Deep Learning

Lecture 2

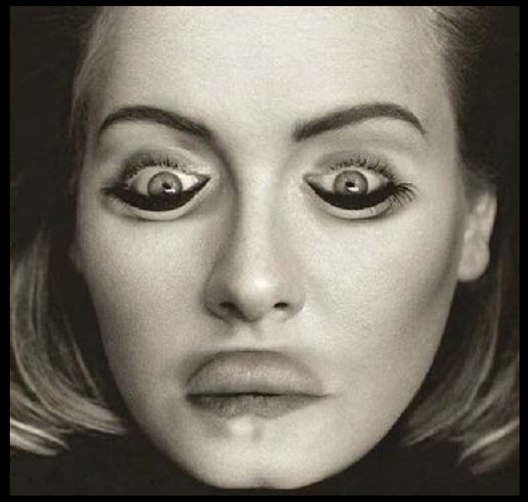
Machine Learning?

A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

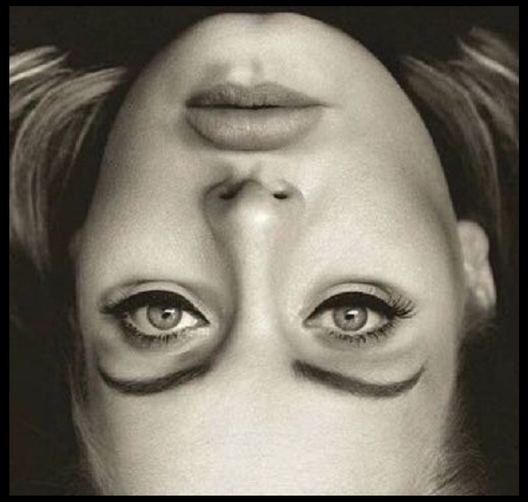
–Tom Mitchell

Learn the “structure” of the observable “world”

$p \gg 0$



$p \approx 0$



$p \approx 0$

What is the probability of observing a particular configuration of the world?

Let $\mathbb{X}$ = set of all possible configurations of the "world"

The "structure" of the world is described by a probability distribution $p_{\mathbb{X}}$

$p_{\mathbb{X}}(X) \propto$ **probability of observing configuration** X

$p_{\mathbb{X}}(Y|X) \propto$ **probability of observing features** Y **given configuration** X

Unsupervised Learning

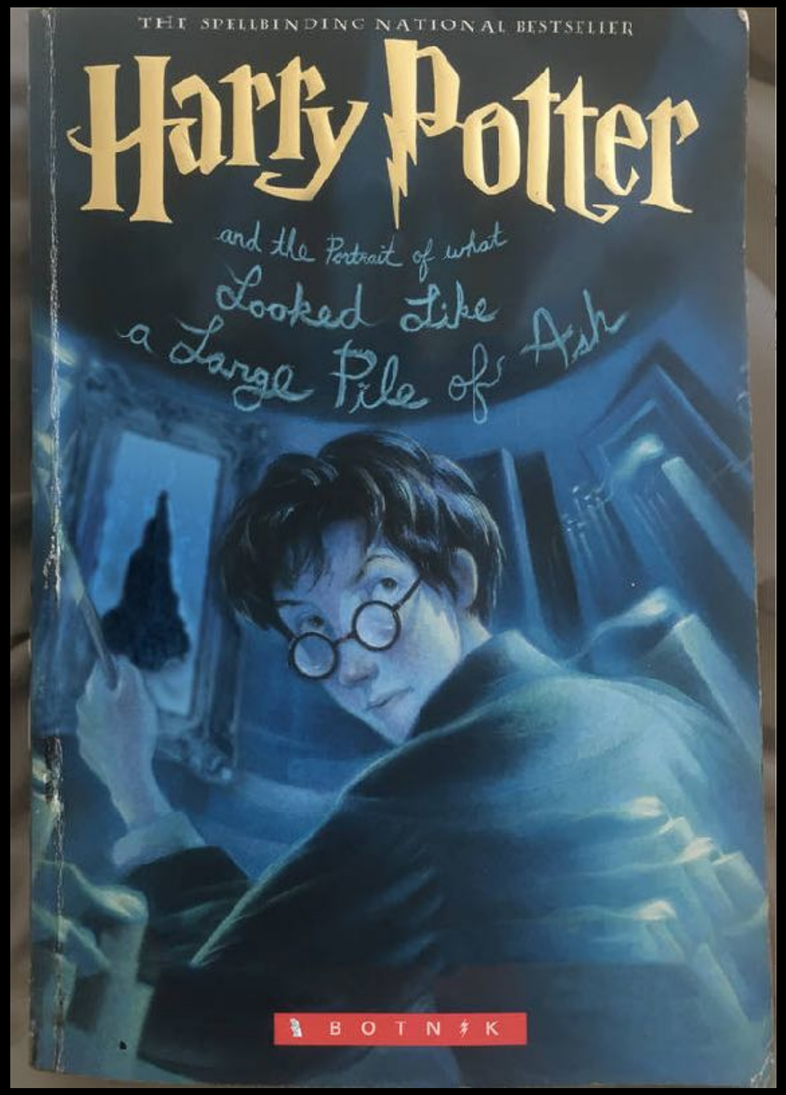
$$p_{model}(X;\theta) \approx p_{\mathbb{X}}(X)$$

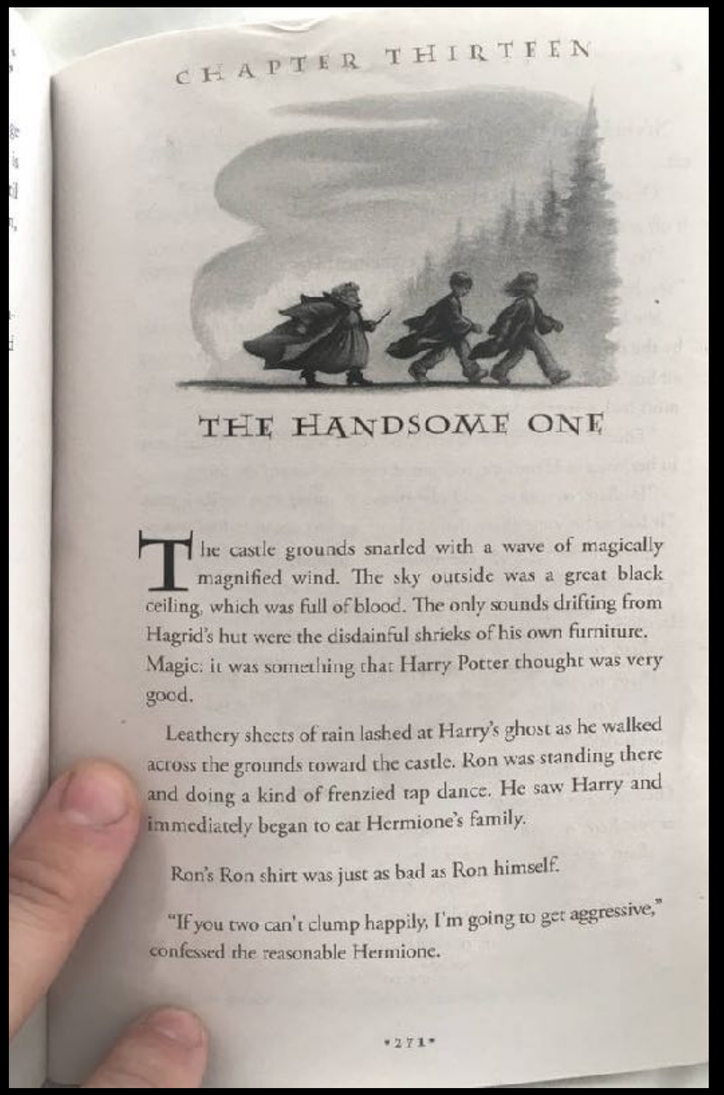
Supervised Learning

$$p_{model}(Y|X;\theta) \approx p_{\mathbb{X}}(Y|X)$$

$$x \sim p_{\text{CelebA}}(X)$$

http://research.nvidia.com/sites/default/files/pubs/2017-10_Progressive-Growing-of/karras2018iclr-paper.pdf





CHAPTER THIRTEEN

THE HANDSOME ONE

The castle grounds snarled with a wave of magically magnified wind. The sky outside was a great black ceiling, which was full of blood. The only sounds drifting from Hagrid's hut were the disdainful shrieks of his own furniture. Magic: it was something that Harry Potter thought was very good.

Leathery sheets of rain lashed at Harry's ghost as he walked across the grounds toward the castle. Ron was standing there and doing a kind of frenzied tap dance. He saw Harry and immediately began to eat Hermione's family.

Ron's Ron shirt was just as bad as Ron himself.

"If you two can't clump happily, I'm going to get aggressive," confessed the reasonable Hermione.

