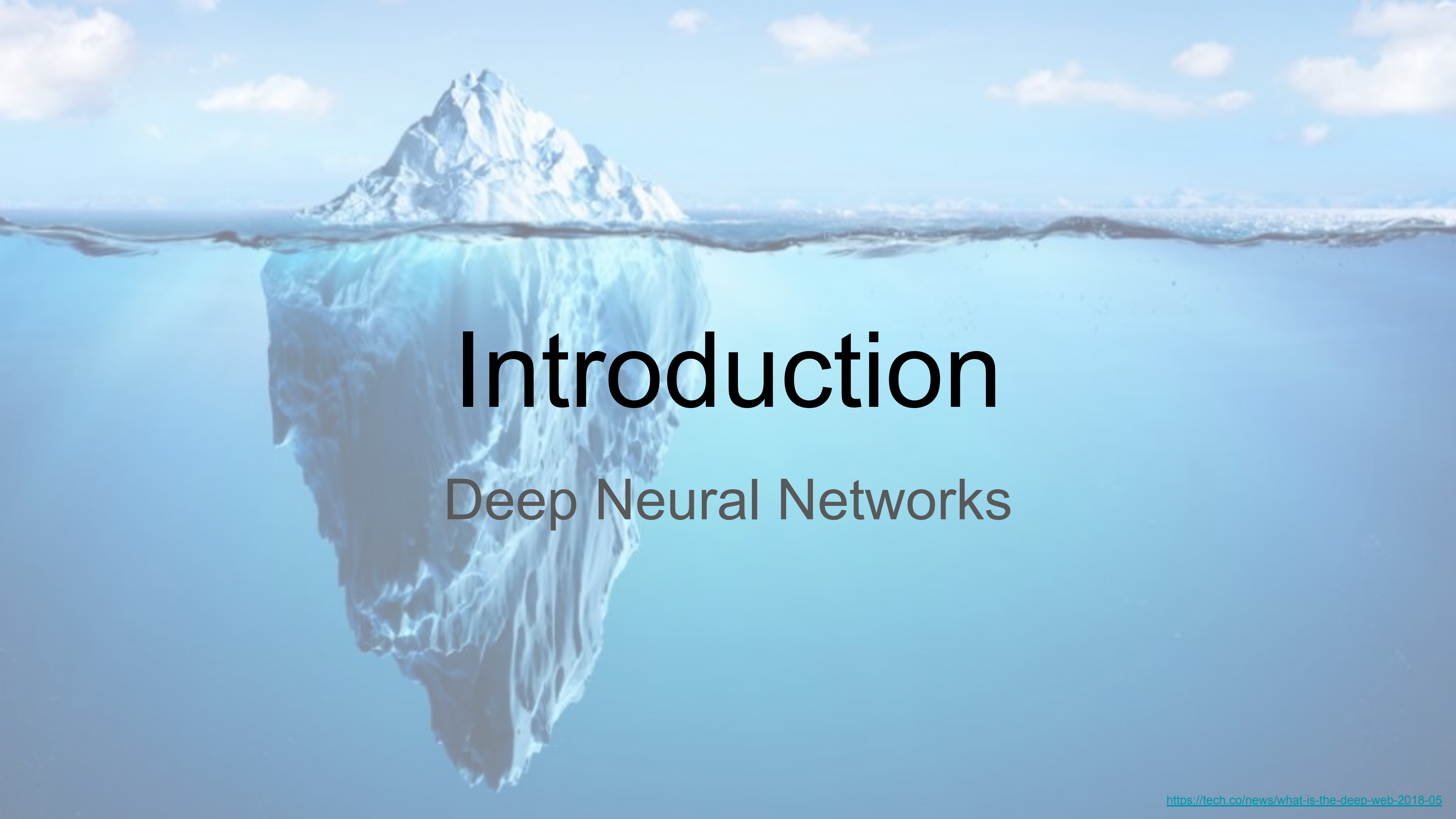


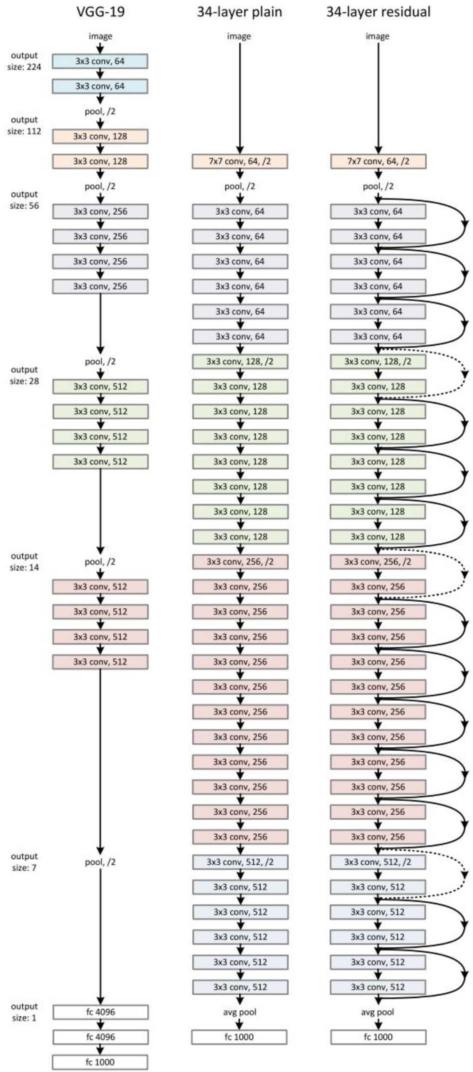
Introduction

Deep Neural Networks

https://tech.co/news/what-is-the-deep-web-2018-05

Contents

- Intro to computer vision, ML, and DL
- Basic classification
- Simple, feedforward nets
- Getting the gradient of a net - backprop
- Training a net - SGD and others
- Making a net behave - regularization
- Convnets for image data (and other types)
- How reconstruction enables compression - autoencoders
- Point cloud processing
- Other funky stuff



https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

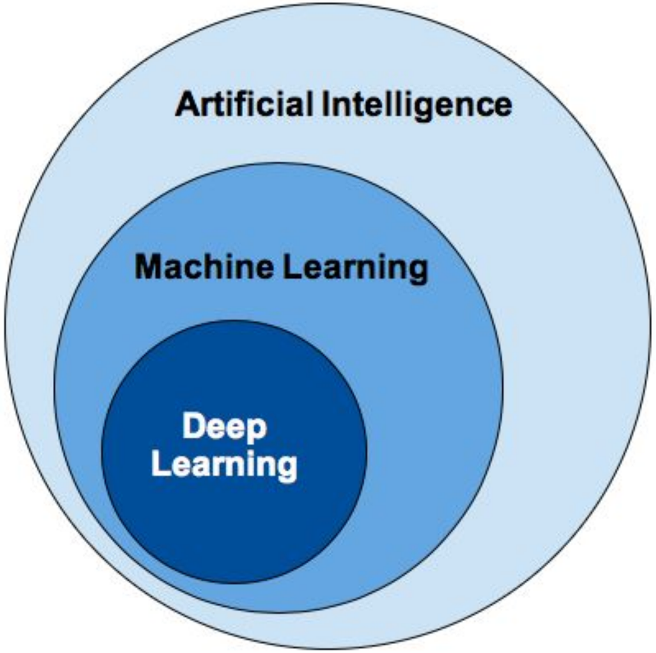
Format

- Weekly 4-hour modules
- Theory
- Exercises in Python
    - More freedom towards the end

