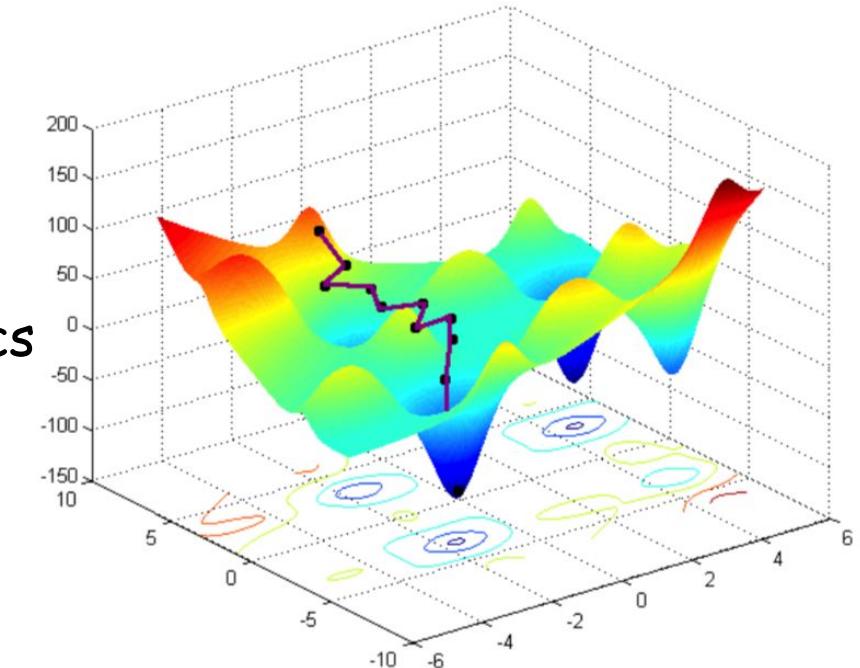
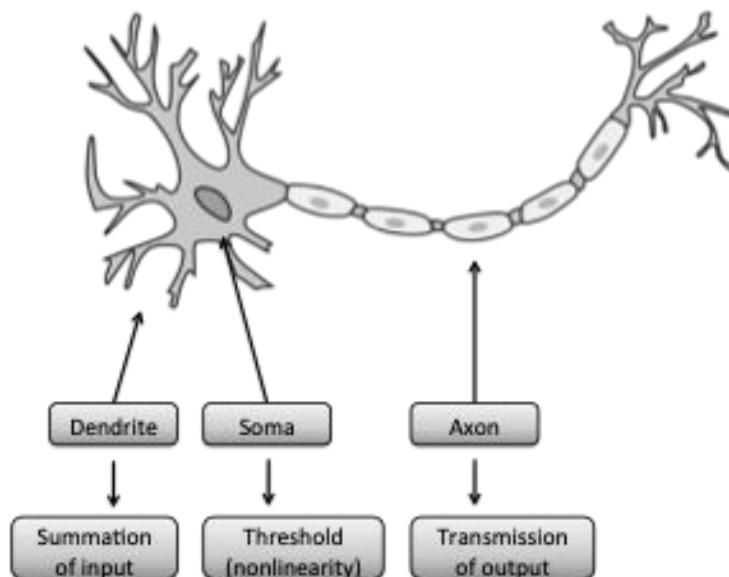


# Deep Learning



By Phil Harrison  
[philip.harrison@farmbio.uu.se](mailto:philip.harrison@farmbio.uu.se)

Dept. of Pharmaceutical bioinformatics  
Uppsala University



...with some slides adapted/inspired  
by material in lectures given by:

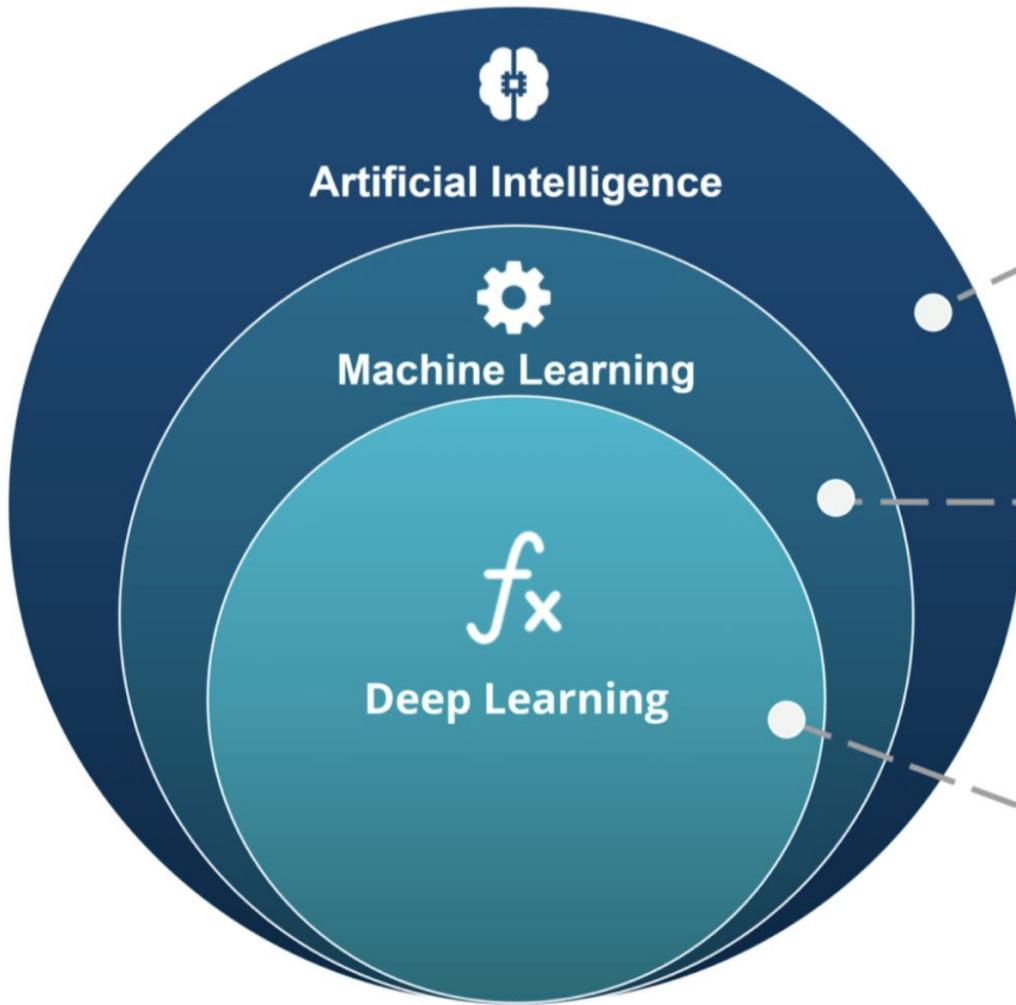
Anindya Gupta, Joakim Lindblad &  
Sajith Sadanandan



Pharmaceutical Bioinformatics  
research group

Statistics and Prediction in the Pharmaceutical Sciences  
20th May 2019

# Deep learning and AI



## ARTIFICIAL INTELLIGENCE

A technique which enables machines to mimic human behaviour

## MACHINE LEARNING

Subset of AI technique which use statistical methods to enable machines to improve with experience

## DEEP LEARNING

Subset of ML which make the computation of multi-layer neural network feasible

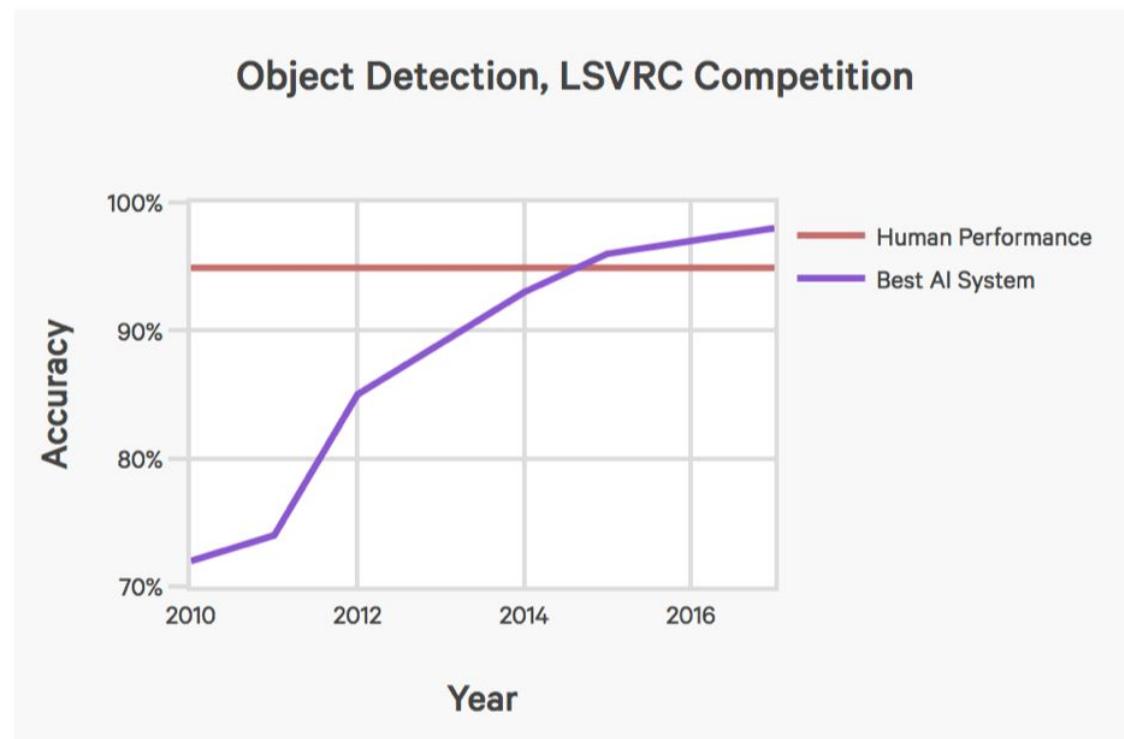
## TECHNICAL PERFORMANCE

### Vision

#### Object Detection

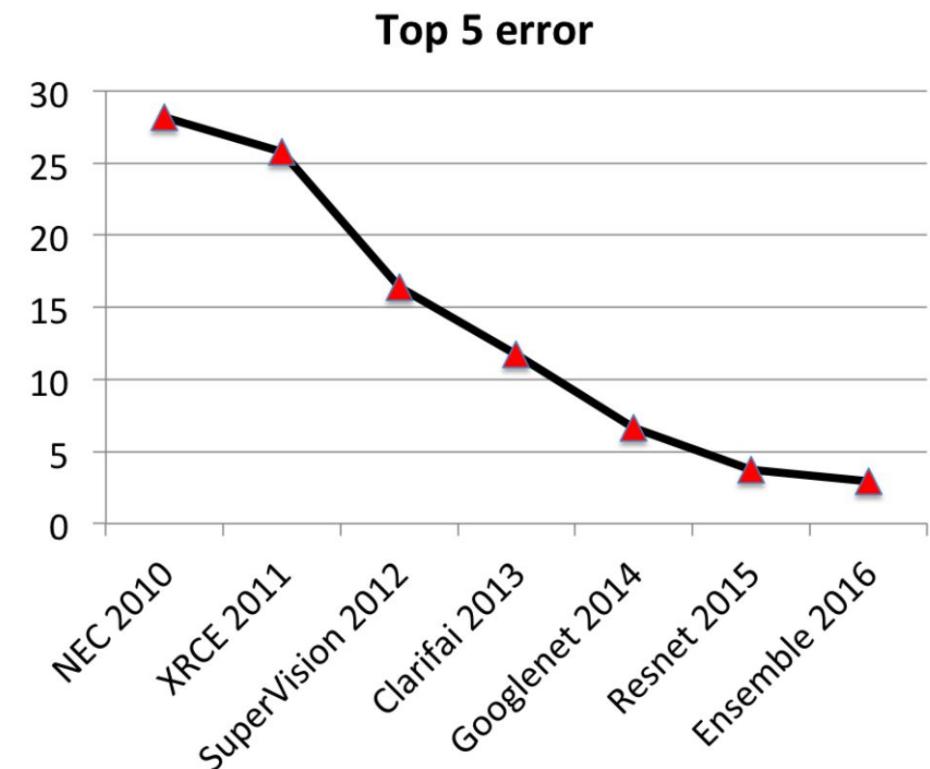
[view more information in appendix A10](#)

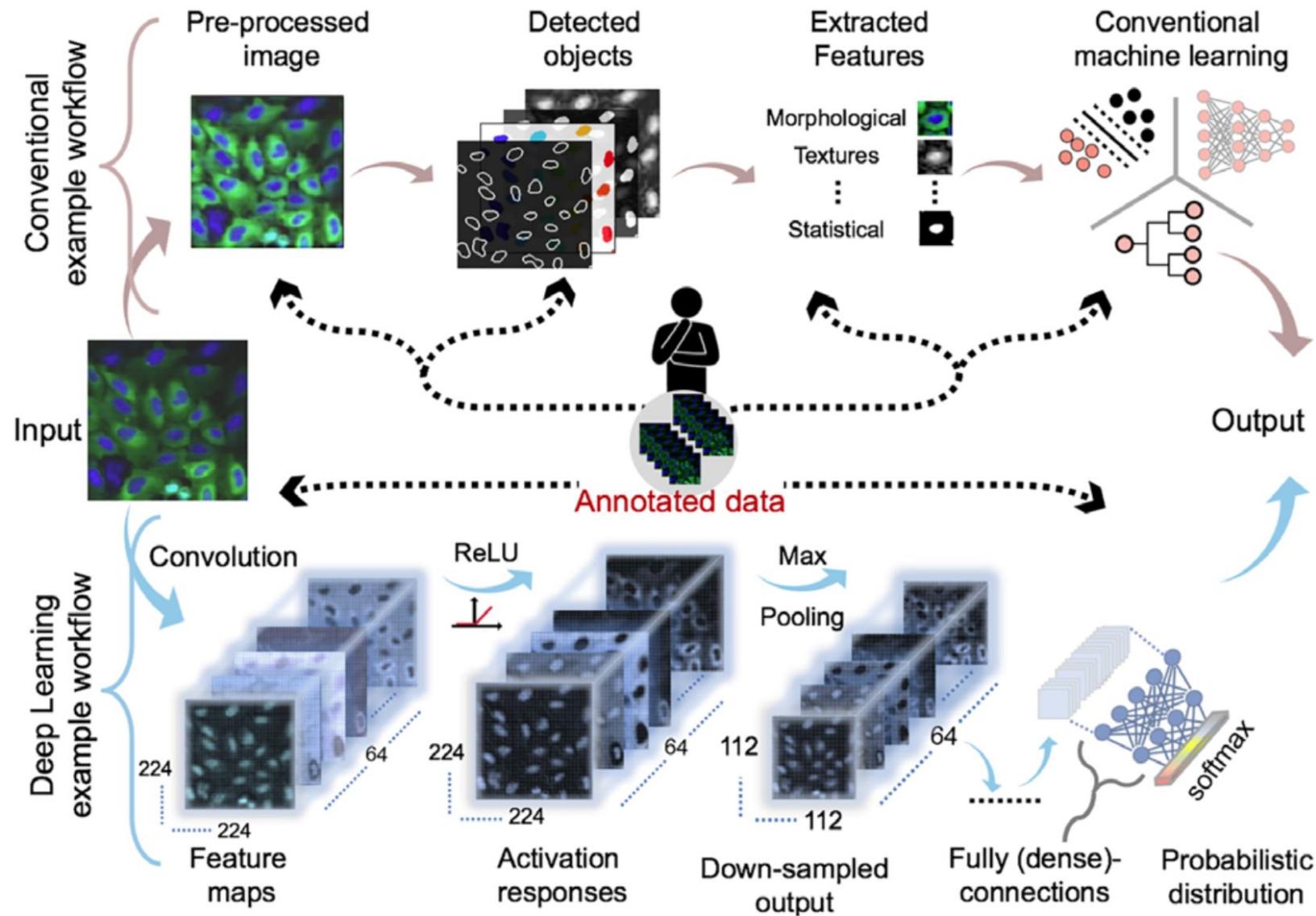
The performance of AI systems on the object detection task in the Large Scale Visual Recognition Challenge (LSVRC) Competition.



