

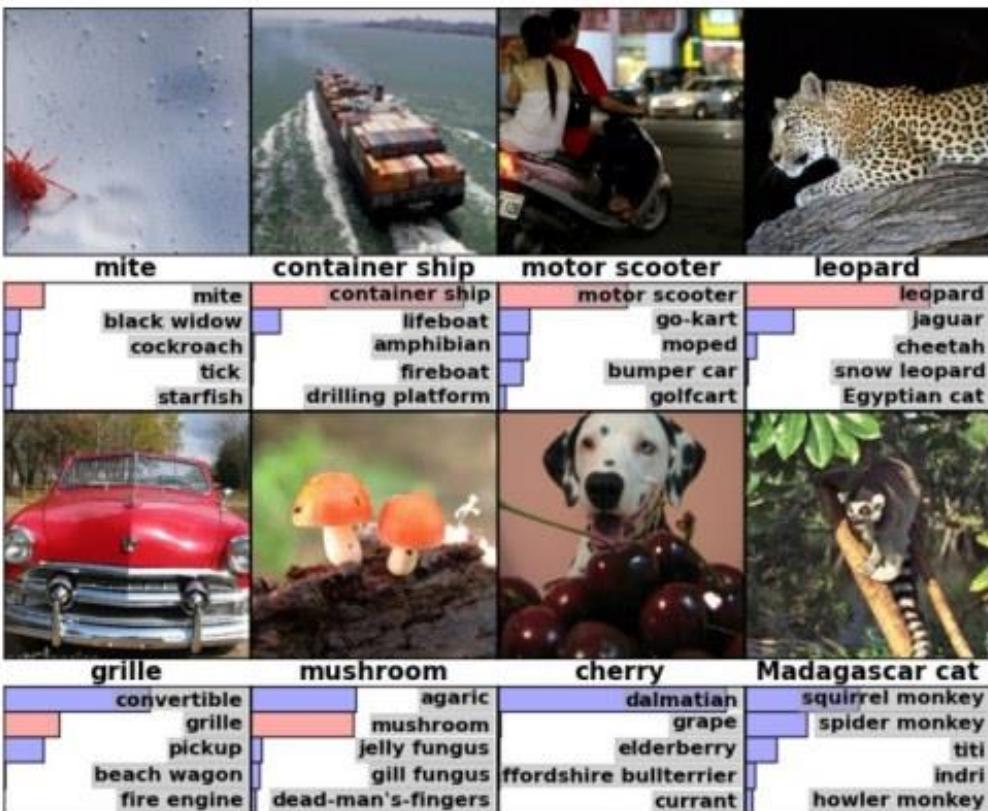
# Deep learning and Computer vision

Dmitry Yashunin  
IntelliVision

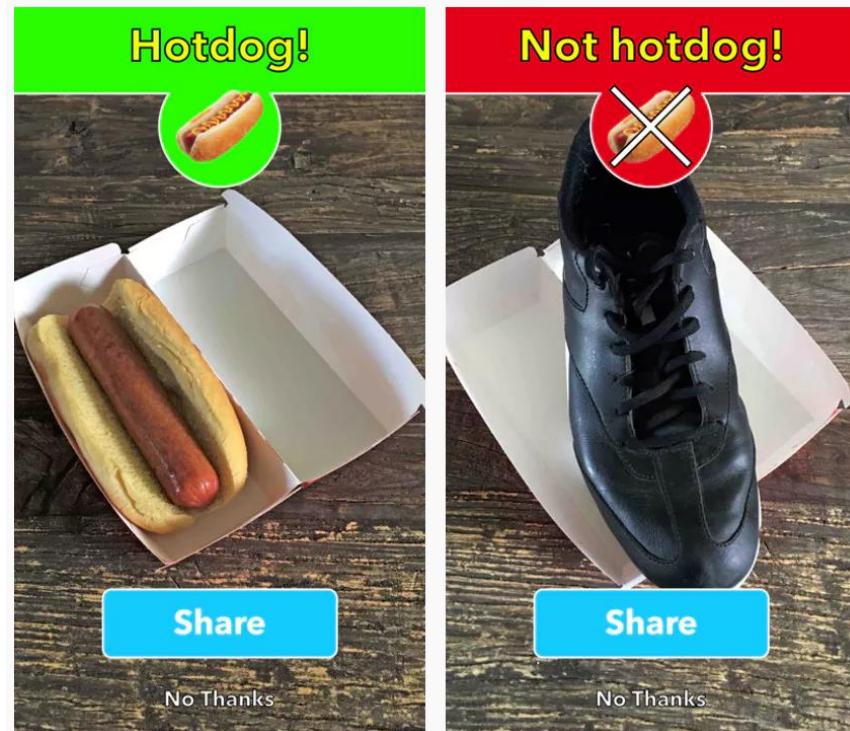
# Lecture 1: Introduction Image classification basics

# Deep learning applications

Image recognition



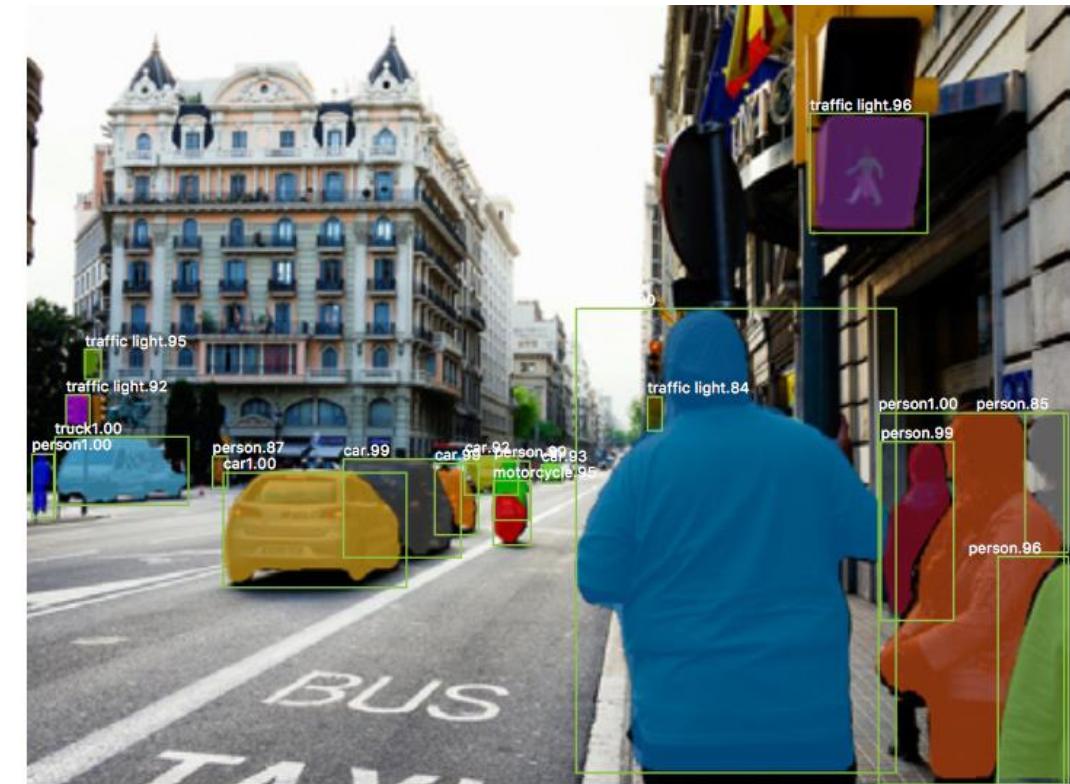
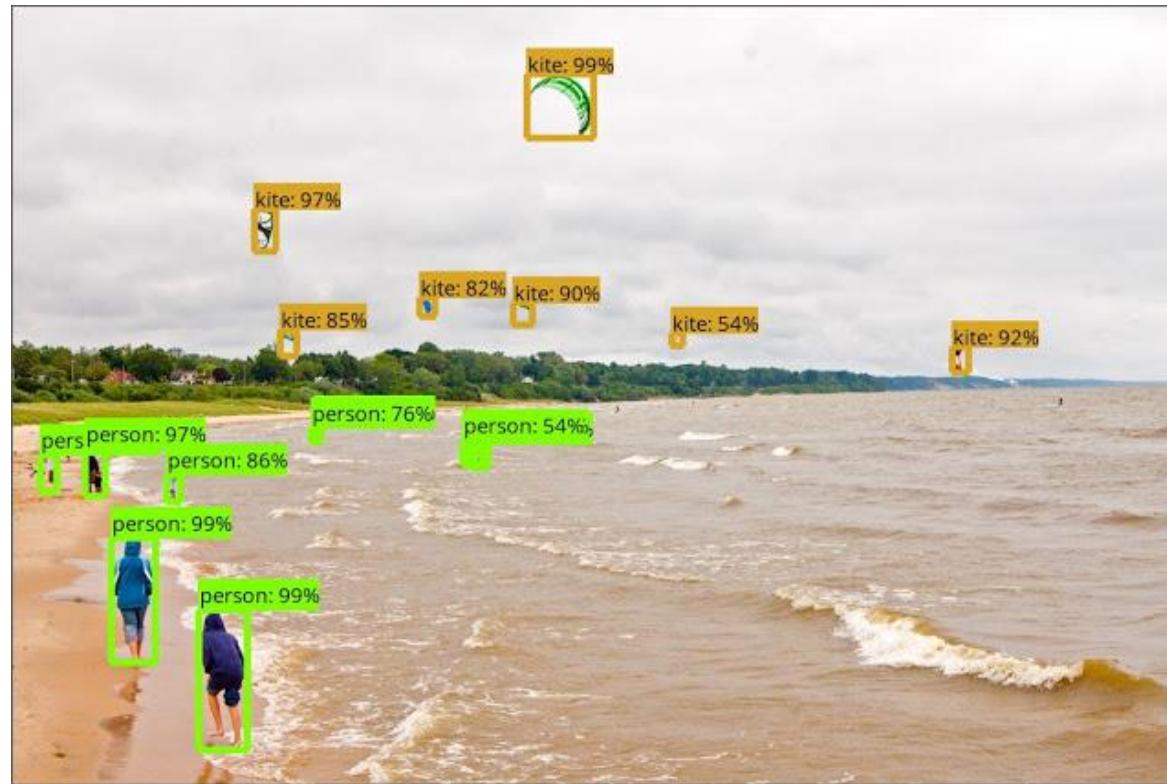
Fun mobile app