Deep learning

Context

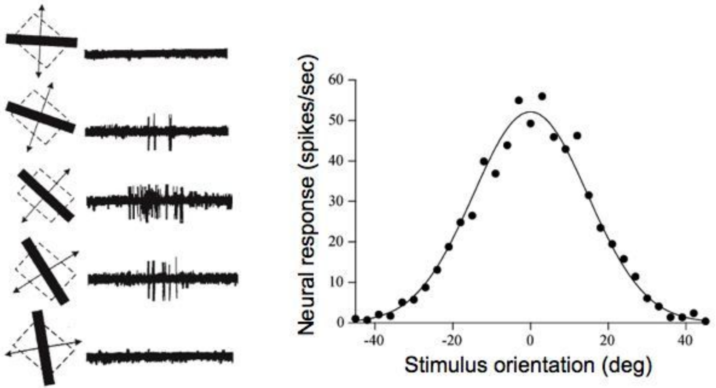
- AI
  - The concept of “human-like” machines
  - Many, many subdisciplines
- ML
  - Focus on **data**
  - Instead of being designed, the machine learns a **model** from the data
- DL
  - A specific class of models to use in ML
  - Layered architecture of computational units
  - High-capacity models possible, called ANNs or DNNs



https://www.sumologic.com/blog/machine-learning-deep-learning

Discovery of S+C cells

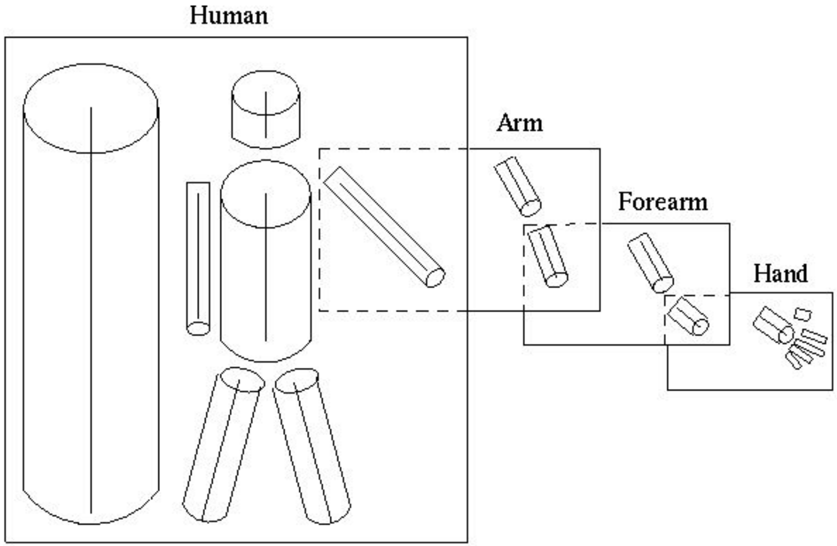
- Around 70 years ago, (later to be) Nobel Prize laureates Hubel & Wiesel made fundamental discoveries about the cat's visual system
- Simple → complex → hypercomplex cells
- Topographical mapping



Hubel & Wiesel, 1968

Marr’s theory of the visual cortex

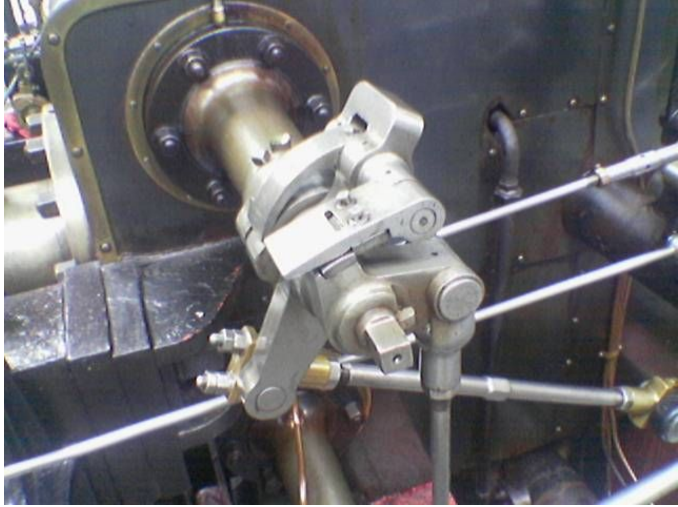
- Marr considered the S+C cells as essentials for computing a *primal sketch*
  - Blobs, lines, edges, curves, etc.
- Then the brain computes a *2.5D sketch* using e.g. textures
  - Local orientation, depth discontinuities
- Finally, a *3D model* is computed
  - Hierarchical
  - Volumetric primitives



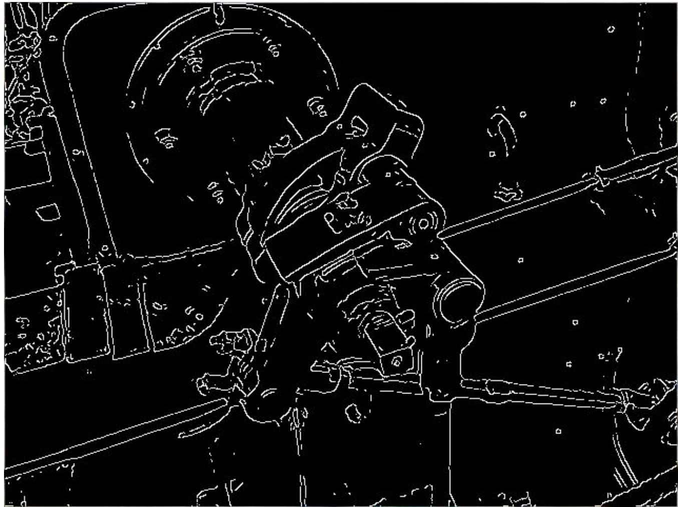
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/GOMES1/marr.html

The Canny edge detector

- A very early computational approach to vision
- Find strong intensity discontinuities in grayscale images