<http://www.aiindex.org/2017-report.pdf>

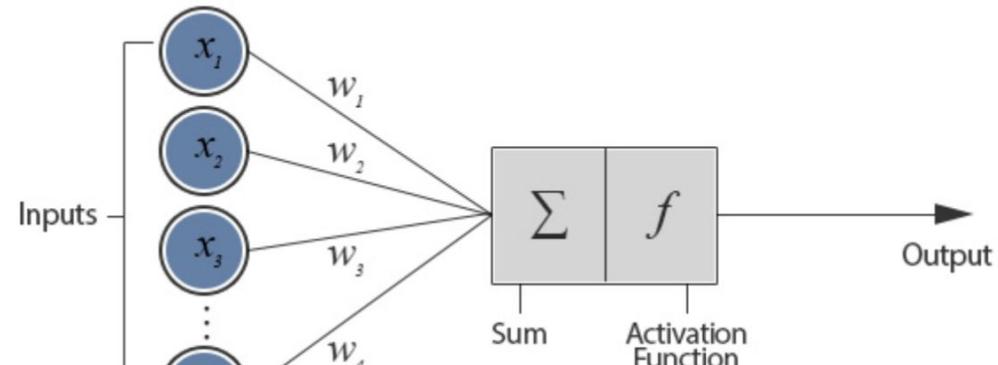
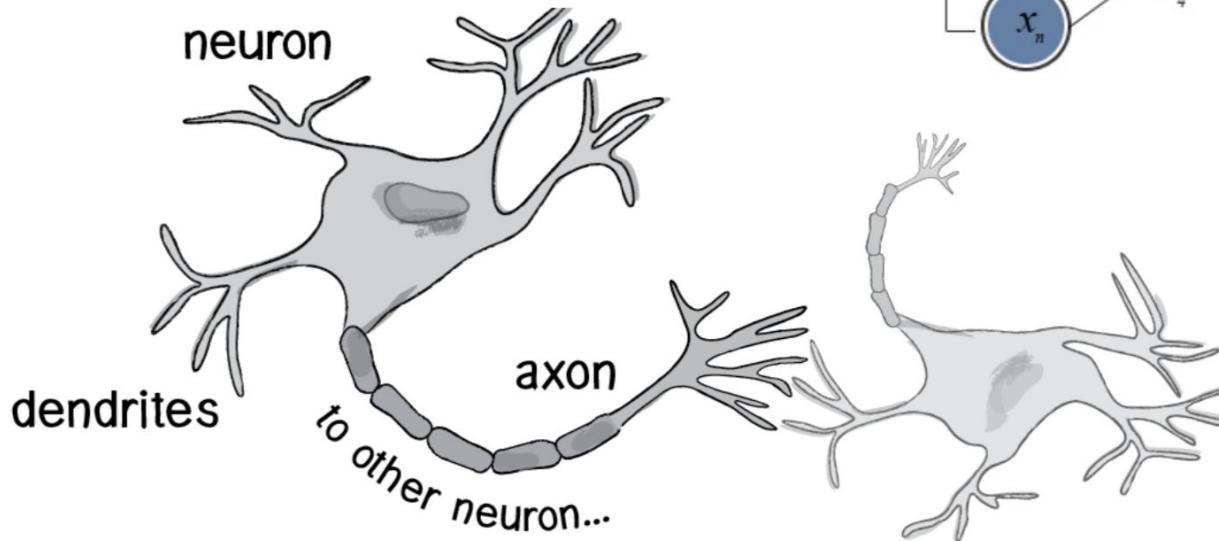
- subset of ImageNet dataset
- 1000 classes
- 1.2 million images
- From 2012 onwards won by CNNs



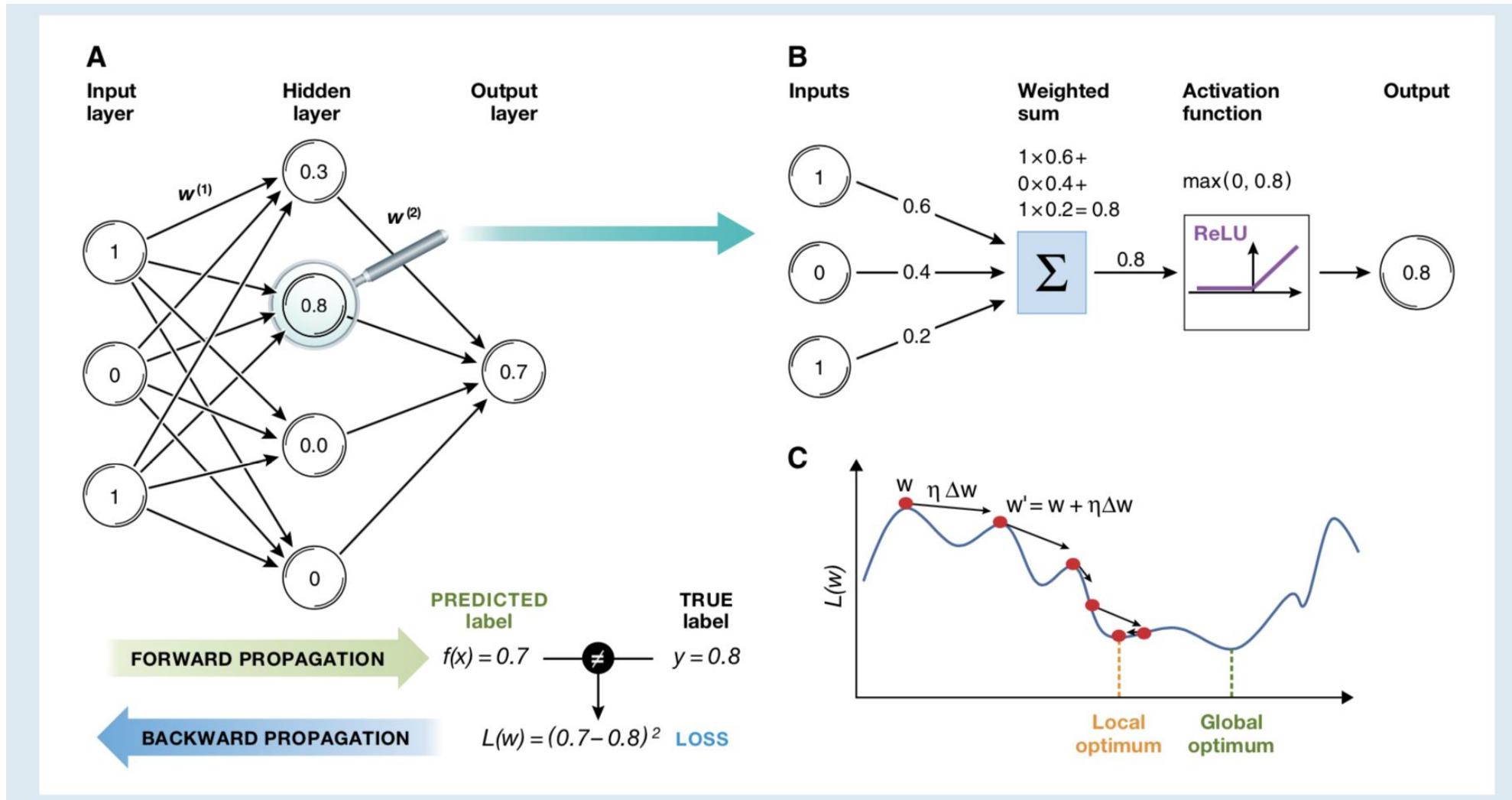


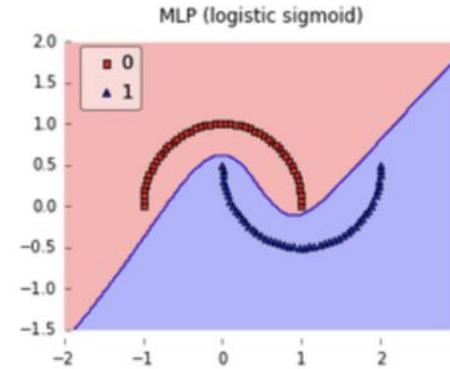
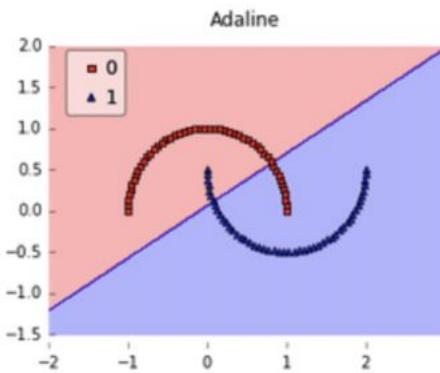
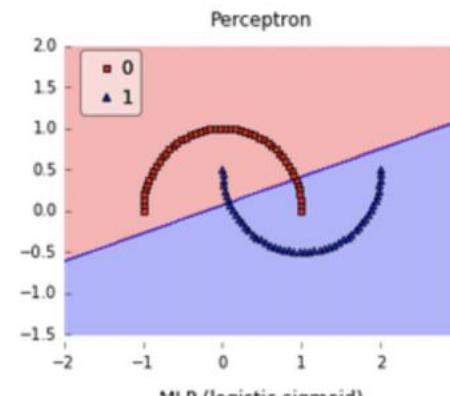
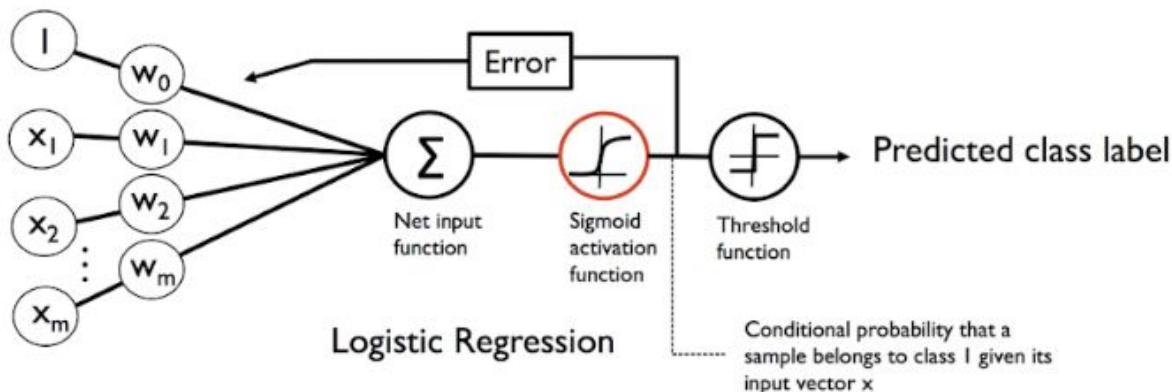
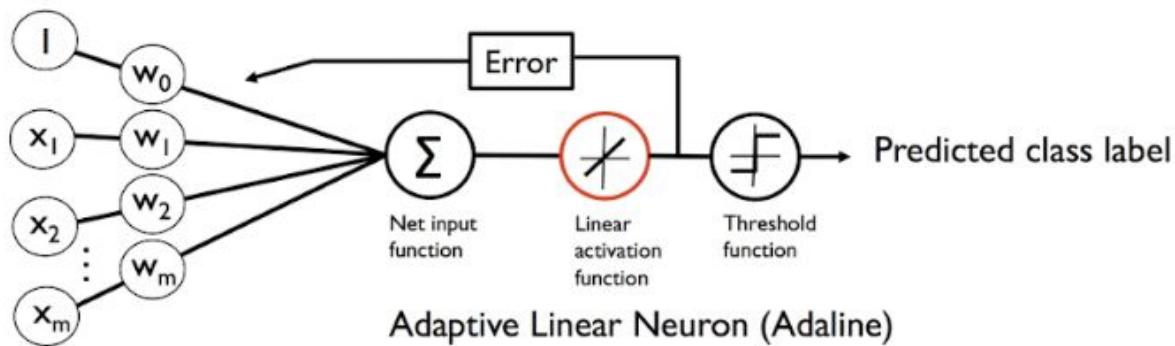
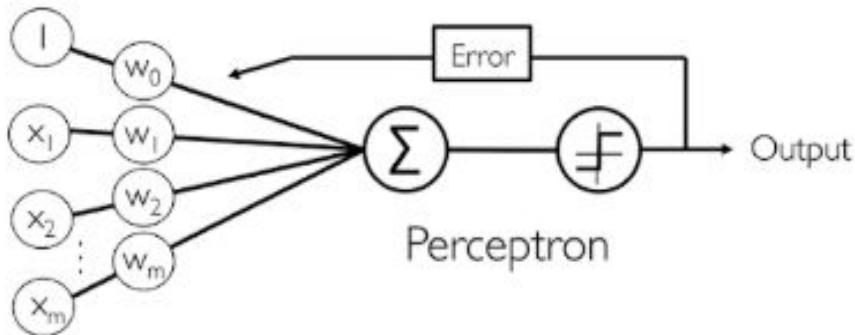
Overview of conventional versus deep learning workflows (from Gupta et al, 2019)

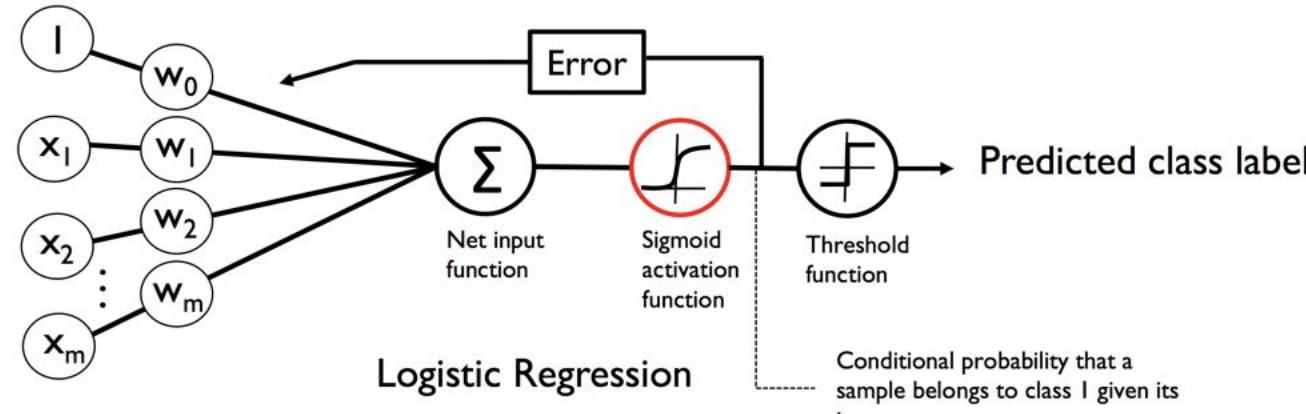
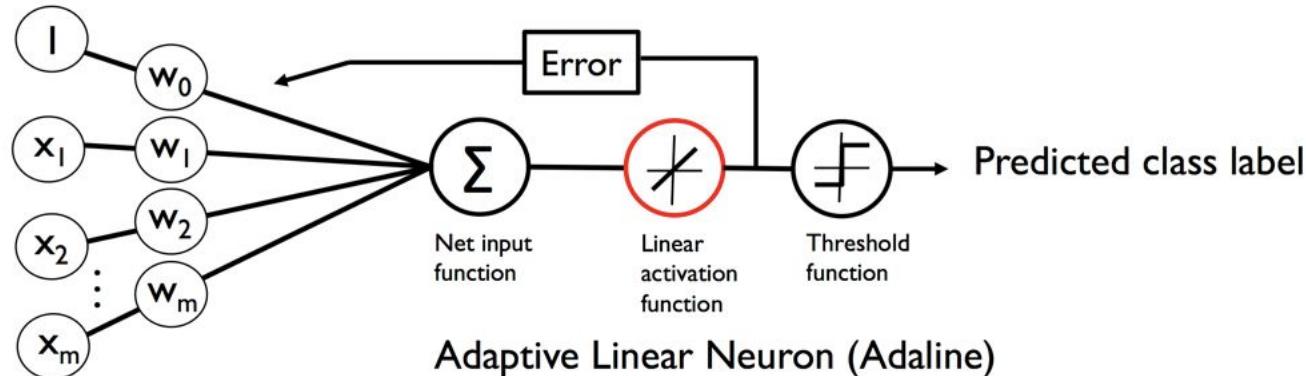
# Biological connection



# An introductory overview

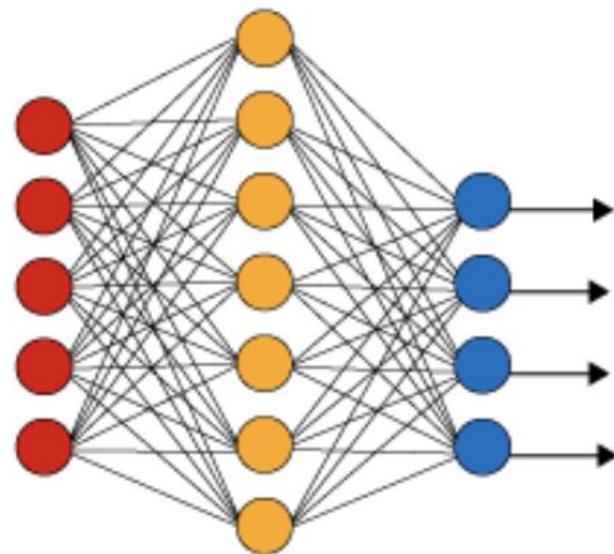






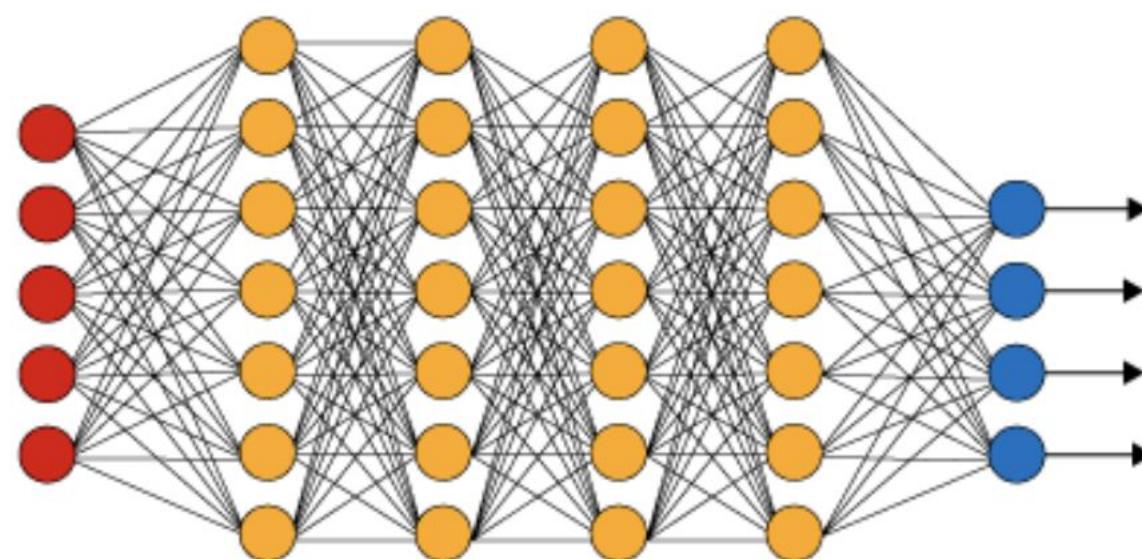
Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

