Deep Learning (for Computer Vision)

Arjun Jain

Torch7

- Torch7 as the deep learning framework (Reason: we will discuss why in detail later)

Aim (by the end of this module)

- Understand the theory behind neural networks and its building blocks
- Be able to code using Torch7 to train models with your own data

Sources

A lot of the material has been shamelessly and gratefully collected from:

- **http://cs231n.stanford.edu/**
- https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training/
- https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html
- https://research.fb.com/learning-to-segment/
- https://research.fb.com/deep-learning-tutorial-at-cvpr-2014/
- https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/practicals/practical4.pdf
- http://torch.ch/docs/developer-docs.html
- https://github.com/torch/nn/blob/31d7d2bc86a914e2a9e6b3874c497c60517dc853/doc/module.md
- https://web.stanford.edu/group/pdplab/pdphandbook/handbookch6.html
- http://neuralnetworksanddeeplearning.com/chap2.html

A bit of history:

Hubel & Wiesel,

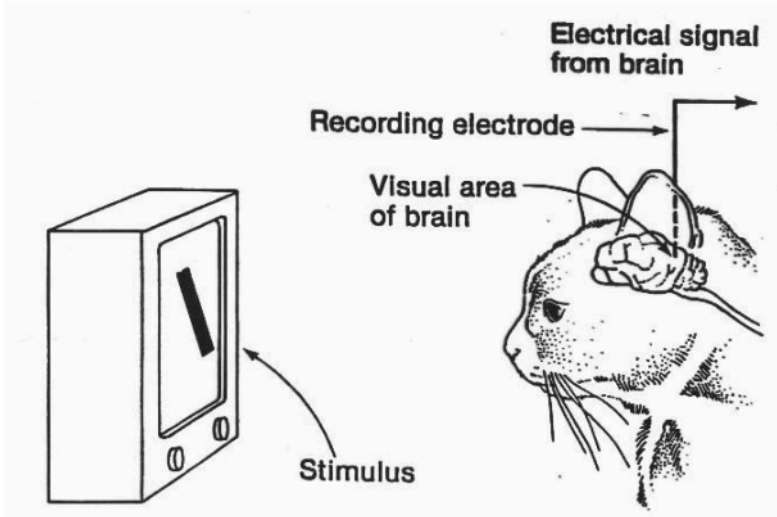
1959

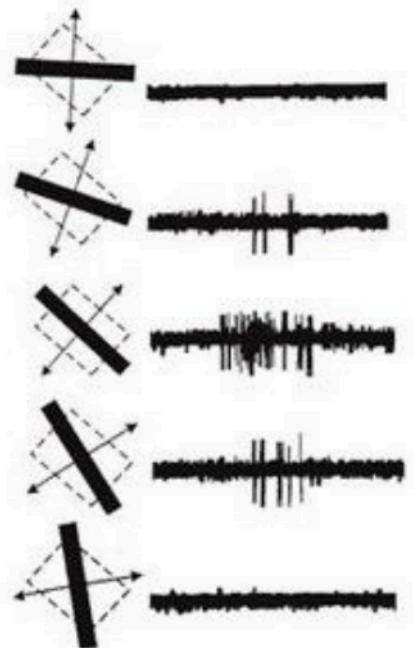
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

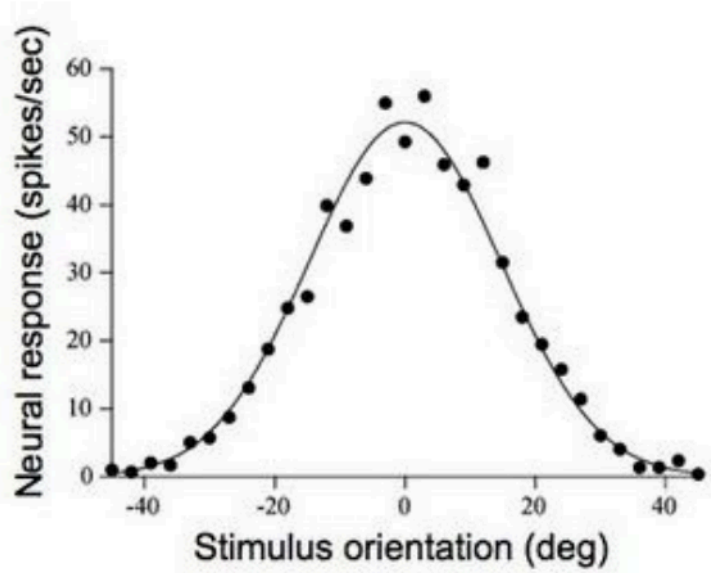
1962

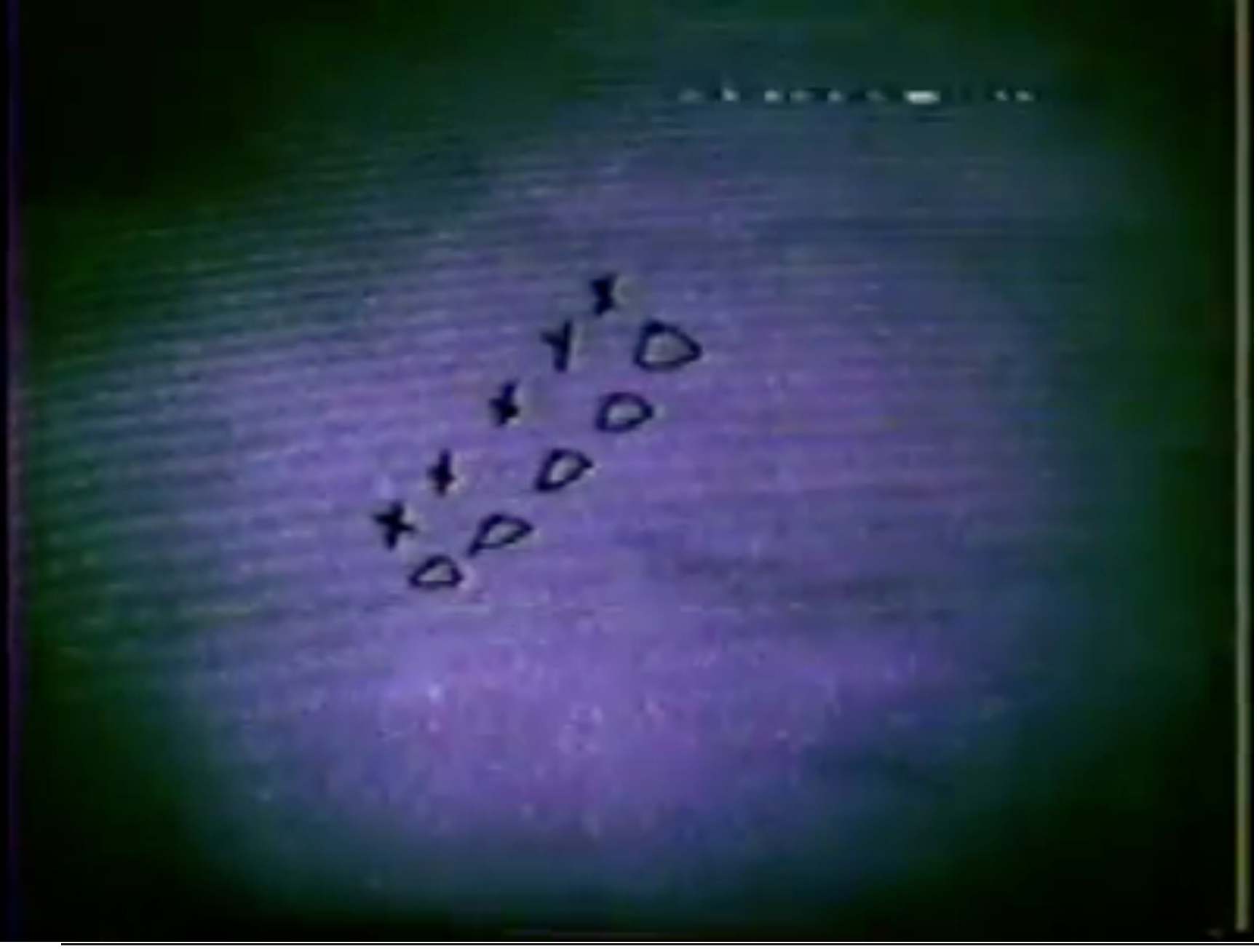
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...





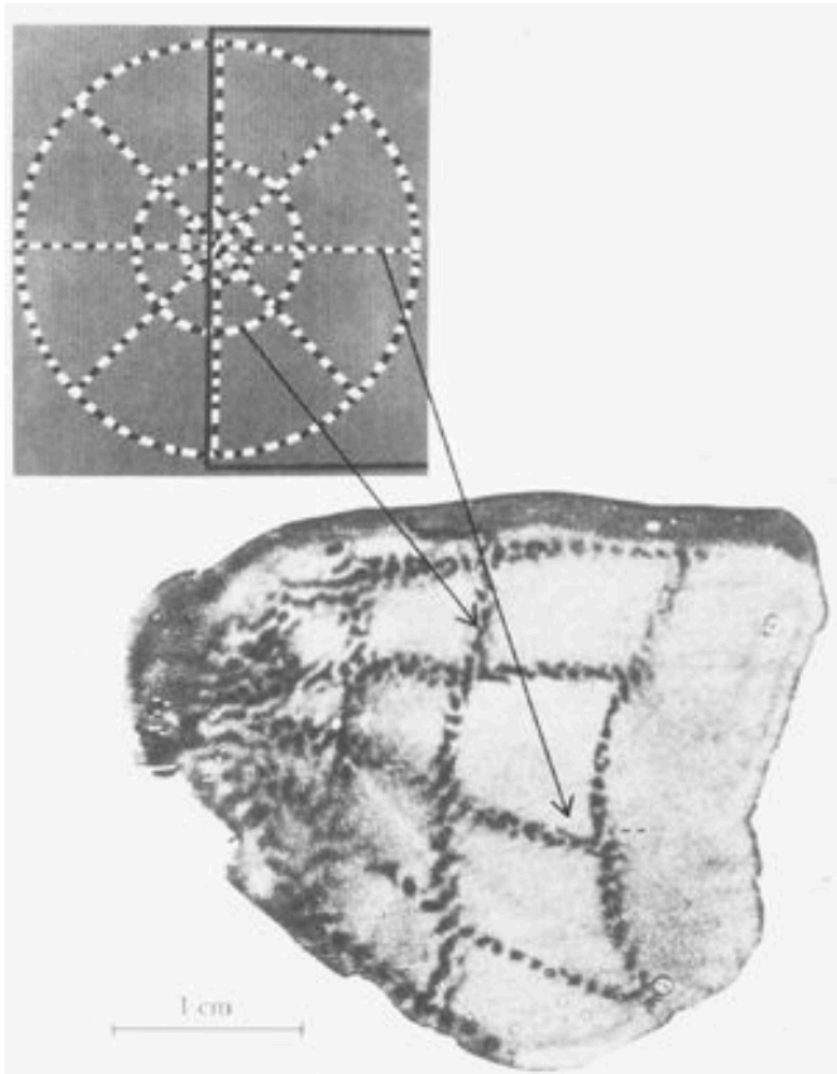




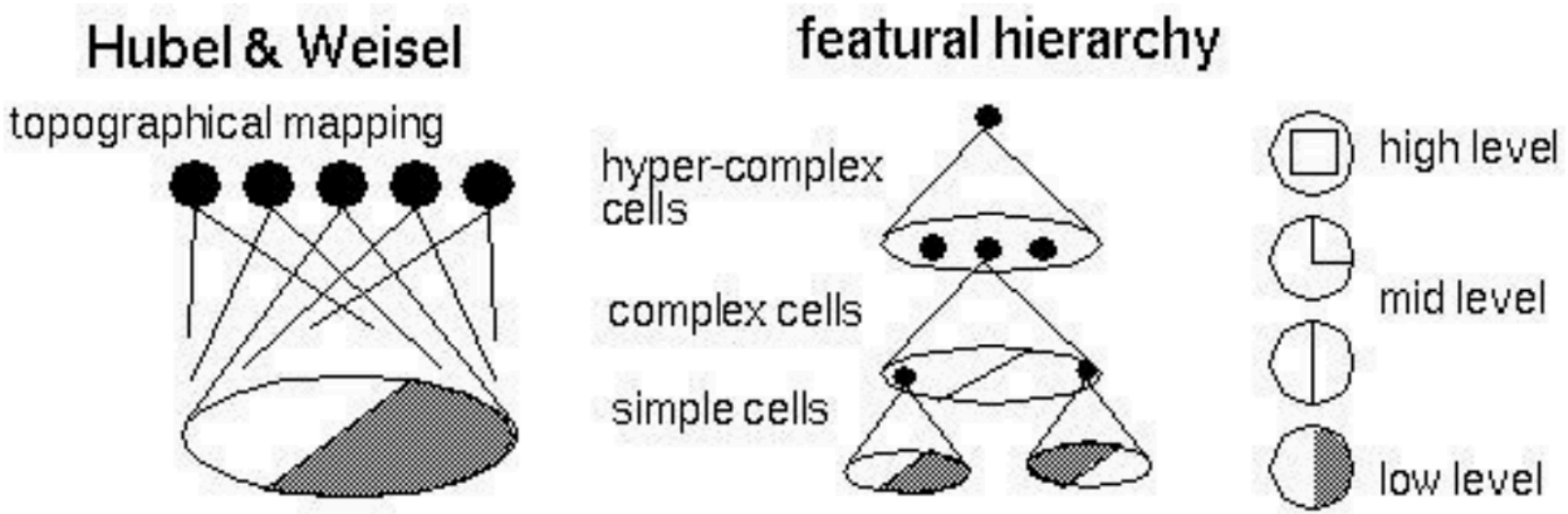
https://youtu.be/8VdFf3egwfg?t=1m10s

A bit of history

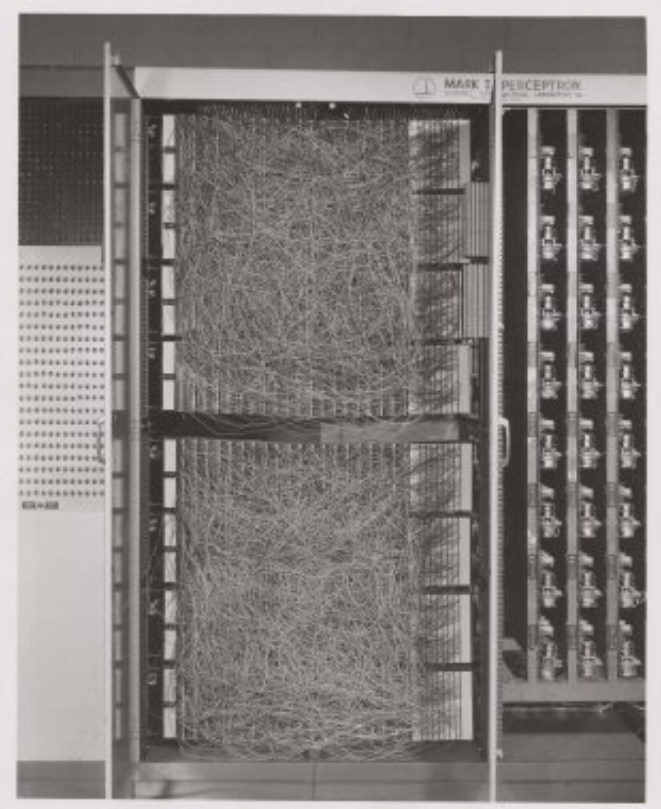
Topographical mapping in the cortex:
nearby cells in cortex represented
nearby regions in the visual field



