

Lecture 09:

Convolutional Neural Networks (CNN)



Markus Hohle

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Fall 2024



Outline

- The Problem
- What is Convolution
- The CNN Architectures
- Data Preparation & Training
- A Simple Example



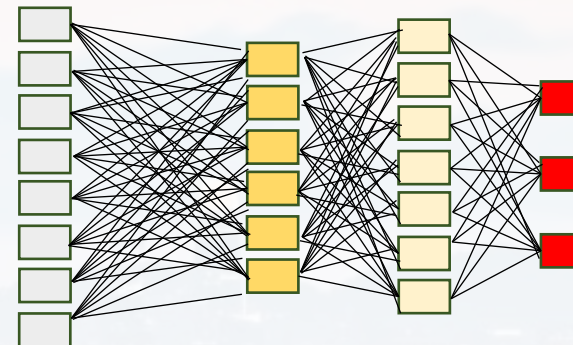
Outline

- **The Problem**
- What is Convolution
- The CNN Architectures
- Data Preparation & Training
- A Simple Example



so far:

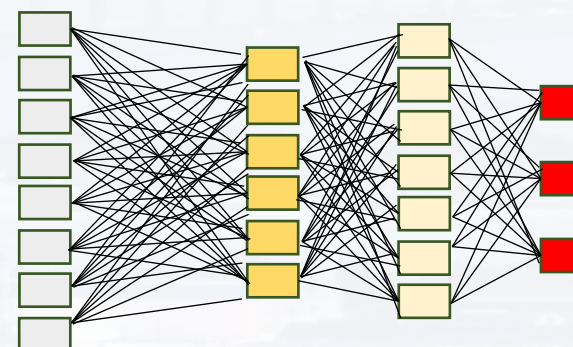
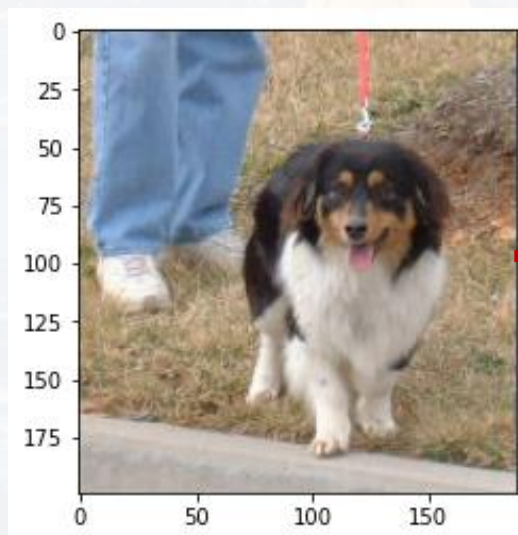
data features					
data points	[5.1	3.5	1.4	0.2]
	[4.9	3.	1.4	0.2]
	[4.7	3.2	1.3	0.2]
	[4.6	3.1	1.5	0.2]
	[5.	3.6	1.4	0.2]
	[5.4	3.9	1.7	0.4]
	[4.6	3.4	1.4	0.3]
	[5.	3.4	1.5	0.2]



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

row = data point

now:



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



so far:

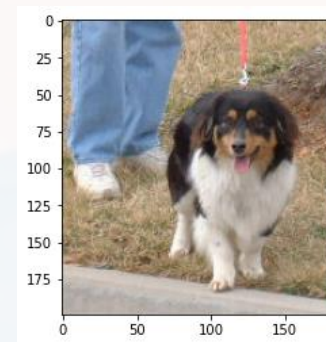
data features

data points ↓

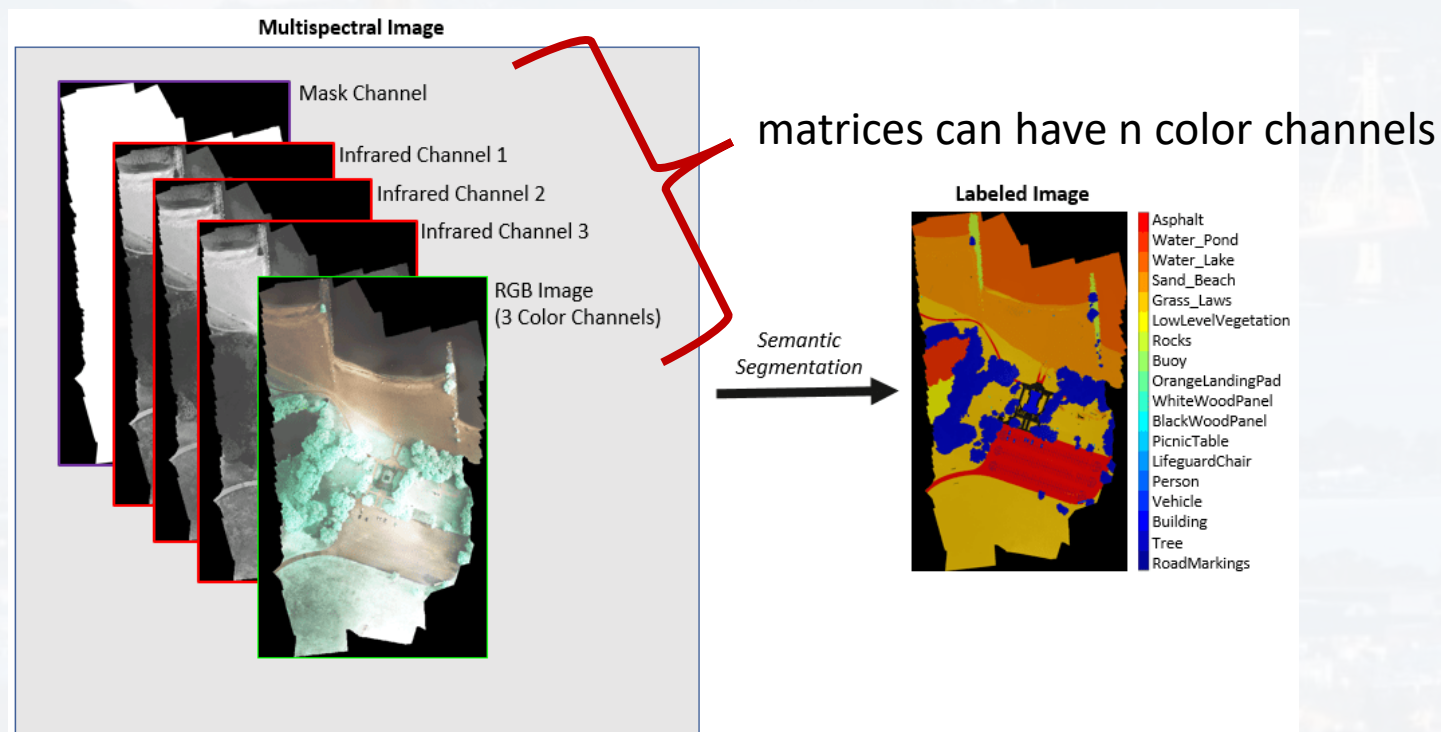
[5.1	3.5	1.4	0.2]
[4.9	3.	1.4	0.2]
[4.7	3.2	1.3	0.2]
[4.6	3.1	1.5	0.2]
[5.	3.6	1.4	0.2]
[5.4	3.9	1.7	0.4]
[4.6	3.4	1.4	0.3]
[5.	3.4	1.5	0.2]

row = data point

now:

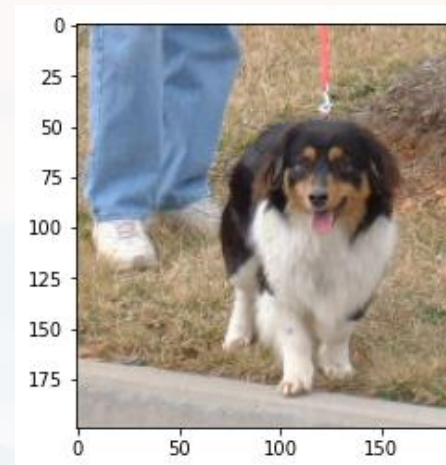


matrix = data point

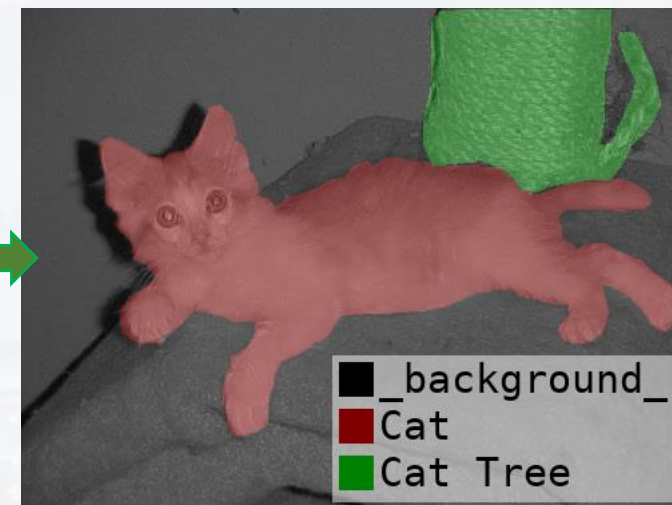
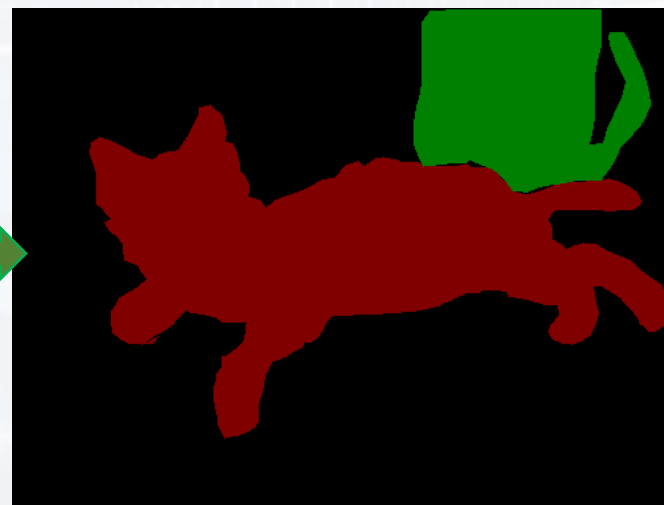
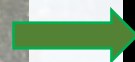




classification: between different images

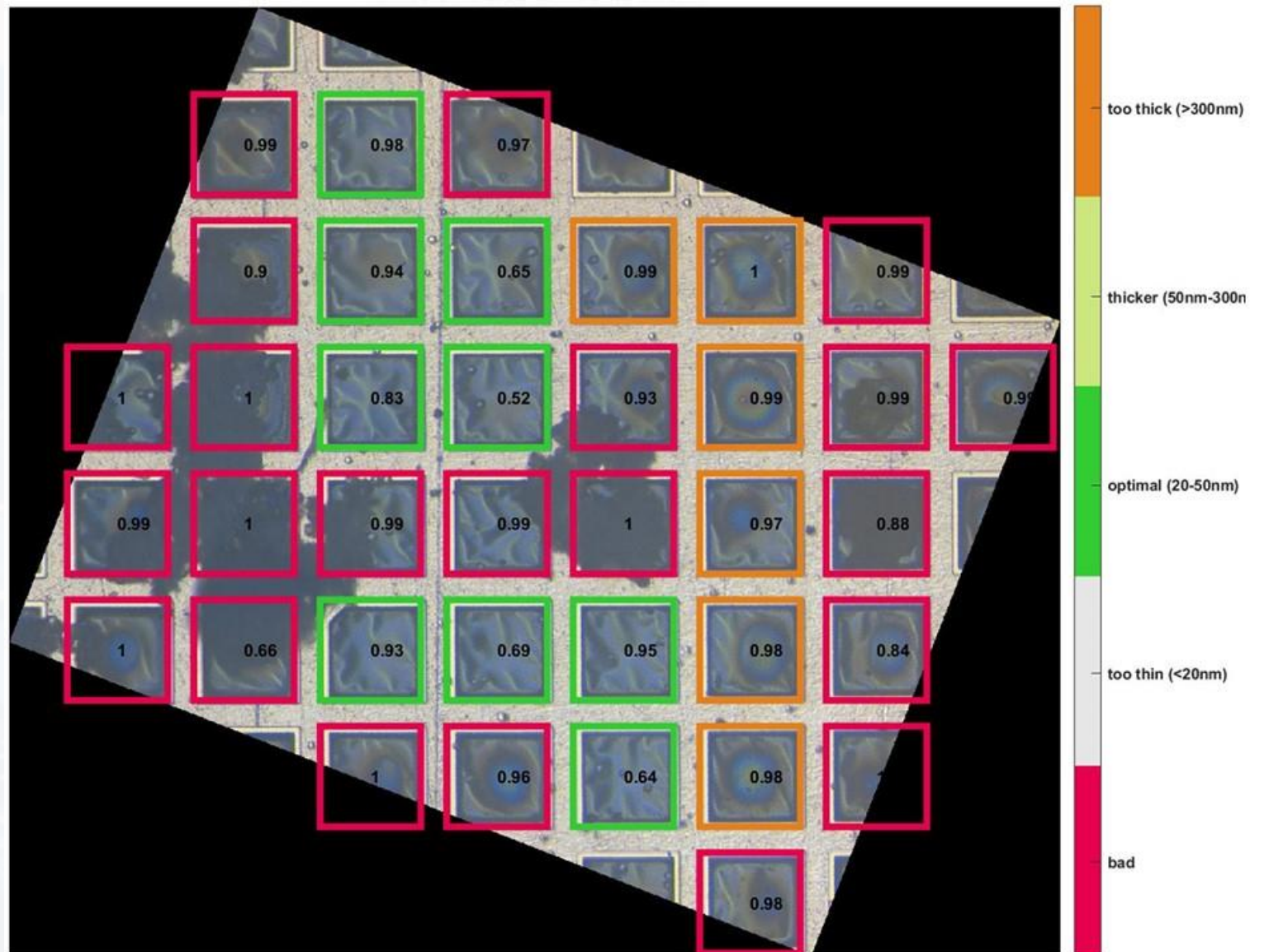


different pixel within images aka **segmentation**





segmentation *for* classification

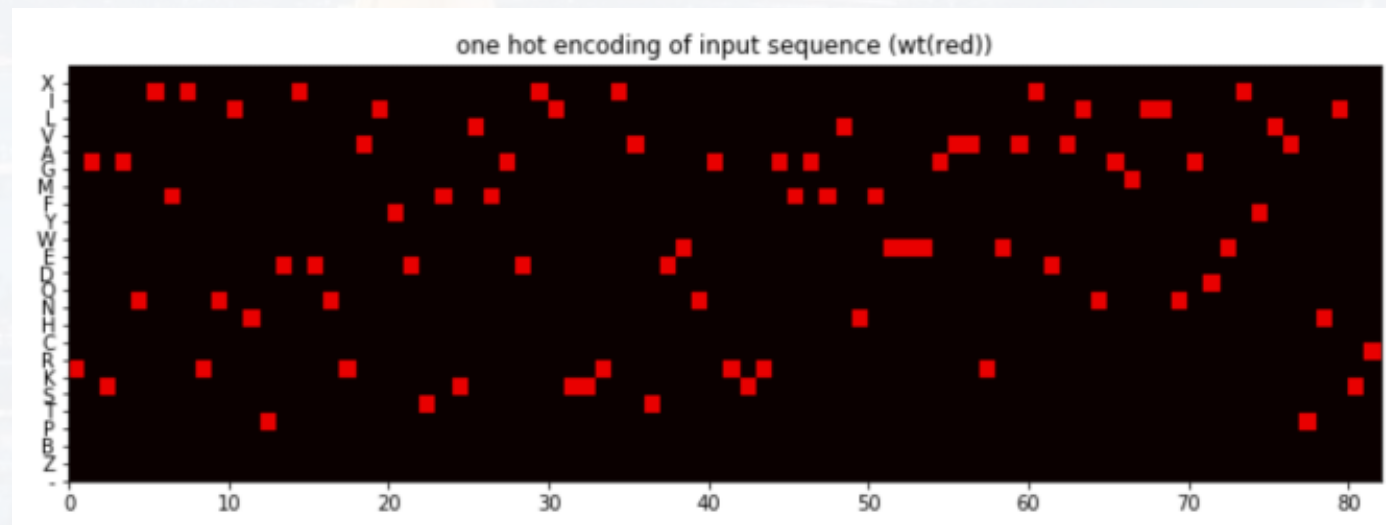




motif finding / sequence analysis

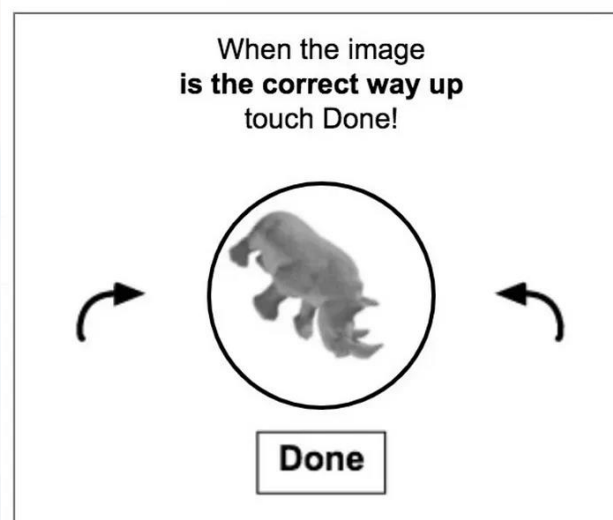
	C	A	G	T	C	T	A
4	1	0	0	0	1	0	0
	0	0	1	0	0	0	0
	0	0	0	1	0	1	0
	0	1	0	0	0	0	1
	Sequence Length						