## Simple Neural Network



● Input Layer

## Deep Learning Neural Network

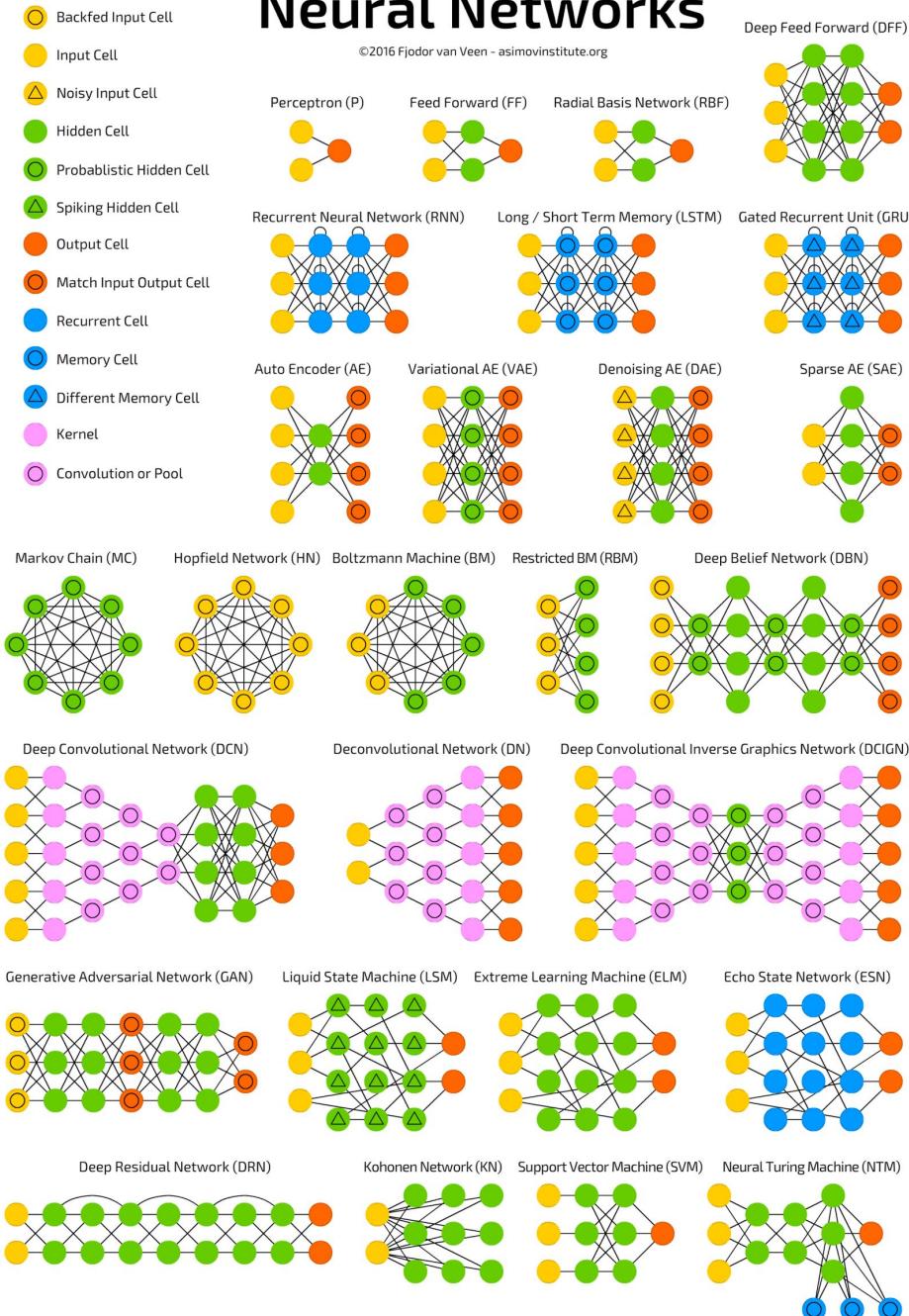


● Hidden Layer

● Output Layer

# Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org



# A multitude of types

- Usually require input data and corresponding output labels for training
- Automatically learn the necessary features to label unseen data
- Choice of network depends on the problem at hand
- **Convolutional neural networks (CNNs) are ideal for image data and will be the focus of this lecture and the lab tomorrow**

<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

# Image classification

- Is a hard problem!
- Switching to Stanford slides for some feline friends to show us why!

# Image Classification: a core task in Computer Vision



(assume given set of discrete labels)  
{dog, cat, truck, plane, ...}



cat

# The problem: *semantic gap*

Images are represented as  
3D arrays of numbers, with  
integers between [0, 255].

E.g.  
300 x 100 x 3

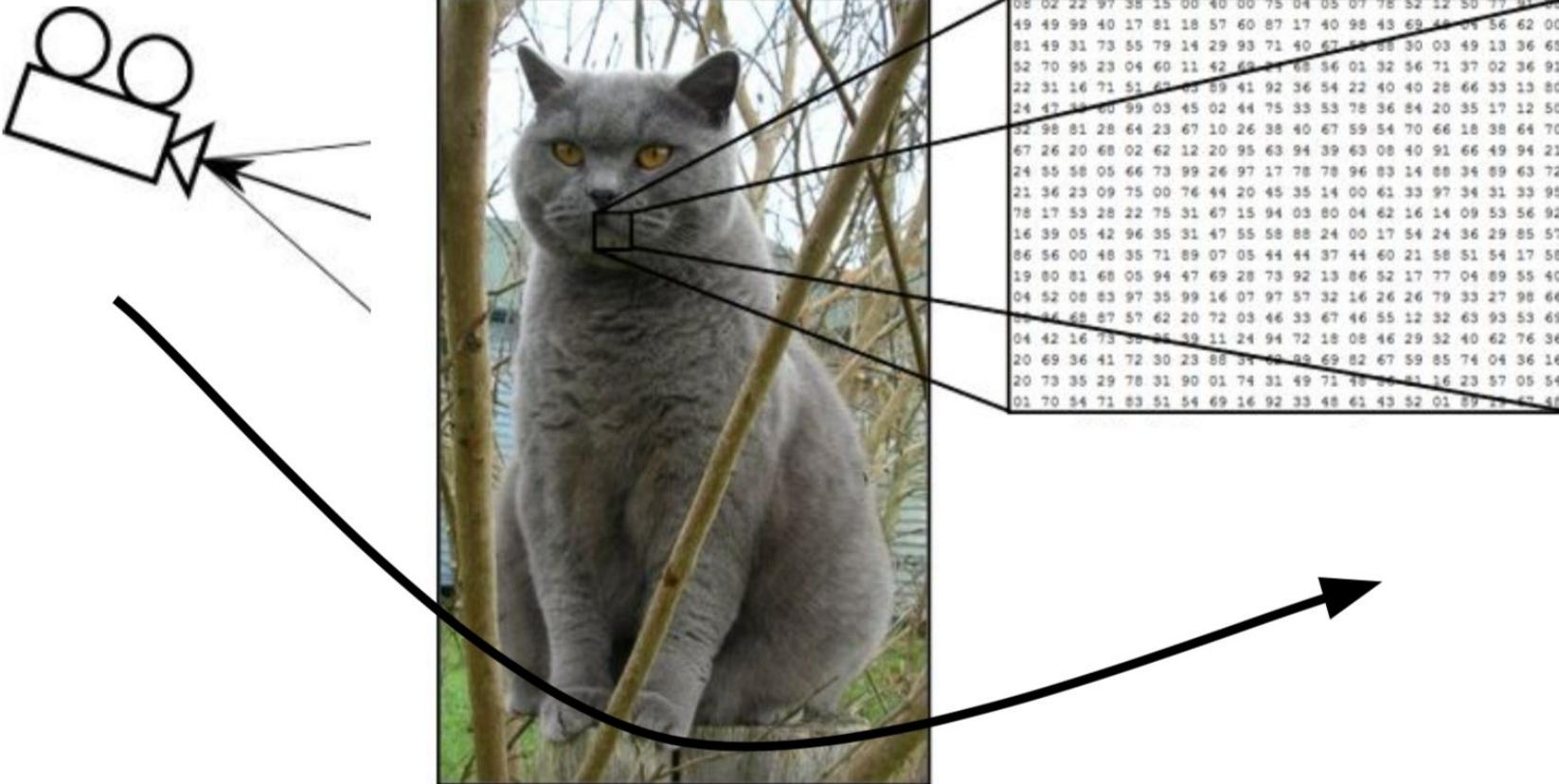
(3 for 3 color channels RGB)



08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	17	01	03
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	55	08	30	03	49	13	36	65
52	70	95	23	04	60	11	42	63	31	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	62	43	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	39	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	03	14	68	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
29	84	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	55	51	59	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	69	92	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	98	64	81	16	23	57	05	54
03	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	31	47	48

What the computer sees

# Challenges: Viewpoint Variation



# Challenges: Illumination



# Challenges: Deformation



# Challenges: Occlusion



# Challenges: Background clutter



# Challenges: Intraclass variation



# An image classifier

```
def predict(image):
    # *****
    return class_label
```

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for  
recognizing a cat, or other classes.

## Data-driven approach:

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):  
    # build a model for images -> labels...  
    return model  
  