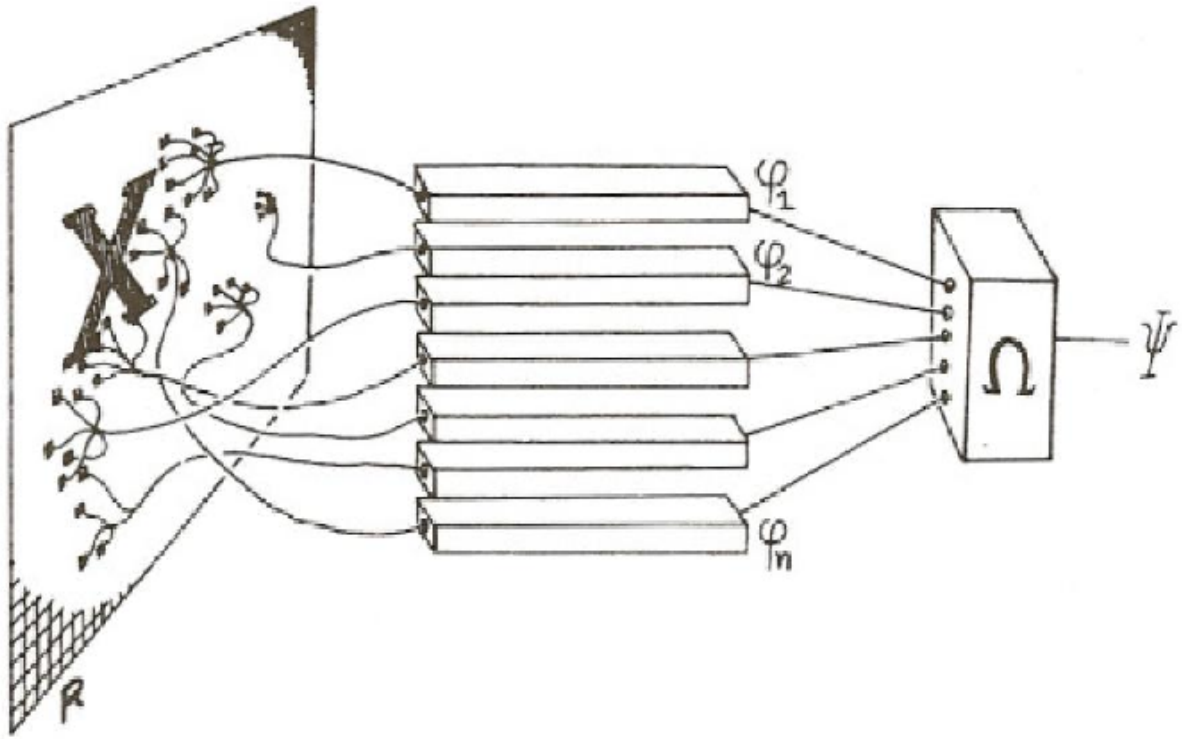
Hierarchical organization



Brief History – Mark I Perceptron – 1958



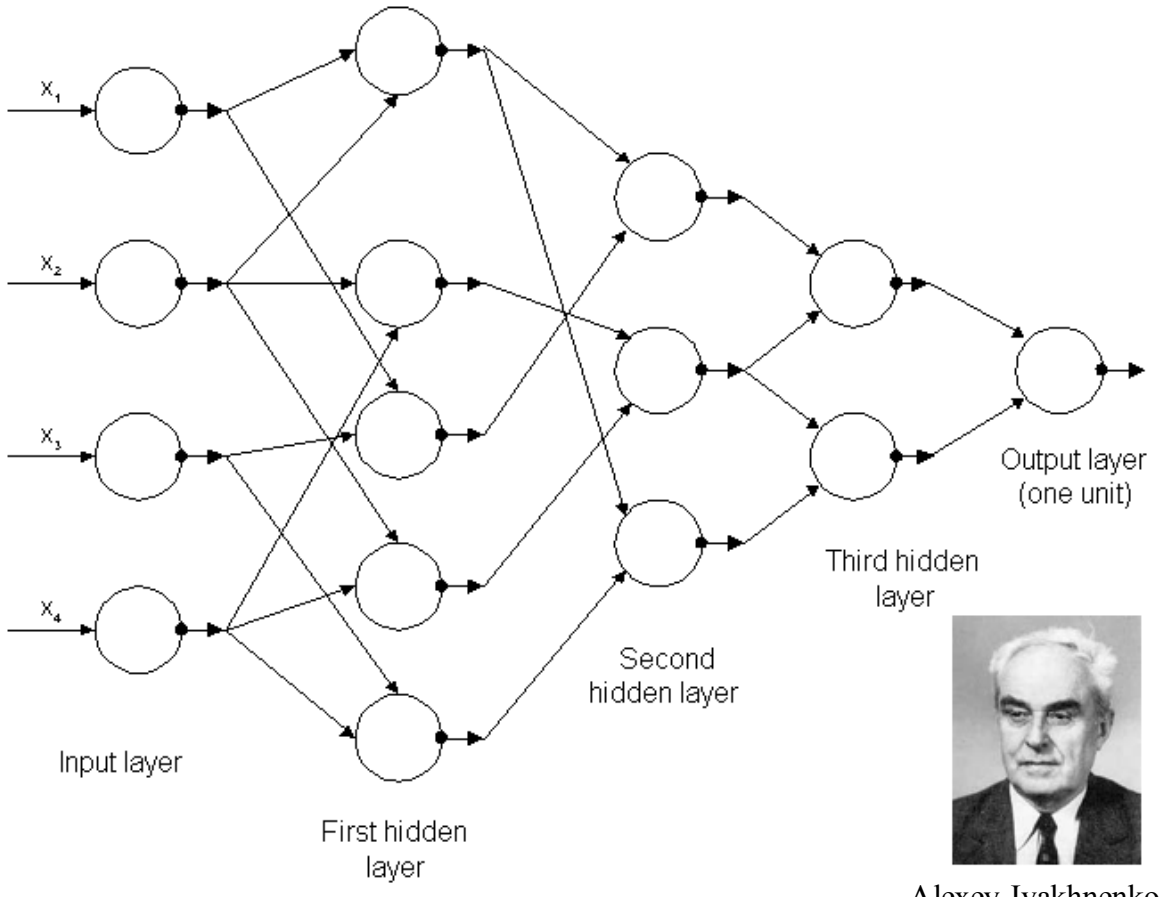
https://en.wikipedia.org/wiki/Perceptron



Perceptrons by M. L Minsky and S. Papert, 1969

Brief History – The First Deep Networks

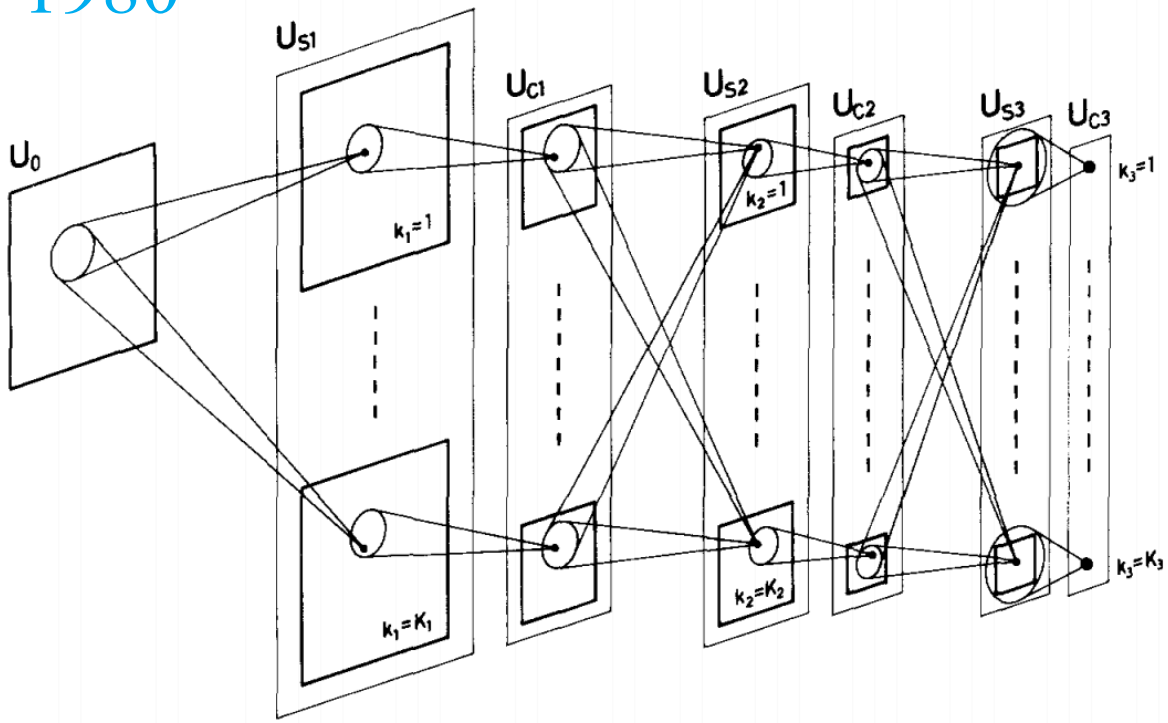
- Perceptron: single layer 1960s
- Multiple layers of non-linear features - Ivakhnenko and Lapa in 1965
- Thin but deep models with polynomial activation functions
- They did not use backpropagation

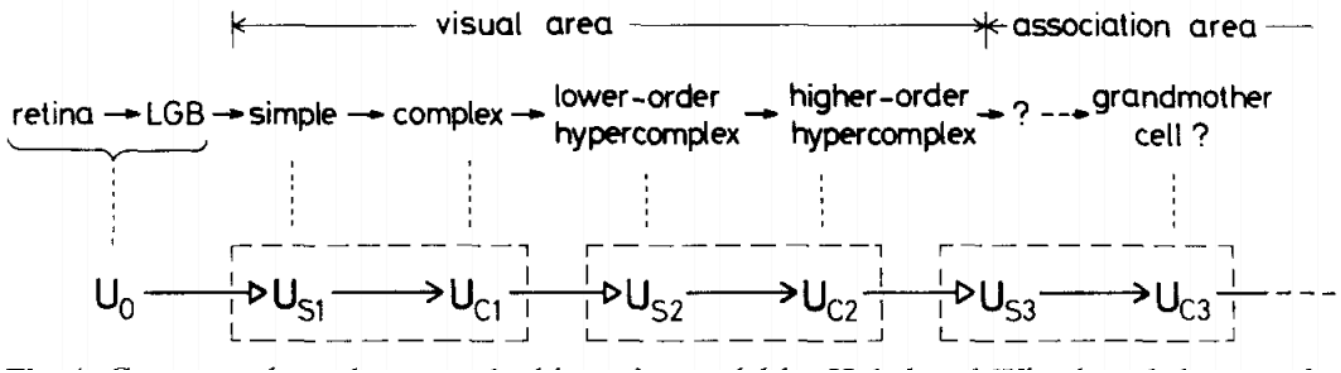


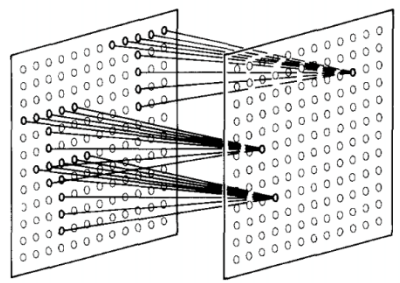
Alexey Ivakhnenko

Brief History – The First ConvNet - 1980

- Neocognitron: multiple convolutional and pooling layers similar to modern networks, but the network was trained by using a reinforcement scheme
- Did not still use backpropagation
- Translational invariant