# Image captioning

Describes without errors	Describes with minor errors	Somewhat related to the image
 A person riding a motorcycle on a dirt road.	 Two dogs play in the grass.	 A skateboarder does a trick on a ramp.
 A group of young people playing a game of frisbee.	 Two hockey players are fighting over the puck.	 A little girl in a pink hat is blowing bubbles.

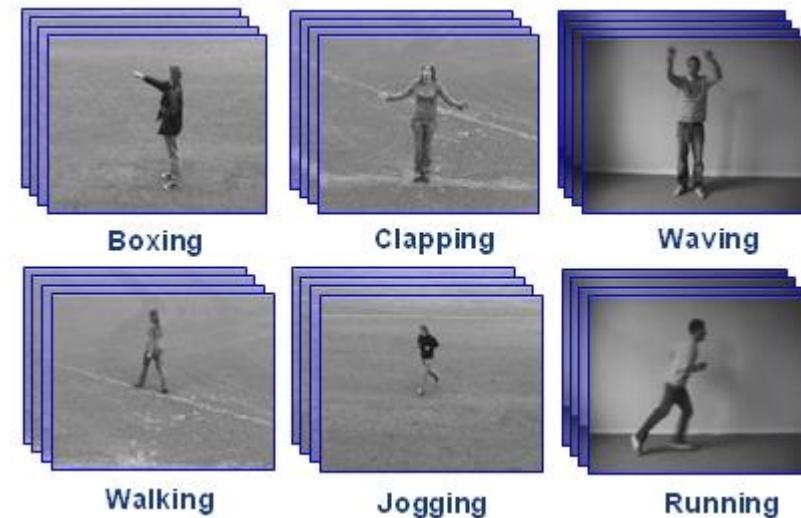
# Detection and instance segmentation



# Optical character recognition



# Action recognition

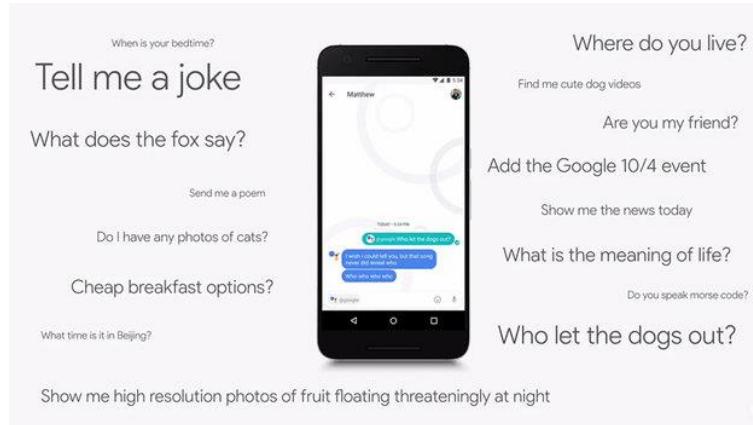


# Speech recognition

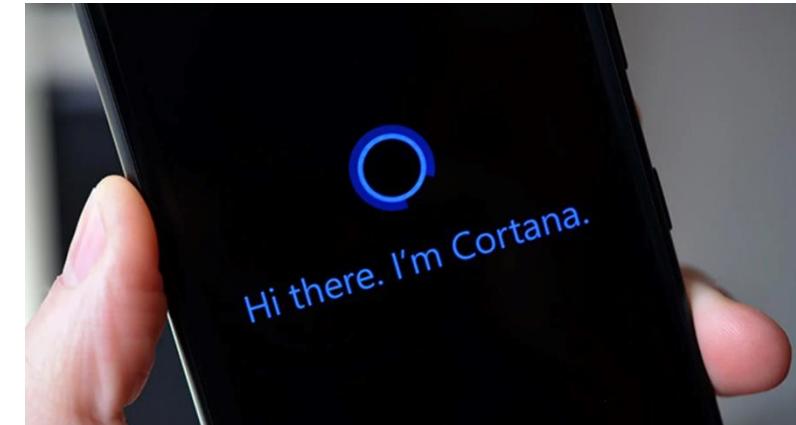
## Apple Siri



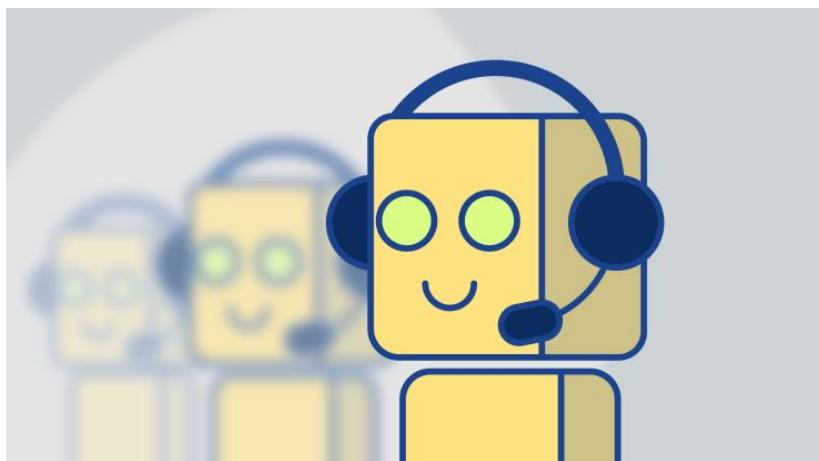
## Google assistant



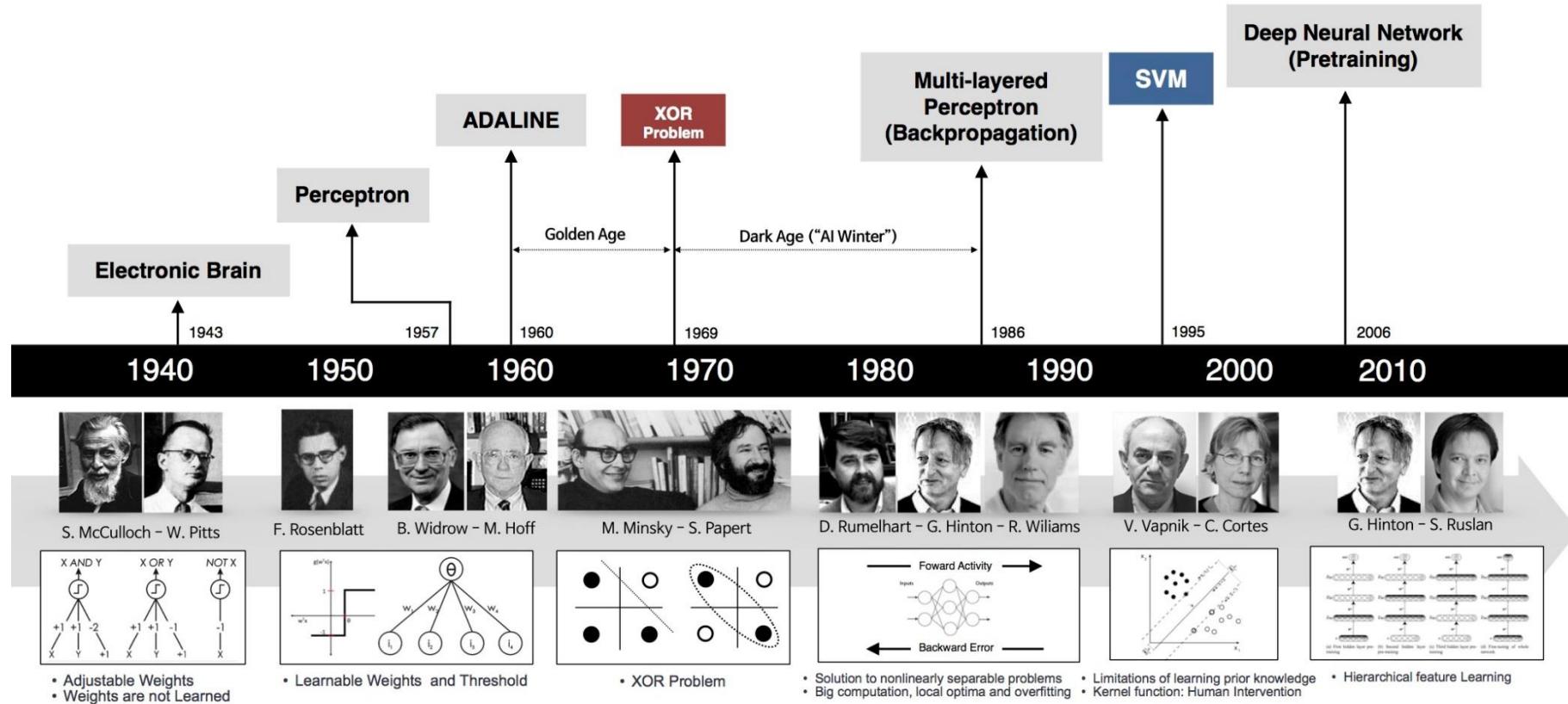
## Microsoft Cortana



## Chat bots

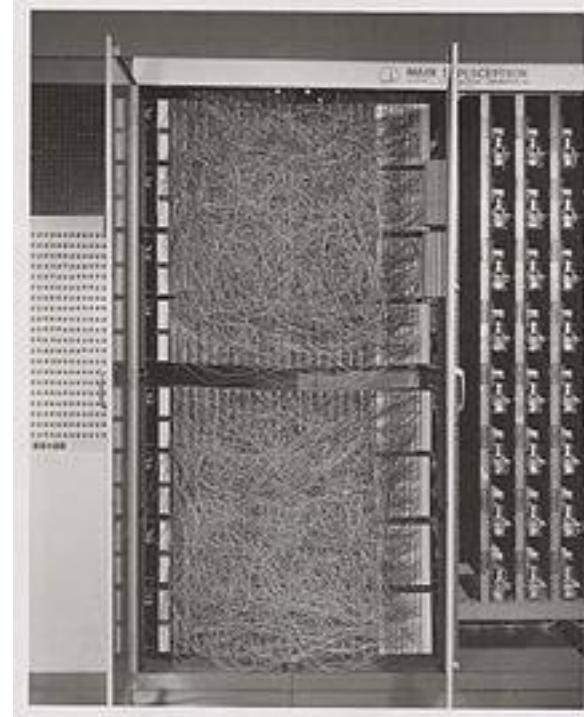
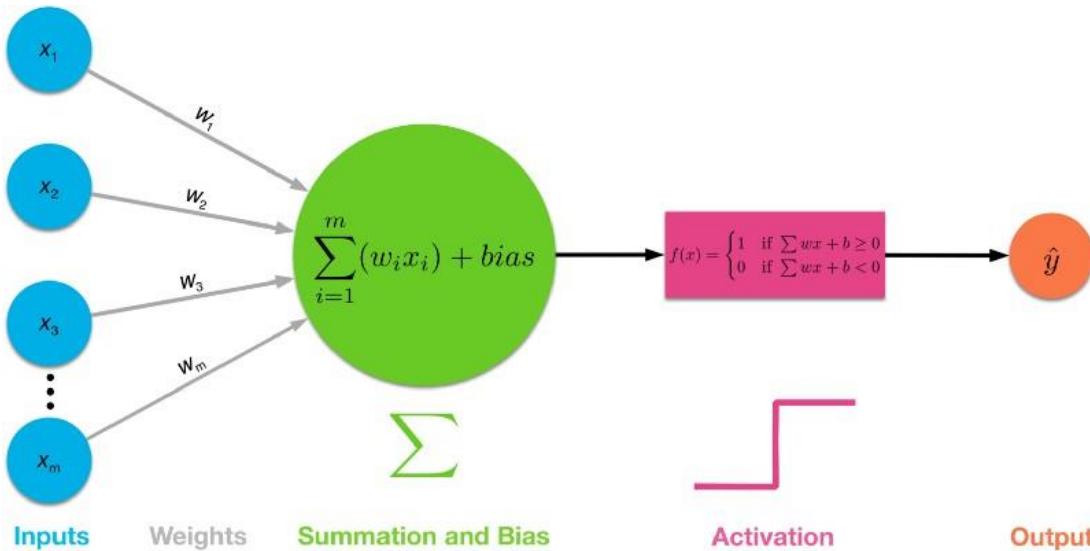


# History of Deep learning



- 1943 – First artificial neuron (Threshold Logic Unit) proposed by Warren McCulloch and Walter Pitts