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



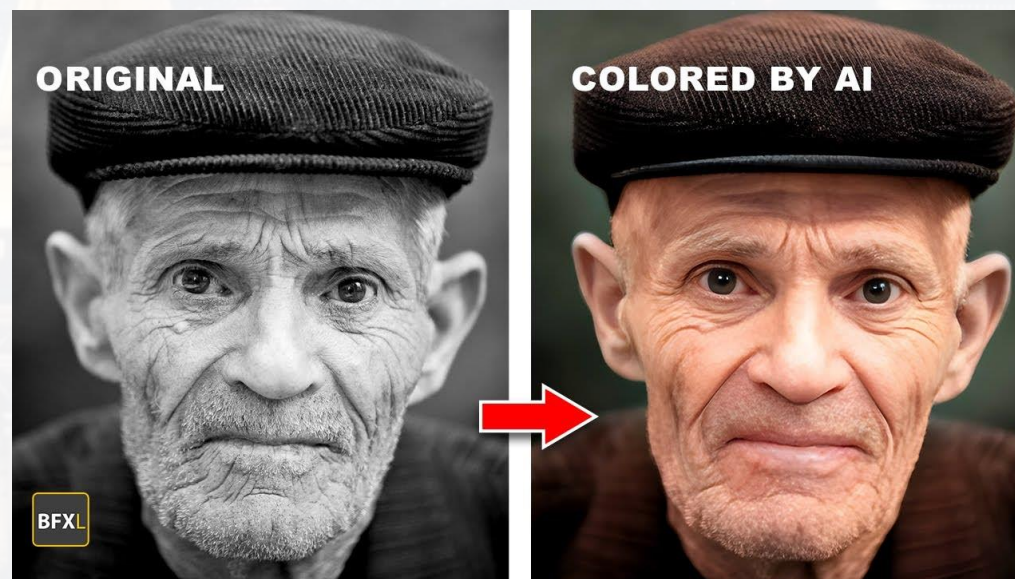


regression:



turning images the right way

part of GenAI:



source: TopviewAI



Outline

- The Problem
- **What is Convolution**
- The CNN Architectures
- Data Preparation & Training
- A Simple Example



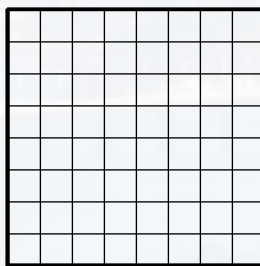
- goal:
- maintaining the spatial information
 - learning which features are important

- convolution
- training the convolution filter

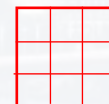
What is convolution?

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

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



image

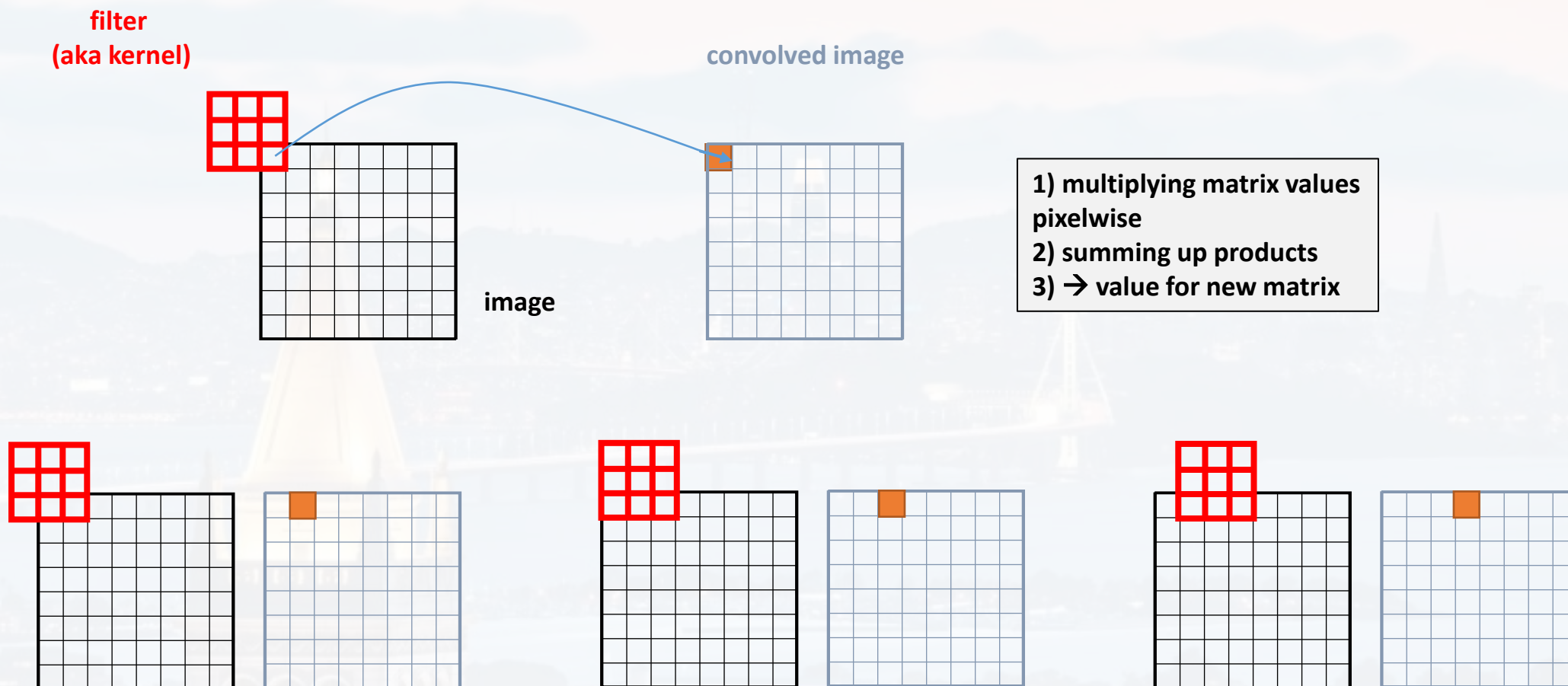


filter
(aka kernel)



What is convolution?

image f and filter g

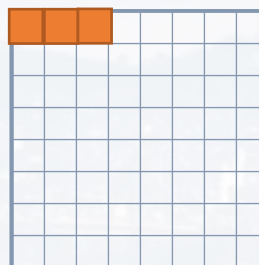
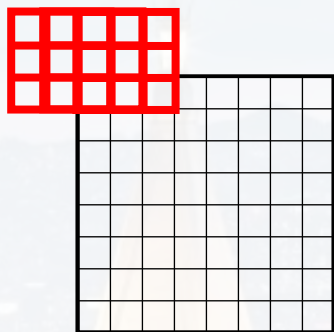




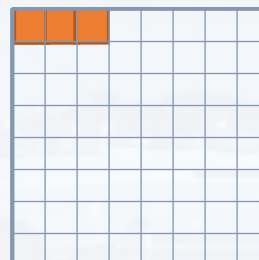
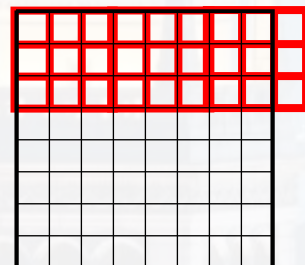
What is convolution?

image f and filter g

different techniques:



padding = 2; stride length = 1



padding = 0; stride length = 3

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

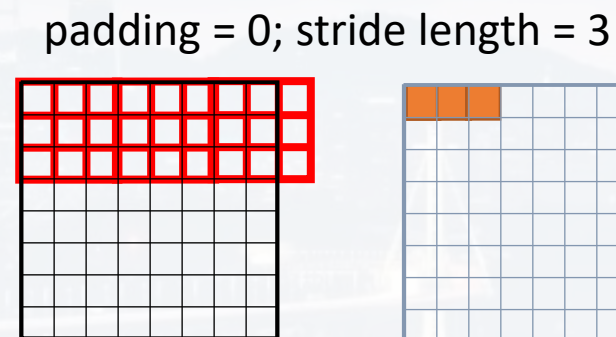
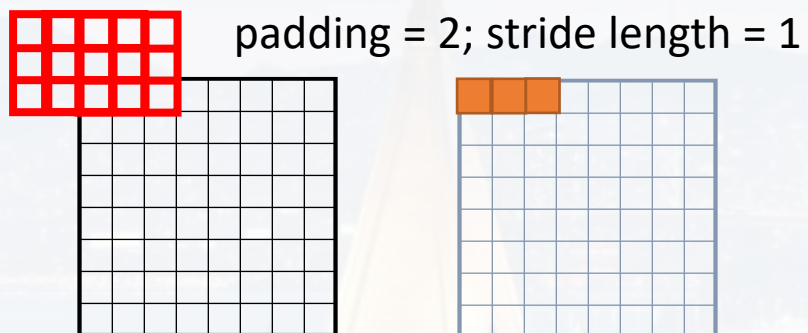


