

Lecture 2: Image Classification

A Core Task in Computer Vision

Administrative: Assignment 1

Due 4/16 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax
- Two-layer neural network
- Image features

Administrative: Course Project

Project proposal due 4/19 (Monday)

Find your teammates on Piazza (the pinned “Search for Teammates” post)

Collaboration: Slack / Zoom

“Is X a valid project for 231n?” --- Piazza private post / TA Office Hours

More info on the website

Administrative: Sections

This Friday 11:30-12:30 pm (recording will be made available)

Python / Numpy, Google Cloud Platform, Google Colab

Presenter: Rachel Gardner (TA)

Syllabus

Neural Network Fundamentals

Data-driven approaches **Linear classification & kNN**

Loss functions
Optimization
Backpropagation
Multi-layer perceptrons
Neural Networks

Convolutional Neural Networks

Convolutions
Pytorch 1.4 / Tensorflow 2.0
Activation functions
Batch normalization
Transfer learning
Data augmentation
Momentum / RMSProp / Adam
Architecture design

Computer Vision Applications

RNNs / LSTMs / Transformers
Image captioning
Interpreting neural networks
Style transfer
Adversarial examples
Fairness & ethics
Human-centered AI
3D vision
Deep reinforcement learning
Scene graphs
Self-supervised learning

Lecture 2: Image Classification

A Core Task in Computer Vision

Today:

- The image classification task
- Two basic data-driven approaches to image classification
 - K-nearest neighbor and linear classifier

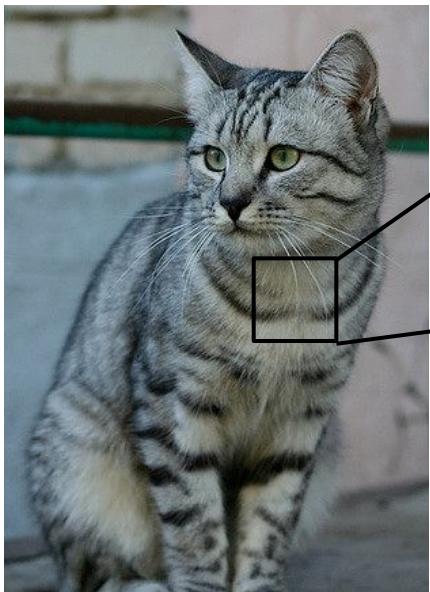
Image Classification: A core task in Computer Vision



cat

This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

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 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 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 81 128 138 135 105 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 88 82 120 124 104 76 48 45 66 88 101 102 109]
[157 170 157 128 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 96 117 150 144 128 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
[122 121 102 88 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]]

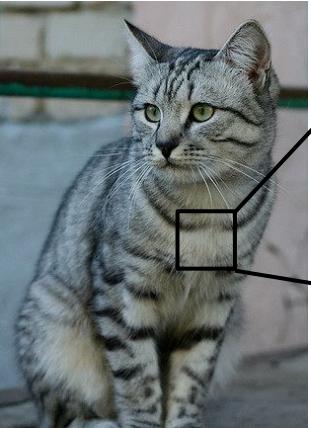
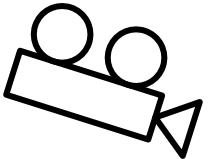
What the computer sees

An image is a tensor of integers between [0, 255]:

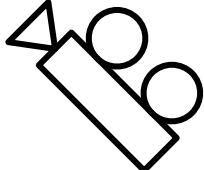
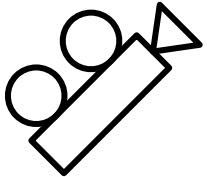
e.g. 800 x 600 x 3
(3 channels RGB)

This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

Challenges: Viewpoint variation



```
[1185 112 188 111 184 99 186 99 96 183 112 119 184 97 93 87]  
[ 91 98 182 106 184 79 98 183 99 185 123 136 118 185 94 85]  
[ 76 85 98 185 128 105 87 96 95 99 115 112 106 183 99 85]  
[ 99 80 81 100 128 105 127 98 98 101 100 109 98 75 84 94]  
[104 91 86 84 60 91 68 85 101 100 109 98 75 84 94 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]  
[133 137 147 183 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 148 105 95 86 78 62 65 63 63 68 73 86 101]  
[102 125 131 147 133 127 116 131 111 98 89 75 61 64 72 84]  
[127 125 131 147 133 127 116 131 111 98 89 75 61 64 72 84]  
[115 115 189 123 150 148 131 118 113 109 108 92 74 65 72 78]  
[ 89 93 98 97 108 147 131 118 113 114 113 108 106 95 77 80]  
[ 63 77 86 81 77 79 182 123 137 115 111 125 125 130 115 87]  
[ 62 85 88 89 73 62 81 128 138 135 105 81 98 118 118]  
[ 63 65 75 88 89 73 62 81 128 138 135 105 81 98 118 118]  
[ 87 65 71 87 100 95 69 45 76 126 126 107 92 94 105 112]  
[118 97 82 86 117 123 116 66 41 51 95 93 89 89 95 102 107]  
[164 147 112 88 100 128 126 184 78 48 66 70 101 102 108]  
[157 98 100 118 100 93 86 104 100 109 100 95 70 89 94 94]  
[138 128 134 161 139 180 109 118 121 134 114 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 83 112 153 149 122 189 184 75 88 107 112 99]  
[122 121 102 88 82 86 94 117 145 148 153 105 58 78 92 107]  
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]
```



All pixels change when
the camera moves!

This image by [Nikita](#) 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

Challenges: Illumination



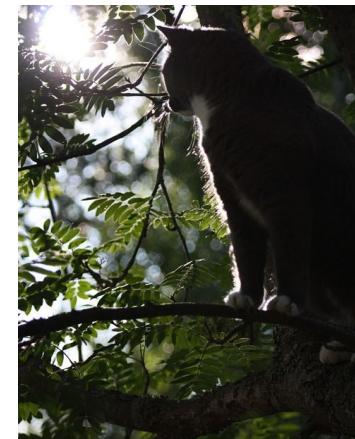
[This image](#) is CC0 1.0 public domain



[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: Occlusion



[This image](#) is [CC0 1.0](#) public domain

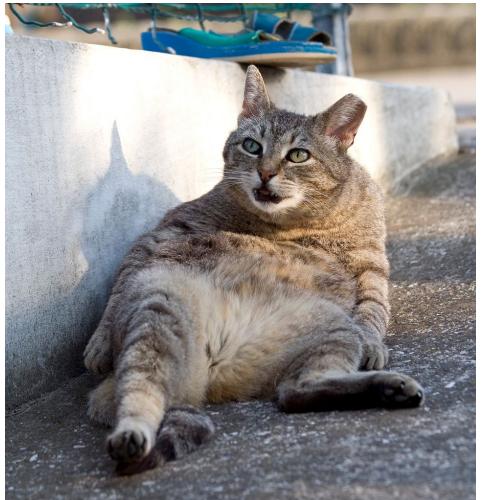


[This image](#) is [CC0 1.0](#) public domain



[This image](#) by [jonsson](#) is licensed
under [CC-BY 2.0](#)

Challenges: 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](#)