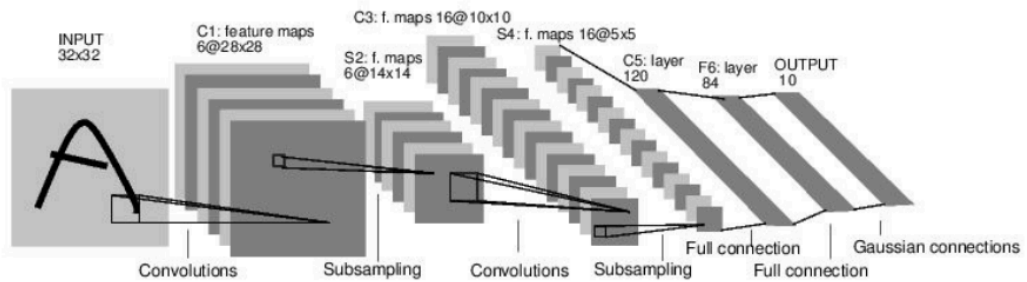


Kunihiko Fukushima

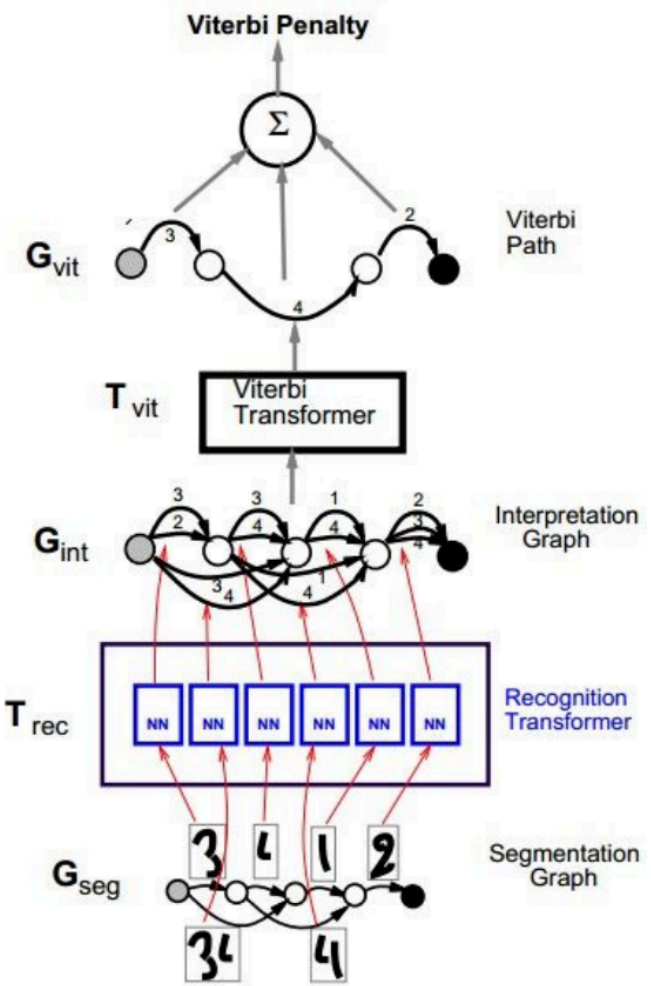
A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

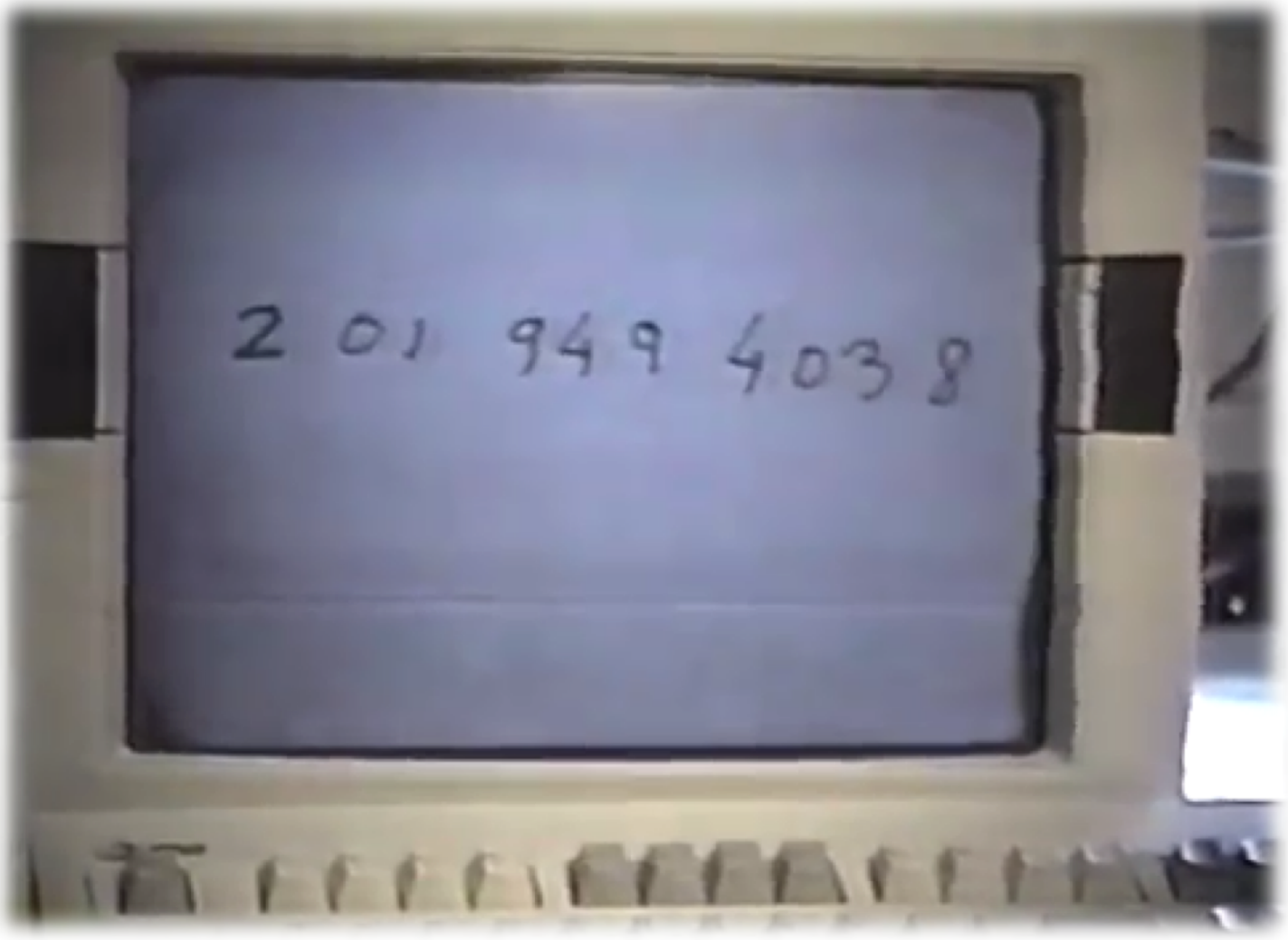


LeNet-5

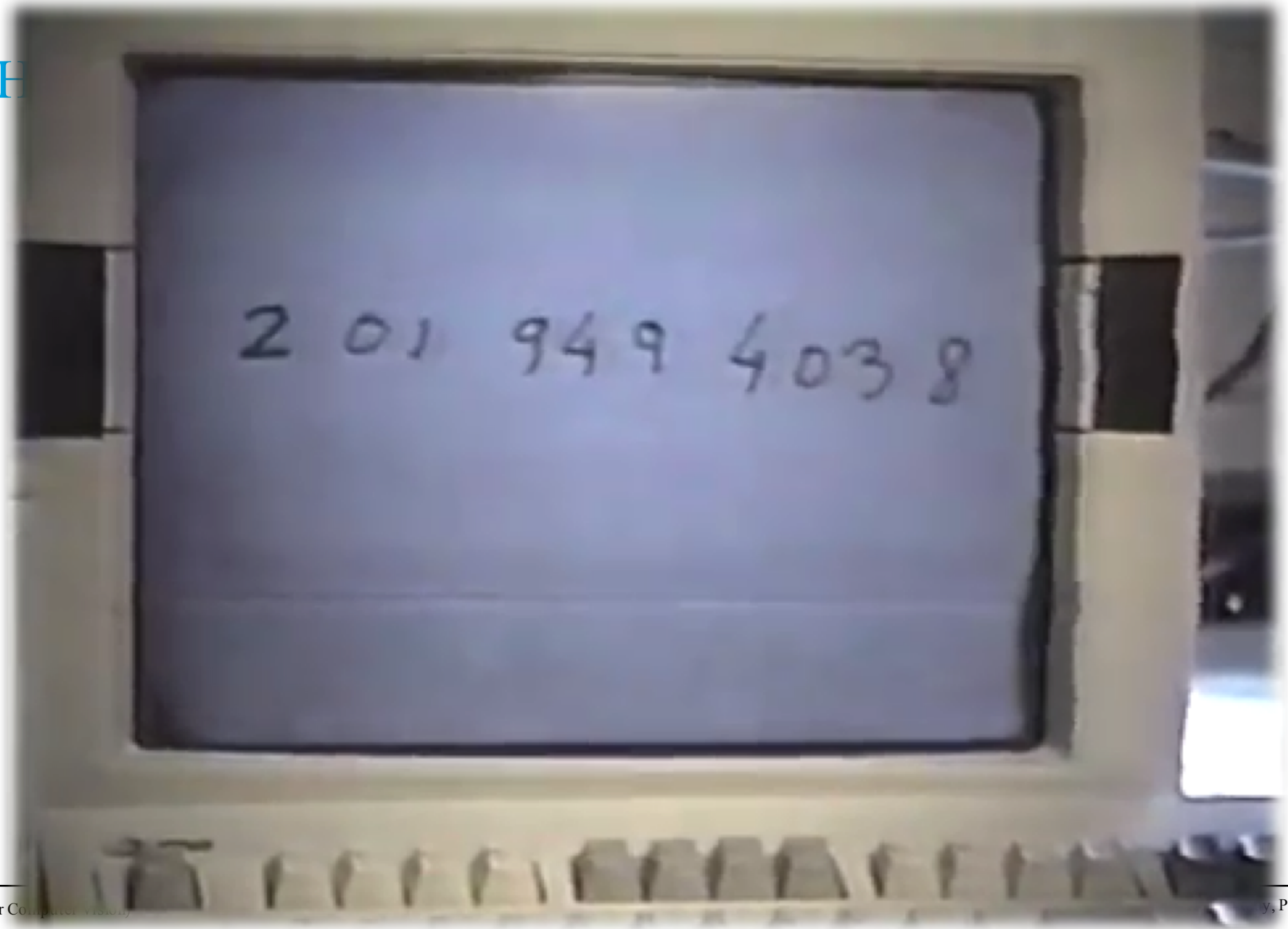


Yann LeCun

Brief History – LeNet-5 In Action



Brief H



Brief History – AI Winter

- Rapid advances led to a hype of artificial intelligence (similar to the buzz around deep learning today)
- Researchers made promises to solve AI and received lots of funding
- In the 1970s it became clear that those promises could not be kept, funding was cut dramatically
- The field of artificial intelligence dropped to near pseudo-science status
- Research became very difficult (little funding; publications almost never made it through peer review)
- Further advances such as SVMs with nice properties in terms of training, provable error bounds were preferred and took the front seat
- However, a handful of researchers continued further down this path

Brief History – AI Winter

- Those handful:

Geoffrey Hinton: University of Toronto & Google

Yann LeCun: New York University & Facebook

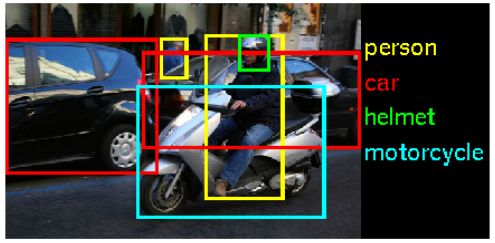
Yoshua Bengio: University of Montreal

Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

Andrew Ng: Stanford & Baidu

Brief History – The Tipping Point

- 2012 ILSVRC: ImageNet Large-Scale Visual Recognition Challenge – Annual World Cup of Computer Vision
- More than a million training images and 1000 categories



