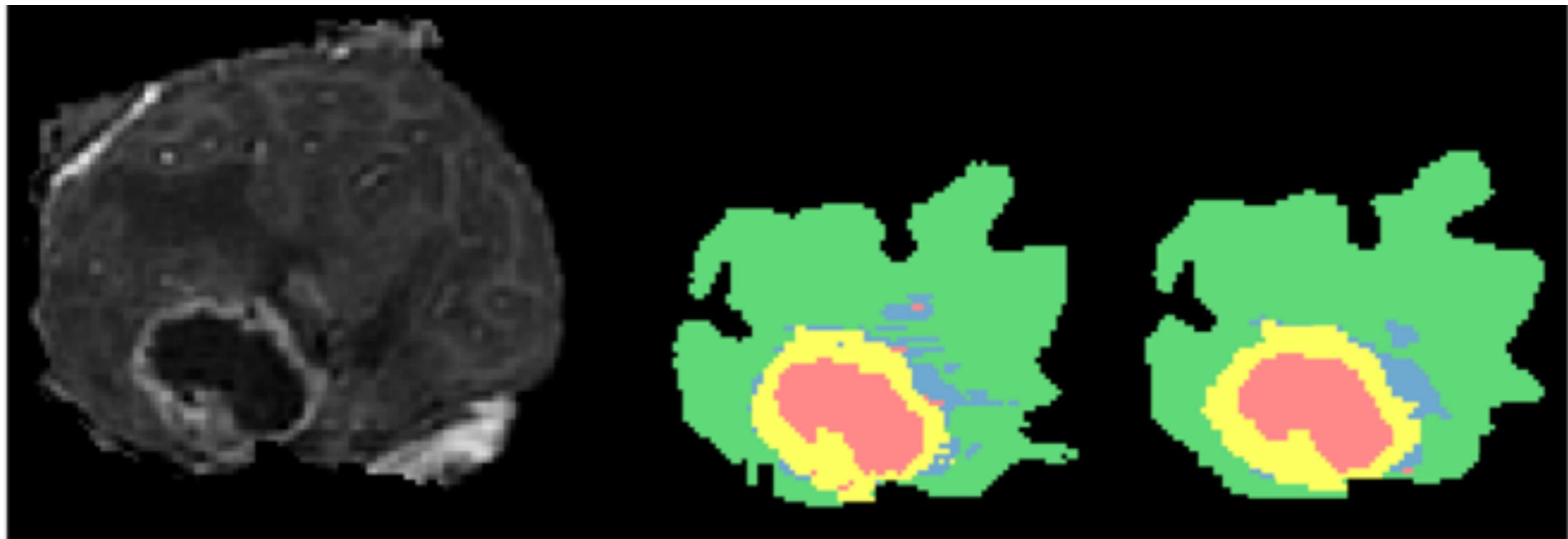
- Combining data
- Application of domain knowledge
- ‘Domain-specific’ normalisation