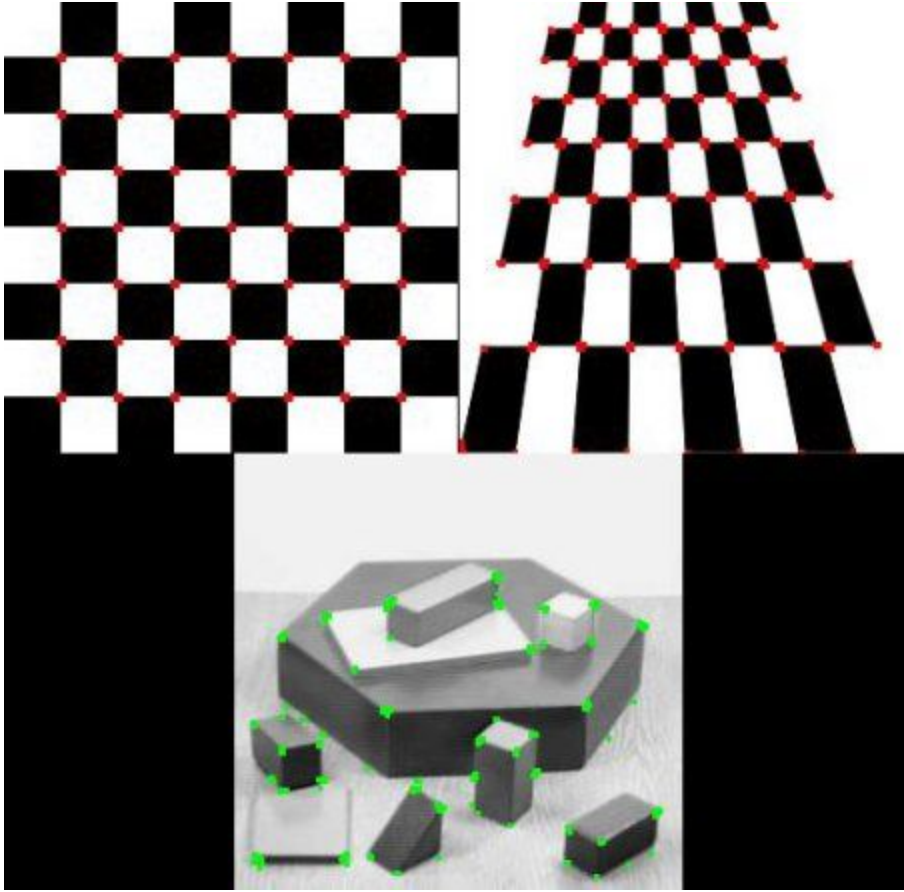


https://en.wikipedia.org/wiki/Canny_edge_detector



The Harris corner detector

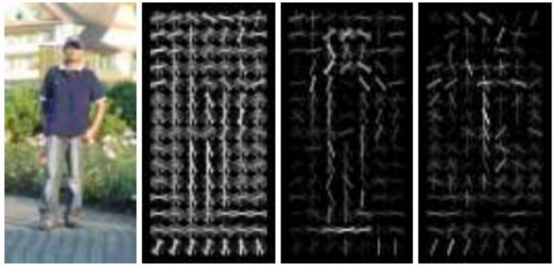
- Use local pixel statistics to find corners instead of edges



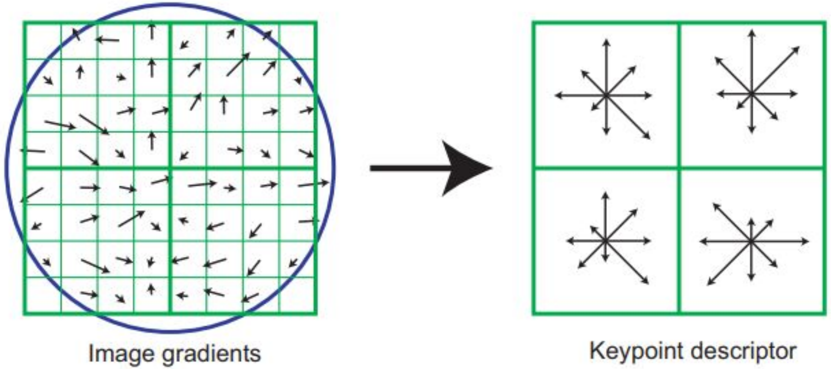
https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

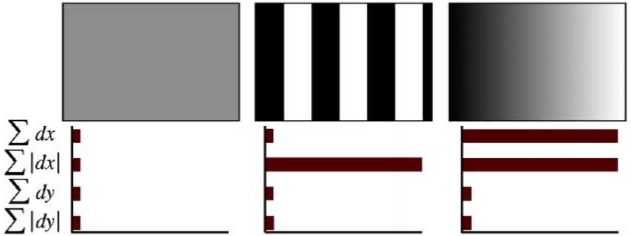
Texture based (local) descriptors

- SIFT (Lowe, 1999 & 2004)
- SURF (Bay et al., 2004)
- HoG (Dalal & Triggs, 2005)



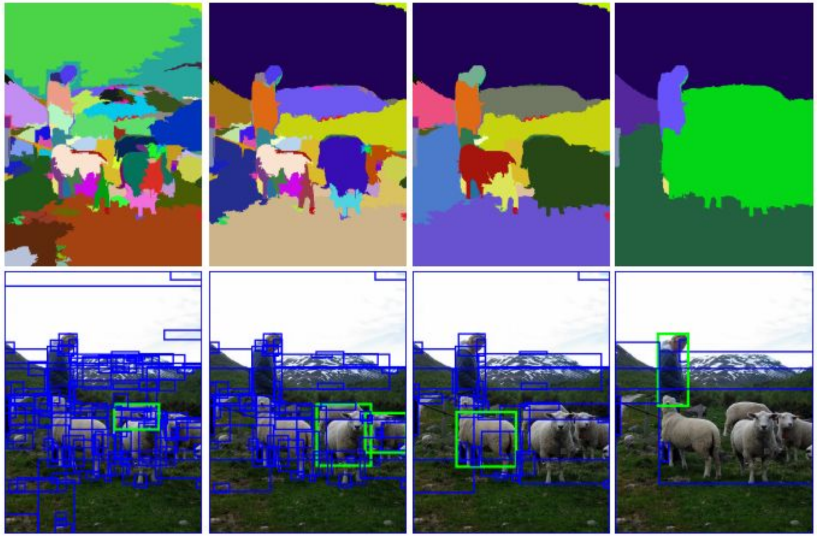
- And countless others...





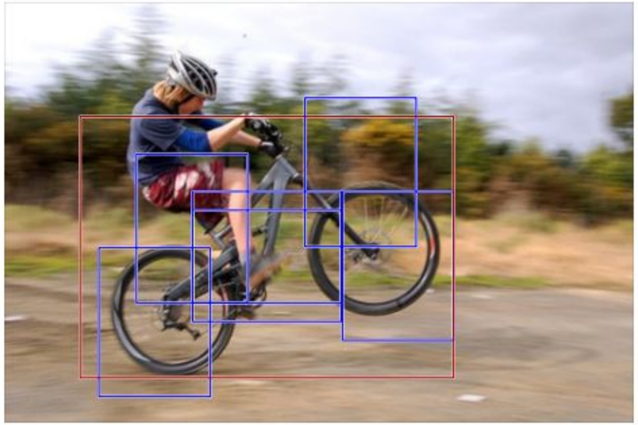
Object detection

- Selective Search by Uijlings et al., 2012
- Use a segmentation algorithm to find object boxes
- Use a SIFT-BoW model to classify boxes



