

Lecture 2:

Image Classification

Administrative: Assignment 0

- Due 4/08 by 11:59pm
- Easy assignment
- Hardest part is learning how to use colab and how to submit on gradescope
- Worth 0% of your grade
- Used to evaluate how prepared you are to take this course

Administrative: Assignment 1

Due 4/16 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax

Administrative: Course Project

Project proposal due 4/29 11:59pm

Find your teammates on EdStem. We will help find teammates as well.

Collaboration: EdStem

“Is X a valid project for 493G1?”

- Anything related to **deep learning + pixels**
- Maximum of 3 students per team
- Make a EdStem private post or come to TA Office Hours

More info on the website

Administrative: Fridays

This Friday 9:30-10:30am and again 12:30-1:30pm

Broadcasting

Presenter: Tanush (Head TA)

Syllabus

Deep learning Fundamentals

Data-driven approaches
Linear classification & kNN
Loss functions
Optimization
Backpropagation
Multi-layer perceptrons
Neural Networks
Convolutions
RNNs / LSTMs
Transformers

Practical training skills

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

Applications

Image captioning
Interpreting machine learning
Generative AI
Fairness & ethics
Data-centric AI
Deep reinforcement learning
Self-supervised learning
Diffusion
LLMs

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



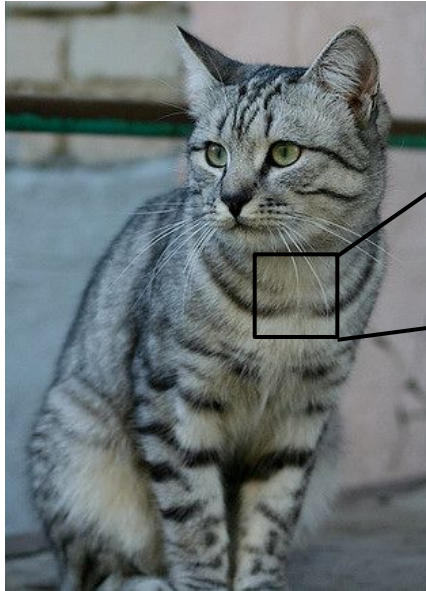
This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

(assume given a set of possible labels)



Cat
Dog
Bird
Truck
Plane

The Problem: Semantic Gap



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

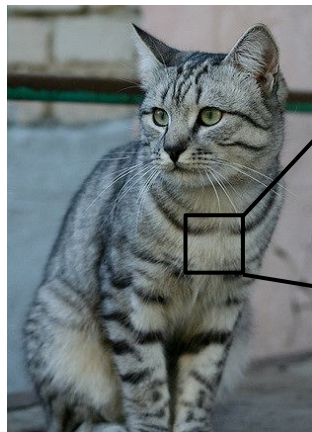
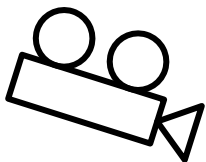
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]  
[ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]  
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]  
[ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]  
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]  
[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]  
[115 114 109 123 150 148 131 110 113 109 100 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

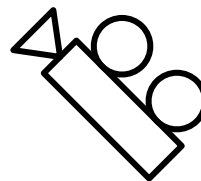
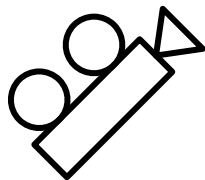
An image is a tensor of integers
between [0, 255]:

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

Challenges: Viewpoint variation



[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	90	115	112	106	103	99	85]
[99	81	91	93	120	131	127	100	95	98	102	99	96	93	101	94]
[106	91	61	64	69	91	88	85	101	107	109	90	75	84	96	95]
[114	108	85	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	140	137	119	121	117	94	65	79	80	65	54	64	72	90]
[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	100	147	131	118	113	114	113	100	106	95	77	80]
[63	77	86	81	77	79	102	123	117	115	117	125	125	130	115	87]
[62	65	82	80	78	71	80	101	124	126	119	101	107	114	131	119]
[63	65	75	80	89	71	62	81	120	130	135	105	61	90	110	110]
[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	63	86	114	132	112	97	69	55	70	82	99	94]
[130	120	134	161	139	100	109	118	121	134	114	87	65	53	69	86]
[120	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	140	153	102	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: Illumination



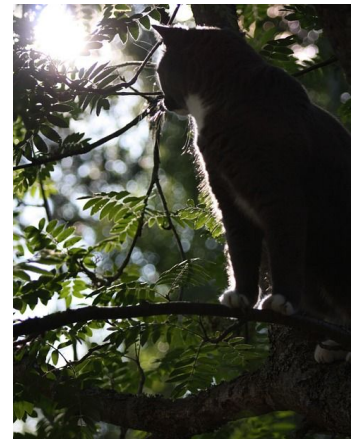
[This image](#) is [CC0 1.0](#) public domain



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



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



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

RGB values are a function of surface materials, color, light source, etc.

Challenges: Background Clutter



[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

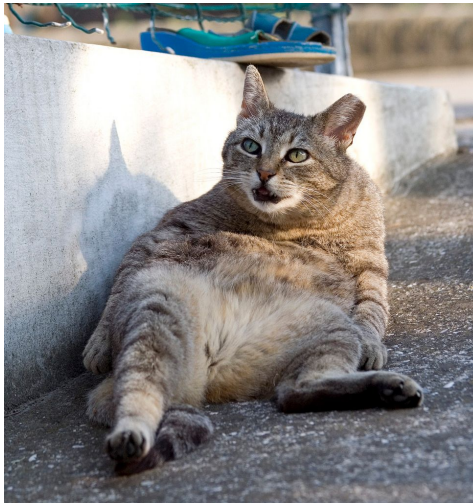


[This image](#) is [CC0 1.0](#) public domain



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

Challenges: Deformation



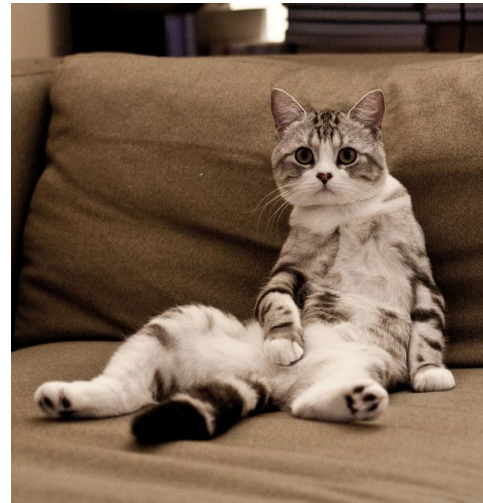
This image by [Umberto Salvagnin](#) is licensed under [CC-BY 2.0](#)



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](#)

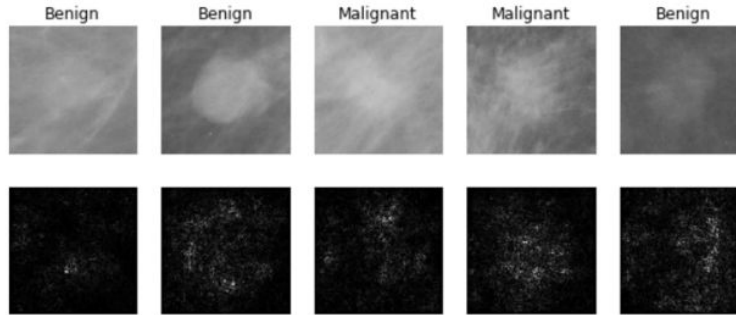
Challenges: Intraclass variation



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

Image classification is a building block for other tasks

Medical Imaging



Levy et al, 2016 Figure reproduced with permission

Galaxy Classification



Dieleman et al, 2014

From left to right: public domain by NASA, image generated by ESA/Hubble, public domain by NASA, and public domain

Whale recognition



[Kaggle Challenge](#)

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Image classification is a building block for other tasks



A white teddy bear sitting in the grass



A man in a baseball uniform throwing a ball



A woman is holding a cat in her hand

Image Captioning

Vinyals et al, 2015

Karpathy and Fei-Fei, 2015



A man riding a wave on top of a surfboard



A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
<https://pixabay.com/en/agege-antique-cat-154300/>
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>
<https://pixabay.com/en/surf-wave-summer-sport-sports-1668718/>
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>
<https://pixabay.com/en/handstand-lake-meditation-496008/>
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [NeuralNet2](#)

Image classification is a building block for other tasks

Example: Playing Go



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



(1, 1)

(1, 2)

...

(1, 19)

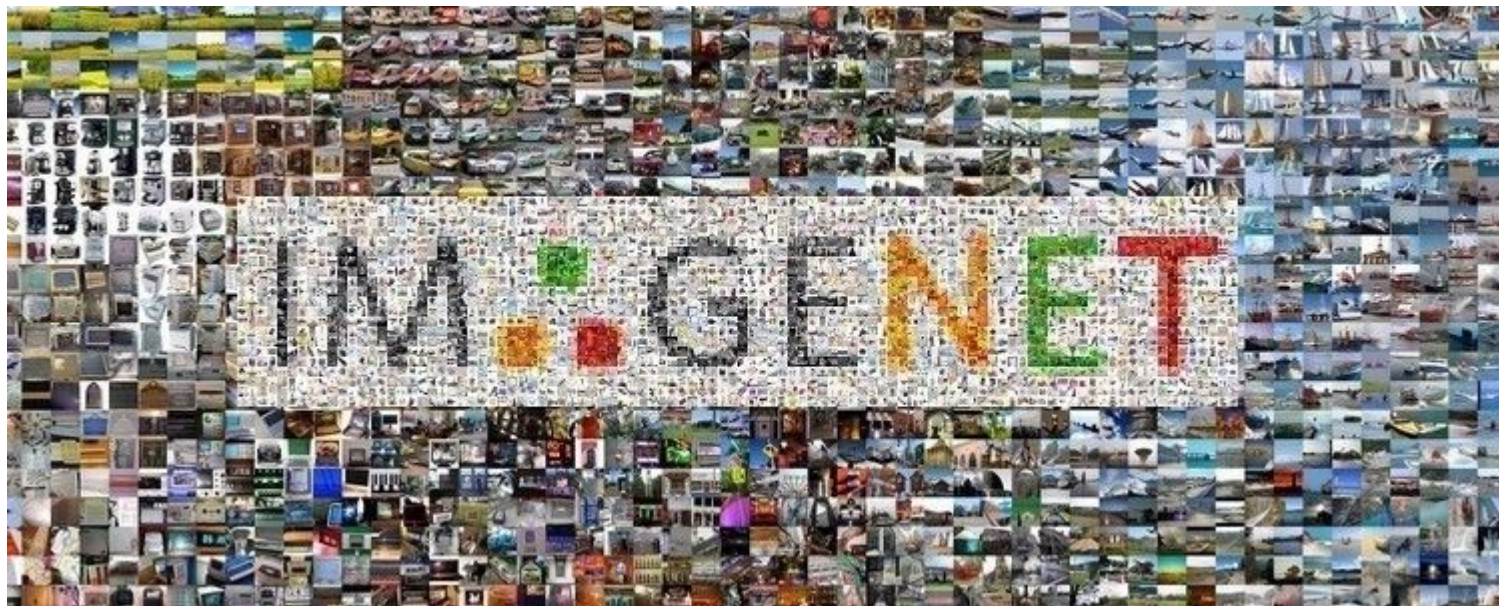
...

(19, 19)

Where to
play next?

Modern computer vision algorithms

Classifiers today take 1ms to classify images. And can handle an unlimited (open-set) of categories.