What is convolution?

image f and filter g

different techniques:

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



the resulting image has the following size (N is the number of rows/columns):

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

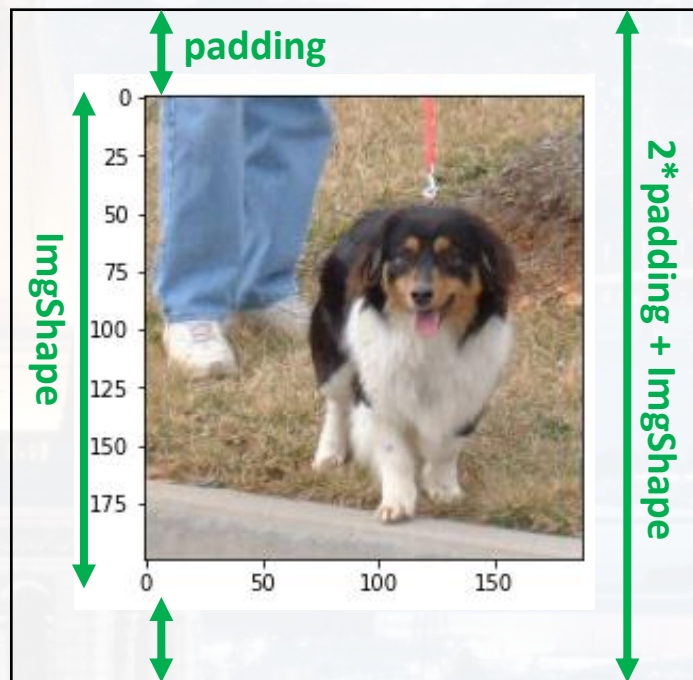


What is convolution?

image f and filter g

the resulting image has the following size (N is the number of rows/columns):

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

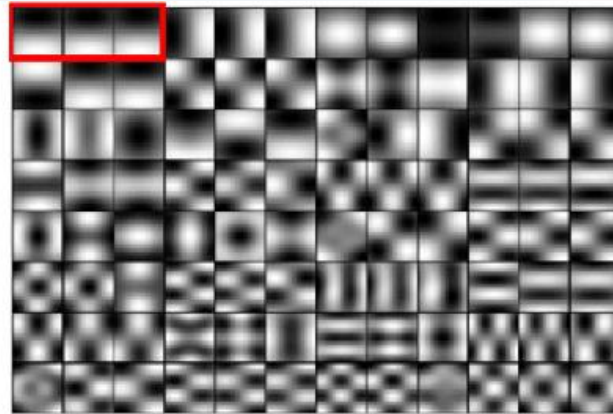




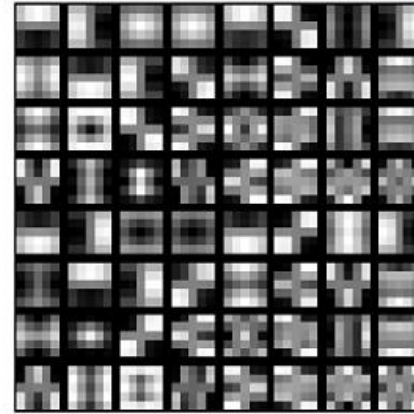
What is convolution?

image f and filter g

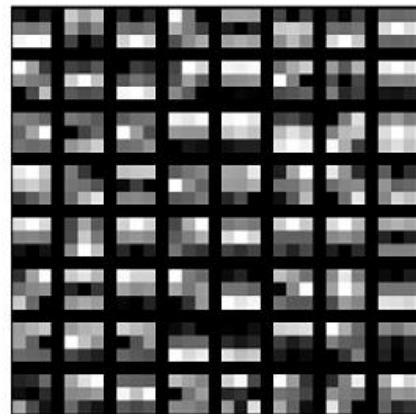
filters:



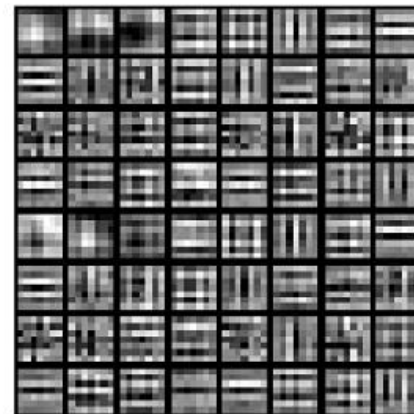
(a)



(b)



(c)



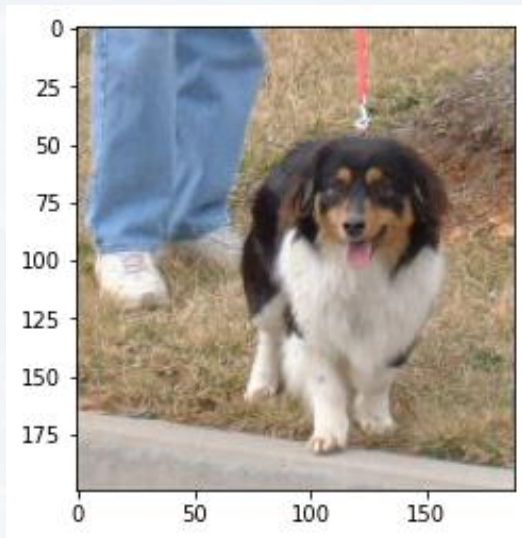
(d)

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

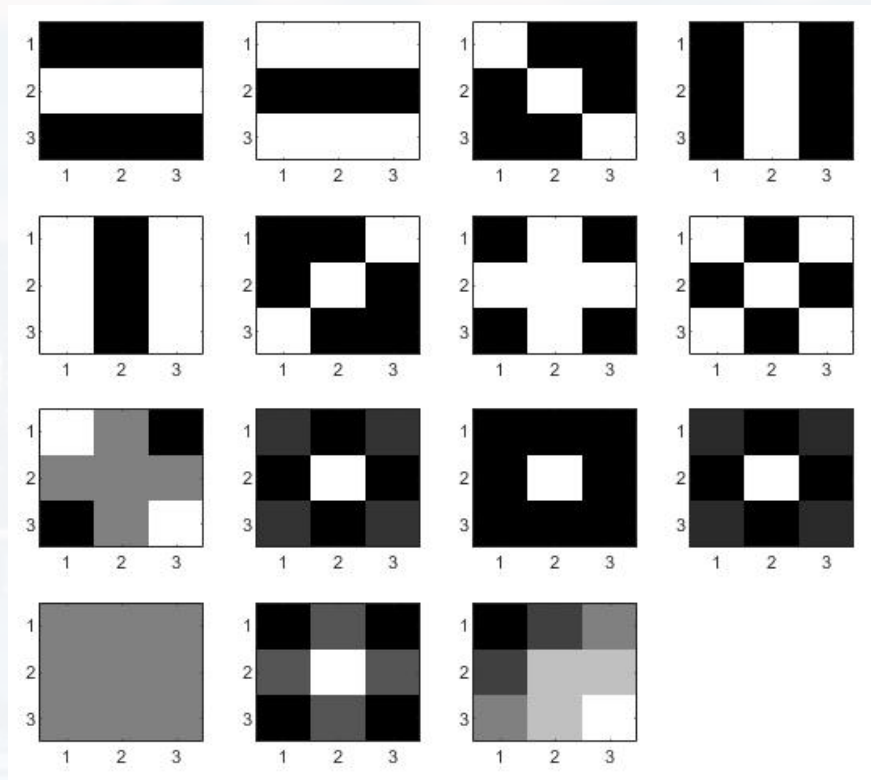


see `Convolution.ipynb` for the impact of different convolution filter on the image

