Deep Learning: Session 08

Convolutional Nets

Outline

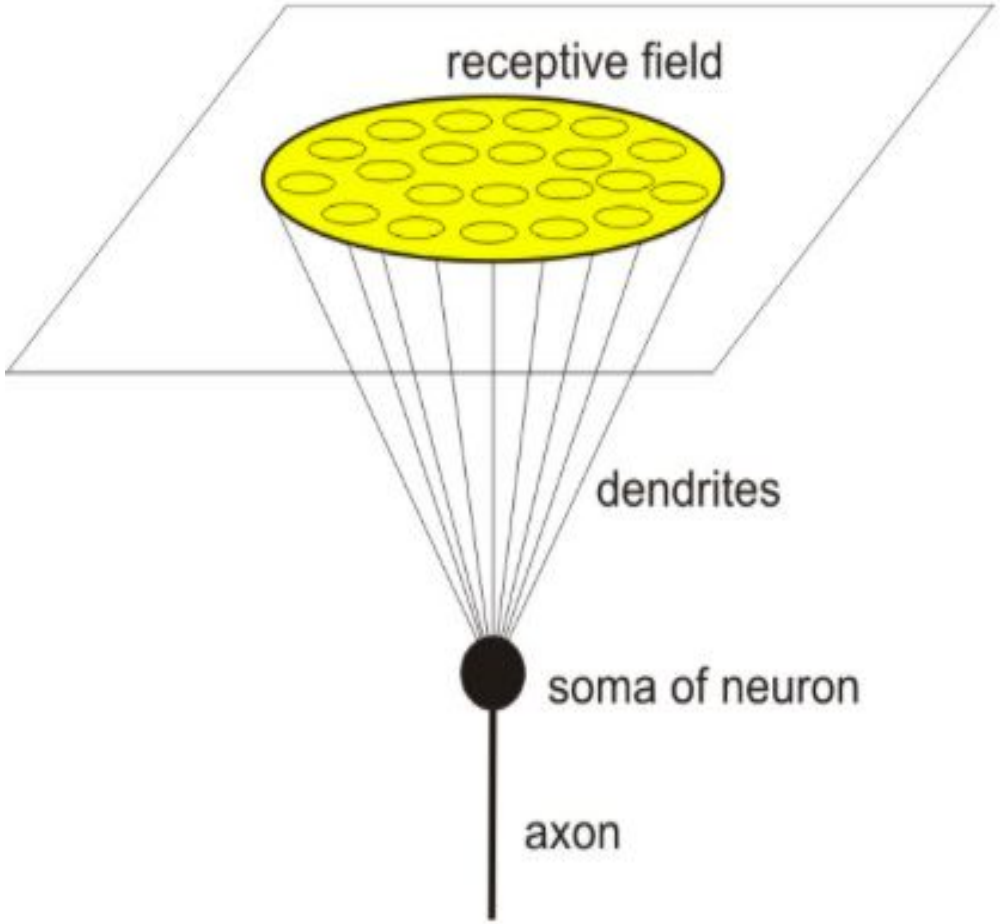
1. Recap
2. Famous CNNs
3. Data Augmentation
4. Transfer Learning
5. Applications

ANNs for images?

A single hidden layer with 1024 units:

- 200 x 200 image -> ~ 40M weights
- 1000 x 1000 image -> ~ 130M weights
- 4000 x 4000 image -> ~ freaking too many !

How our brain sees an image



- Visual neurons have a **receptive field**
- The receptive field is the specific region of the retina in which something could fire the neuron.
- Some visual neurons have large receptive fields and this has an important effect: **location invariance**
- Location invariance: if some pattern is able to fire the neuron, the location of this pattern does not matter as long as it is within the receptor field
- … in other words, this visual neuron is able to find a pattern no matter how is located in the receptive field.

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	

The Convolution Operator

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 1

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 1

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 2

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 2

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

Padding is the number of rows and columns of zeros that are added to the input image before the convolutional filter is applied

P = 1

0	**0**	**0**	**0**	**0**	**0**
0	5	2	3	4	**0**
0	6	7	8	5	**0**
0	4	5	4	5	**0**
0	1	0	2	3	**0**
0	**0**	**0**	**0**	**0**	**0**

Stride & Padding

Padding is the number of rows and columns of zeros that are added to the input image before the convolutional filter is applied