•271•

$$x \sim p_{\text{Harry Potter}}(X)$$

http://botnik.org/content/harry-potter.html

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

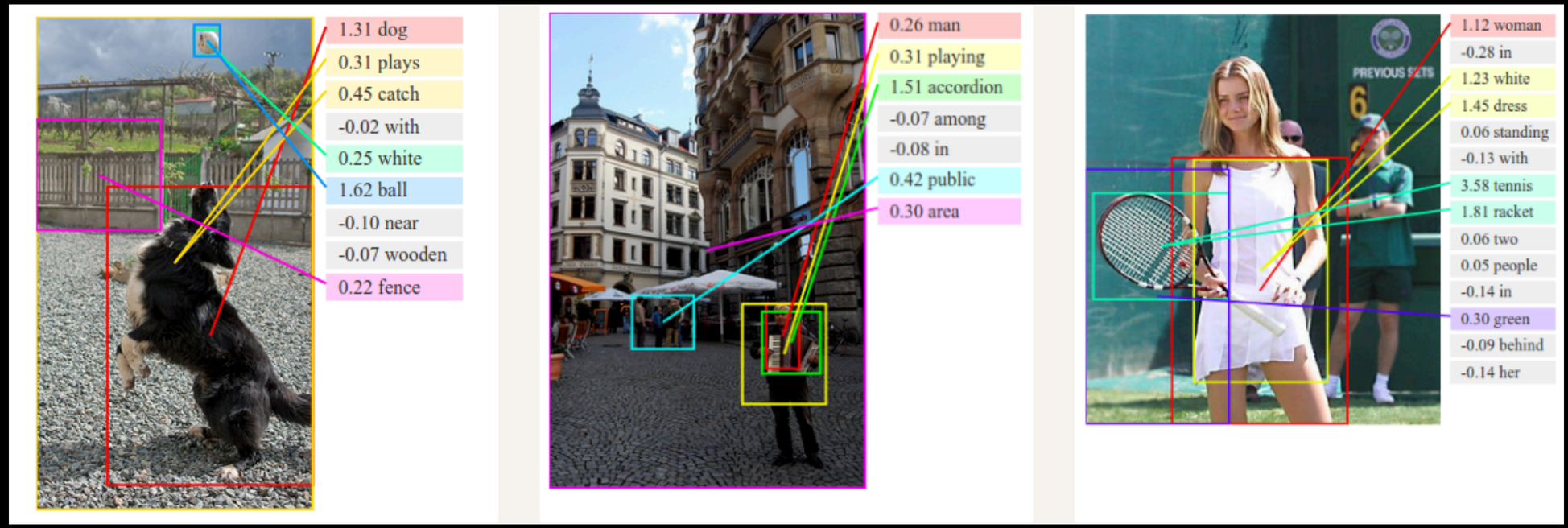
"man in blue wetsuit is surfing on wave."

$$y \sim p_{\text{coco}}(Y \mid X)$$

https://cs.stanford.edu/people/karpathy/deepimagesent/

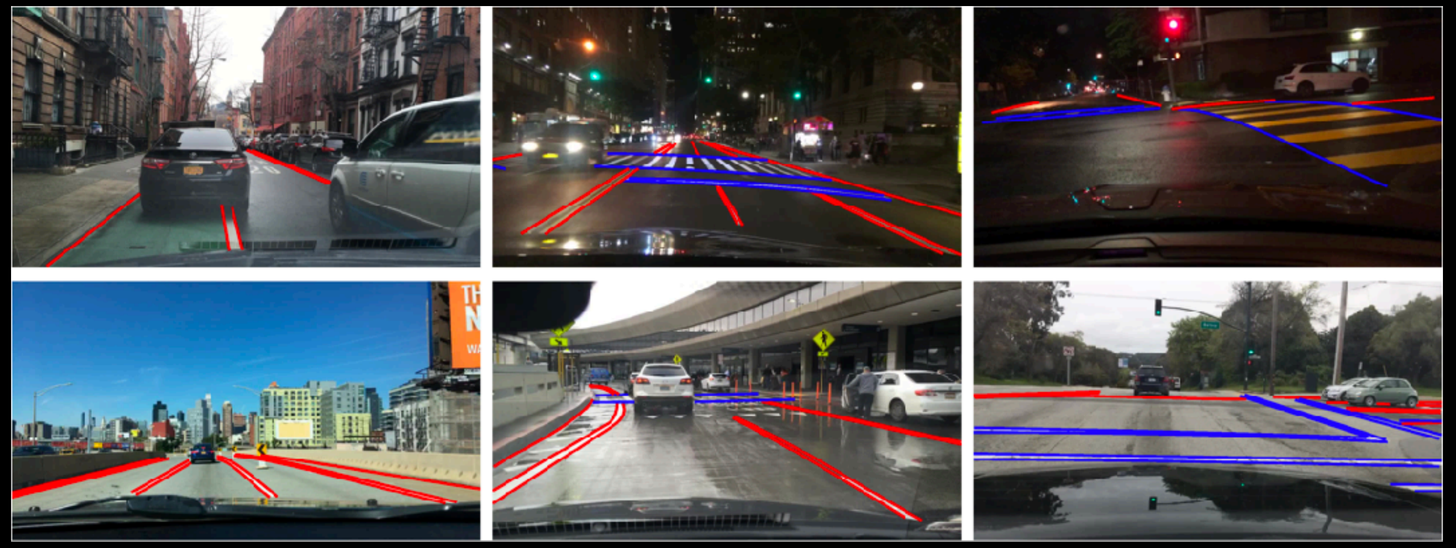
$$y \sim p_{\textbf{model}}(Y \mid X; \theta) \approx p_{\textbf{COCO}}(Y \mid X)$$

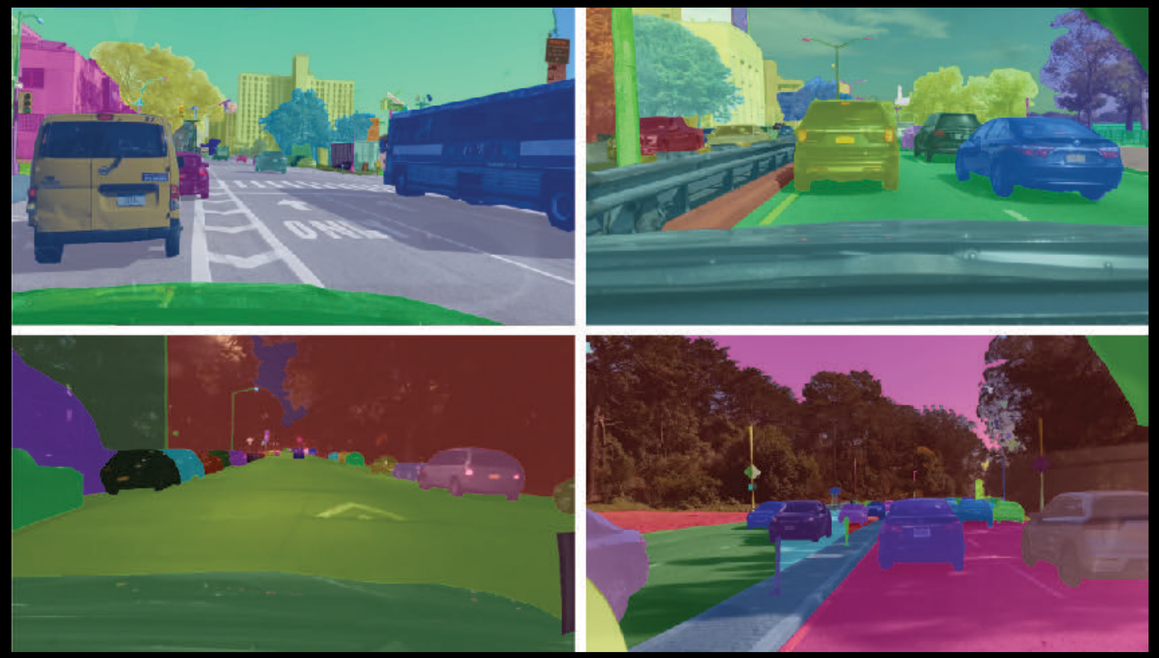
https://cs.stanford.edu/people/karpathy/deepimagesent/