- 1957 – Perceptron algorithm for pattern recognition. Learning was performed using simple addition and subtraction. Developed by Frank Rosenblatt.

Single-layer perceptron

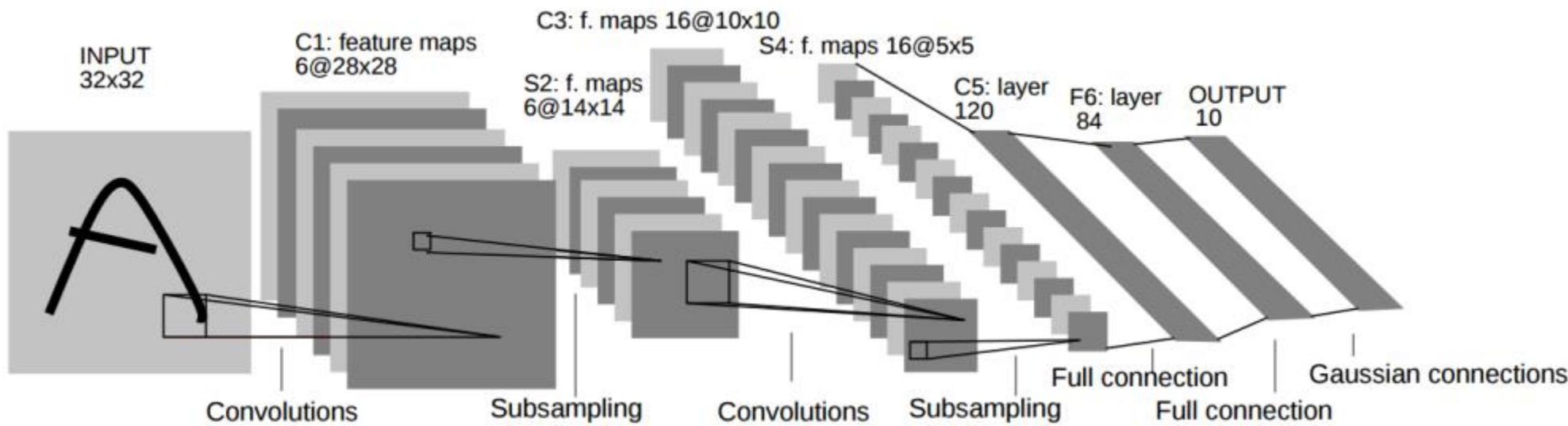


The Mark I Perceptron machine for image recognition – the first implementation of the perceptron algorithm.

The machine was connected to a camera that used  $20 \times 20$  cadmium sulfide photocells to produce a 400-pixel image.

Recognized letters of the alphabet.

- 1986 – Geoffrey Hinton discovered backpropagation algorithm for training multi-layer neural nets
- 1989 – Yann LeCun combined Convolutional Neural Networks (CNN) with back propagation onto read “handwritten” digits  
Demo [https://www.youtube.com/watch?v=FwFduRA\\_L6Q](https://www.youtube.com/watch?v=FwFduRA_L6Q)



- **2012** – Alex Krizhevsky’s CNN wins ImageNet challenge



# Large Scale Visual Recognition Challenge (ILSVRC)



<http://image-net.org/>

22,000 object classes  
14,000,000 images

- Animal
  - fish
  - bird
- Plant
  - tree
  - flower
- Activity
  - sport
- Material
  - Fabric
- Food
  - beverage