Challenges: Intraclass variation



[This image](#) is [CC0 1.0](#) public domain

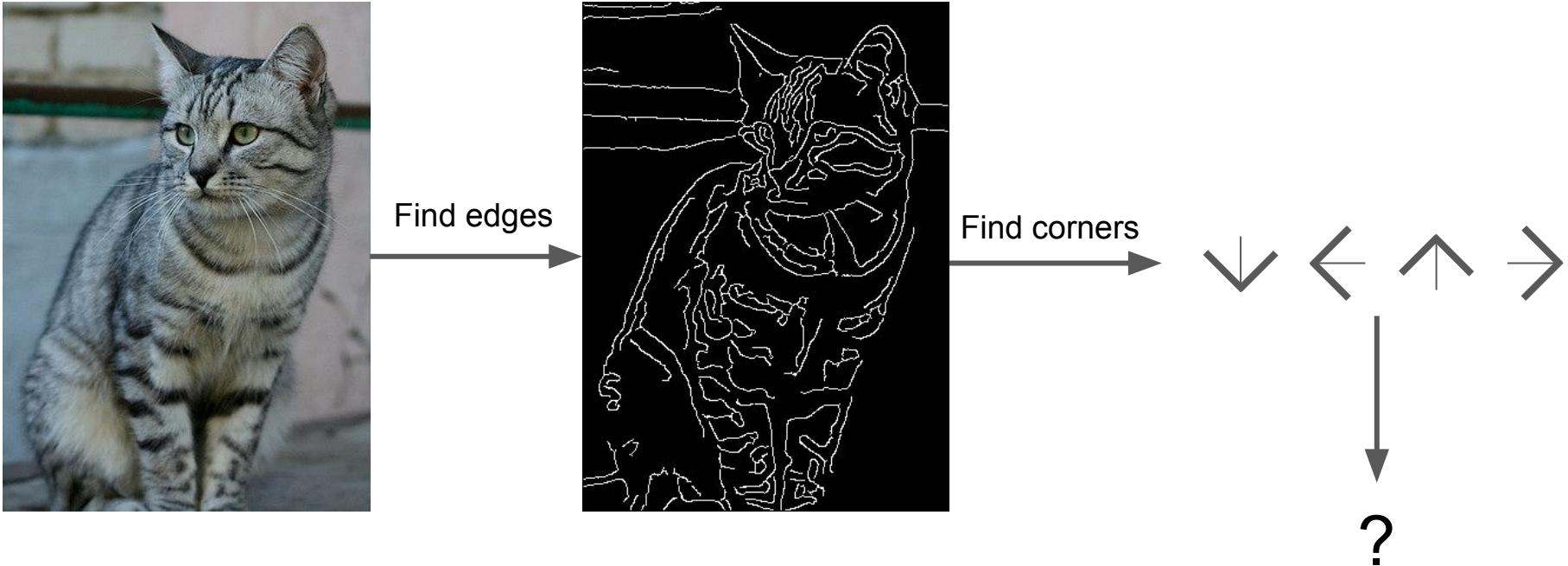
An image classifier

```
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



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

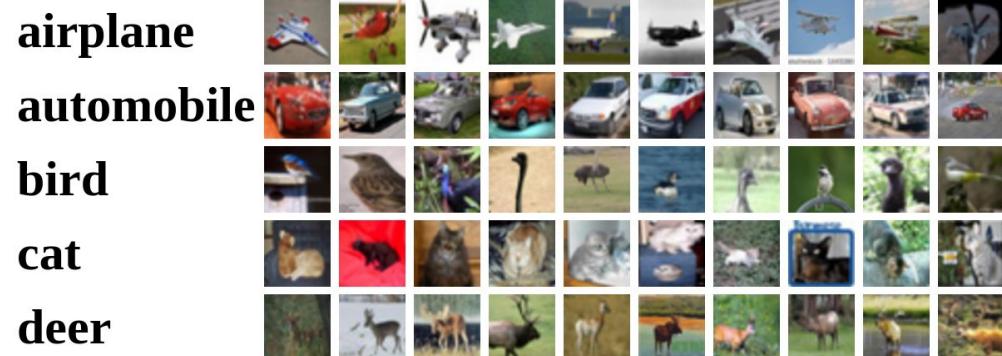
Machine Learning: Data-Driven Approach

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

Example training set

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

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



Nearest Neighbor Classifier

First classifier: Nearest Neighbor

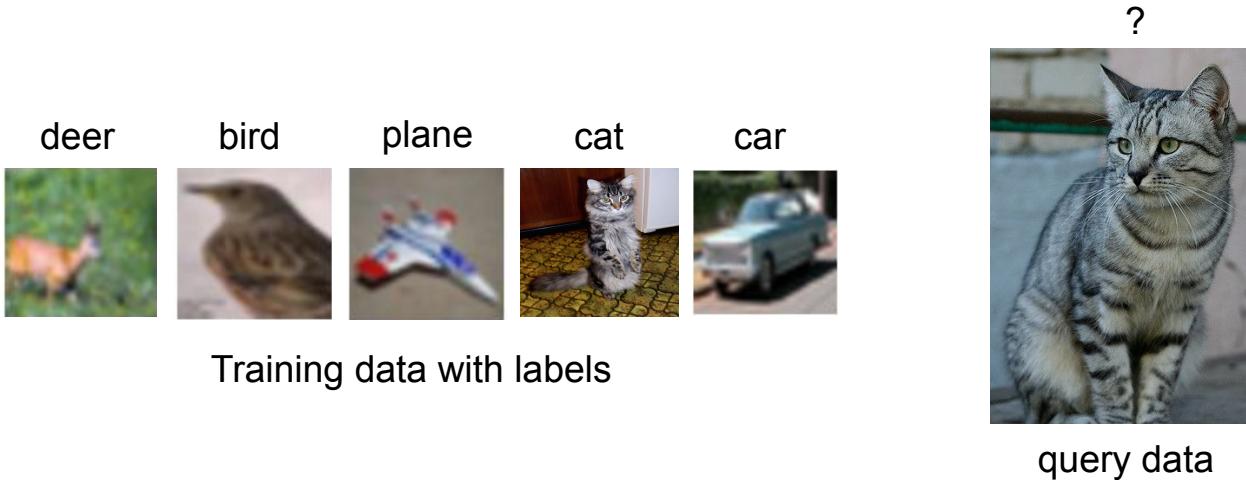
```
def train(images, labels):  
    # Machine learning!  
    return model
```

→ 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

First classifier: Nearest Neighbor



Distance Metric |  | $\rightarrow \mathbb{R}$

Distance Metric to compare images

L1 distance:

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

test image				training image				pixel-wise absolute value differences			
56	32	10	18	10	20	24	17	46	12	14	1
90	23	128	133	8	10	89	100	82	13	39	33
24	26	178	200	12	16	178	170	12	10	0	30
2	0	255	220	4	32	233	112	2	32	22	108

-

=

add → 456

```
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

```

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

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?

Ans: Train O(1), predict O(N)

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