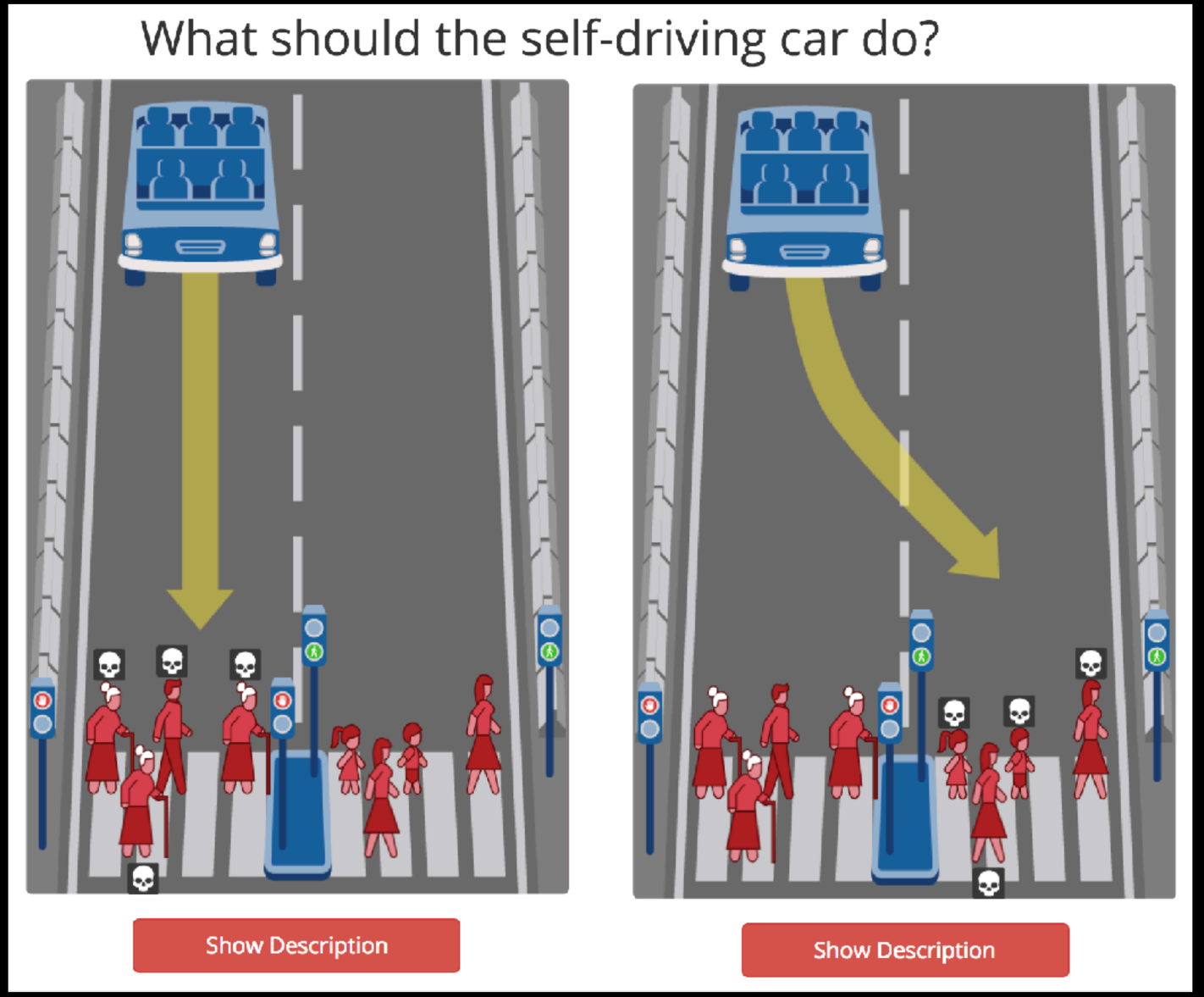
$$y \sim p_{\text{COCO+}}(Y \mid X)$$

https://cs.stanford.edu/people/karpathy/deepimagesent/





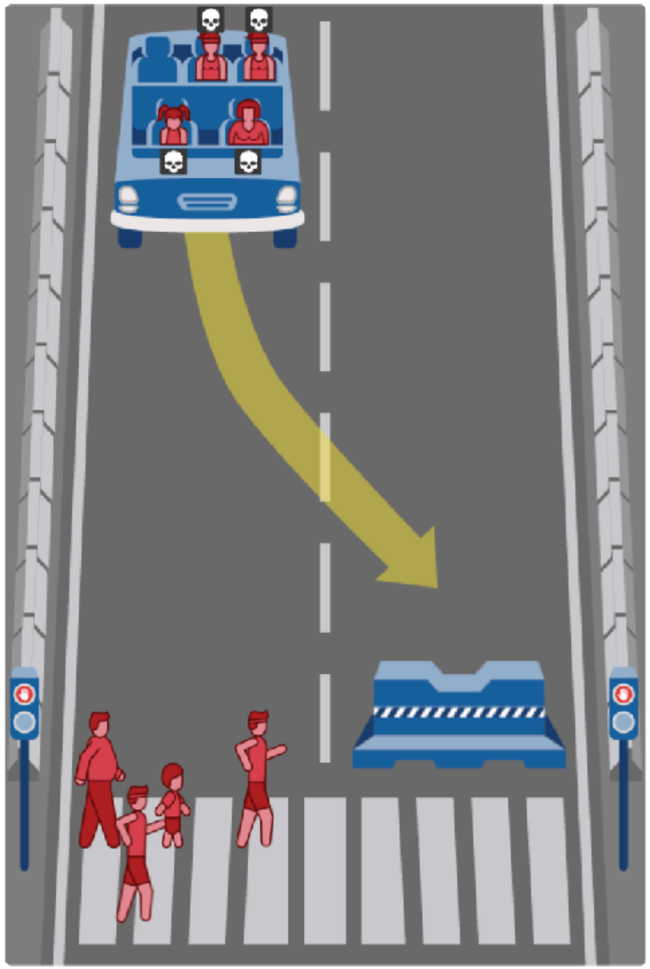
$$y_{t+1} \sim p_{\textbf{BDD100K}}(Y_t \mid X)$$



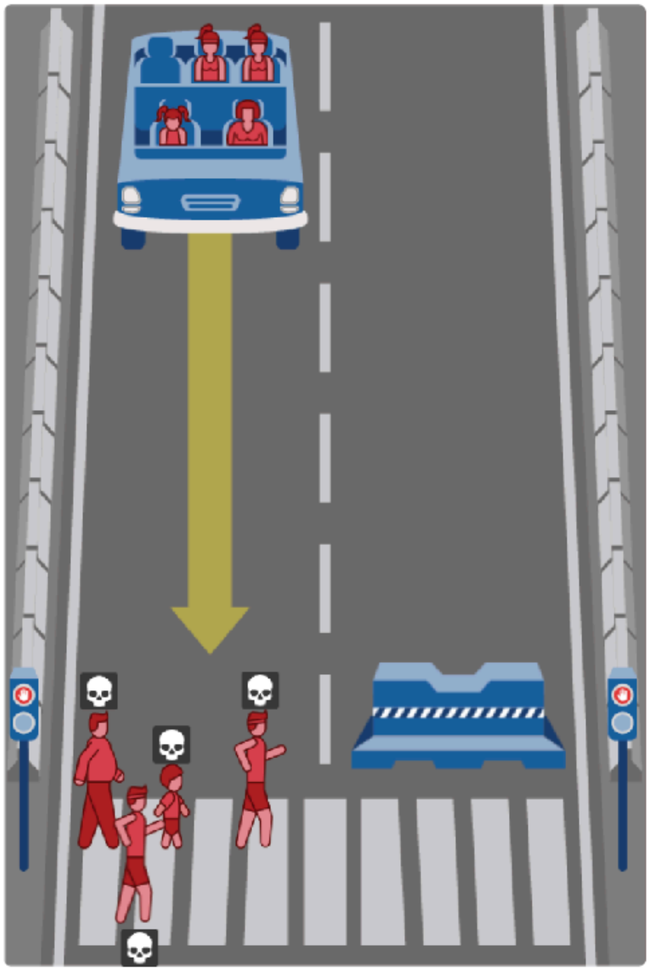
???

http://moralmachine.mit.edu/

What should the self-driving car do?



Show Description



Show Description

???!@#$%^&*???

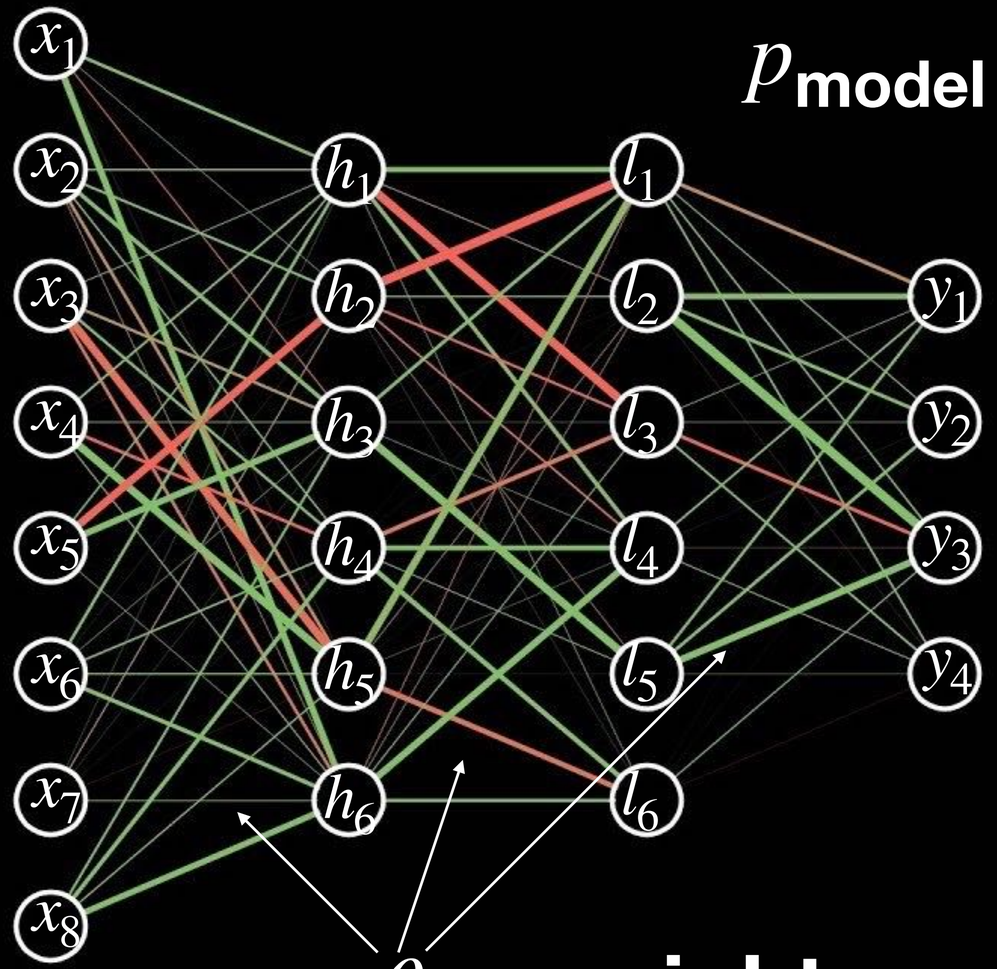
http://moralmachine.mit.edu/

How is any of this possible?

