


FACULTY OF MATHEMATICS,
PHYSICS AND INFORMATICS
Comenius University
Bratislava

Neural Networks for Computer Vision

Lecture 2: Image Classification

Ing. Viktor Kocur, PhD., RNDr. Zuzana Černeková, PhD.

20.9.2022

Acknowledgment

The majority of slides are directly adopted from slides for CS231n[1] course at Stanford University!

[1] *Stanford CS231n lecture slides.* http://cs231n.stanford.edu/slides/

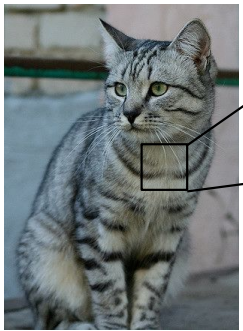
Contents

Image Classification

Image Classification

$\Rightarrow$ Cat

The Problem: Semantic Gap



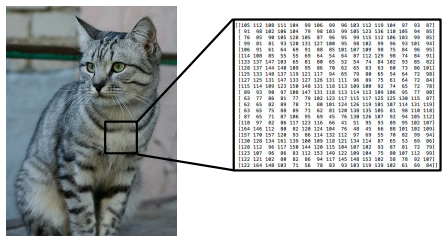

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 120 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 100  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]
 [125 133 140 137 119 121 117  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  98  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]
 [110  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

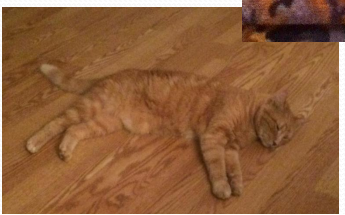


All pixels change when the camera moves!

Challenges - background





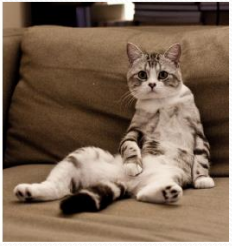


Challenges - illumination

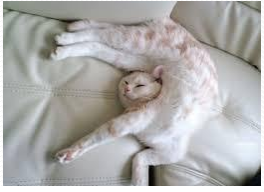



Challenges - deformation



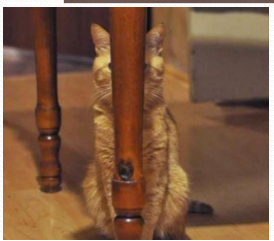






Challenges - occlusion





Challenges - intraclass variation

Image Classifier

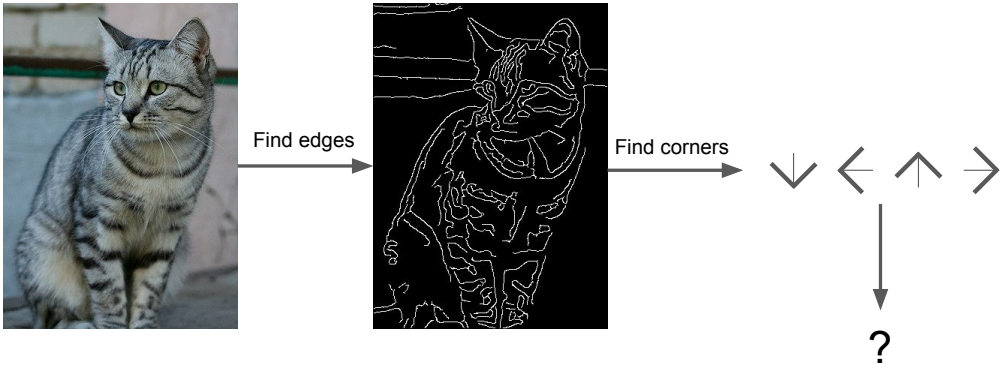
```
def classify_image(image):
    # some magic
    return class_label
```

Image Classifier

```
def classify_image(image):
    # some magic
    return class_label
```

There is no obvious way how to do this!

Classical CV



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

Machine Learning

Machine learning algorithms are algorithms which can improve their performance from experience. In case of CV experience usually means large annotated datasets.

Typical ML workflow:

1. Collect a dataset of images and corresponding labels
2. Train a classifier using ML
3. Evaluate the classifier on a new set of images

Nearest Neighbors classifier

One of the simplest classifiers is the nearest neighbor classifier:

- We train the classifier by simply remembering all of the training images and labels.
- To classify a new image we assign it the label of the image from the training set which is closest the new image.