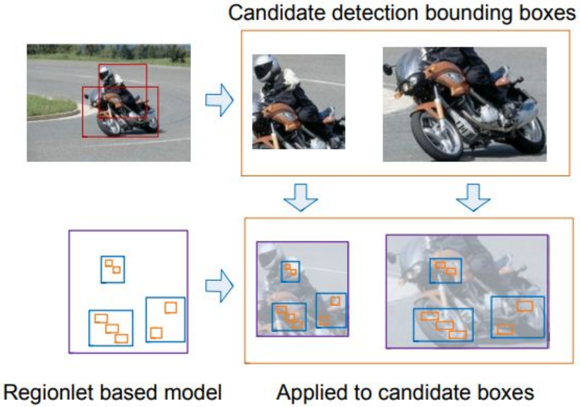
Object detection

- Deformable Part Models by Felzenswalb et al., 2009
- A hierarchical arrangement of HoGs
- A special type of SVMs is used for the final classification



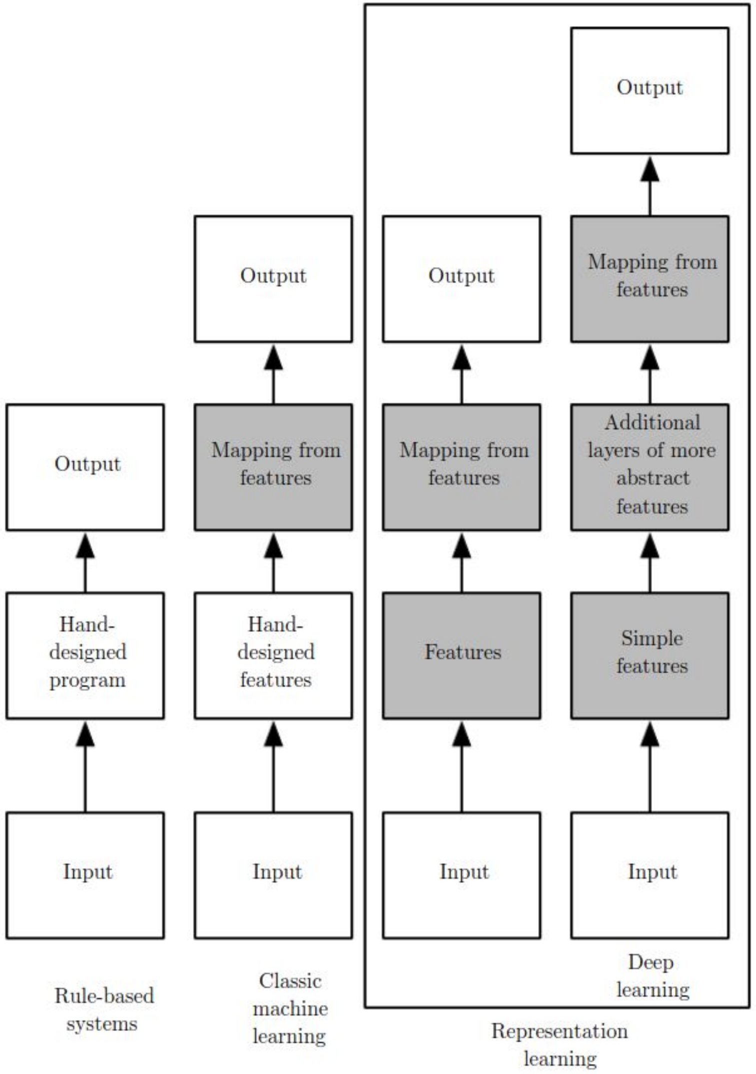
Object detection

- Regionlets by Wang et al., 2013
- Candidate boxes are represented by smaller "regionlets"
- Again HoG (and other local descriptors) are used
- ML is used to map from regionlet representation → object decision



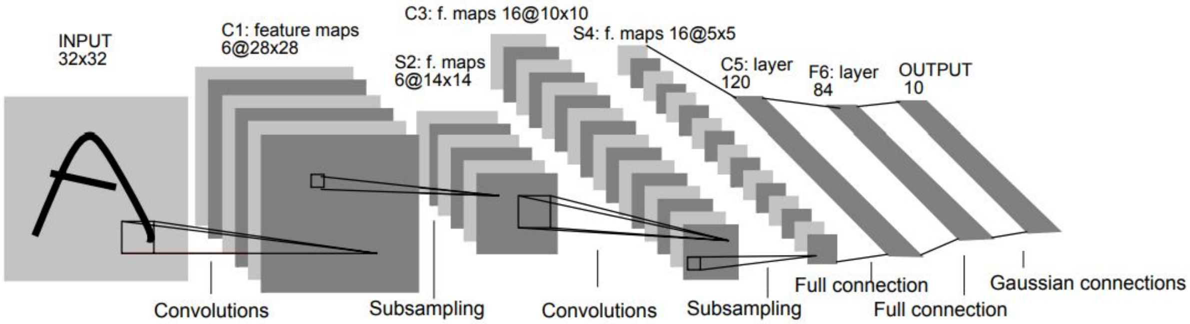
Another view of AI/ML/DL

- Rule-based systems
  - Write code for every action
- Machine learning trends until around 2010
  - Hand-designed features
  - ML on top of these representations
- Representation learning
  - Learn both the representation and how to produce the required output



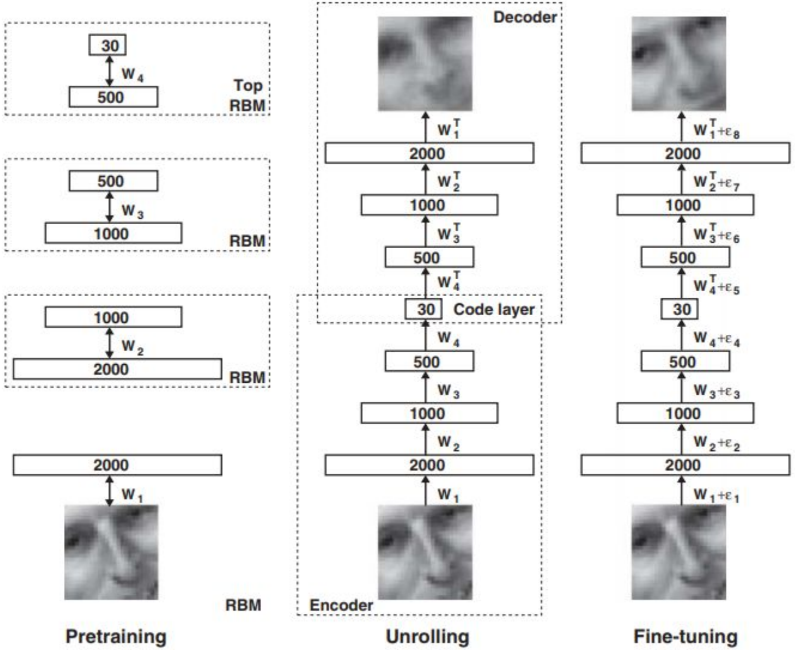
Advances in DL

- Neural models were investigated as early in the 1950s
- Many methods drew inspiration from neuroscience
- Backprop was invented
- One of the earliest “modern” DL algorithms was by LeCun et al. in 1998