p_{model} **is "good" if** $\hat{y} \approx y$

Given sample (x, y)

x →

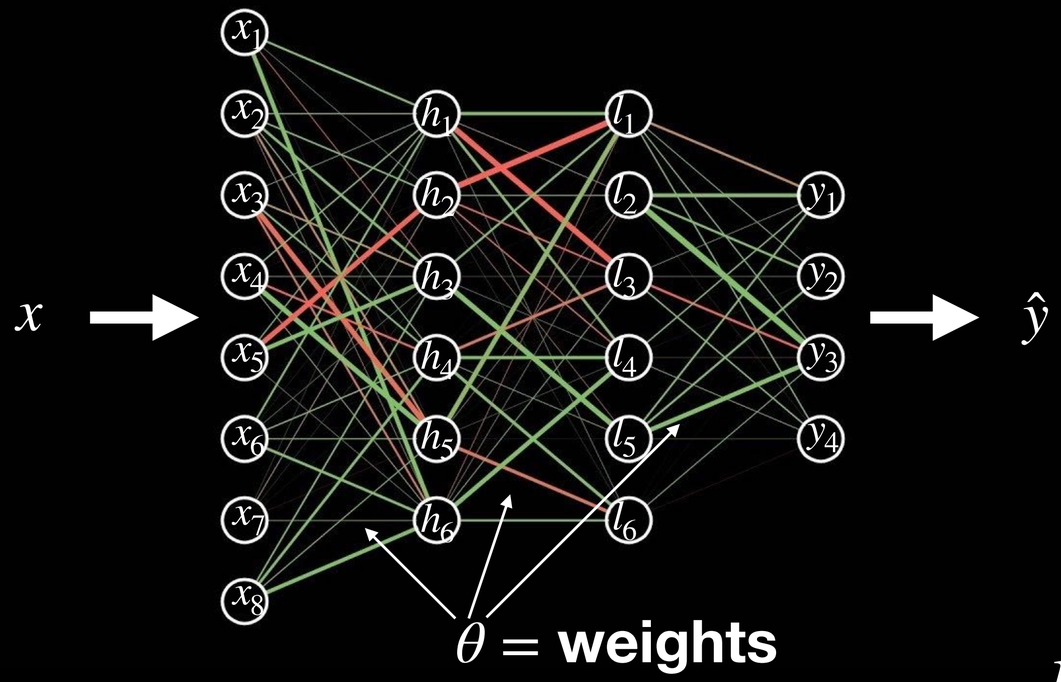


→ $\hat{y}$

True output is y

$\theta =$ **weights**

$$p_{\text{model}}(Y \mid X; \theta) \approx p_{\text{data}}(Y \mid X)$$



Classification

$$\hat{y}_1 = p(Y_1 \mid x; \theta) = 0.1$$
$$\hat{y}_2 = p(Y_2 \mid x; \theta) = 0.2$$
$$\hat{y}_3 = p(Y_3 \mid x; \theta) = 0.7$$
$$\hat{y}_4 = p(Y_4 \mid x; \theta) = 0.0$$

Regression

$$p_{\textbf{model}}(y \mid x; \theta) = \mathcal{N}(y \mid \mu = \hat{y}(x; \theta), \Sigma)$$

Maximum likelihood estimation of weights

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, p_{\textbf{model}}(\mathbb{X} \mid \theta)$$

Bayes' Rule

$$p(\theta \mid \mathbb{X}) \propto p(\mathbb{X} \mid \theta) p(\theta)$$

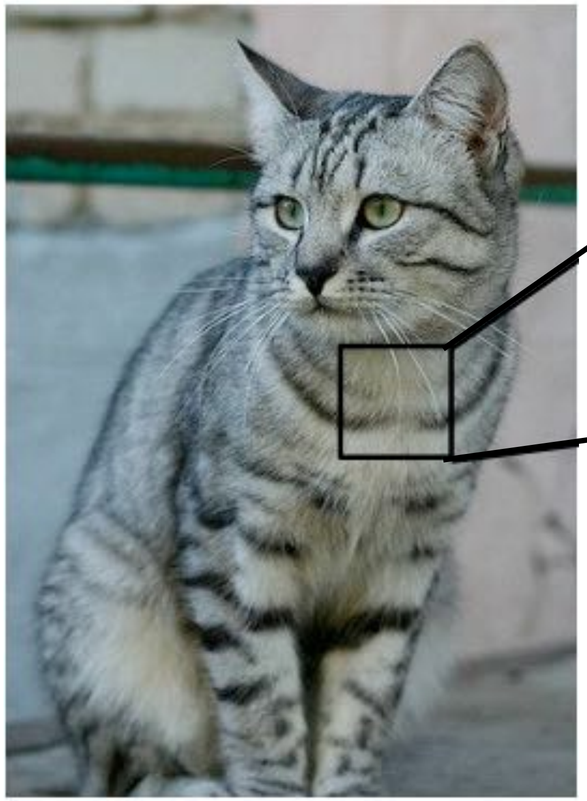
Believe it or not, this is how we train most networks!

Image Classification : A core task in Computer Vision

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

cat

The Problem : Semantic Gap



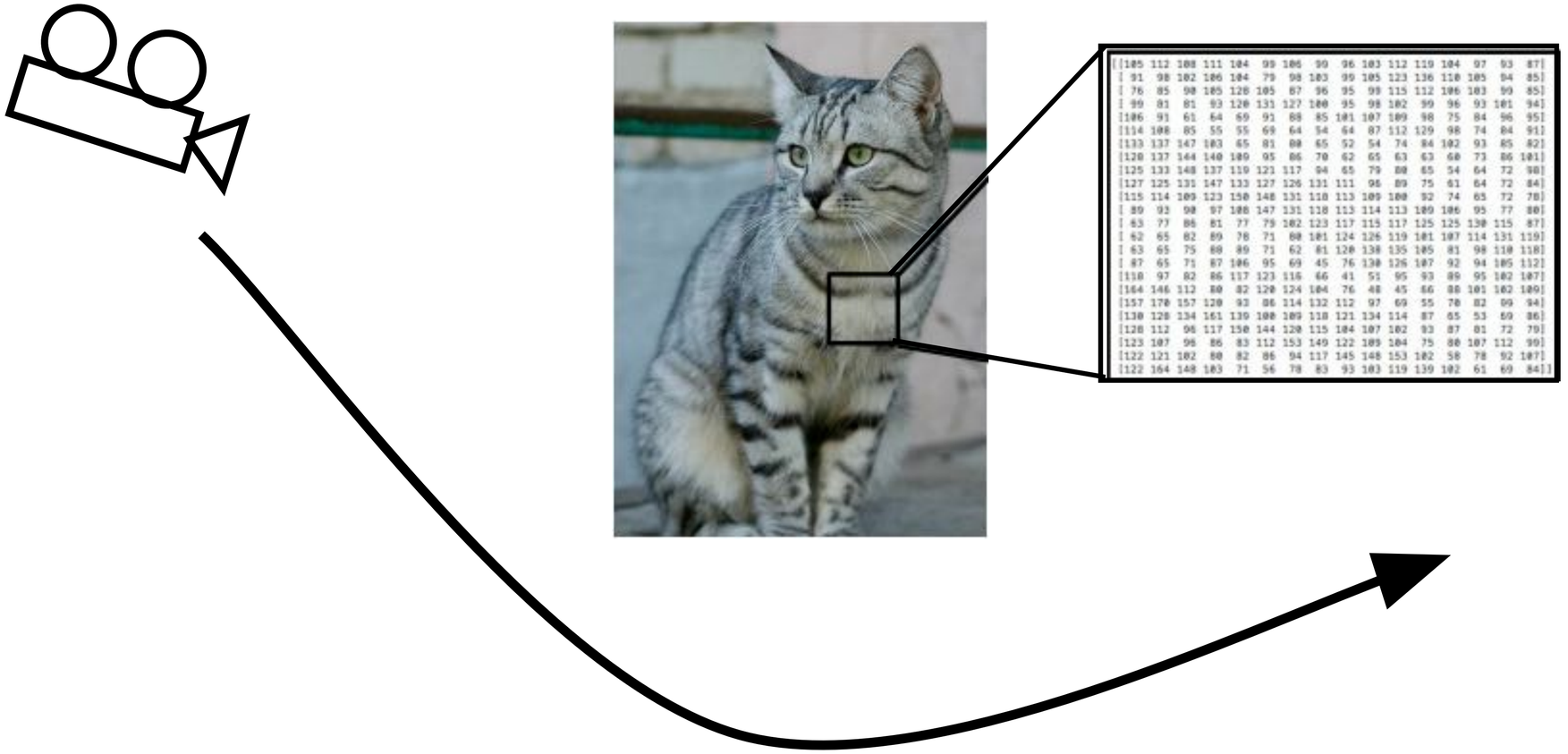

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  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 big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges : Viewpoint variation



All pixels change when the camera moves!