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

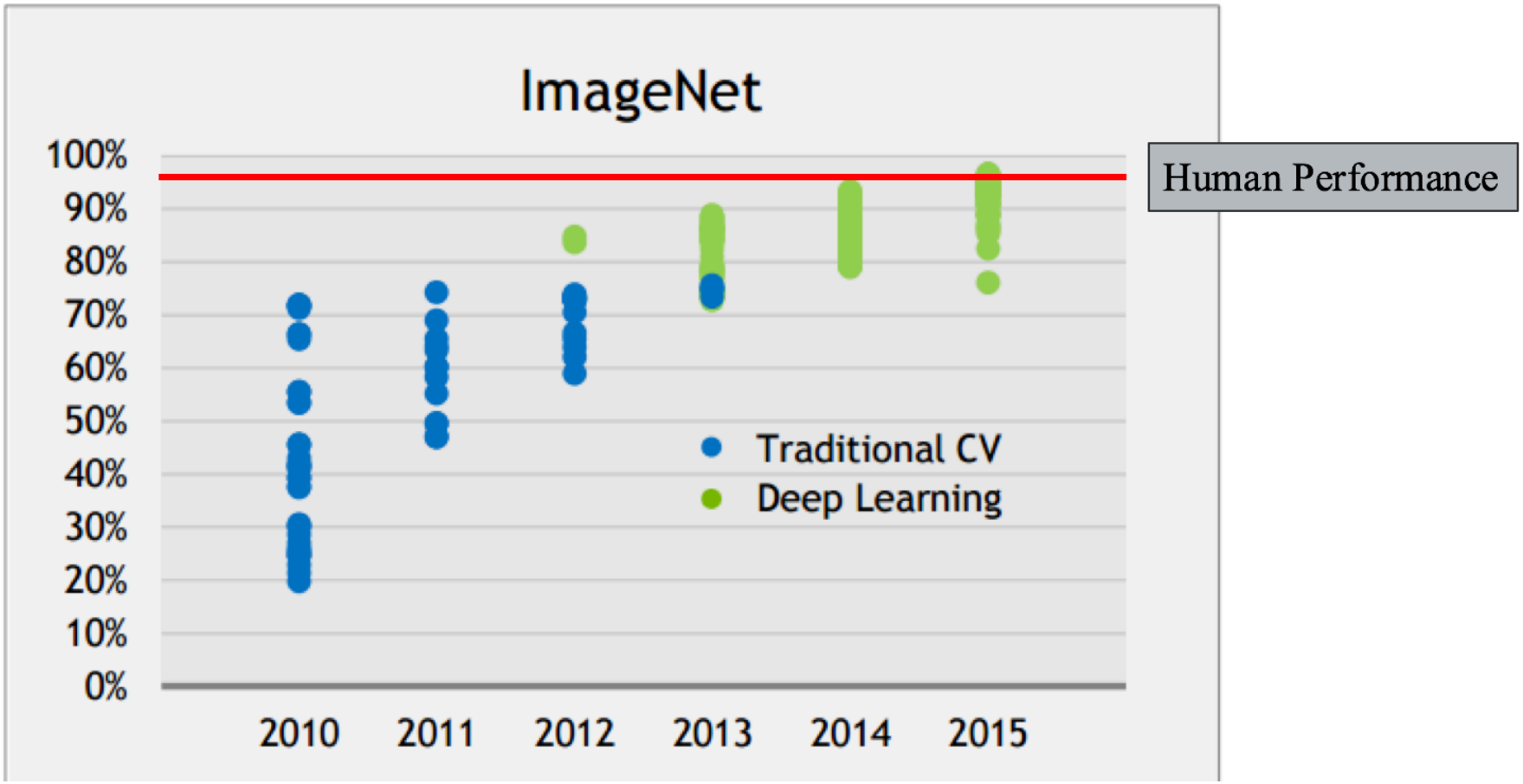
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Brief History – The Tipping Point

- Reported 15.4% Top 5 error rate. The next best entry achieved an error of 26.2%
- > 8000 citations (last year), by today >19000!
- The coming out party for CNNs in the computer vision community
- Shocked the computer vision community. Trained end-to-end on raw pixels, without using any feature engineering methods
- From here it was apparent that deep learning would take over computer vision and that other methods would not be able to catch up

Why ConvNets?

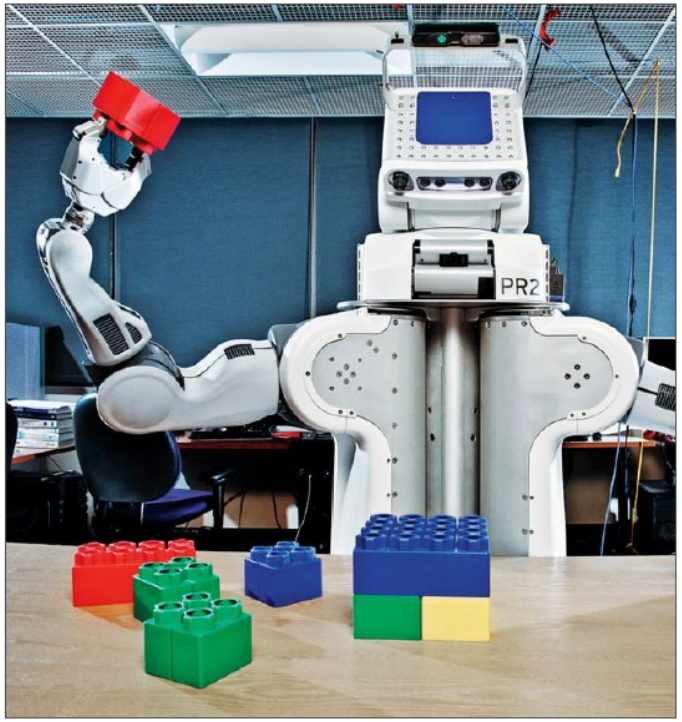


Brief History – So What Changed (since the 1970s)?

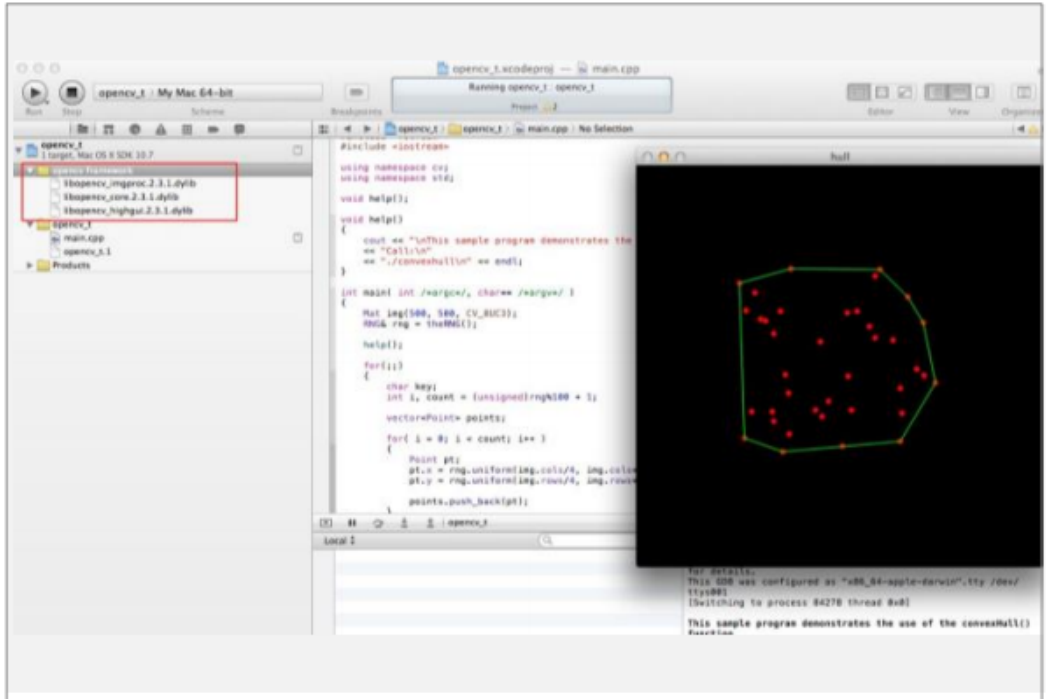
- Three things:
  - Availability of large amounts of labeled data e.g. ImageNet
  - Compute power – A single NVidia TITAN X card churns of 11 TFLOPS with ~3500 cores, **TITAN V**?
  - Algorithms:
    - ReLU - Found to decrease training time
    - Dropout – prevent overfitting to the training data

Deep Learning – Today – One Net To Rule Them All

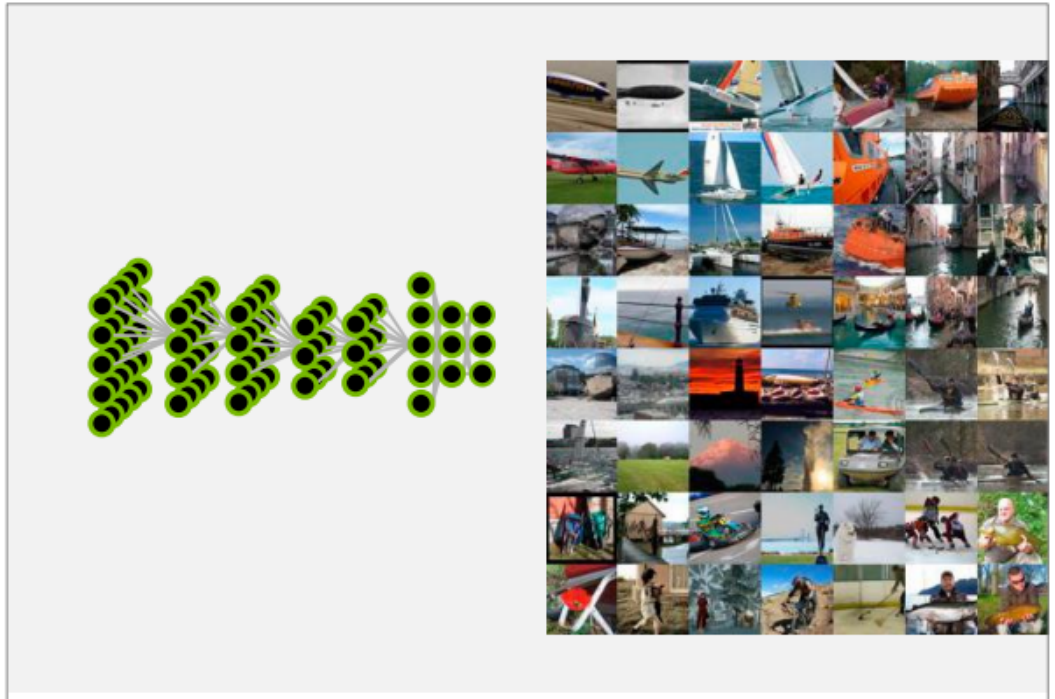
- Deep Learning == AI
- Solves problems previously unsolvable



A New Programming Model – Data Driven Paradigm



Traditional Computer Vision
Experts + Time



Deep Learning Object Detection
DNN + Data + HPC

Why Data Driven Paradigm?

- Consider Image Classification: a core task in Computer Vision

(assume given set of discrete labels)

{dog, cat, truck, plane, ...}

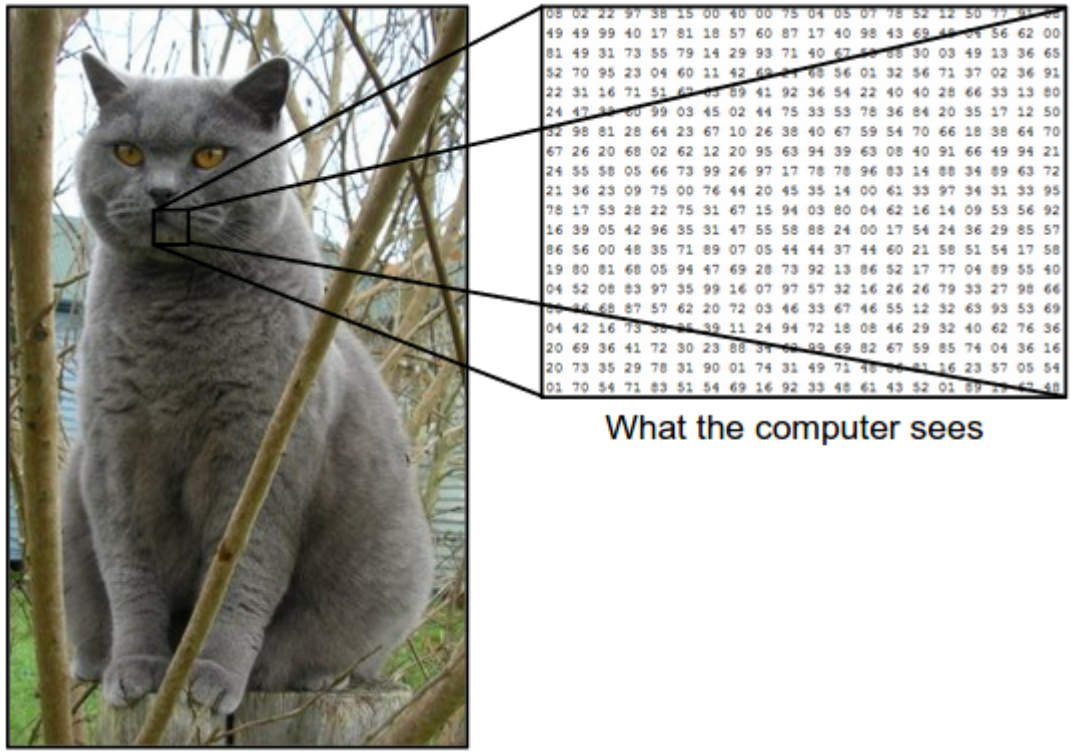
cat

Why Data Driven Paradigm?

