

## Lecture 11:

# Convolution and Image Classification & Segmentation



Markus Hohle

University California, Berkeley

Bayesian Data Analysis and  
Machine Learning for Physical  
Sciences



## Course Map

Module 1	Maximum Entropy and Information, Bayes Theorem
Module 2	Naive Bayes, Bayesian Parameter Estimation, MAP
Module 3	MLE, Lin Regression
Module 4	Model selection I: Comparing Distributions
Module 5	Model Selection II: Bayesian Signal Detection
Module 6	Variational Bayes, Expectation Maximization
Module 7	Hidden Markov Models, Stochastic Processes
Module 8	Monte Carlo Methods
Module 9	Machine Learning Overview, Supervised Methods & Unsupervised Methods
Module 10	ANN: Perceptron, Backpropagation, SGD
<b>Module 11</b>	<b>Convolution and Image Classification and Segmentation</b>
Module 12	RNNs and LSTMs
Module 13	RNNs and LSTMs + CNNs
Module 14	Transformer and LLMs
Module 15	Graphs & GNNs



## Outline

**The Problem**

**Convolution**

**CNN Architectures**

**Data Preparation & Training**

**Example**

- LeNet numpy only
- LeNet TensorFlow
- sequences as images
- segmentation





```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

**The Problem**

**part I**

**Convolution**

**CNN Architectures**

**Data Preparation & Training**

**Example**

**part II**

- LeNet numpy only
- LeNet TensorFlow
- sequences as images
- segmentation



```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

### **The Problem**

### Convolution

### CNN Architectures

### Data Preparation & Training

### Example

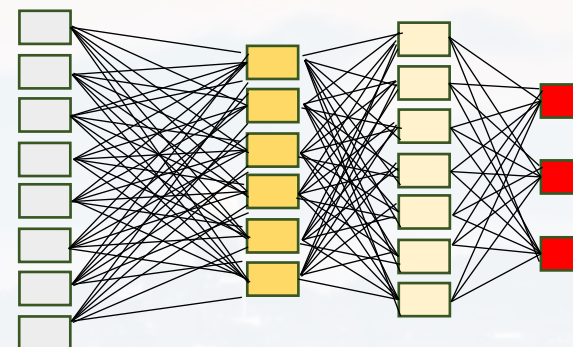
- LeNet numpy only
- LeNet TensorFlow
- sequences as images
- segmentation





so far:

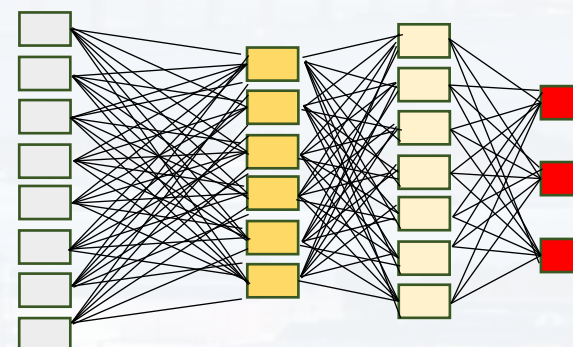
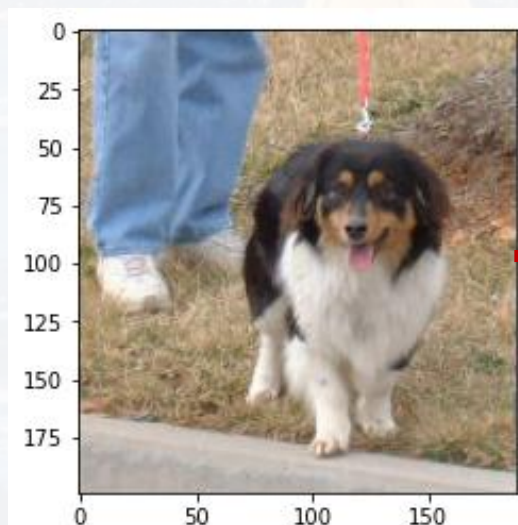
data features					
data points ↓	[	5.1	3.5	1.4	0.2]
	[	4.9	3.	1.4	0.2]
	[	4.7	3.2	1.3	0.2]
	[	4.6	3.1	1.5	0.2]
	[	5.	3.6	1.4	0.2]
	[	5.4	3.9	1.7	0.4]
	[	4.6	3.4	1.4	0.3]
	[	5.	3.4	1.5	0.2]



data points are **not related**,  
no spatial information

row = data point

now:



data points are **related**,  
**lots** of spatial information



so far:

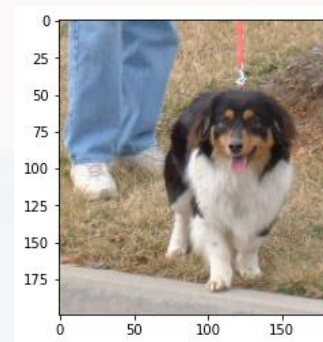
**data features**

data points ↓

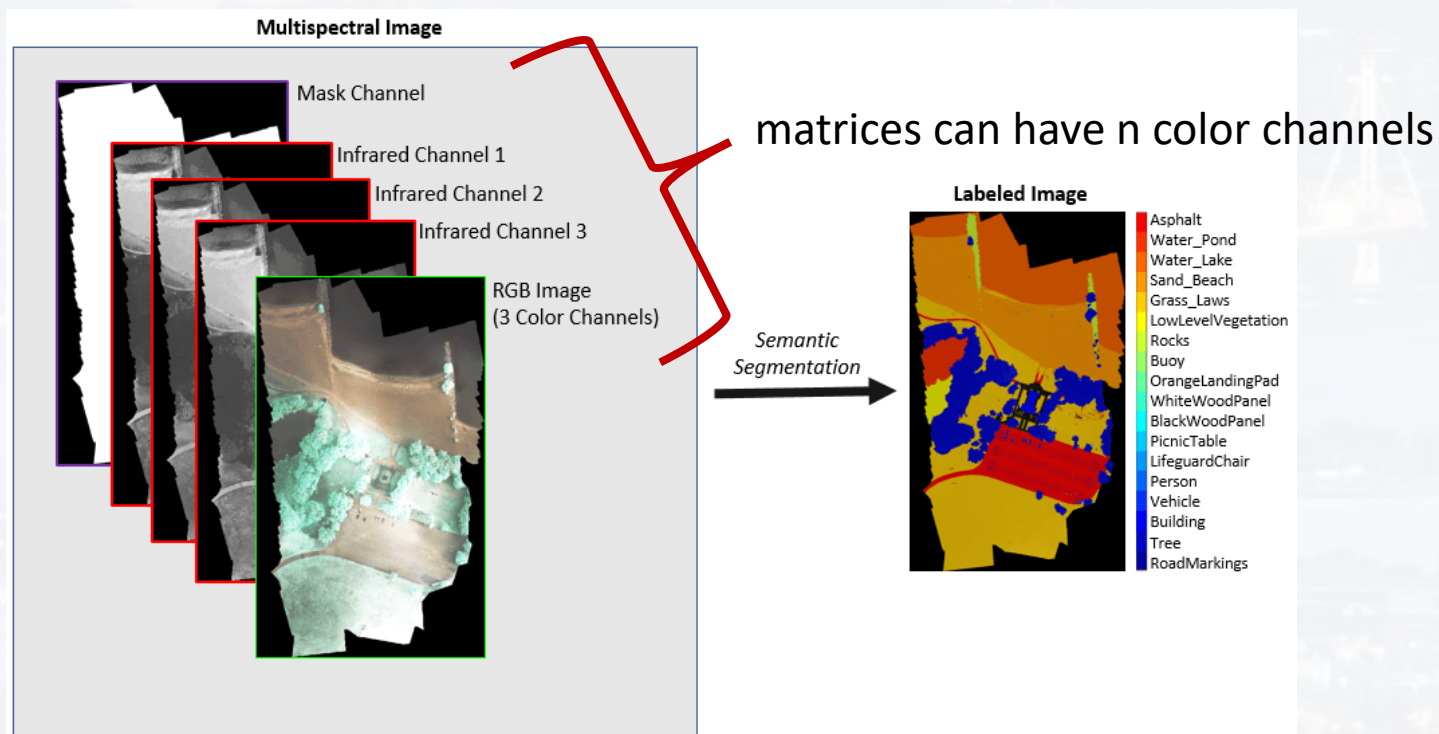
[	5.1	3.5	1.4	0.2]
[	4.9	3.	1.4	0.2]
[	4.7	3.2	1.3	0.2]
[	4.6	3.1	1.5	0.2]
[	5.	3.6	1.4	0.2]
[	5.4	3.9	1.7	0.4]
[	4.6	3.4	1.4	0.3]
[	5.	3.4	1.5	0.2]

row = data point

now:



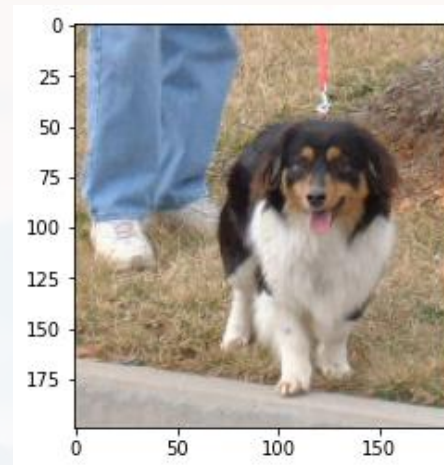
matrix = data point



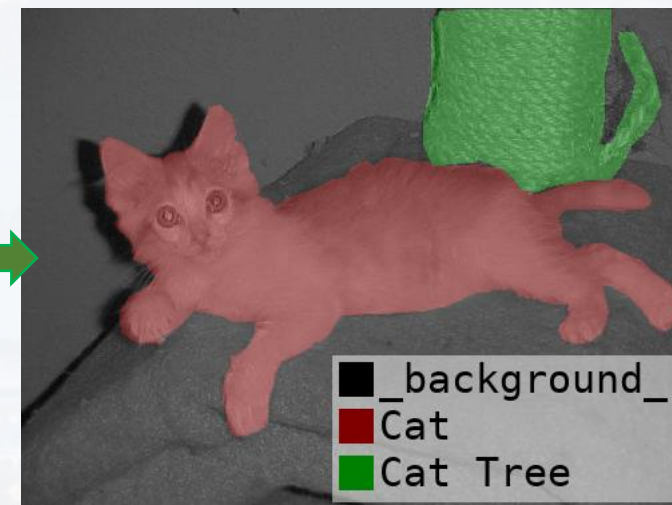
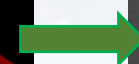
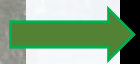