image



filter

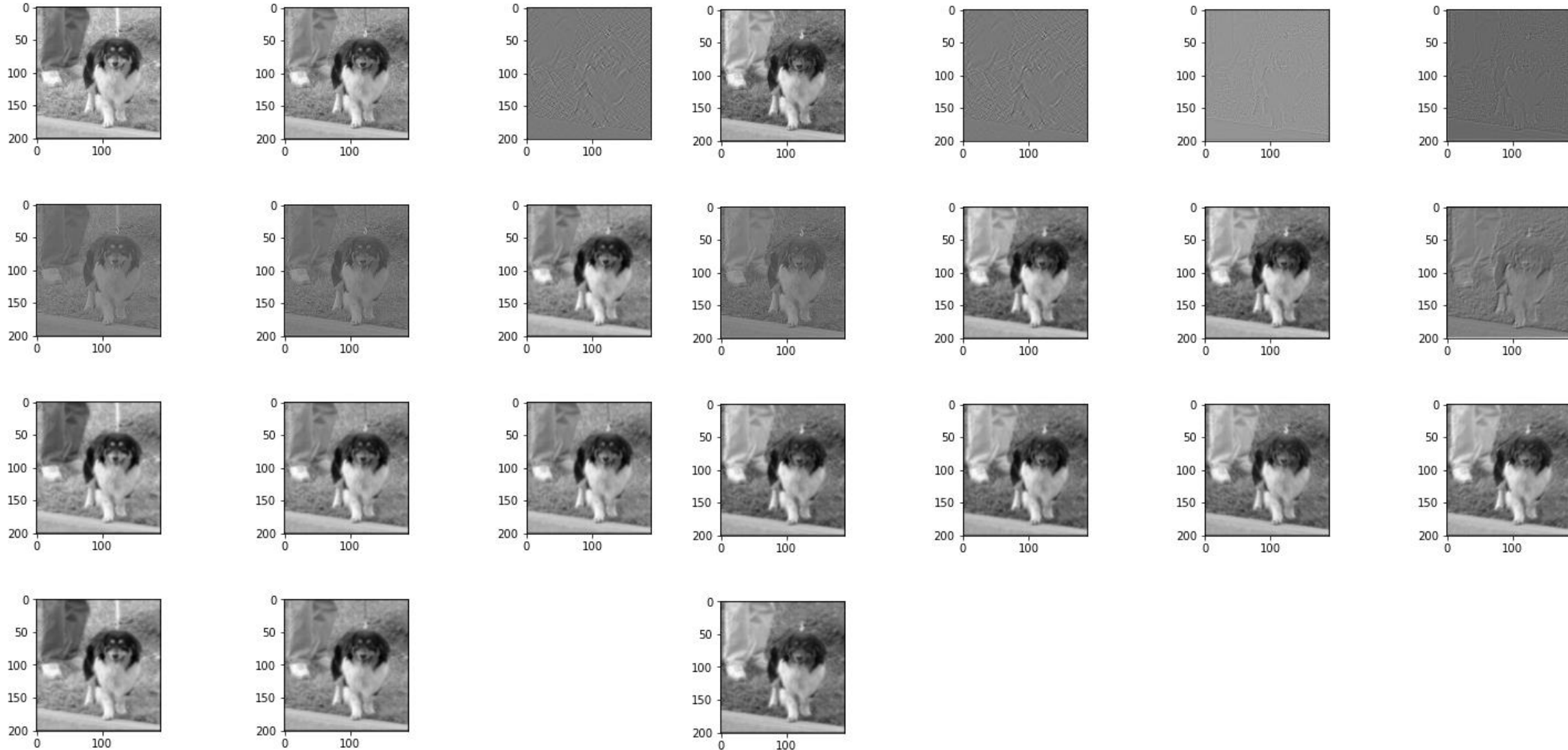




see `Convolution.ipynb` for the impact of different convolution filter on the image

after convolut

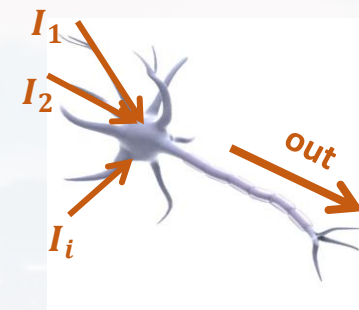
after convolution of channel 3





- 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

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



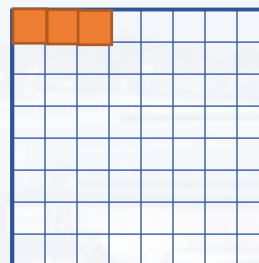
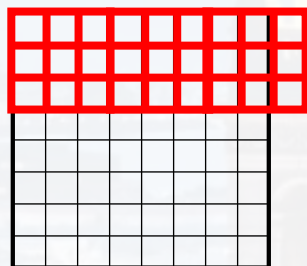
$$net = \sum_i I_i \cdot w_i + b$$

inputs are pixel values

kernel weights

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

can be interpreted as a neuron!



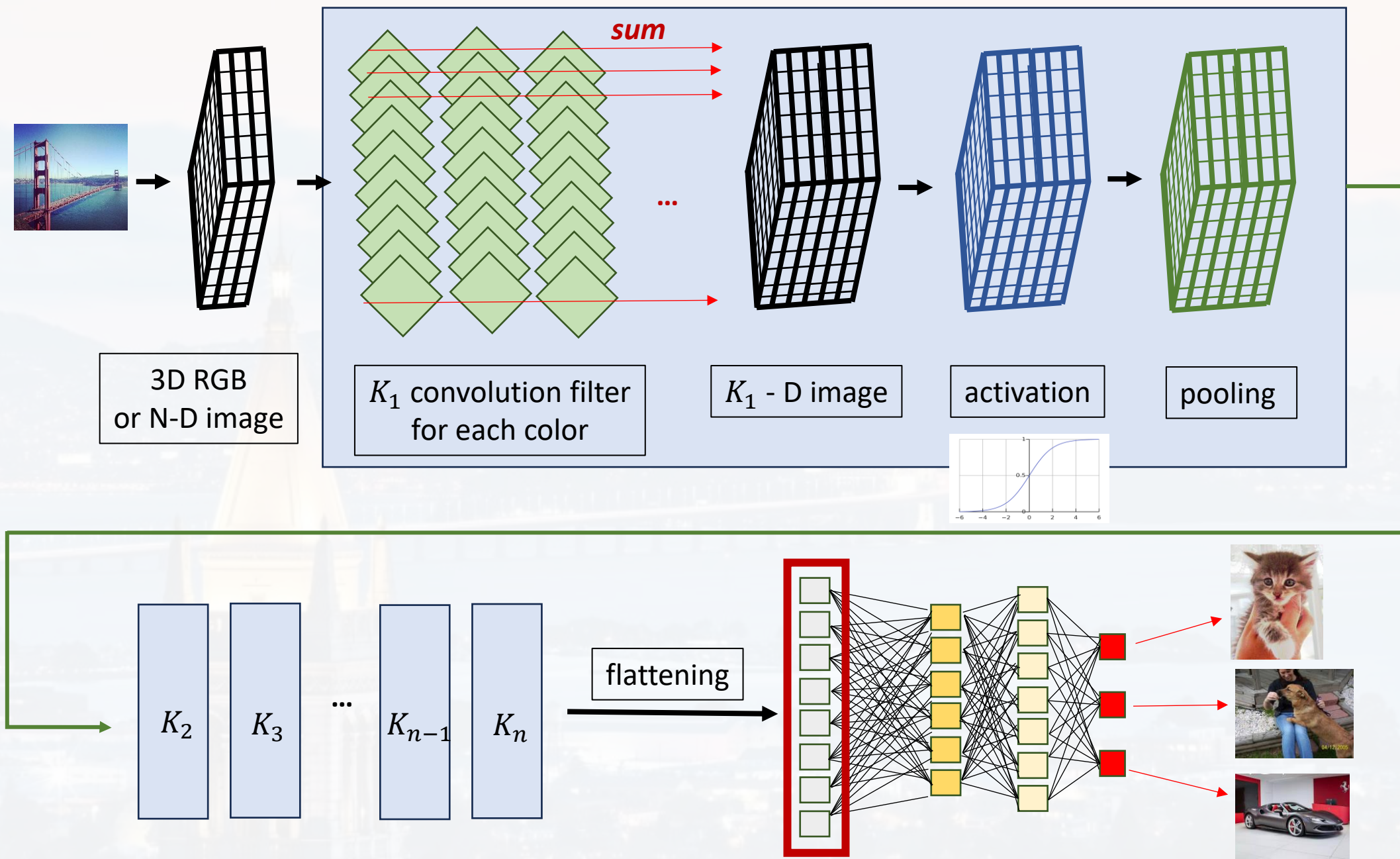


Outline

- The Problem
- What is Convolution
- **The CNN Architectures**
- Data Preparation & Training
- A Simple Example



classification





classification pooling:

there are three different main pooling methods

→ **average pool:**

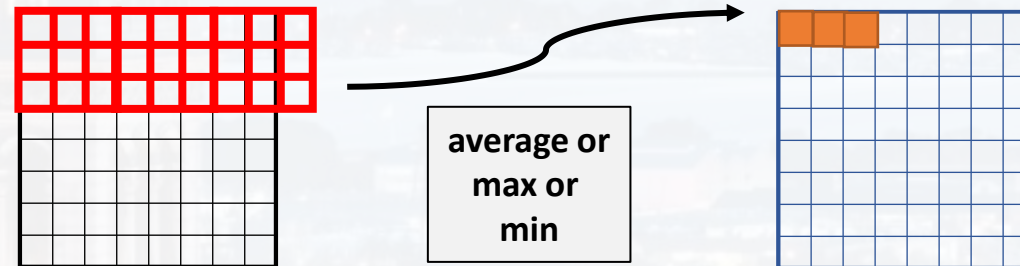
blurs the image, reduces edges
(not what we want here)