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

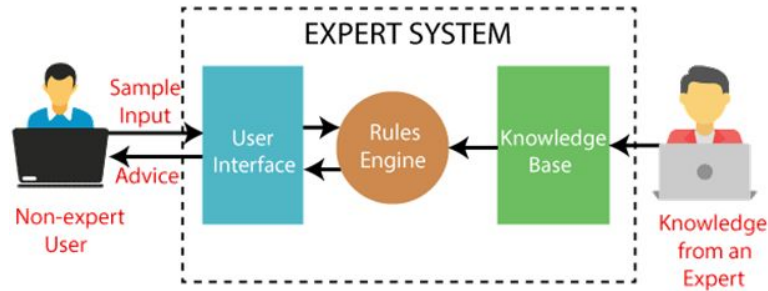
An image classifier: can we implement this as a normal software function?

```
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.

This is why expert systems in the 80s led to the AI winter.

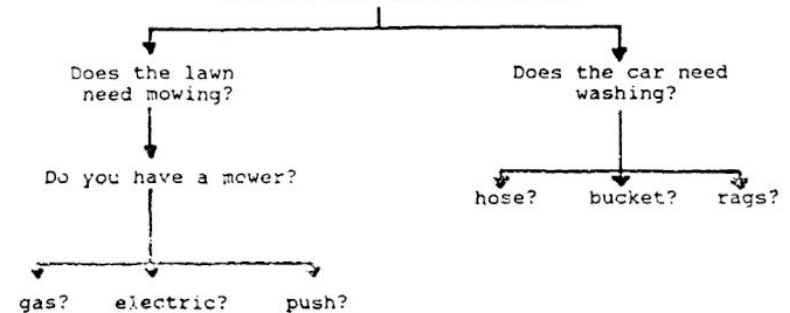


Originally called heuristic programming project.

BACKWARD CHAINING

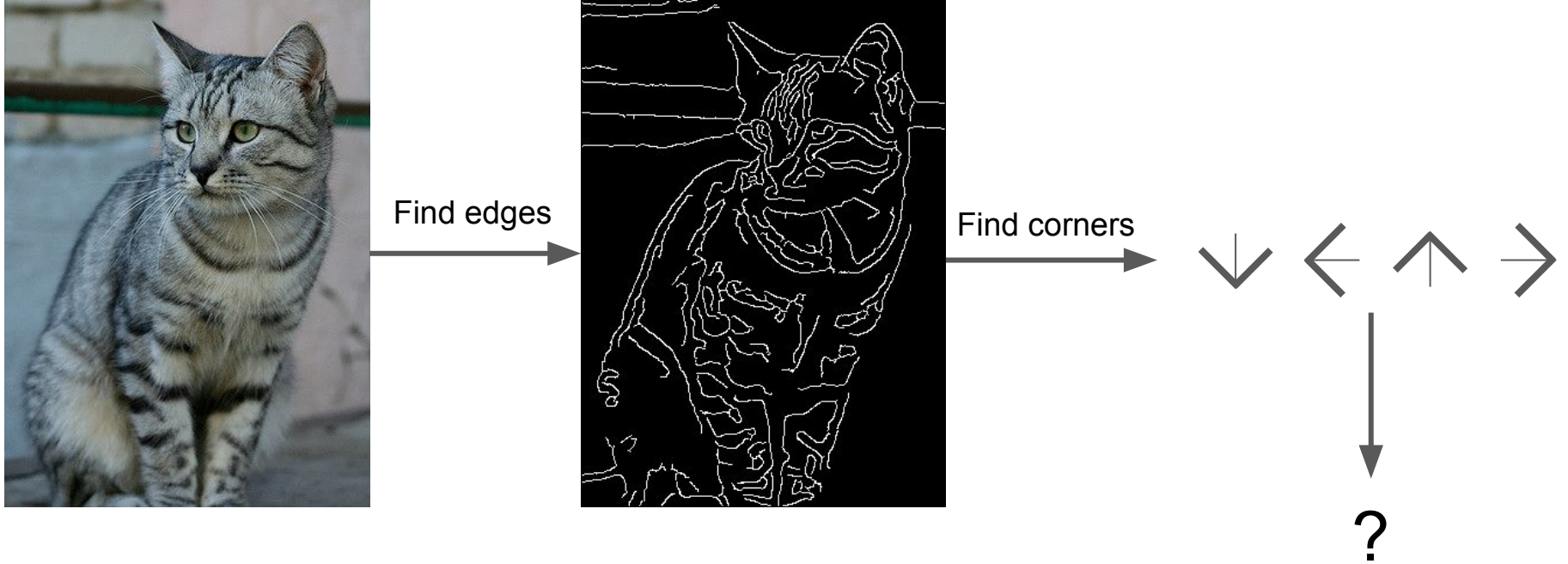
GOAL: Make \$20.00

RULE: If the lawn is shaggy and the car is dirty and you mow the lawn and wash the car, then Dad will give you \$20.00



*** The inference engine will test each rule or ask the user for additional information.

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

Example training set

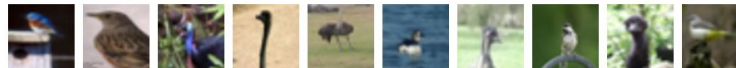
airplane



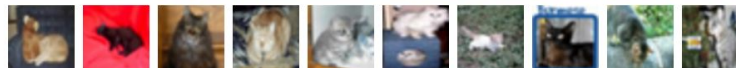
automobile



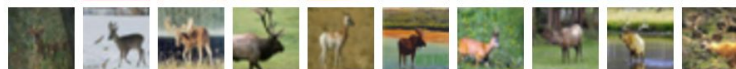
bird



cat



deer



Example dataset: MNIST



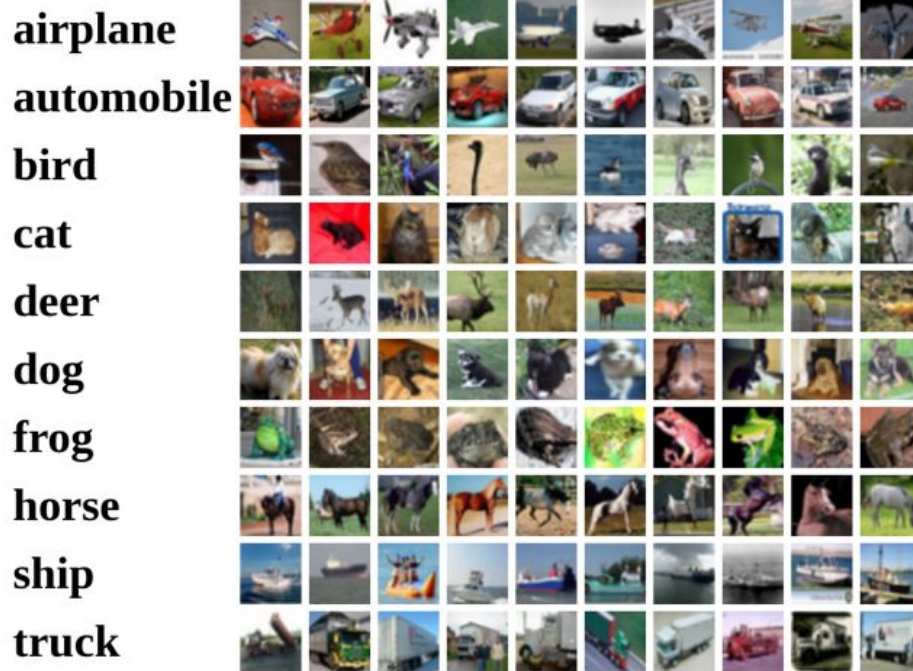
10 classes: Digits 0 to 9

28x28 grayscale images

50k training images

10k test images

Example dataset: CIFAR10



10 classes

50k training images (5k per class)

10k testing images (1k per class)

32x32 RGB images

We will use this dataset for
homework assignments

Example dataset: ImageNet (ILSVRC challenge)

ILSVRC = ImageNet Large-Scale Visual Recognition Challenge

1000 classes

~1.3M training images (~1.3K per class)

50K validation images (50 per class)

100K test images (100 per class)

Performance metric: **Top 5 accuracy**

Algorithm predicts 5 labels for each image; one of them needs to be right



flamingo



cock



ruffed grouse



quail



partridge

...



Egyptian cat



Persian cat



Siamese cat



tabby



lynx

...



dalmatian



keeshond



miniature schnauzer



standard schnauzer



giant schnauzer

Example dataset: MIT Places



365 classes of different scene types

~8M training images

18.25K val images (50 per class)

328.5K test images (900 per class)

Images have variable size, often
resize to **256x256** for training

Example dataset: Omniglot



1623 categories: characters
from 50 different alphabets

20 images per category

Meant to test **few shot learning**

Machine Learning: Data-Driven Approach

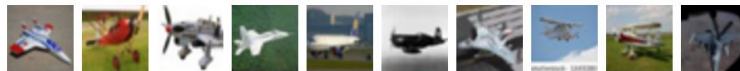
1. Collect a dataset of images and labels
2. Use Machine Learning algorithms to train a classifier
3. Evaluate the classifier on new images

Example training set

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

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

airplane



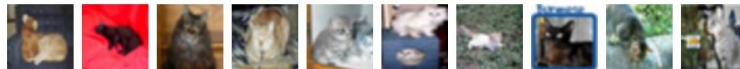
automobile



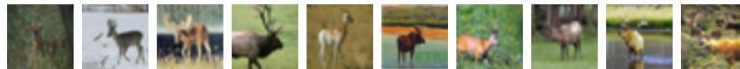
bird



cat



deer



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**



Training data with labels



query data

Distance Metric $\left| \begin{array}{c} \text{query cat} \end{array} \right|, \left| \begin{array}{c} \text{training cat} \end{array} \right| \rightarrow \mathbb{R}$

What is a
good
distance
metric?

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

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

```
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

```
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?

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

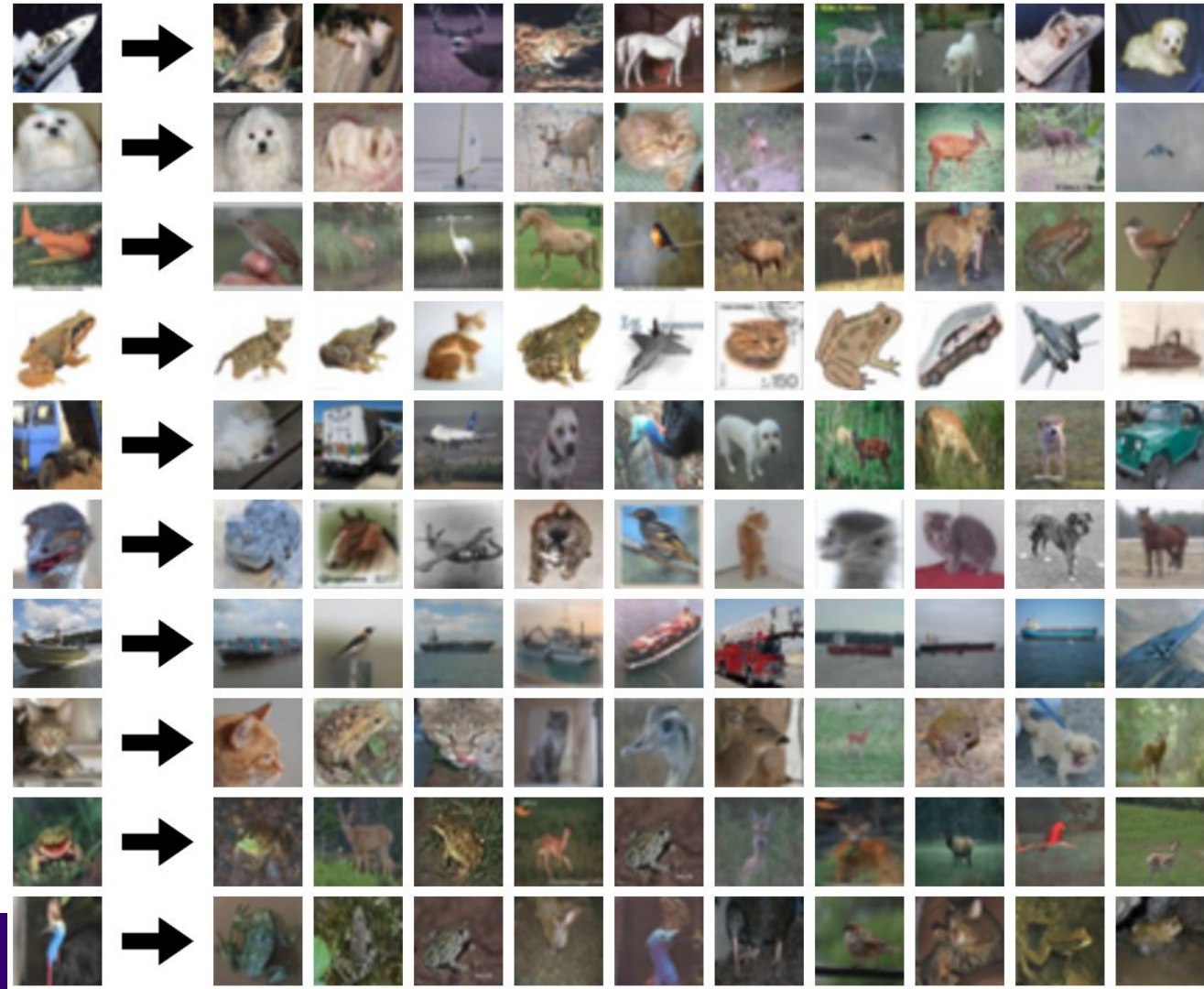
```

Nearest Neighbor classifier

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

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

Example
outputs from
a NN
classifier on
CIFAR:



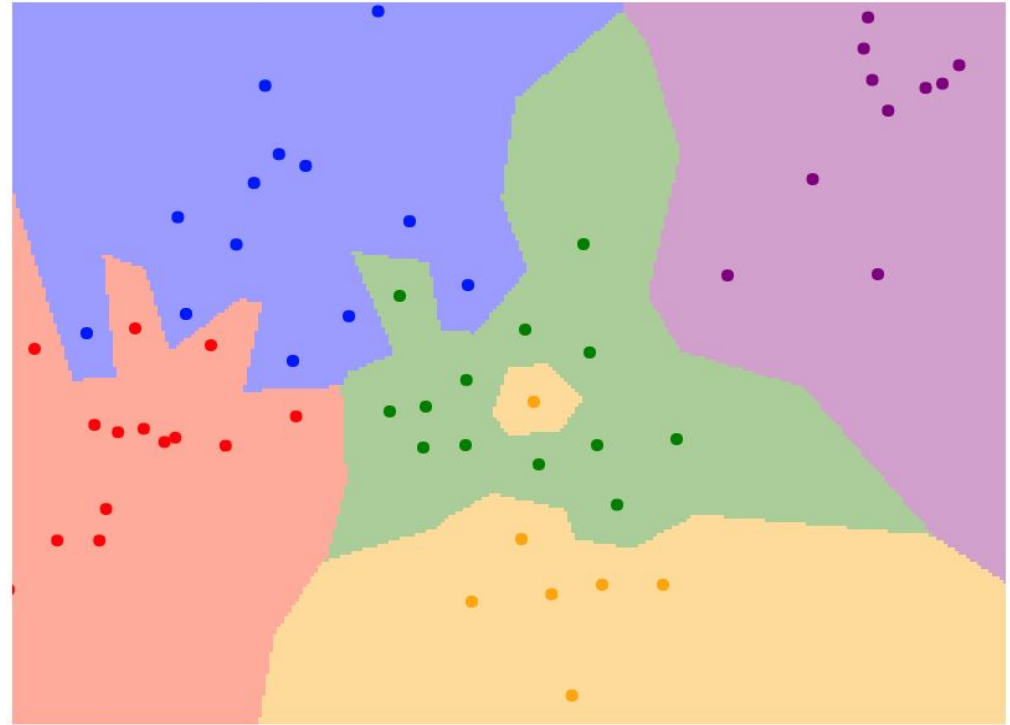
Example
outputs from
a NN
classifier on
CIFAR:



Assume each dot is a training image.

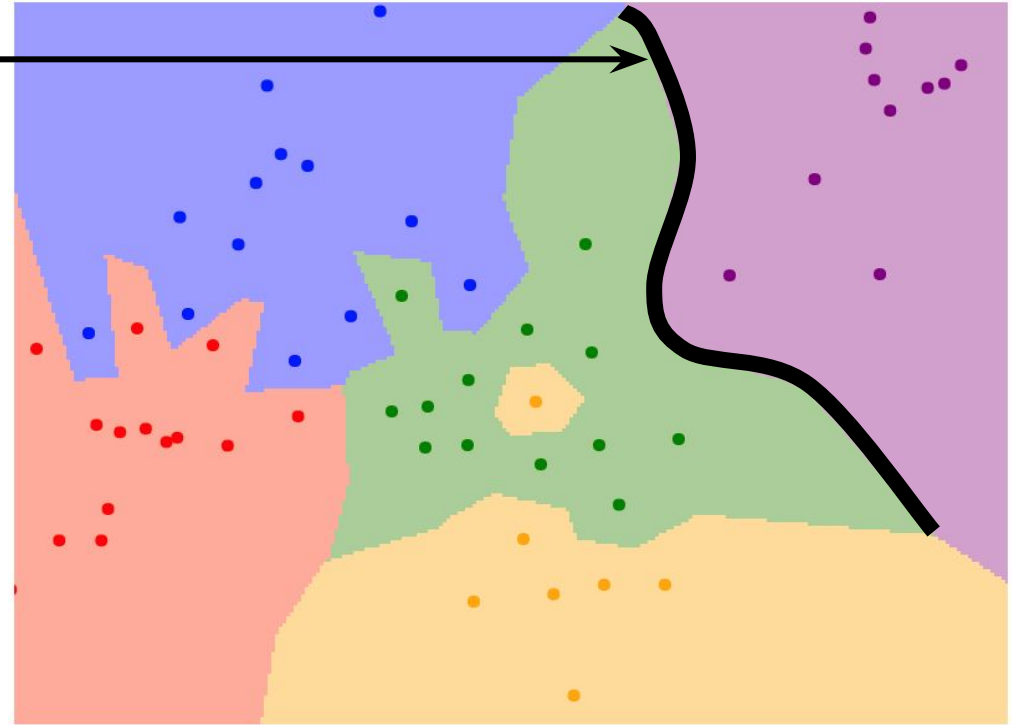
Assume all images are two dimensional.