1) **classification:** between different images



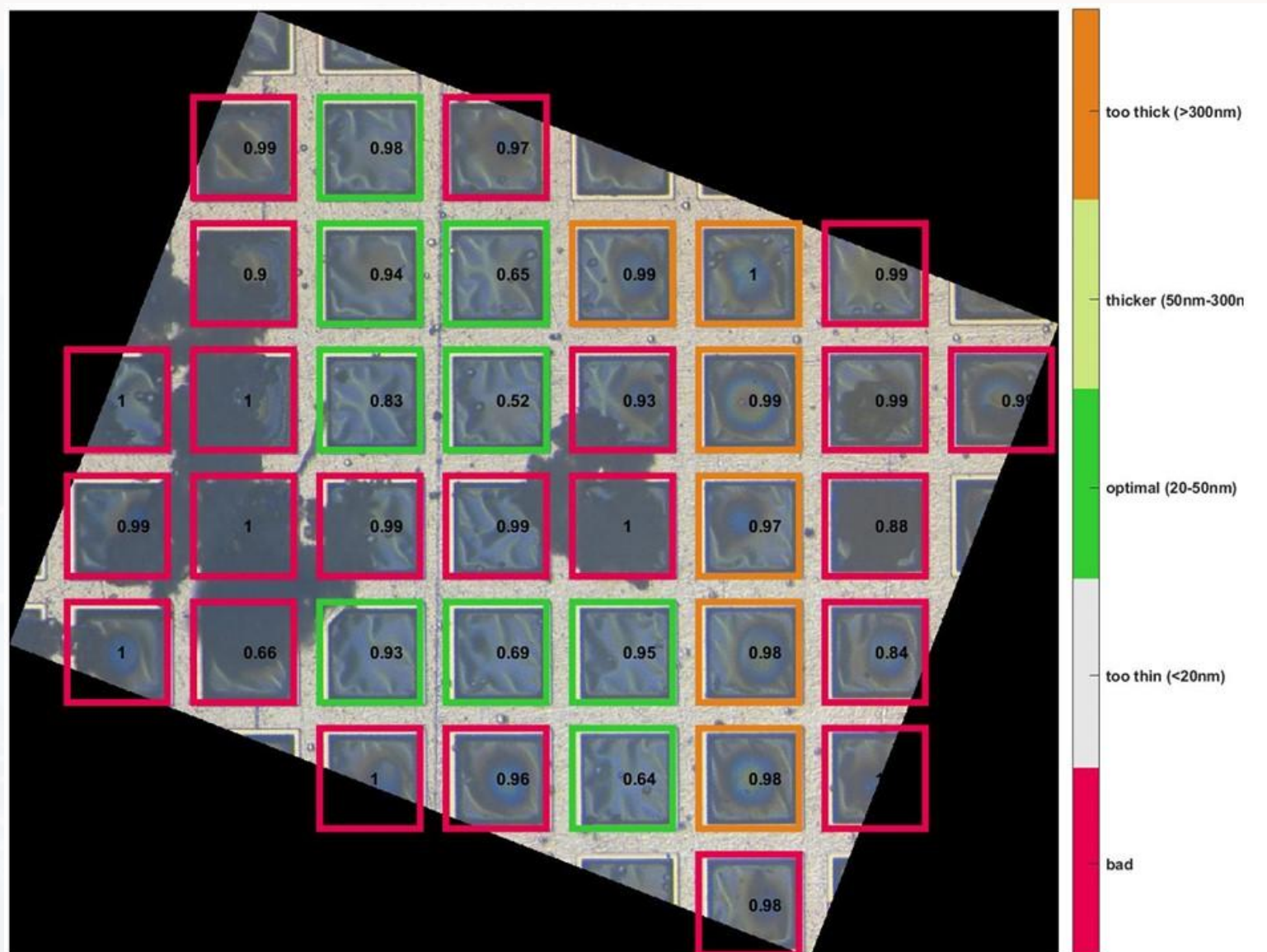
different pixel within images aka 2) **segmentation**







segmentation *for* classification



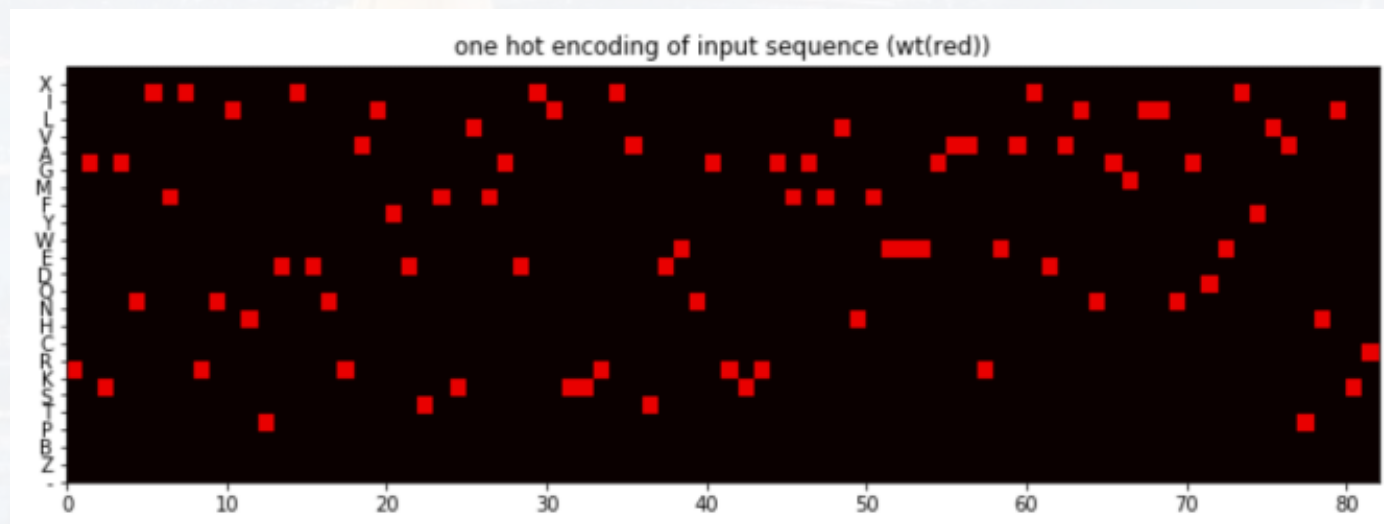
see [here](#)



### motif finding / sequence analysis

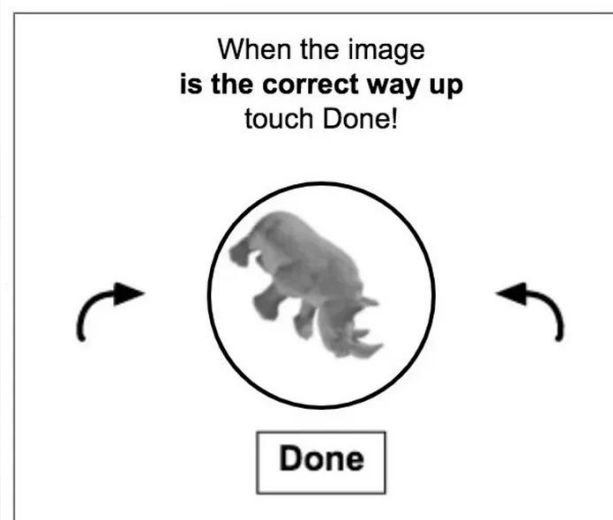
	C	A	G	T	C	T	A
4	1	0	0	0	1	0	0
	0	0	1	0	0	0	0
	0	0	0	1	0	1	0
	0	1	0	0	0	0	1
	Sequence Length						

one – hot encoded NT or AA sequences  
can be interpreted as b/w images!



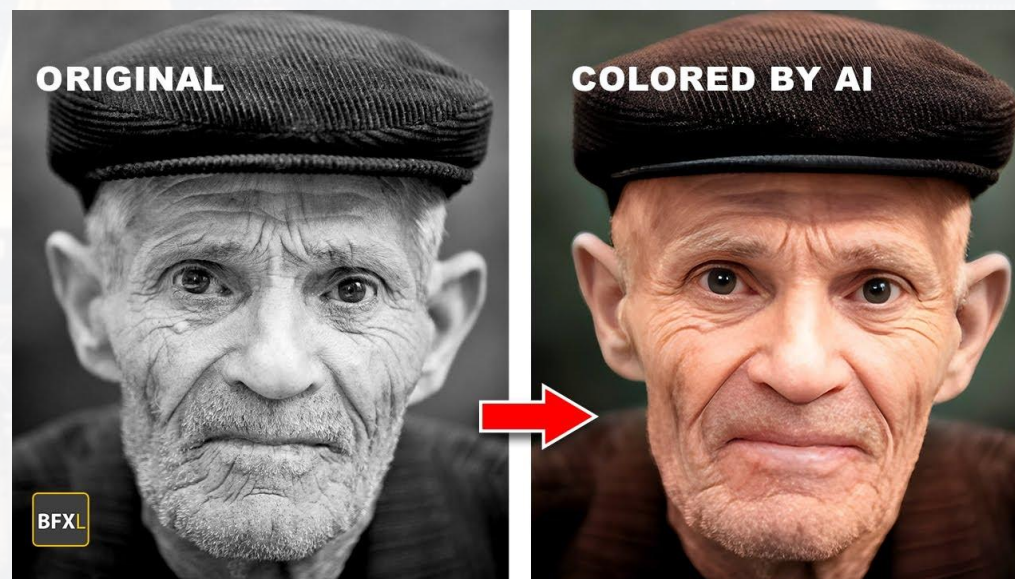


### 3) regression:



turning images the right way

### 4) generation



source: TopviewAI





before we had CNNs: k-means for segmentation & classification:

```
rows, cols, channels = Image.shape  
x_flat               = Image.reshape(rows * cols, channels)
```

```
kmeans               = KMeans(n_clusters = k)  
pxl_labels           = kmeans.fit_predict(x_flat)  
colors               = kmeans.cluster_centers_
```

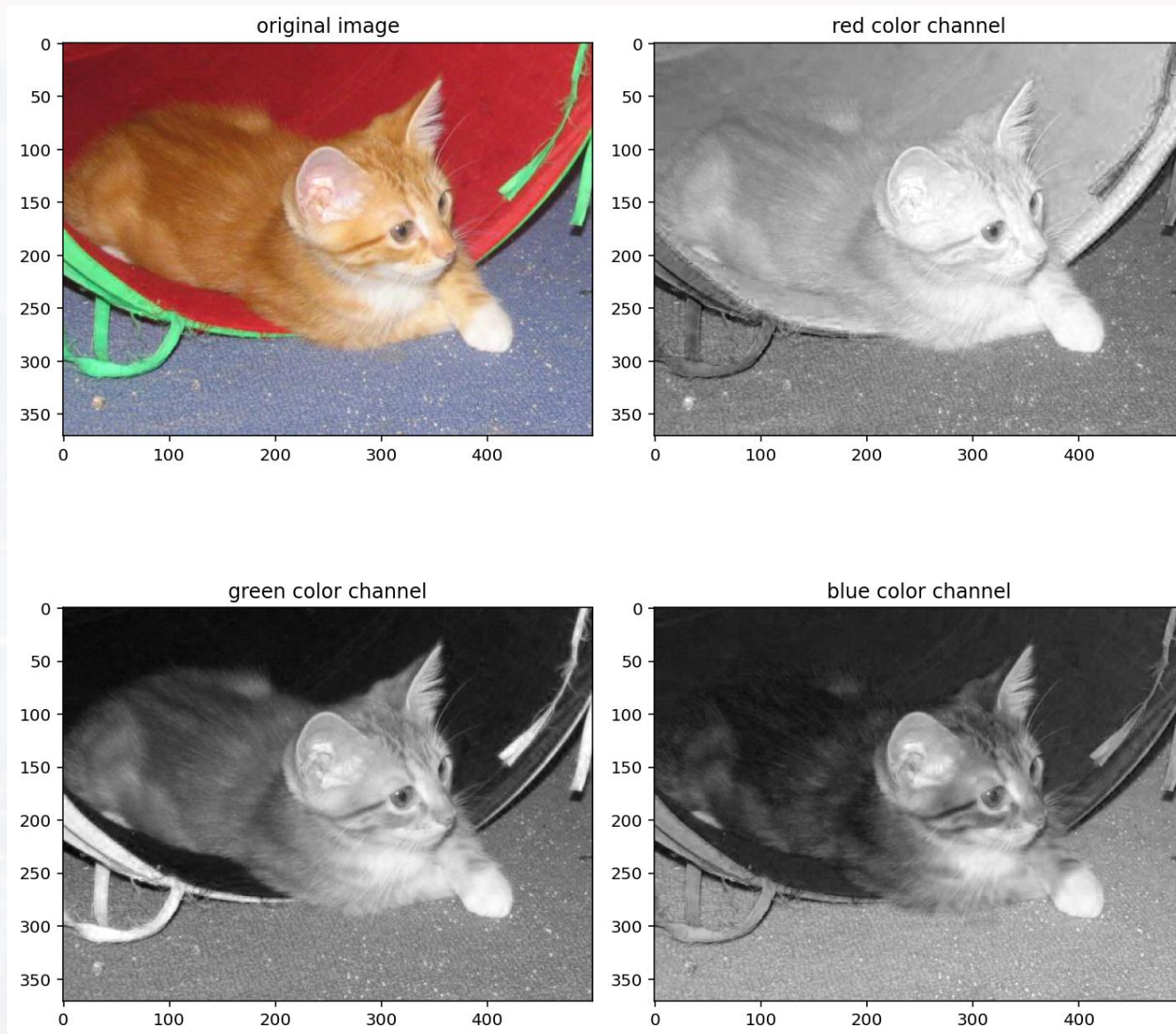
turning image into table  
(**spatial information is lost!**)