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

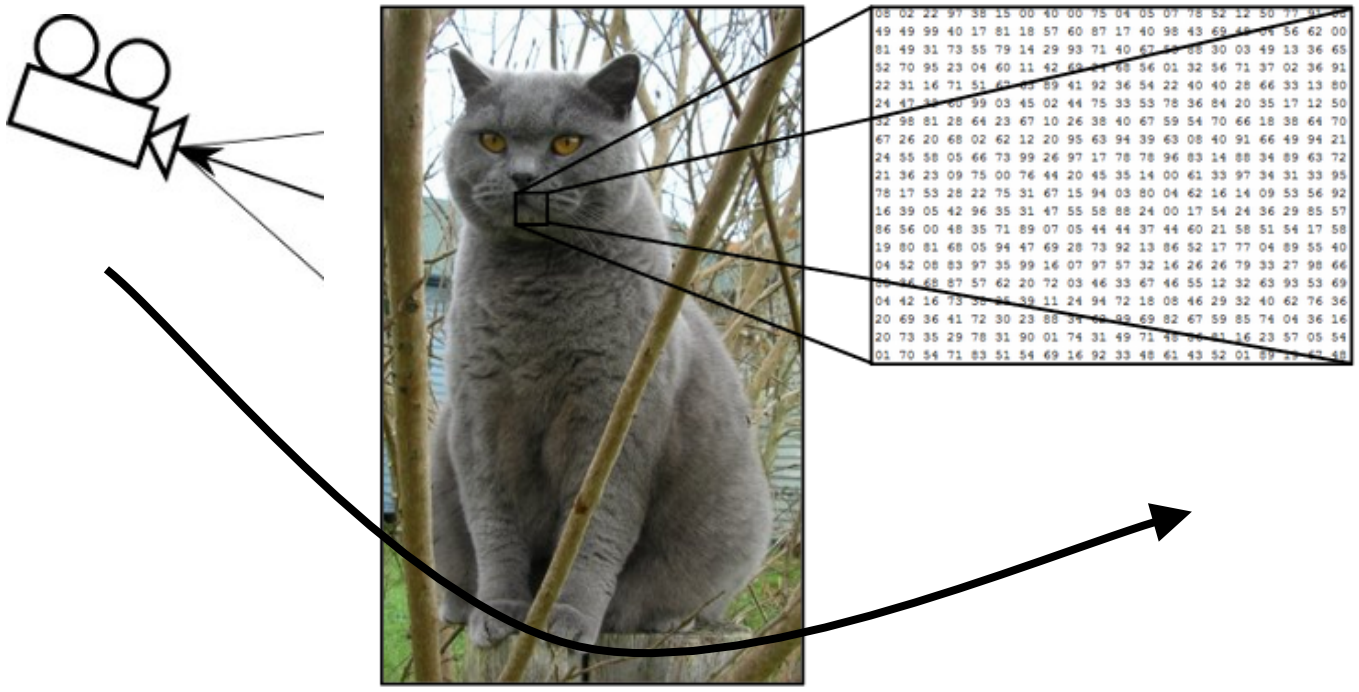
E.g.
300 x 100 x 3

(3 for 3 color channels RGB)

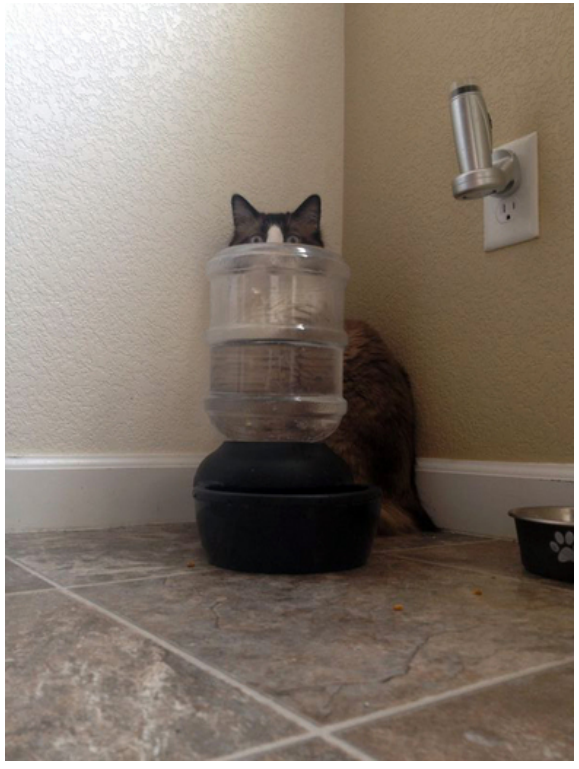


Why Data Driven Paradigm? – Invariant to Illumination

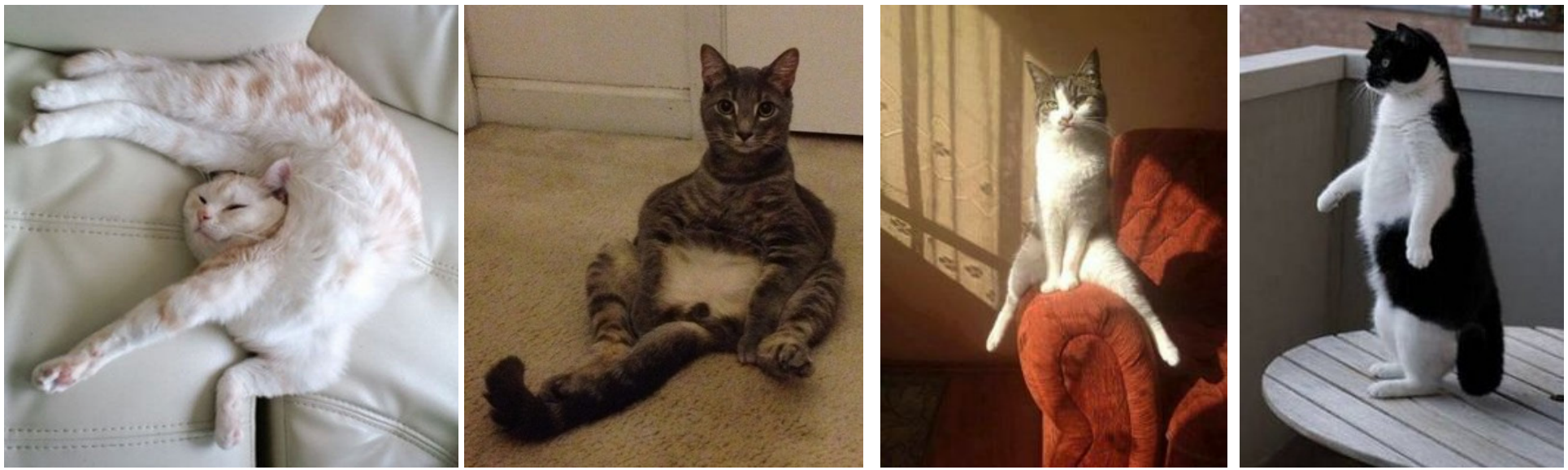
Why Data Driven Paradigm? – Invariant to Viewpoint



Why Data Driven Paradigm? – Deal with Occlusion



Why Data Driven Paradigm? – Invariant to Deformation



Why Data Driven Paradigm? – Deal with Background Clutter

Why Data Driven Paradigm? – Deal with Intra-class Variation

Why Data Driven Paradigm? – No Way To Hand Code It!

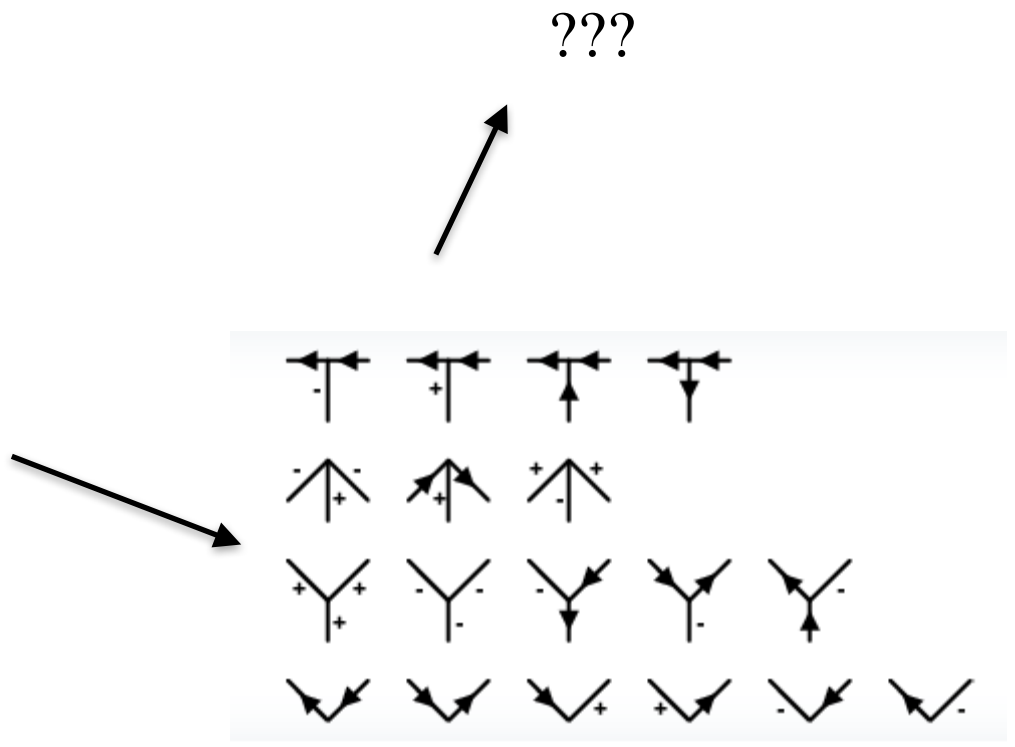
- Image classification:

```
function predict(image)
  -- ????
  return class_label
end
```

- Unlike e.g. sorting a list of numbers
- No obvious way to hard-code the algorithm for recognizing a cat, or other classes

Why Data Driven Paradigm? – People Have Attempted

- Image classification:



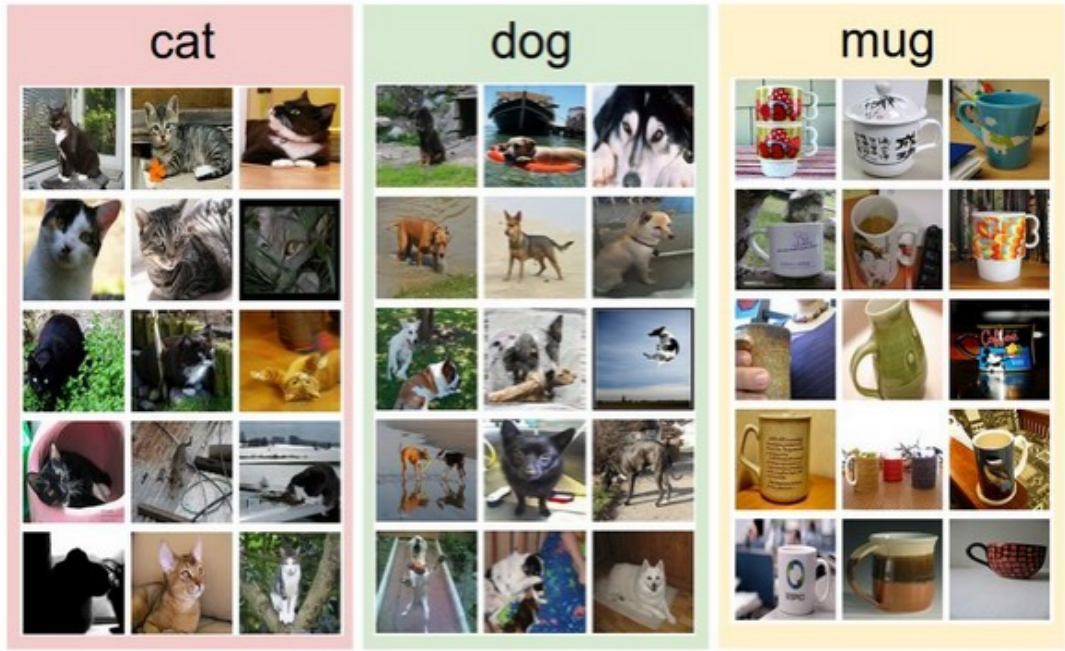
The Data Driven Paradigm

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

```
function train(train_images, train_labels)
  -- Build model: images -> labels
  return model
end


function predict(model, test_images)
  -- Predict test_labels using the model
  return test_labels
end
```

Example Training Set



Classifier 1: Nearest Neighbor Classifier

```
function train(train_images, train_labels)
  -- Build model: images -> labels
  return model
end
```

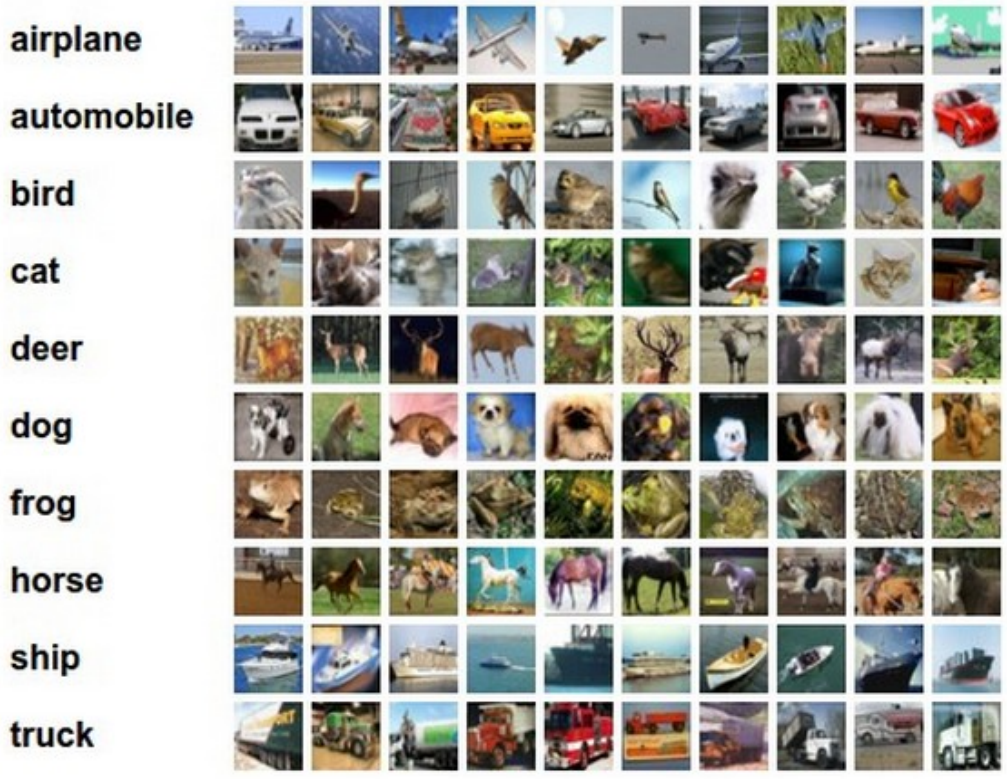
Remember all training images and their labels

```
function predict(model, test_images)
  -- Predict test_labels using the model
  return test_labels
end
```