→ **max pool:**

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

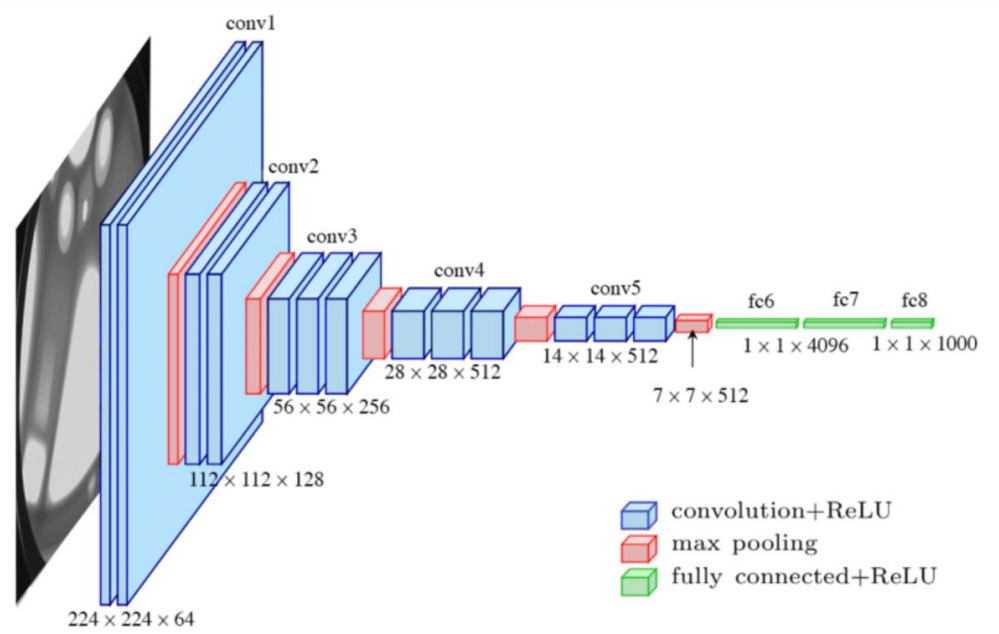
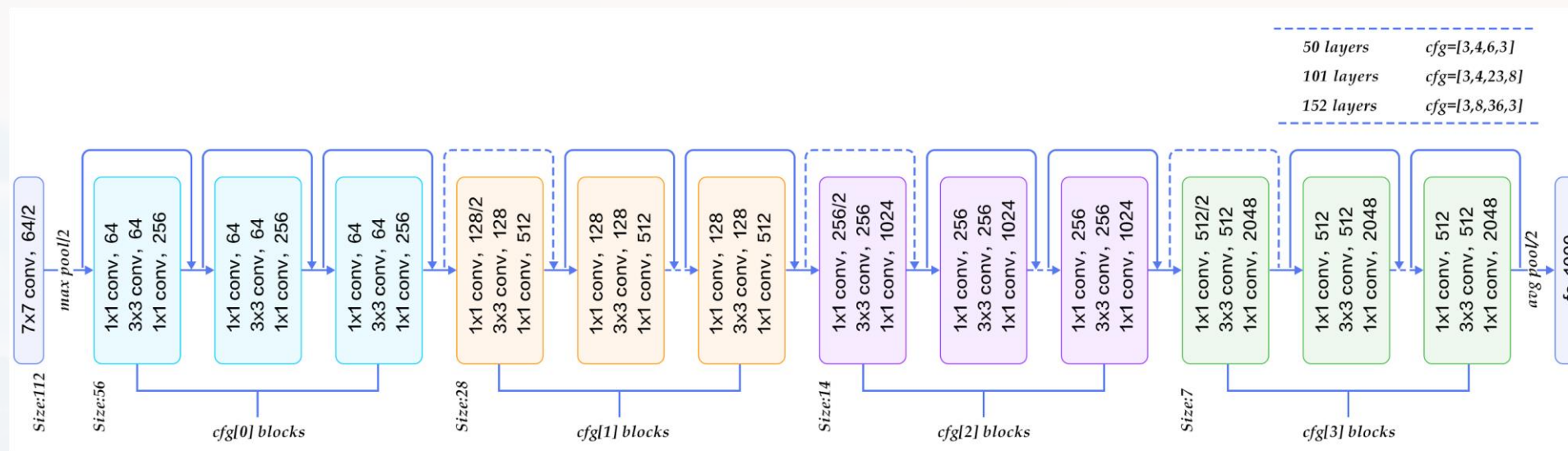
→ **min pool:**

does the opposite of max pool





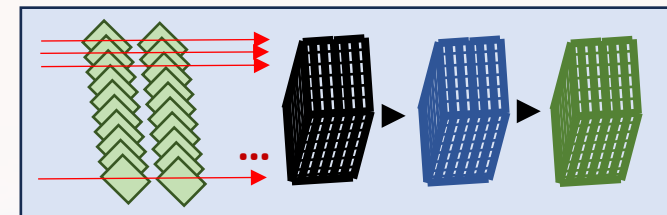
classification



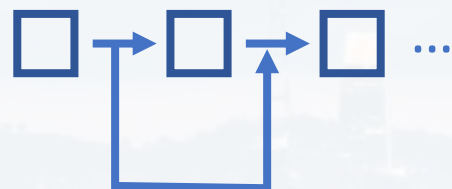


classification

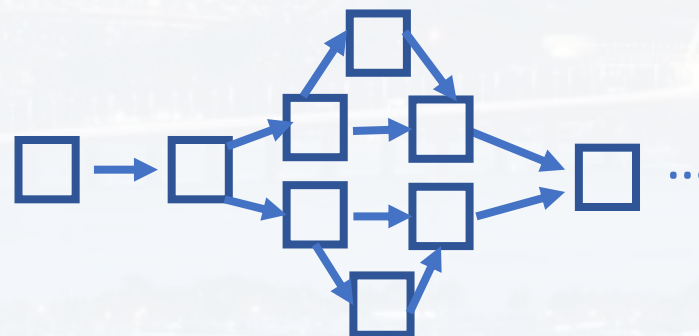
sequential CNNs



ResidualNet



Inception Net



many others...