- labels are assigned according to cluster in **color space** (RGB, HSV etc)
- cluster means are mean colors

see `KMeansSegmentation.py`

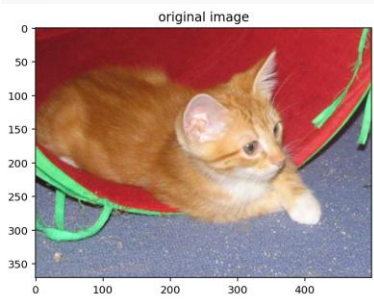


see `KMeansSegmentation.py`



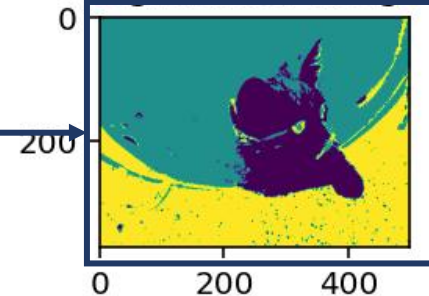
```
K = KMeansSegmentation()
```



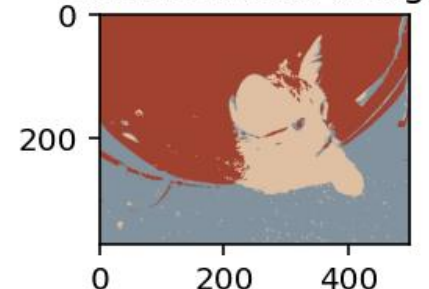


`K.Segment()`

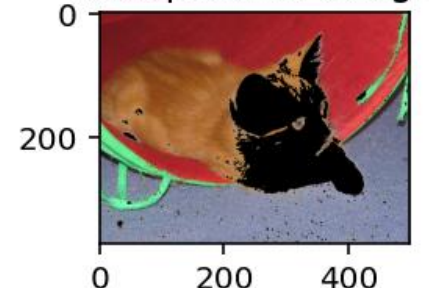
segmented image



reconstructed image

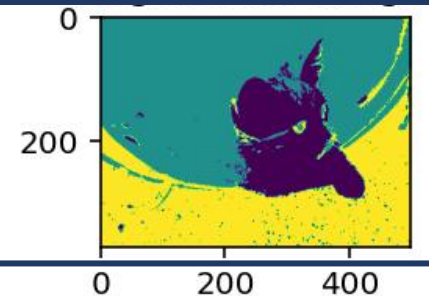


manipulated image

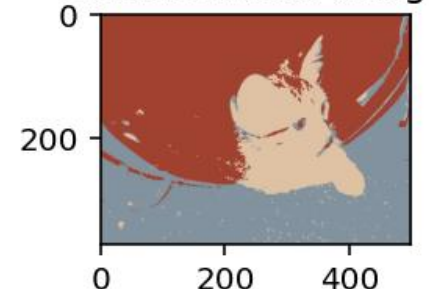


`K.Segment(values\  
= [255, 255, 255])`

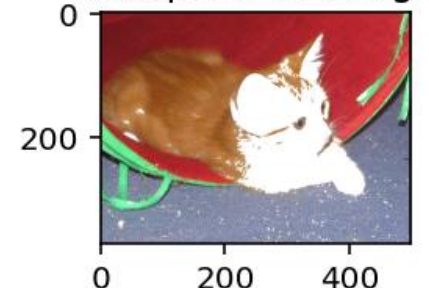
segmented image



reconstructed image

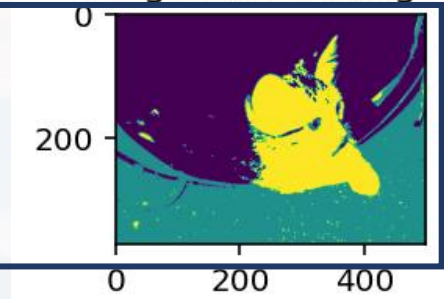


manipulated image

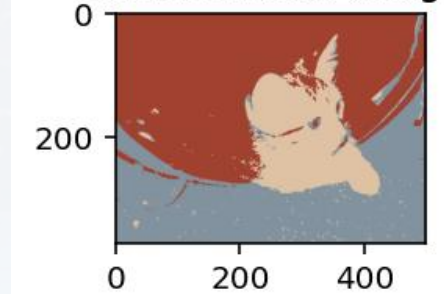


`K.Segment(values\  
= [255, 0, 0], label = 1)`

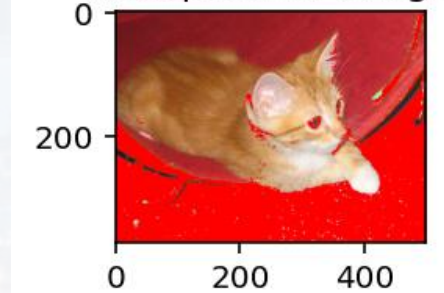
segmented image



reconstructed image



manipulated image

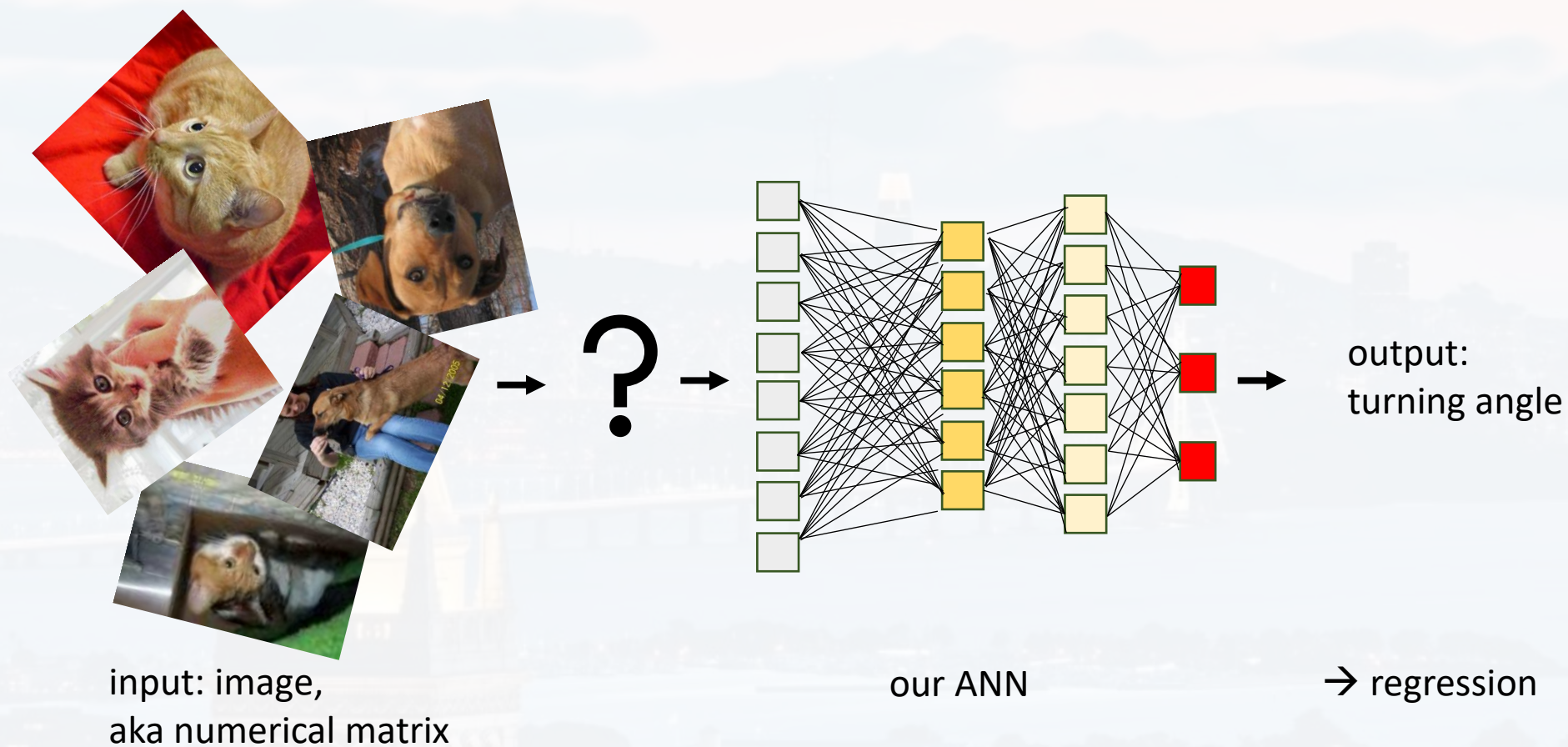


labels can swap  
(k-means is  
unsupervised)



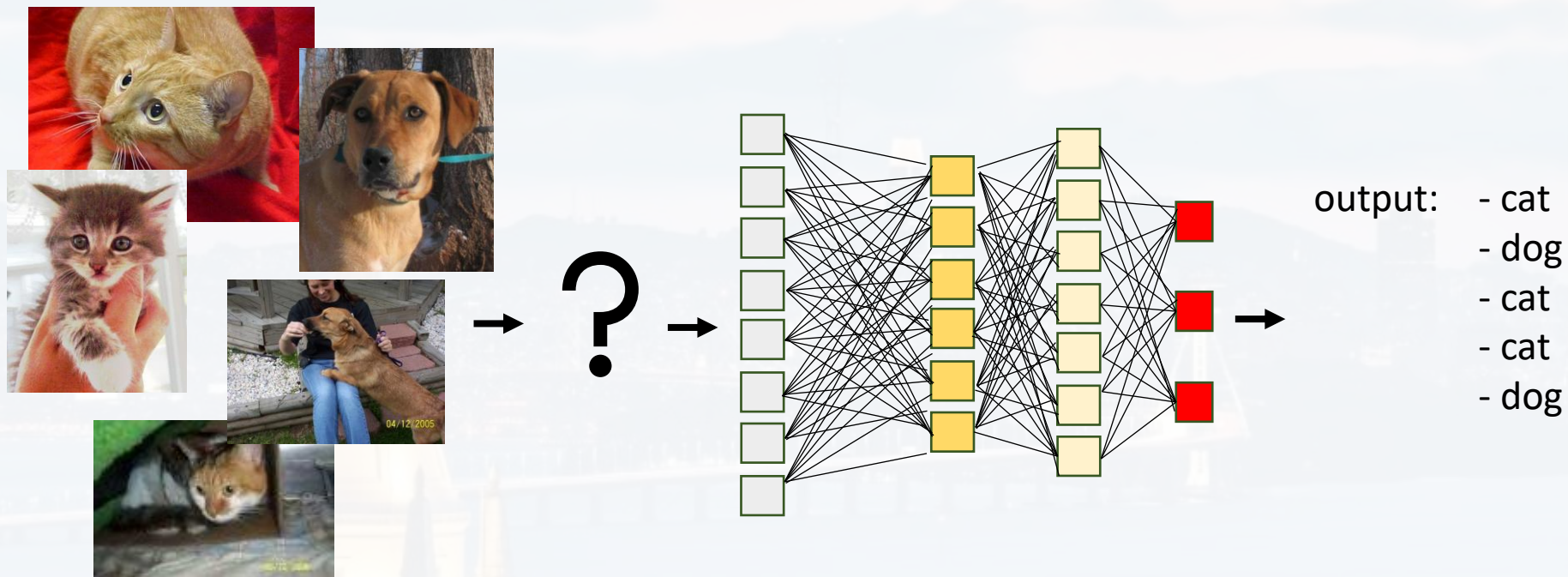


task: find a way to keep both: **color** and **spatial** information and link the image to our ANN





task: find a way to keep both: color and spatial information and link the image to our ANN