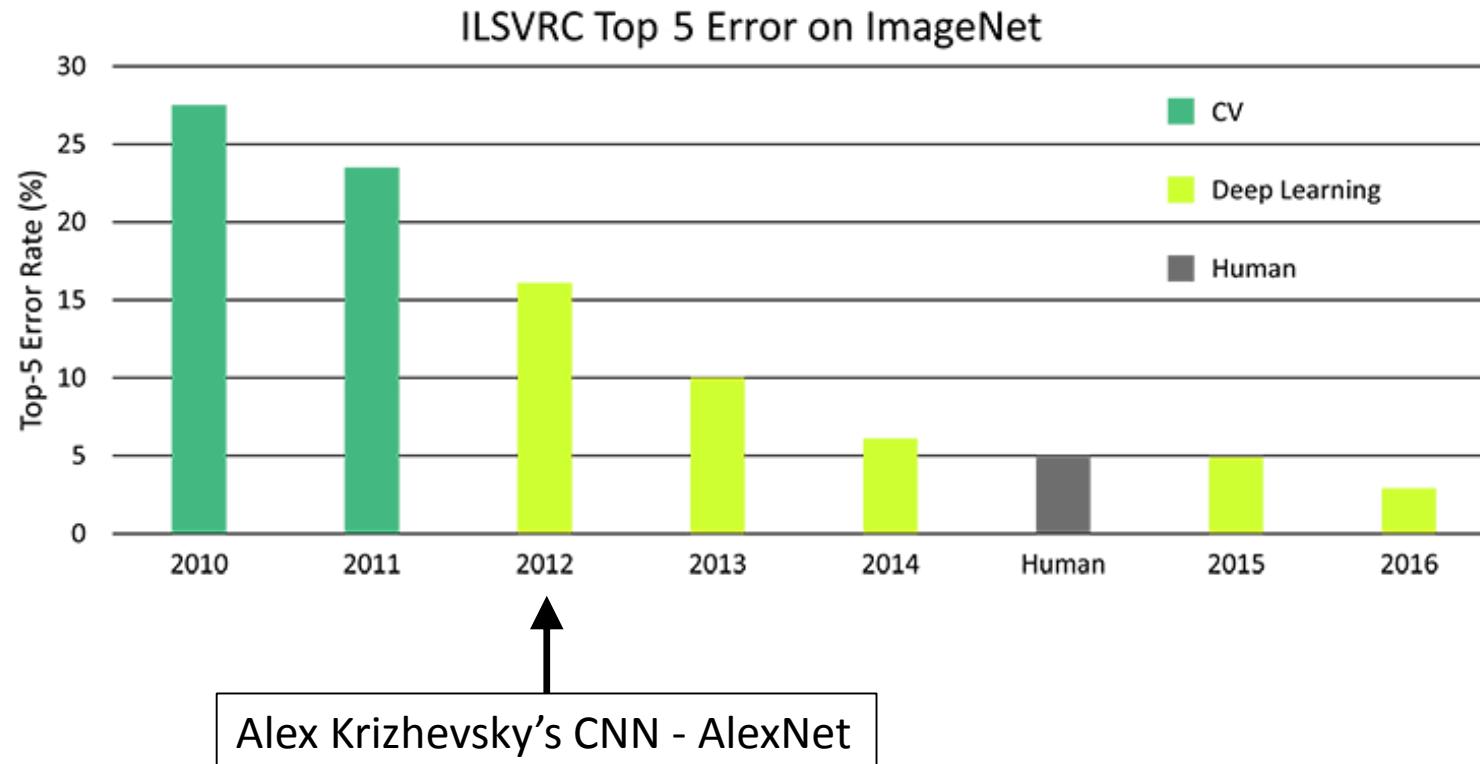


Large Scale Visual Recognition Challenge (ILSVRC)





# Large Scale Visual Recognition Challenge (ILSVRC)

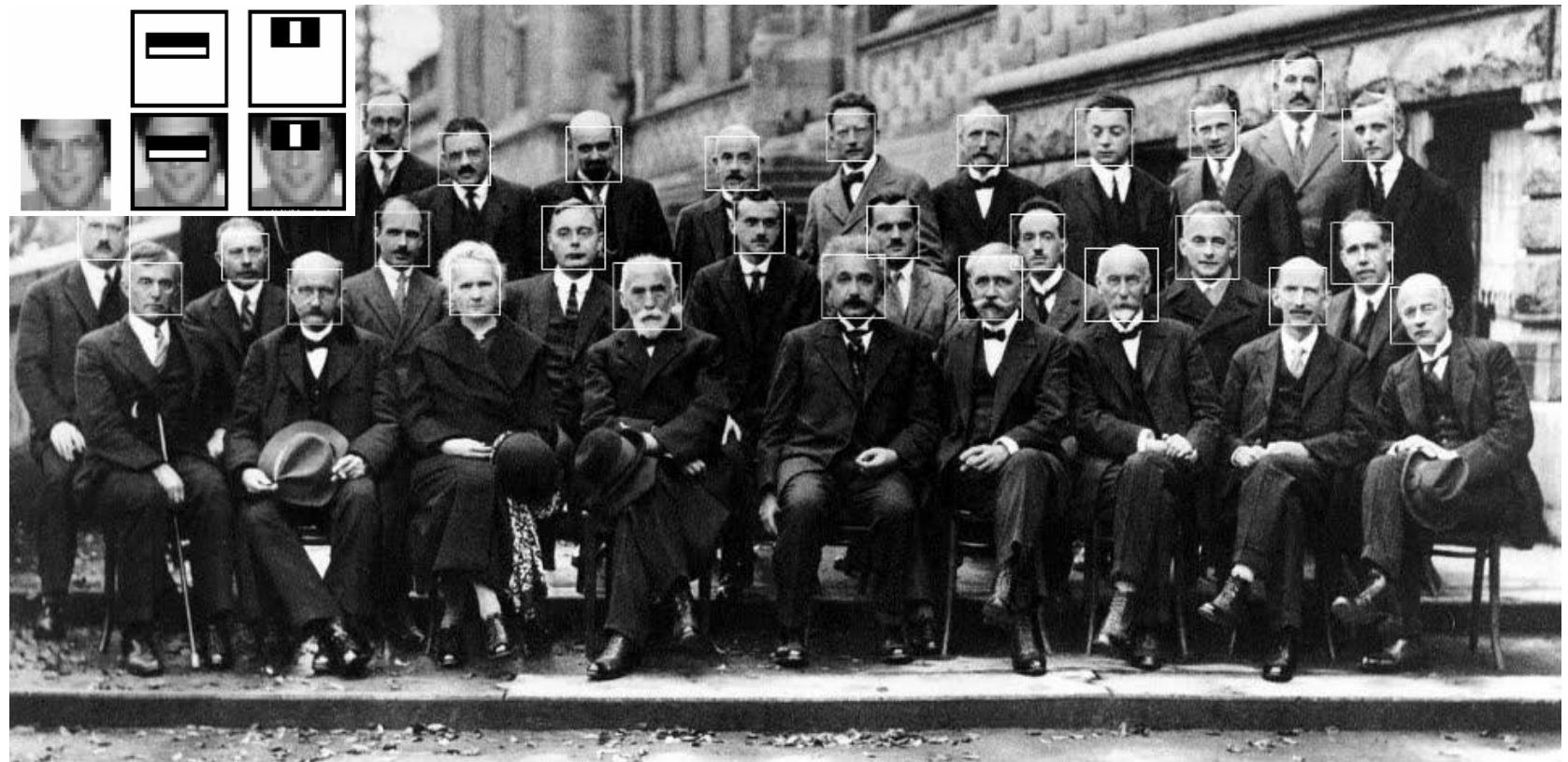


ImageNet classification challenge  
1,000 object classes  
1,431,167 images

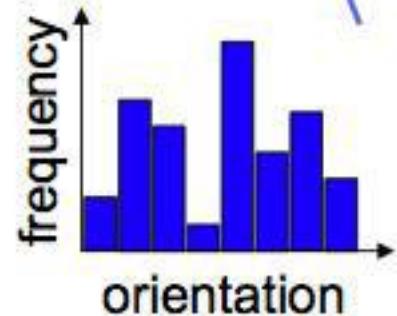
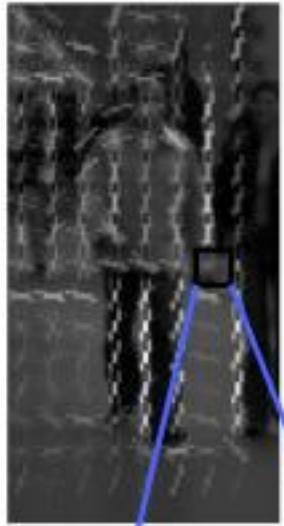
# Before deep learning

Hand crafted features  
and heuristics:

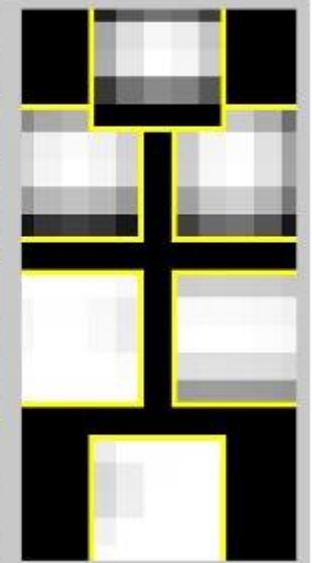
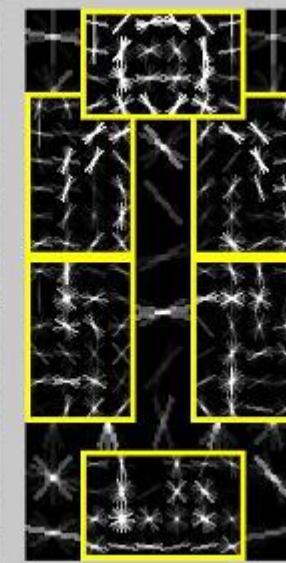
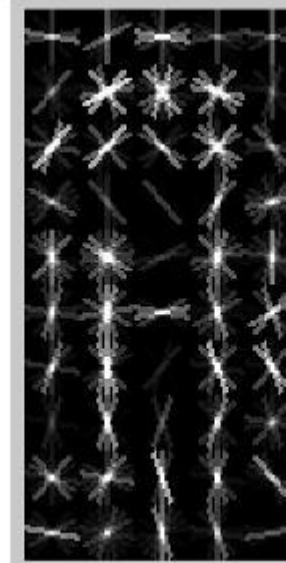
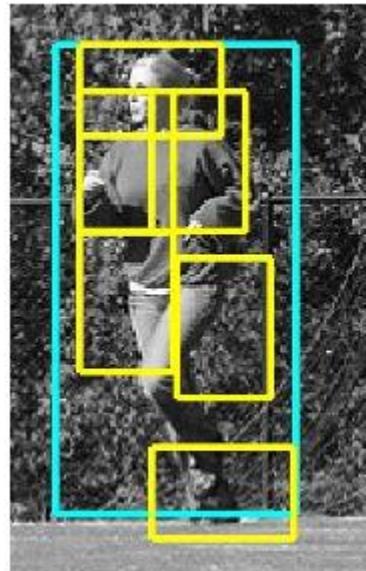
- Edges
- Blobs
- Haar like features
- LBP features
- HOG - Histogram  
of Oriented  
Gradients
- ...