```

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

```

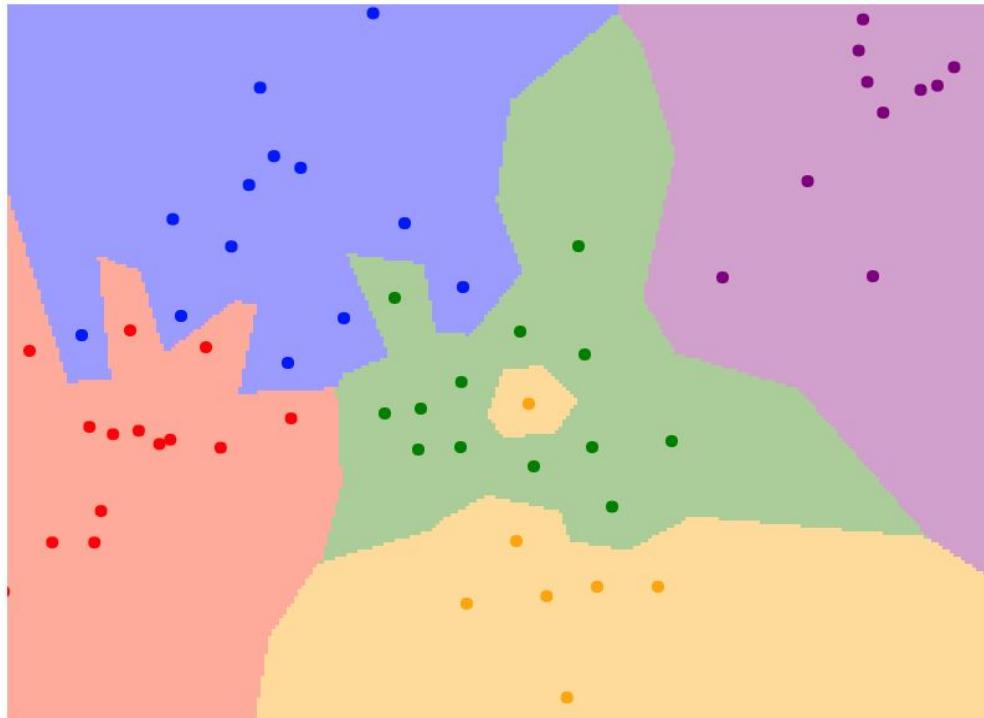
Nearest Neighbor classifier

Many methods exist for fast / approximate nearest neighbor (beyond the scope of 231N!)

A good implementation:
<https://github.com/facebookresearch/faiss>

Johnson et al, “Billion-scale similarity search with GPUs”, arXiv 2017

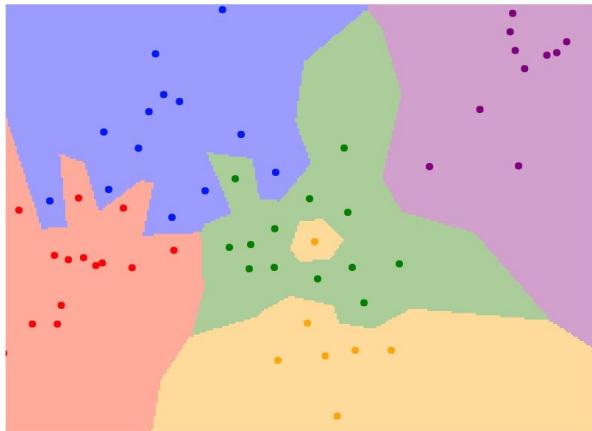
What does this look like?



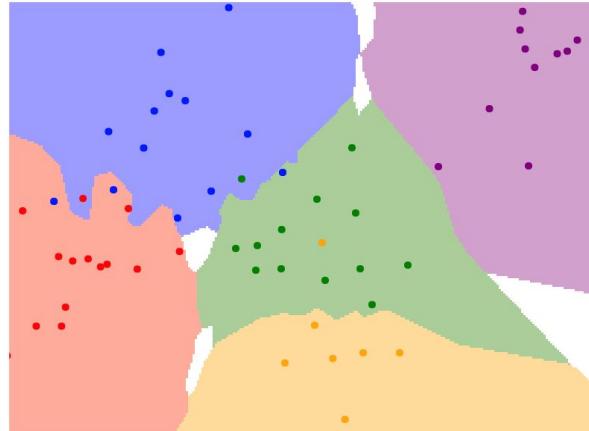
1-nearest neighbor

K-Nearest Neighbors

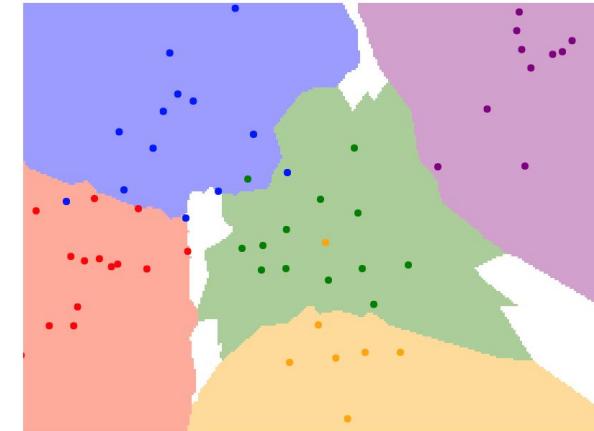
Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



$K = 1$



$K = 3$

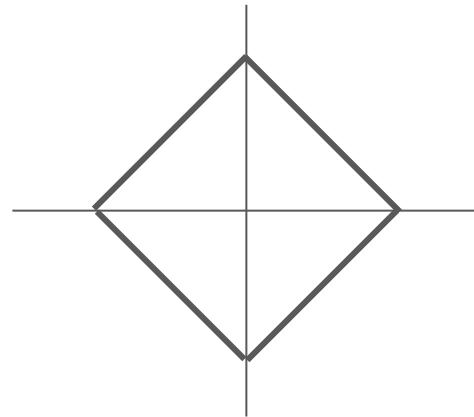


$K = 5$

K-Nearest Neighbors: Distance Metric

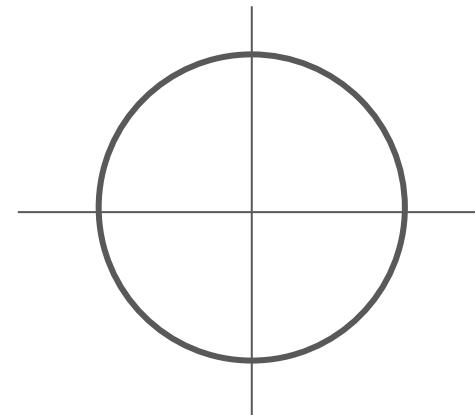
L1 (Manhattan) distance

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



L2 (Euclidean) distance

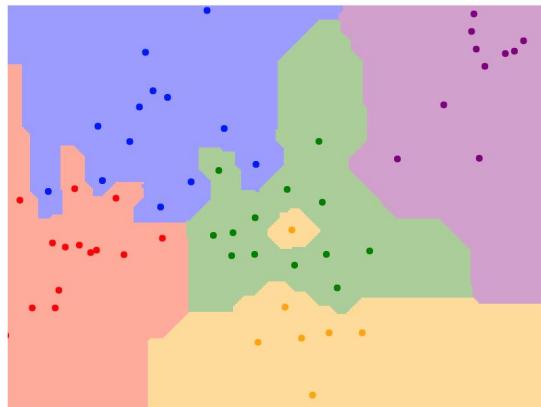
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

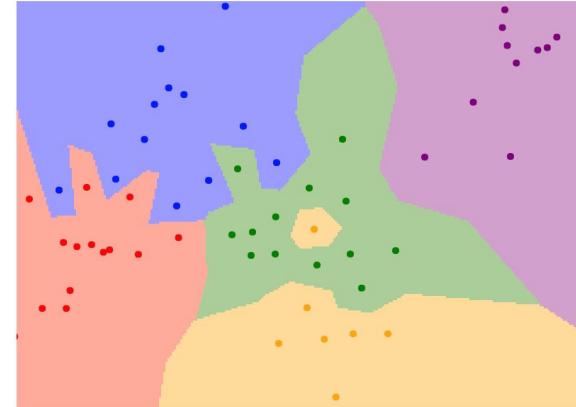
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



$K = 1$

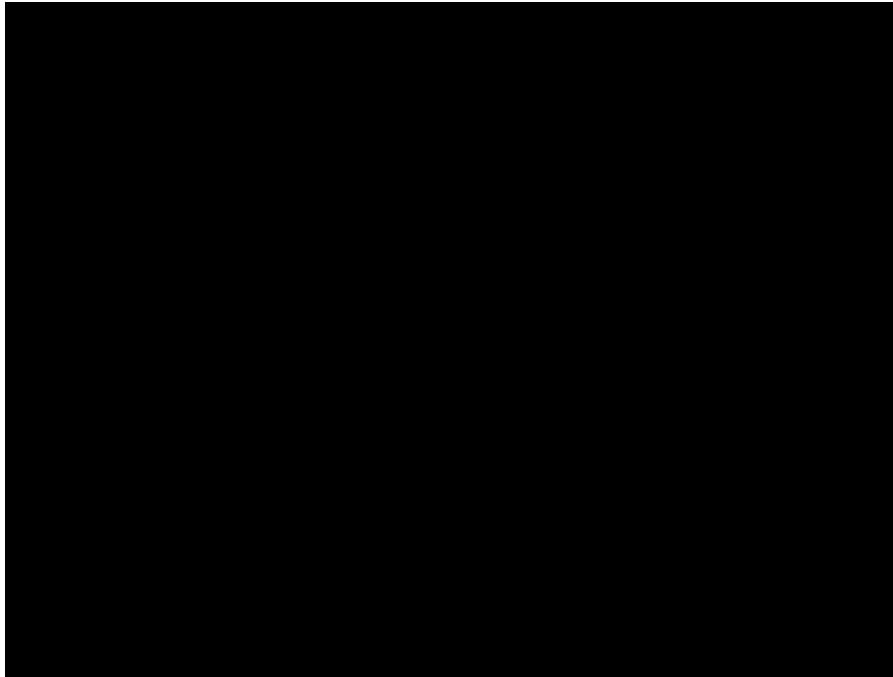
L2 (Euclidean) distance