Challenges : Illumination

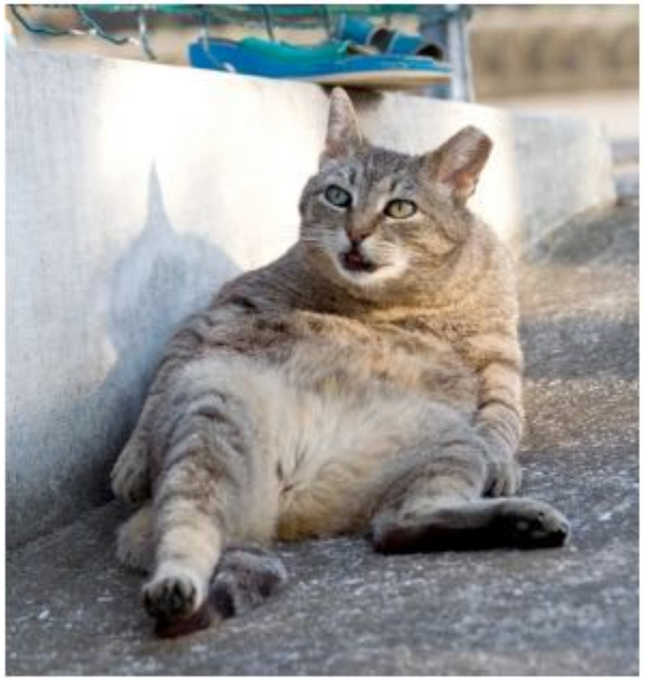
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

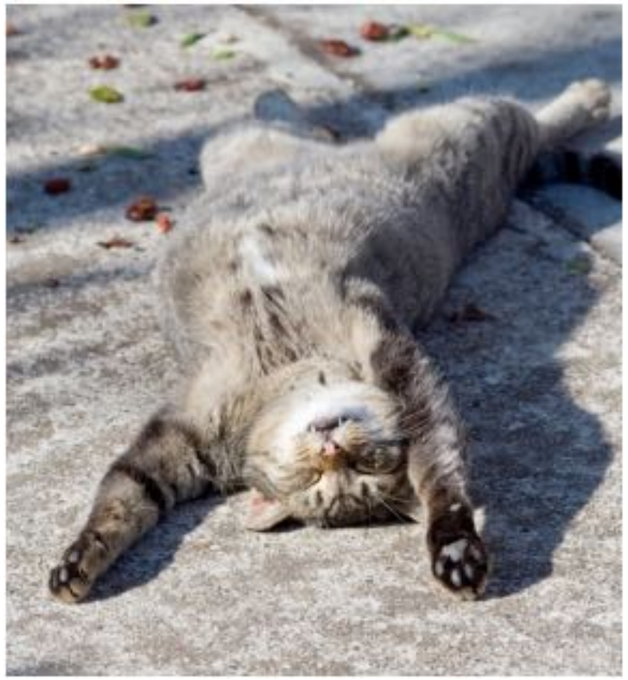
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges : Deformation

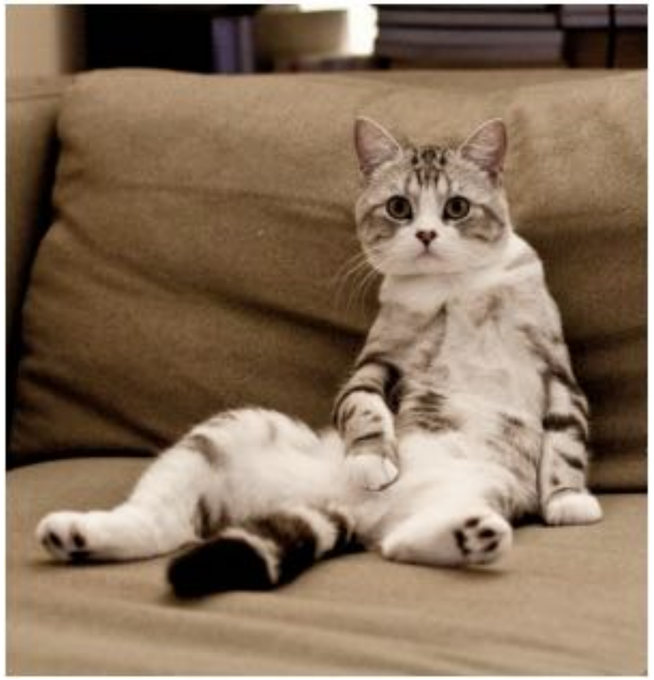


This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by Umberto Salvagnin
is licensed under CC-BY 2.0

This image by sare bear is
licensed under CC-BY 2.0



This image by Tom Thai is
licensed under CC-BY 2.0

Used with permission from Justin Johnson

Challenges : Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

Challenges : Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Used with permission from Justin Johnson

Challenges : Intraclass variation

An image classifier

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

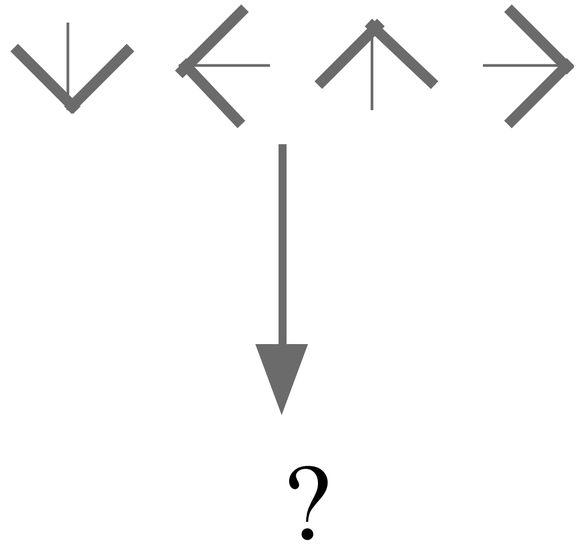
Unlike e.g. sorting a list of numbers,

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

Attempts have been made

Find edges

Find corners



?

John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986

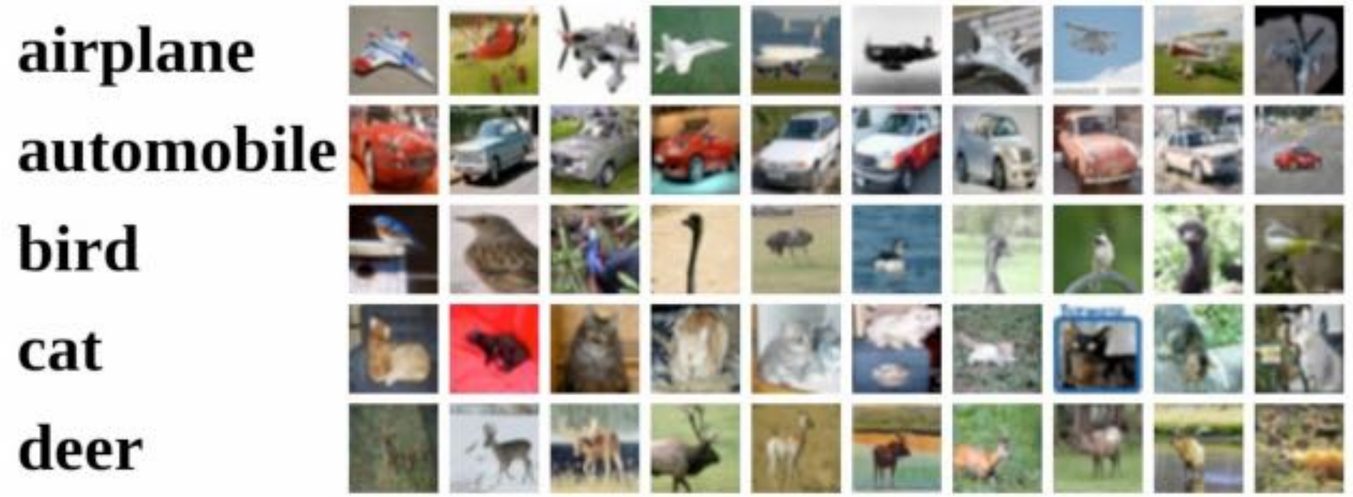
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