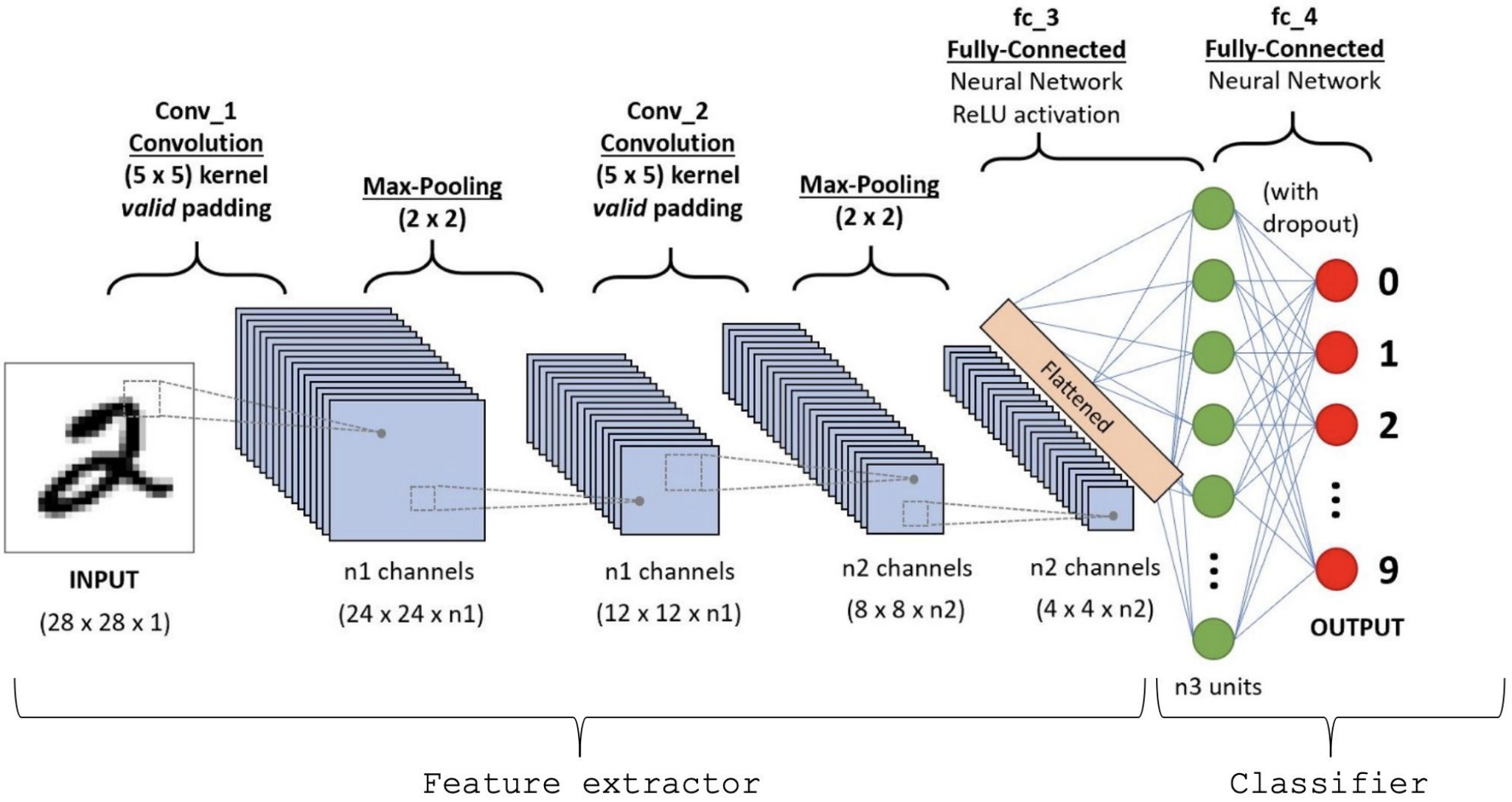
P = 2

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	5	2	3	4	**0**	**0**
0	**0**	6	7	8	5	**0**	**0**
0	**0**	4	5	4	5	**0**	**0**
0	**0**	1	0	2	3	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Stride & Padding

- The padding is a hyperparameter.
- Padding can be used to:
  - Control the size of the output feature maps
  - Preserve the spatial dimensions of the input data
  - Prevent the loss of information at the borders of the input image
- Typical padding strategies in common frameworks (keras, torch) are:
  - Valid: Do not use padding
  - Same: Adds enough padding so that the output feature maps keep the same size as the input image.