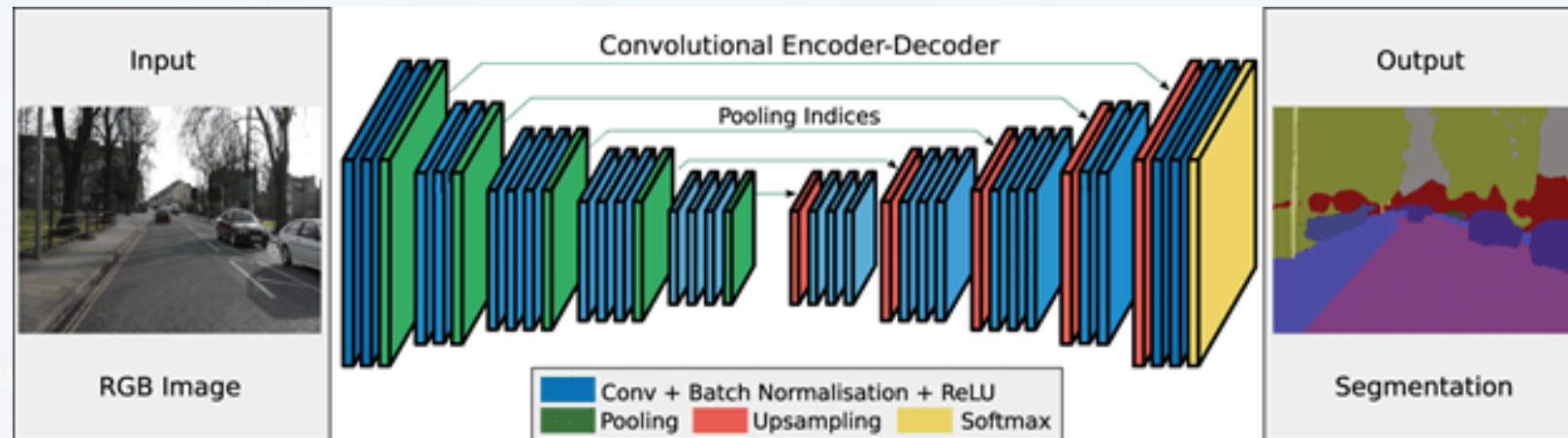
classification

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

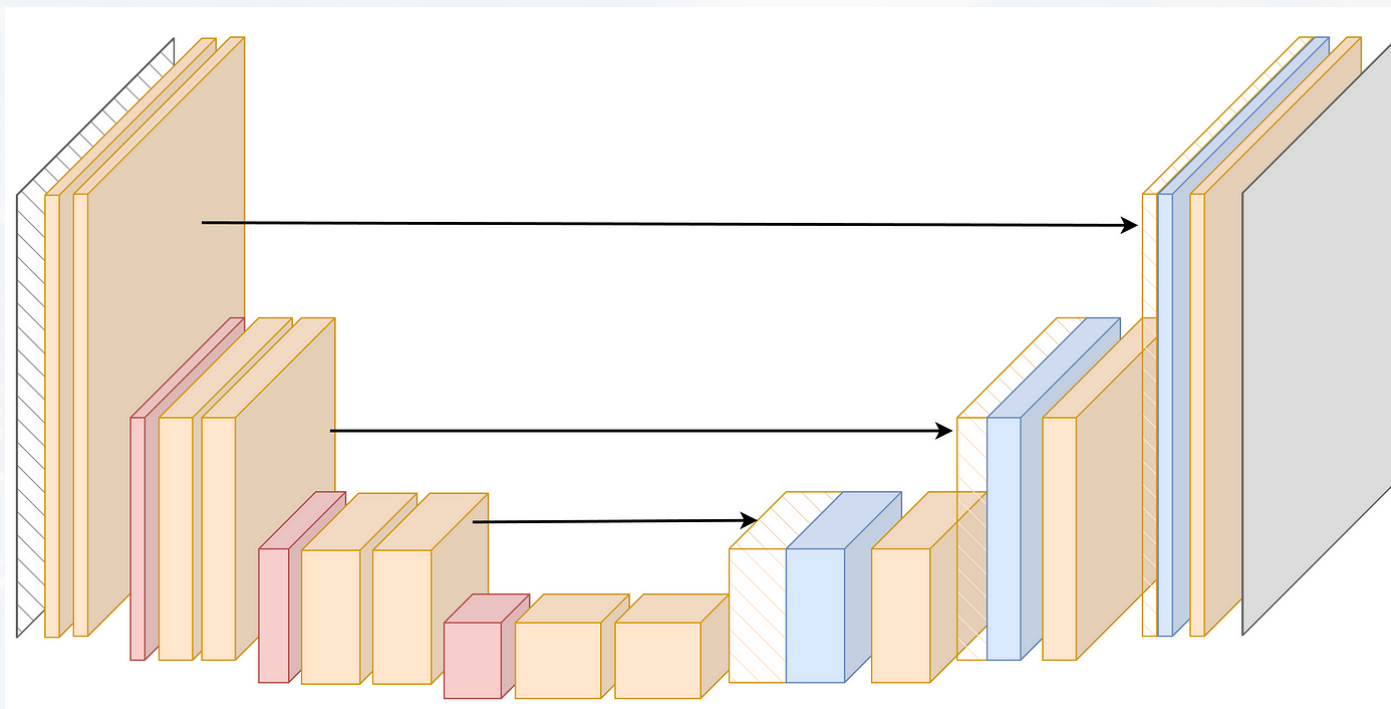


Vijay Badrinarayanan et. al 2017 "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"



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'



Outline

- The Problem
- What is Convolution
- The CNN Architectures
- **Data Preparation & Training**
- A Simple Example



data acquisition

1) classes should be **well balanced**

2) dataset should be **diverse**



example Cryo-EM:

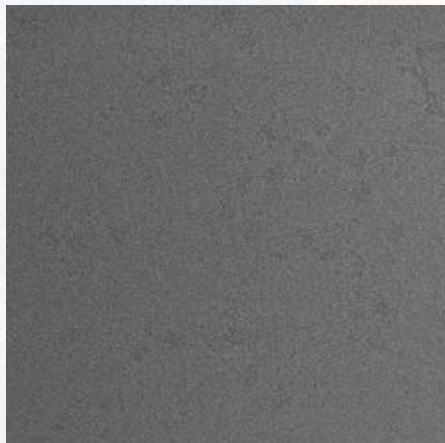
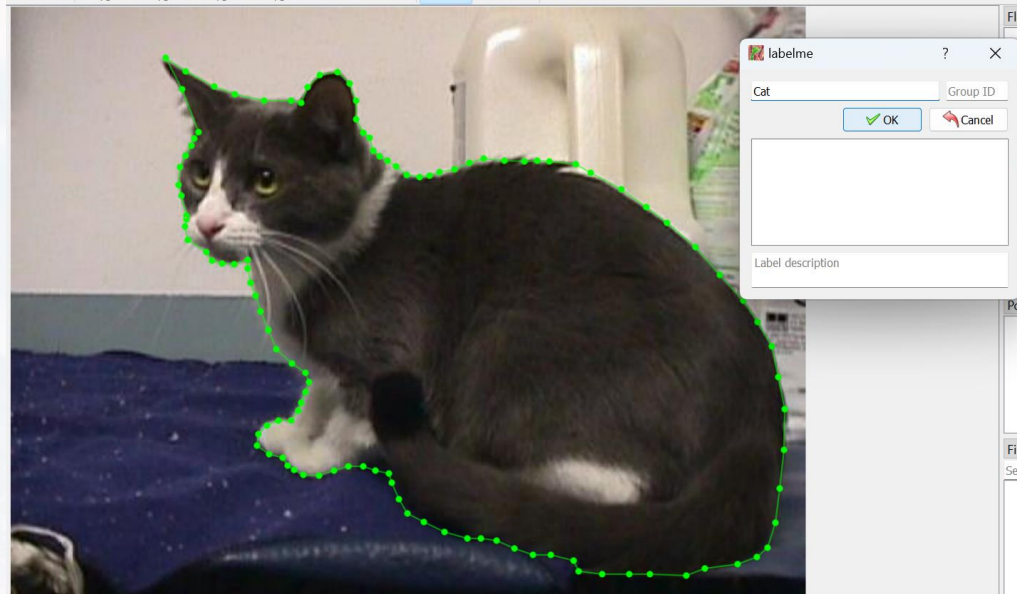
all grids (Cu, Au, ...)
all cameras
all grid manufacturers
all resolutions

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**

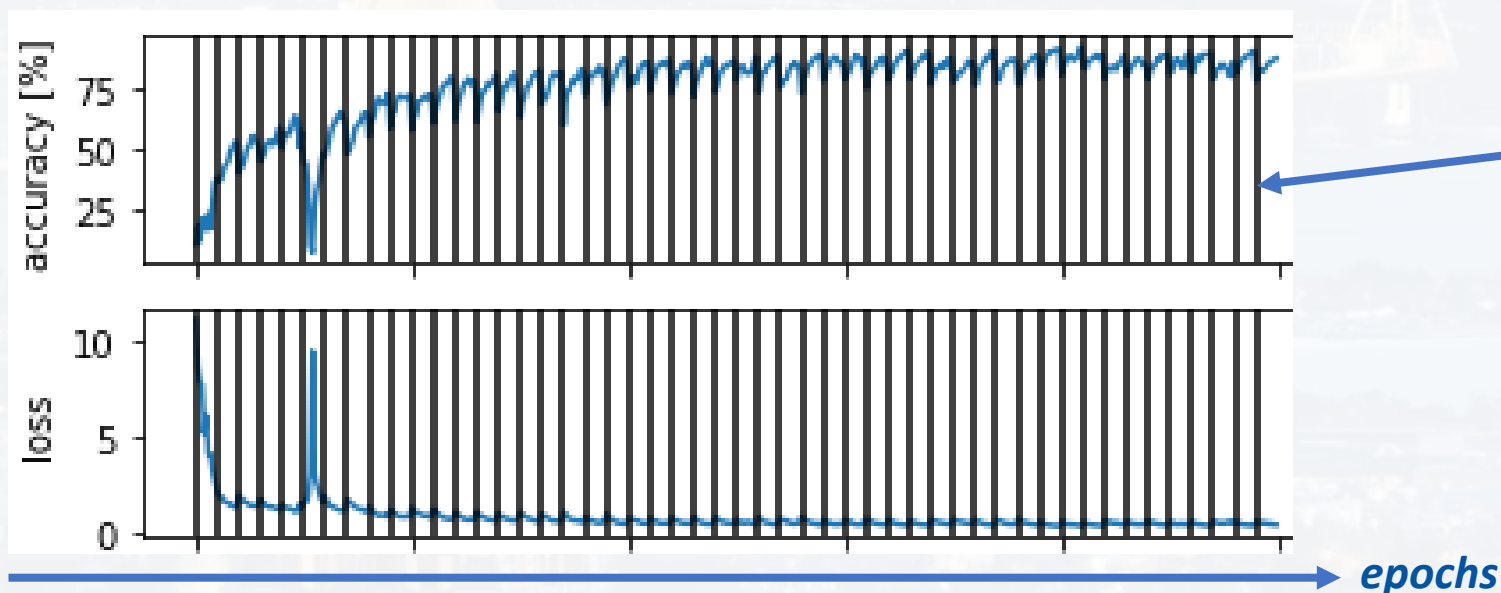
the training set is huge

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

→ the larger the mini batch, the better

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

→ check **training loss vs evaluation loss**





training

check out:

[Training MLP](#)

[Training CNN 2D](#)

[Training CNN 3D](#)



Outline

- The Problem
- What is Convolution
- The CNN Architectures
- Data Preparation & Training
- **A Simple Example**



constructing, running and testing LeNet → LeNet.ipynb

Model: "my_le_net_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 6)	156
average_pooling2d_2 (AveragePooling2D)	(None, 14, 14, 6)	0
conv2d_4 (Conv2D)	(None, 10, 10, 16)	2416
average_pooling2d_3 (AveragePooling2D)	(None, 5, 5, 16)	0
conv2d_5 (Conv2D)	(None, 1, 1, 120)	48120
flatten_1 (Flatten)	(None, 120)	0
dense_2 (Dense)	(None, 84)	10164
dense_3 (Dense)	(None, 10)	850

=====
Total params: 61706 (241.04 KB)
Trainable params: 61706 (241.04 KB)
Non-trainable params: 0 (0.00 Byte)

input = (N images, 28 x 28 pixel, 3 colors)
output = (N images, 28 x 28 pixel, 6 Conv filter)

6 Conv filter * shape (5, 5) = 150
plus 1 bias for each filter → total 156

Each image is represented by a vector of length 120

output layer: one neuron for each class



constructing, running and testing LeNet → LeNet.ipynb

```
running model...
Epoch 1/20
94/94 [=====] - 9s 92ms/step - loss: 1.1172 - accuracy: 0.7160 - val_loss: 0.4217 - val_accuracy: 0.8838
Epoch 2/20
94/94 [=====] - 8s 88ms/step - loss: 0.3577 - accuracy: 0.8990 - val_loss: 0.3096 - val_accuracy: 0.9109
Epoch 3/20
94/94 [=====] - 7s 73ms/step - loss: 0.2876 - accuracy: 0.9163 - val_loss: 0.2606 - val_accuracy: 0.9231
Epoch 4/20
94/94 [=====] - 8s 90ms/step - loss: 0.2444 - accuracy: 0.9287 - val_loss: 0.2238 - val_accuracy: 0.9333
Epoch 5/20
94/94 [=====] - 7s 69ms/step - loss: 0.2113 - accuracy: 0.9376 - val_loss: 0.1944 - val_accuracy: 0.9423
```

epoch: passing the entire dataset through the network

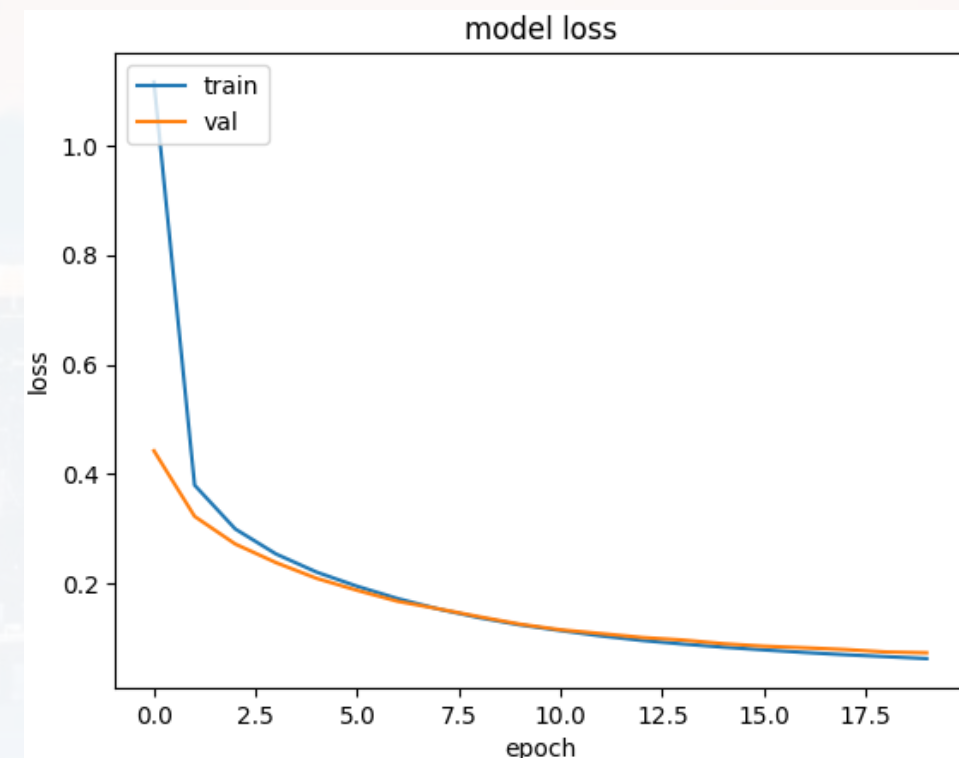
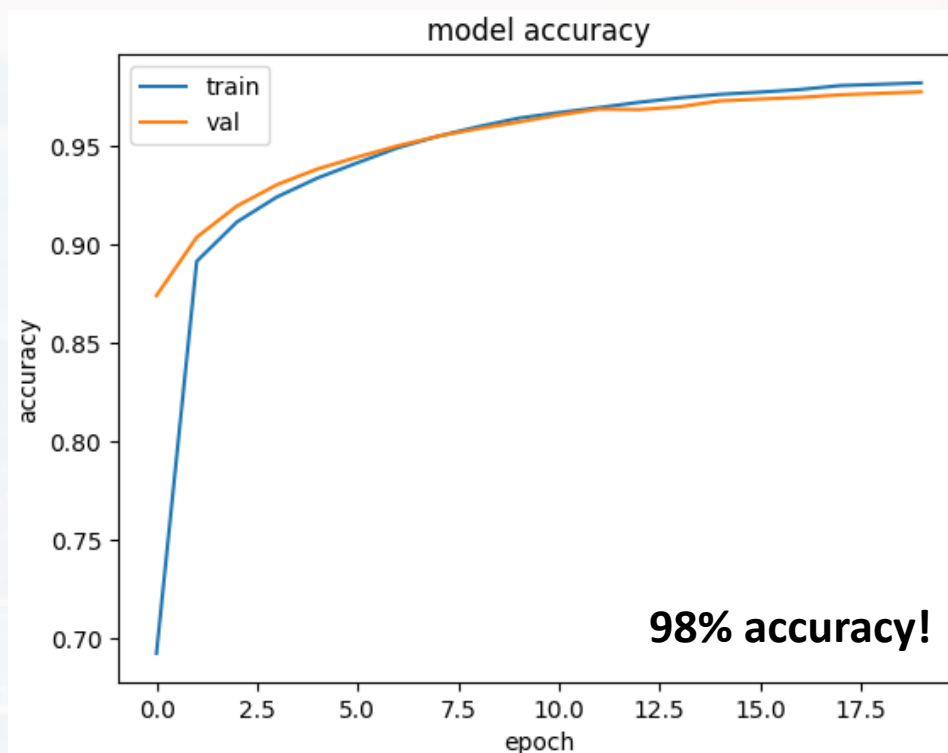
60,000 images / batch size = **512**

= **117** iterations per epoch

= **117 * 80%** for training = **94 iterations per epoch**



constructing, running and testing LeNet → LeNet.ipynb



training loss should \approx validation loss

if validation loss \gg training loss → overfitting

- too many parameter
- too few images in batch
- too specific/unique batch)



constructing, running and testing LeNet → LeNet.ipynb

7, P = 1.0 7	7, P = 0.99 7	7, P = 1.0 7	5 3	8, P = 1.0 8	4, P = 1.0 4	0, P = 1.0 0
6, P = 1.0 6	4, P = 1.0 4	2, P = 0.85 0	3 7	0, P = 1.0 0	5, P = 0.99 5	3, P = 1.0 3
5, P = 1.0 5	5, P = 0.99 5	2, P = 0.98 2	1, P = 1.0 1	5, P = 0.99 5	8, P = 0.98 8	7, P = 1.0 7
6, P = 0.99 6	8, P = 1.0 8	7, P = 0.99 7	7, P = 1.0 7	1, P = 1.0 1	7, P = 0.74 7	8, P = 1.0 8
3, P = 0.97 3	6, P = 0.93 6	7, P = 0.99 7	1, P = 1.0 1	8, P = 0.91 8	3, P = 1.0 3	9, P = 1.0 9
3, P = 1.0 3	6, P = 1.0 6	4, P = 1.0 4	5, P = 1.0 5	5, P = 1.0 5	7, P = 1.0 7	9 7
4, P = 0.99 4	0, P = 1.0 0	1, P = 1.0 1	5, P = 0.99 5	5, P = 1.0 5	5, P = 1.0 5	7, P = 0.99 7



Thank you for your attention!