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

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

Example training set



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

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

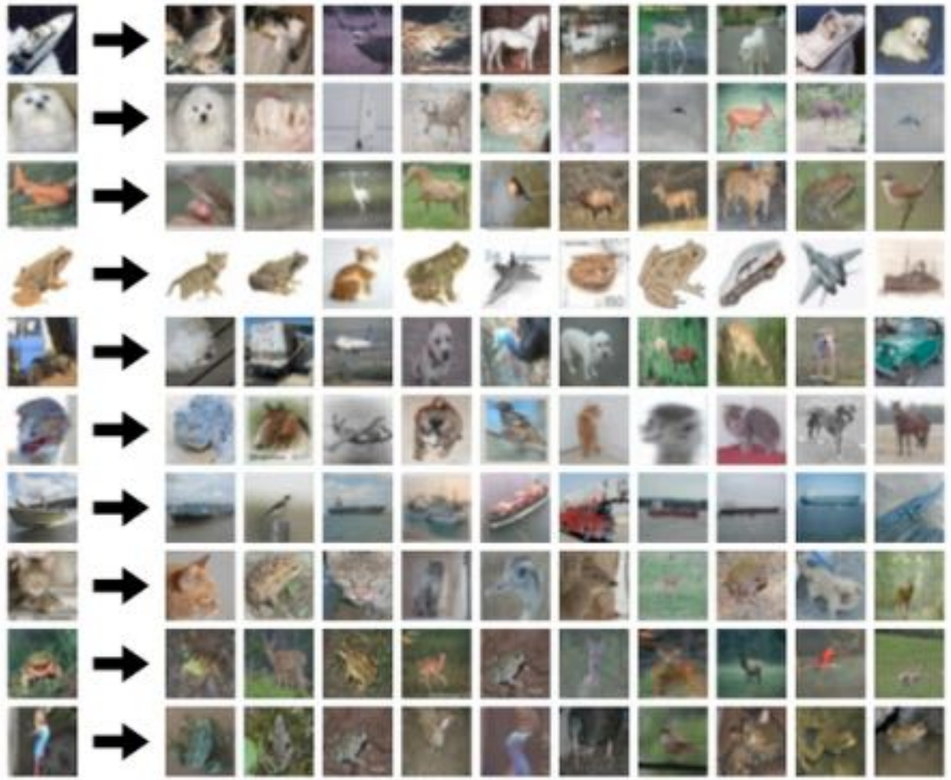
Example Dataset: **CIFAR10**

10 classes

50,000 training images

10,000 testing images

Test images and nearest neighbors

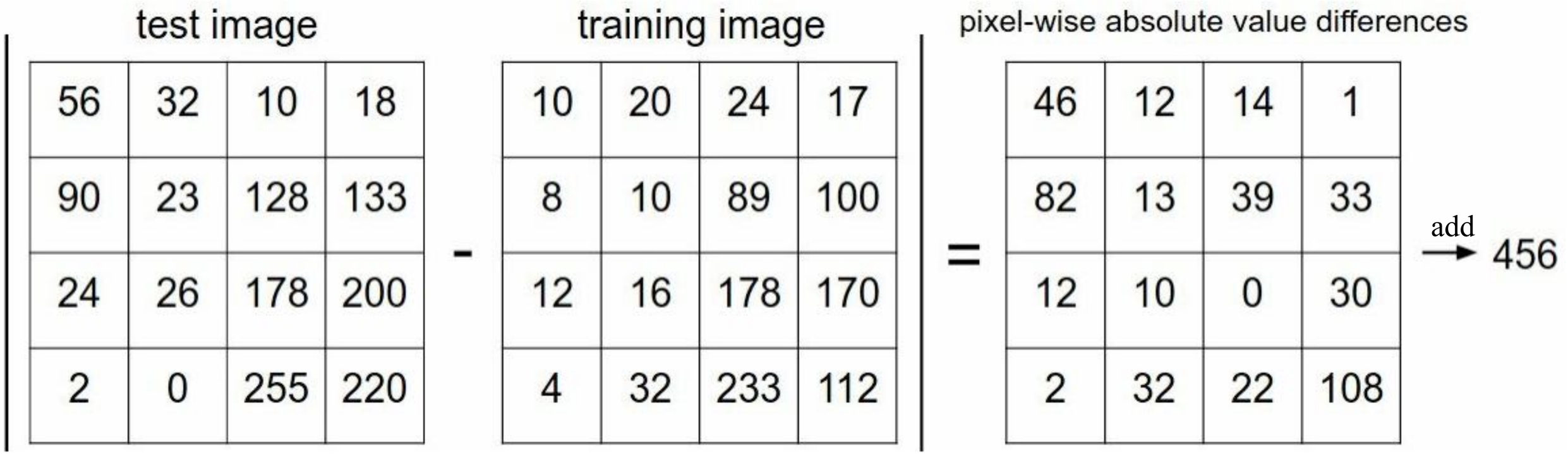


Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Distance Metric to compare images

L1 distance:

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



Nearest Neighbor classifier

```python
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

```python
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
```

Memorize training data

Nearest Neighbor classifier

```python
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

```python
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?

```python
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)

```python
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
```

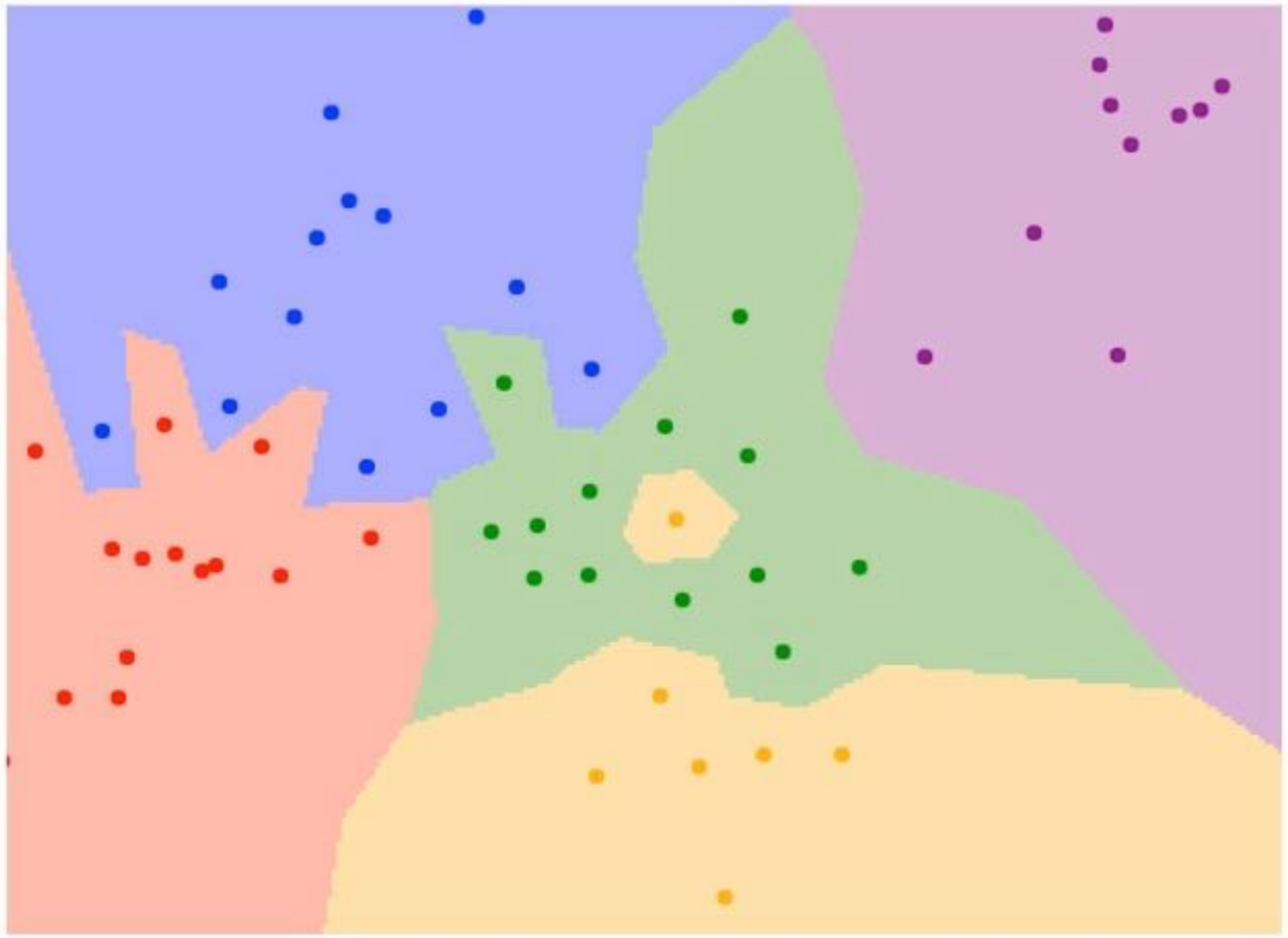
Nearest Neighbor classifier

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

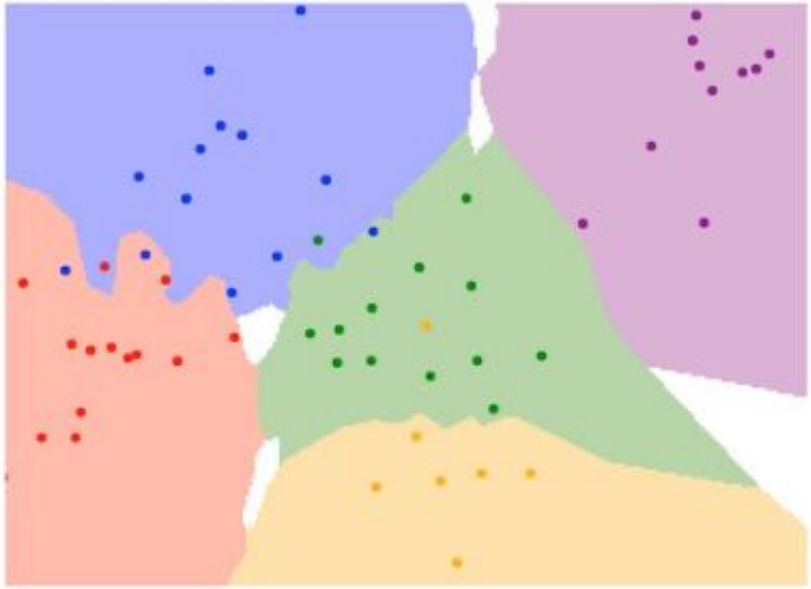
What does this look like?