Face detection, Viola & Jones, 2001



HOG - Histogram of Oriented Gradients  
Dalal, Triggs, 2005



Deformable Part Model (DPM)  
Felzenszwalb, McAllester, Ramanan, 2008

# What we will use

- Proficiency in Python
  - Numpy  
see cs231n tutorial <http://cs231n.github.io/python-numpy-tutorial/>
- Calculus, Linear Algebra
  - Differentiation
  - Matrix multiplication
- Machine Learning
  - Basic staff (kNN, SVM, loss functions, cross validation)

# Image classification

A core task in computer vision

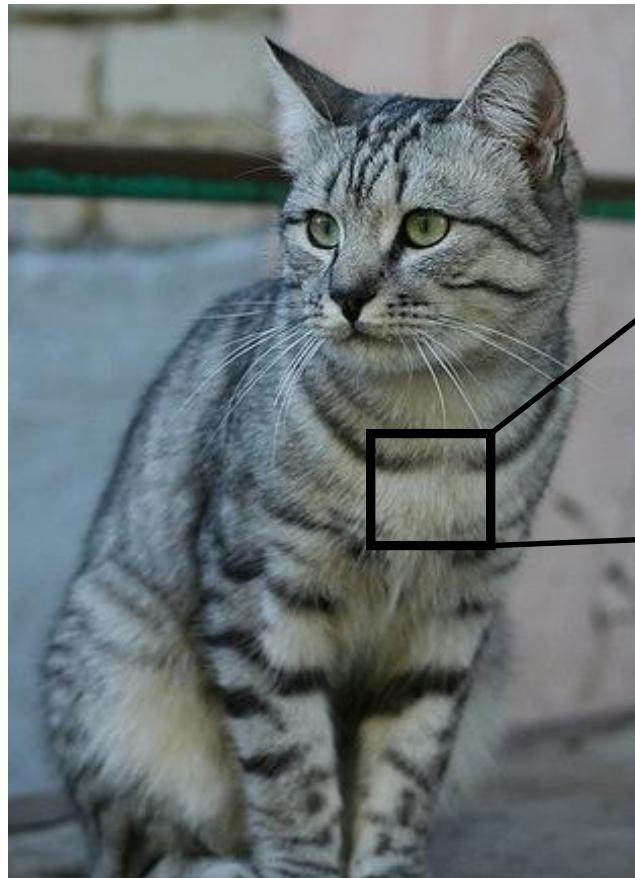


predict classes/labels for given images  
labels: person, animal, vehicle ...



cat

# The problem: semantic gap

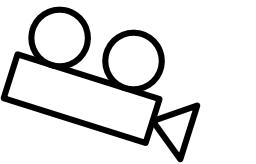


[ [105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
[ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
[ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
[128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]
[ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
[ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]
[ 62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]
[ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]
[ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
[118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]
[164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]
[157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
[128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]

What the computer sees

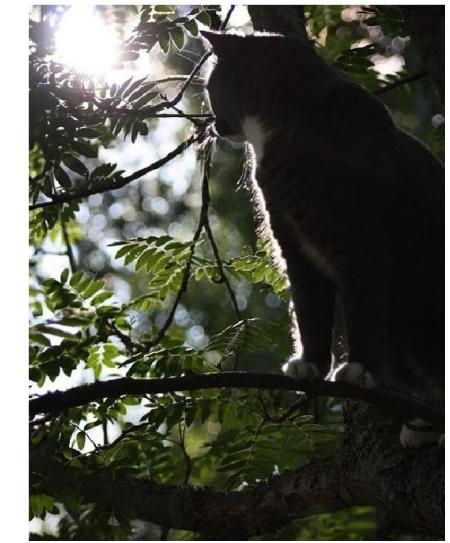
An image is just a sequence of numbers

# Challenges: Viewpoint variation

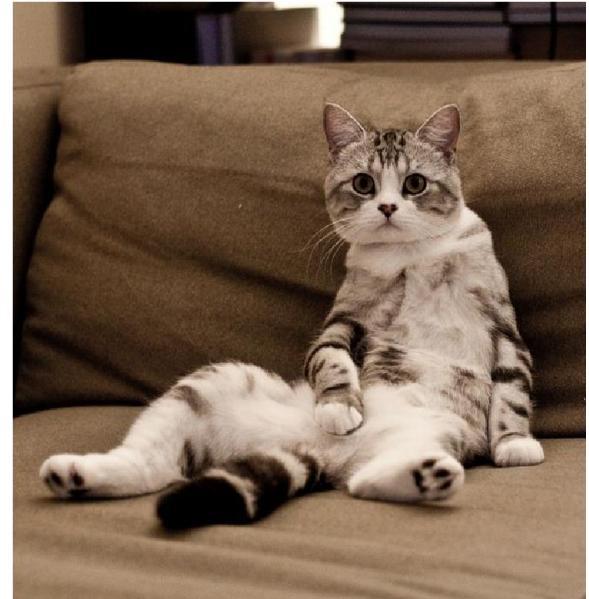


All pixels change when  
the camera moves!

# Challenges: Illumination



# Challenges: Deformation



# Challenges: Occlusion



# Challenges: Background Clutter



# Challenges: Intraclass variation



# Image classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

**no obvious** way to write 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 a classifier
3. Evaluate the classifier on new images