What does this classifier look like?



1-nearest neighbor

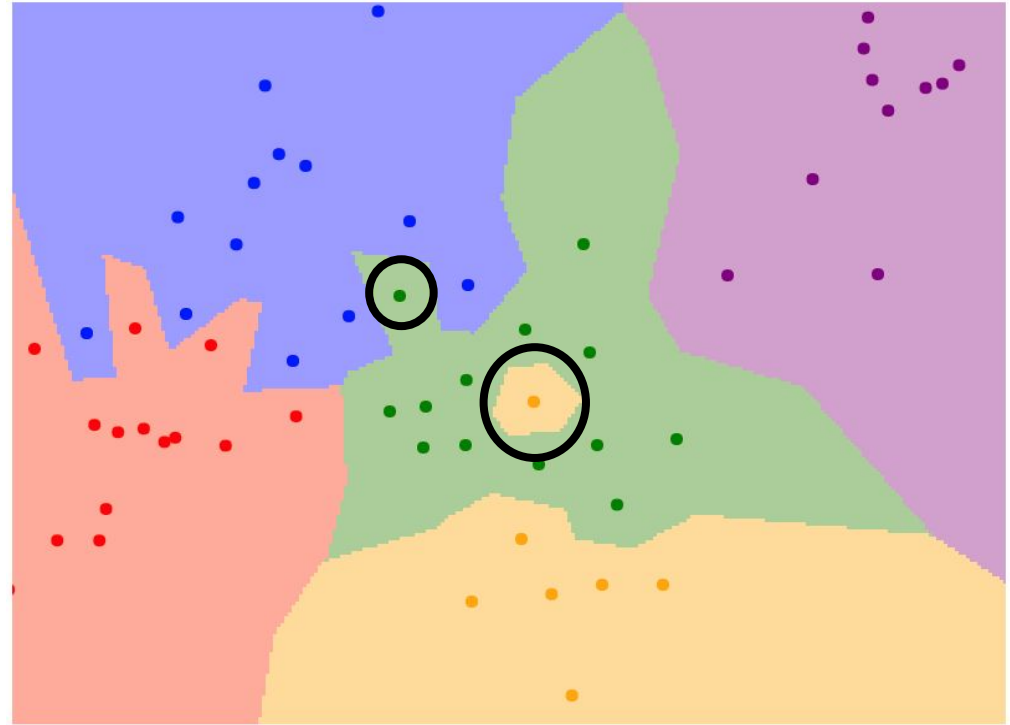
Decision boundary is the boundary between two classification regions



1-nearest neighbor

Yellow point in the middle of green might be mislabeled.

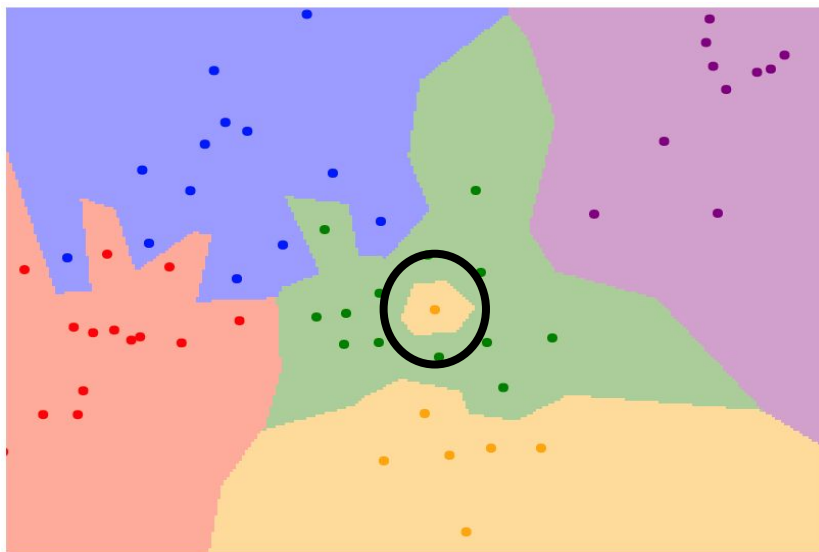
1-NN is not robust to label noise.



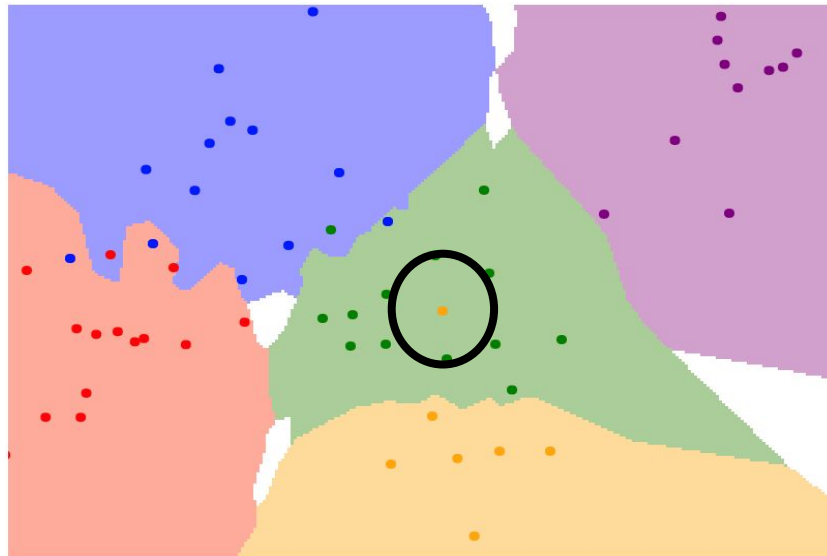
1-nearest neighbor

K-Nearest Neighbors

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



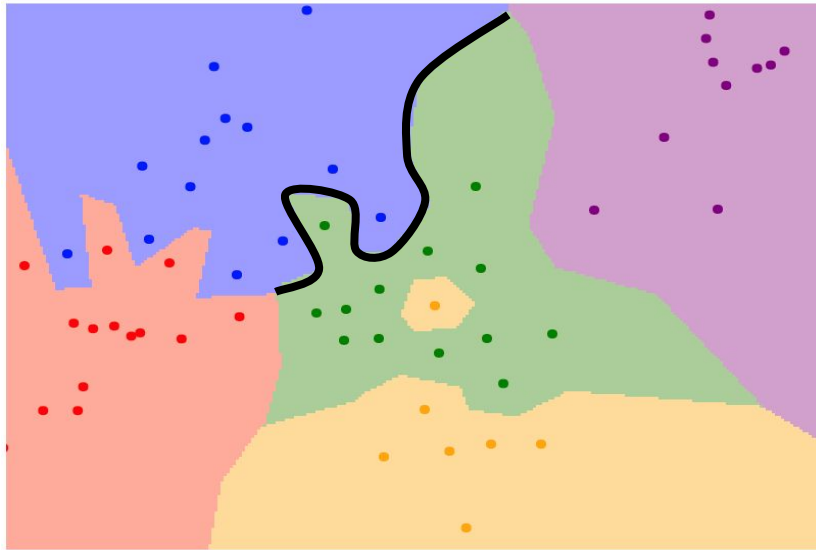
$K = 1$



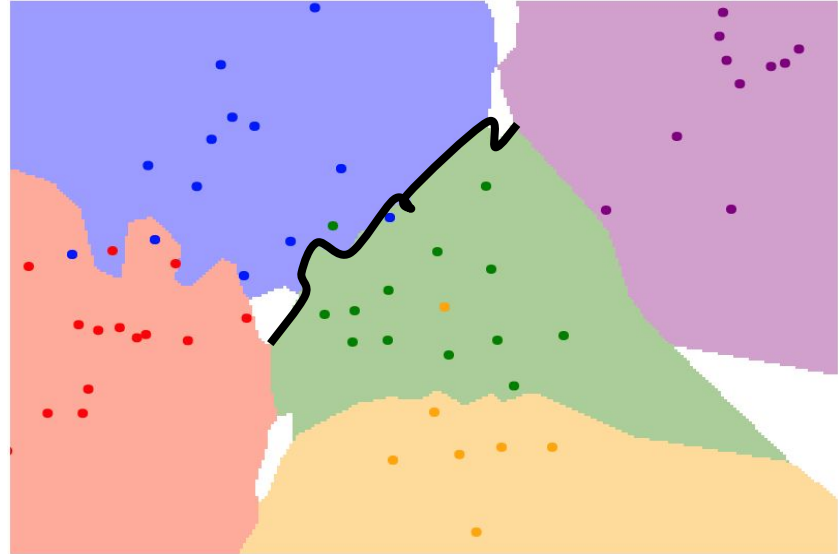
$K = 3$

K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries



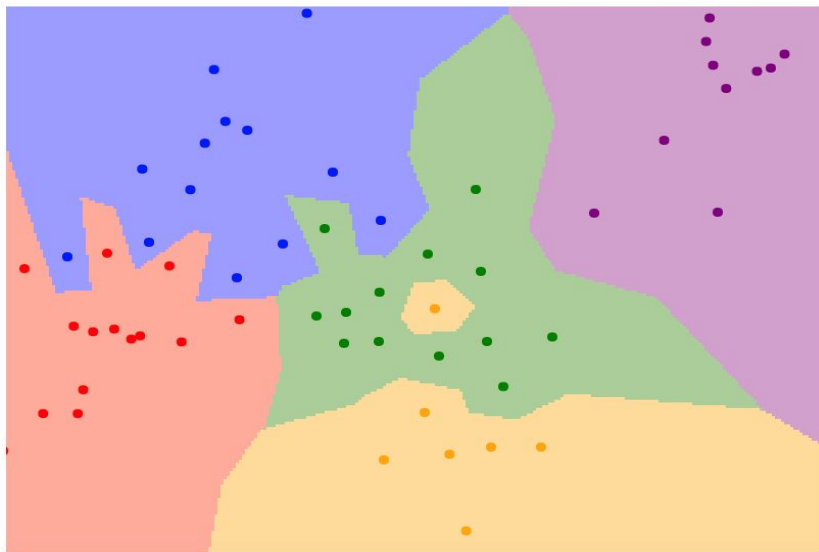
$K = 1$



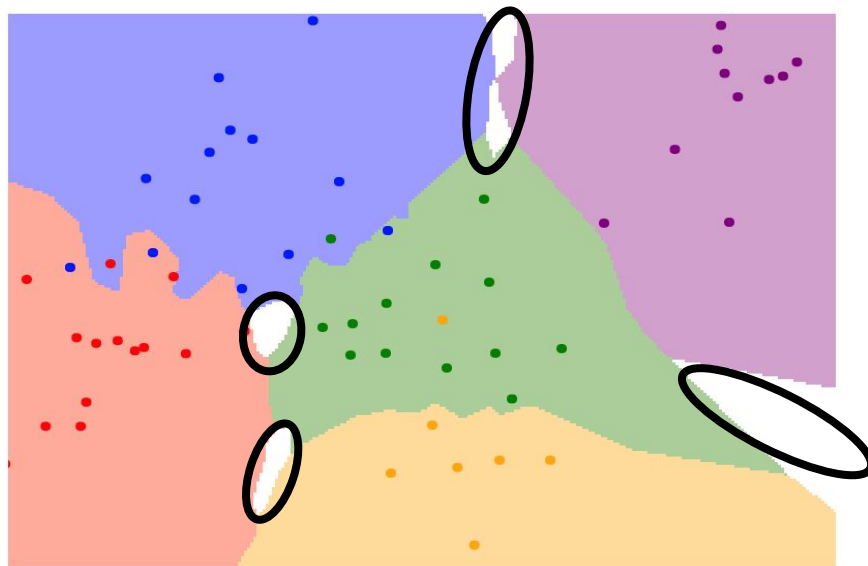
$K = 3$

K-Nearest Neighbors

Find more labels near uncertain white regions



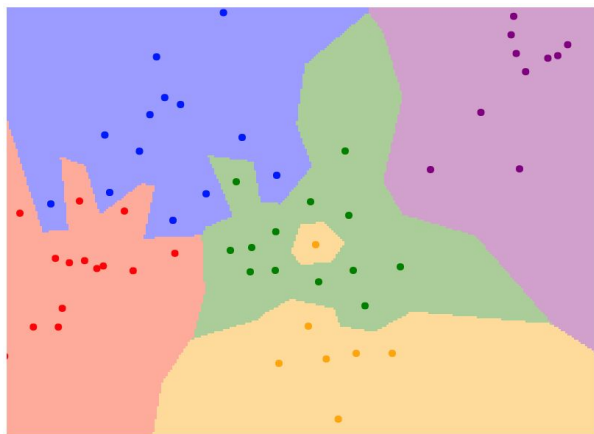
$K = 1$



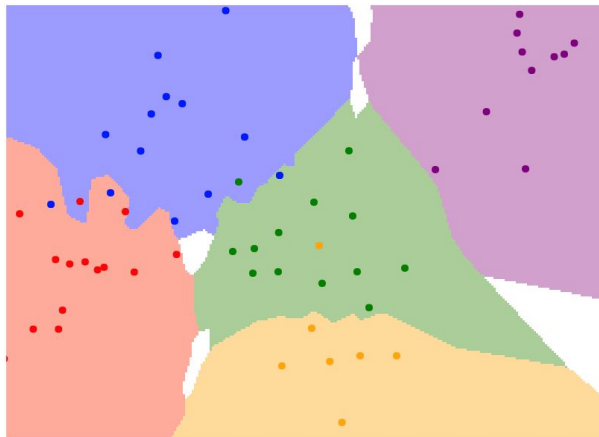
$K = 3$

K-Nearest Neighbors

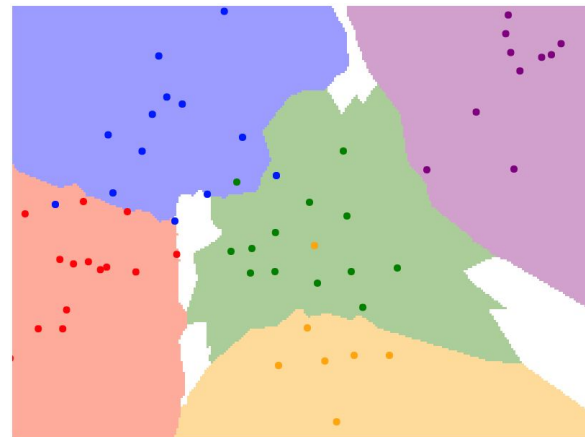
Larger K smooths boundaries more and leads to more uncertain regions