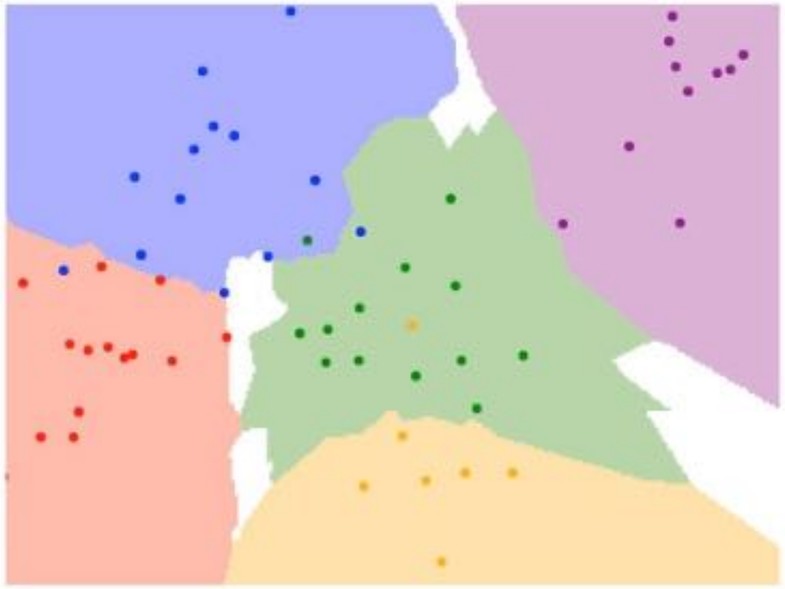
K-Nearest Neighbors

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

K = 1

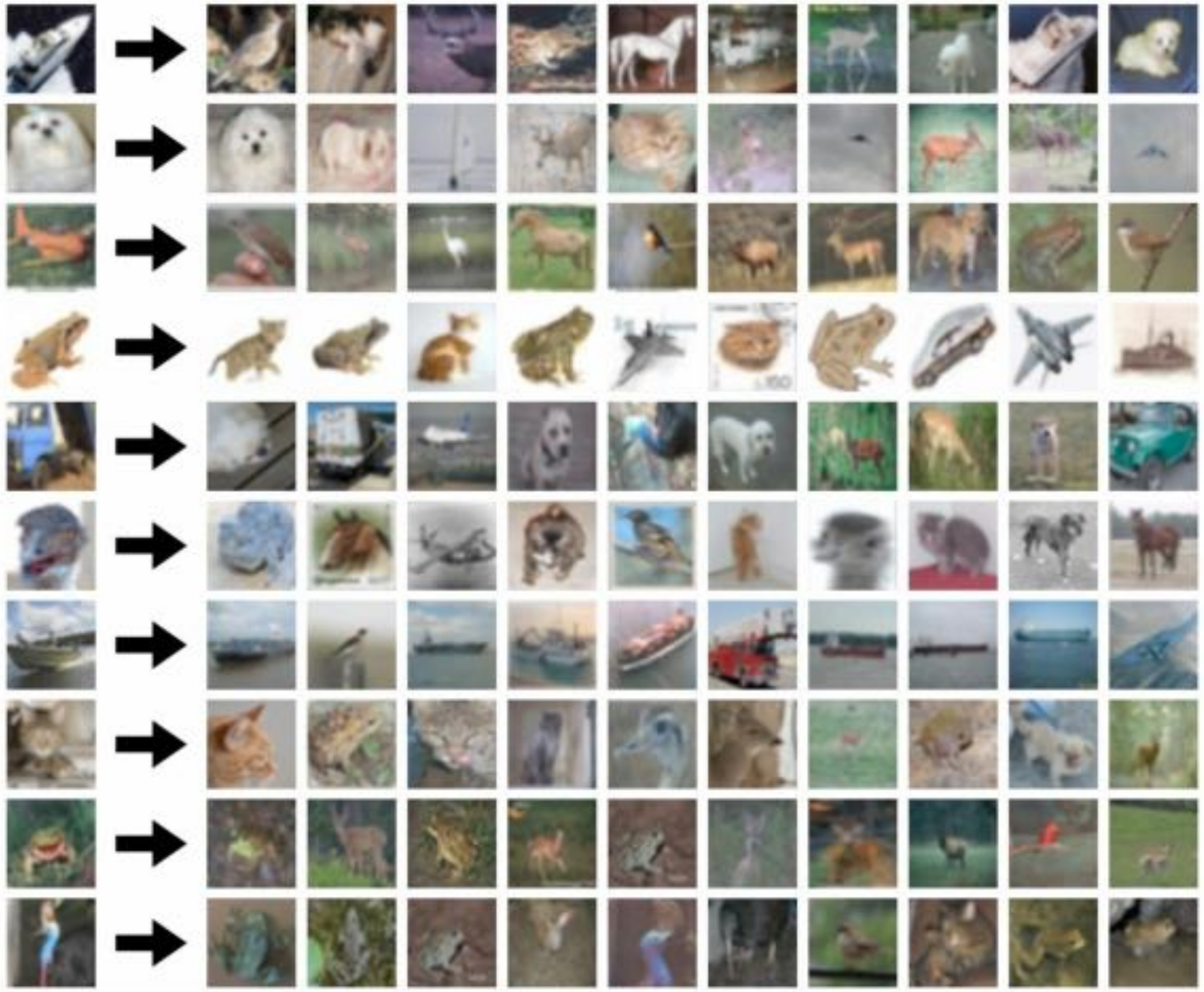


K = 3

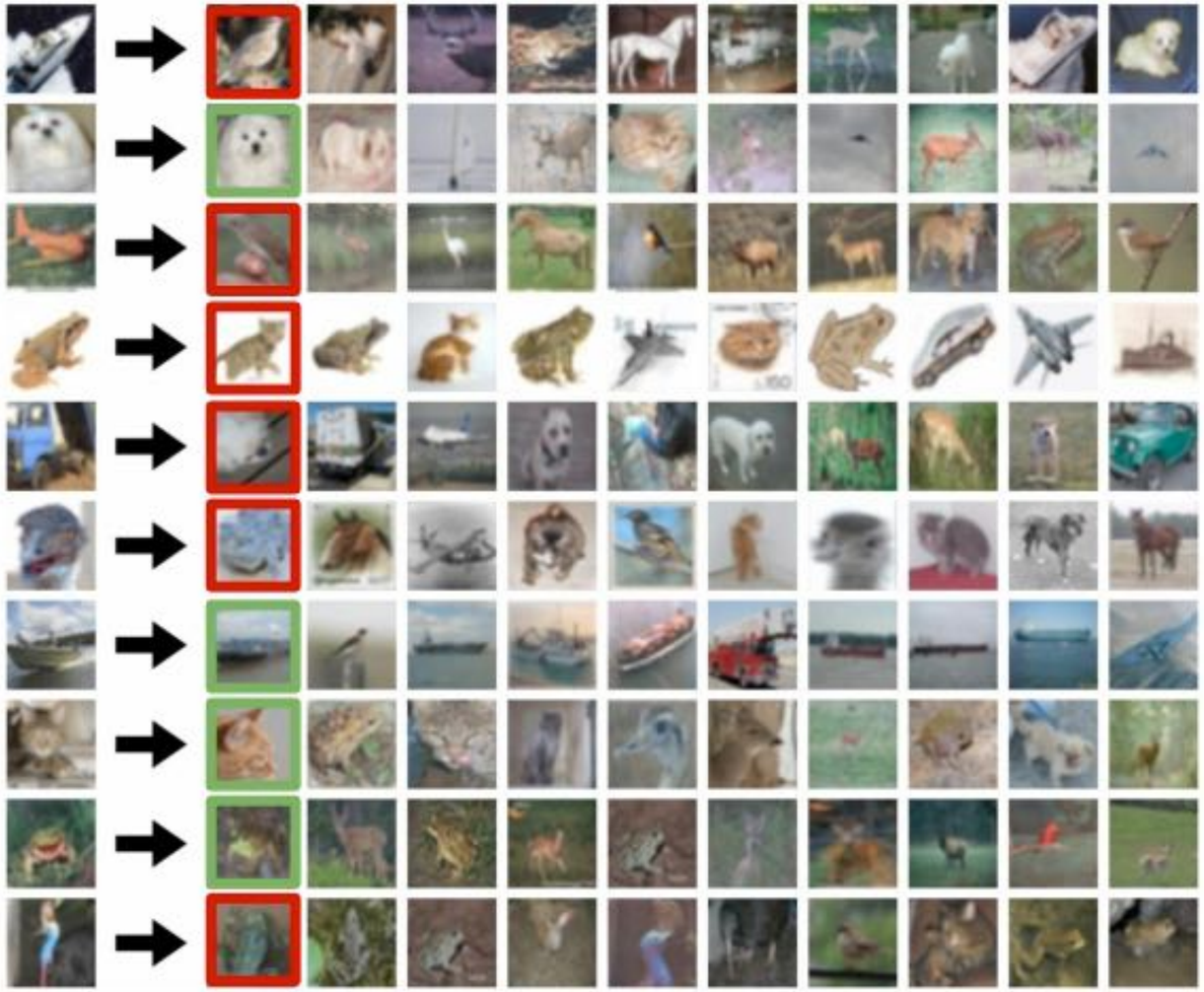


K = 5

What does this look like?