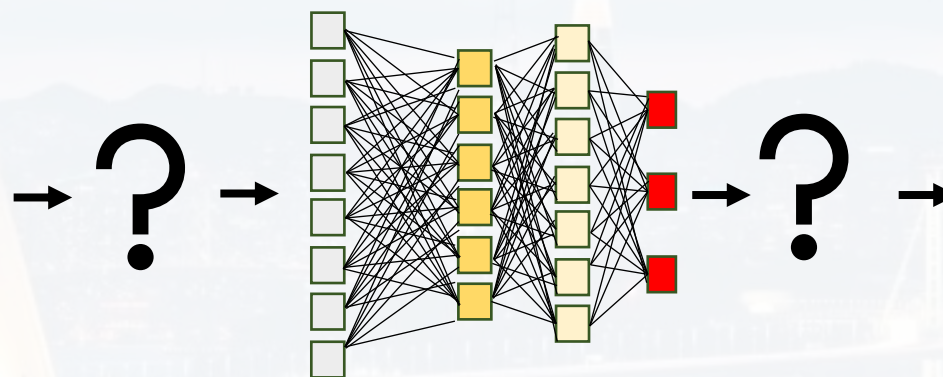




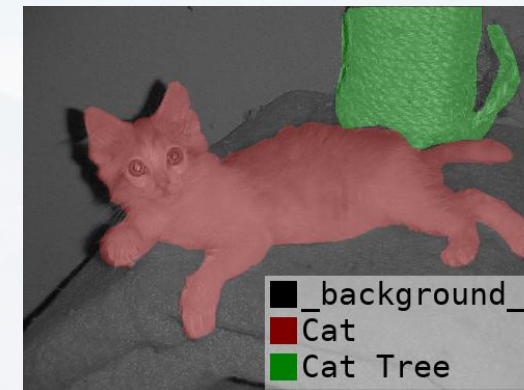
task: find a way to keep both: color and spatial information and link the image to our ANN



input: image,  
aka numerical matrix



our ANN



segmented image





```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

The Problem

**Convolution**

CNN Architectures

Data Preparation & Training

**Example**

- LeNet numpy only
- LeNet TensorFlow
- sequences as images
- segmentation

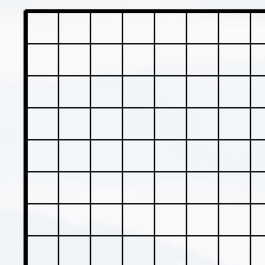


- goal:
- maintaining the spatial information
  - learning which features are important

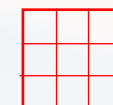
- convolution
- training the convolution filter

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\zeta) g(x - \zeta) d\zeta$$

image  $f$  and filter  $g$

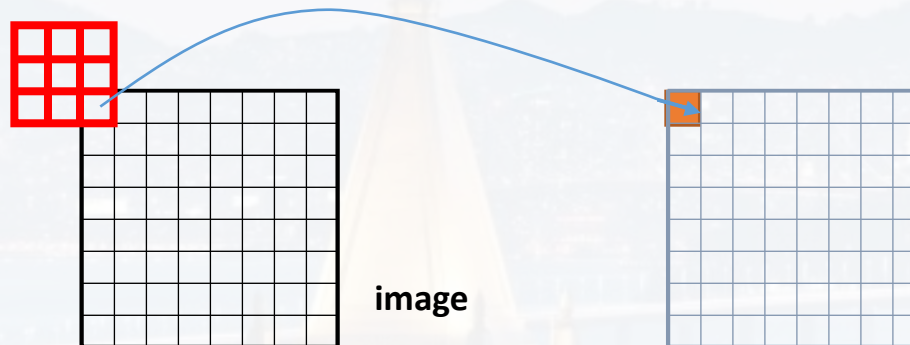


image

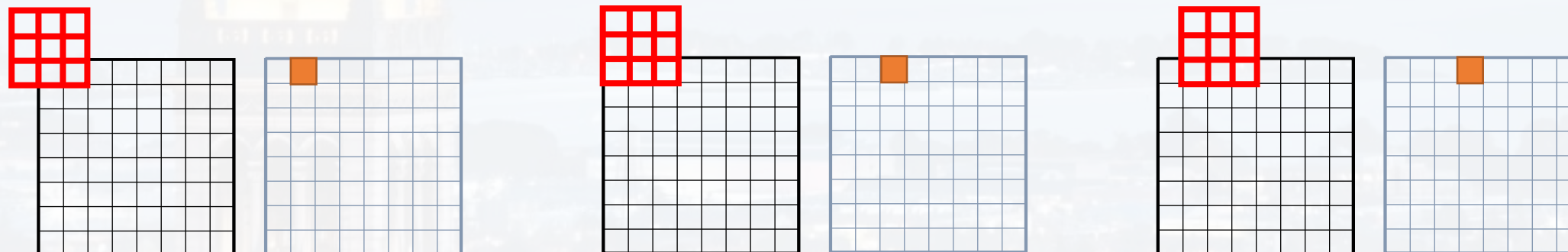


filter  
(aka kernel)

convolved image



- 1) multiplying matrix values pixelwise
- 2) summing up products
- 3) → value for new matrix



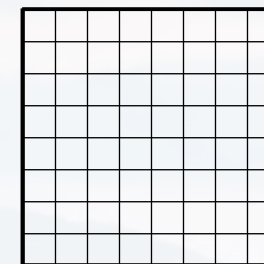


- goal:
- maintaining the spatial information
  - learning which features are important

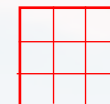
- convolution
- training the convolution filter

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\zeta) \mathbf{g}(\mathbf{x} - \zeta) d\zeta$$

image  $\mathbf{f}$  and filter  $\mathbf{g}$

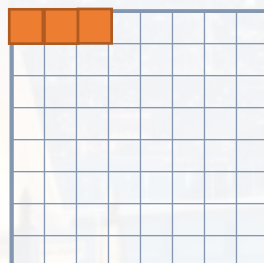
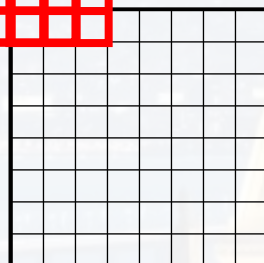
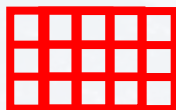


image

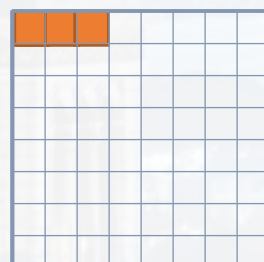
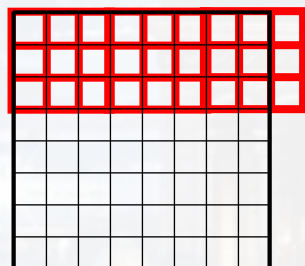


filter  
(aka kernel)

different techniques:



padding = 2; stride length = 1



padding = 0; stride length = 3

N: number of rows/columns

$$N_{out} = \frac{(N_{in} - N_{filt} + 2 * padding)}{stride\ length} + 1$$



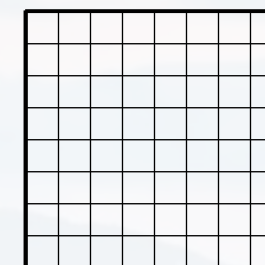


- goal:
- maintaining the spatial information
  - learning which features are important

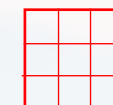
- convolution
- training the convolution filter

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\zeta) \mathbf{g}(\mathbf{x} - \zeta) d\zeta$$

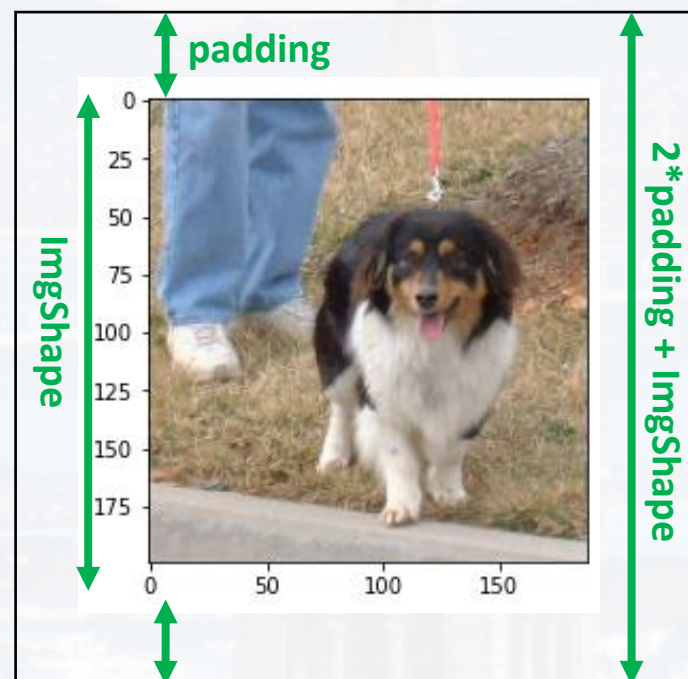
image  $\mathbf{f}$  and filter  $\mathbf{g}$



image



filter  
(aka kernel)

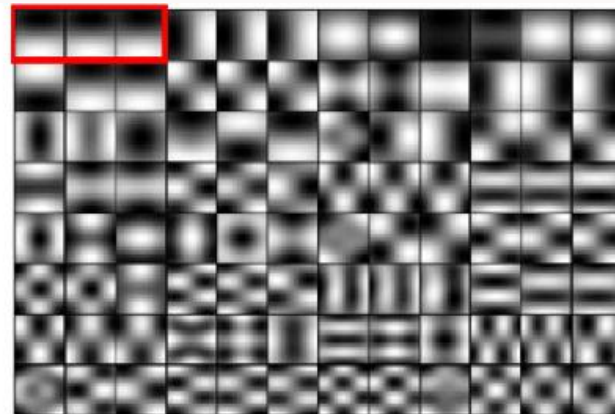


N: number of rows/columns

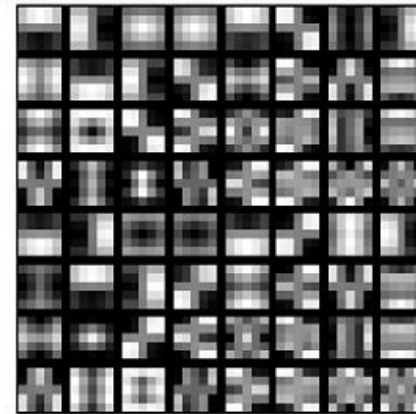
$$N_{out} = \frac{(N_{in} - N_{filt} + 2 * padding)}{stride\ length} + 1$$



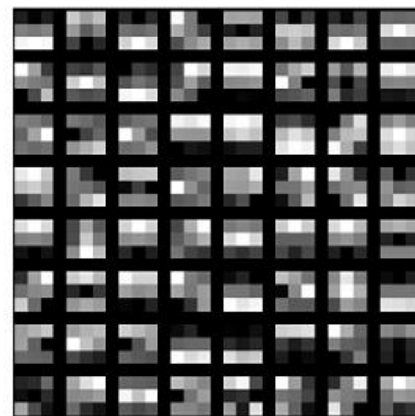
filters:



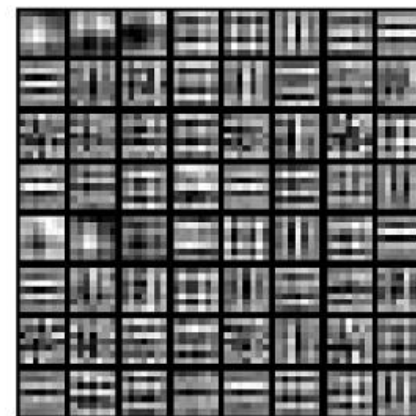
(a)



(b)



(c)



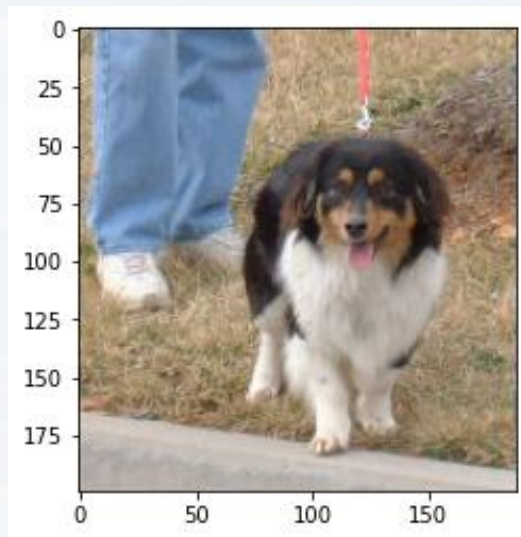
(d)

DOI:10.1016/j.actbio.2017.09.025

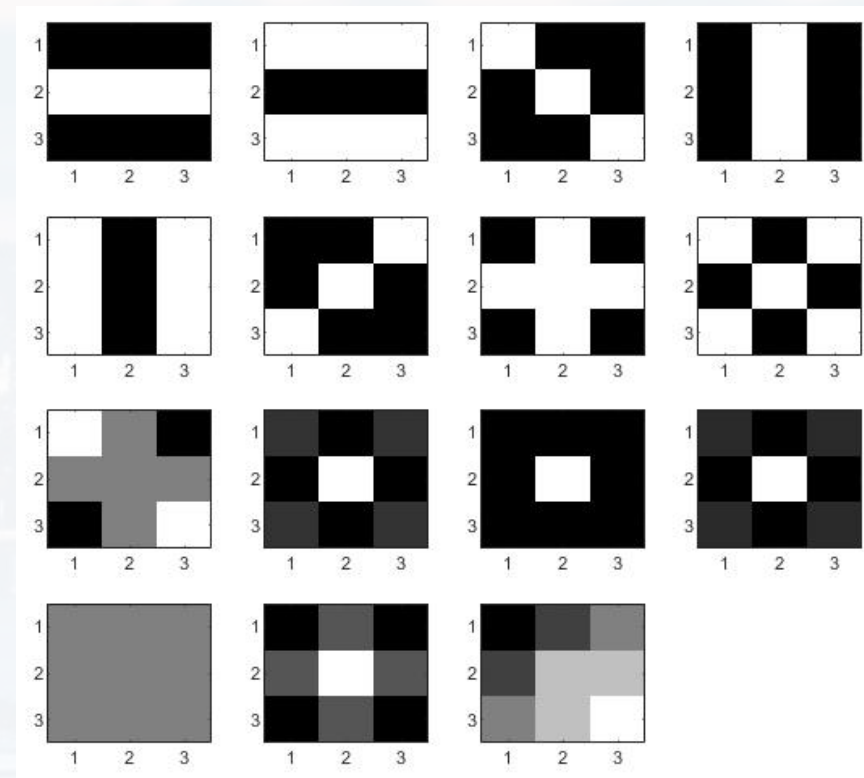


see `Convolution.ipynb` for visualizing the impact of different convolution filter on the image

image



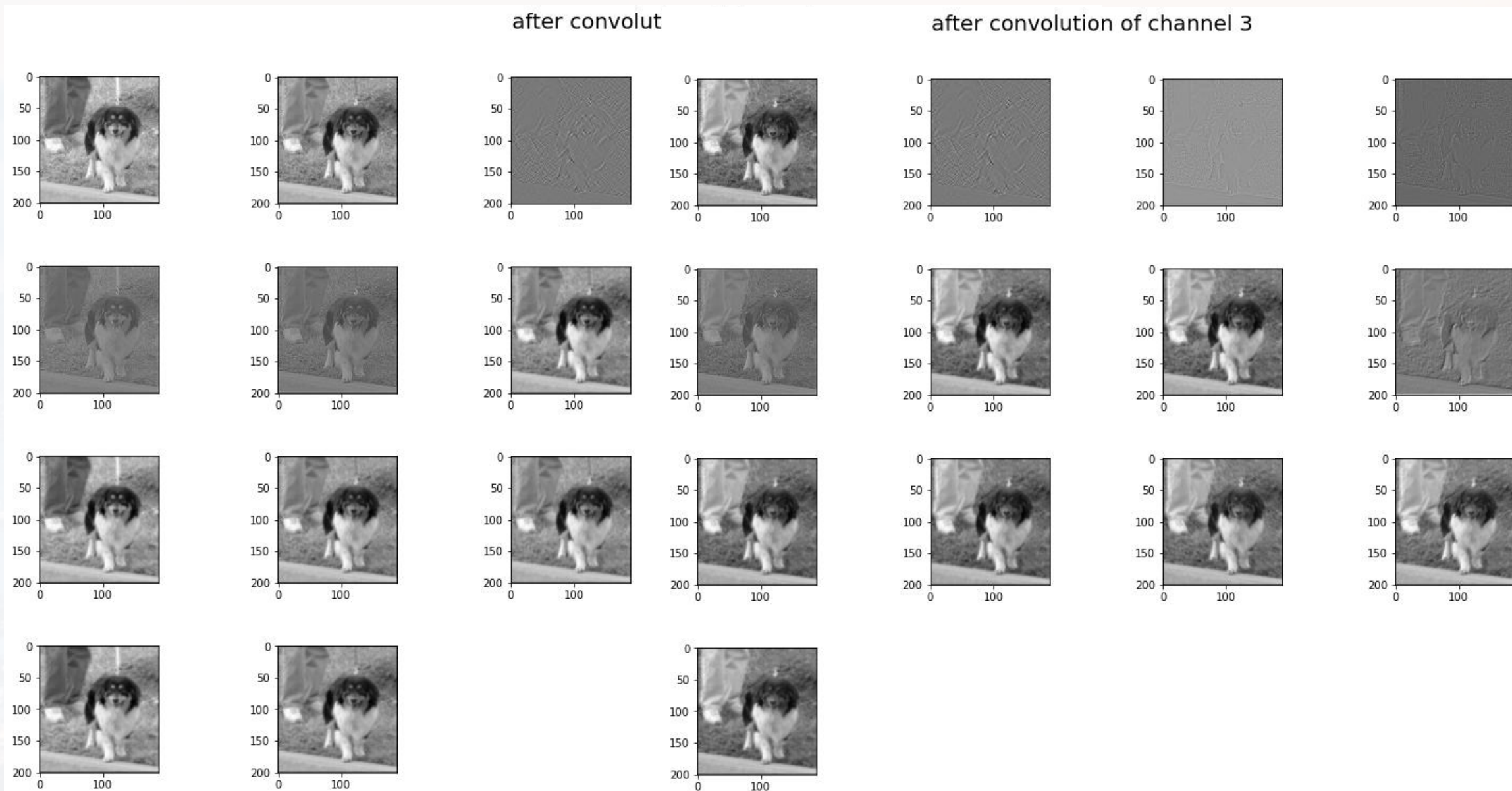
filter





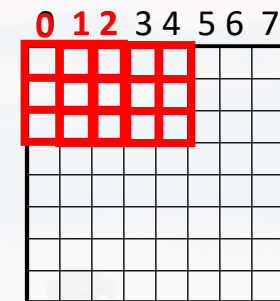


see `Convolution.ipynb` for visualizing the impact of different convolution filter on the image

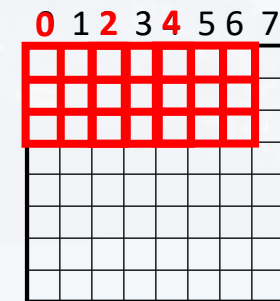




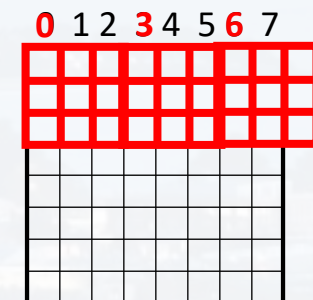
```
def ConvSelfMade(*Image, K, padding = 0, stride = 1):  
    ...  
  