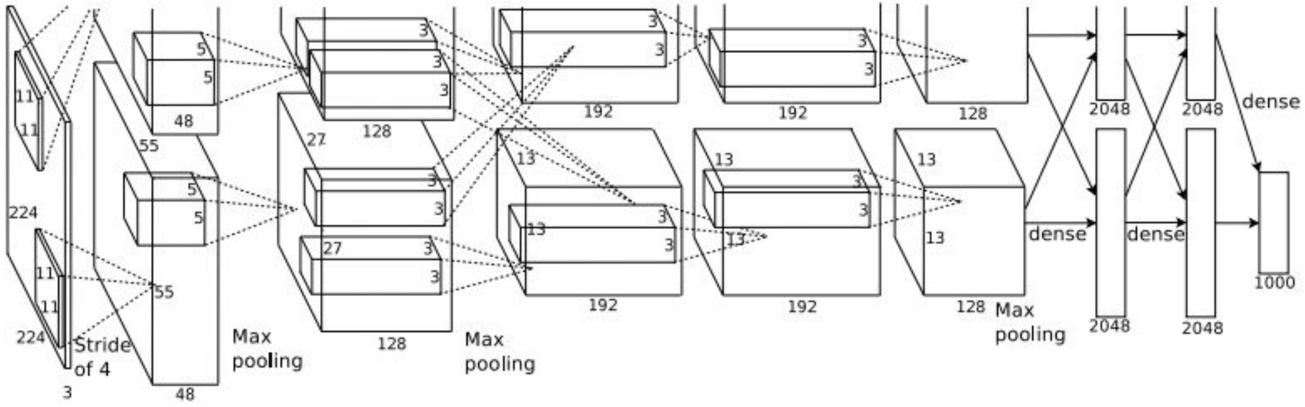
The DL breakthrough

- In 2006, Hinton and colleagues published a new method to train neural nets deeper than before
- A paper in *Science* showed improved classification performance and representation learning abilities (compression)



The DL breakthrough

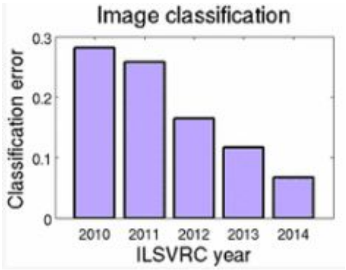
- In 2012, Krizhevsky et al. trained a (then) outrageously big CNN with 8 layers for weeks on a dual-GPU setup
- In the large scale ImageNet classification benchmark ILSVRC, AlexNet achieved an error rate of 15.3 %
- The 2nd place achieved 26.2% error with classical computer vision

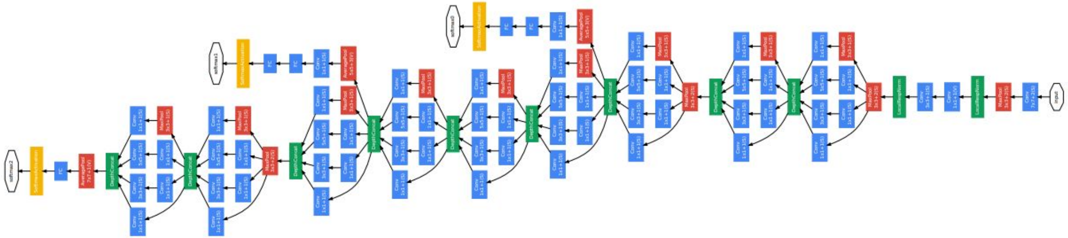


DL advances in image classification

- AlexNet, 2012 ~ 15 % error
- ZFNet, 2013 ~ 14 % error
- GoogLeNet, 2014 ~ 7 % error
- ResNet, 2015 ~ 4 % error

Russakovsky et al., 2015



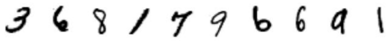


Szegedy et al., 2014

Datasets

- MNIST, 1998
- PASCAL (VOC challenge), 2005-2012
  - Classification, detection, and more
- ImageNet (ILSVRC challenge), since 2010
  - Classification+localization, detection
- COCO, since 2015
  - Detection, segmentation