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



$K = 1$

K-Nearest Neighbors: try it yourself!



<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 algorithms themselves.

Very problem/dataset-dependent.

Must try them all out and see what works best.

Setting Hyperparameters

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

train

Setting Hyperparameters

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

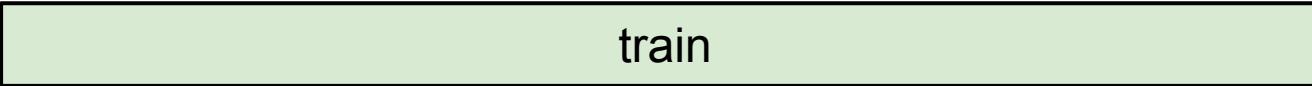
BAD: $K = 1$ always works
perfectly on training data

train

Setting Hyperparameters

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

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



train

Idea #2: choose hyperparameters that work best on **test data**

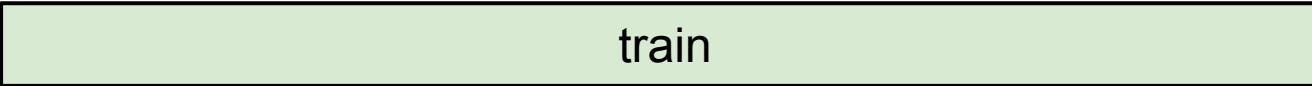


train

test

Setting Hyperparameters

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



train

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

Idea #2: choose hyperparameters that work best on **test data**



train

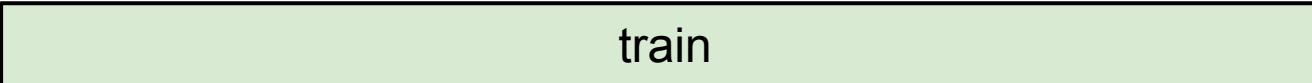
BAD: No idea how algorithm will perform on new data

Never do this!

Setting Hyperparameters