    for c in range(numChans):# loop over channels  
        for y in range(yOutput):# loop over y axis of output  
            for x in range(xOutput):# loop over x axis of output  
  
                # finding corners of the current "slice"  
                y_start = y*stride  
                y_end   = y*stride + yK  
                x_start = x*stride  
                x_end   = x*stride + xK  
  
                #selecting the current part of the image  
                current_slice = imagePadded[x_start:x_end,\  
                                             y_start:y_end, c]  
  
                #the actual convolution part  
                s              = np.multiply(current_slice, K)  
                output[x,y,c] = np.sum(s)
```



stride = 1



stride = 2

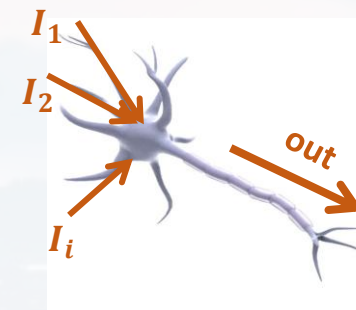


stride = 3



- CNN:
- kernel act like **neurons with weights**
  - start with random values for all kernels
  - the ANN **learns the filter values**
  - that's how the ANN learns which features are important

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\zeta) \mathbf{g}(\mathbf{x} - \boldsymbol{\zeta}) d\zeta$$



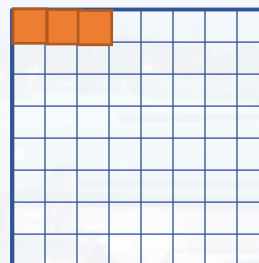
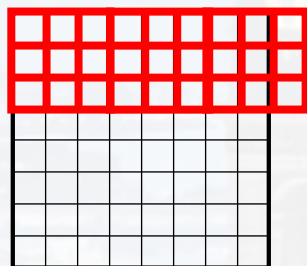
$$net = \sum_i I_i \cdot w_i + b$$

inputs are pixel values

**kernel weights**

$$\sum_i I_i \mathbf{w}_i + b$$

can be interpreted as a neuron!



i. e. a filter of shape  $N \times N$   
has  $N^2 + 1$  learnables ( $N^2$  weights + bias)





```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

The Problem

Convolution

**CNN Architectures**

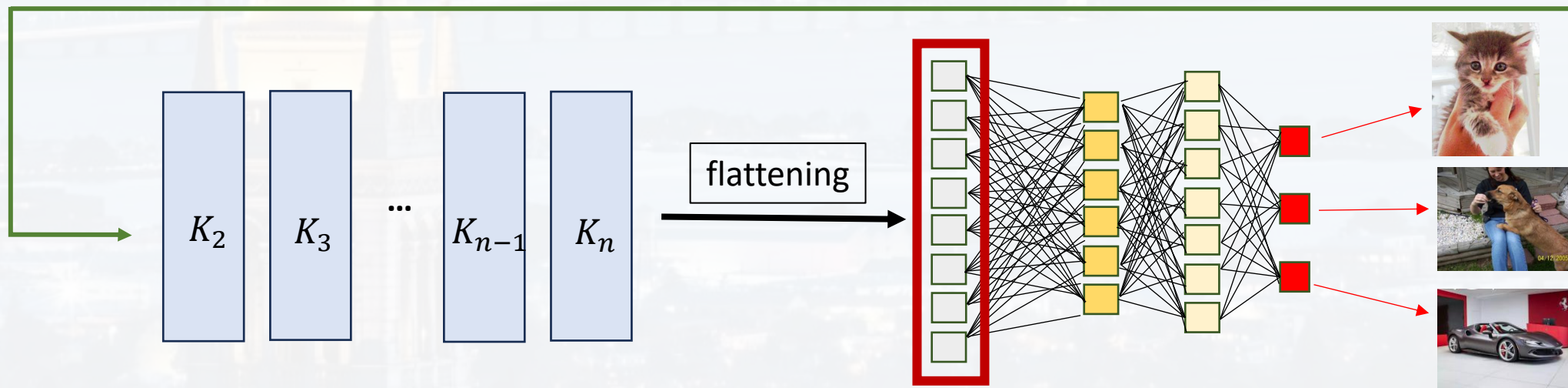
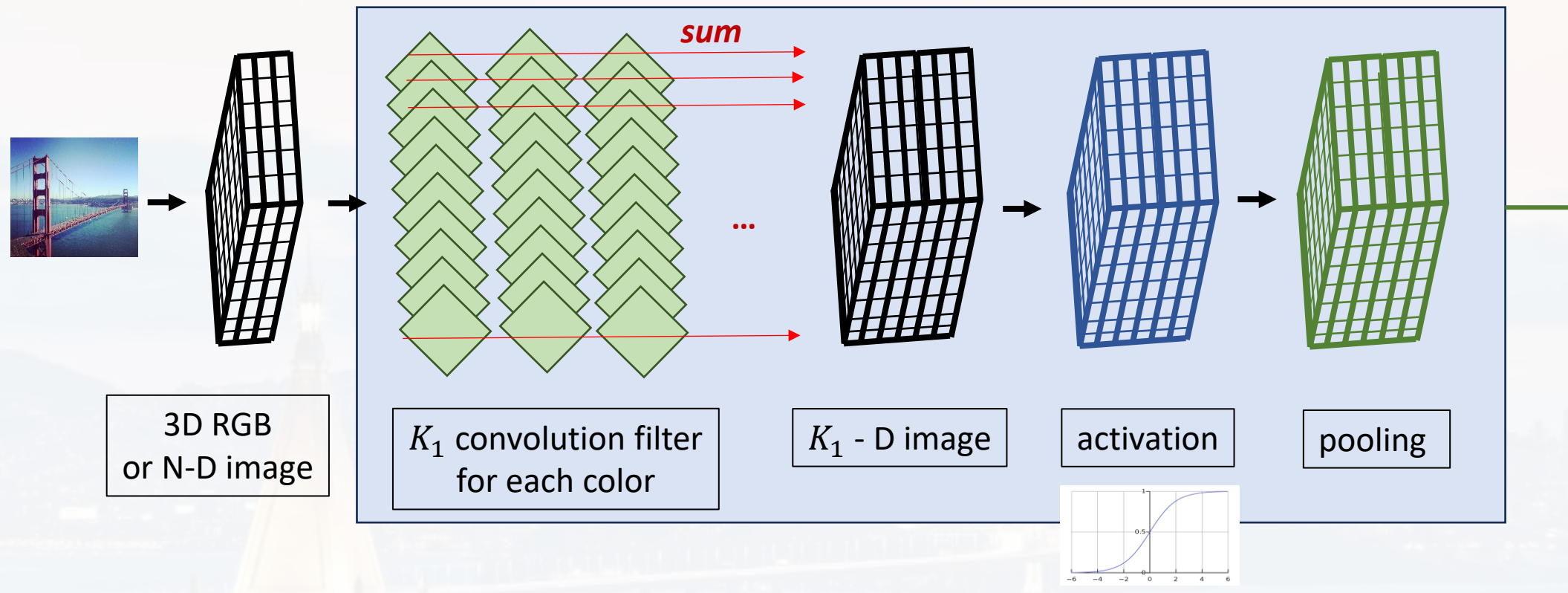
Data Preparation & Training