$K = 1$



$K = 3$

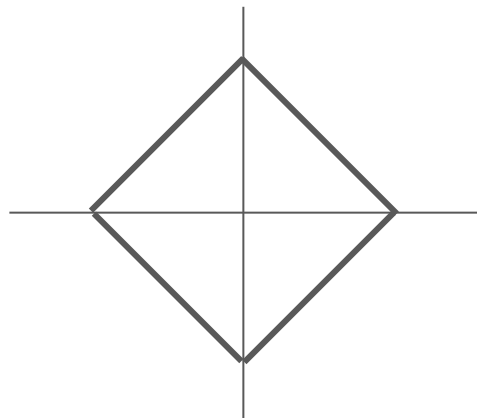


$K = 5$

K-Nearest Neighbors: Distance Metric

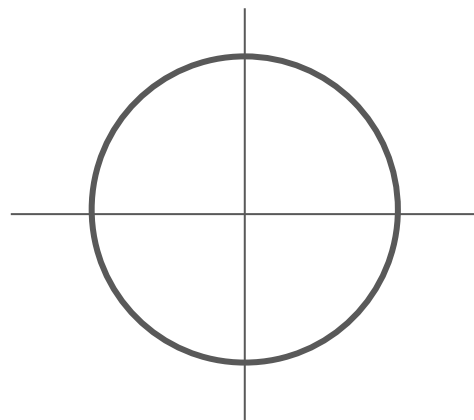
L1 (Manhattan) distance

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



L2 (Euclidean) distance

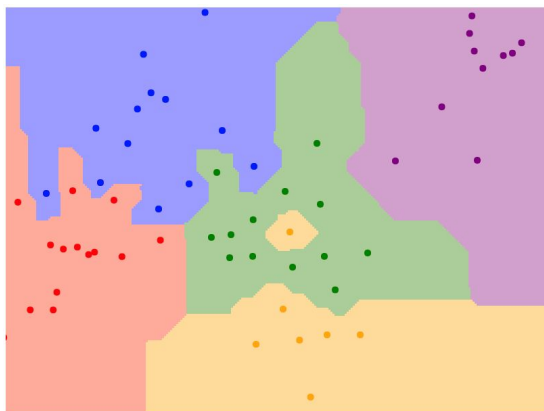
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

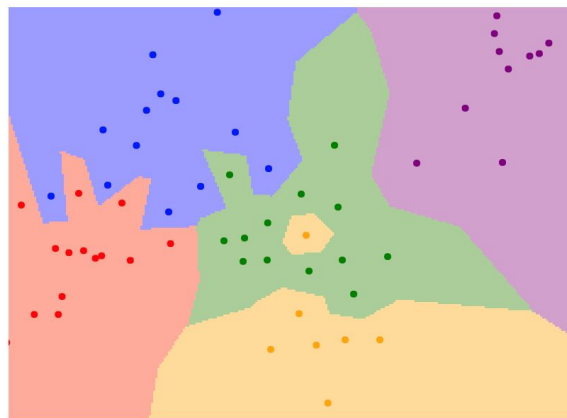
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



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 Neighbor with pixel distance is **never used**.

- Distance metrics on pixels are not informative

[Original image](#) is
[CC0 public domain](#)

Original



Occluded



Shifted (1 pixel)



Tinted



(All three images on the right have the same pixel distances to the one on the left)

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

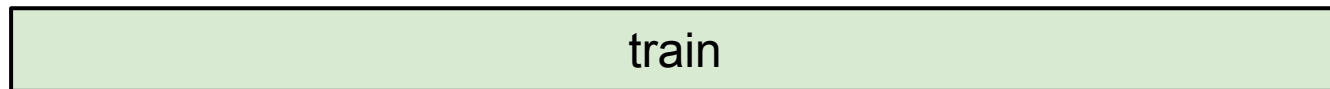


train

Setting Hyperparameters

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

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



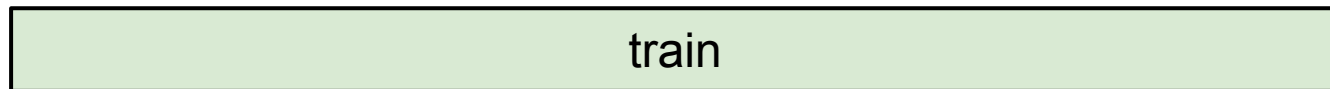
Idea #2: choose hyperparameters that work best on **test** data



Setting Hyperparameters

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

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



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

BAD: No idea how algorithm will perform on new data

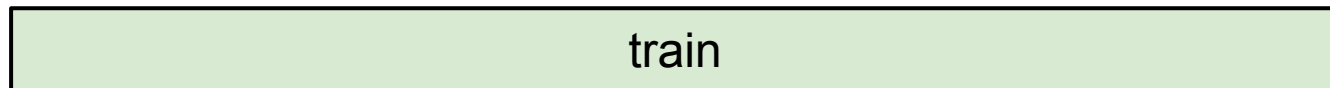


Never do this!

Setting Hyperparameters

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

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



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

Better!



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

airplane



automobile



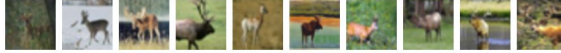
bird



cat



deer



dog



frog



horse



ship



truck



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

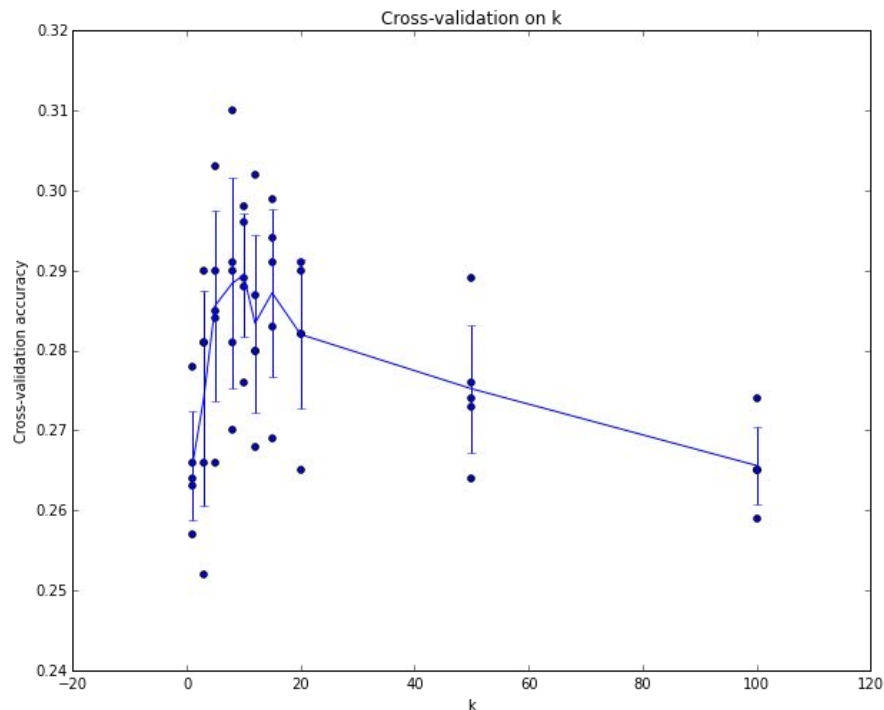


Test images and nearest neighbors



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**.

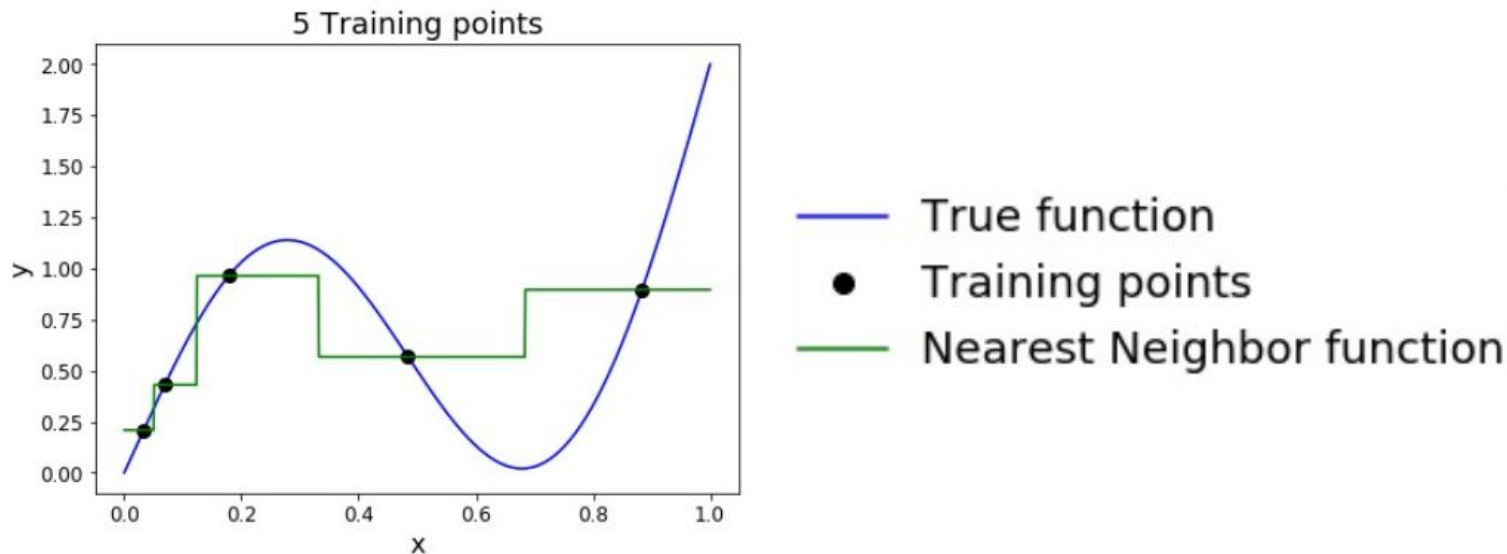
Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

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

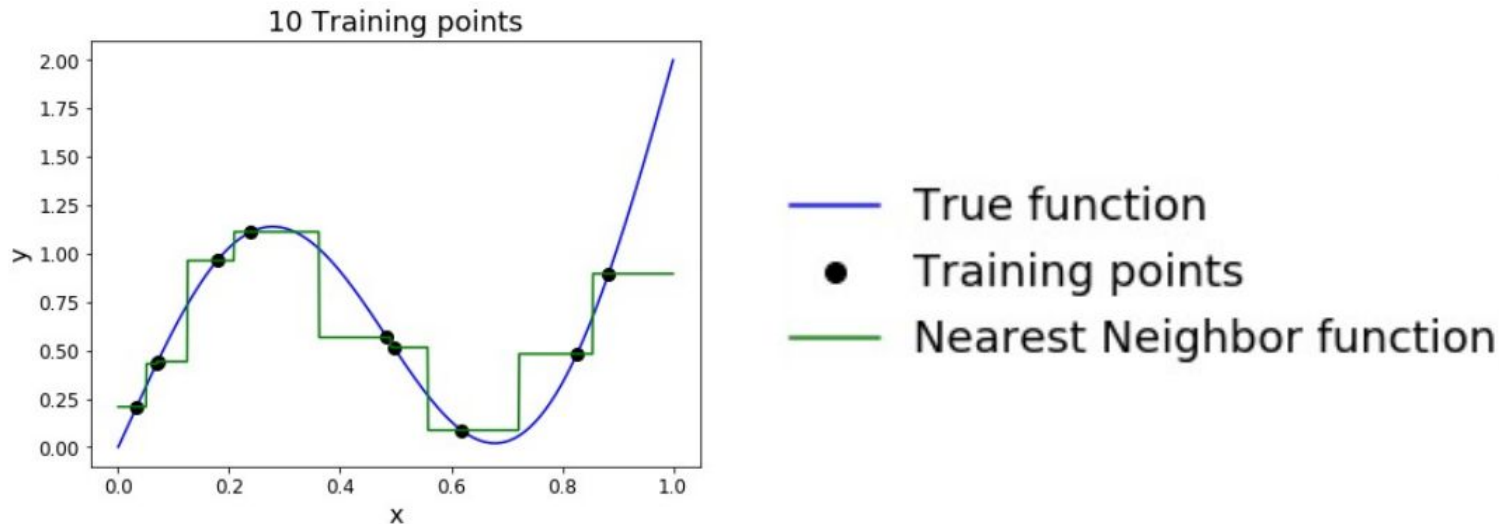
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



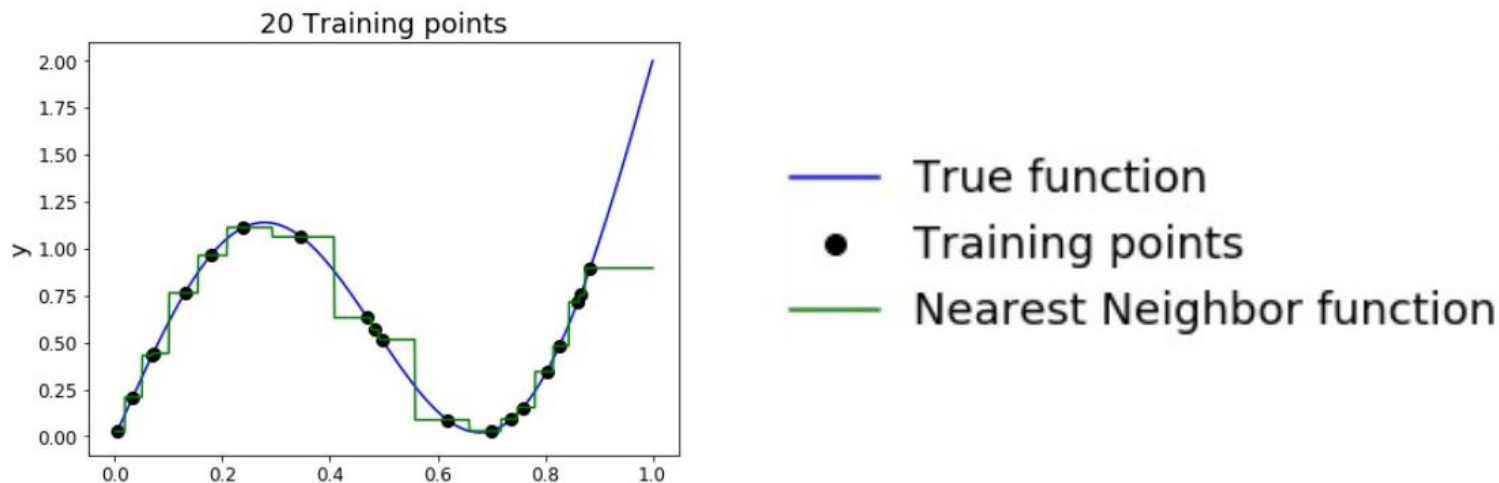
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