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

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



train

Idea #2: 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**; choose hyperparameters on val and evaluate on test

Better!



train

validation

test

Setting Hyperparameters

train

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
--------	--------	--------	--------	--------	------

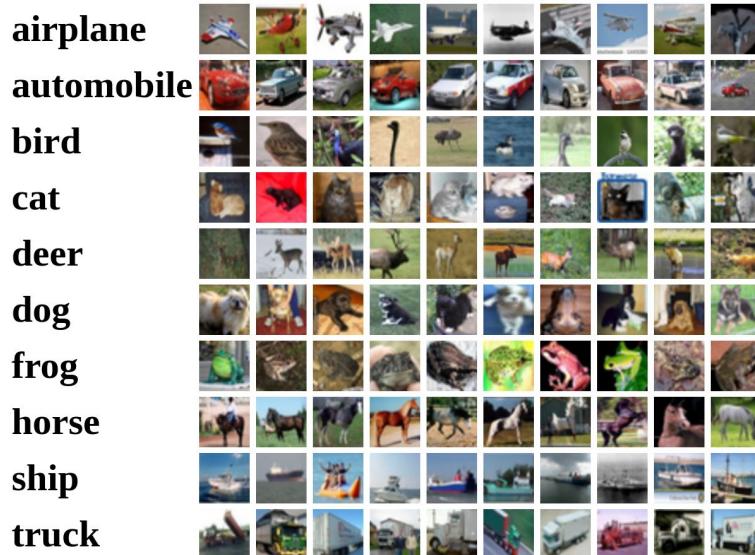
Useful for small datasets, but not used too frequently in deep learning

Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images



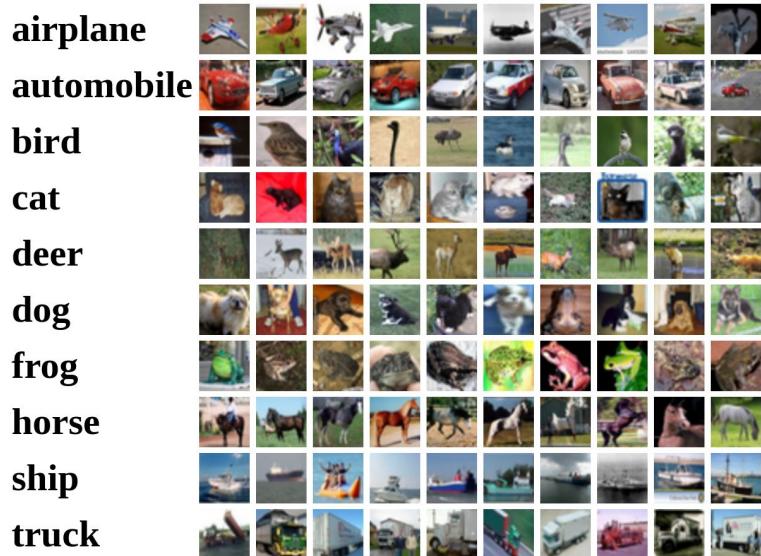
Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Example Dataset: CIFAR10

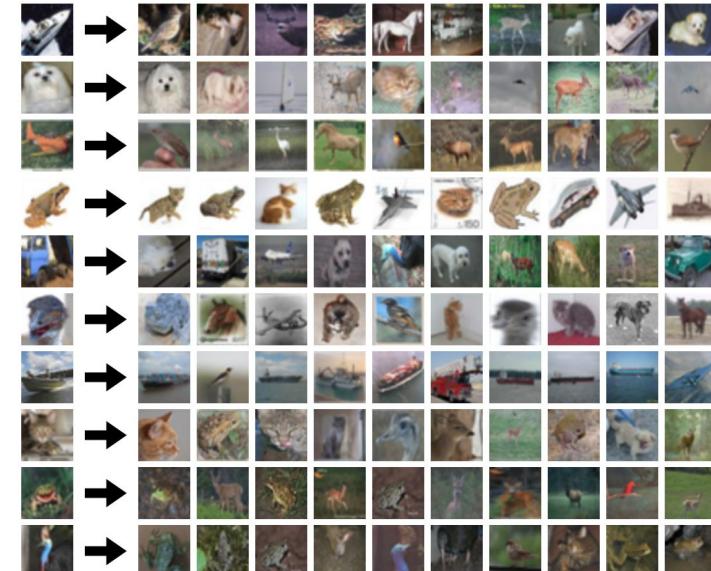
10 classes

50,000 training images

10,000 testing images

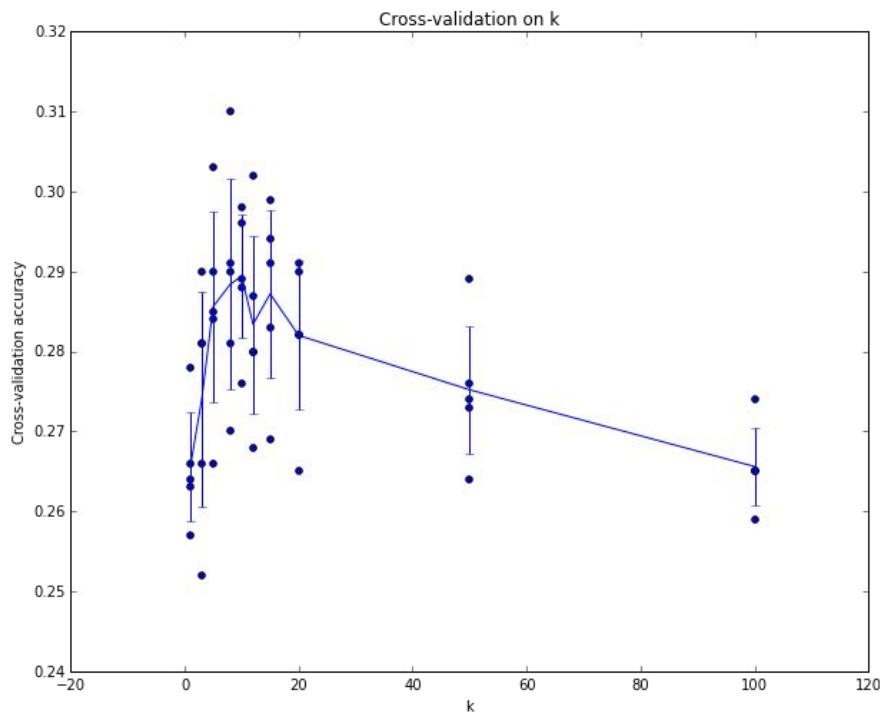


Test images and nearest neighbors



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

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 $k \sim 7$ works best
for this data)

What does this look like?



What does this look like?



k-Nearest Neighbor with pixel distance **never used**.