Distance metric

Training data with labels

?

query data

Distance Metric $\left| \right.$  ,  $\left| \right. \rightarrow \mathbb{R}$

Simple metric

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

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

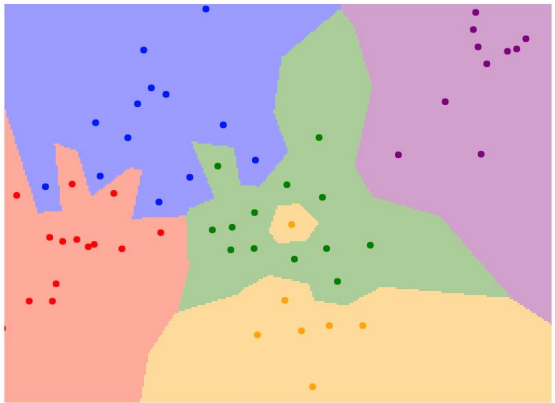
=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

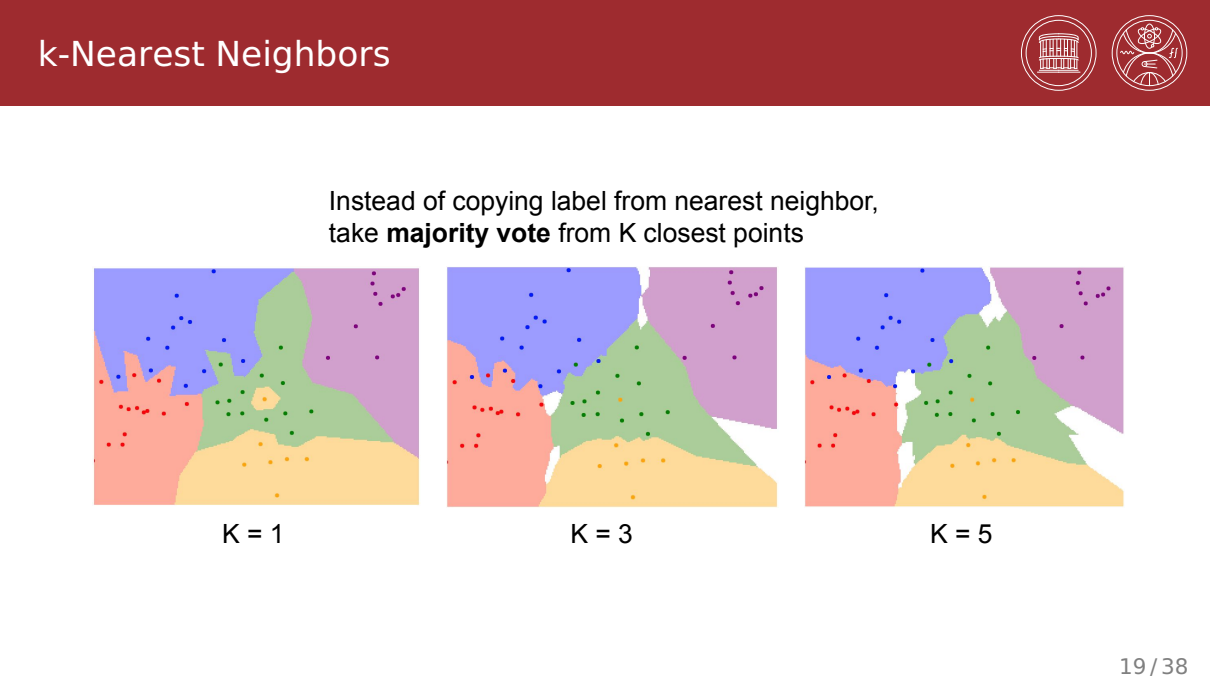
add → 456

Decision boundary



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

Hyperparameters

Before using kNN we have to make few decisions:

- Which k do we use? E.g. number of neighbors.
- What kind of metric we use to calculate the distances?

Hyperparameters

Before using kNN we have to make few decisions:

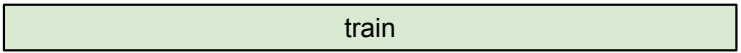
- Which k do we use? E.g. number of neighbors.
- What kind of metric we use to calculate the distances?