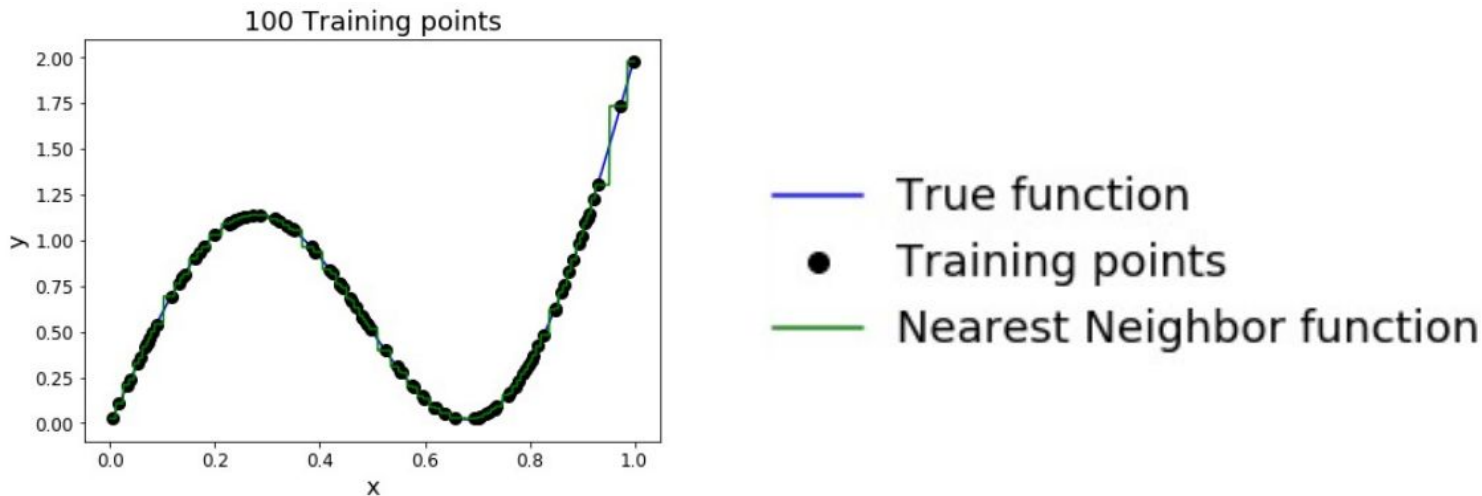
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



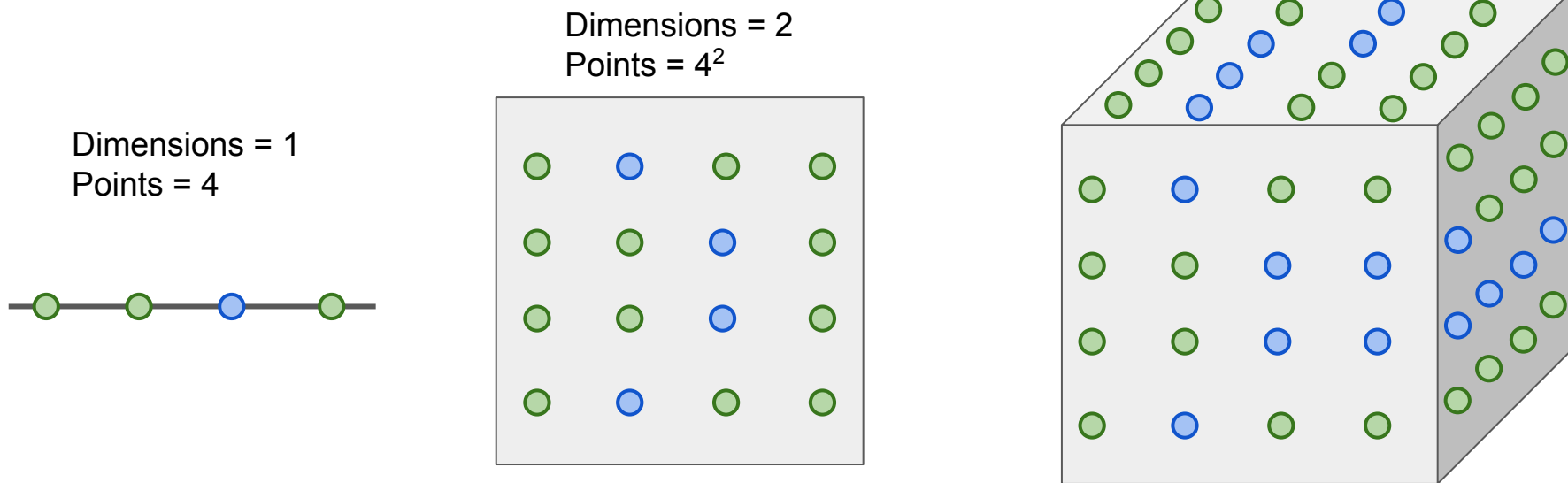
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



Problem: curse of dimensionality

Curse of dimensionality: : For uniform coverage of space, number of training points needed grows exponentially with dimension



Problem: curse of dimensionality

Curse of dimensionality: : For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible 32x32 binary images:

$$2^{32 \times 32} = 10^{308}$$

Number of elementary particles in the visible universe:
 10^{97}

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!

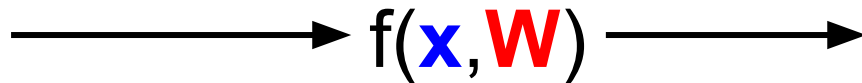
Linear Classifier

Parametric Approach

Image



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



W

parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier

Image



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

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

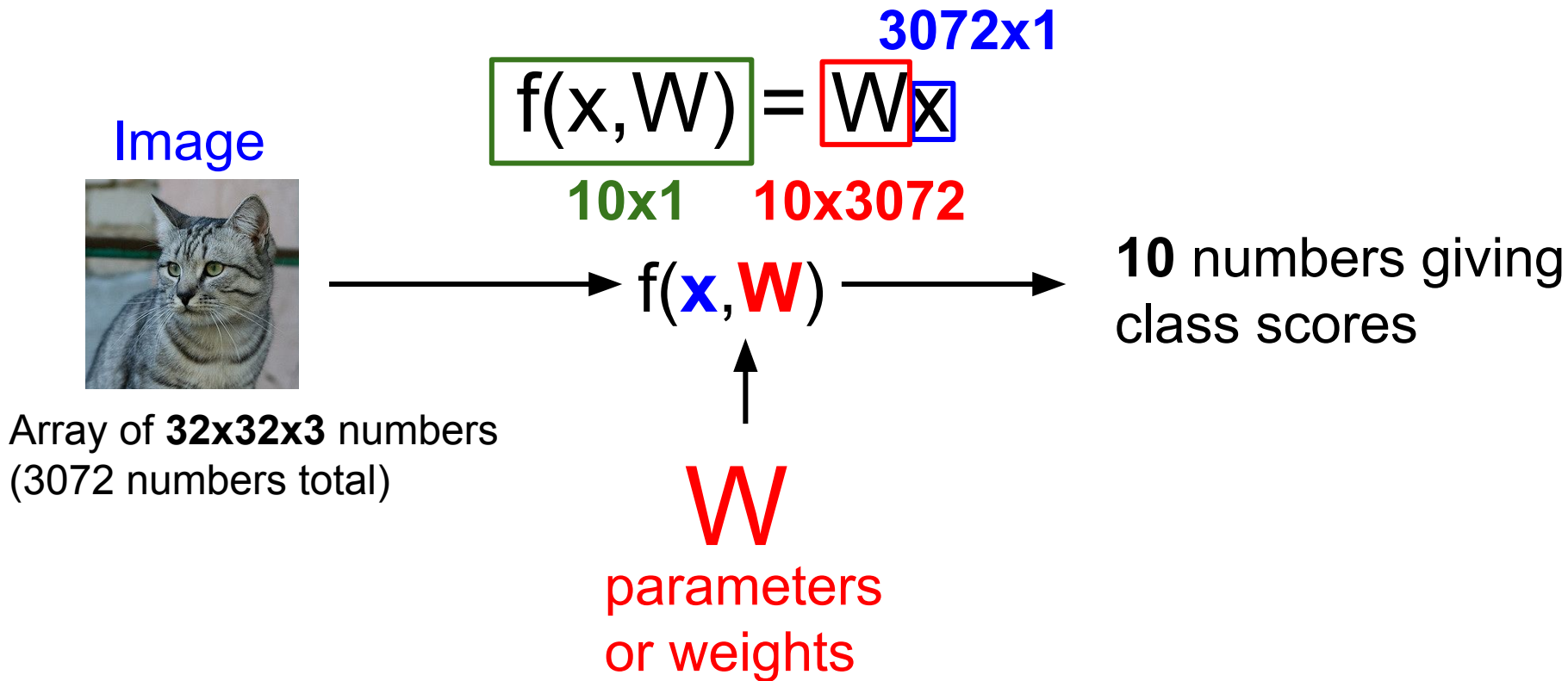
$f(\mathbf{x}, \mathbf{W})$

\mathbf{W}

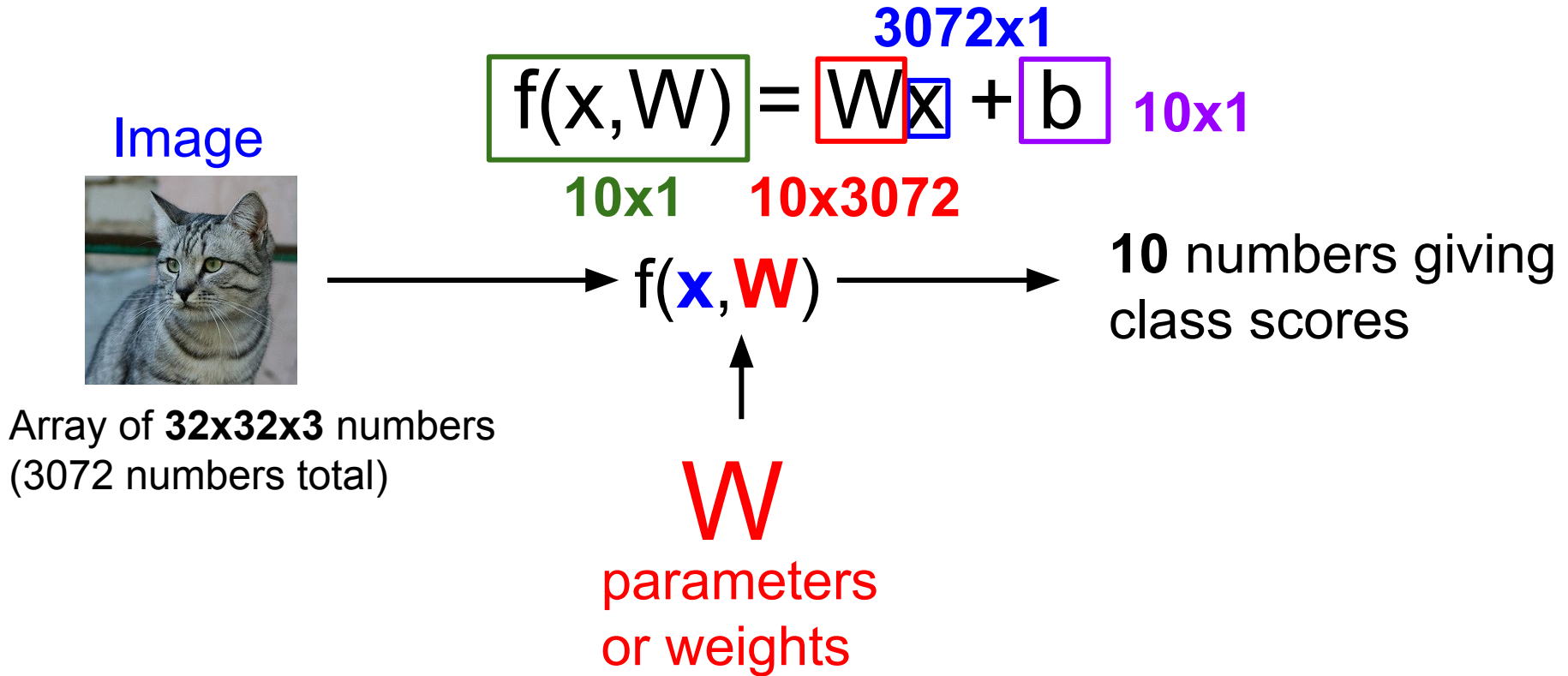
parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier



Parametric Approach: Linear Classifier

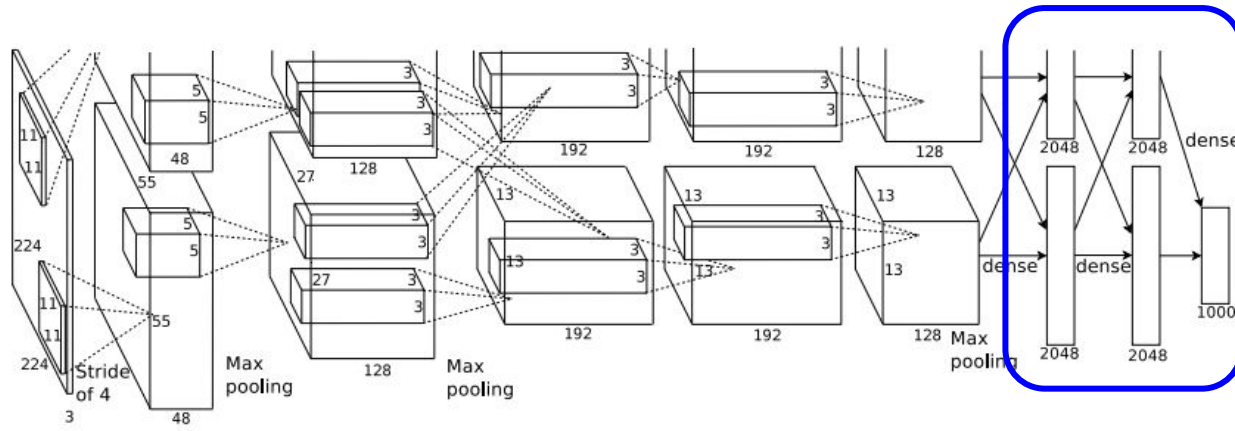


Neural Network

Linear
classifiers

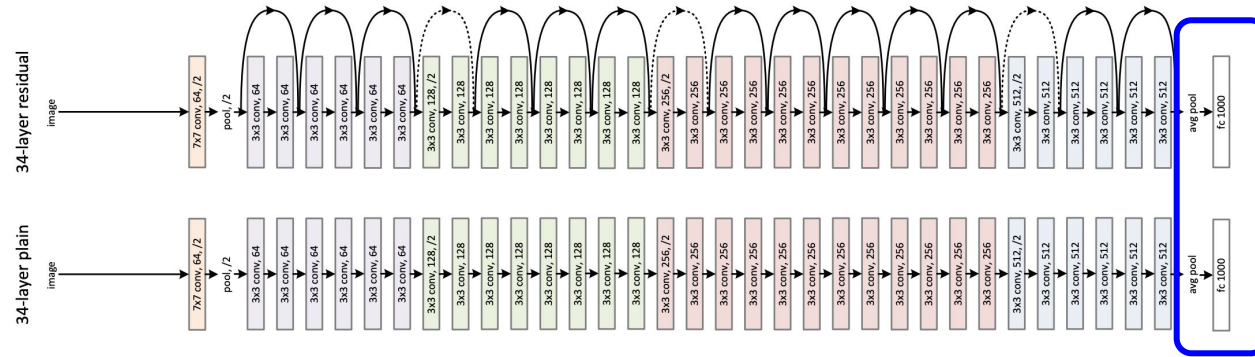


[This image](#) is [CC0 1.0](#) public domain



[Krizhevsky et al. 2012]

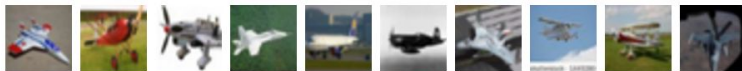
Linear layers



[He et al. 2015]

Recall CIFAR10

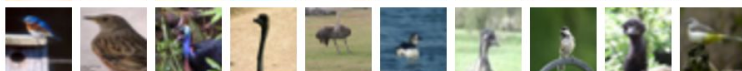
airplane



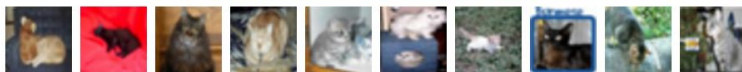
automobile



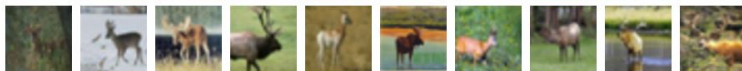
bird



cat



deer



dog



frog



horse



ship



truck



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

10,000 test images.