The architecture: Classifier



Key features of CNNs

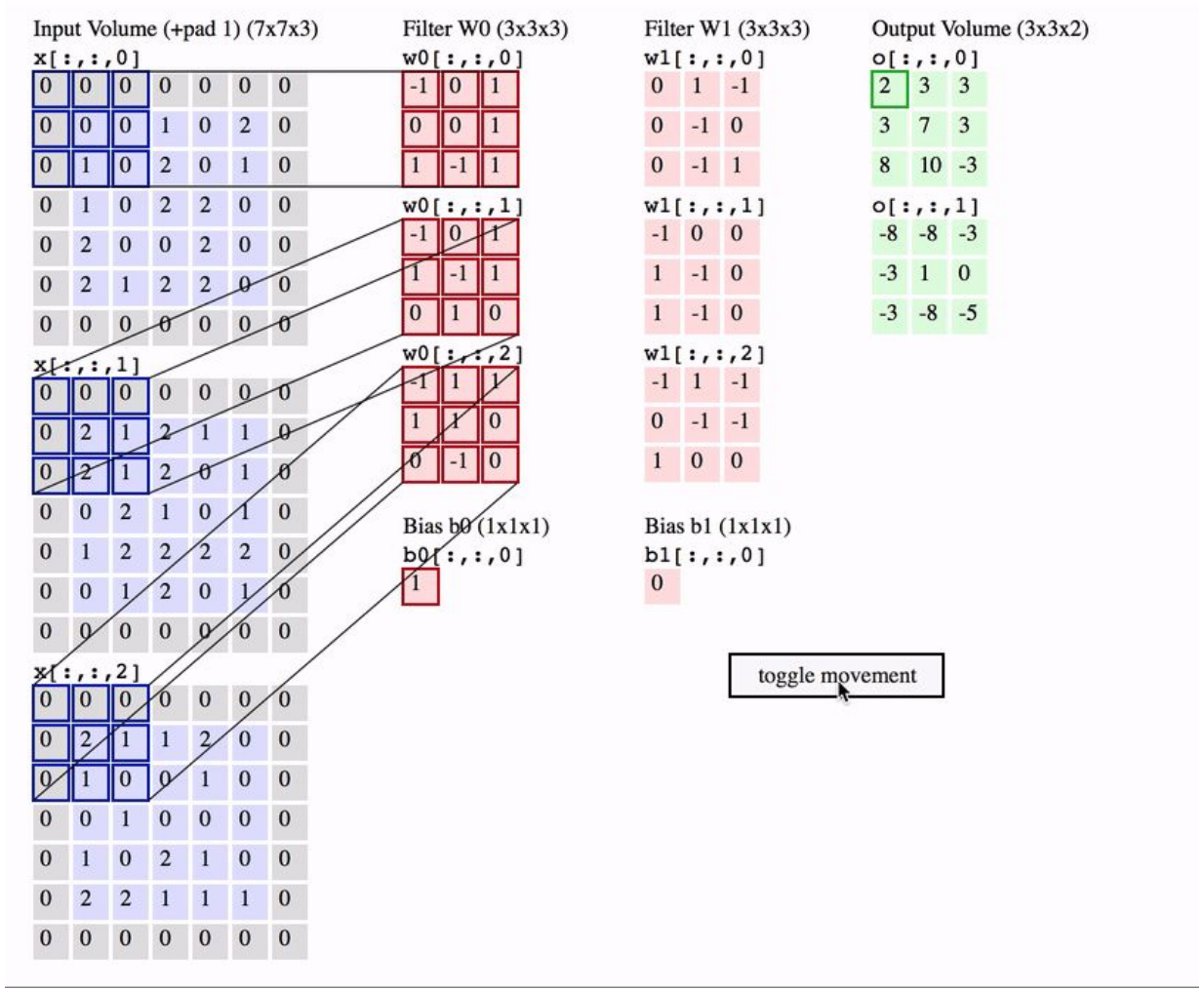
- **Parameter sharing.** Filters are applied to the input image no matter its size, using the same set of trainable params -- those which form the filter itself.
- **Space Invariance.** CNNs can learn translation-invariant features, which means that they can recognize patterns in an image regardless of their position in the image – the filters are moved across the entire image to detect patterns.
- As DNNS, **CNNs can learn hierarchical features**, which means that they can learn increasingly complex patterns by learning simple patterns first and then building on them.
- CNNs can learn features that are **robust to noise**, occlusion, and other image variations, which makes them resistant to overfitting and enables them to generalize well to unseen data. This is thanks to the pooling layers which downsample the input data making the model more tolerant to small variations.