def predict(model, test_images):  
    # predict test_labels using the model...  
    return test_labels
```

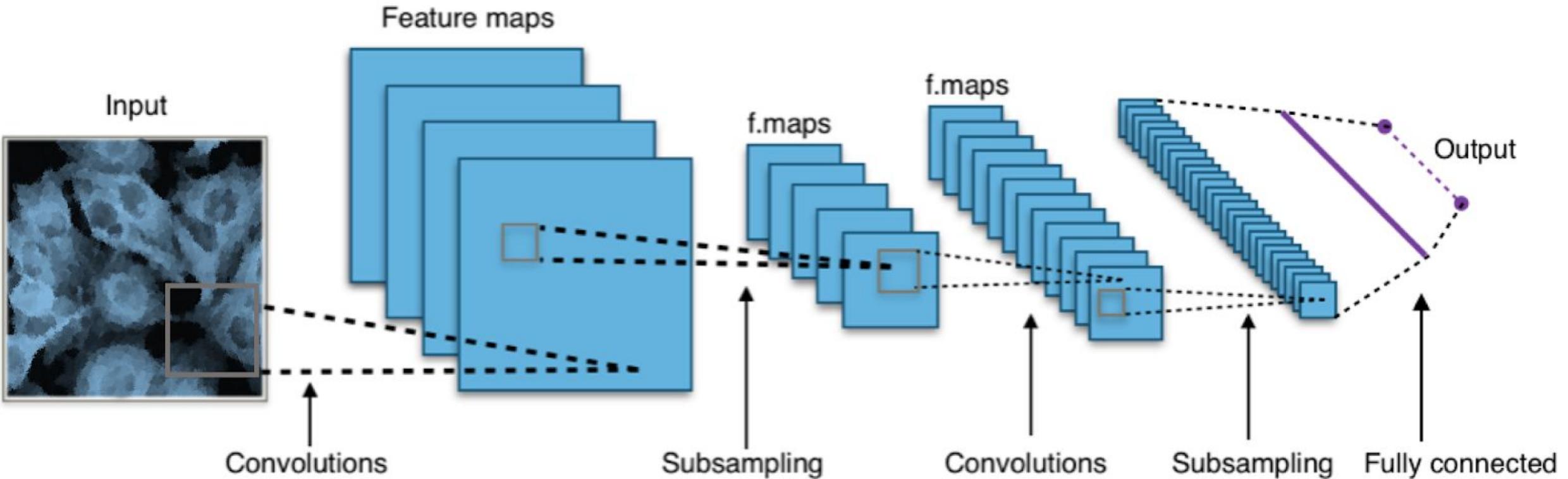
Example training set



# Convolutional Neural Networks (CNNs)

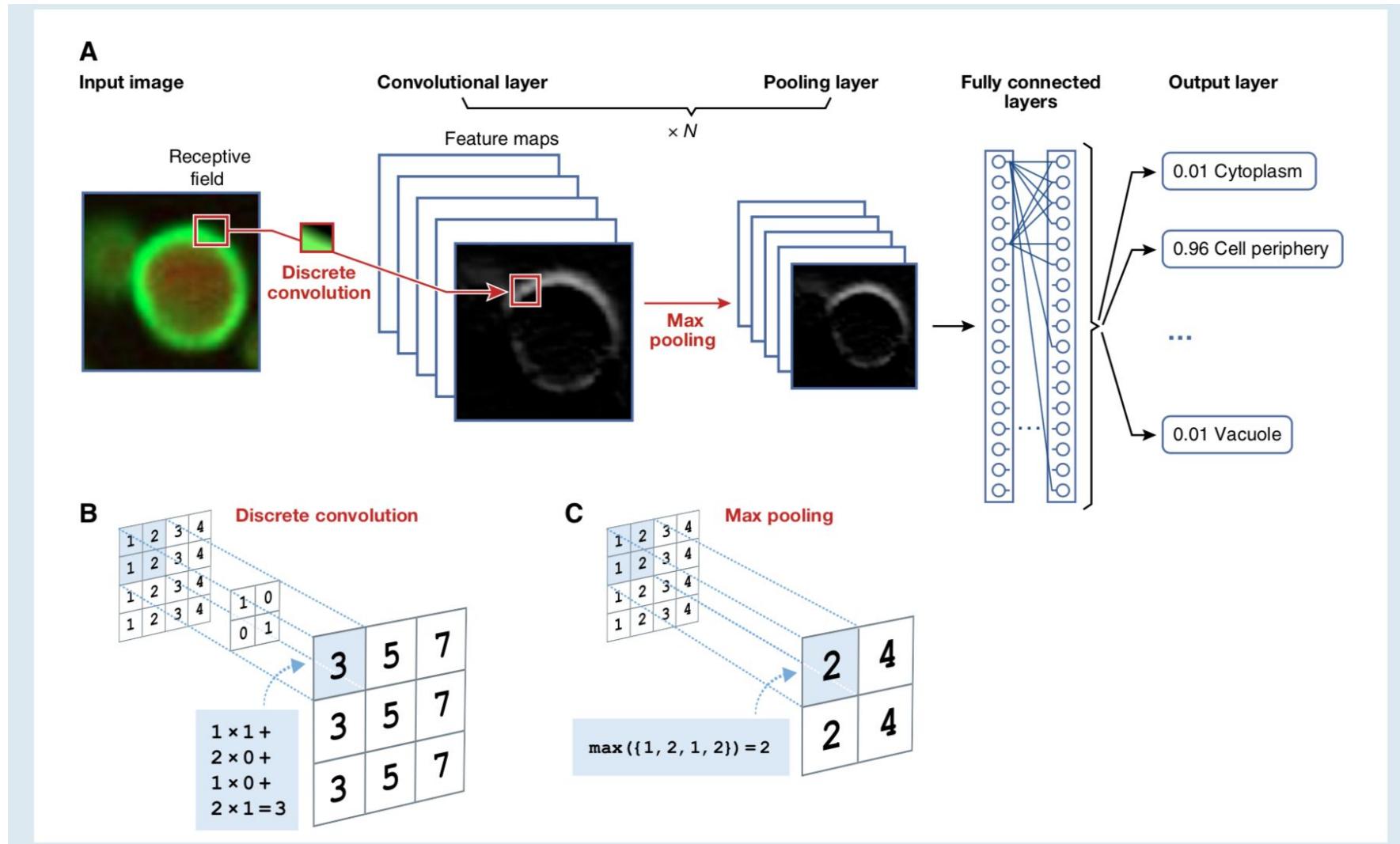
- Employ convolution operations
- Only local connectivity
- Spatial relationship is preserved
- Parameter sharing
- Translations equivariant
- Widely used in image analysis

# Illustration of a typical CNN

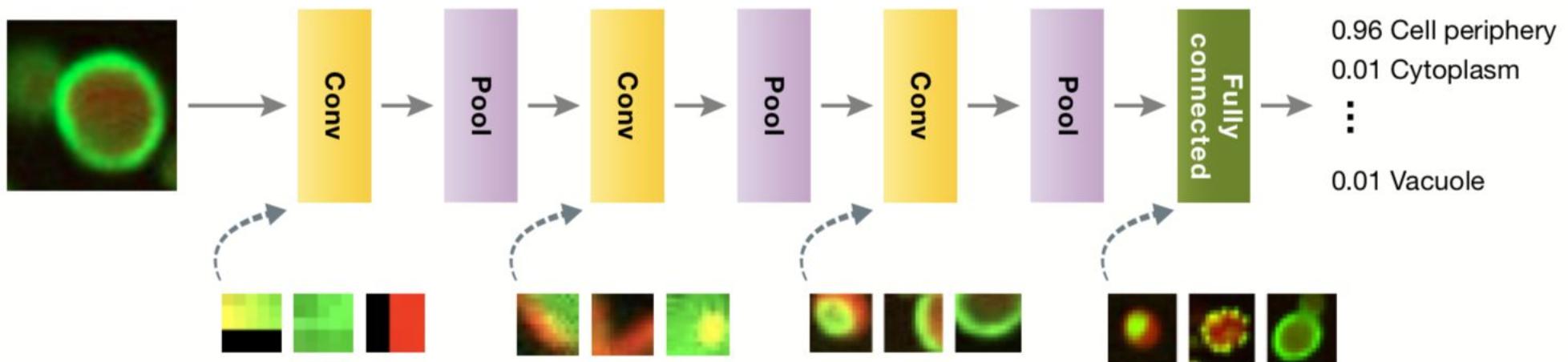


With today's computers much deeper networks are applied with great predictive power on image classification tasks. Figure modified (by Alexander Kensert) from Wikimedia Commons File:Typical cnn.png

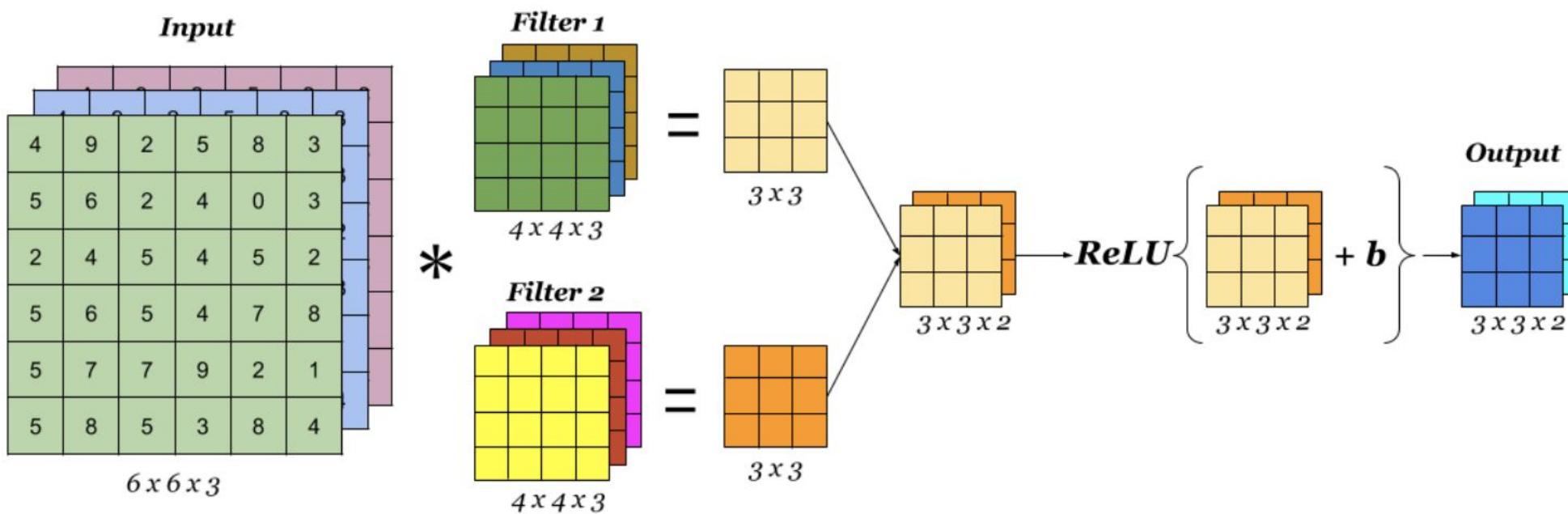
# An introduction to convolution and pooling



# What does it learn?

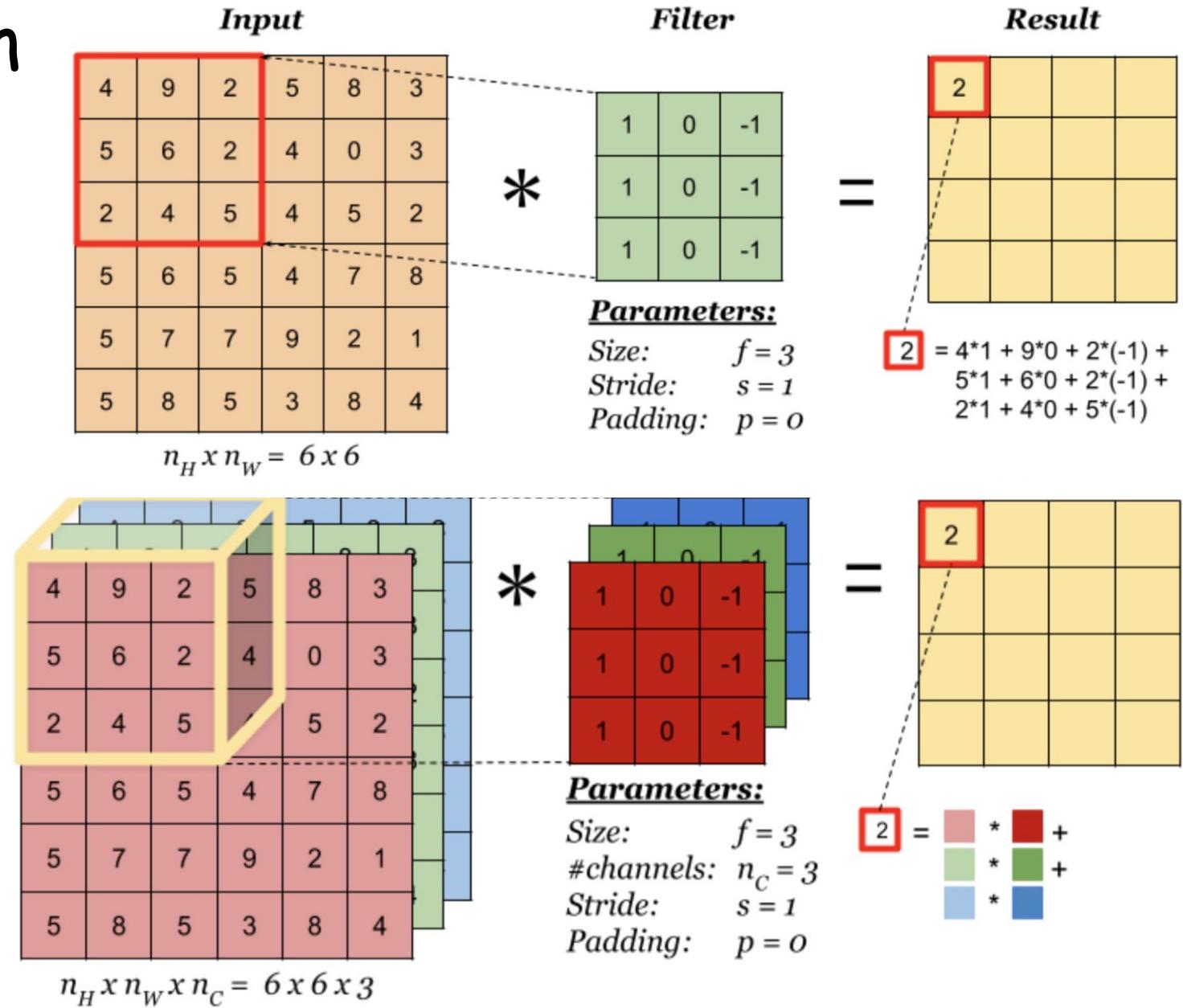


# One convolutional layer



# Convolution operation

- Filters slide over the input to perform element wise multiplication and addition to generate feature maps
- Filter values corresponds to the parameters (or **weights**)
- Hyper parameters:
  - number of filters
  - stride
  - padding



Another  
borrowed slide

## Sobel filter

Approximates the first derivatives:  $\frac{\partial}{\partial x}$  ,  $\frac{\partial}{\partial y}$



1	0	-1
2	0	-2
1	0	-1

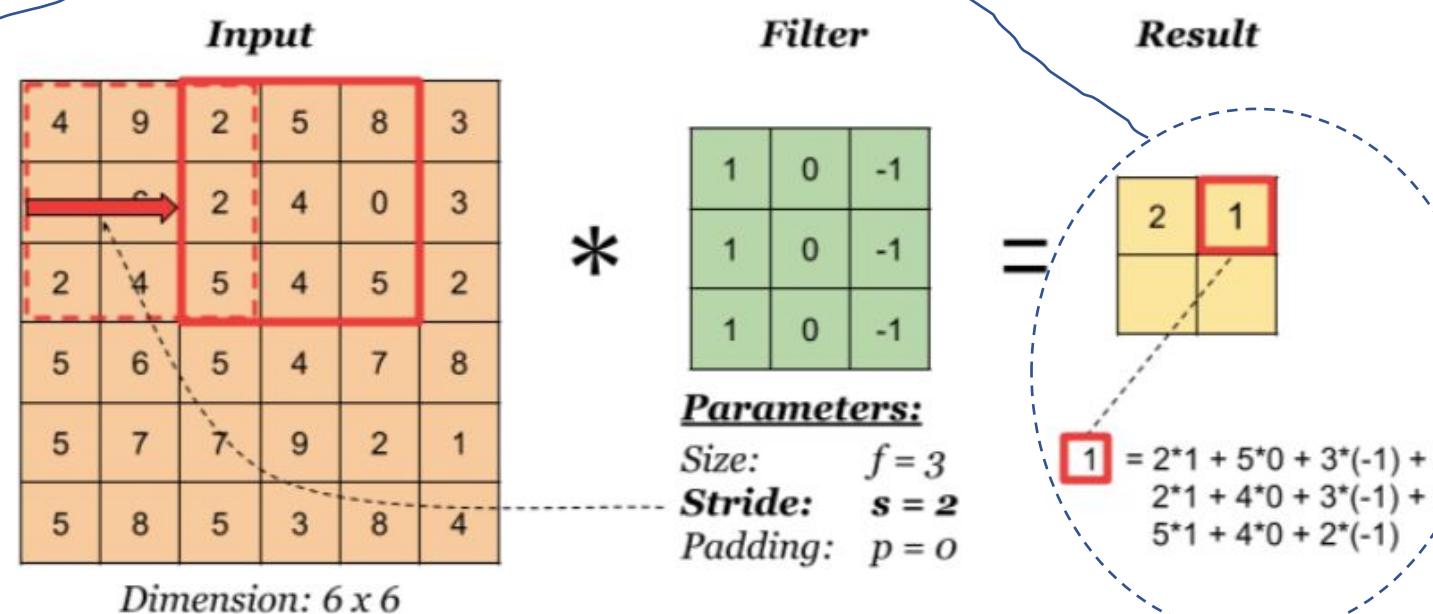
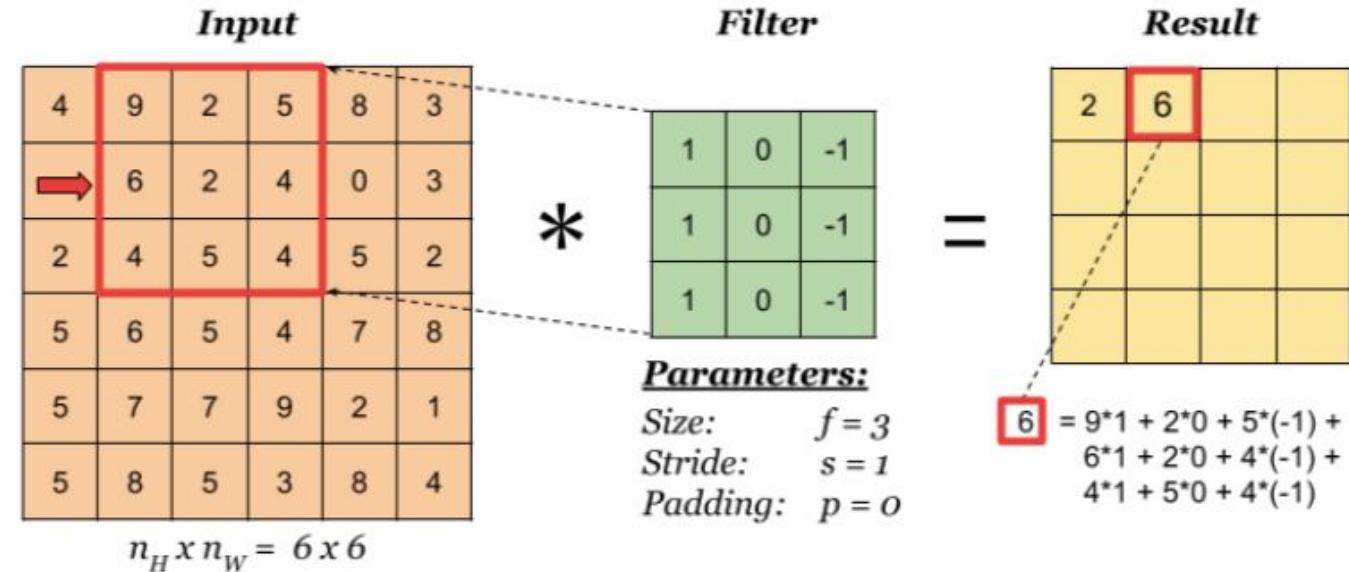
$S_x$

1	2	1
0	0	0
-1	-2	-1

$S_y$

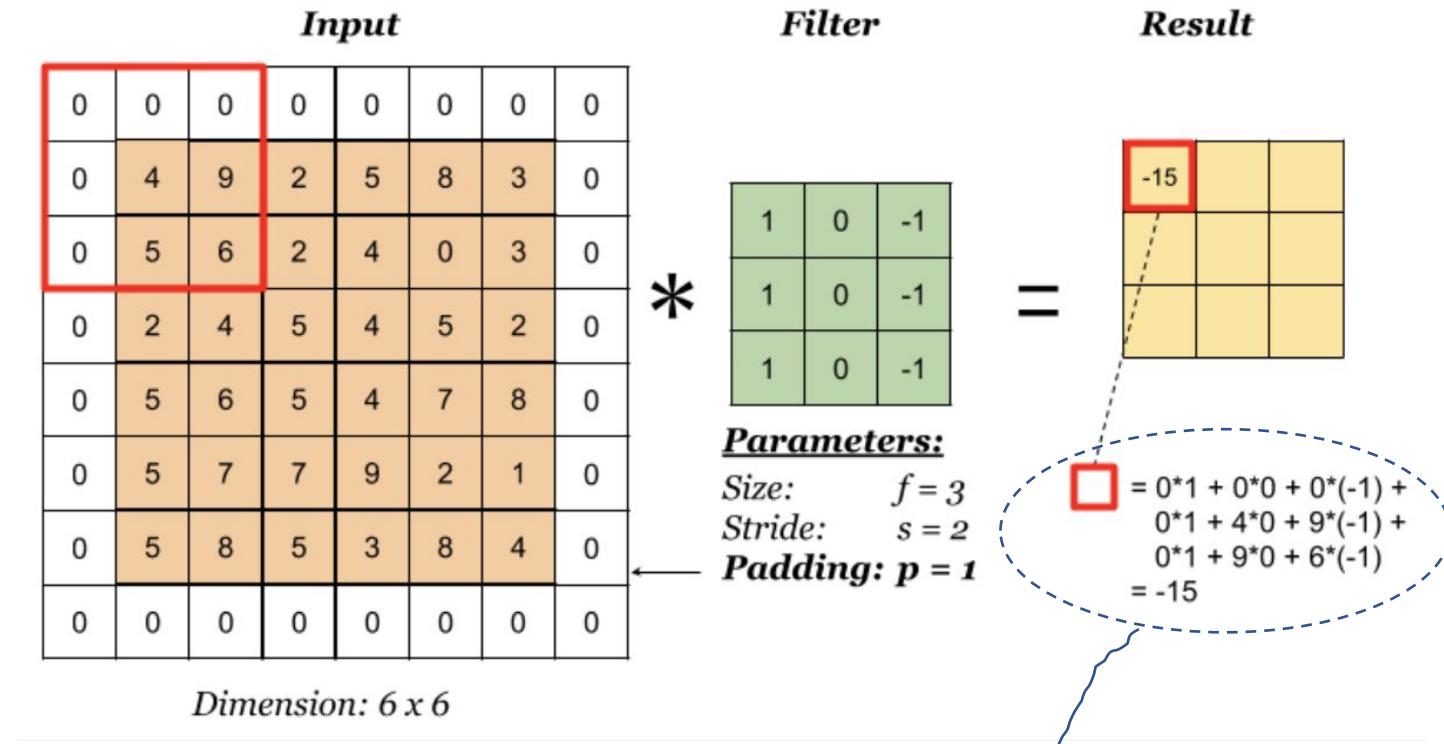
# Stride

- Stride governs how many cells the filter is moved across the input to calculate the next cell in the result
- Note the mistake made in this figure! It should have 8, 0 and 5 multiplied by (-1), not 3, 3, and 2, and the result should be -4



# Zero Padding

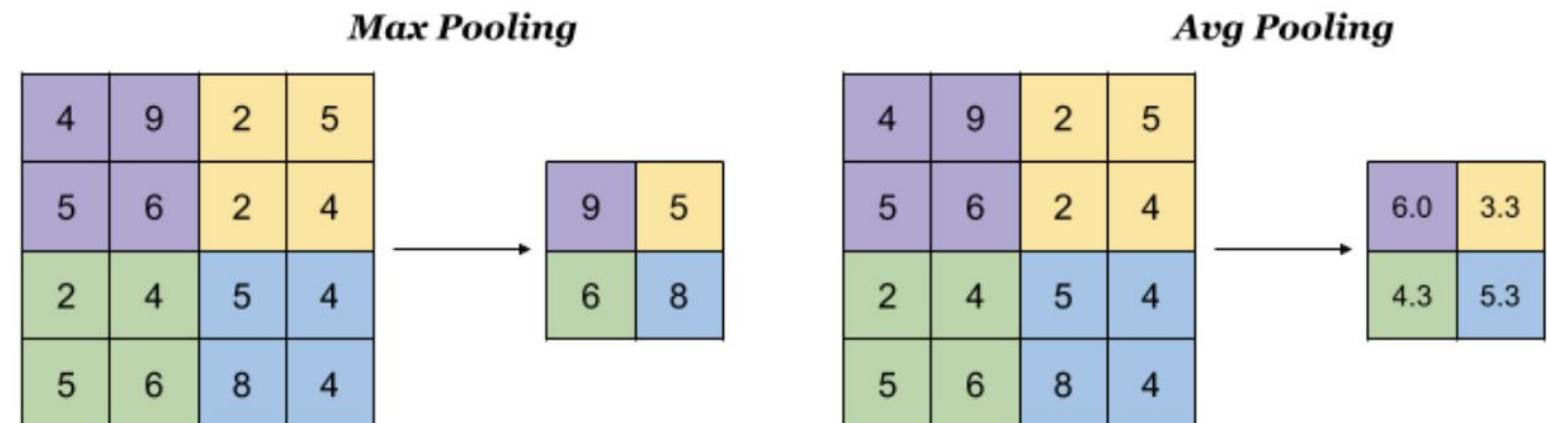
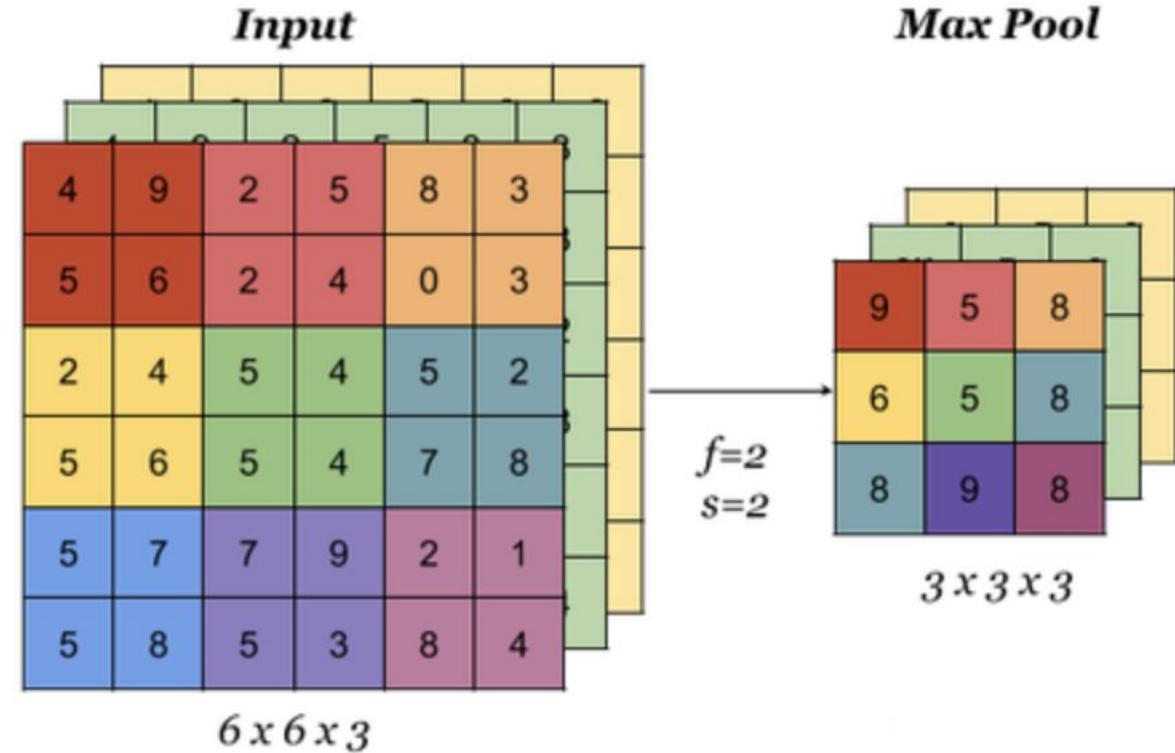