Both of these are **hyperparameters**. In general, hyperparameters change the algorithms themselves. We use the prefix hyper, since more sophisticated usually have parameters which are learned during training.

Setting these is difficult and dependent on the given problem! Usually you have to try and see what works best!

Setting Hyperparameters

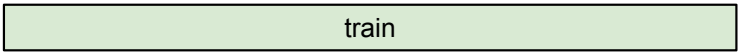
Idea #1: Choose hyperparameters
that work best on the **training data**



Setting Hyperparameters

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

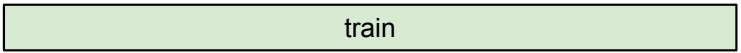
BAD: K = 1 always works perfectly on training data



Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

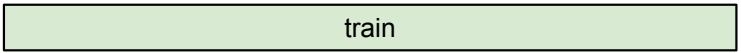


Idea #2: choose hyperparameters that work best on **test** data

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

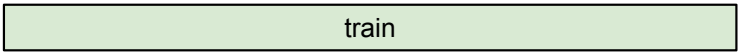
BAD: No idea how algorithm will perform on new data

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



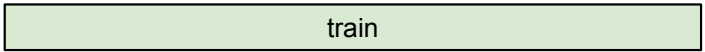
Idea #2: choose hyperparameters that work best on **test** data

BAD: No idea how algorithm will perform on new data

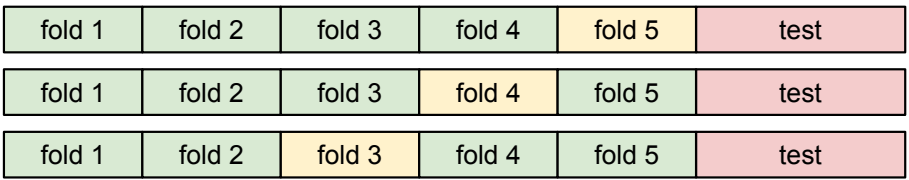
Idea #3: Split data into **train**, **val**; choose hyperparameters on val and evaluate on test

Better!

Setting Hyperparameters



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



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

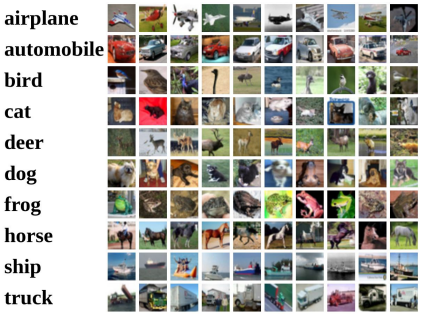
Example Dataset - CIFAR10

10 classes
50,000 training images
10,000 testing images

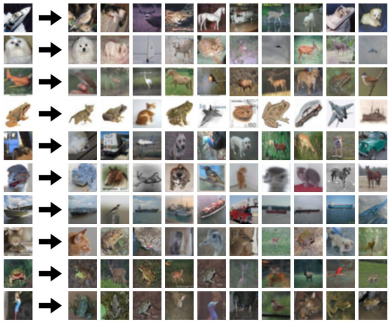
Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Example Dataset - CIFAR10

10 classes
50,000 training images
10,000 testing images



Test images and nearest neighbors

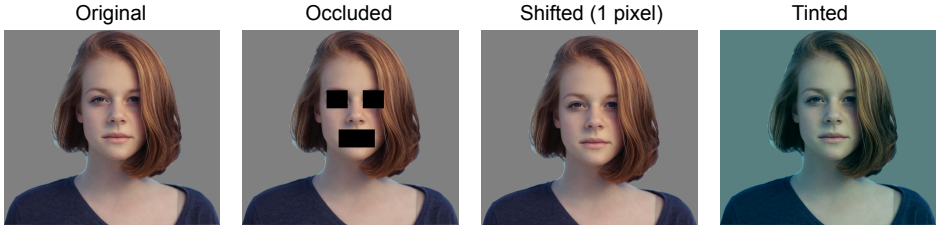


Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

kNN - not a good choice

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

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



Original image is CC0 public domain

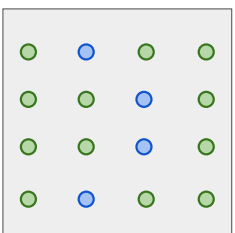
kNN - not a good choice

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

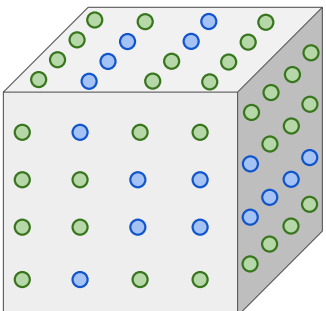
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