CNNs: More complex! Color images

R

G

B



Output dimensions:

$N_h = (N_h + 2P - F)/S + 1$
$N_w = (N_w + 2P - F)/S + 1$
$N_c = \#filters$

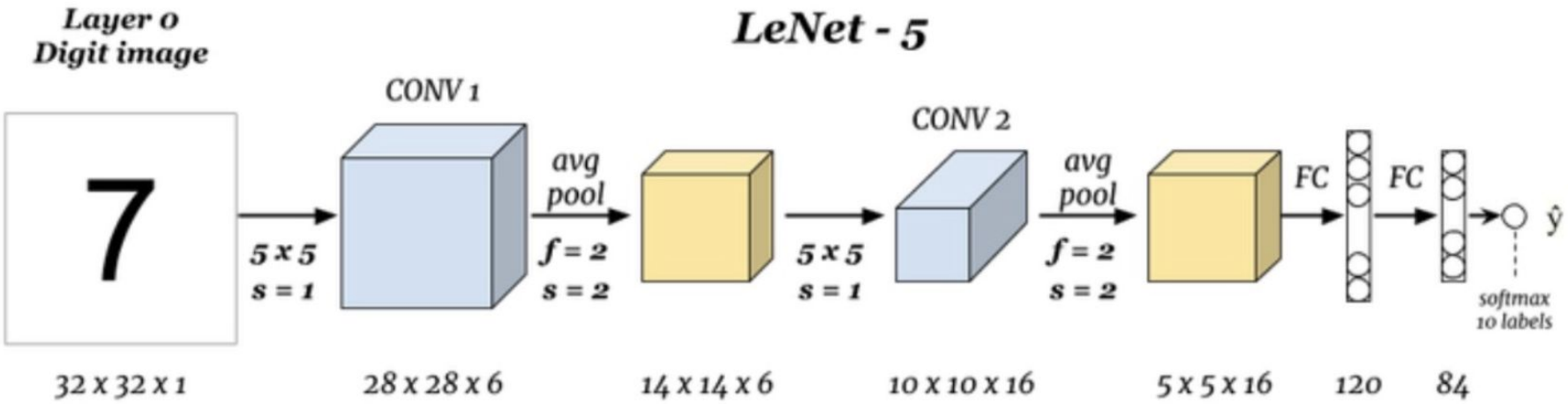
S = Stride □ 2
P = Padding □ 1
F = Size of the filter □ 3

Outline

1. Recap
2. Famous CNNs
3. Data Augmentation
4. Transfer Learning
5. Applications

Famous CNNs

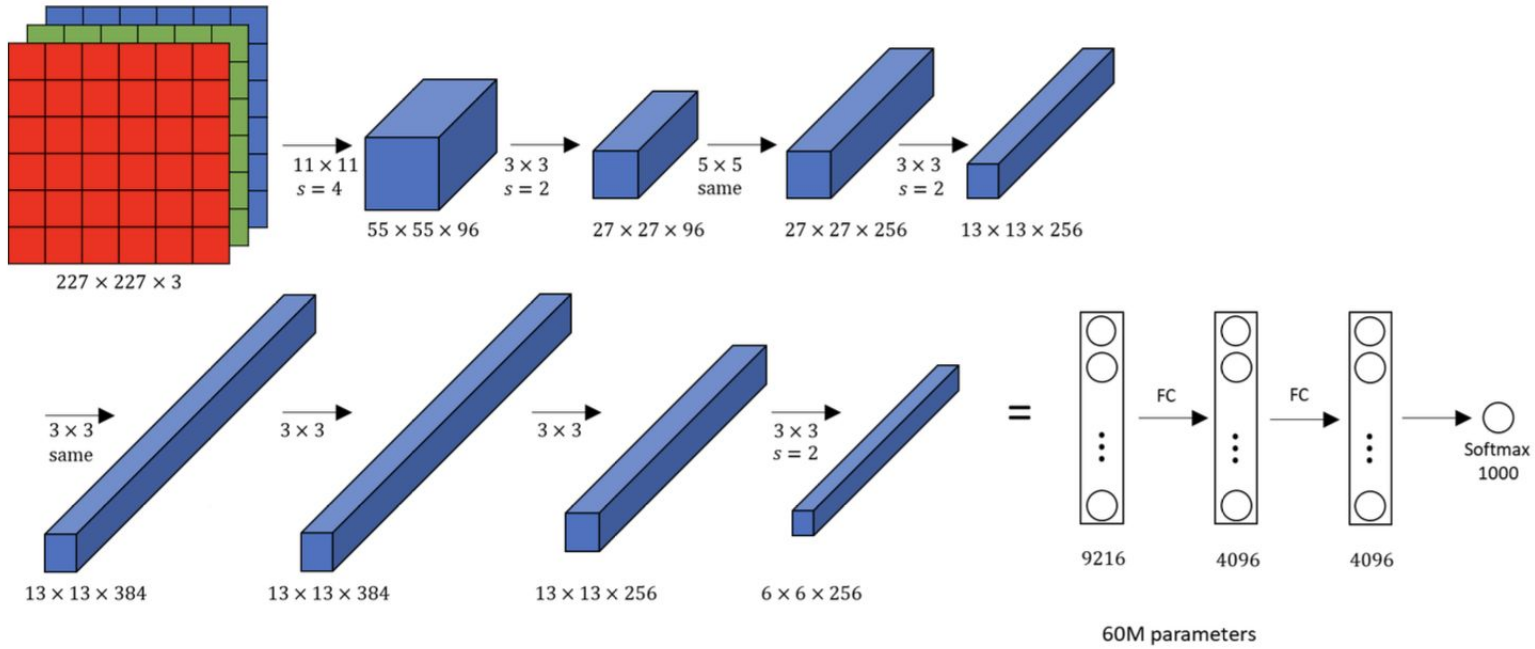
- **LeNet - 5 (1998)**
  - ~60k trainable params.
  - Trained on and for MNIST handwritten recognition problem.