- Zero padding helps to keep the information at the borders of an image
- Without zero padding very few values at the next layer would be affected by pixels at the edges of an image



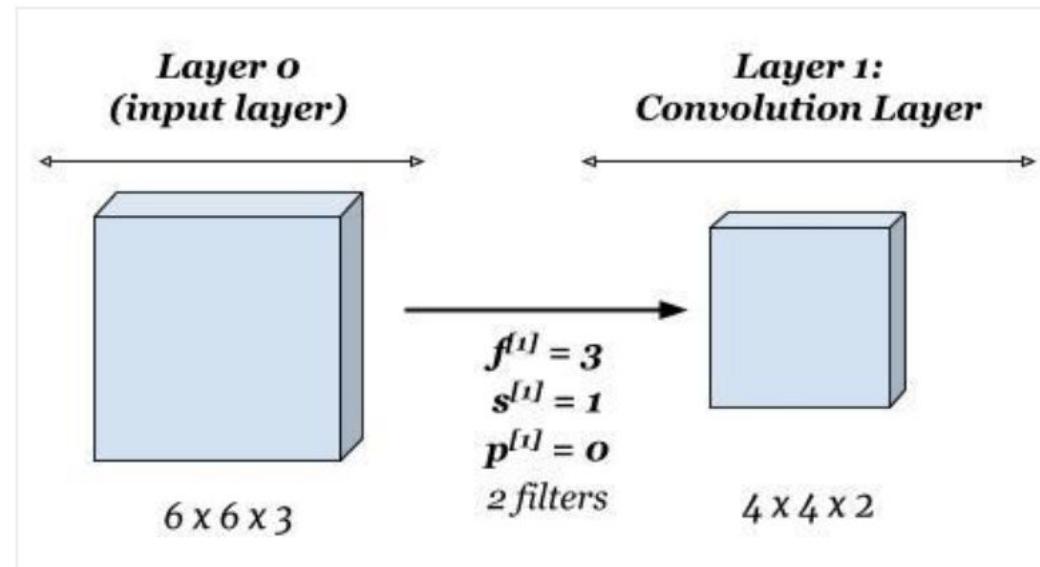
Another mistake!! Can anybody spot it?

# Pooling layers

- Pooling layers reduce the size of the representations
- Reduce the number of trainable parameters
- Give translation-invariance
- Control overfitting
- Hyper-parameters:
  - size ( $f$ )
  - stride ( $s$ )
  - type (max or avg)



# Shorthand representation



# Notes

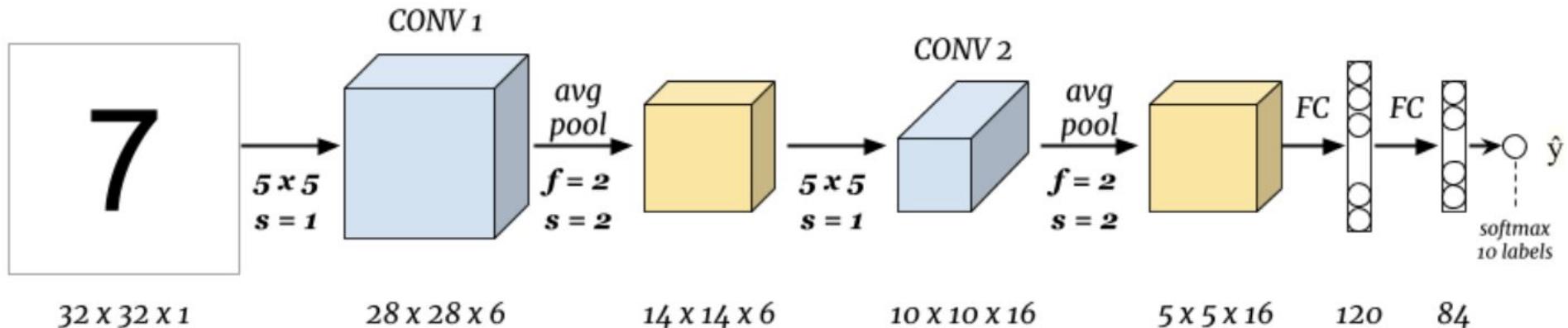
- Filter weights are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible. We will use train CNNs in the lab but unfortunately only on CPUs
- Implemented as tensor multiplications/additions.  
[Hence the name "TensorFlow"  
- we will use TensorFlow, via Keras, in the lab]

my notes from a Coursera course showing why neural networks need matrix algebra

# Famous CNNs

1. LeNet-5
2. AlexNet
3. VGG-16
4. ResNet
5. Inception -> GoogLeNet

# LeNet-5



# AlexNet

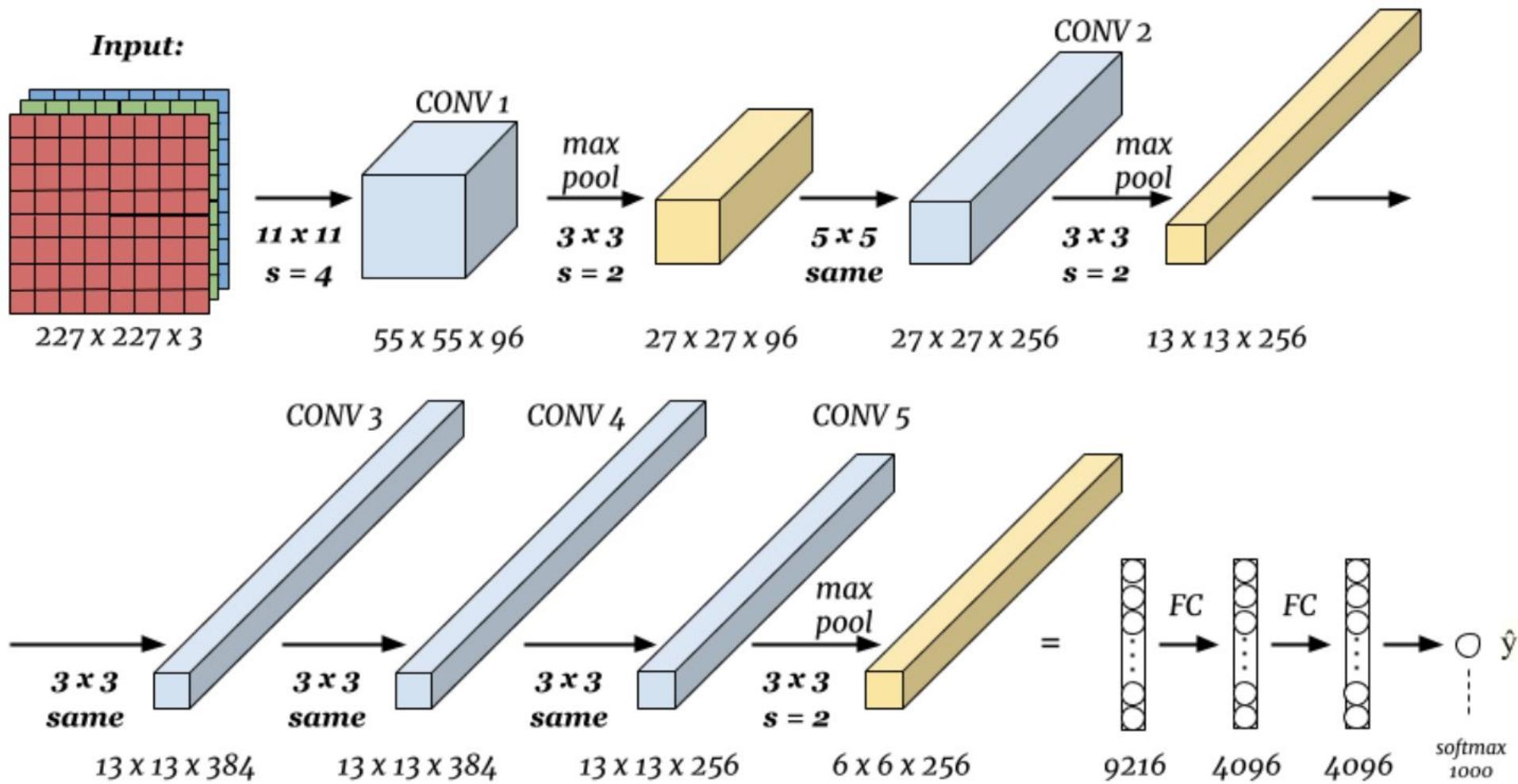


image from <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>

# VGG-16

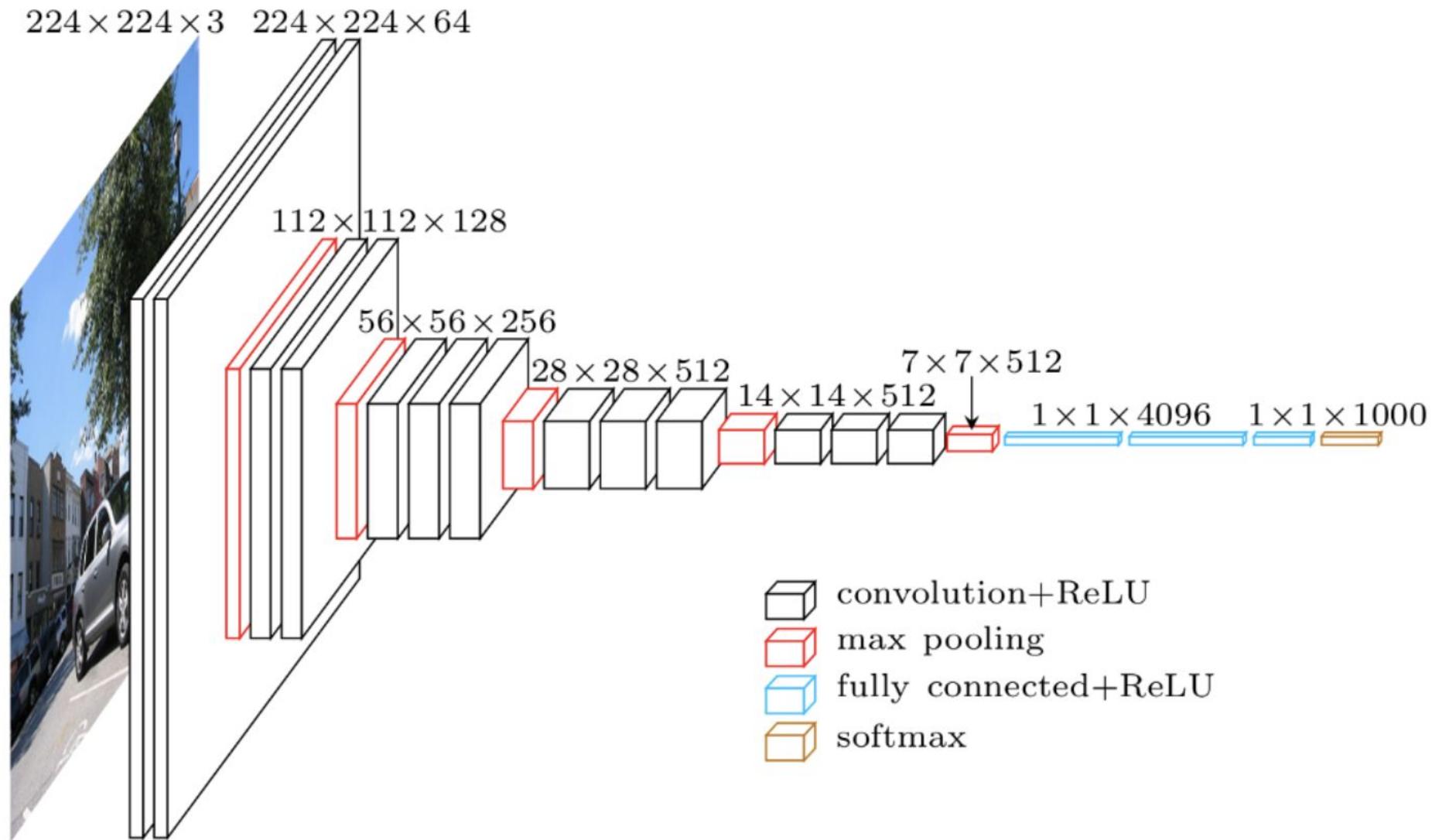


image from <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>

# ResNet

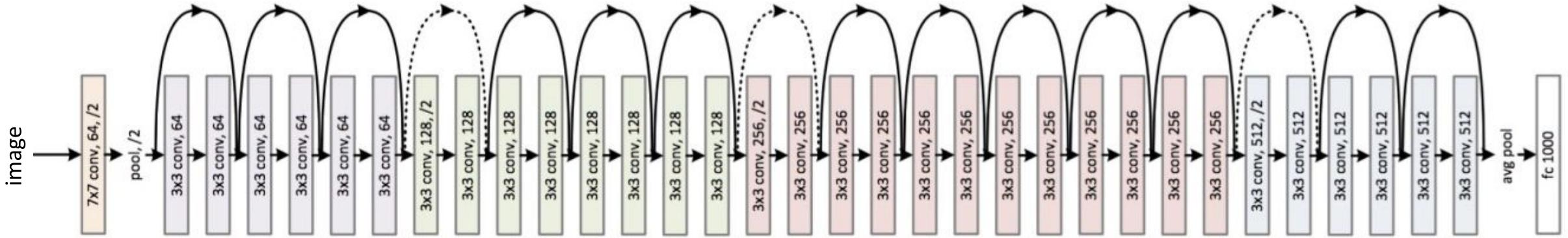
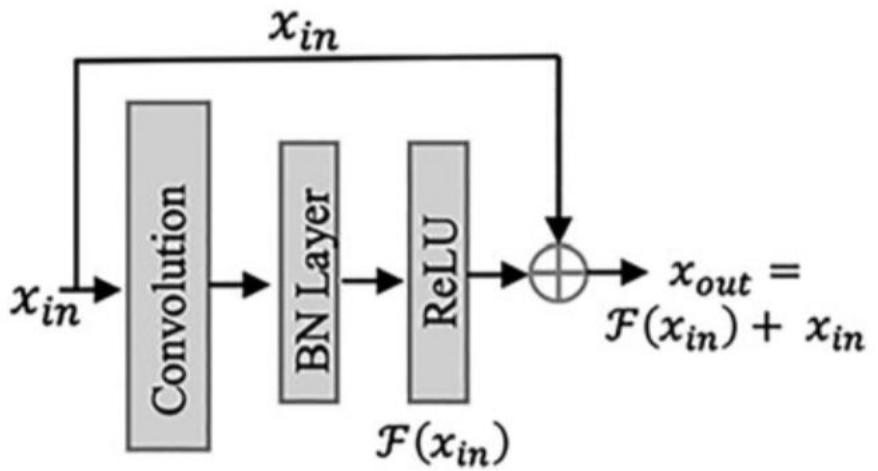


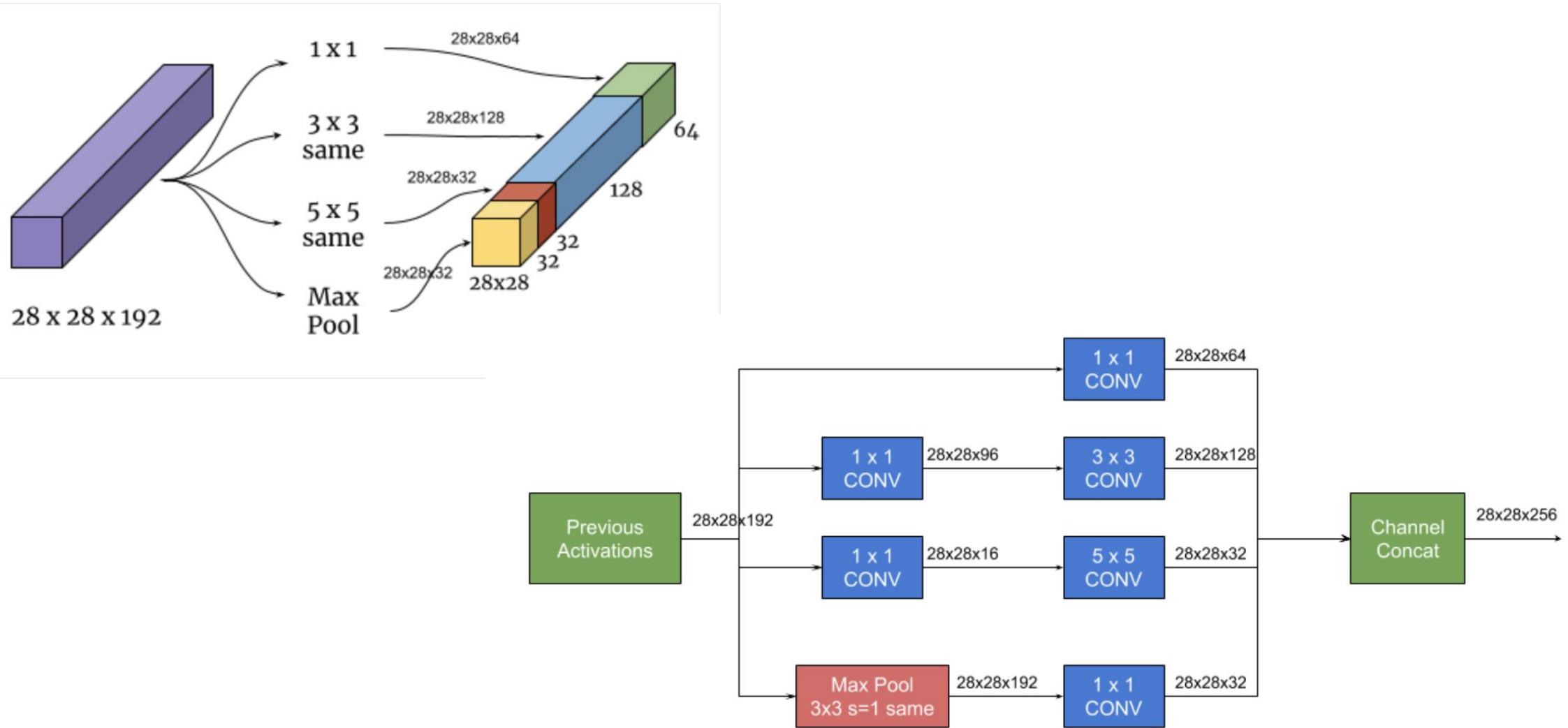
image from <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>

## Skip/residual connections



An example of a skip connection, connecting the input with the output of one convolution block (consisting of a convolutional layer, a batch normalization layer, and a ReLU activation function) (from Gupta et al, 2019)

# Inception block



# GoogLeNet

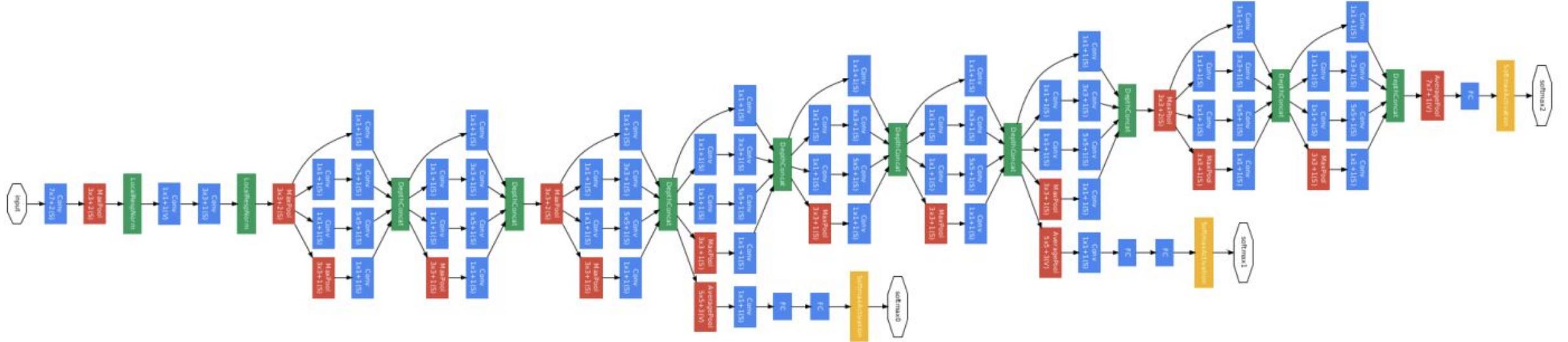
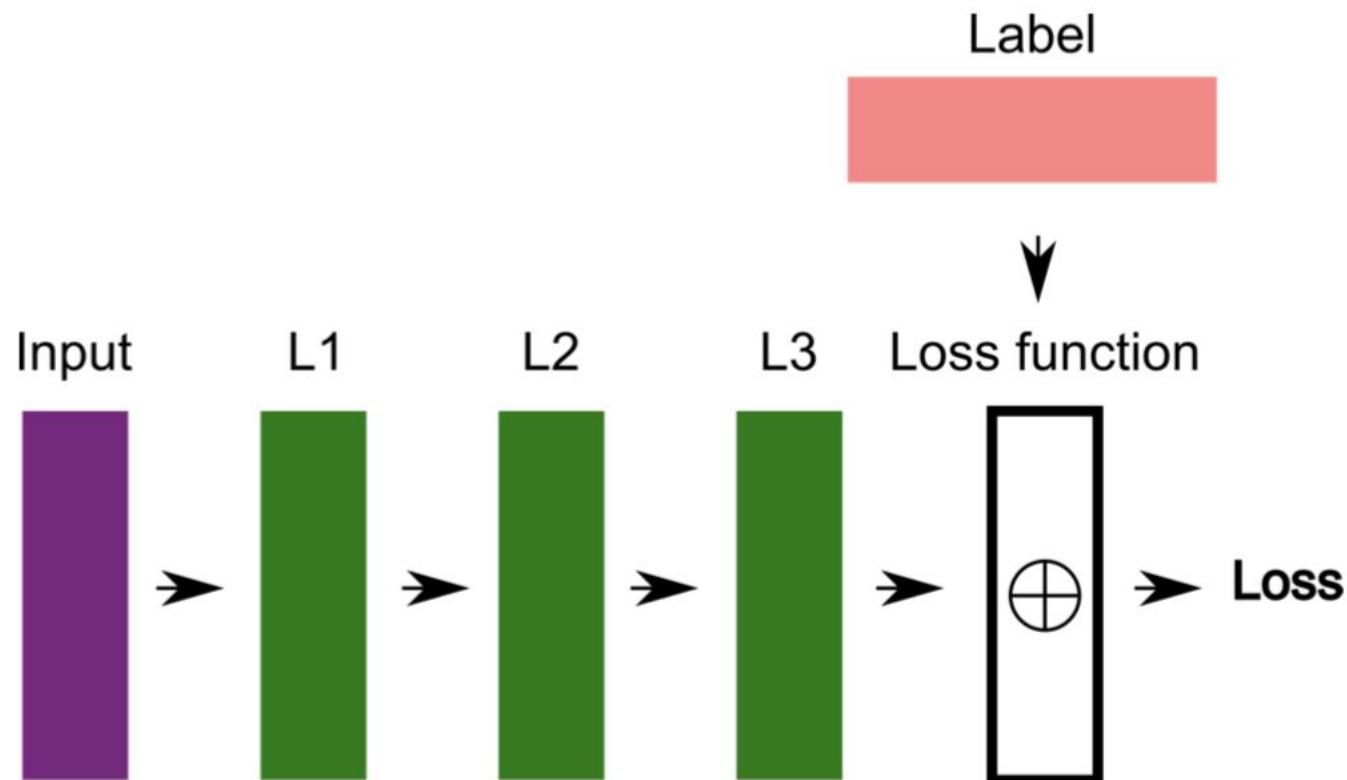