Linear Classifier

Image

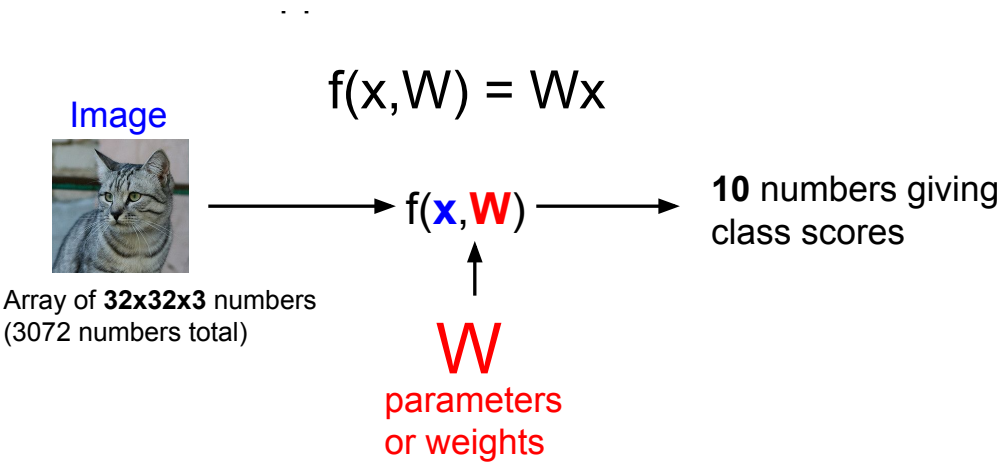
Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