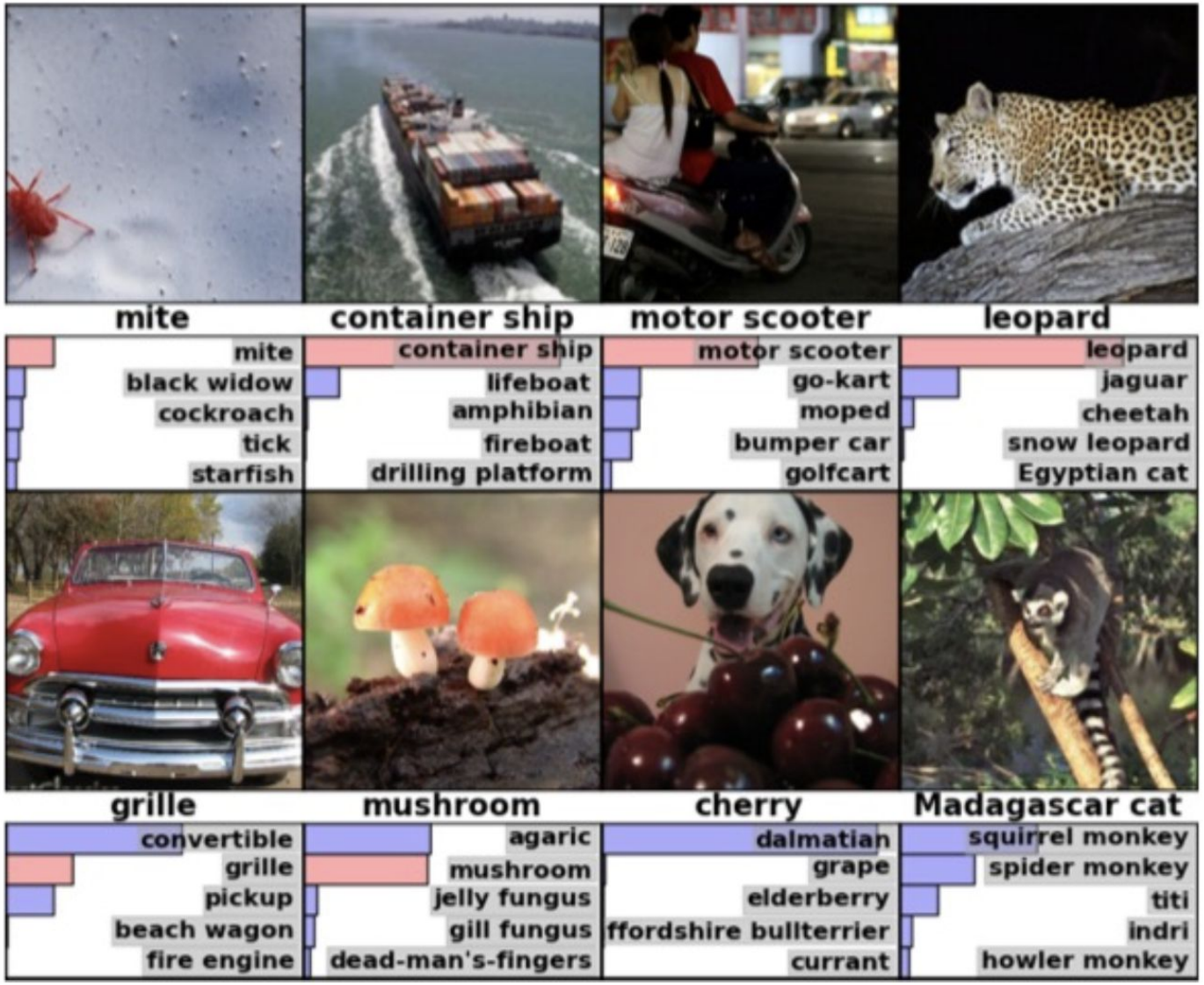
Famous CNNs

Famous CNNs

- **AlexNet (2012)**
  - ~60M trainable params
  - ILSVRC'13 (ImageNet Large Scale Visual Recognition Challenge)
    - The challenge consists of classifying images in 1000 different classes!
    - ImageNet ~1.2 millions of images in training. 150k test data set.