image from <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>

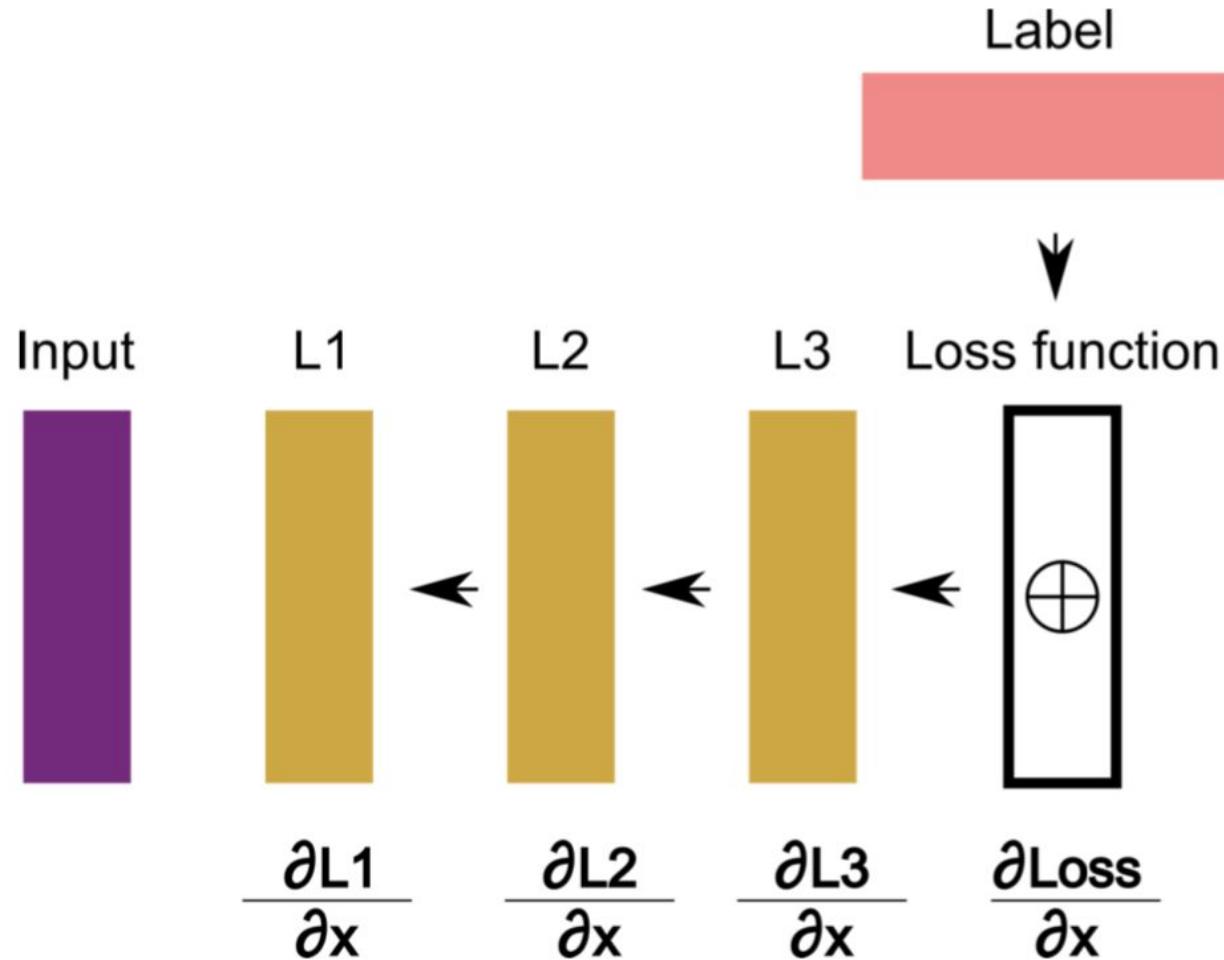
# How does the network learn?

- Learns from its mistakes using a **loss function**
- Contains hundreds of parameters/variables
- Find the effect of each parameter in making mistakes using **back propagation**
- Increase/decrease the parameter values so as to make less mistakes using **Gradient Descent**
- Do all the above several times **iteratively**

# Forward pass



# Backward pass



# Deep neural network working

- Training
- Loss function
- Forward pass
- Backward pass/Back propagation
- Gradient descent optimization

# Training

- Data partitioning - training, validation, testing
- Preprocessing
- Network architecture
- Hardware
- Weight initialization

# Loss function

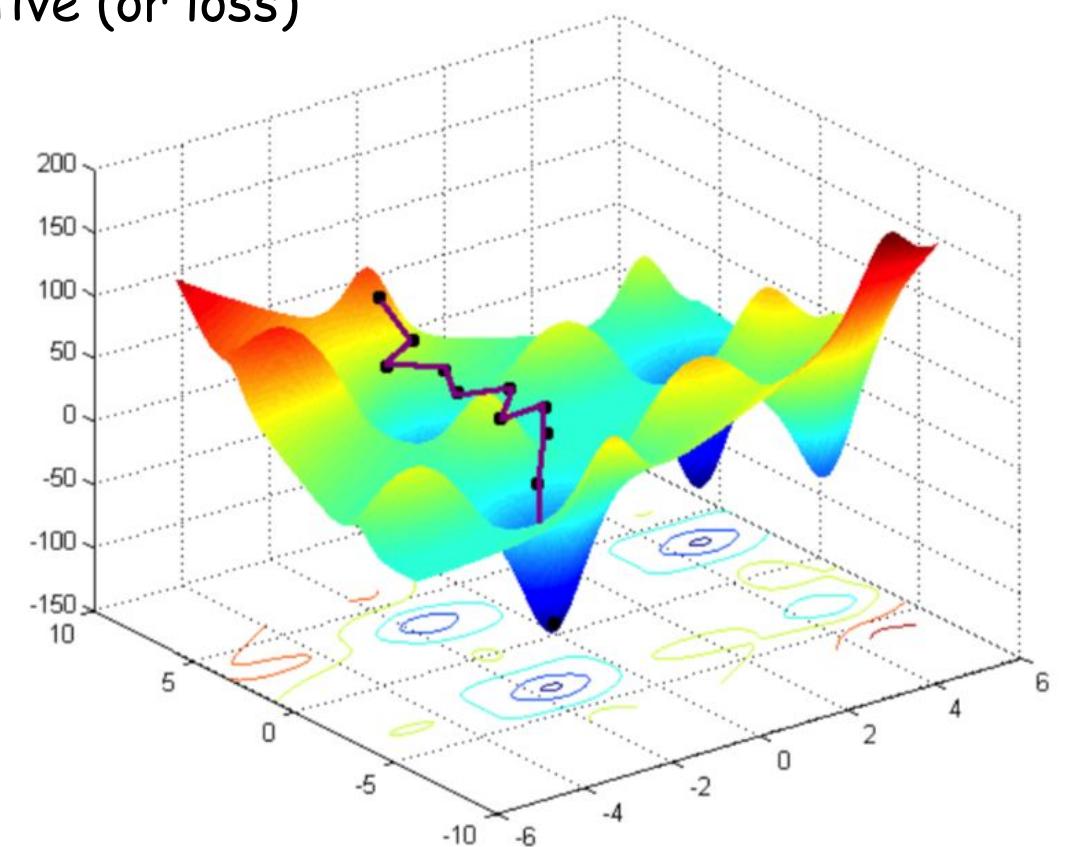
- Mean square error loss
- Softmax cross-entropy loss
- etc

# Back propagation

- Follow chain rule
- Find the analytical gradient of each components
- Find derivative of activation functions
- Total gradient is the product of the gradient at that node multiplied with the gradient coming to the node
- If connected to multiple nodes, add the gradients
- For ReLU activations gradient pass to positive outputs only

# Gradient descent

- Gradient descent is a way to minimize an objective (or loss) function,  $J(\theta)$
- Parameters  $\theta$
- Gradient of the objective function  $\nabla_{\theta} J(\theta)$
- Update in the opposite direction of gradient
- Learning rate  $\eta$
- Its variants:
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent



[http://www.phoenix-int.com/software/benchmark  
report/bird.php](http://www.phoenix-int.com/software/benchmark_report/bird.php)

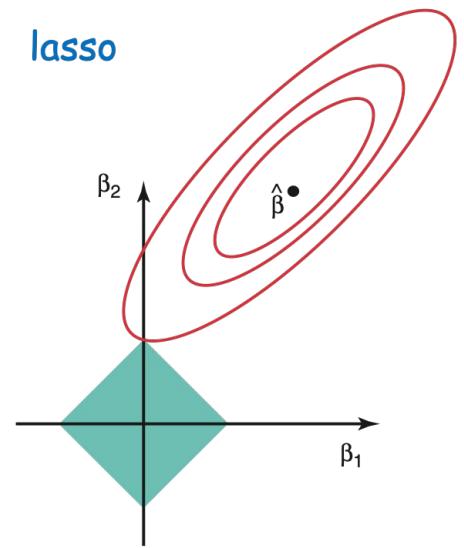
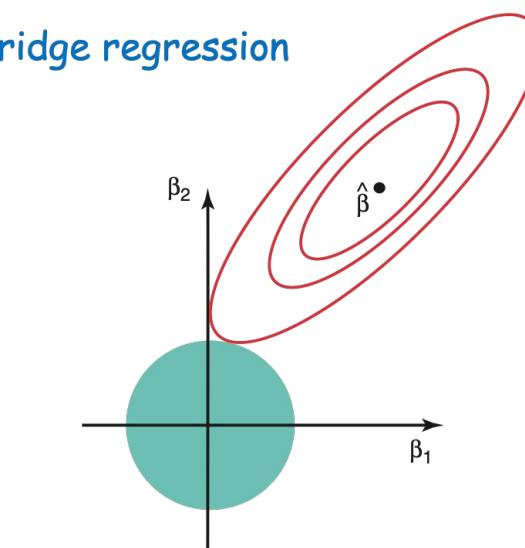
# Regularization

To reduce over-fitting to training data:

- Data augmentation
- Drop out
- Batch normalization
- $L2, L1$  regularization of weights

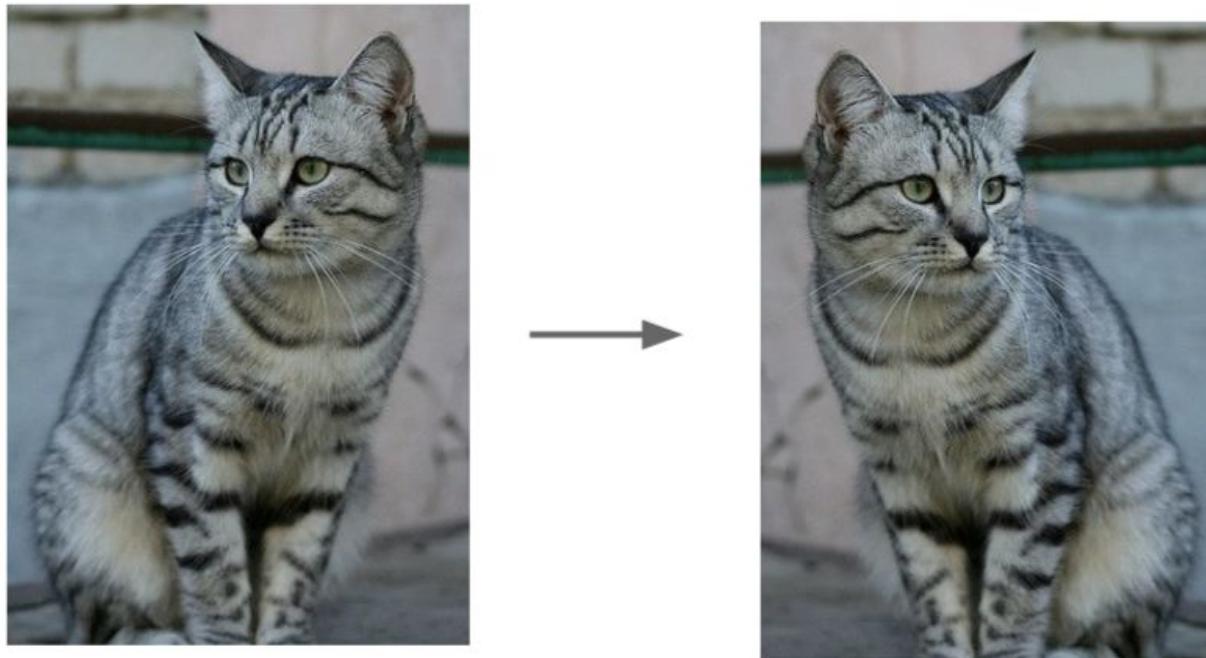


recall from Ola's slides:



# Data Augmentation

## Horizontal Flips



# Data Augmentation

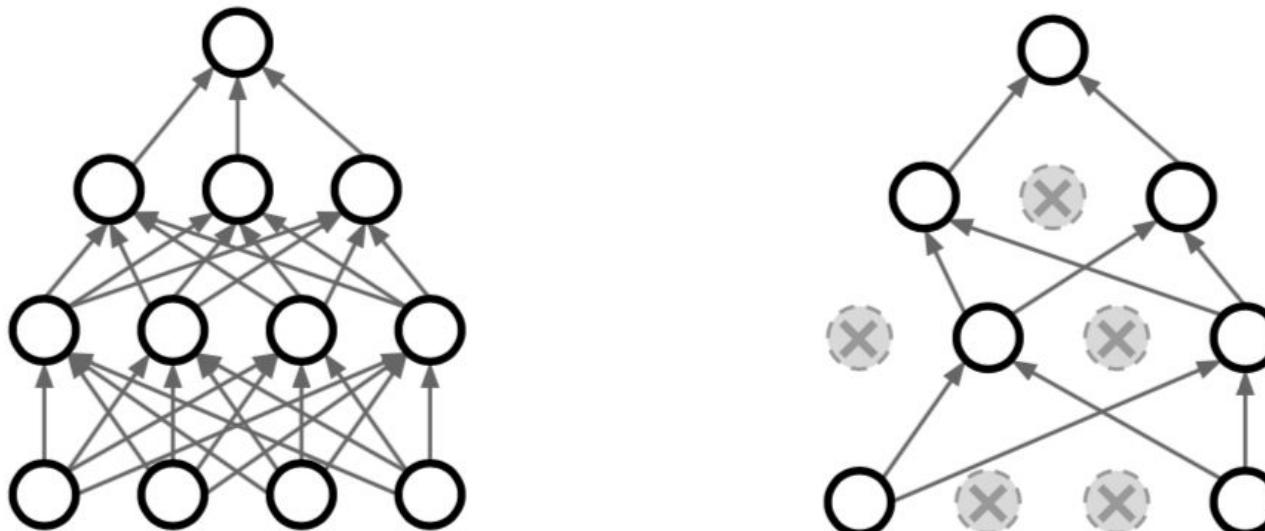
## Get creative for your problem!

Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, ... (go crazy)

# Regularization: Dropout

In each forward pass, randomly set some neurons to zero  
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

# The problem of insufficient dataset size

To fit deep CNNs to data you need lots of data! More than we often have for cell image data 😞

solution = **transfer learning**

- use weights from a pretrained model with a similar architecture on a large dataset (e.g. ImageNet)
- either:
  - freeze the initial layers (basic image feature extractor) with the pretrained weights. Then train the weights in the final layer(s);
  - use the pretrained weights as good starting values and train the entire network

# Additional hyper parameters

- Learning rate
- Learning rate decay
- Momentum
- Network depth
- Network width
- Mini batch size

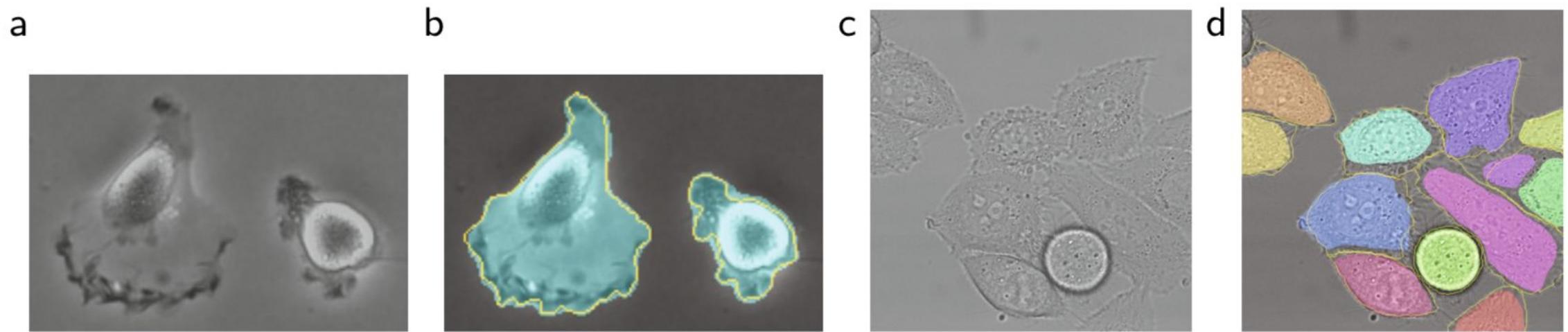
# Deep learning tools

- Tensorflow
- Keras
- 
- Caffe
- Torch
- PyTorch
- Theano
- Neon
- Chainer

# Deep learning (Keras) workflow

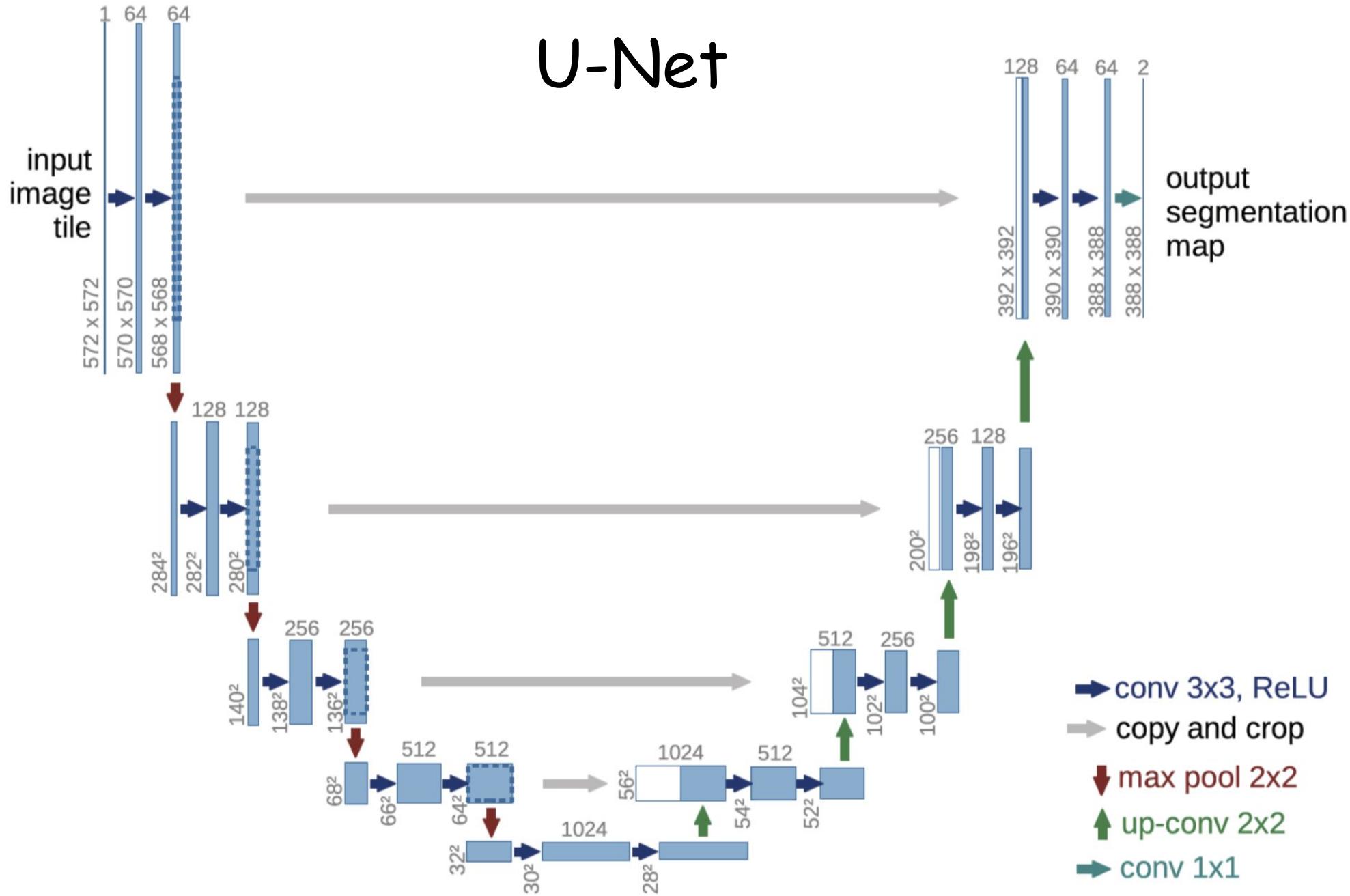
1. read in required packages
2. read in data - inspect/plot
3. standardize or normalize data & one-hot encoding for categorical responses
4. train-validation-test split (or use cross validation)
5. define & summarize model
6. compile model (choose loss function, optimizer and metrics to record)
7. fit model (train for several epochs)
8. evaluate model (accuracy on test set, inspect confusion matrices etc)

# Bonus material: Cell segmentation with U-Net

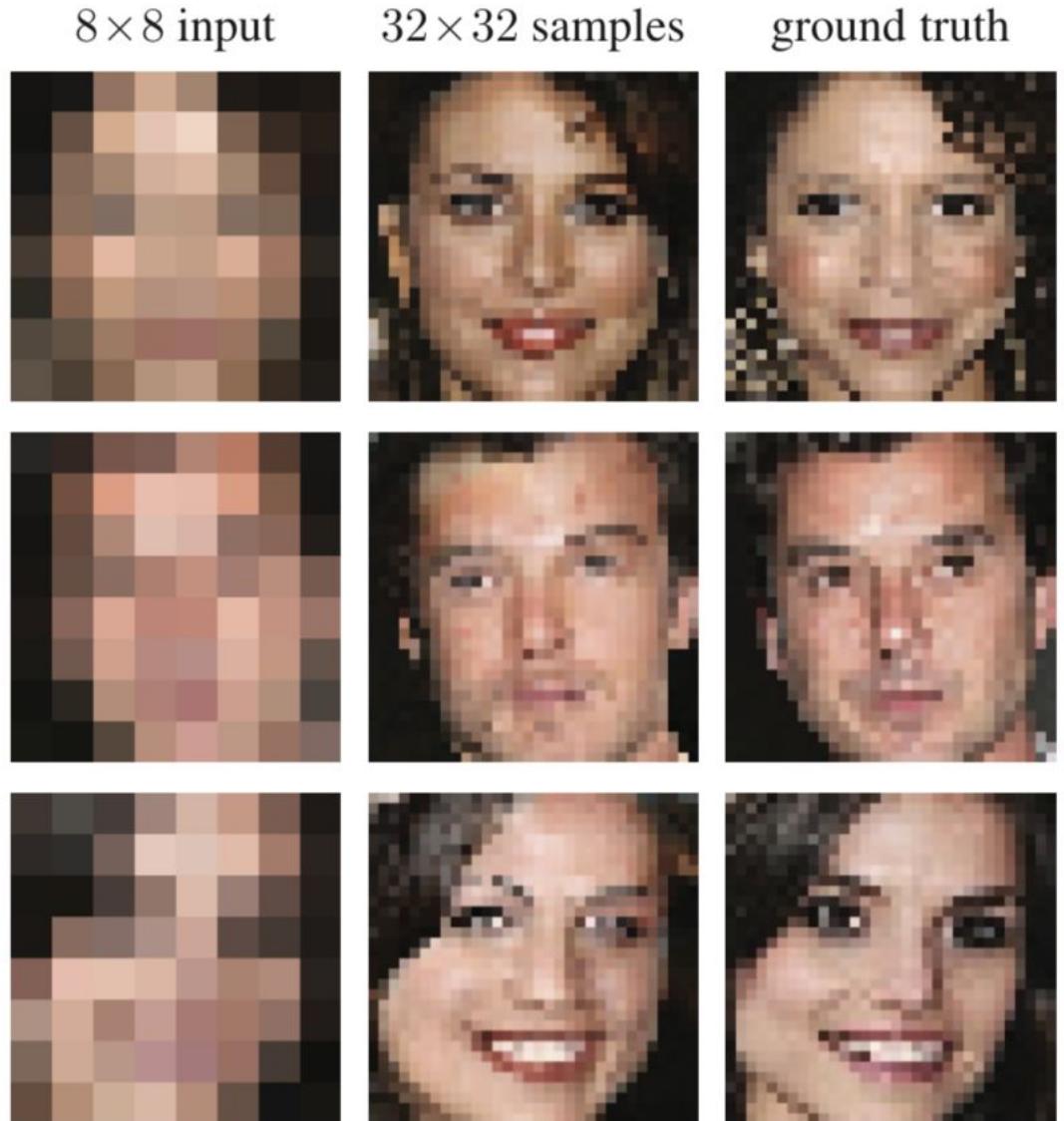