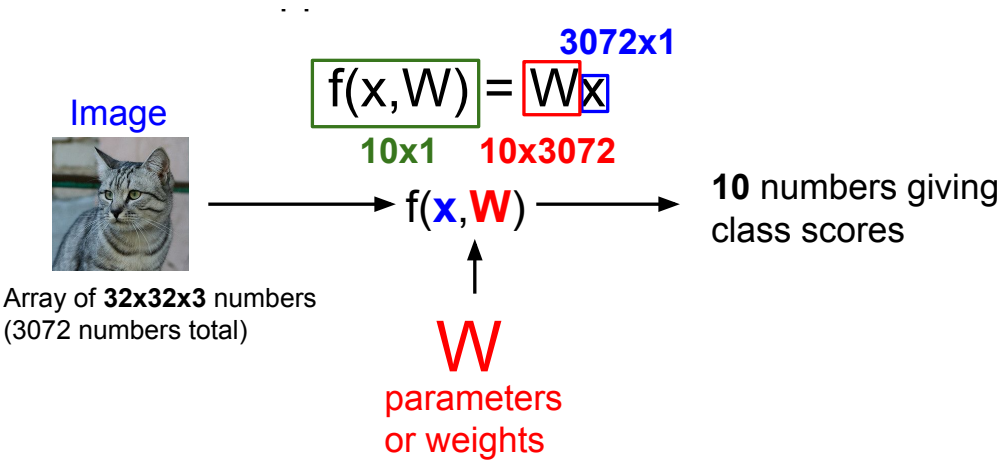
10 numbers giving
class scores

W
parameters
or weights

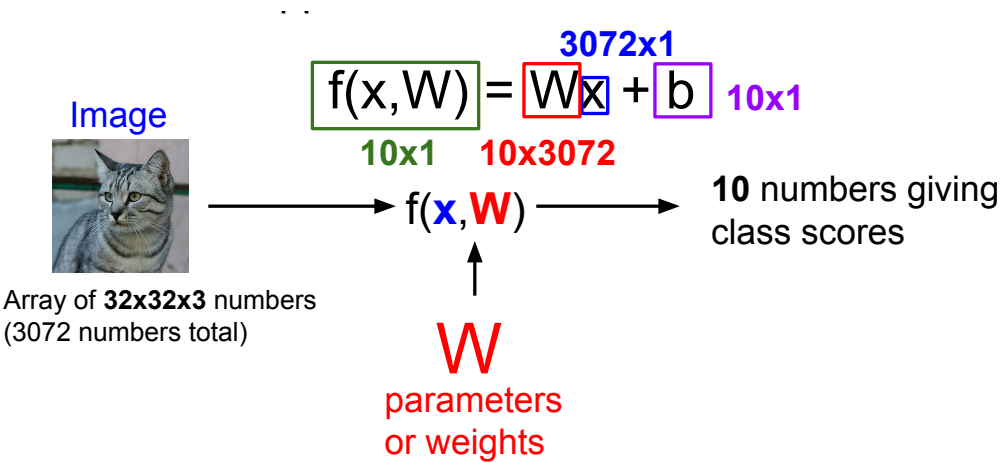
Linear Classifier



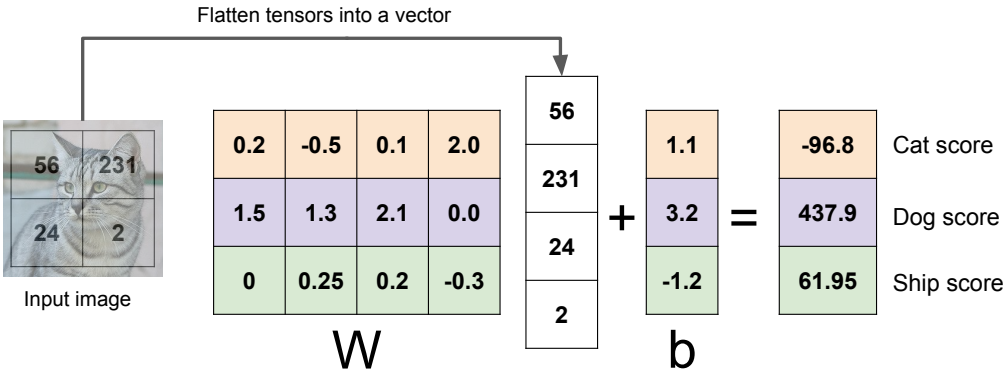
Linear Classifier



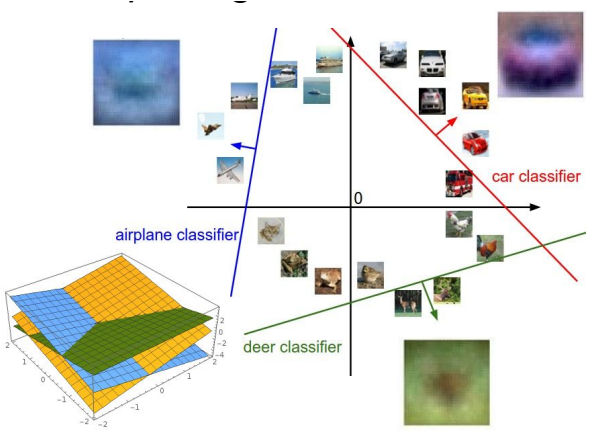
Linear Classifier



Linear Classifier - Example



Linear Classifier - Decision Boundary



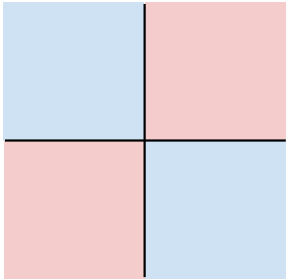
$$f(x,W) = Wx + b$$

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

Linear Classifier - Hard cases

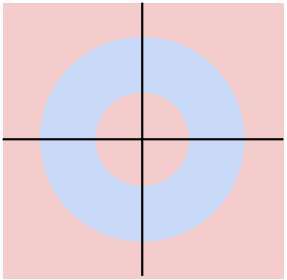
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



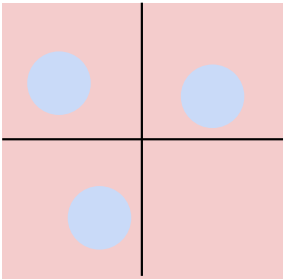
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



Next week - optimization

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

- Loss function
  - Quantifying *good* parameters
- Optimization
  - Finding optimal parameters to minimize the loss function
- Convolutional Neural Networks