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

Original



Occluded



Shifted (1 pixel)



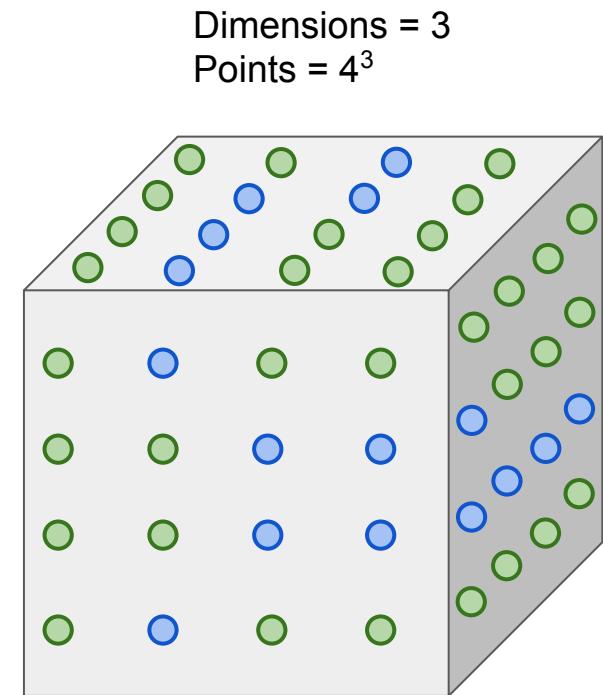
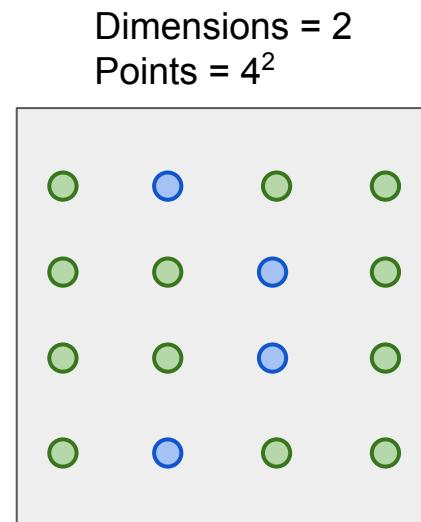
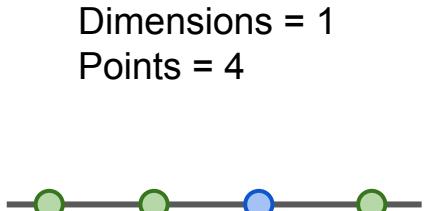
Tinted



[Original image](#) is
CC0 public domain

k-Nearest Neighbor with pixel distance **never used**.

- Curse of dimensionality



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 the K 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!

Linear Classifier

Parametric Approach

Image



Array of **32x32x3** numbers
(3072 numbers total)

$$\xrightarrow{f(x, W)}$$

W
parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier

Image



$$f(x, W) = Wx$$

Array of **32x32x3** numbers
(3072 numbers total)

$$f(\mathbf{x}, \mathbf{W})$$

W
parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier



Image

Array of **32x32x3** numbers
(3072 numbers total)

$$f(x, W) = \boxed{W} \boxed{x}$$

10x1 **10x3072**

$$f(\boxed{x}, \boxed{W})$$

W
parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier



Image

Array of **32x32x3** numbers
(3072 numbers total)