Lecun et al., 1998

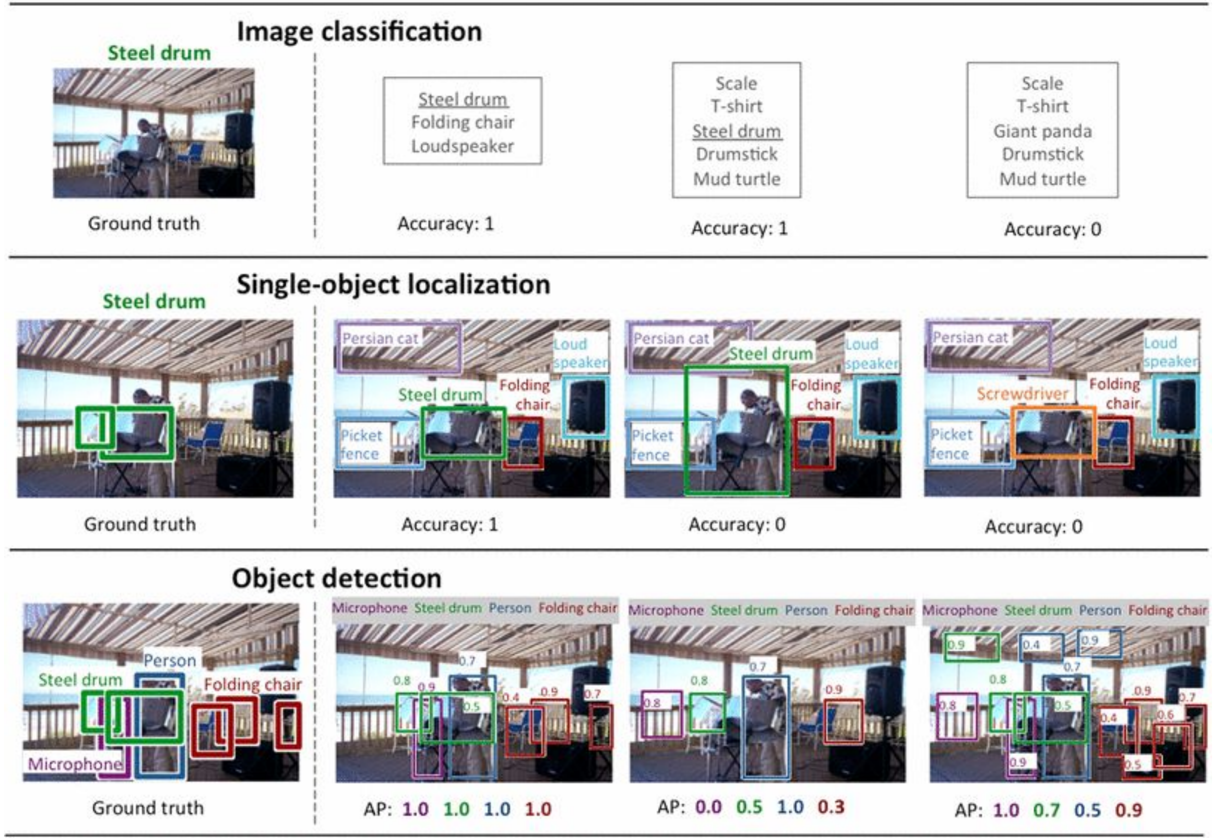


COCO 2019 Keypoint Detection Task

http://cocodataset.org

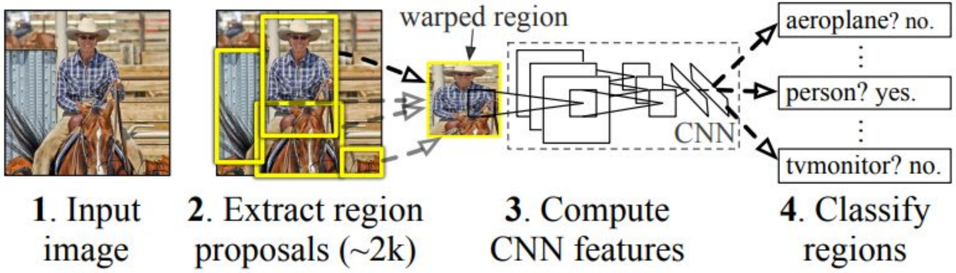
Example tasks from ILSVRC

Russakovsky et al., 2015



Object detection in images

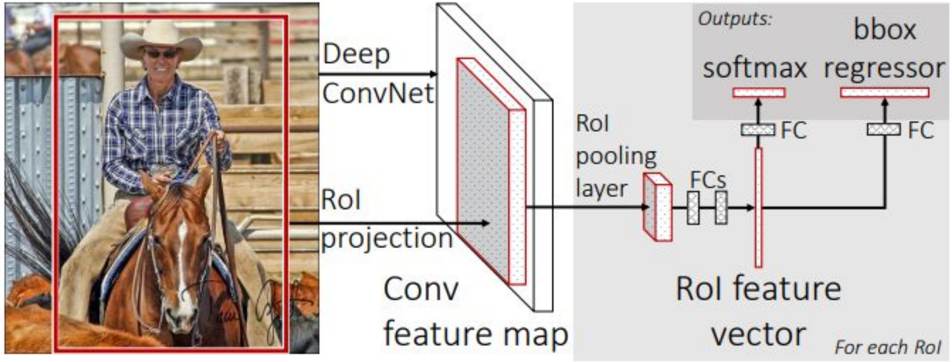
- R-CNN
- Due to Girshick et al., 2014
- Really just a Selective Search + a classifier



VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [17]†	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [32]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [35]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [15]†	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

Object detection

- Fast R-CNN
- Girshick, 2015



method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Object detection

- You guessed it… Faster R-CNN
- Ren et al., 2015

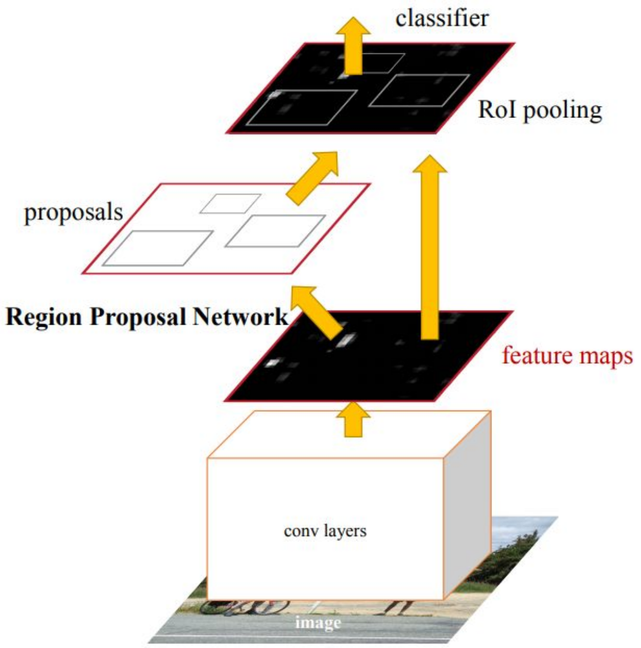
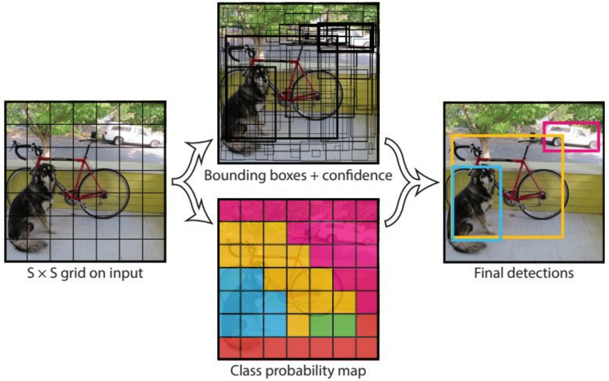


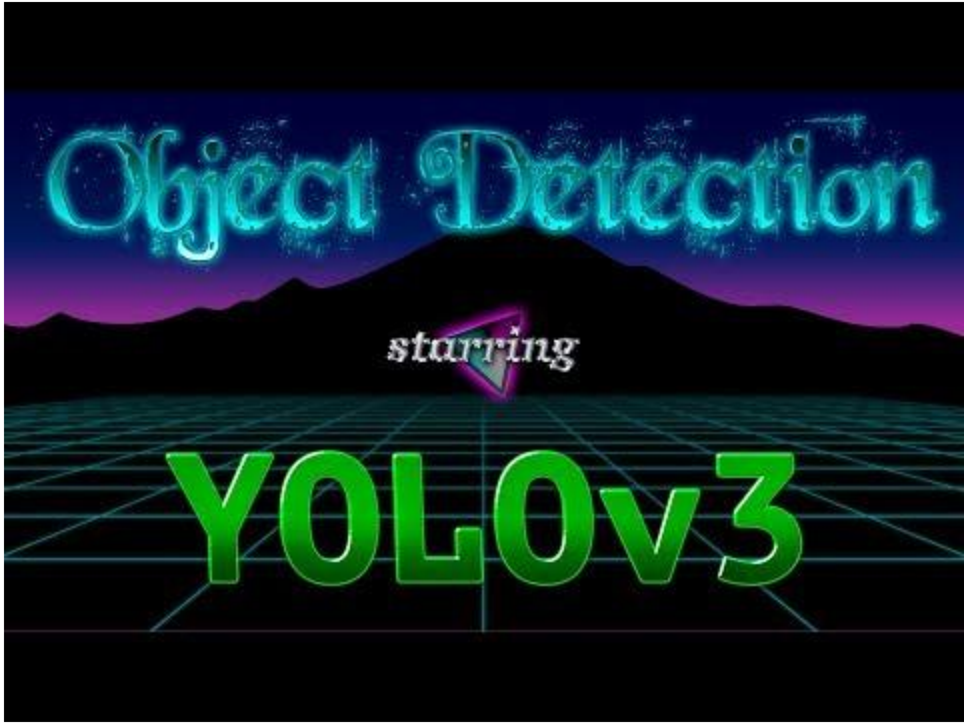
Table 7: Results on PASCAL VOC 2012 test set with Fast R-CNN detectors and VGG-16. For RPN, the train-time proposals for Fast R-CNN are 2000.