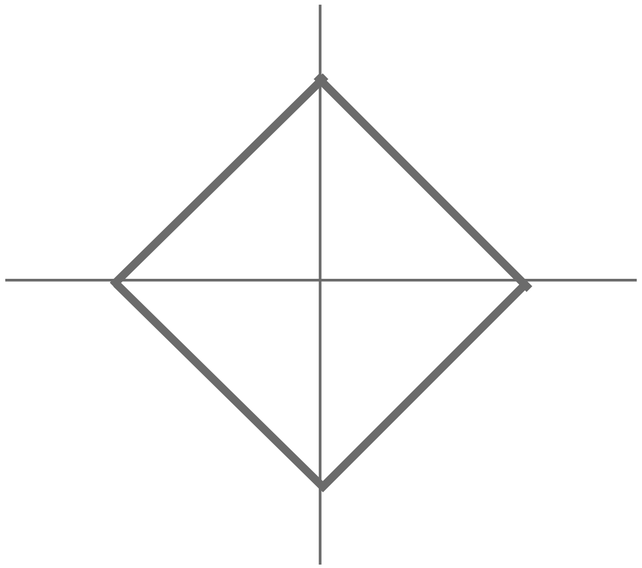
What does this look like?



K-Nearest Neighbors: Distance Metric

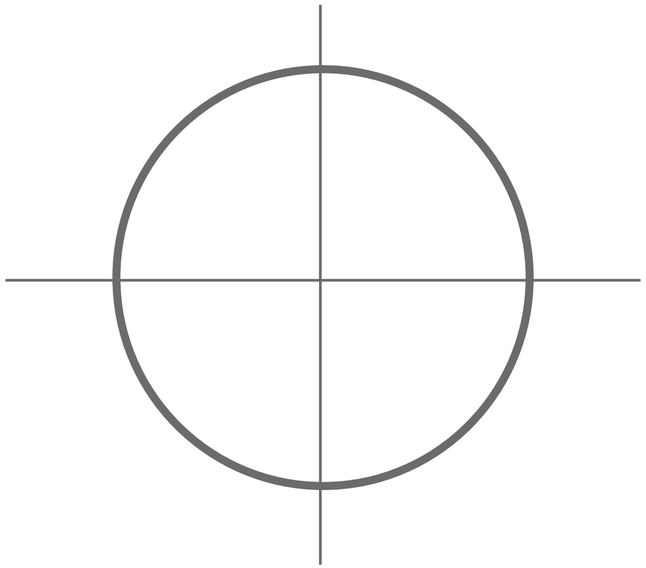
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

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

K = 1

L2 (Euclidean) distance

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

K = 1

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

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

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

Setting Hyperparameters

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

BAD : K = 1 always works perfectly on training data

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

Setting Hyperparameters

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

BAD : K = 1 always works perfectly on training data

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

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

Better!

Setting Hyperparameters

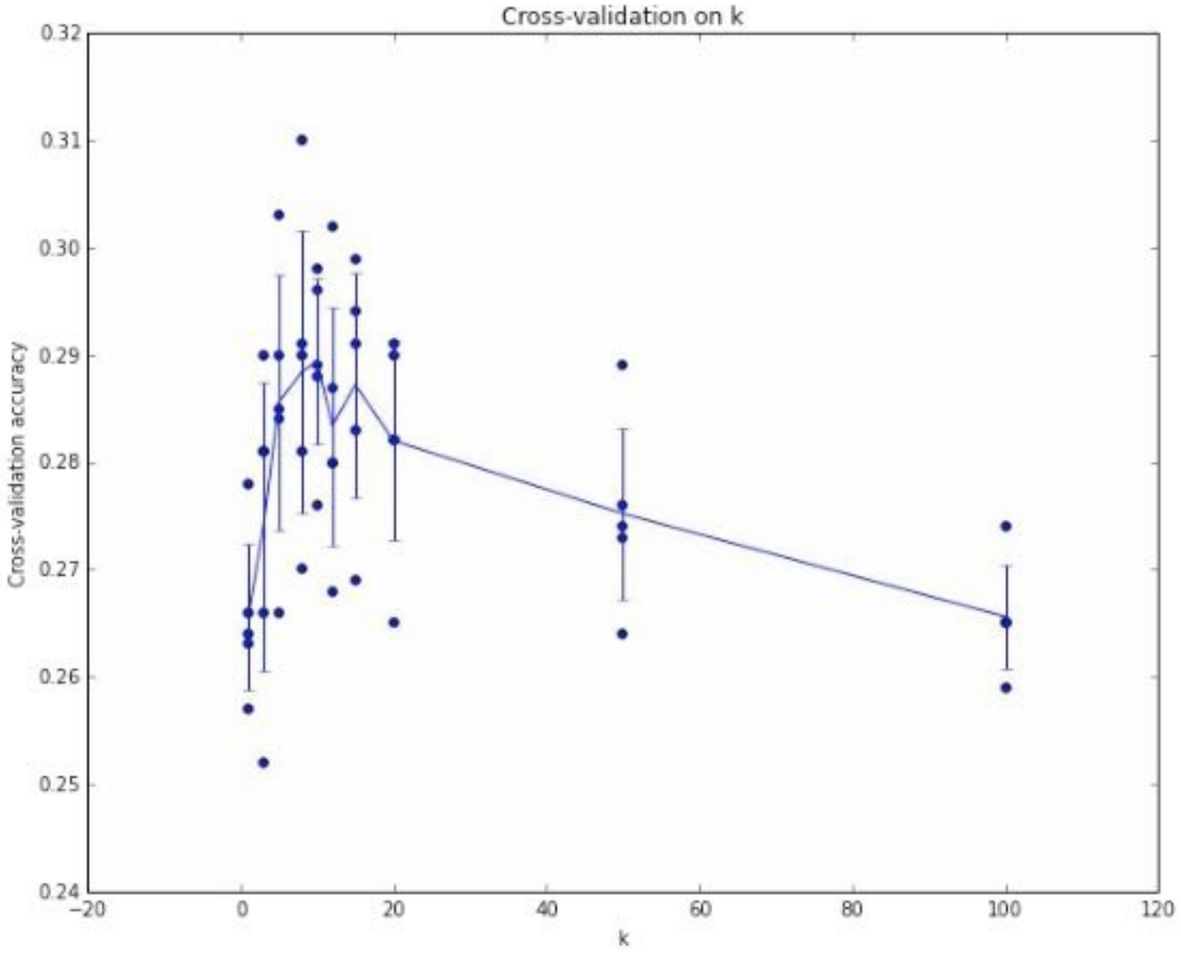
Your Dataset

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

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

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.**

Each point: single
outcome.

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

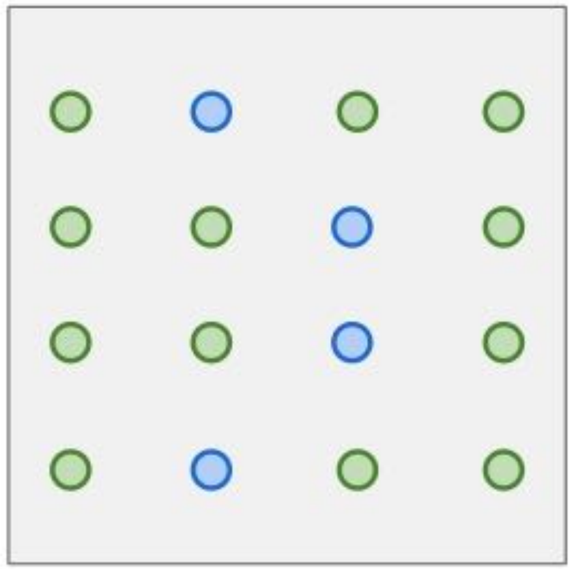
(all 3 images have same L2 distance to the one on the left)

k-Nearest Neighbor on images **never used.**

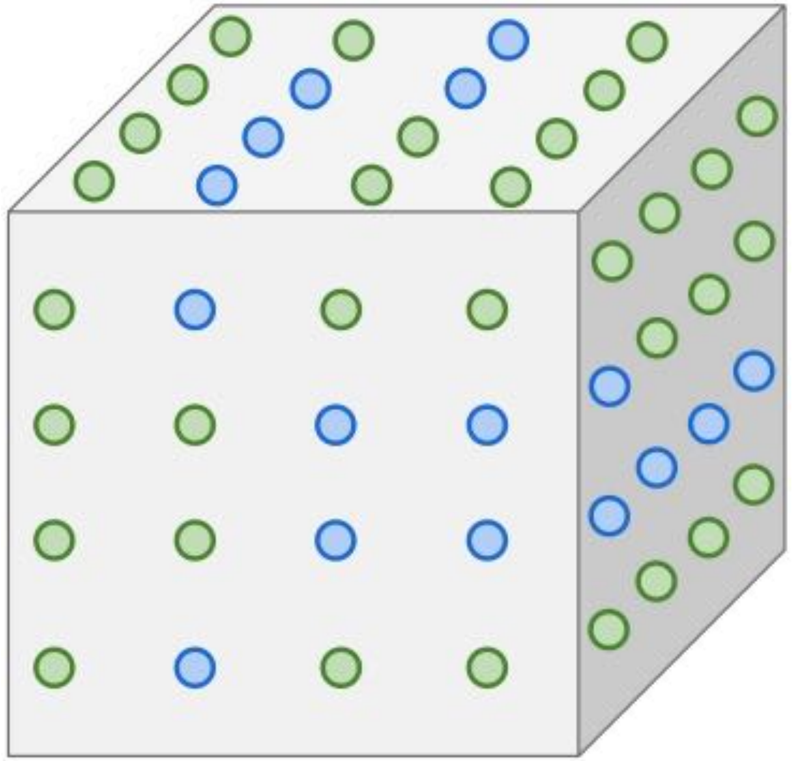
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



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!