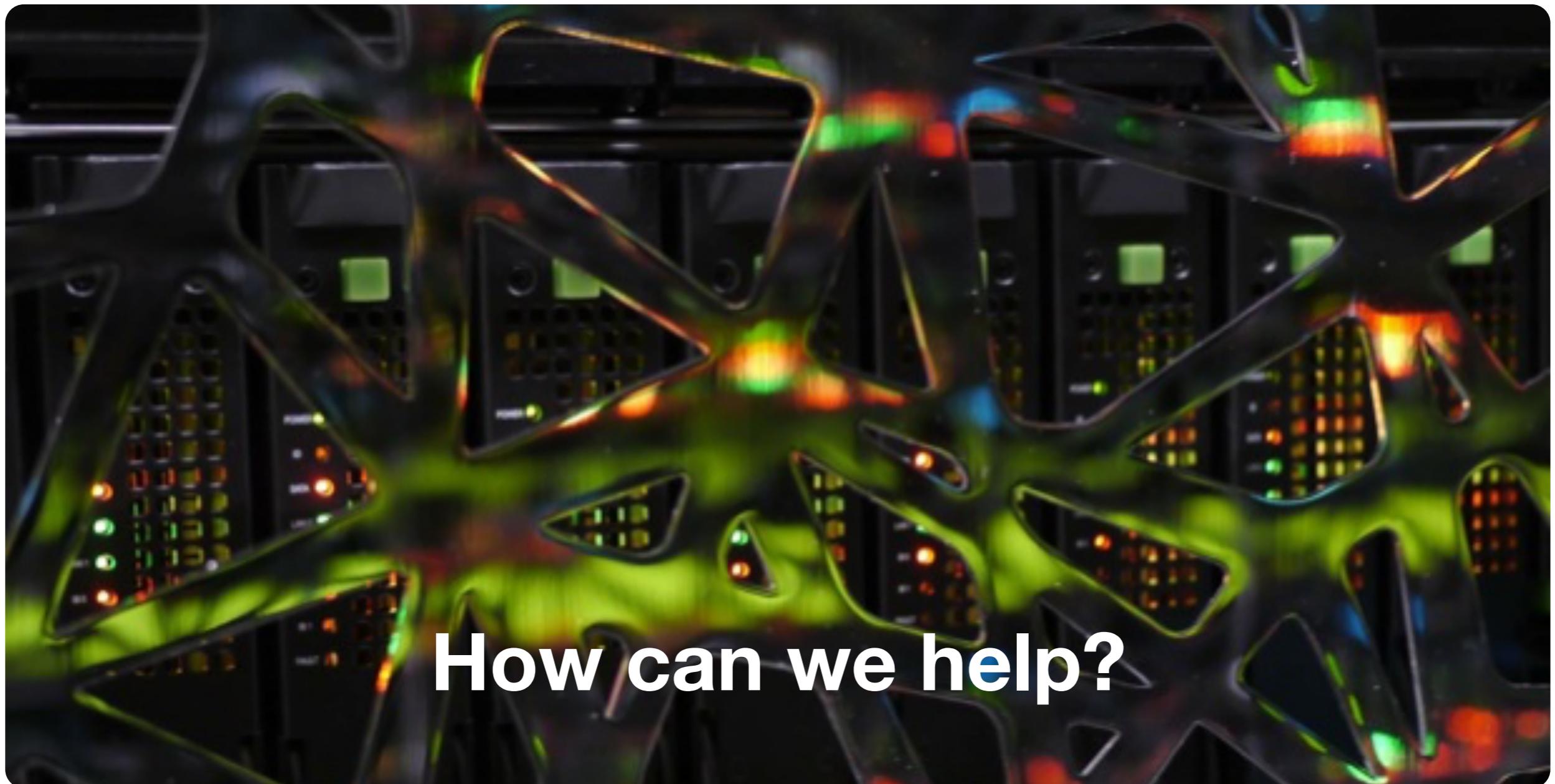
# Combining data



# Interpretability

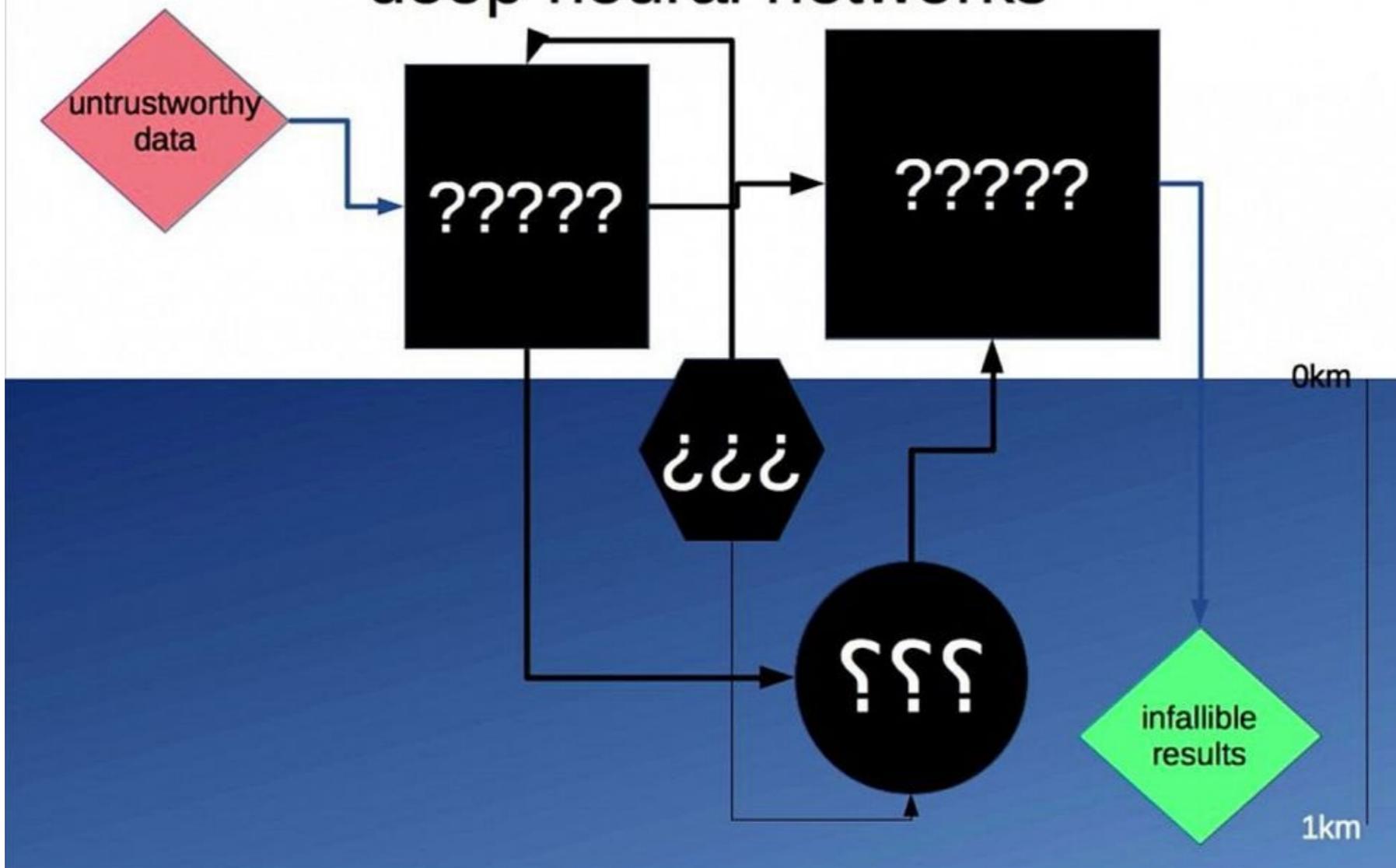


# Consultancy



How can we help?

## deep neural networks



[helpdesk@surfsara.nl](mailto:helpdesk@surfsara.nl)