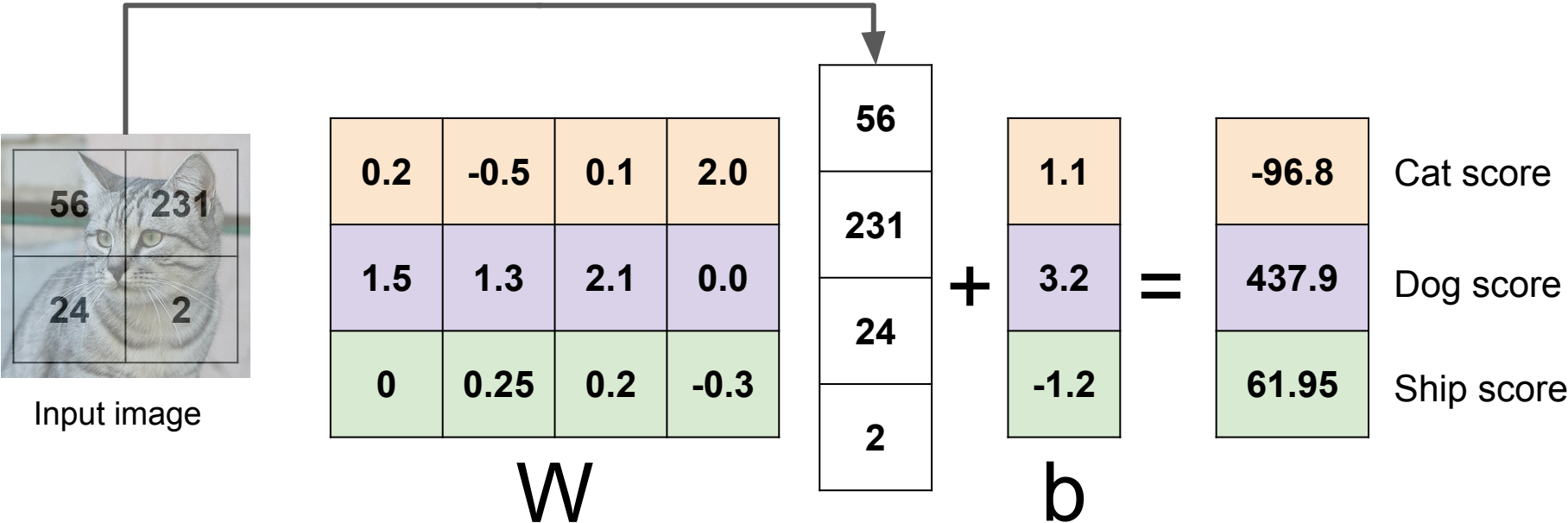
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

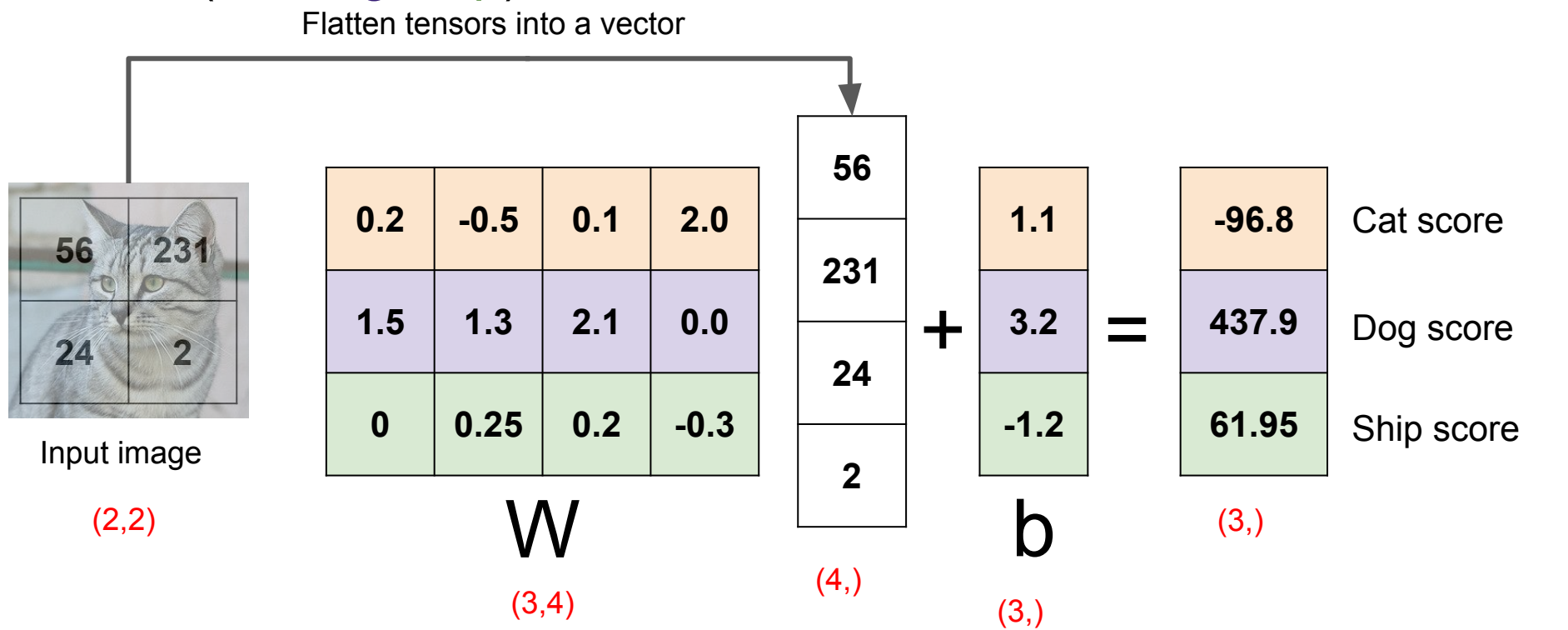


Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

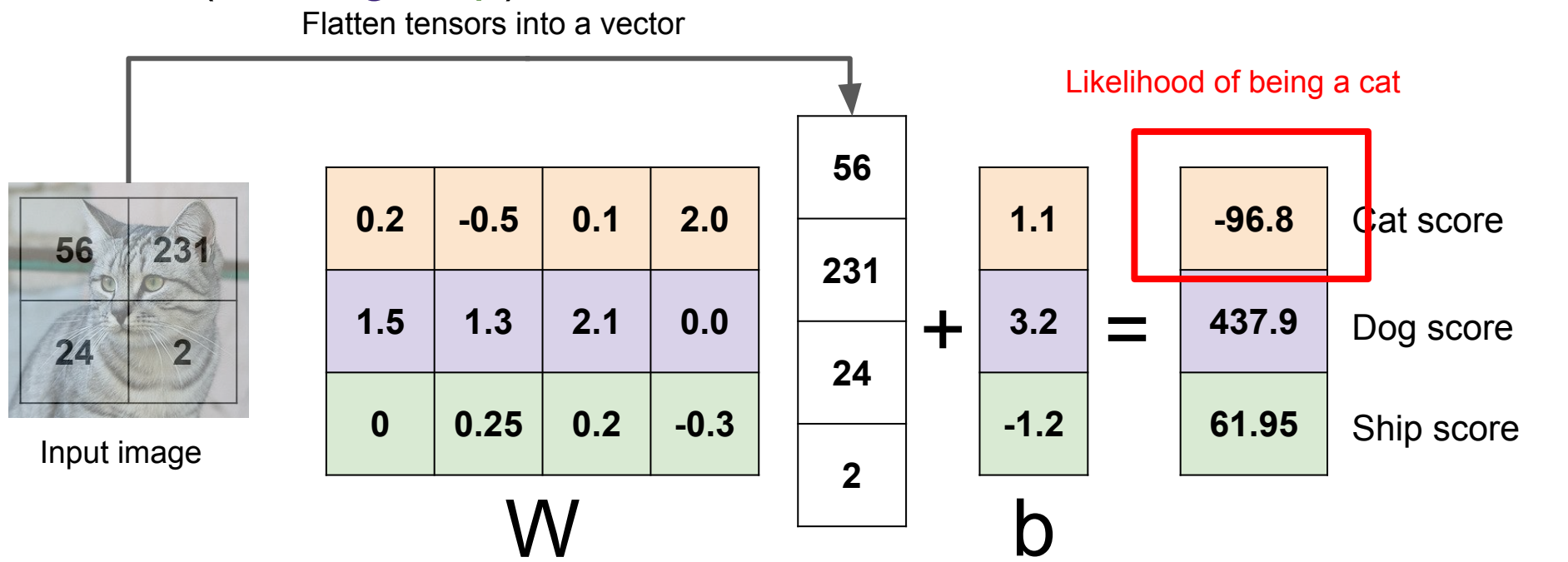
Flatten tensors into a vector



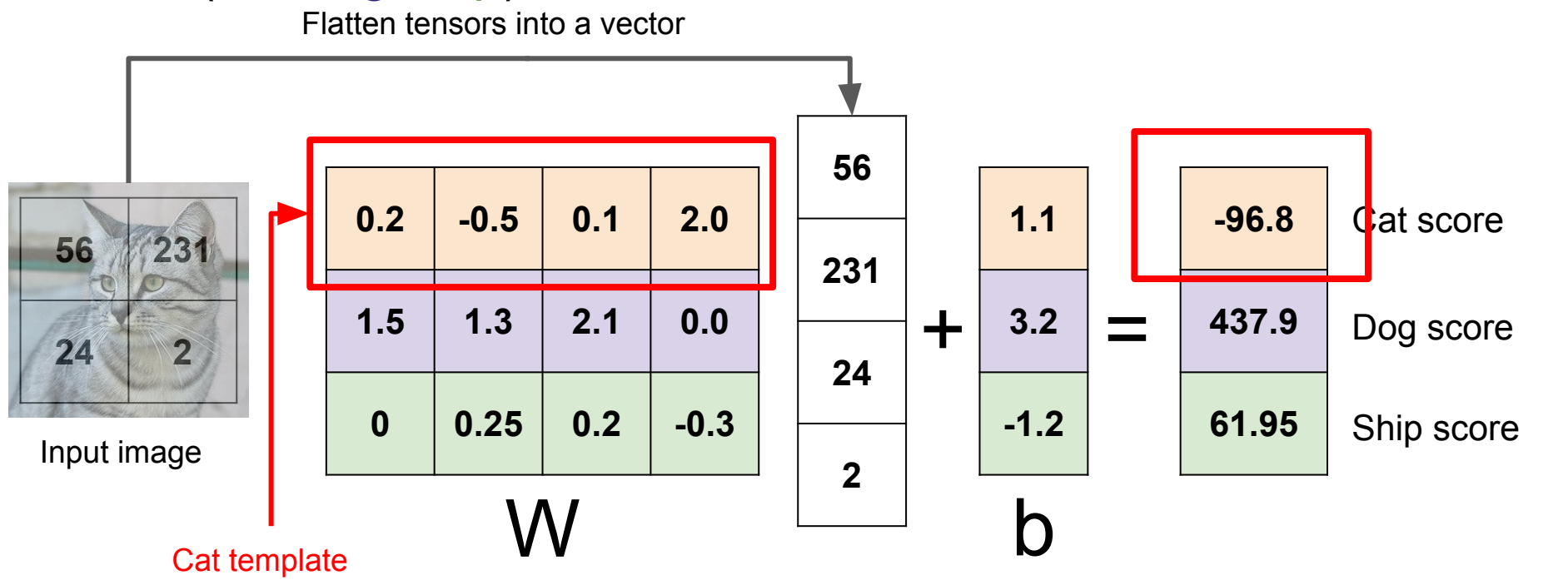
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