Example training dataset

airplane



automobile



bird



cat



deer



# First classifier: Nearest Neighbor

```
def train(images, labels):  
    # Machine learning!  
    return model
```



Memorize all data  
and labels

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```



Predict the label of  
the most similar  
training image

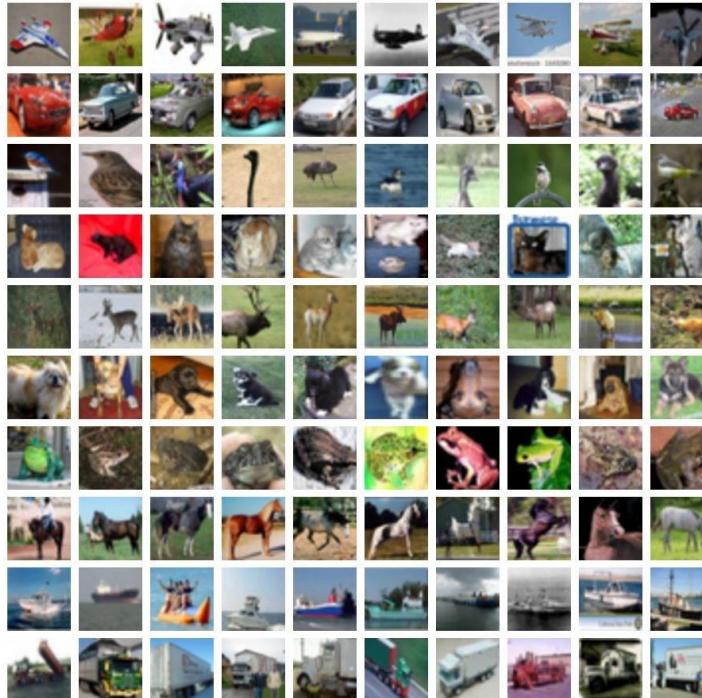
# Example Dataset: CIFAR10

**10** classes

**50,000** training images

**10,000** testing images

**airplane**



**automobile**



**bird**



**cat**



**deer**



**dog**



**frog**



**horse**



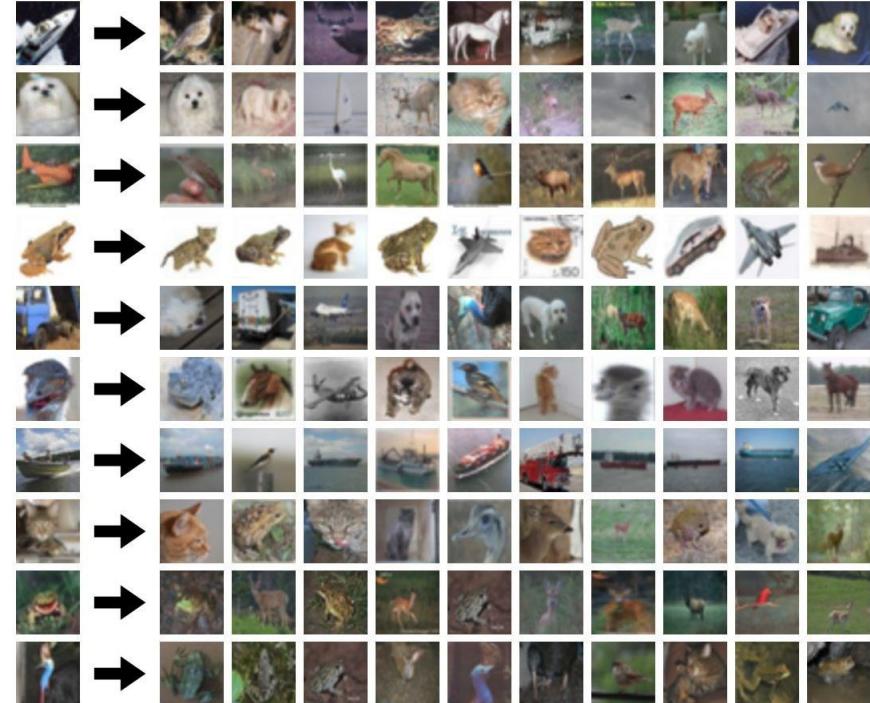
**ship**



**truck**



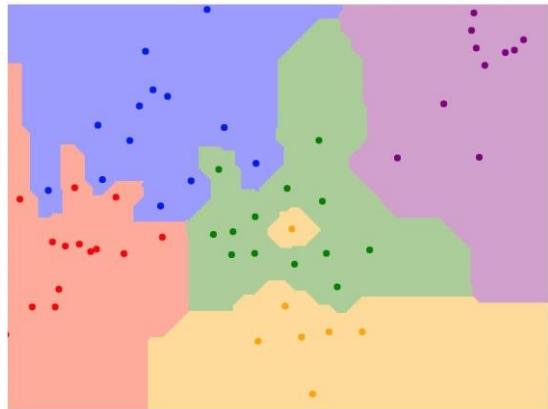
Test images and nearest neighbors



# Distance Metric to compare images

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_{p=pixels} |I_1^p - I_2^p|$$



$K=1$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_{p=pixels} (I_1^p - I_2^p)^2}$$



$K=1$

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

## Nearest Neighbor classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

## Nearest Neighbor classifier

Memorize training data

## Nearest Neighbor classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

For each test image:  
Find closest train image  
Predict label of nearest image

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

## Nearest Neighbor classifier

**Q:** With N examples,  
how fast are training  
and prediction?

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

## Nearest Neighbor classifier

**Q:** With N examples,  
how fast are training  
and prediction?

**A:** Train O(1),  
predict O(N)

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

## Nearest Neighbor classifier

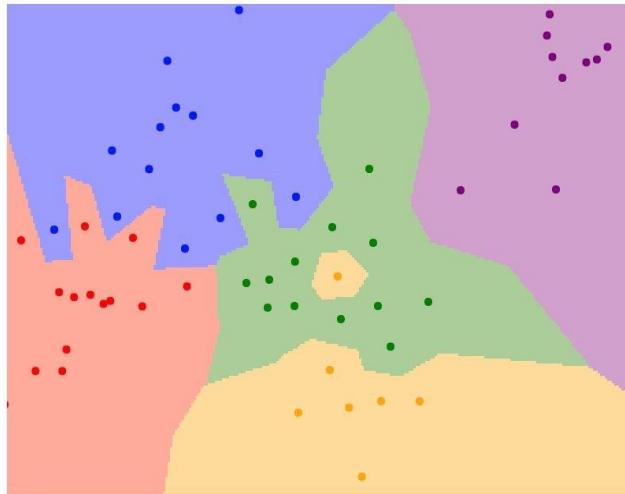
**Q:** With  $N$  examples,  
how fast are training  
and prediction?

**A:** Train  $O(1)$ ,  
predict  $O(N)$

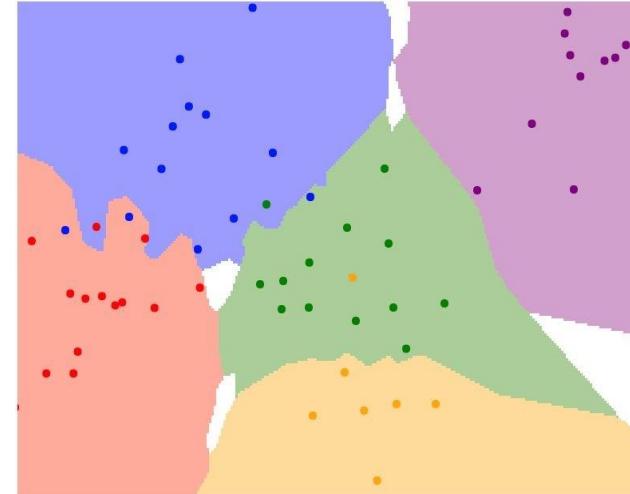
This is bad: we want  
classifiers that are **fast**  
at prediction; **slow** for  
training is ok

# K-Nearest Neighbors

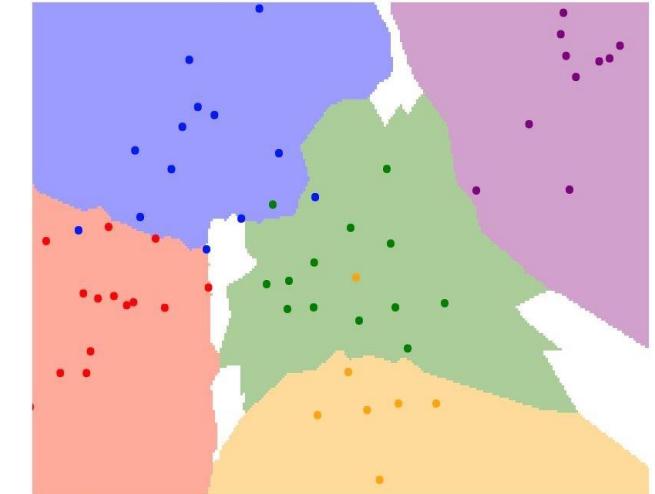
Instead of copying label from nearest neighbor,  
take **majority vote** from K closest points