$$f(x, W) = \boxed{W} \boxed{x} + \boxed{b}$$

10x1 **3072x1**
10x1 **10x3072**

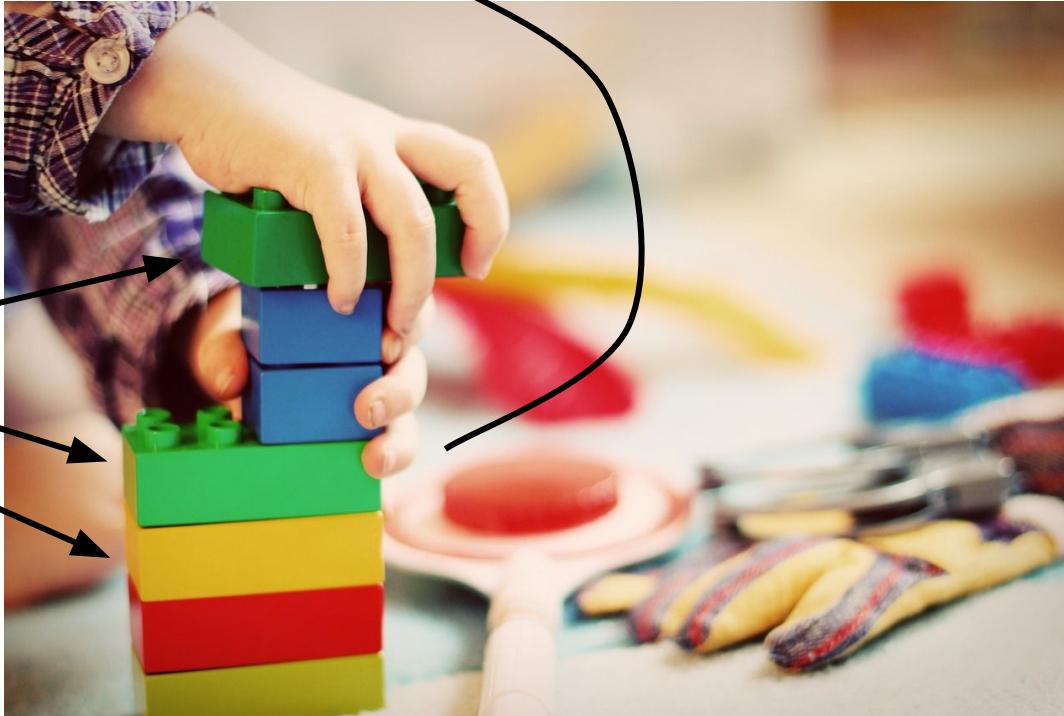
$$f(\boxed{x}, \boxed{W})$$

10 numbers giving
class scores

W
parameters
or weights

Neural Network

Linear
classifiers



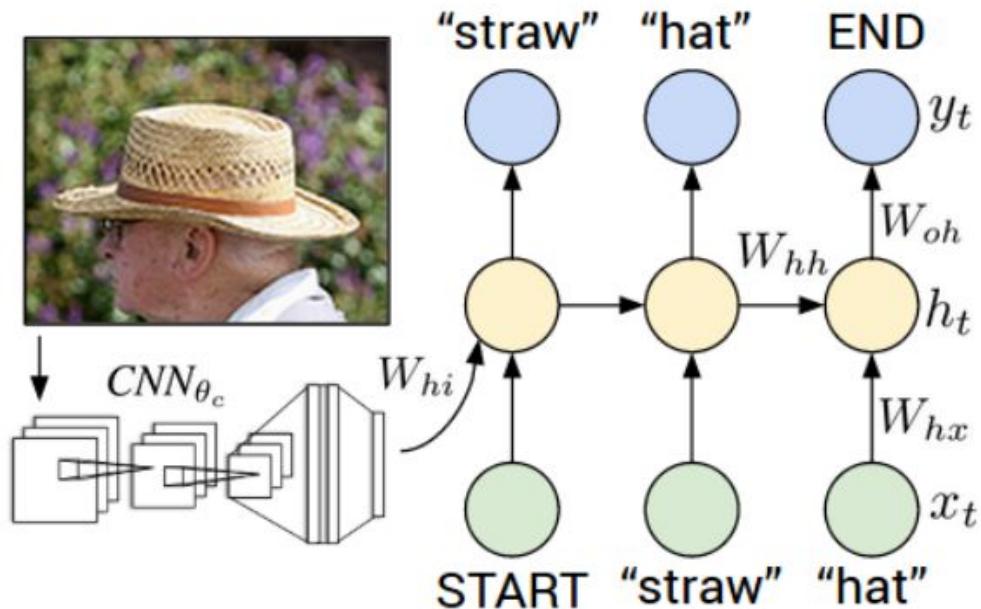
[This image](#) is [CC0 1.0](#) public domain

Two young girls are playing with lego toy. *Boy is doing backflip on wakeboard*

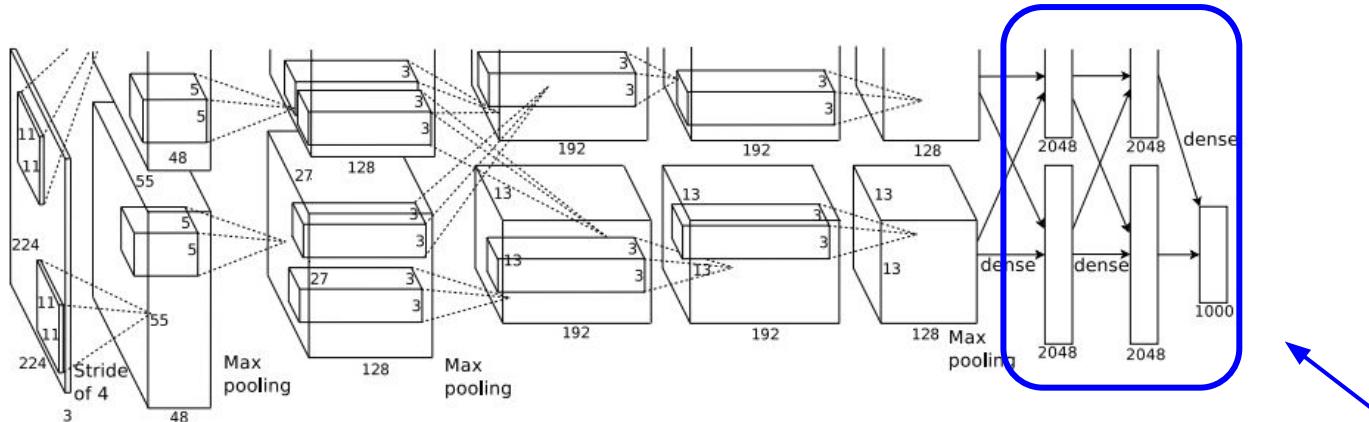


Man in black shirt is playing guitar.

Construction worker in orange safety vest is working on road.

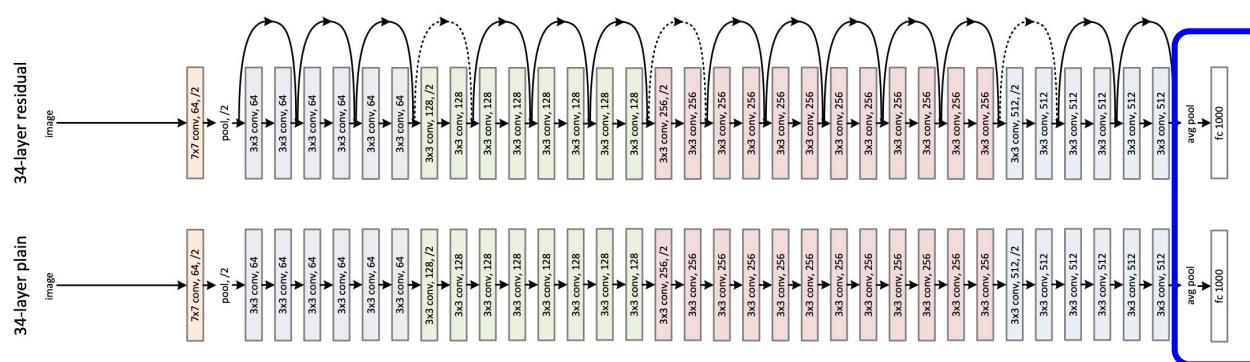


Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figures copyright IEEE, 2015. Reproduced for educational purposes.



[Krizhevsky et al. 2012]

Linear layers



[He et al. 2015]

Recall CIFAR10

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck

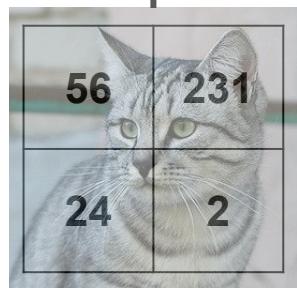


50,000 training images
each image is **32x32x3**

10,000 test images.

Example with an image with 4 pixels, and 3 classes (**cat/dog/ship**)

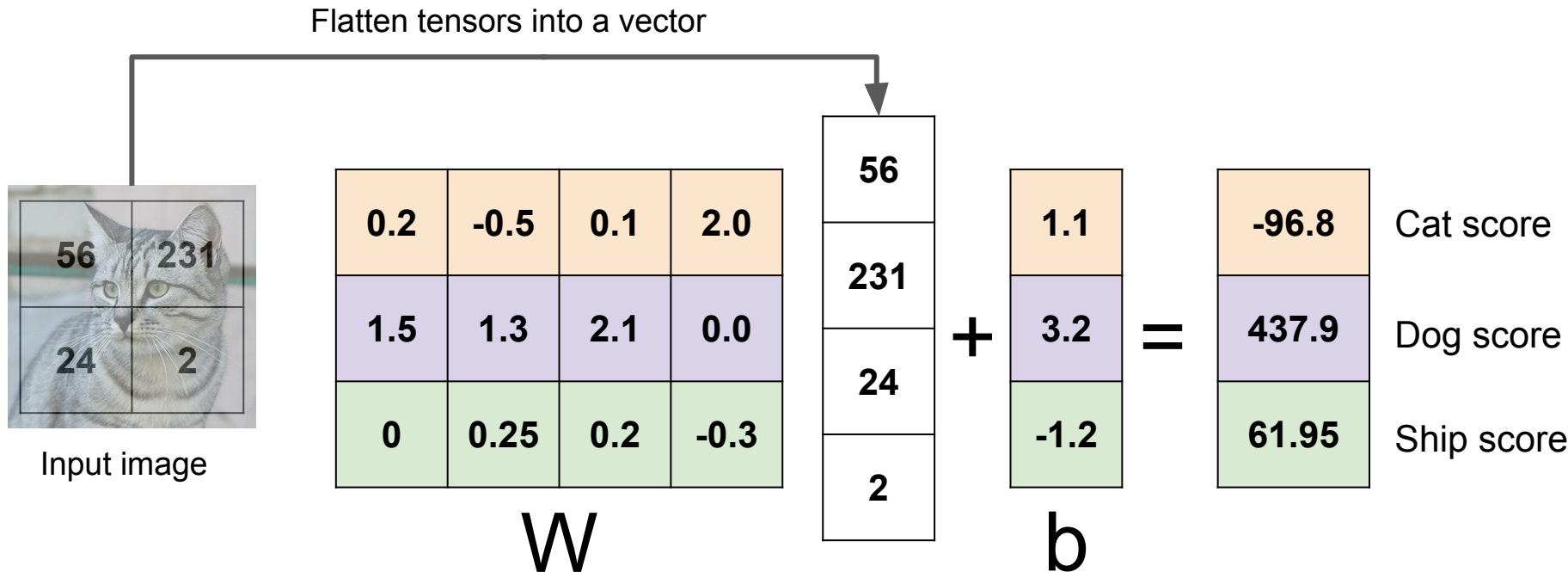
Flatten tensors into a vector



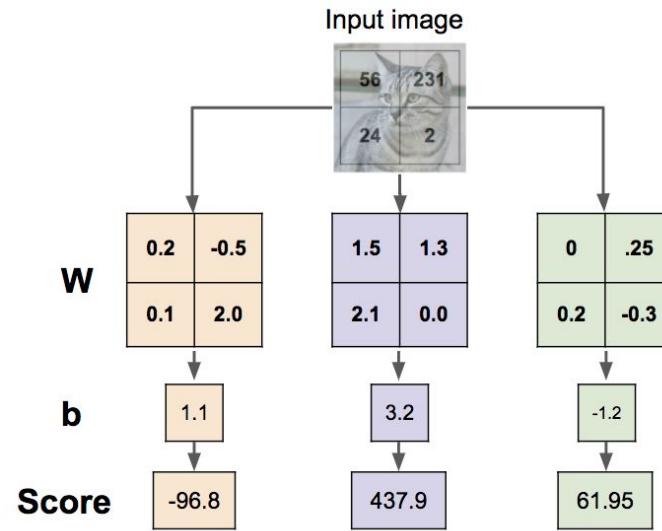