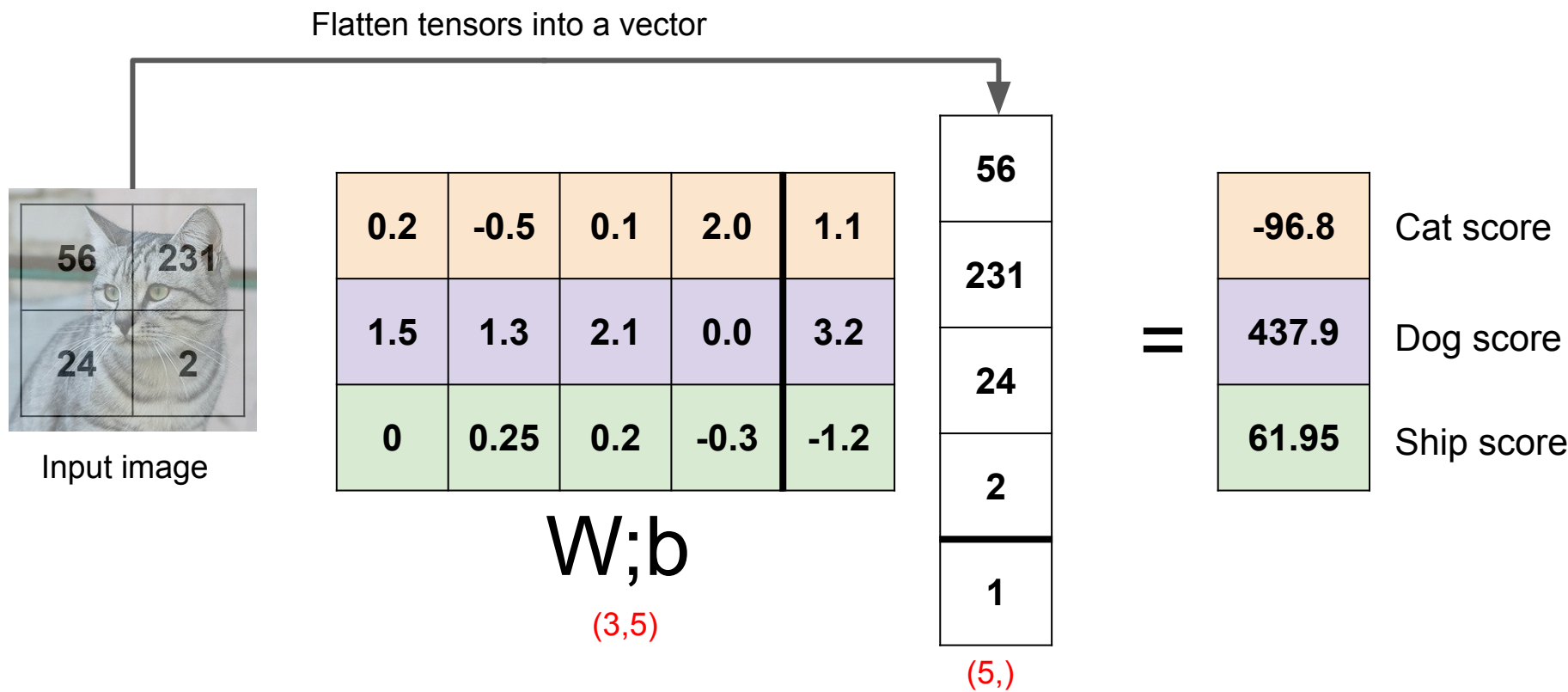
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



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

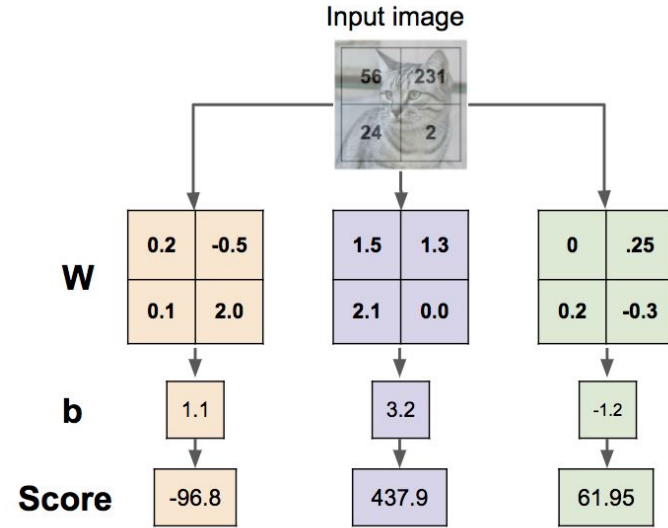
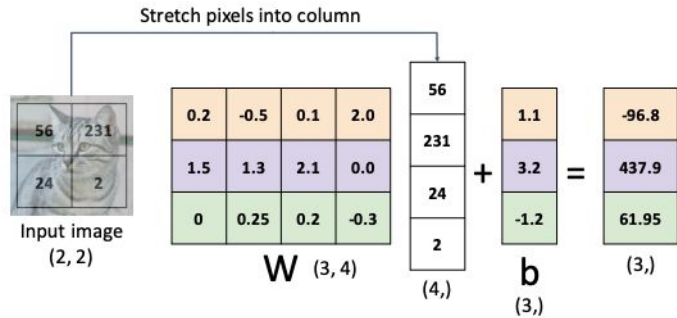


Algebraic viewpoint: Bias trick to simplify computation

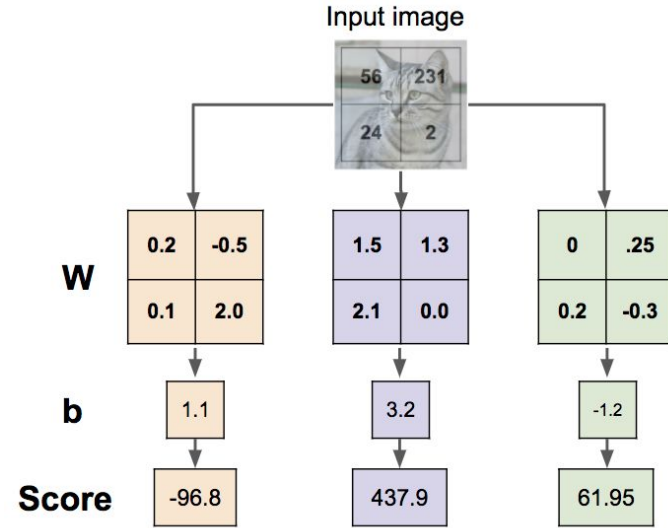
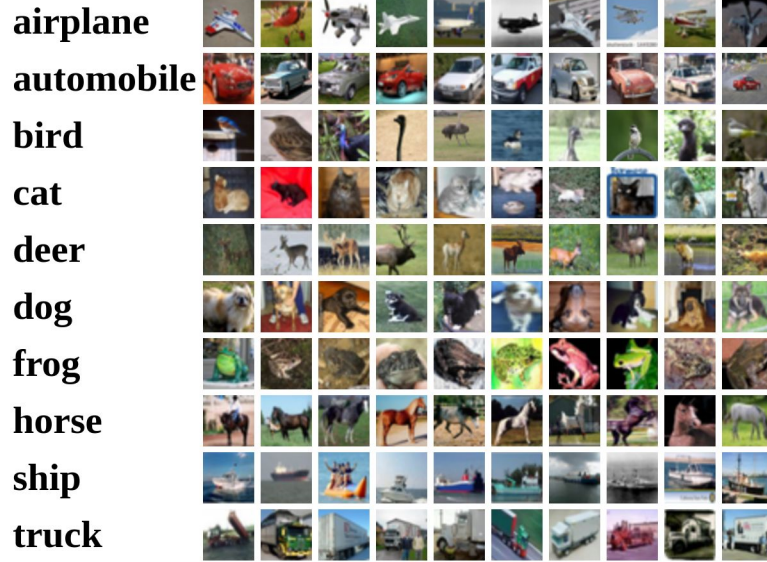


Visual Viewpoint: learning templates

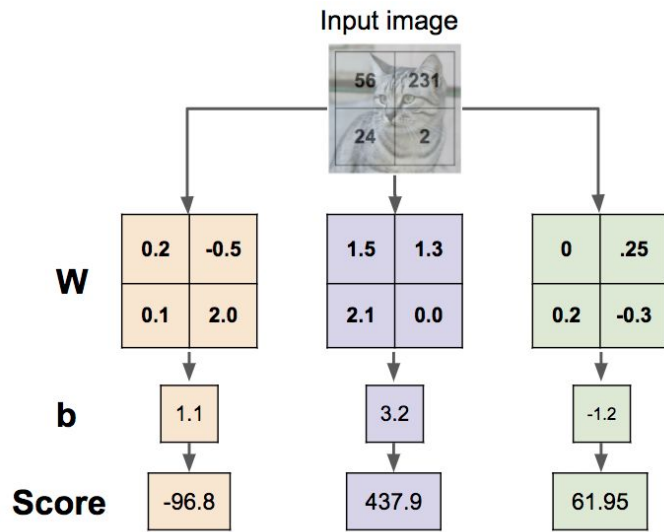
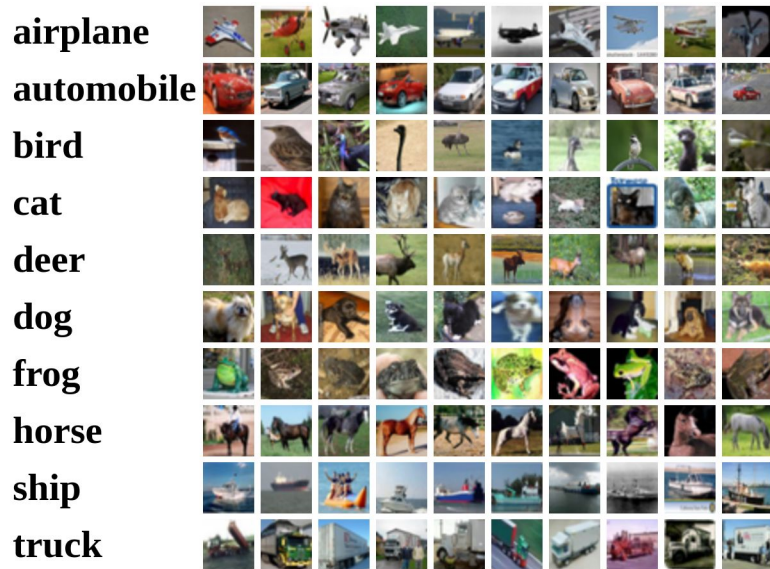
Algebraic viewpoint:



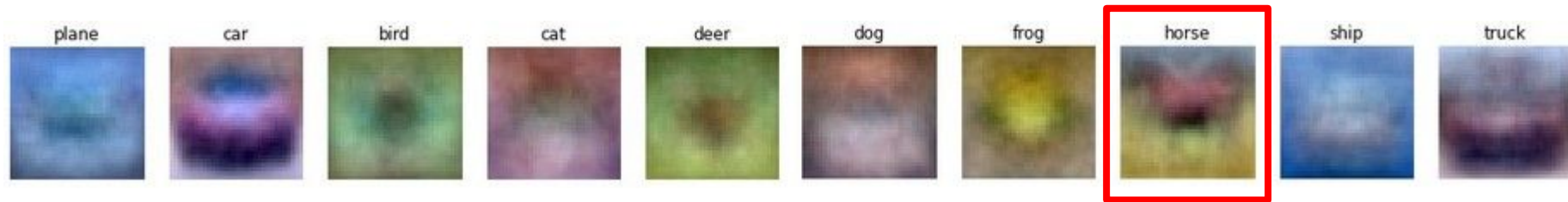
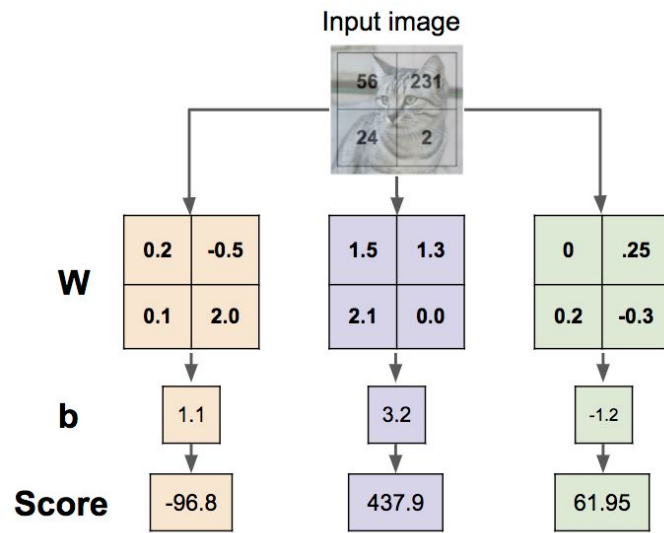
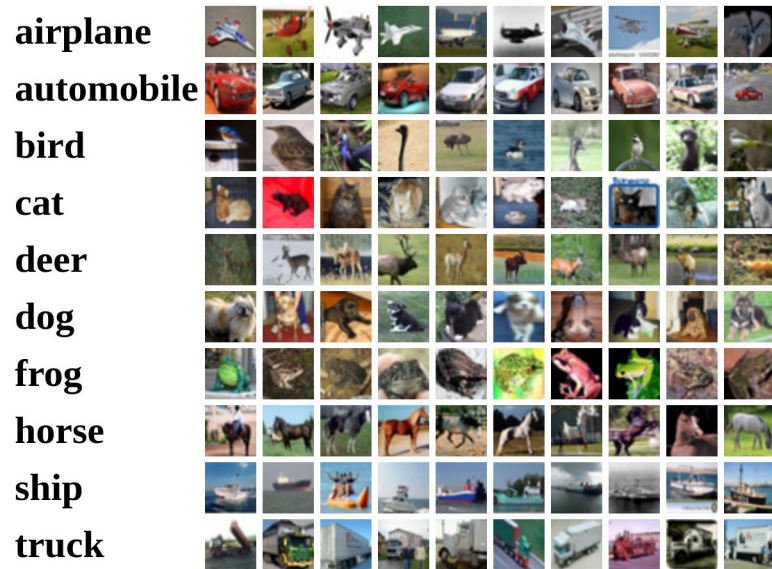
Visual Viewpoint: learning templates