Example

- LeNet numpy only
- LeNet TensorFlow
- Segmentation



classification





### classification pooling:

there are three different main pooling methods

→ **average pool:**

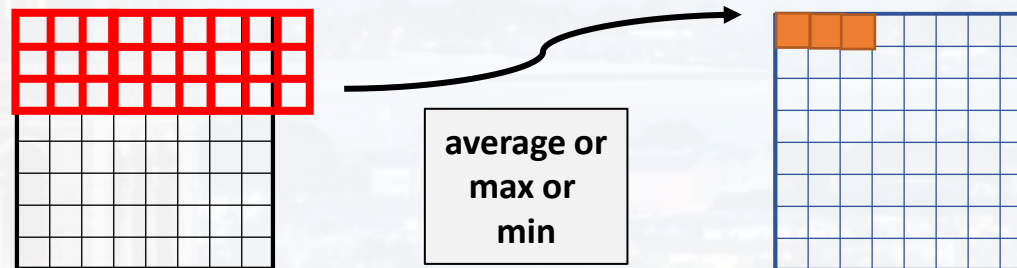
**blurs the image**, reduces edges  
(not what we want here)

→ **max pool:**

**reduces dark background** (those pixel values are usually low) and **enhances brighter foreground objects** (exactly what we need here)

→ **min pool:**

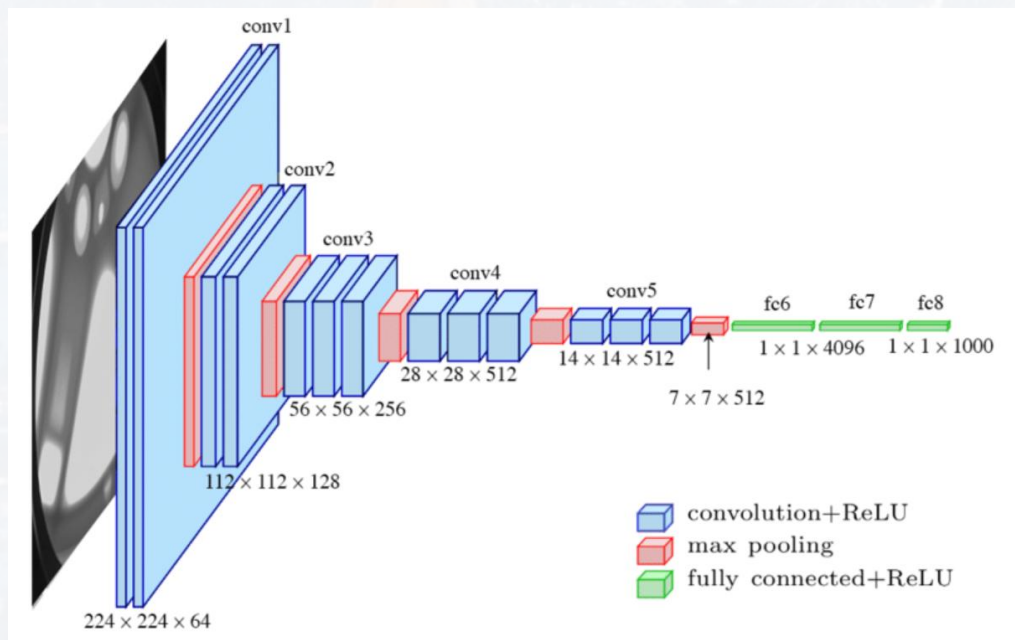
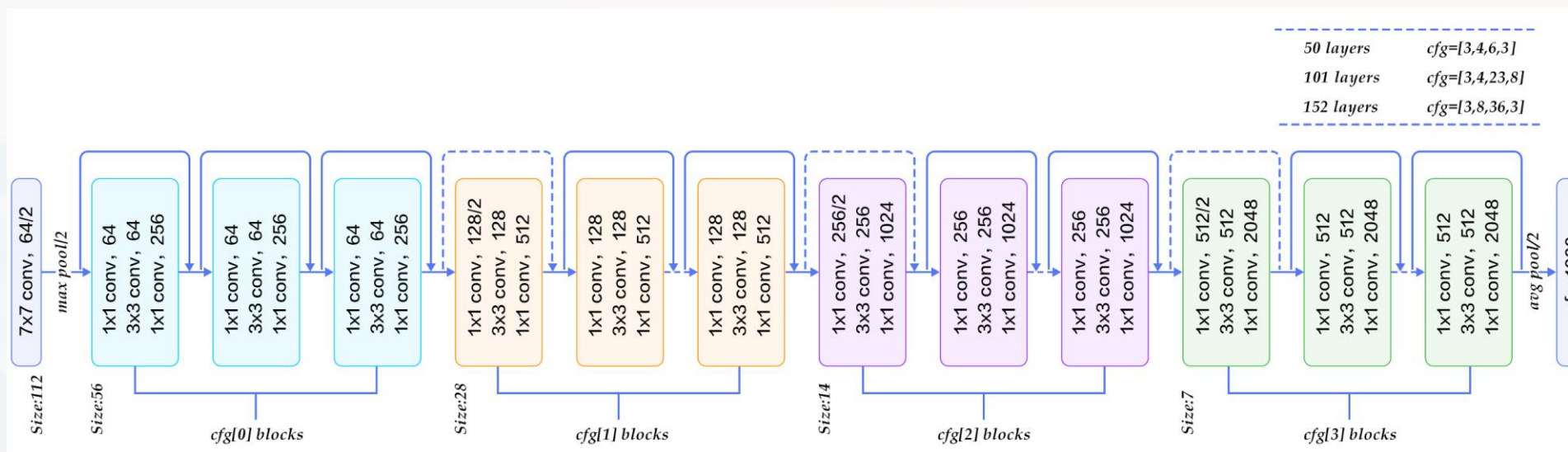
does the opposite of max pool







### classification



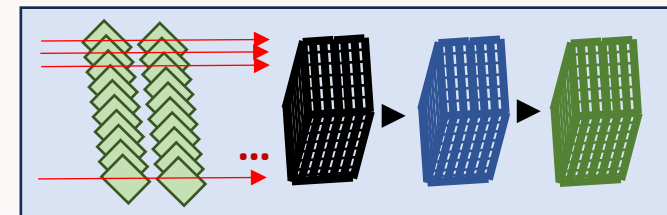
convolution process down-samples the image

- network focuses on important features
- reduces computational cost

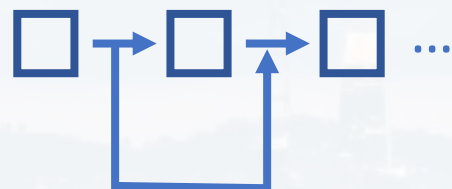


### classification

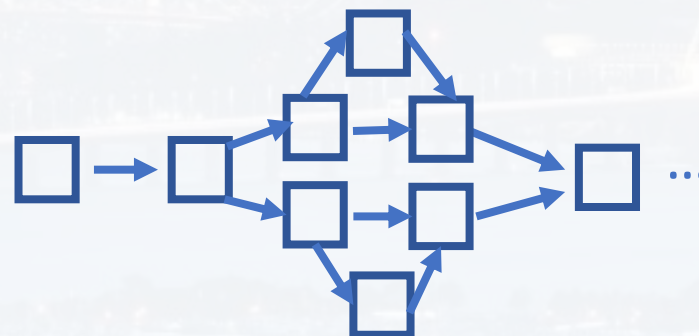
sequential CNNs



**ResidualNet**



**Inception Net**



many others...



### classification

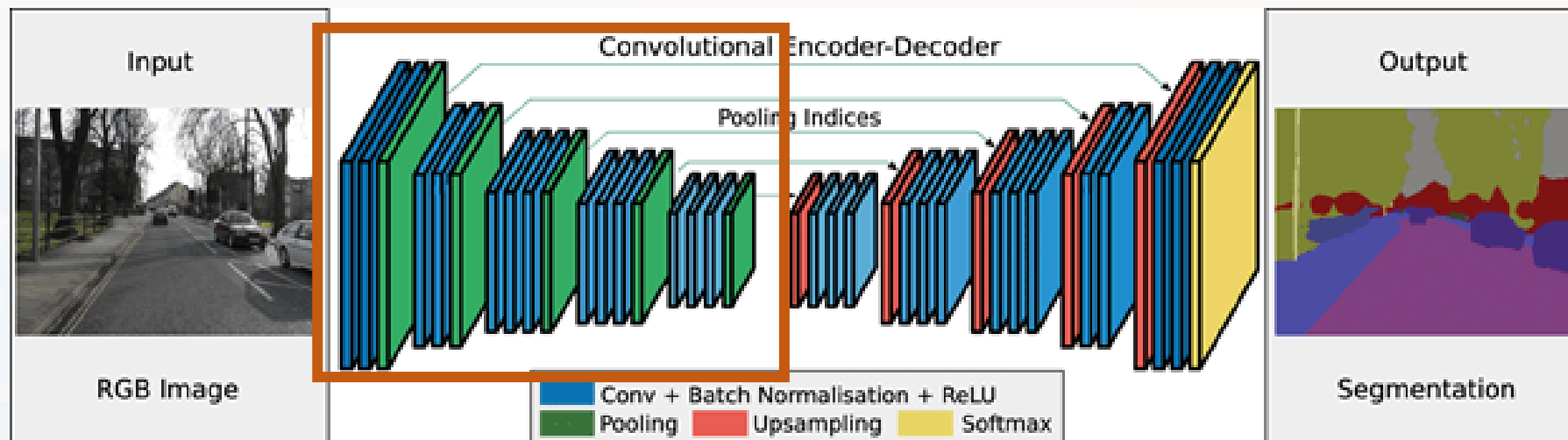
common pretrained classification CNNs

Network	Depth	Size	Parameters (Millions)	rel computation time	Image Input Size
nasnetlarge	*	360 MB	88,9	45	331-by-331
darknet19	19	72.5 MB	21	5,5	256-by-256
densenet201	201	77 MB	20	22	224-by-224
resnet50	50	96 MB	25,6	3,5	224-by-224
resnet101	101	167 MB	44,6	5	224-by-224
inceptionv3	48	89 MB	23,9	8	299-by-299
resnet18	18	44 MB	11,7	1,8	224-by-224
xception	71	85 MB	22,9	12	299-by-299
darknet53	53	145 MB	41	10	256-by-256
inceptionresnetv2	164	209 MB	55,9	14	299-by-299
shufflenet	50	6.3 MB	1,4	1,5	224-by-224
googlenet	22	27 MB	7	2	224-by-224
mobilenetv2	53	13 MB	3,5	4	224-by-224
alexnet	8	227 MB	61	1,2	227-by-227
nasnetmobile	*	20 MB	5,3	5	224-by-224
squeezenet	18	4.6 MB	1,24	1	227-by-227
vgg16	16	515 MB	138	6,5	224-by-224
vgg19	19	535 MB	144	8,5	224-by-224