Predict the label of the most similar training image

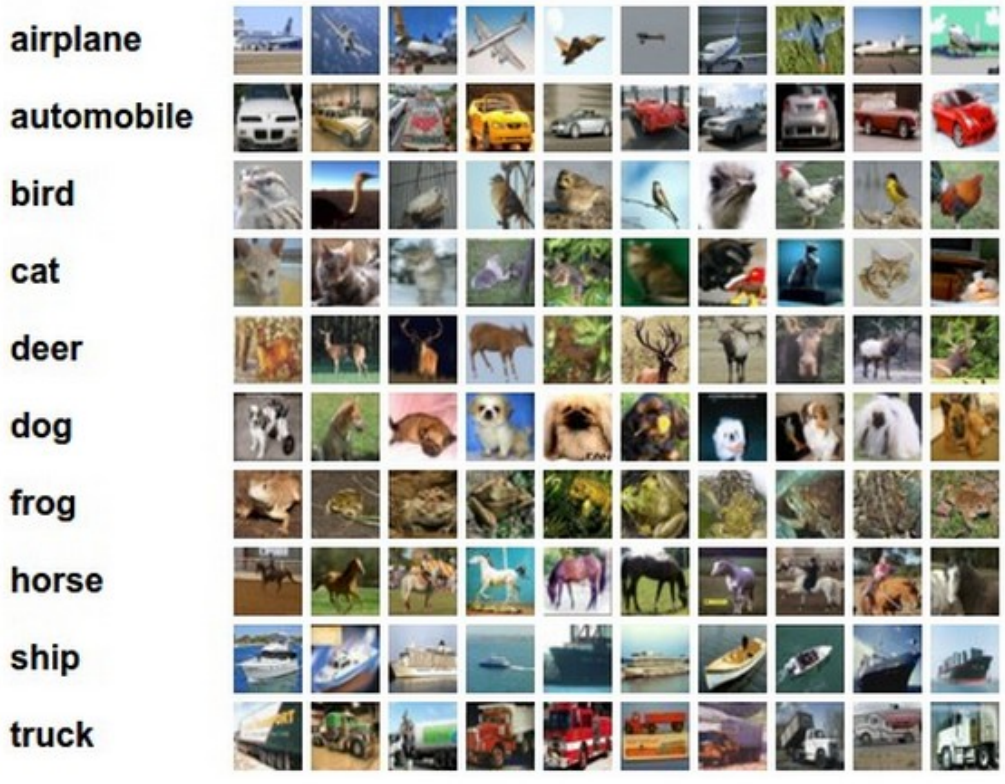
Example Dataset: CIFAR-10

10 labels
50,000 training images
10,000 test images.

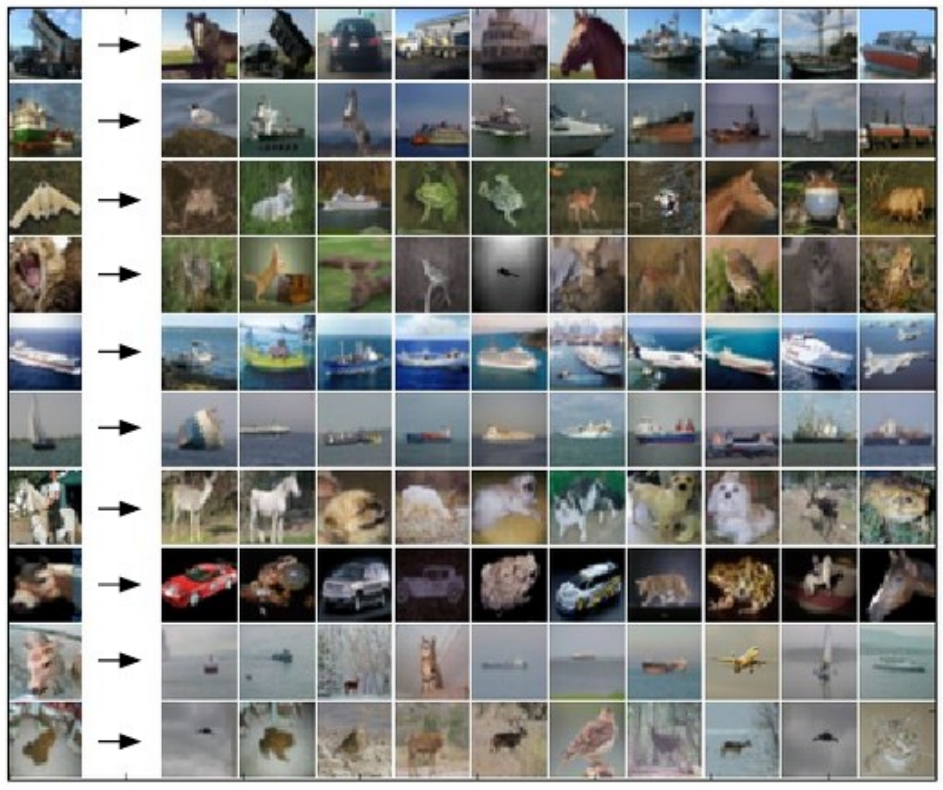


Example Dataset: CIFAR-10

10 labels
50,000 training images
10,000 test images.

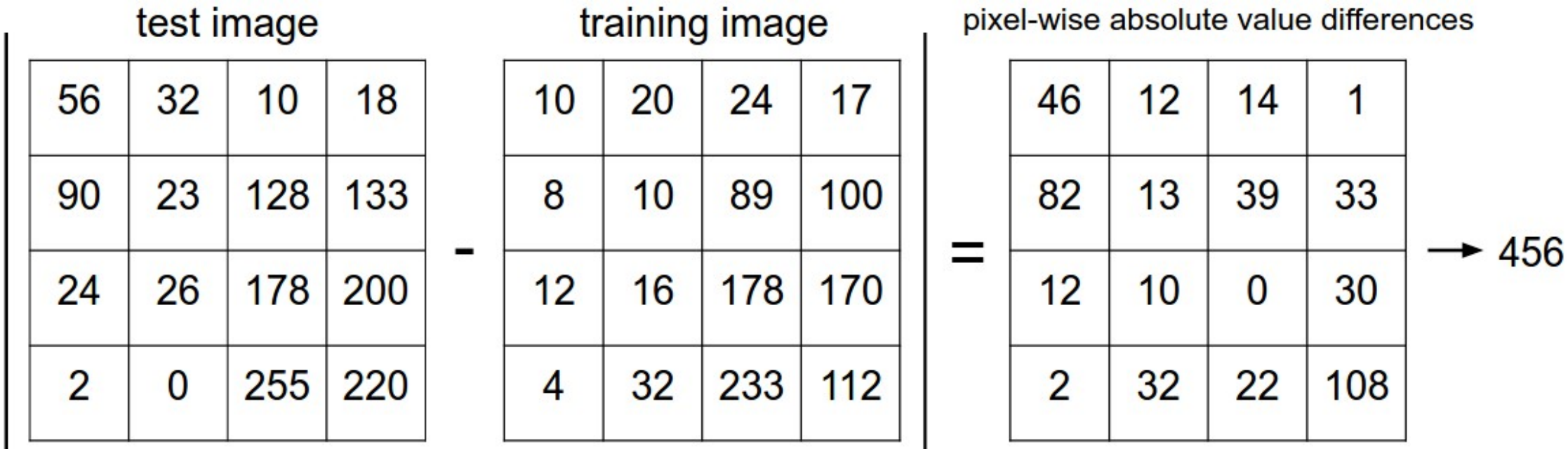


For every test image (first column), examples of nearest neighbors in rows



How Do We Compare the Images? What is the Distance Metric?

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



Nearest Neighbor Classifier

```
function predict(image)
  -- ????
  return class_label
end
```

Nearest Neighbor Classifier

In [4]:

```
local NN = torch.class("NN")

function NN:__init()
end

function NN:train(X, y)
  -- X is 2D of size N x D = 32x32x3, so each row is an example
  -- Y is 1D of size N
  self.tr_x = X
  self.tr_y = y
end

function NN:predict(x)
  -- x is of size D = 32x32x3 for which we want to predict the label
  -- returns the predicted label for the input x
  local min_idx = nil
  local min_dist = 1e10
  for i=1, self.tr_x:size(1) do
    local dist = (self.tr_x[i] - x):float():abs():sum()
    if (dist < min_dist) then
      min_dist = dist
      min_idx = i
    end
  end
  return self.tr_y[min_idx]
end
```

Nearest Neighbor Classifier

In [4]:

```
local NN = torch.class("NN")

function NN:__init()
end

function NN:train(X, y)
  -- X is 2D of size N x D = 32x32x3, so each row is an example
  -- Y is 1D of size N
  self.tr_x = X
  self.tr_y = y
end

function NN:predict(x)
  -- x is of size D = 32x32x3 for which we want to predict the label
  -- returns the predicted label for the input x
  local min_idx = nil
  local min_dist = 1e10
  for i=1, self.tr_x:size(1) do
    local dist = (self.tr_x[i] - x):float():abs():sum()
    if (dist < min_dist) then
      min_dist = dist
      min_idx = i
    end
  end
  return self.tr_y[min_idx]
end
```

remember the training data

Nearest Neighbor Classifier

In [4]:

```
local NN = torch.class("NN")

function NN:__init()
end

function NN:train(X, y)
  -- X is 2D of size N x D = 32x32x3, so each row is an example
  -- Y is 1D of size N
  self.tr_x = X
  self.tr_y = y
end

function NN:predict(x)
  -- x is of size D = 32x32x3 for which we want to predict the label
  -- returns the predicted label for the input x
  local min_idx = nil
  local min_dist = 1e10
  for i=1, self.tr_x:size(1) do
    local dist = (self.tr_x[i] - x):float():abs():sum()
    if (dist < min_dist) then
      min_dist = dist
      min_idx = i
    end
  end
  return self.tr_y[min_idx]
end
```

For the test image:

- find nearest train image with L1 distance
- predict the label of nearest training image

Nearest Neighbor Classifier

In [4]:
```
local NN = torch.class("NN")

function NN:__init()
end

function NN:train(X, y)
  -- X is 2D of size N x D = 32x32x3, so eac
  -- Y is 1D of size N
  self.tr_x = X
  self.tr_y = y
end

function NN:predict(x)
  -- x is of size D = 32x32x3 for which we w
  -- returns the predicted label for the inp
  local min_idx = nil
  local min_dist = 1e10
  for i=1, self.tr_x:size(1) do
    local dist = (self.tr_x[i] - x):float():a
    if (dist < min_dist) then
      min_dist = dist
      min_idx = i
    end
  end
  return self.tr_y[min_idx]
end
```

In [5]:
```
classifier = NN.new()
classifier:train(tr_x, tr_y)

itorch.image(te_x[100])
print (classifier:predict(te_x[100]))

itorch.image(te_x[110])
print (classifier:predict(te_x[110]))

itorch.image(te_x[120])
print (classifier:predict(te_x[120]))

itorch.image(te_x[130])
print (classifier:predict(te_x[130]))

itorch.image(te_x[140])
print (classifier:predict(te_x[140]))
```

Out[5]: 7

Out[5]: 7

Out[5]: 7

Nearest Neighbor Classifier

In [4]:

```
local NN = torch.class("NN")

function NN:__init()
end

function NN:train(X, y)
  -- X is 2D of size N x D = 32x32x3, so each row is an example
  -- Y is 1D of size N
  self.tr_x = X
  self.tr_y = y
end

function NN:predict(x)
  -- x is of size D = 32x32x3 for which we want to predict the label
  -- returns the predicted label for the input x
  local min_idx = nil
  local min_dist = 1e10
  for i=1, self.tr_x:size(1) do
    local dist = (self.tr_x[i] - x):float():abs():sum()
    if (dist < min_dist) then
      min_dist = dist
      min_idx = i
    end
  end
  return self.tr_y[min_idx]
end
```

Q: how does the classification speed depend on the size of the training data?