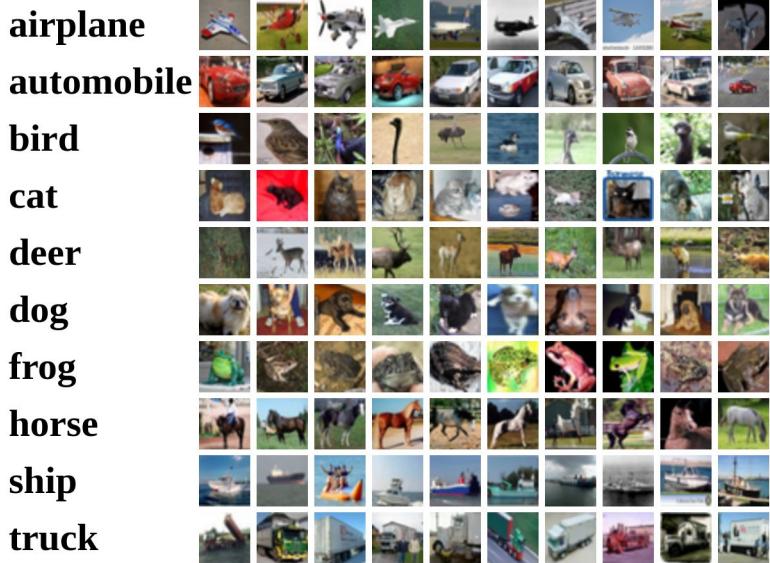
Input image



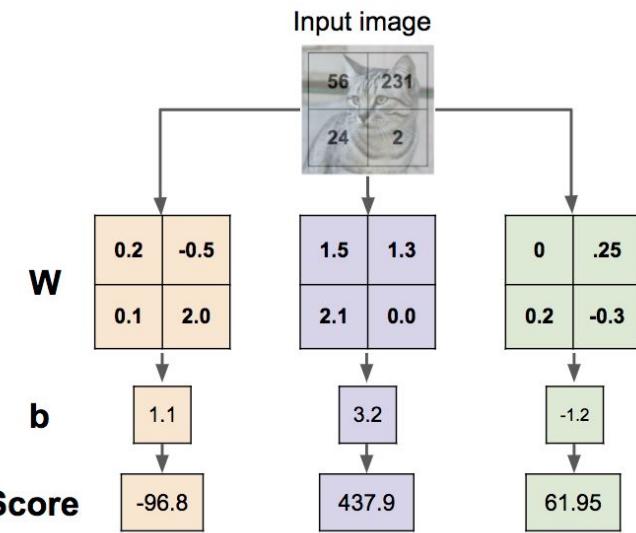
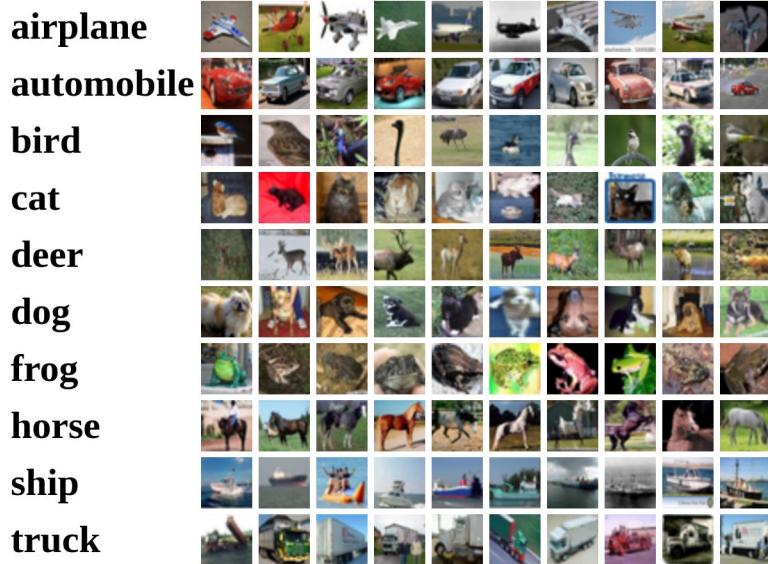
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



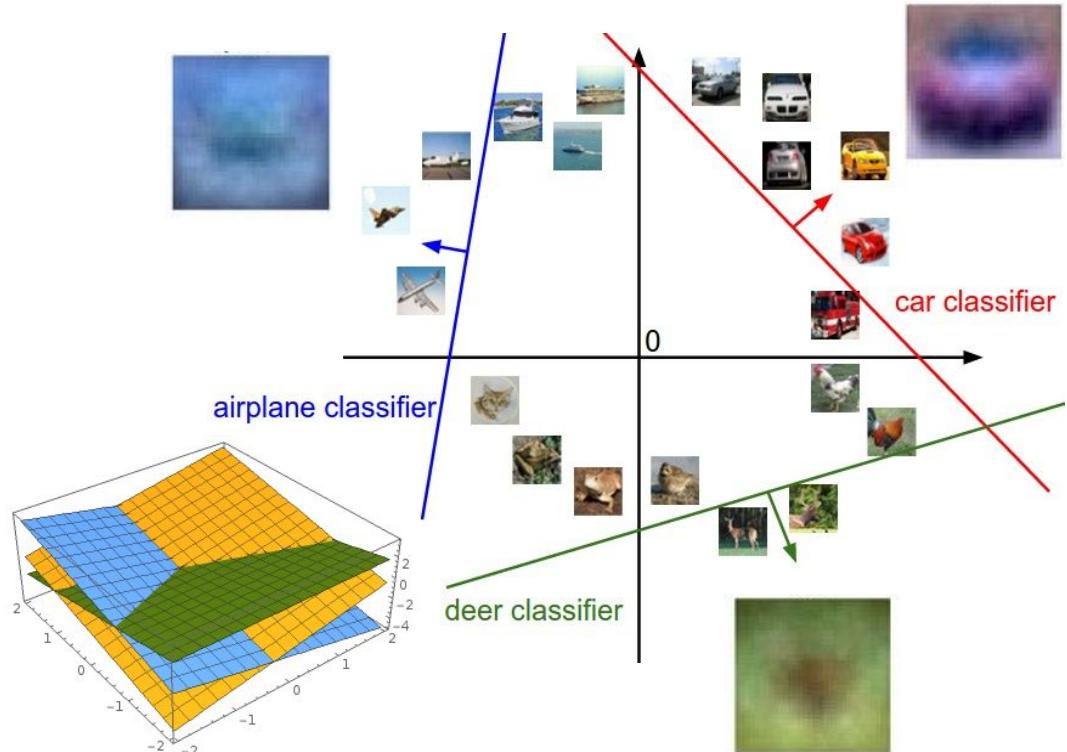
Interpreting a Linear Classifier



Interpreting a Linear Classifier: Visual Viewpoint



Interpreting a Linear Classifier: Geometric Viewpoint



Plot created using [Wolfram Cloud](#)

$$f(x, W) = Wx + b$$



Array of **32x32x3** numbers
(3072 numbers total)

[Cat image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#)

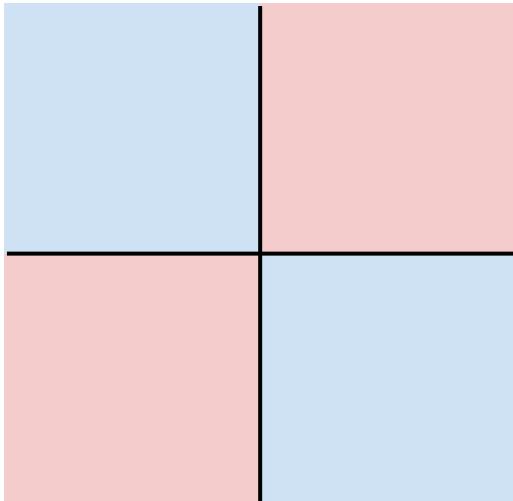
Hard cases for a linear classifier

Class 1:

First and third quadrants

Class 2:

Second and fourth quadrants

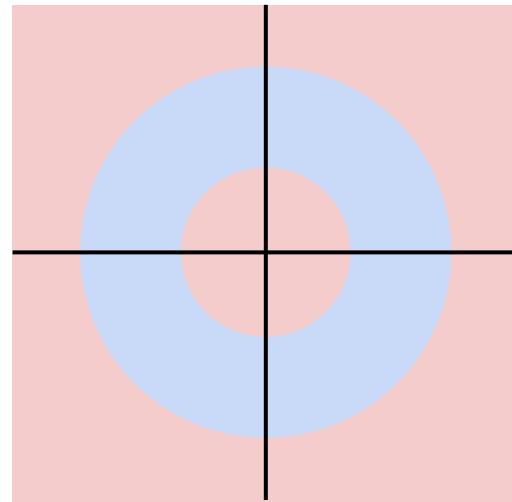


Class 1:

$1 \leq L_2 \text{ norm} \leq 2$

Class 2:

Everything else

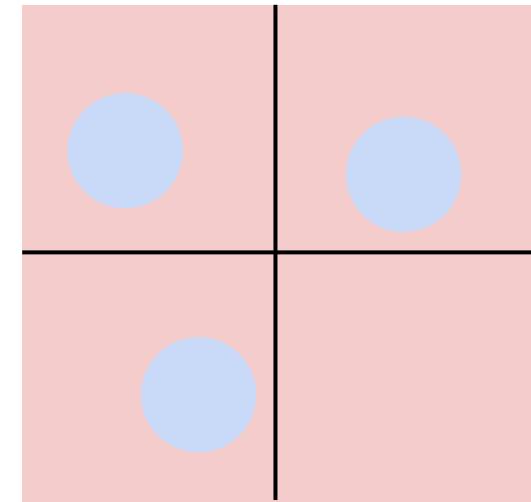


Class 1:

Three modes

Class 2:

Everything else



$$f(x, W) = Wx + b$$

Coming up:

- Loss function
- Optimization
- ConvNets!

(quantifying what it means to have a “good” W)

(start with random W and find a W that minimizes the loss)

(tweak the functional form of f)