### segmentation

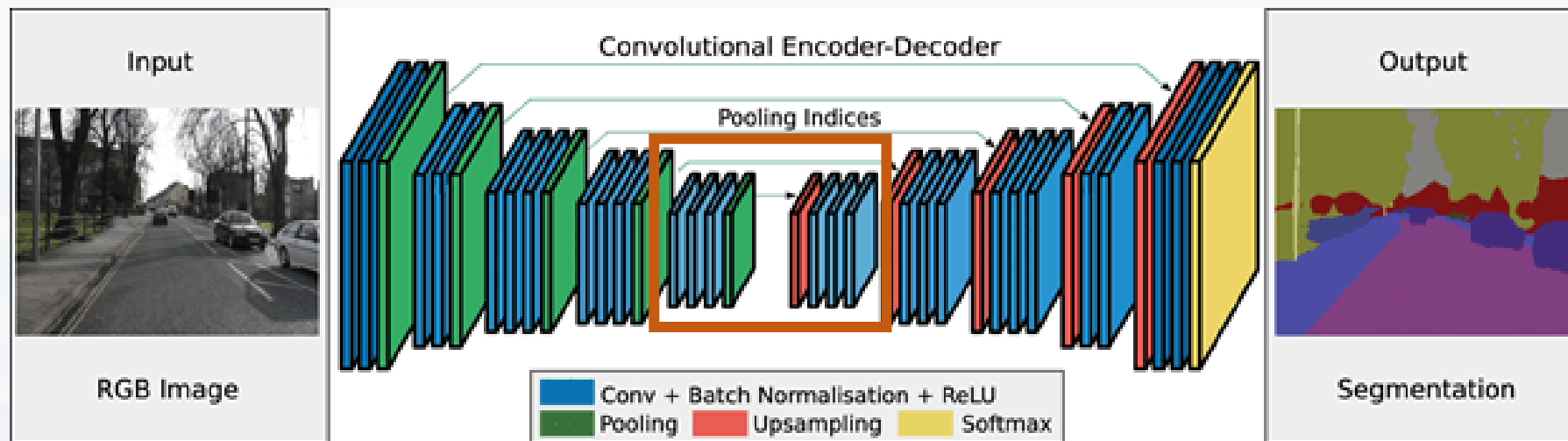


*Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"*

1) down-sampling as before (= **encoder**)



### segmentation

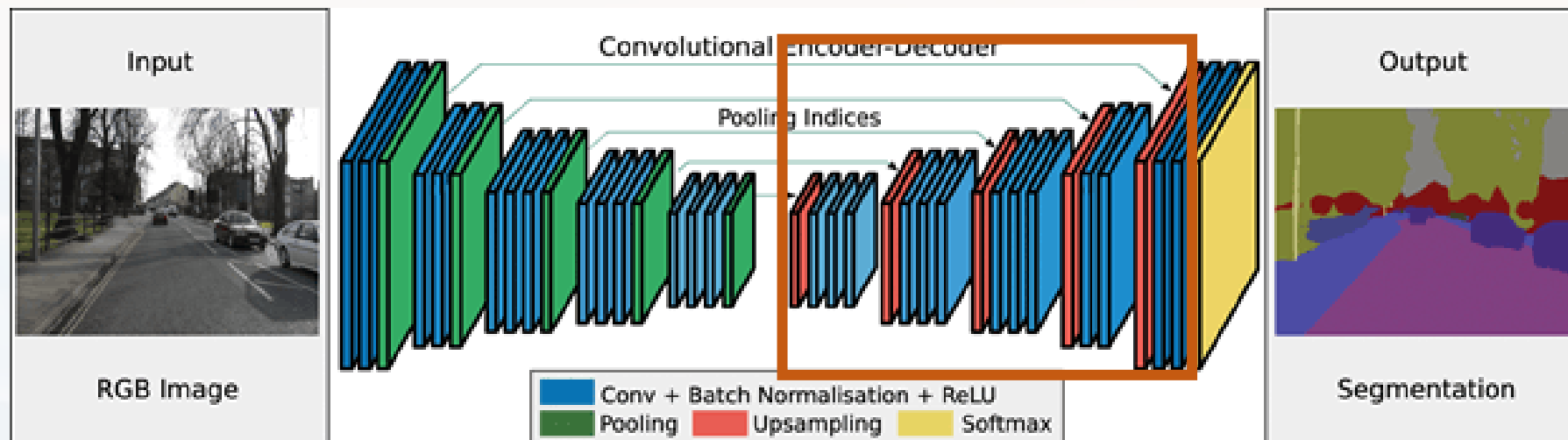


*Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"*

- 1) down-sampling as before (= **encoder**)
- 2) down to bottle neck



### segmentation



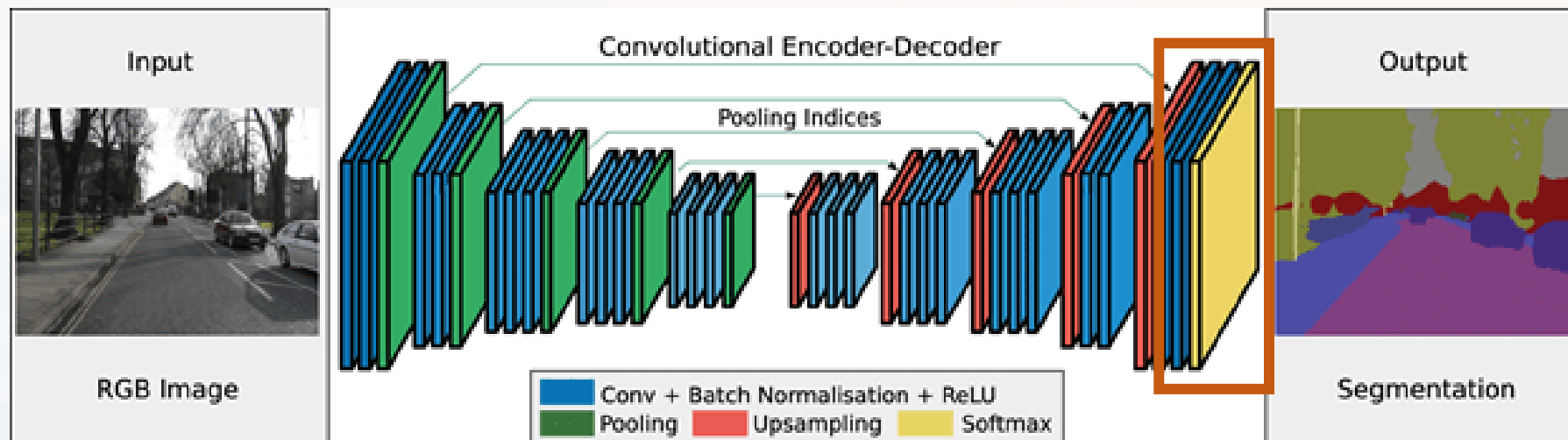
*Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"*

- 1) down-sampling as before (= **encoder**)
- 2) down to bottle neck
- 3) up-sampling (= **decoder**; how: see later) in order to generate output image of the same size as input image, where number of channels = number of pixel classes





### segmentation

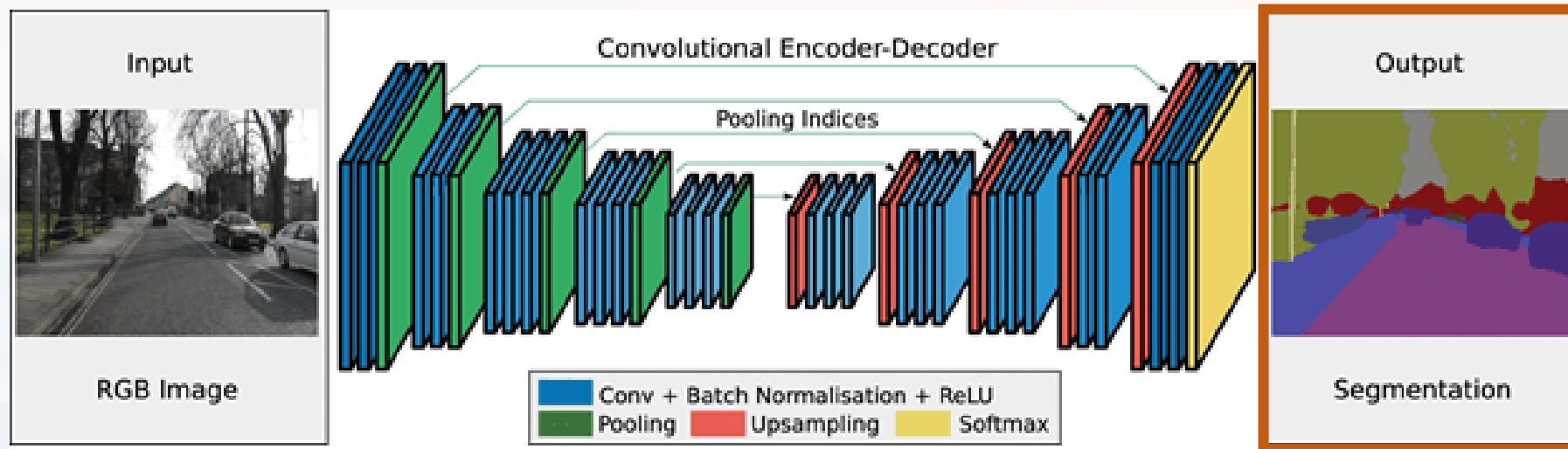


*Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"*

- 1) down-sampling as before (= **encoder**)
- 2) down to bottle neck
- 3) up-sampling (= **decoder**; how: see later) in order to generate output image of the same size as input image, where number of channels = number of pixel classes
- 4) softmax, in order to turn output of last layer into probabilities



### segmentation

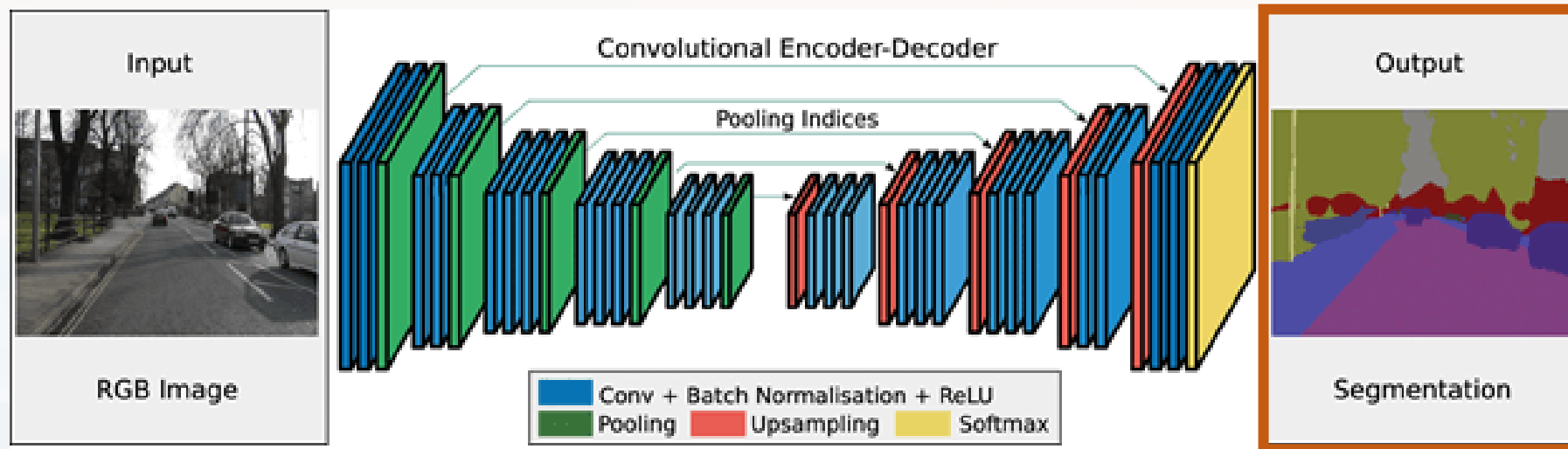


*Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"*

- 1) down-sampling as before (= encoder)
  - 2) down to bottle neck
  - 3) up-sampling (= **decoder**; how: see later) in order to generate output image of the same size as input image, where number of channels = number of pixel classes
  - 4) softmax, in order to turn output of last layer into probabilities
  - 5) generates segmentation mask from highest probabilities
- = (arbitrary) colors are class labels and correspond to pixel class