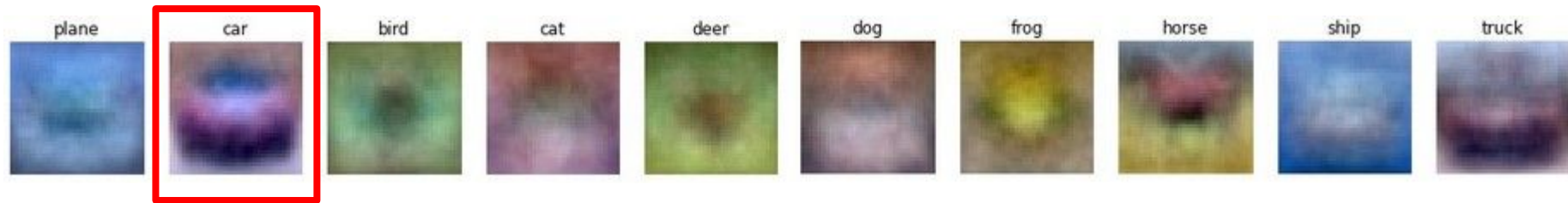
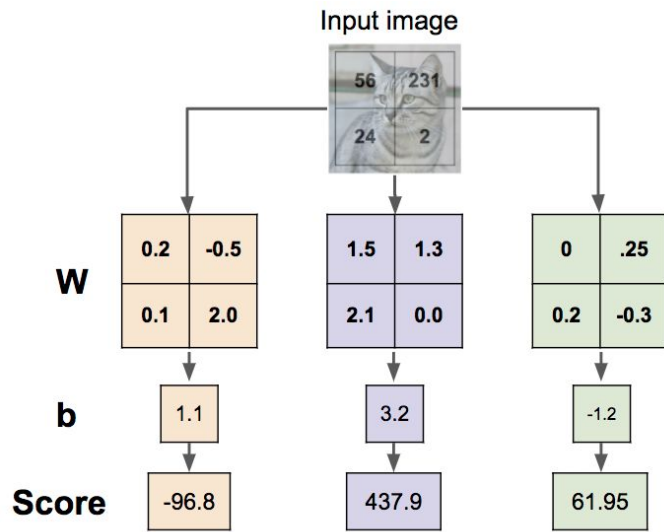
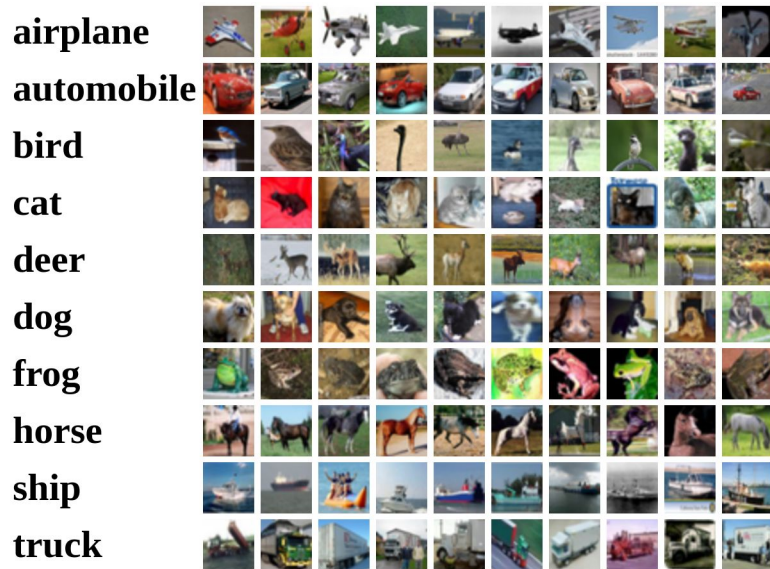
Visual Viewpoint: learning templates



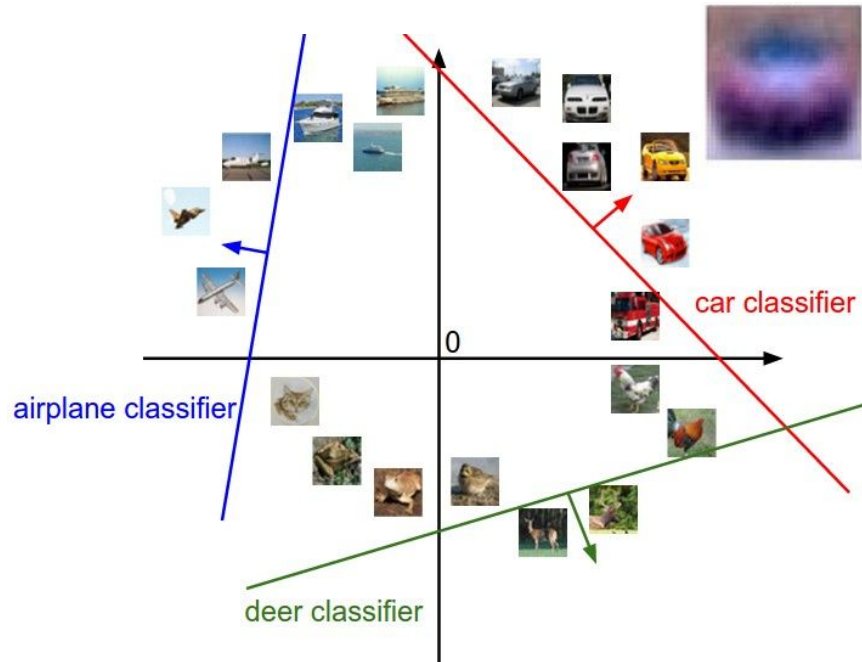
Visual Viewpoint: learning templates



Visual Viewpoint: learning templates



Geometric Viewpoint: linear decision boundaries

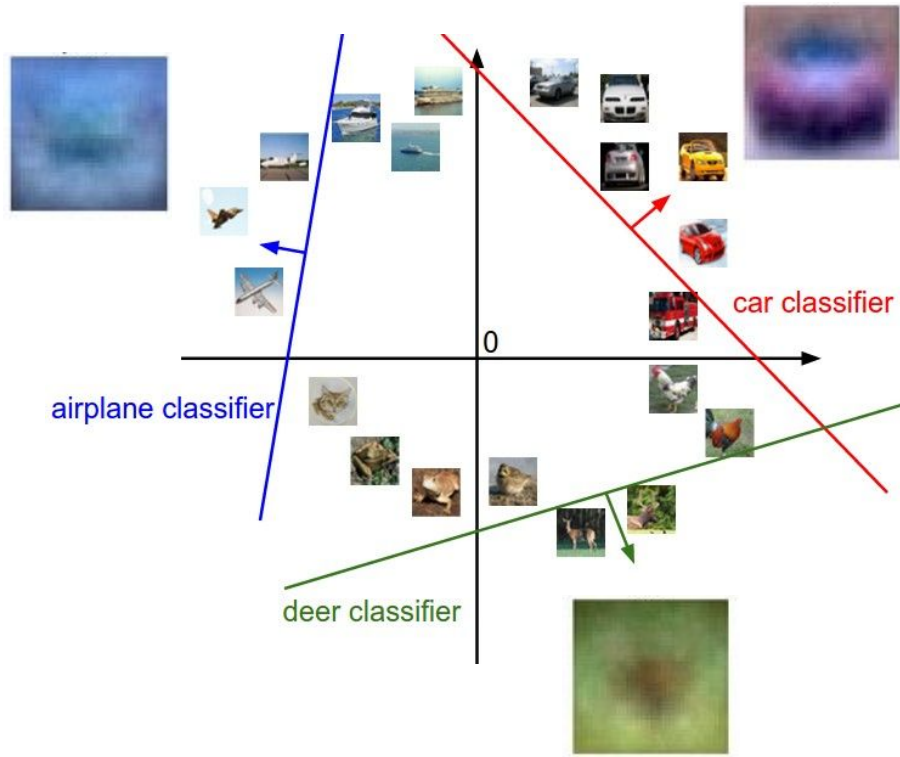


$$f(x, W) = Wx + b$$



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

Geometric Viewpoint: linear decision boundaries

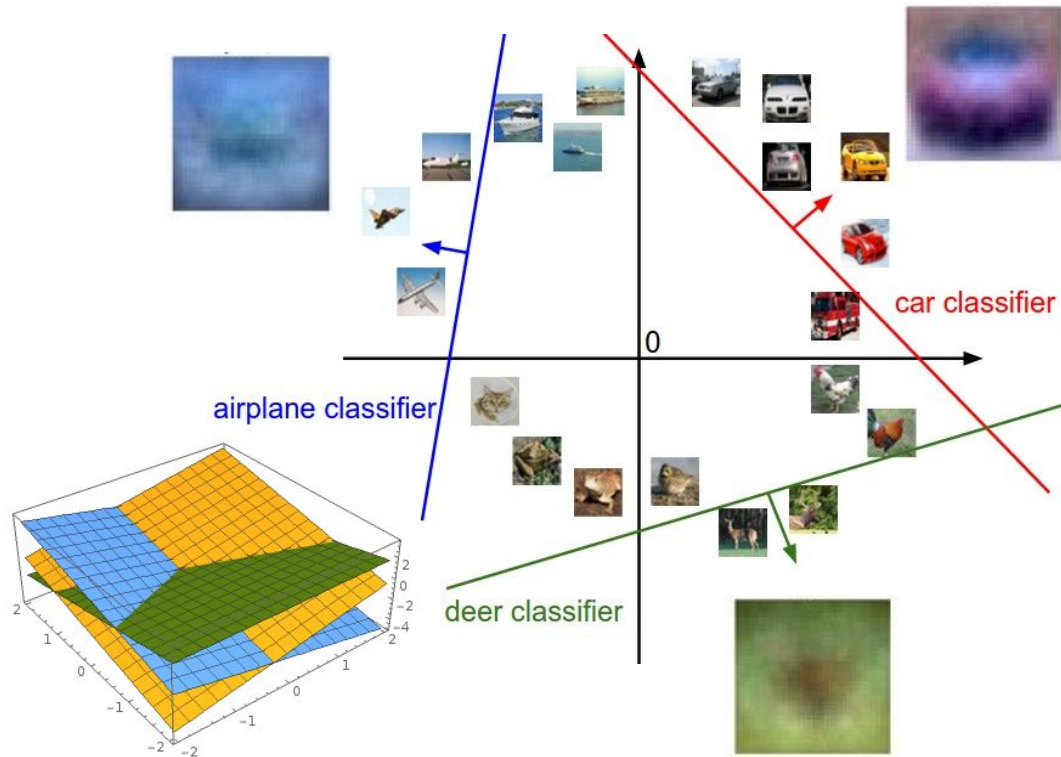


$$f(x, W) = Wx + b$$



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

Geometric Viewpoint: linear decision boundaries



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](#)

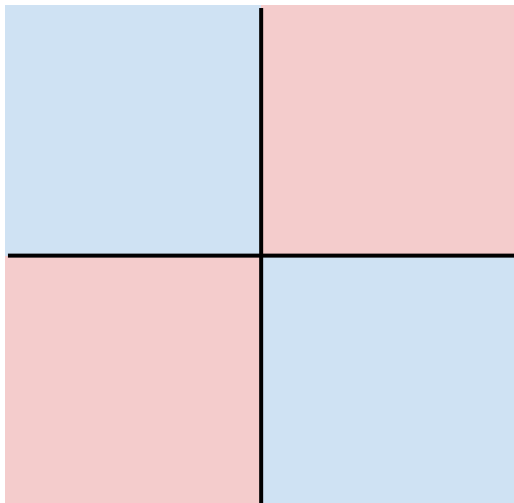
Hard cases for a linear classifier

Class 1:

First and third quadrants

Class 2:

Second and fourth quadrants

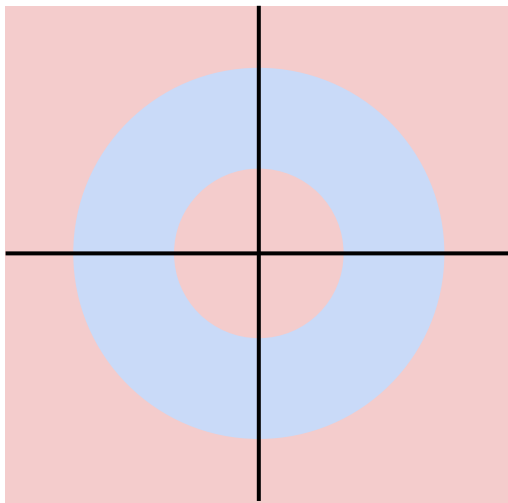


Class 1:

$1 \leq \text{L2 norm} \leq 2$

Class 2:

Everything else

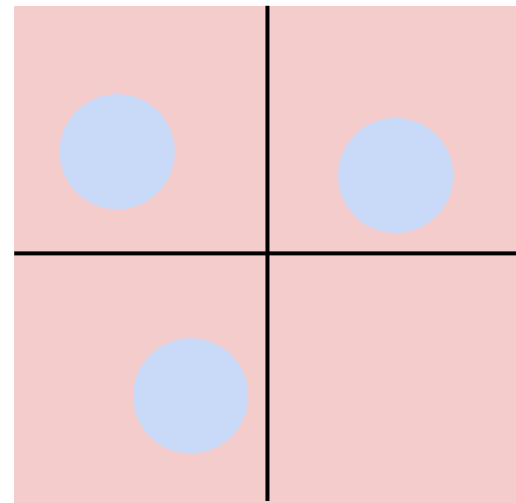


Class 1:

Three modes

Class 2:

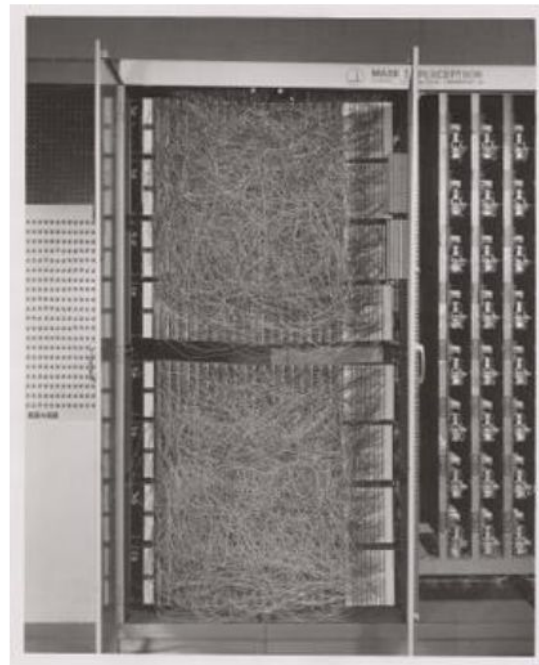
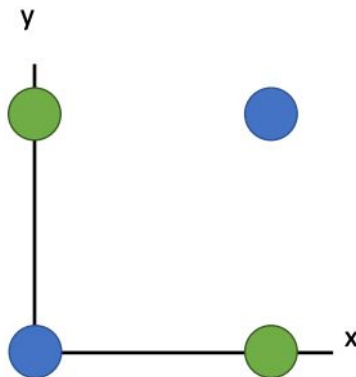
Everything else



Recall the Minsky report 1969 from last lecture

Unable to learn the XNOR function

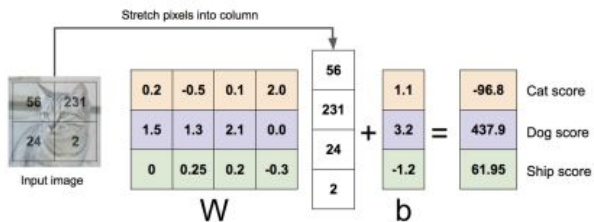
X	Y	$F(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0



Three viewpoints for interpreting linear classifiers

Algebraic Viewpoint

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



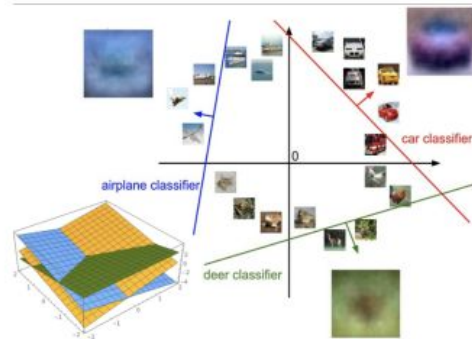
Visual Viewpoint

One template
per class



Geometric Viewpoint

Hyperplanes
cutting up space



Coming up:

- Loss function
- Optimization
- Neural Networks

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

(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)