$K=1$



$K=3$

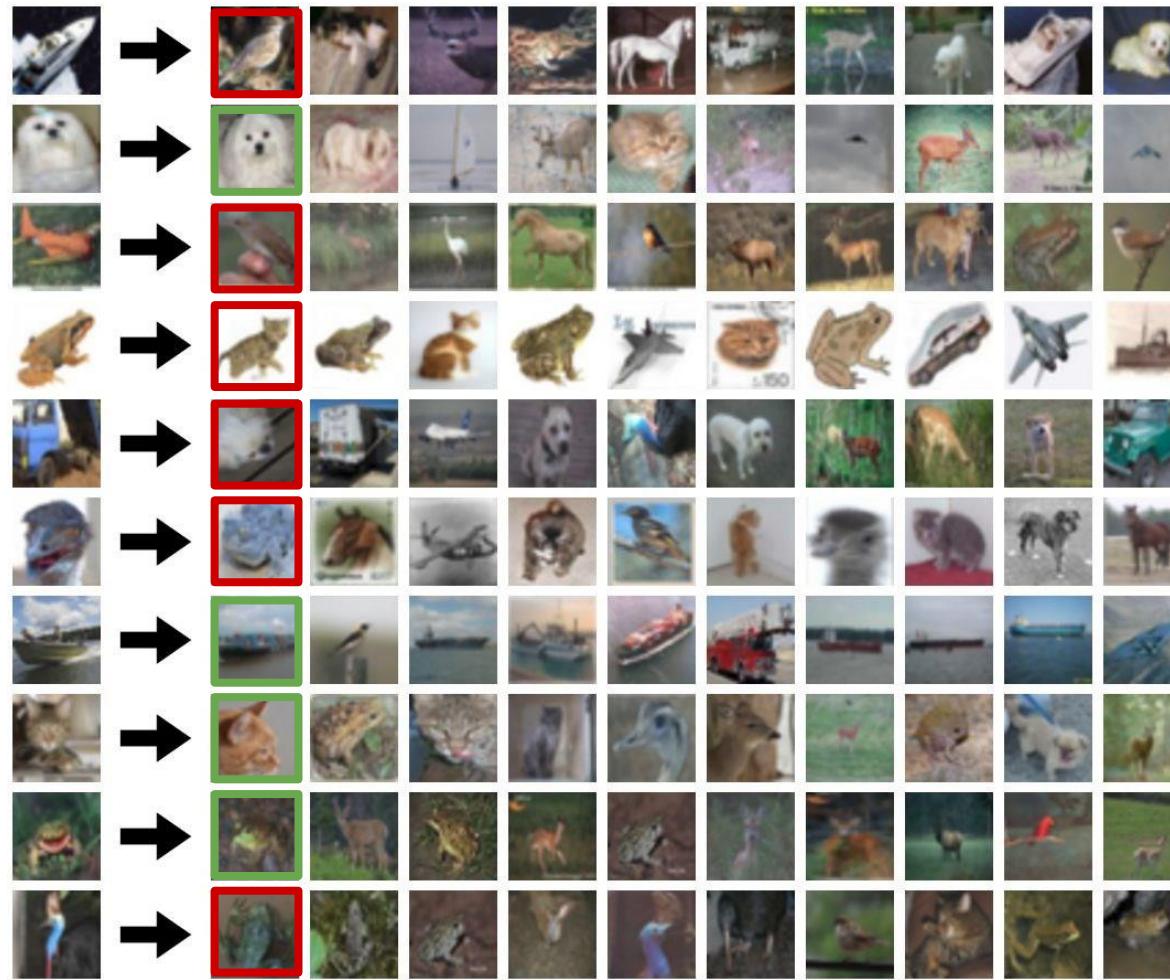


$K=5$

# What does this look like?



# What does this look like?



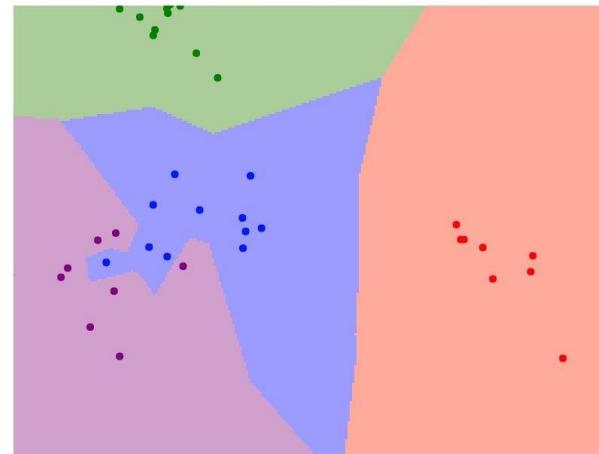
# K-Nearest Neighbors: Demo Time

## K-Nearest Neighbors Demo

This interactive demo lets you explore the K-Nearest Neighbors algorithm for classification.

Each point in the plane is colored with the class that would be assigned to it using the K-Nearest Neighbors algorithm. Points for which the K-Nearest Neighbor algorithm results in a tie are colored white.

You can move points around by clicking and dragging!



Metric

L1 L2

Num classes

2 3 4 5

Num Neighbors (K)

1 2 3 4 5 6 7

Num points

20 30 40 50 60

<http://vision.stanford.edu/teaching/cs231n-demos/knn/>

# Hyperparameters

What is the best value of  $k$  to use?

What is the best **distance** to use?

These are **hyperparameters**: choices about the algorithm that we set rather than learn

Very problem-dependent.

Must try them all out and see what works best.

# Setting Hyperparameters

**Idea #1:** Choose hyperparameters  
that work best on the data

Your Dataset

# Setting Hyperparameters

**Idea #1:** Choose hyperparameters  
that work best on the data

**BAD:**  $K = 1$  always works  
perfectly on training data

Your Dataset

# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

**BAD:**  $K = 1$  always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

train

test

# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

**BAD:**  $K = 1$  always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

**BAD:** No idea how algorithm will perform on new data

train

test

# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

**BAD:**  $K = 1$  always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

**BAD:** No idea how algorithm will perform on new data

train

test

**Idea #3:** Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

**Better!**

train

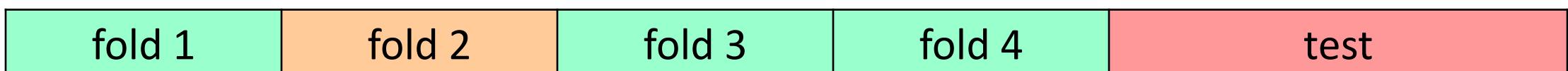
validation

test

# Setting Hyperparameters

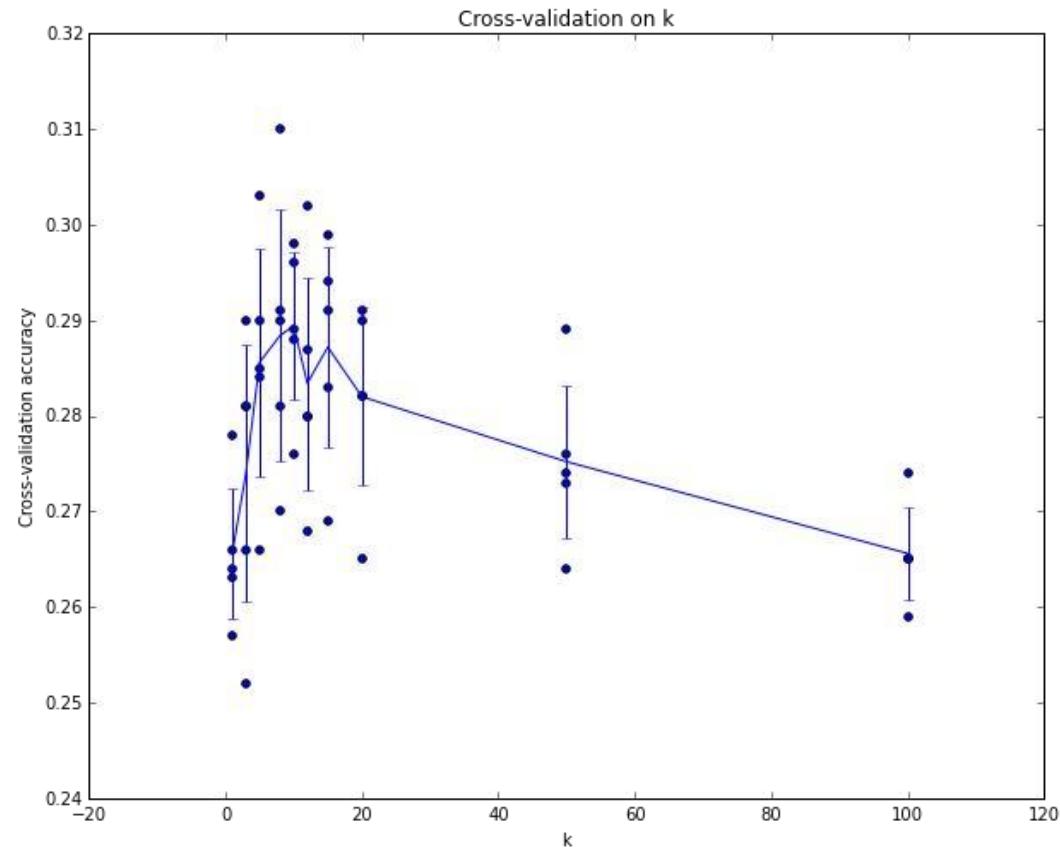
Your Dataset

**Idea #4: Cross-Validation:** Split data into **folds**,  
try each fold as validation and average the results



Useful for small datasets, but not used too frequently in deep learning

# Setting Hyperparameters



Example of  
5-fold cross-validation  
for the value of  $k$ .

The line goes through the mean, bars indicated standard deviation

(Seems that  $k = 7$  works best  
for this data)

# k-Nearest Neighbor on images never used.

- Very slow at test time
- Distance metrics on pixels are not informative

Original



Boxed



Shifted



Tinted



(all 3 images have same L2 distance to the one on the left)

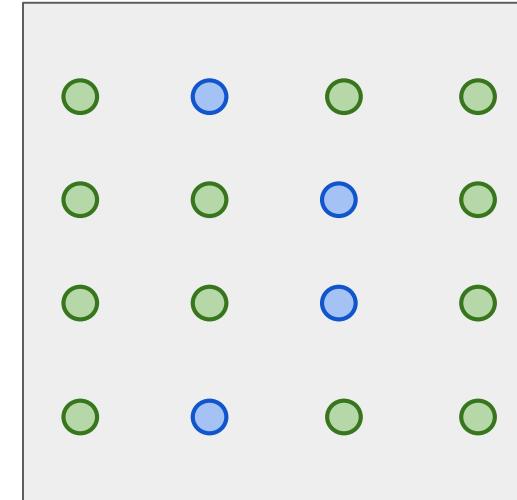
# k-Nearest Neighbor on images never used.