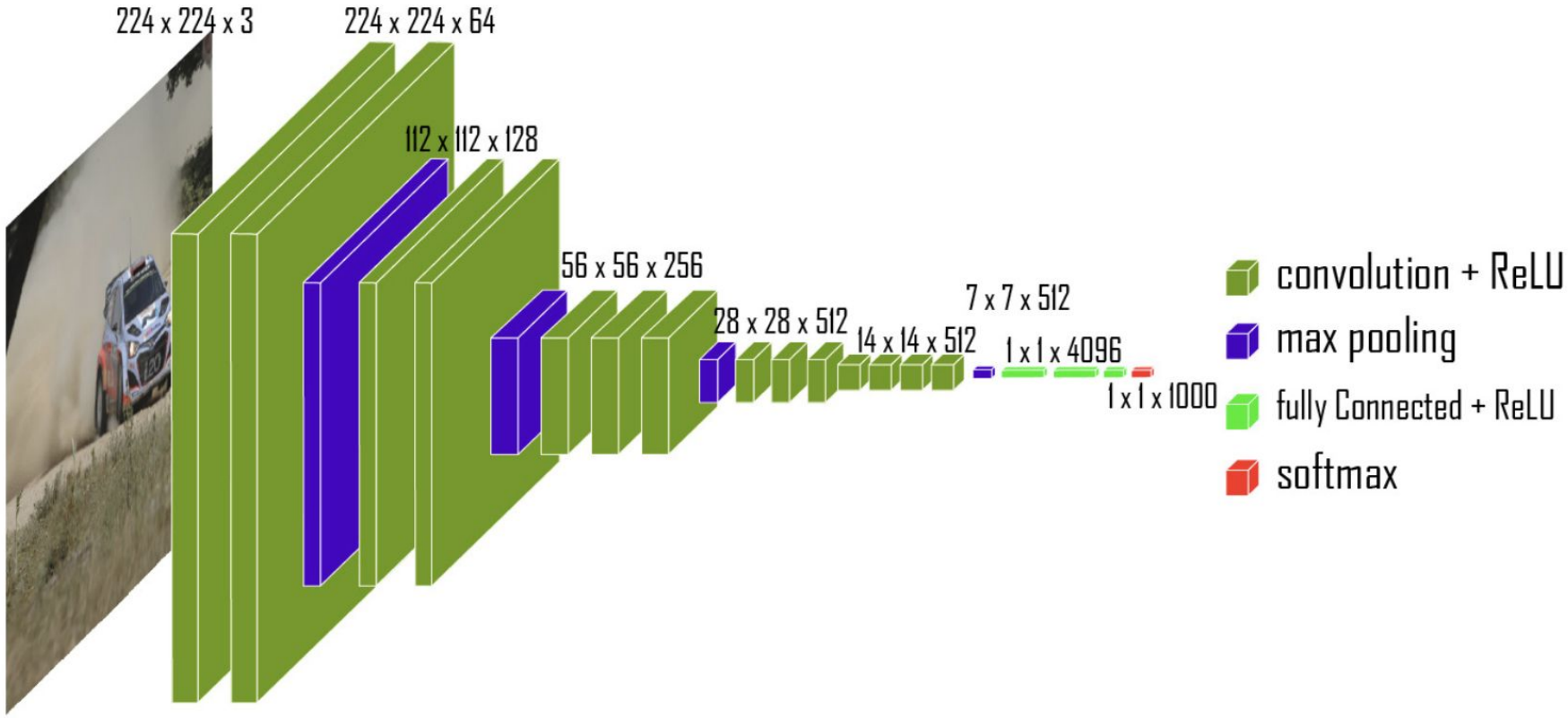


Famous CNNs


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

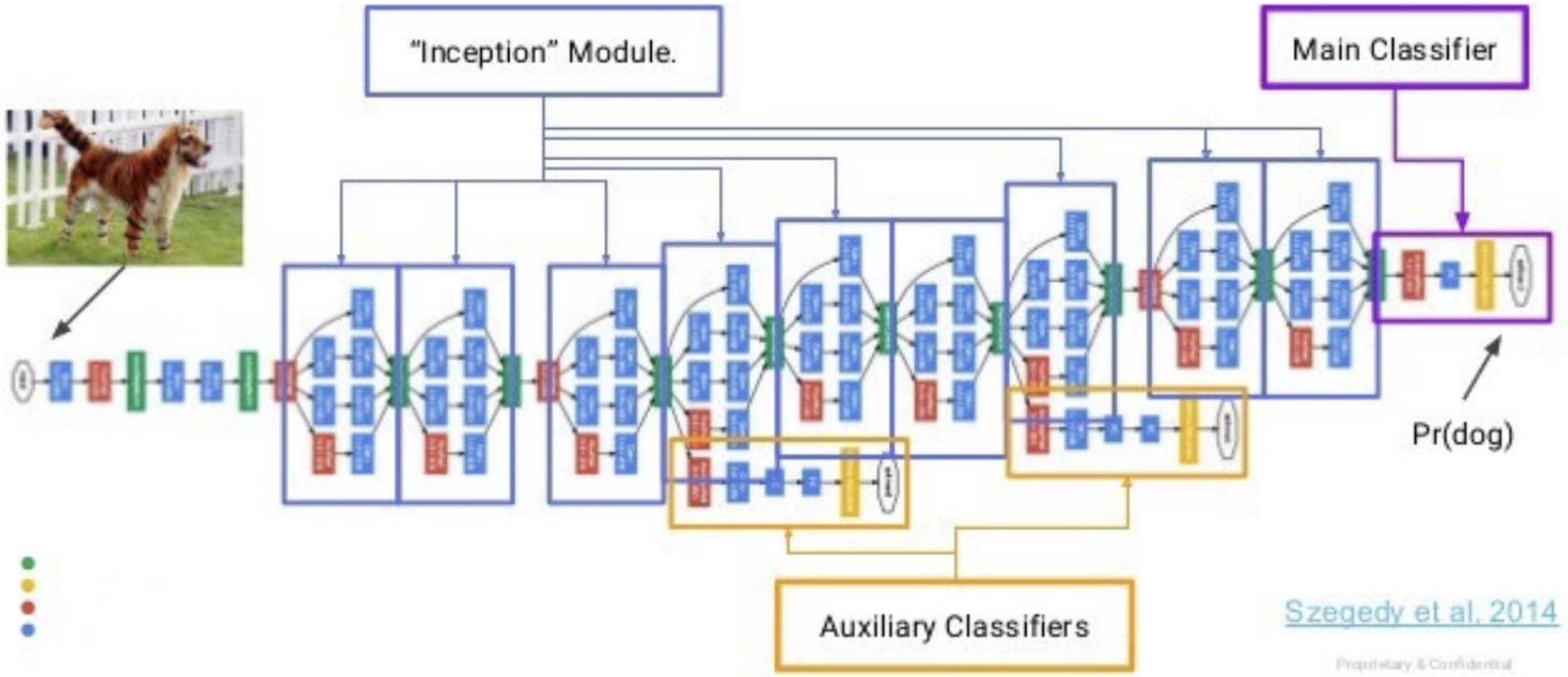
Famous CNNs

- **VGG – 16 (2014)**
  - ~138M trainable params