### segmentation



*Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"*

depending on the architecture, the decoder has **different learning mechanisms**

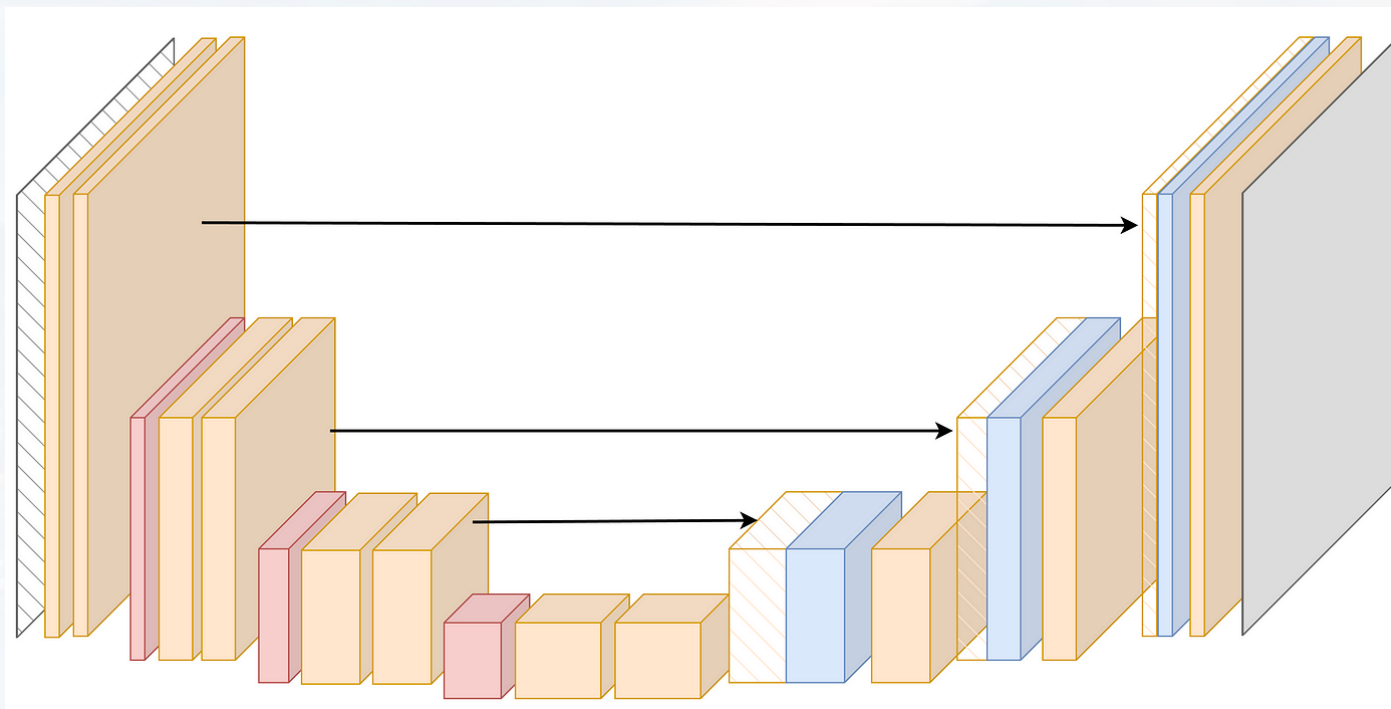
- standard: up-sampling = inverse convolution: weights and biases are learnables
- U-Net: skips connections and concatenates decoder layer with corresponding encoder layer information
- transformer encoder: use attention (see later)





### segmentation

#### U-net segmentation CNN



<https://towardsdatascience.com/u-net-explained-understanding-its-image-segmentation-architecture-56e4842e313a>

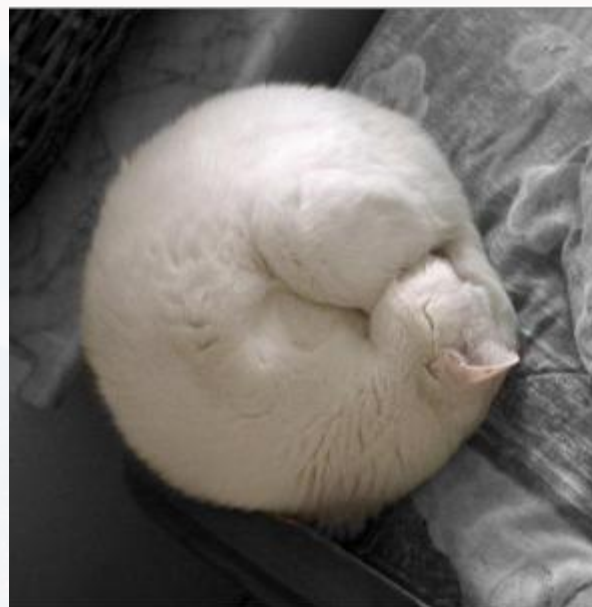


### segmentation

common pretrained segmentation CNNs

**note: the input size is usually 5 – 10 times larger than compared to a classification CNN!**

Type	Names
VGG	'vgg16' 'vgg19'
ResNet	'resnet18' 'resnet34' 'resnet50' 'resnet101' 'resnet152'
SE-ResNet	'seresnet18' 'seresnet34' 'seresnet50' 'seresnet101' 'seresnet152'
ResNeXt	'resnext50' 'resnext101'
SE-ResNeXt	'seresnext50' 'seresnext101'
SENet154	'senet154'
DenseNet	'densenet121' 'densenet169' 'densenet201'
Inception	'inceptionv3' 'inceptionresnetv2'
MobileNet	'mobilenet' 'mobilenetv2'
EfficientNet	'efficientnetb0' 'efficientnetb1' 'efficientnetb2' 'efficientnetb3' 'efficientnetb4' 'efficientnetb5' 'efficientnetb6' 'efficientnetb7'



```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

The Problem

Convolution

CNN Architectures

**Data Preparation & Training**

Example

- LeNet numpy only
- LeNet TensorFlow
- sequences as images
- segmentation

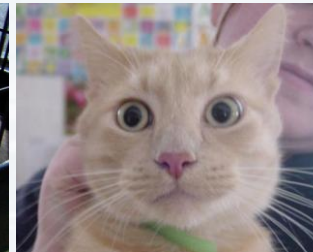




### data acquisition

1) classes should be **well balanced**

2) dataset should be **diverse**



**example Cryo-EM:**

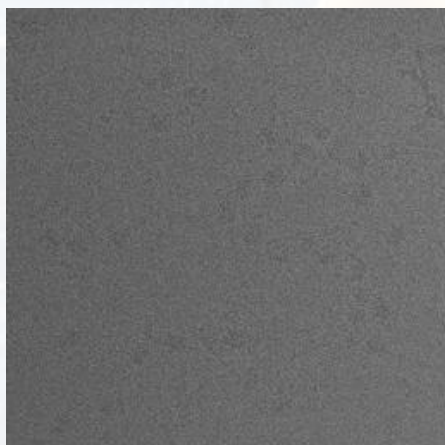
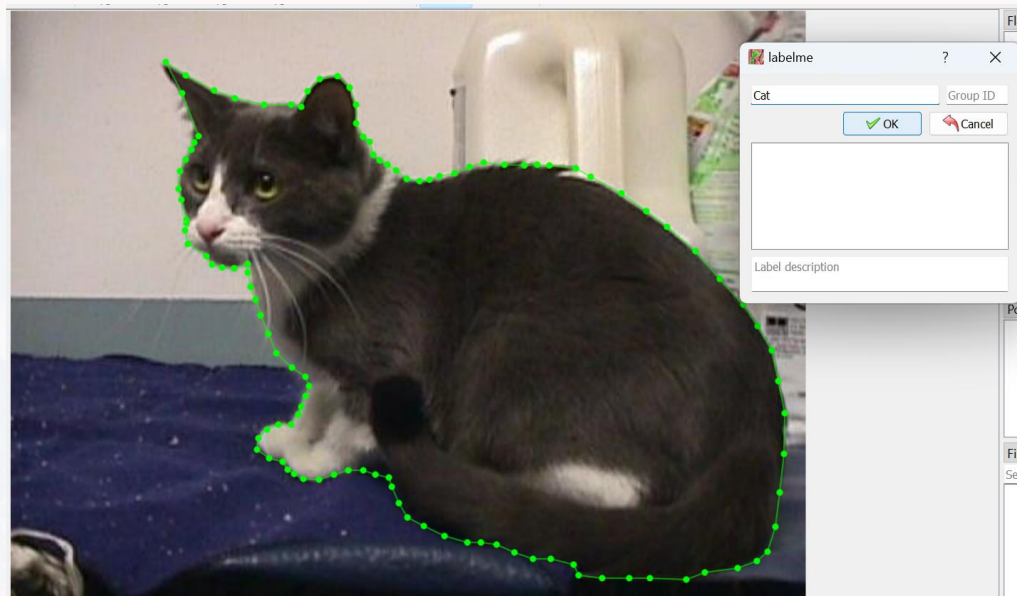
all grids (Cu, Au, ...)  
all cameras  
all grid manufacturers  
all resolutions

3) **augmentation:** blurred, skewed, fragmented, stretched, turned etc  
**tip: write your own augmentation routine!**



### data labeling

be as accurate as possible!



micrograph Cryo-EM image

→ good, medium and bad based on ice crystals

→ Undergrad, Grad, PostDoc, **Senior Scientist**



### data preprocessing

scaling:

Image Input Size
331-by-331
256-by-256
224-by-224
224-by-224

All images have to be scaled to the input size of the CNN!

normalization: images can be

- logical (values are zero **or** one)
- gray scale (2D) → adding two more “color channels”
- 8bit (range 255), 16bit (range 512) etc





### training

normalization: complex CNNs have many layers for normalization/re-centering/re-scaling

→ **batch normalization**

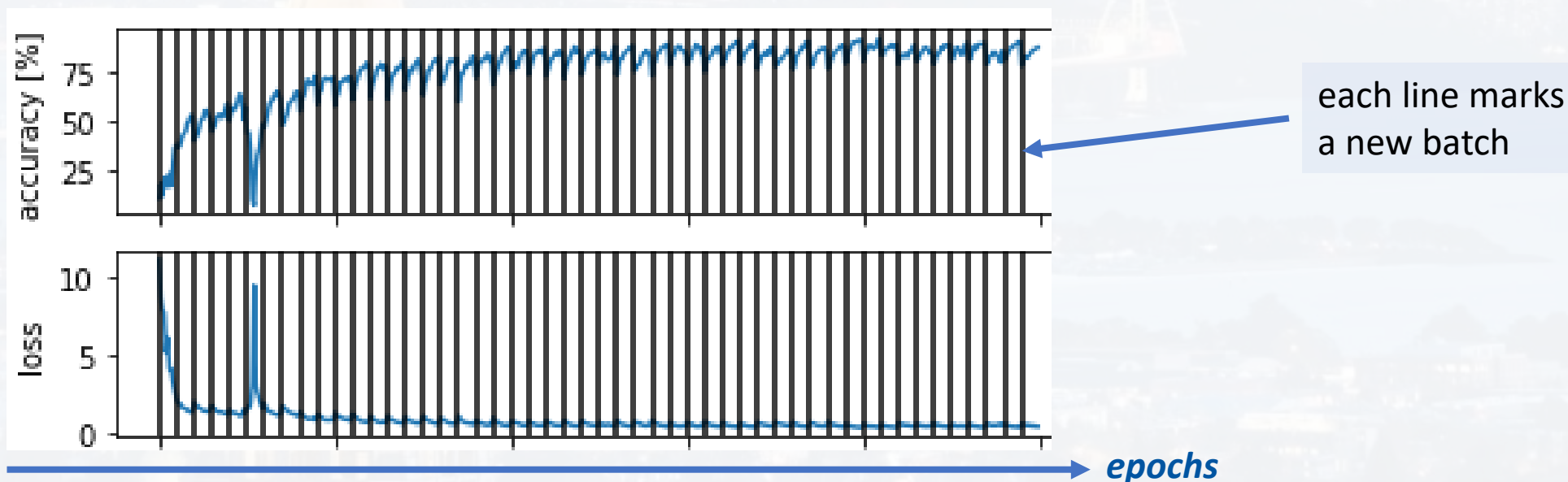
the training set is huge

→ loading only a few images at the time (**batches**)

→ the larger the batch, the better

→ run only **a few iterations** per batch (avoiding local minima)

→ check **training loss vs evaluation loss**





training

check out:

[Training MLP](#)

[Training CNN 2D](#)

[Training CNN 3D](#)



## End of Part I