- Curse of dimensionality

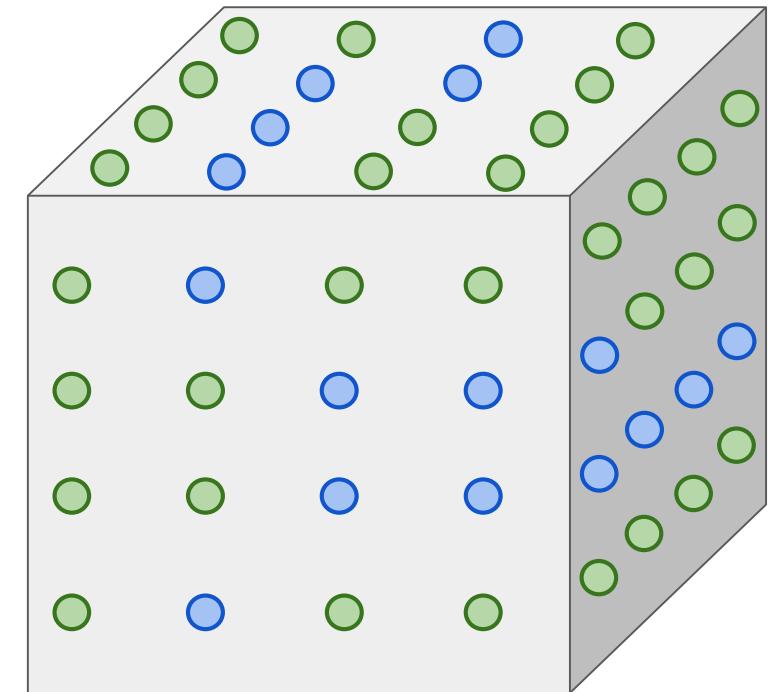
Dimensions = 1  
Points = 4



Dimensions = 2  
Points =  $4^2$

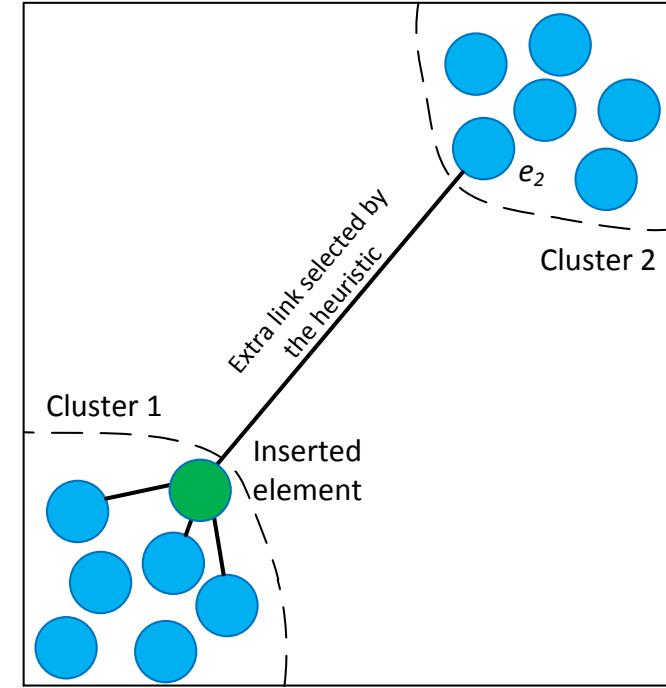
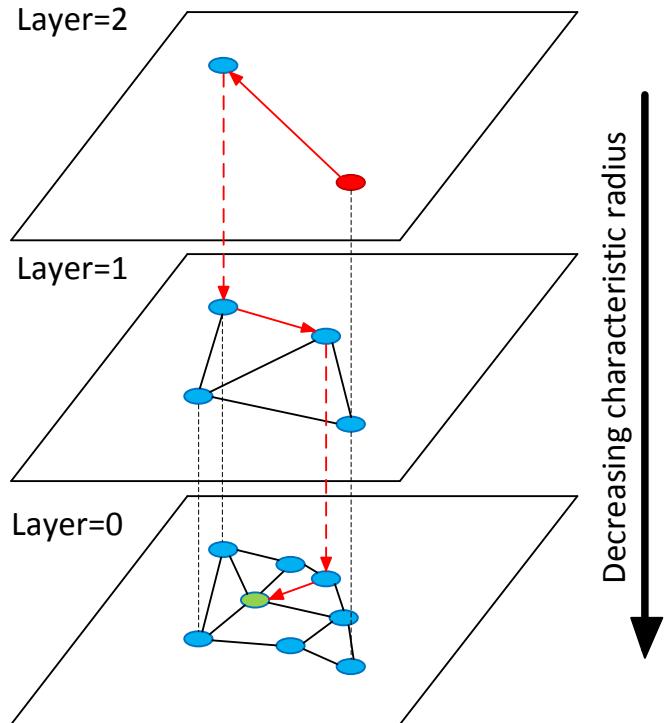


Dimensions = 3  
Points =  $4^3$

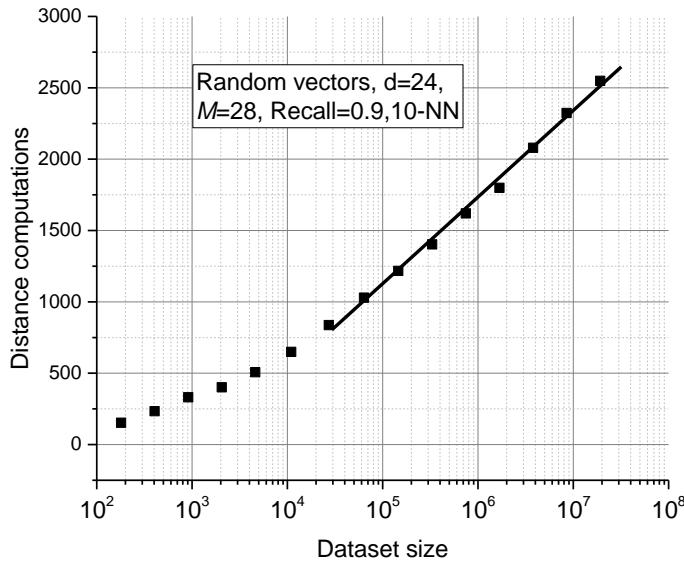


# Approximate Nearest Neighbor Search (ANNS)

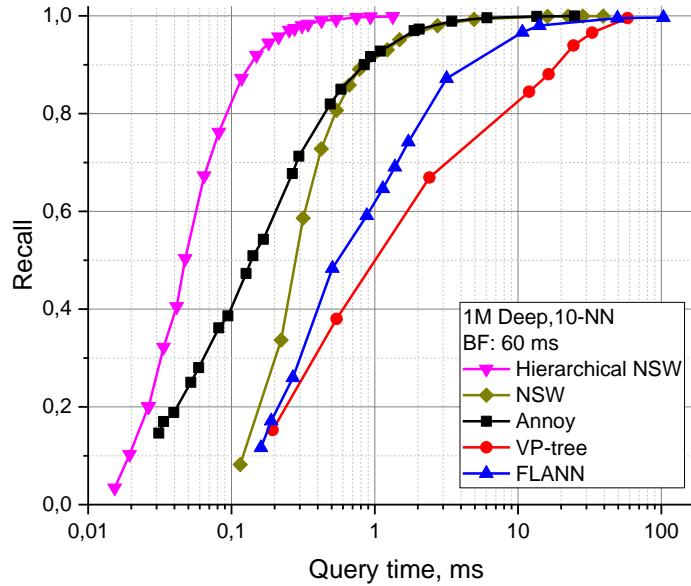
Hierarchical Navigable Small World (HNSW) – state of the art algorithm for ANNS



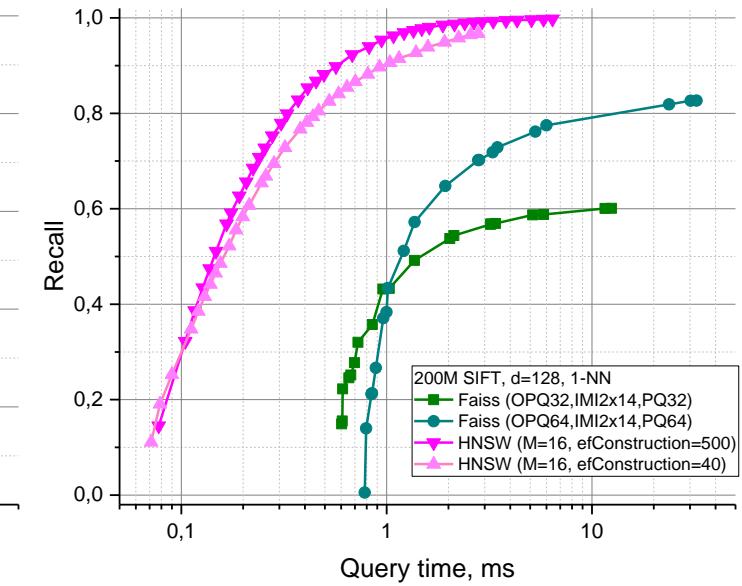
# Approximate Nearest Neighbor Search (ANNS)



Scaling for the number of distance computations



Comparison of the Hierarchical NSW with open source implementations



<https://arxiv.org/abs/1603.09320>

<https://github.com/yurymalkov/hnsw>

<https://github.com/erikbern/ann-benchmarks>

# K-Nearest Neighbors: Summary

In **Image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

Distance metric and  $K$  are **hyperparameters**

Choose hyperparameters using the **validation set**; only run on the test set once at the very end!

# References

- **Stanford courses (in English)**

CS231n: Convolutional Neural Networks for Visual Recognition

<http://cs231n.stanford.edu/>

Video lectures (2016)

<https://www.youtube.com/playlist?list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AIC>

CS224d: Deep Learning for Natural Language Processing

<http://cs224d.stanford.edu/>

Video lectures (2017)

[https://www.youtube.com/watch?v=OQQ-W\\_63UgQ&list=PL3FW7Lu3i5Jsnh1rnUwq\\_TcylNr7EkRe6](https://www.youtube.com/watch?v=OQQ-W_63UgQ&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6)

- **Articles and blogs**

arxiv.org

medium.com

# Assignments

- Install Anaconda3 with python 3.5
- Install PyCharm – python IDE
- Get started with Python
- Take a look at the CS231n assignment  
<http://cs231n.github.io/assignments2017/assignment1/>
- Do k-Nearest Neighbor classifier task  
knn.ipynb