method	# box	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SS	2000	12	65.7	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7
SS	2000	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	**65.7**	80.4	64.2
RPN	300	12	67.0	82.3	76.4	71.0	48.4	45.2	72.1	72.3	87.3	42.2	73.7	50.0	86.8	78.7	78.4	77.4	34.5	70.1	57.1	77.1	58.9
RPN	300	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
RPN	300	COCO+07++12	**75.9**	**87.4**	**83.6**	**76.8**	**62.9**	**59.6**	**81.9**	**82.0**	**91.3**	**54.9**	**82.6**	**59.0**	**89.0**	**85.5**	**84.7**	**84.1**	**52.2**	**78.9**	65.5	**85.4**	**70.2**

Object detection

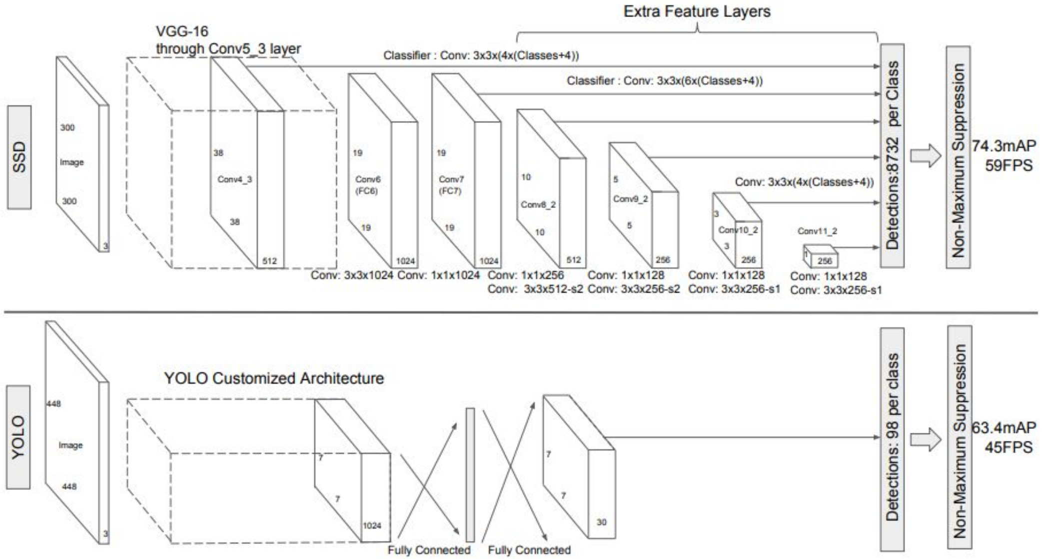
- YOLO by Redmon et al., 2015





Object detection

- SSD by Liu et al., 2016



Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast[6]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster[2]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
Faster[2]	07++12+COCO	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2
YOLO[5]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD300	07++12+COCO	77.5	90.2	83.3	76.3	63.0	53.6	83.8	82.8	92.0	59.7	82.7	63.5	89.3	87.6	85.9	84.3	52.6	82.5	**74.1**	**88.4**	74.2
SSD512	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
SSD512	07++12+COCO	**80.0**	**90.7**	**86.8**	**80.5**	**67.8**	**60.8**	**86.3**	**85.5**	**93.5**	**63.2**	**85.7**	**64.4**	**90.9**	**89.0**	**88.9**	**86.8**	**57.2**	**85.1**	72.8	**88.4**	**75.9**

Table 4: **PASCAL VOC2012 `test` detection results.** Fast and Faster R-CNN use images with minimum dimension 600, while the image size for YOLO is 448×448. data: ”07++12”: union of VOC2007 `trainval` and `test` and VOC2012 `trainval`. ”07++12+COCO”: first train on COCO `trainval35k` then fine-tune on 07++12.

Similar advances from DL

- Sequence (e.g. speech) processing
- Machine translation
- Image segmentation
- Image captioning

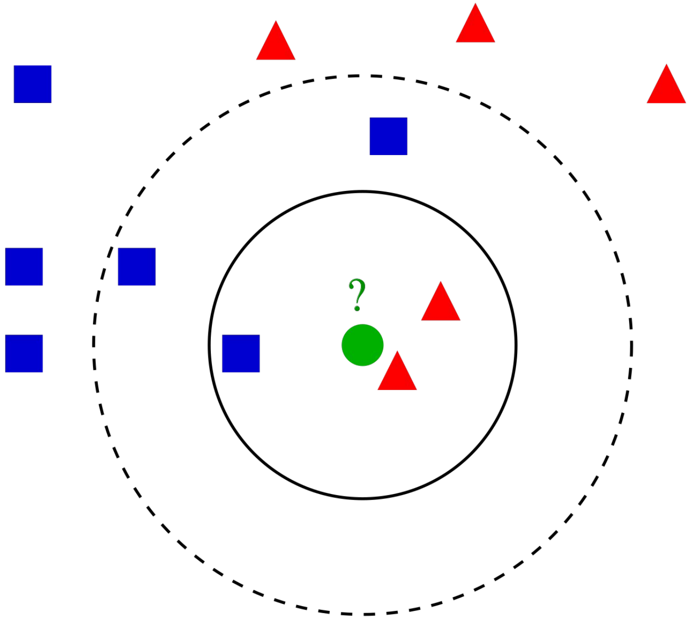
Karpathy and Fei-Fei, 2015

Types of ML tasks

- Classification
  - From the input data, output the label/category which the input belongs to
  - Discrete output (sometimes a pdf over labels)
- Regression
  - Predict a continuous output from the input
- Dimensionality reduction (compression)
- Clustering
- - and many others...

The *k*-NN classifier

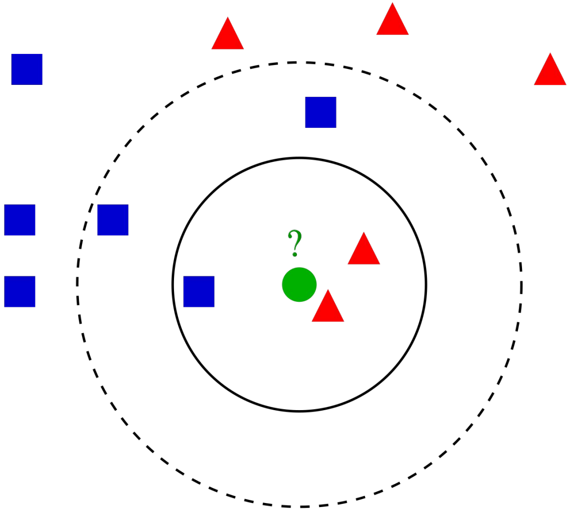
The *k*-NN classifier



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

The *k*-NN classifier

- Training
    - Simply store the images and labels
- Testing
    - For each data point (image), find the k nearest training examples
    - Decide on incoming label by majority vote