Nearest Neighbor Classifier

In [4]:
```
local NN = torch.class("NN")

function NN:__init()
end

function NN:train(X, y)
  -- X is 2D of size N x D = 32x32x3, so each row is an example
  -- Y is 1D of size N
  self.tr_x = X
  self.tr_y = y
end

function NN:predict(x)
  -- x is of size D = 32x32x3 for which we want to predict the label
  -- returns the predicted label for the input x
  local min_idx = nil
  local min_dist = 1e10
  for i=1, self.tr_x:size(1) do
    local dist = (self.tr_x[i] - x):float():abs():sum()
    if (dist < min_dist) then
      min_dist = dist
      min_idx = i
    end
  end
  return self.tr_y[min_idx]
end
```

Q: how does the classification speed depend on the size of the training data?
linearly :(

This is **backwards**:
- test time performance is usually much more important in practice.
- CNNs flip this: expensive training, cheap test evaluation

The Choice of Distance is a **Hyperparameter**

Common choices:

L1 distance

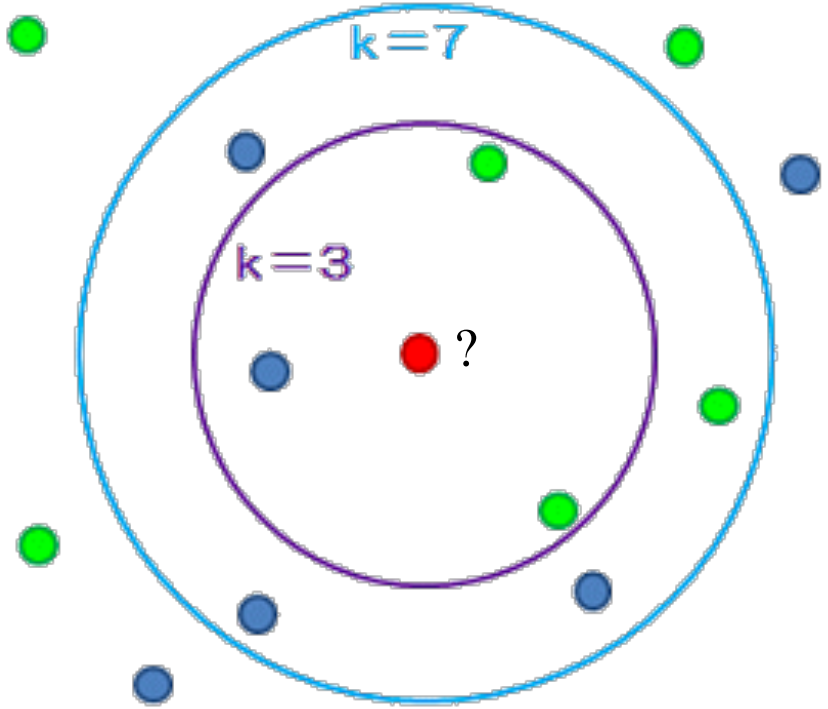
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

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

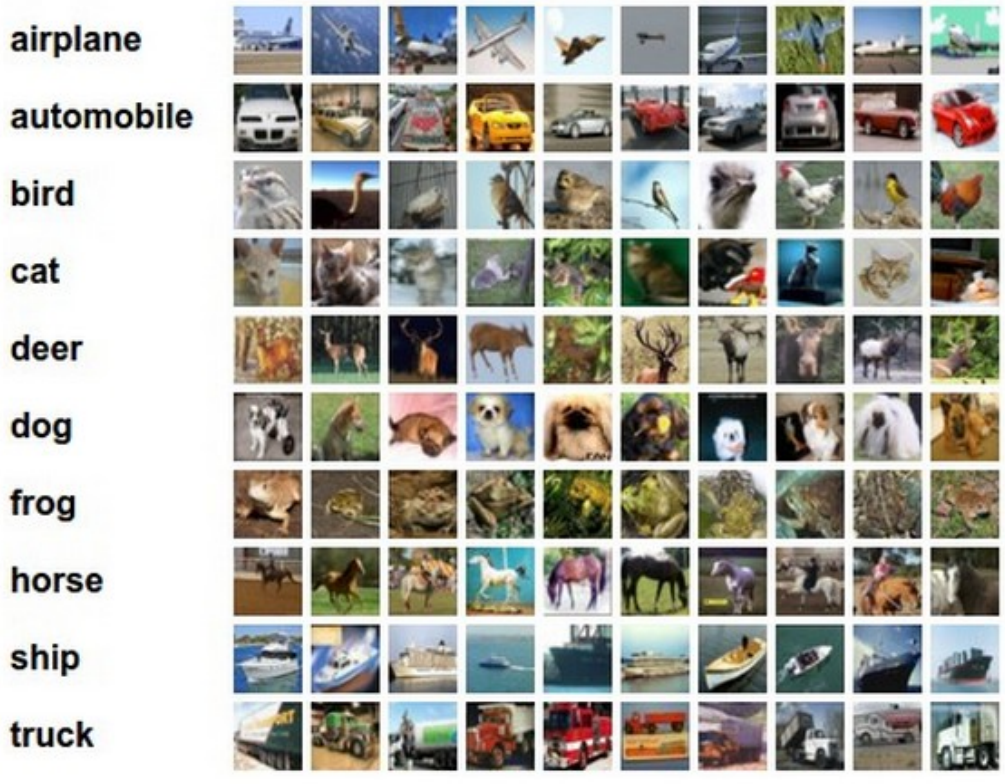
K-Nearest Neighbor Algorithm

Find the k nearest images, have them vote on the label

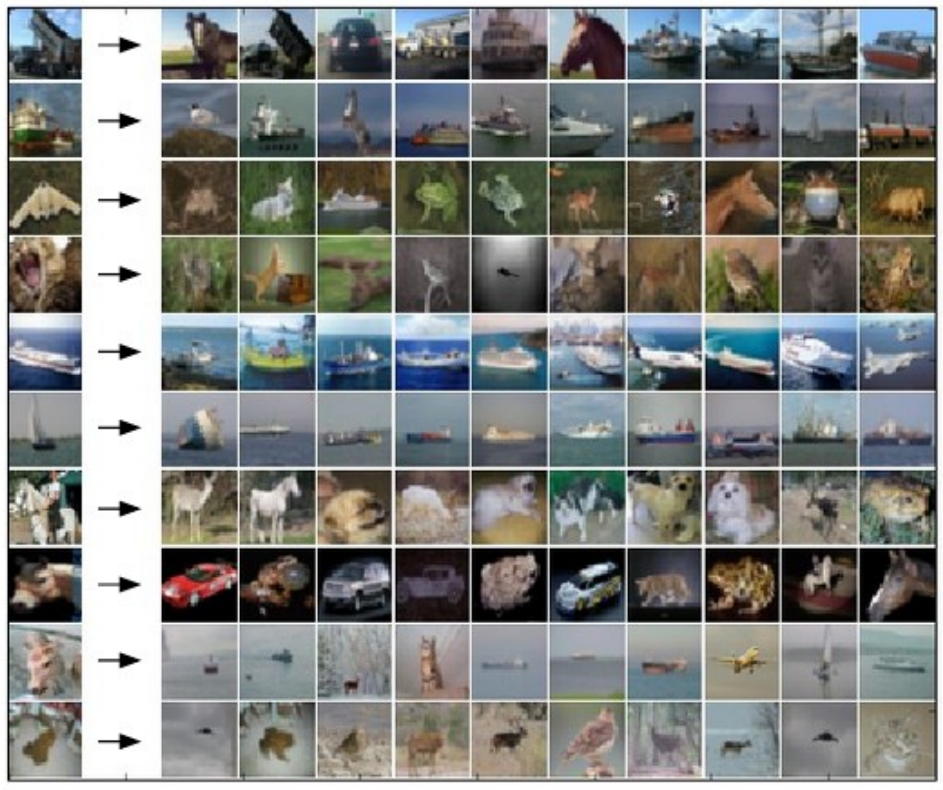


Example Dataset: CIFAR-10

10 labels
50,000 training images
10,000 test images.

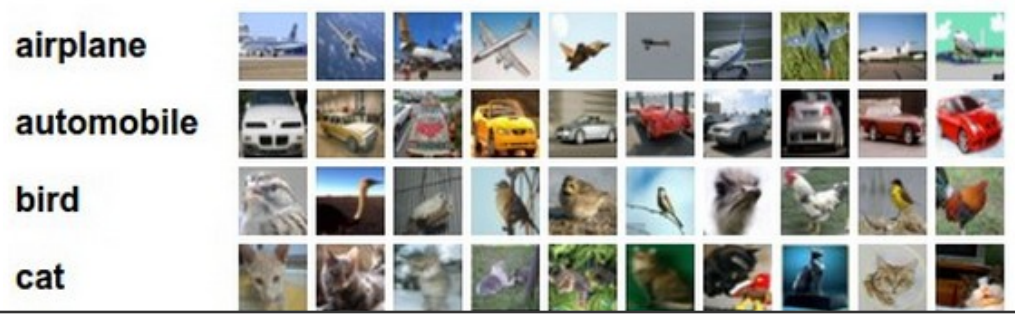


For every test image (first column), examples of nearest neighbors in rows

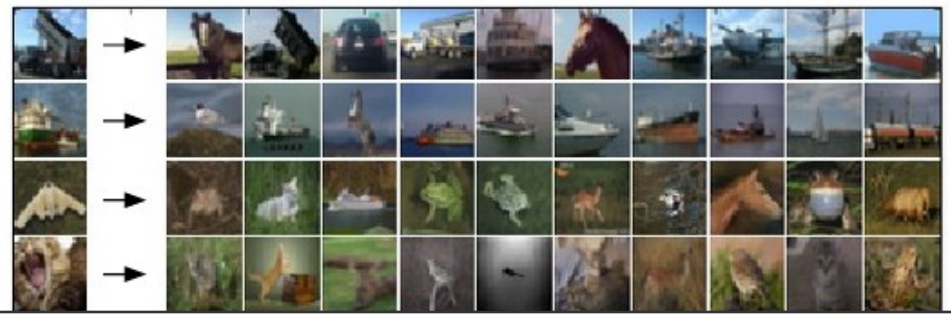


Example Dataset: CIFAR-10

10 labels
50,000 training images
10,000 test images.



For every test image (first column),
examples of nearest neighbors in rows



Q1: What is the accuracy of the nearest neighbor classifier on the **test** data, when using the Euclidean distance? What about L1 distance?

Example Dataset: CIFAR-10

10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows

Q2: What is the accuracy of the **k**-nearest neighbor classifier on the **test** data? What is the best value of **k**?

How Do We Set the **Hyperparameters**?

What is the best distance to use?

What is the best value of k to use?

How Do We Set the **Hyperparameters**?

What is the best distance to use?

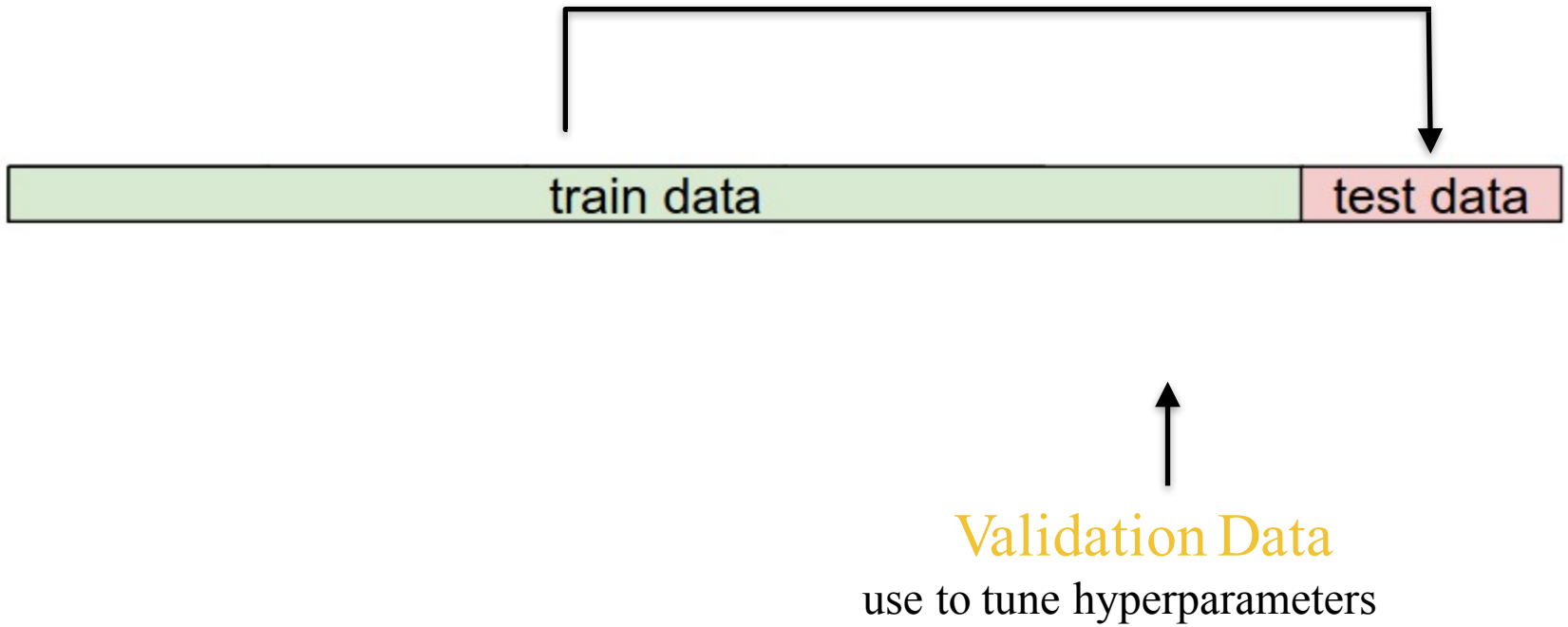
What is the best value of k to use?

Very problem-dependent.

Must try them all out and see what works best.

Try Out What Hyperparameters that Work Best on **Test Set**

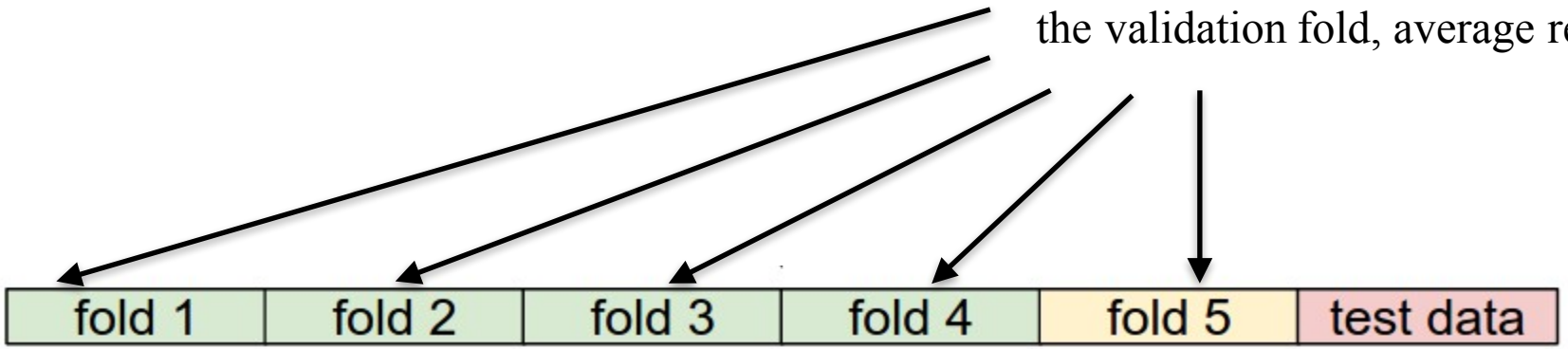
Bad idea (unless trying to win a competition where test set is given).
The test set is a proxy for the generalization performance!
Use only **VERY SPARINGLY,** at the end.