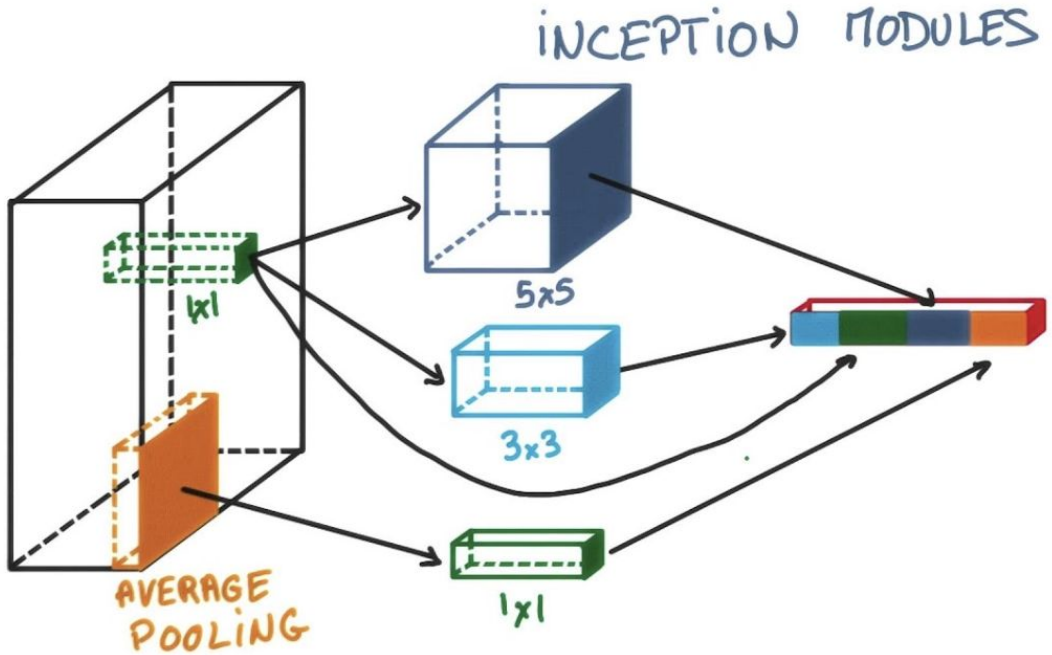
Famous CNNs

- **GoogleLenet – Inception (2014-)**
  - ~6.6M trainable params
  - 22 layers
  - Use blocks of layers as a combination of convolutional, max pooling, average pooling