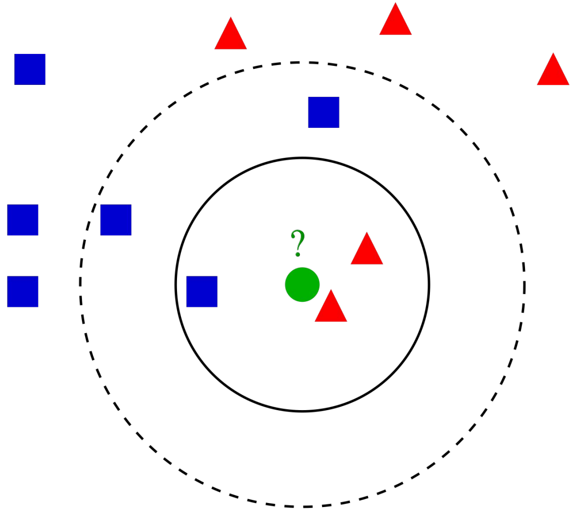


https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

The *k*-NN classifier

- Given a data point *x*, decide its class *C*
- In other words, we want, for each *k*'th class: $p(C_k|x)$
- The winning class is the one with highest probability
- Good ol' Bayes said:

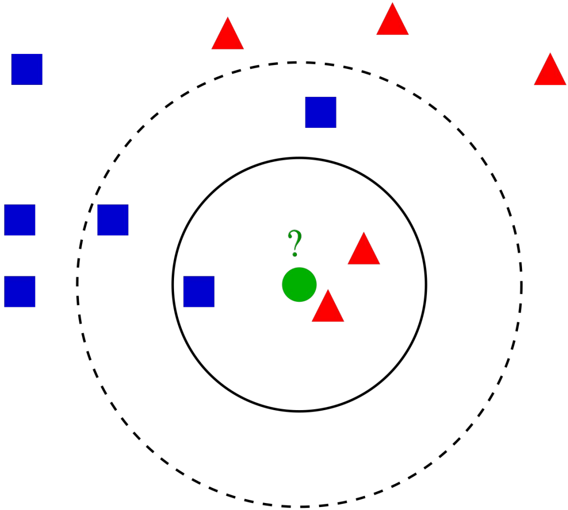
$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

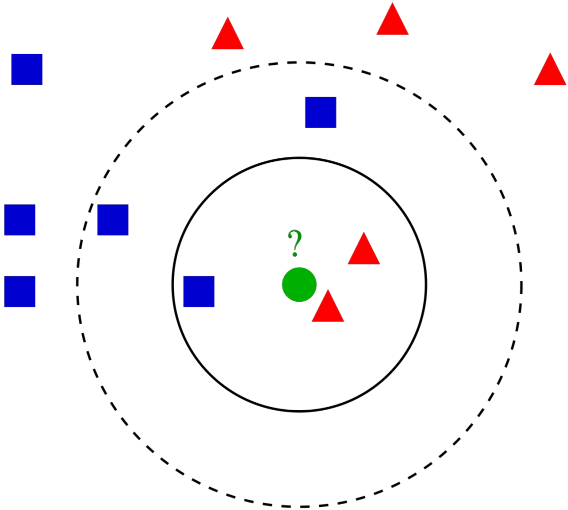
The k-NN classifier

- Given a data point x, decide its class C
- Find the K nearest data points around x
- Denote the total number of training data points N
- Probability of x conditioned on class k:
  $p(x|C_k) = \frac{K_k}{N_k}$
- Unconditional probability of x:
  $p(x) = \frac{K}{N}$
- Unconditional probability of C_k:
  $p(C_k) = \frac{N_k}{N}$



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

The *k*-NN classifier



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

- Probability of *x* conditioned on class *k*:
  $p(x|C_k) = \frac{K_k}{N_k}$
- Unconditional probability (density) of *x*:
  $p(x) = \frac{K}{N}$
- Unconditional probability of C_k (class prior):
  $p(C_k) = \frac{N_k}{N}$
- Final solution using Bayes' theorem
  $p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} = \frac{K_k}{K}$

Getting setup

Google Colab (cloud) or local setup

Google Colab

- Google's free cloud service
- Jupyter notebooks in the cloud (can be saved to Drive)
- Python and TF
- Allows for CPU, GPU, and TPU processing
    - Remember to set to GPU runtime in new notebooks (if you need it)
- Time limit, but unlimited number of runs
- Available at colab.research.google.com

Basics: Python with Numpy and Matplotlib

Why?

Python: General purpose programming language with large ecosystem.
Numpy: Manipulate data in a sane way. Tensorflow, Pytorch etc. have similar interfaces at their core, so Numpy-skills transfer almost directly.
Matplotlib: Visualization is important.

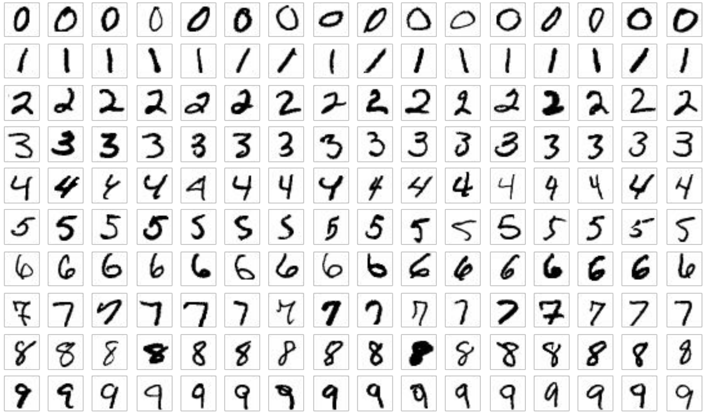
If you are not familiar with the tools, skim through these quickstart tutorials:

Python 3: https://www.programiz.com/python-programming/tutorial
Numpy: https://numpy.org/devdocs/user/quickstart.html
Matplotlib: https://matplotlib.org/tutorials/introductory/pyplot.html

First try at ML (not DL)

https://en.wikipedia.org/wiki/MNIST_database

Our first classifier



- Let's try to classify MNIST
- Training data
  - 60,000 grayscale images with 28x28 px
  - Integer labels from 0 to 9
- Test data
  - 10,000 images and labels
- Objective
  - Train a classifier on the 60,000 training images/labels
  - Run on the 10,000 test images
  - Measure performance using the 10,000 test labels

The *k*-NN classifier

- Details
  - Loading MNIST can be done easily using TF:

```
import torch
from torchvision import datasets
mnist_train_dataset = datasets.MNIST('', train=True, download=True)
mnist_test_dataset = datasets.MNIST('', train=False, download=True)
```

  - Images should be “flattened” to 1D vectors (use NumPy’s reshape())
  - “Storage” can mean both raw (brute force search) or in a search structure, e.g. *k*-d tree
  - With scikit-learn you train with fit() and test with predict()
- **Try it out with the KNeighborsClassifier in scikit-learn**

The *k*-NN classifier

- Now try with the (more difficult) CIFAR-10 (not 100) dataset

```
cifar10_train_dataset = datasets.CIFAR10('../', train=True, download=True)
cifar10_test_dataset = datasets.CIFAR10('../', train=False, download=True)
```

Challenge

- Try different values for *k* (passed to the KNeighborsClassifier constructor)
- What can you achieve for MNIST and CIFAR-10?
- Visualize some of the errors (image, ground truth, predicted)

Bonus:

- Try different metrics (passed to the KNeighborsClassifier constructor)
- Try extracting features by decomposing the images with PCA (sklearn)