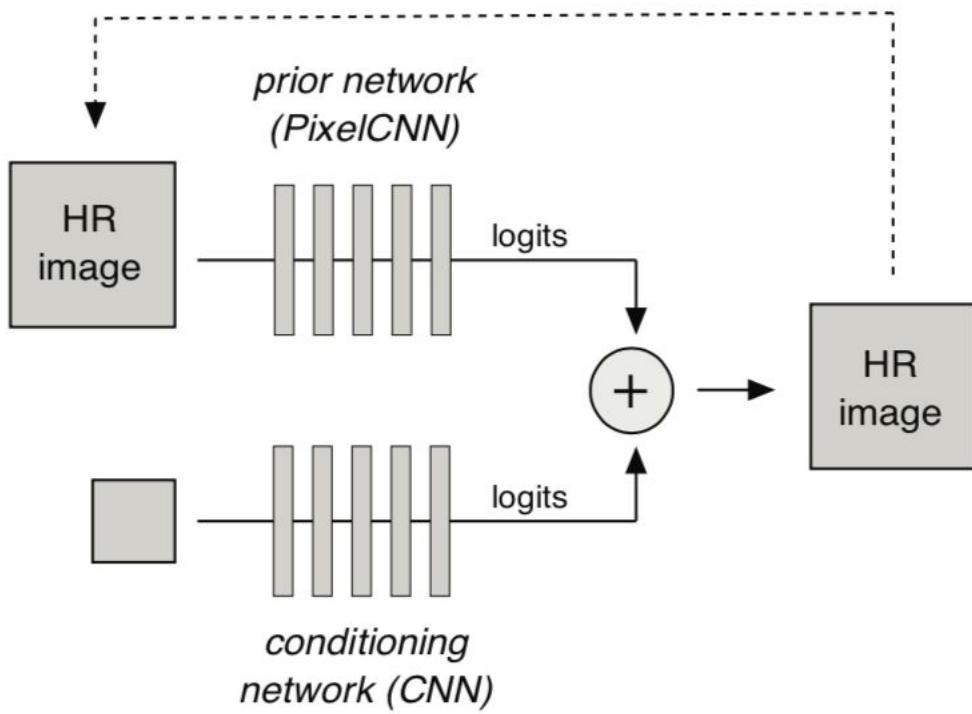


**Fig. 4.** Result on the ISBI cell tracking challenge. **(a)** part of an input image of the “PhC-U373” data set. **(b)** Segmentation result (cyan mask) with manual ground truth (yellow border) **(c)** input image of the “DIC-HeLa” data set. **(d)** Segmentation result (random colored masks) with manual ground truth (yellow border).

# U-Net



# Super resolution



Dahl, R., Norouzi, M., and Shlens, J. (2017). Pixel Recursive Super Resolution. In 2017 IEEE International Conference on Computer Vision (ICCV), pp. 5449-5458.

# Other cool deep learning stuff

- Recurrent Neural Networks (RNNs) & LSTMs
- Autoencoders and variational autoencoders
- Generative Adversarial Networks (GANs)
- etc.

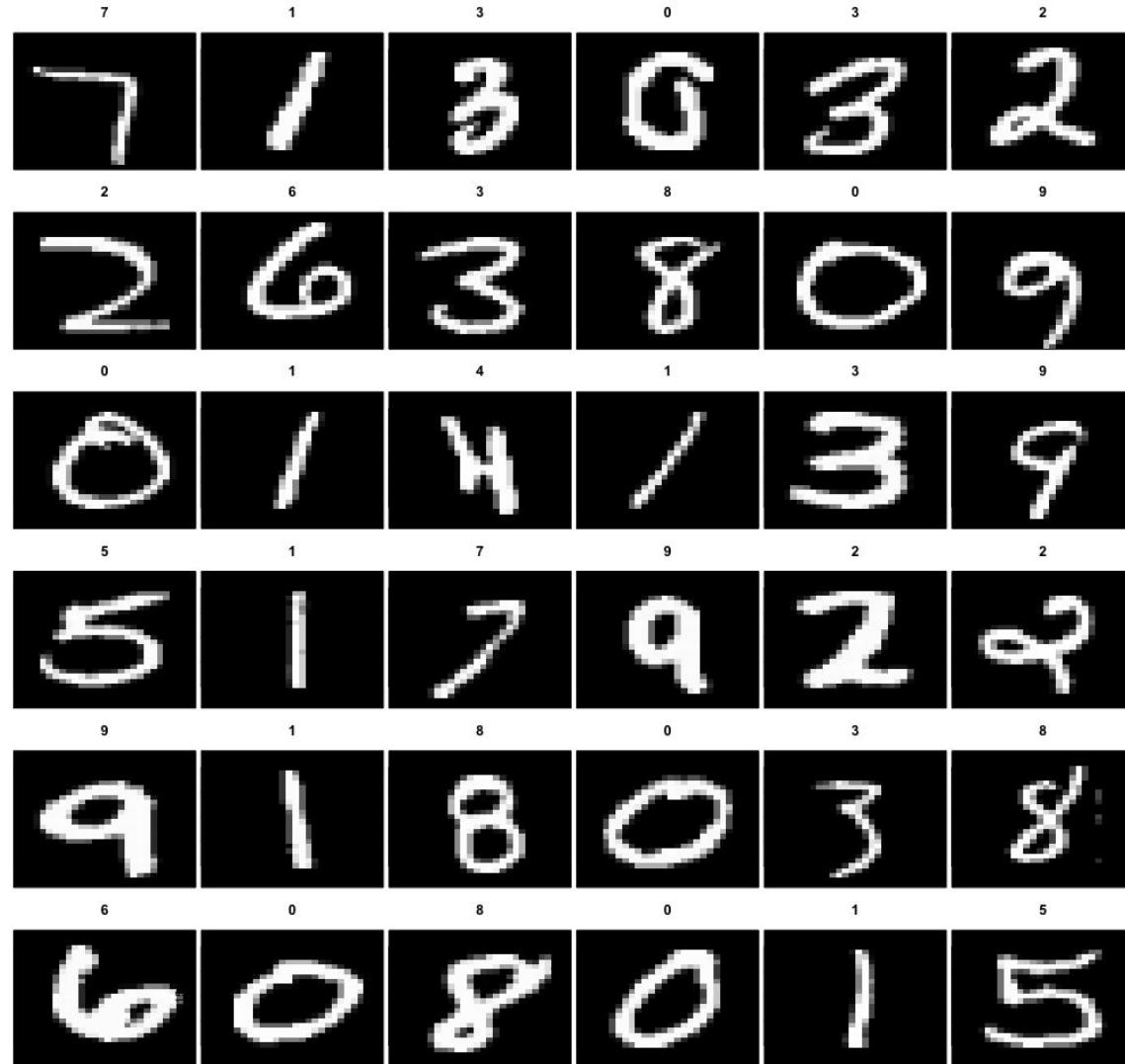
# Details about tomorrow's computer lab

## Warm-up exercise:

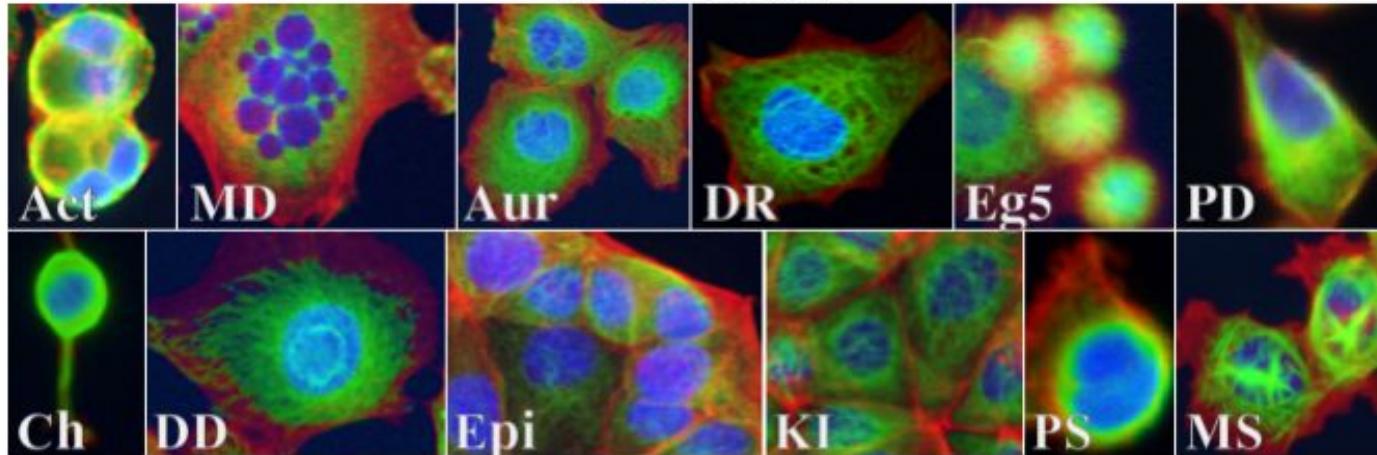
Train a simple CNN on the [MNIST](#) data

- [MNIST](#) = database of handwritten digits
- 60,000 training images and 10,000 test images
- Digits size-normalized and centered in a  $28 \times 28$  pixel image.
- A classic benchmark dataset for machine learning!
- Comes pre-installed in Keras!

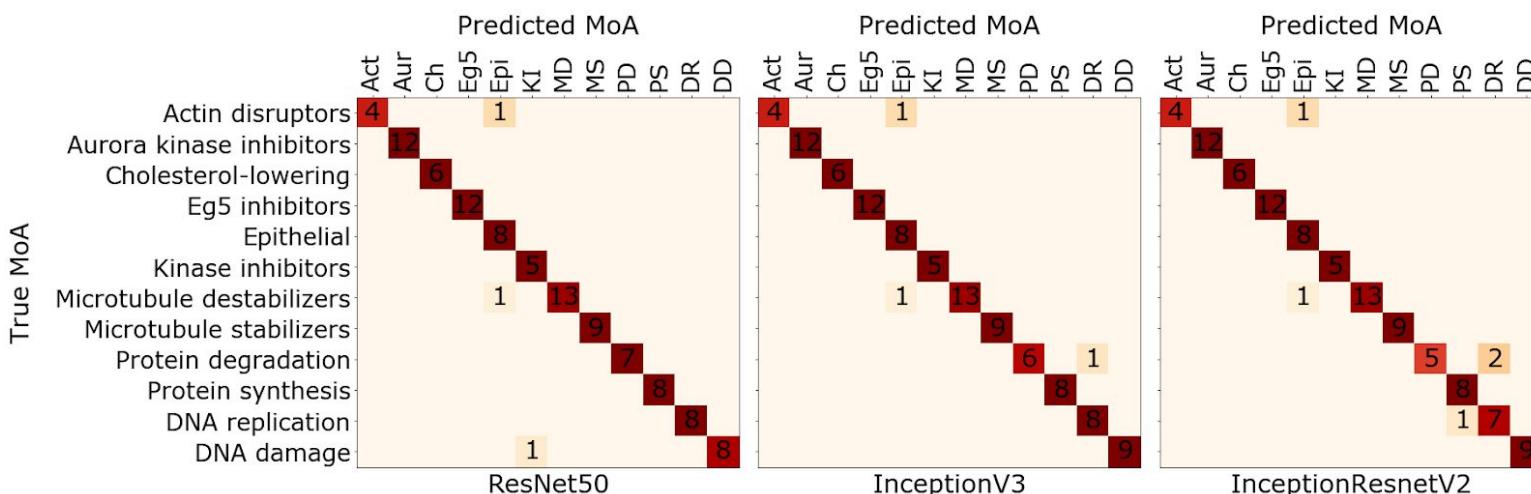
Go to



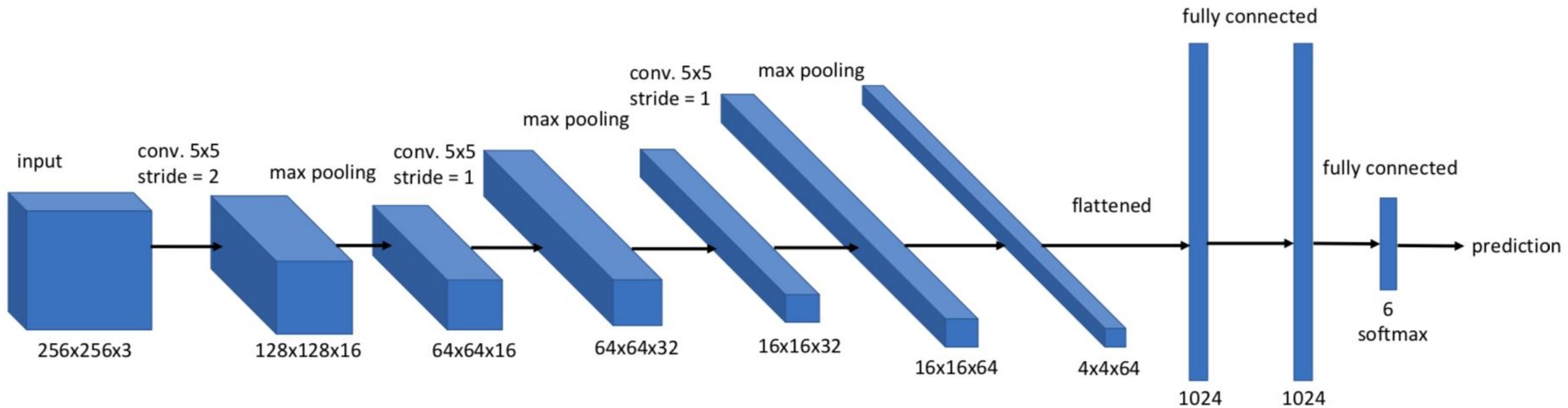
# Main lab exercise: Classification of cell morphological changes with CNNs

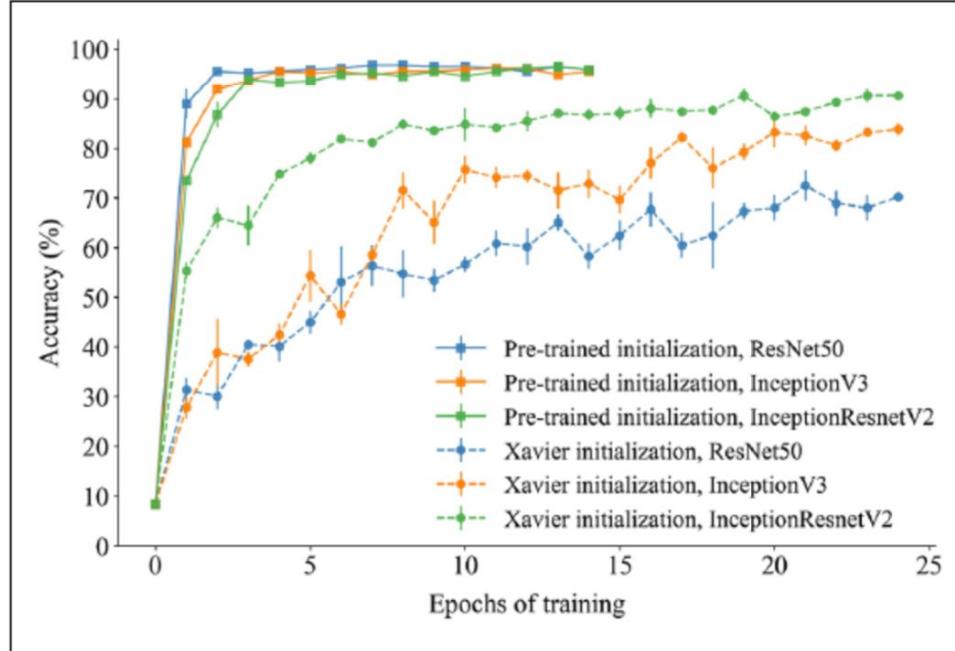


Kensert, A., Harrison, P.J., and Spjuth, O. (2019). Transfer Learning with Deep Convolutional Neural Networks for Classifying Cellular Morphological Changes. SLAS DISCOVERY: Advancing Life Sciences R&D

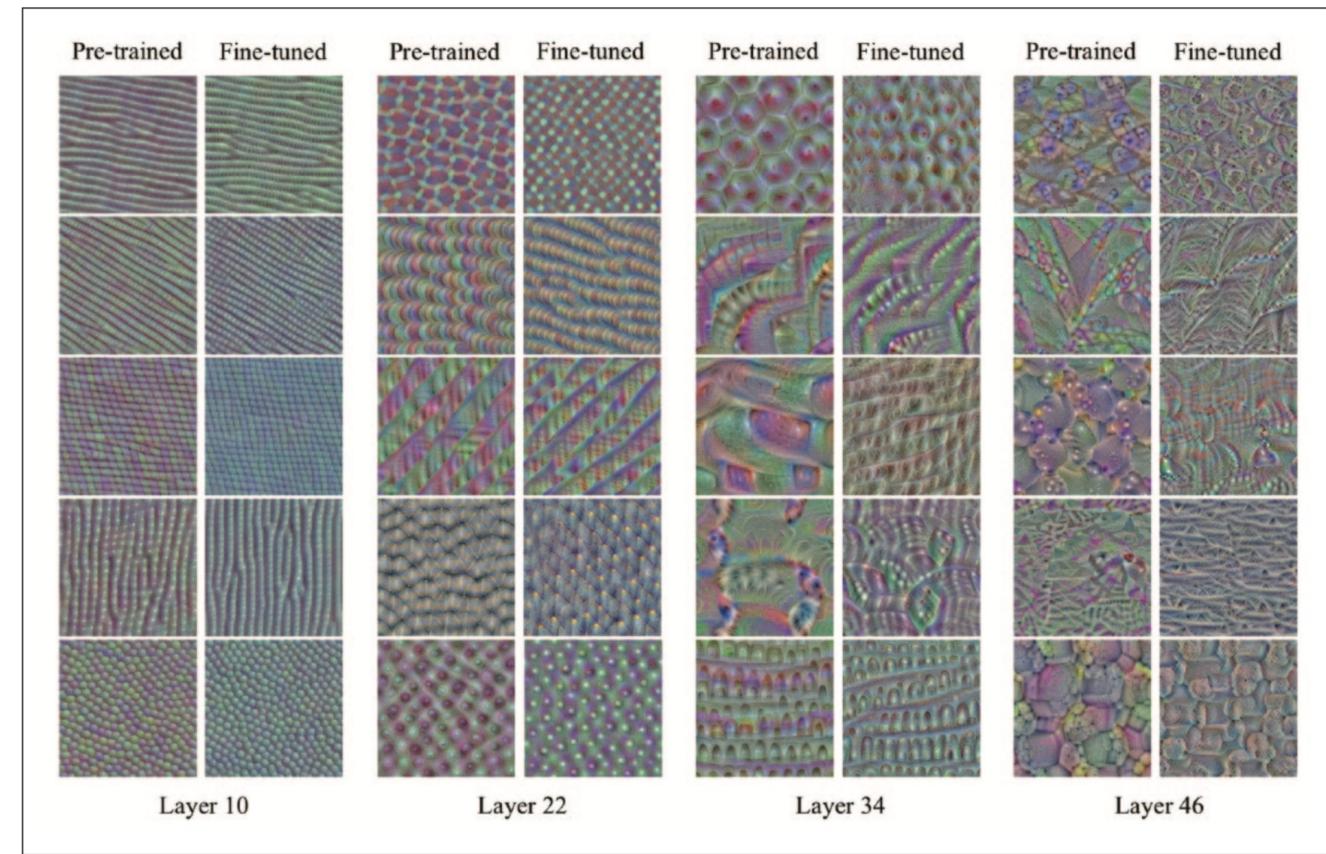
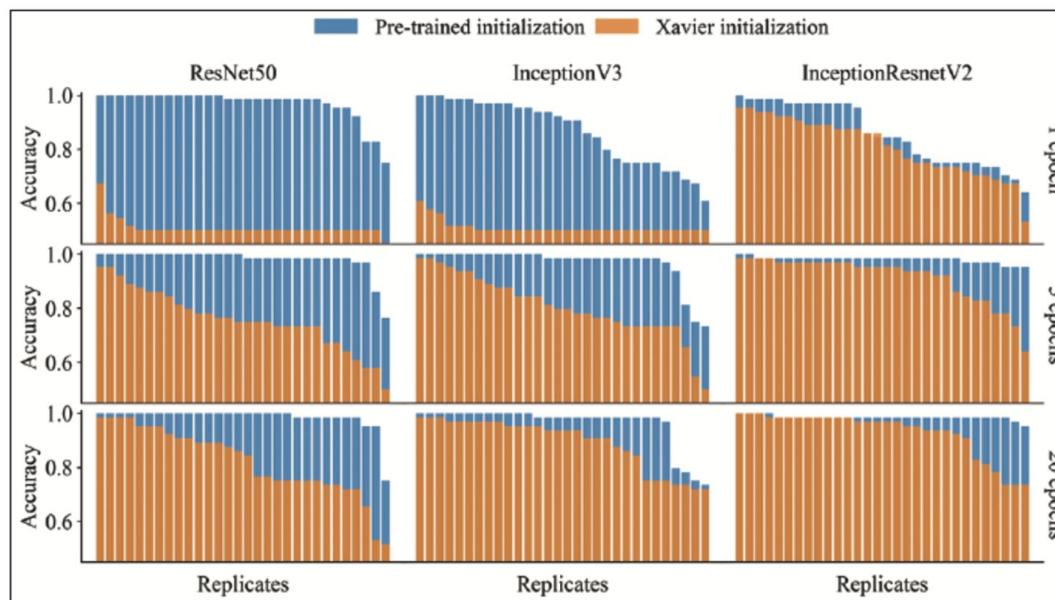


*LeNet inspired CNN that you will define and fit  
to the MoA dataset during the lab*





More figures from Kensert, A., Harrison, P.J., and Spjuth, O. (2019).



THE END