Try Out What Hyperparameters that Work Best on **Test Set**

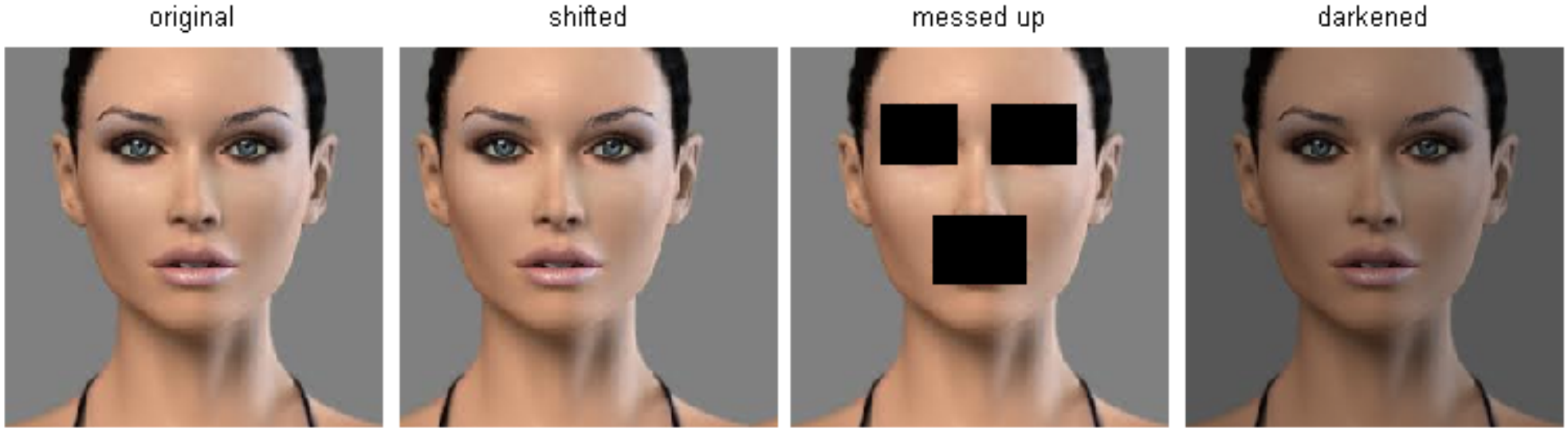
Cross-validation

cycle through the choice of which fold is the validation fold, average results.



K-Nearest Neighbor on Images Never Used

- Terrible performance at test time
- Distance metrics on level of whole images can be very unintuitive



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

K-Nearest Neighbor on Images Never Used

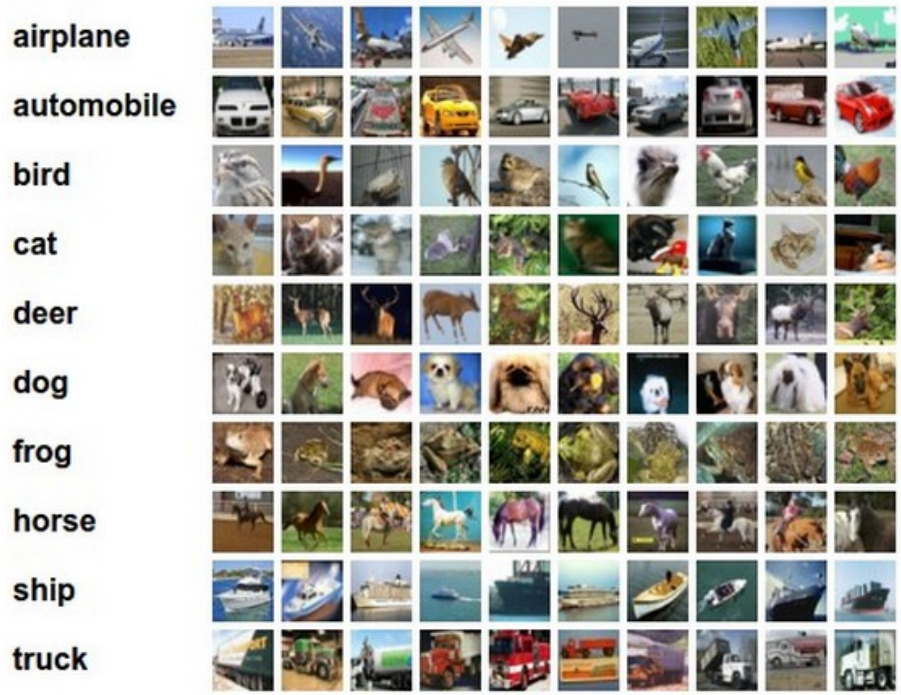
- **Image Classification:** We are given a **Training Set** of labeled images, asked to predict labels on **Test Set**. Common to report the **Accuracy** of predictions (fraction of correctly predicted images)
- We introduced the **k-Nearest Neighbor Classifier**, which predicts the labels based on nearest images in the training set
- We saw that the choice of **distance** and the value of are **hyperparameters** that are tuned using a validation set, or through **cross-validation** if the size of the dat**k**a is small.
- Once the best set of hyperparameters is chosen, the classifier is evaluated once on the test set, and reported as the performance of kNN on that data.

K-Nearest Neighbor on Images Never Used

- k-NN classifier:
  - must remember all of the training data and store it for future comparisons with the test data
  - This is space inefficient because datasets may easily be gigabytes in size
  - Classifying a test image is expensive since it requires a comparison to all training images

Parametric Approach: CIFAR-10

10 labels
50,000 training images
10,000 test images
each image is an array of size **32 x 32 x 3 = 3072** numbers total



Parametric Approach

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Image

$f(x, W)$

10 numbers, indicating class scores

parameters, or "weights"

Parametric Approach: Linear Classifier

[32x32x3]
array of numbers 0...1
(3072 numbers total)

10x1

3072x1

$f(x, W) = Wx$

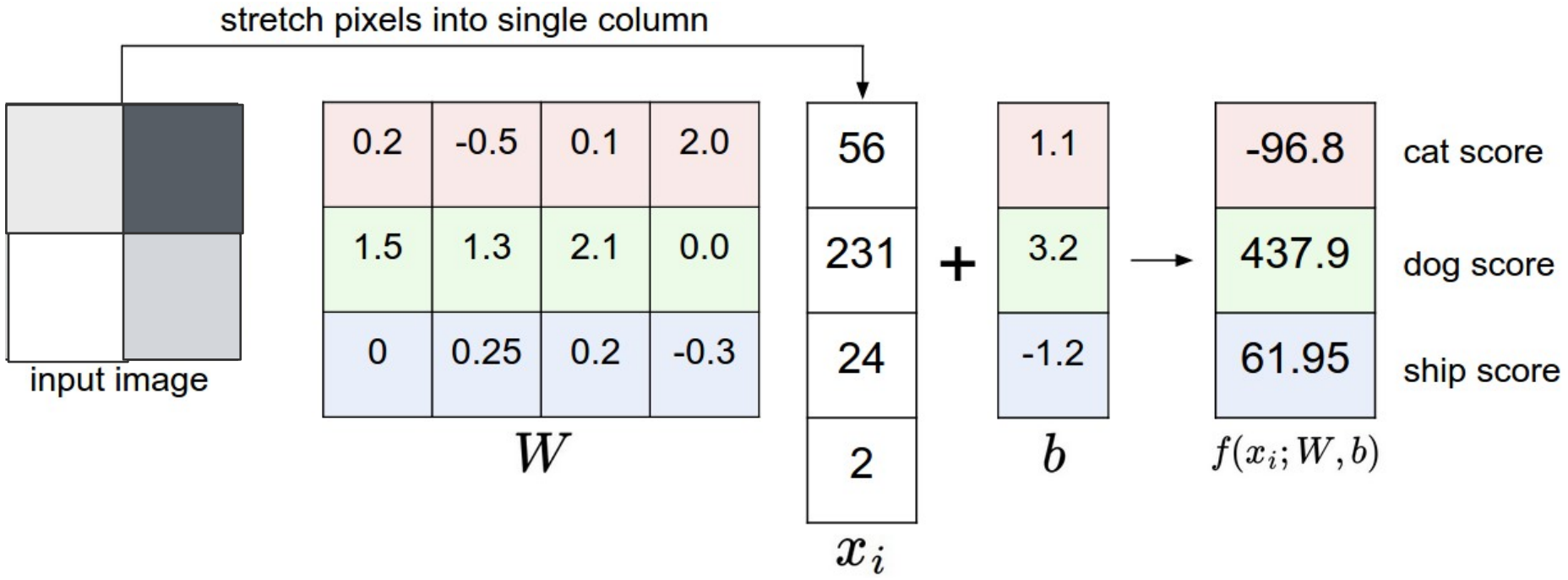
(+b) 10x1

10x3072

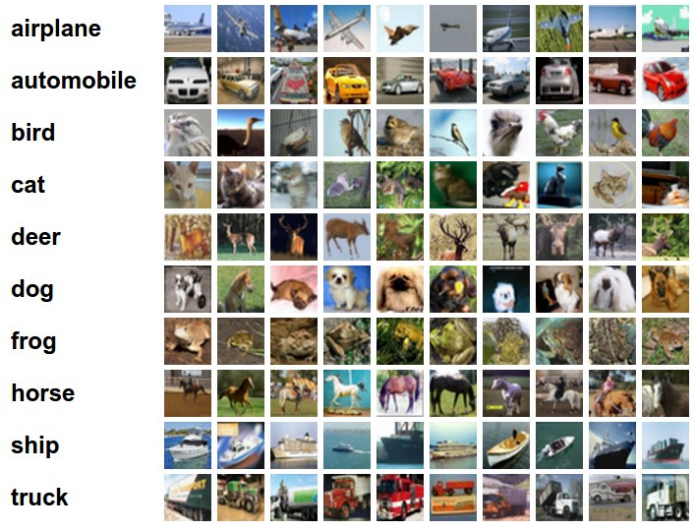
10 numbers, indicating class scores

parameters, or "weights"

Example with an Image with 4 Pixels, and 3 Classes (**cat**/**dog**/**ship**)

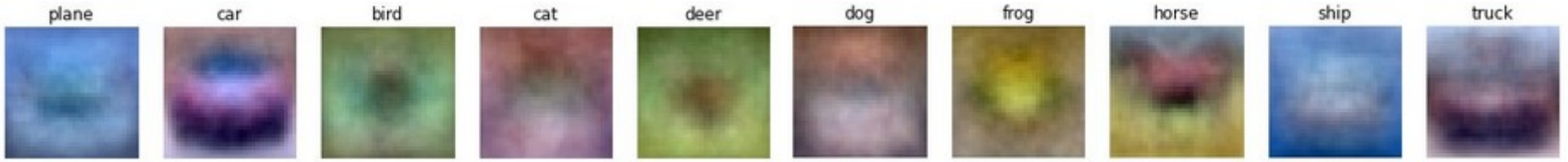


Interpreting a Linear Classifier

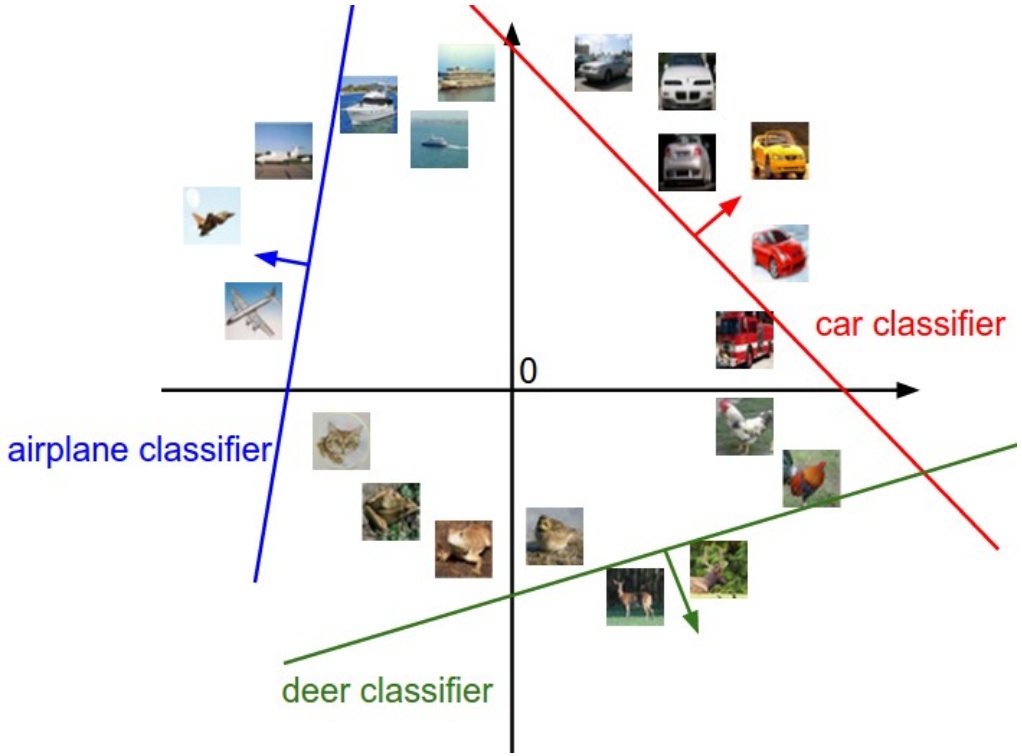


$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:



Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Thank you!