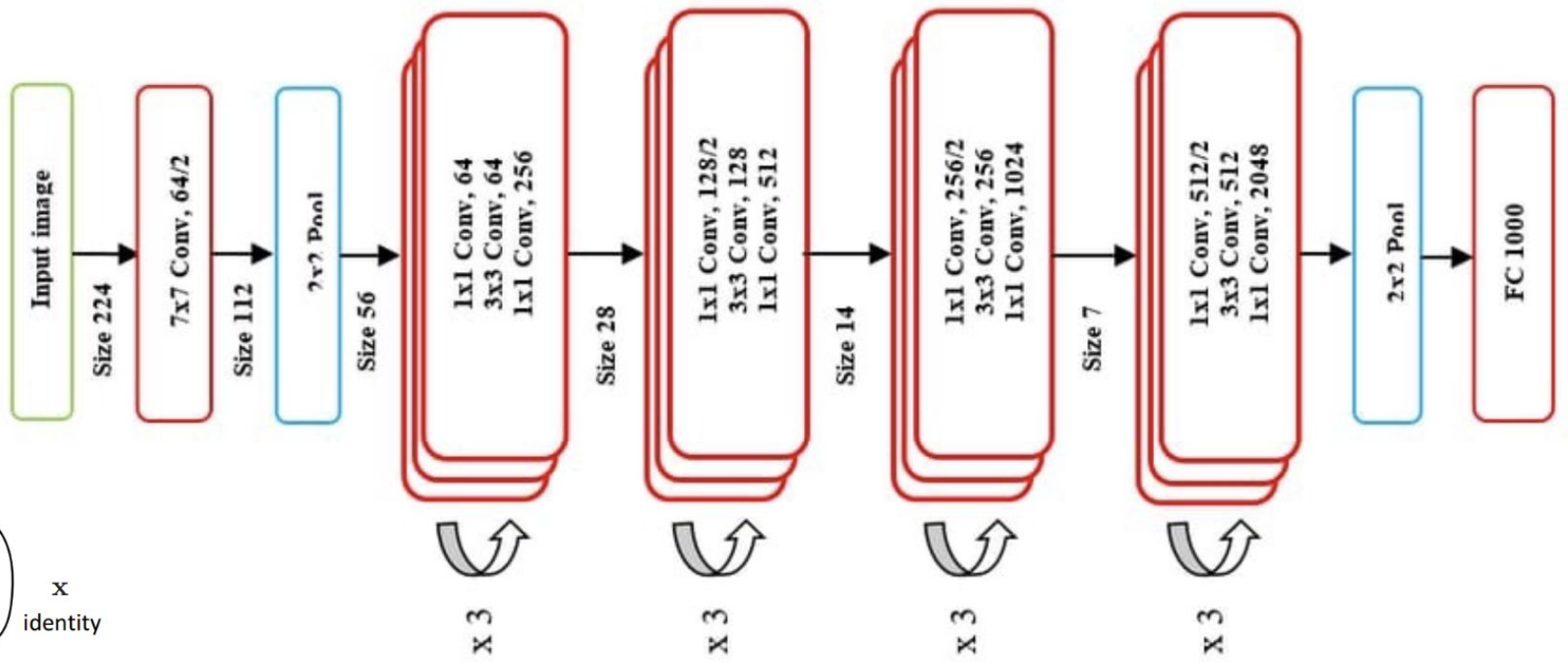
Famous CNNs

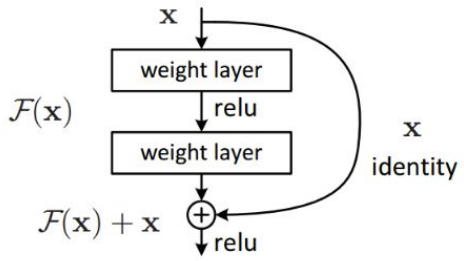
- **GoogleLenet - Inception (2014-)**
  - ~6.6M trainable params
  - 22 layers
  - Inception modules combine different filter sizes and pooling layers



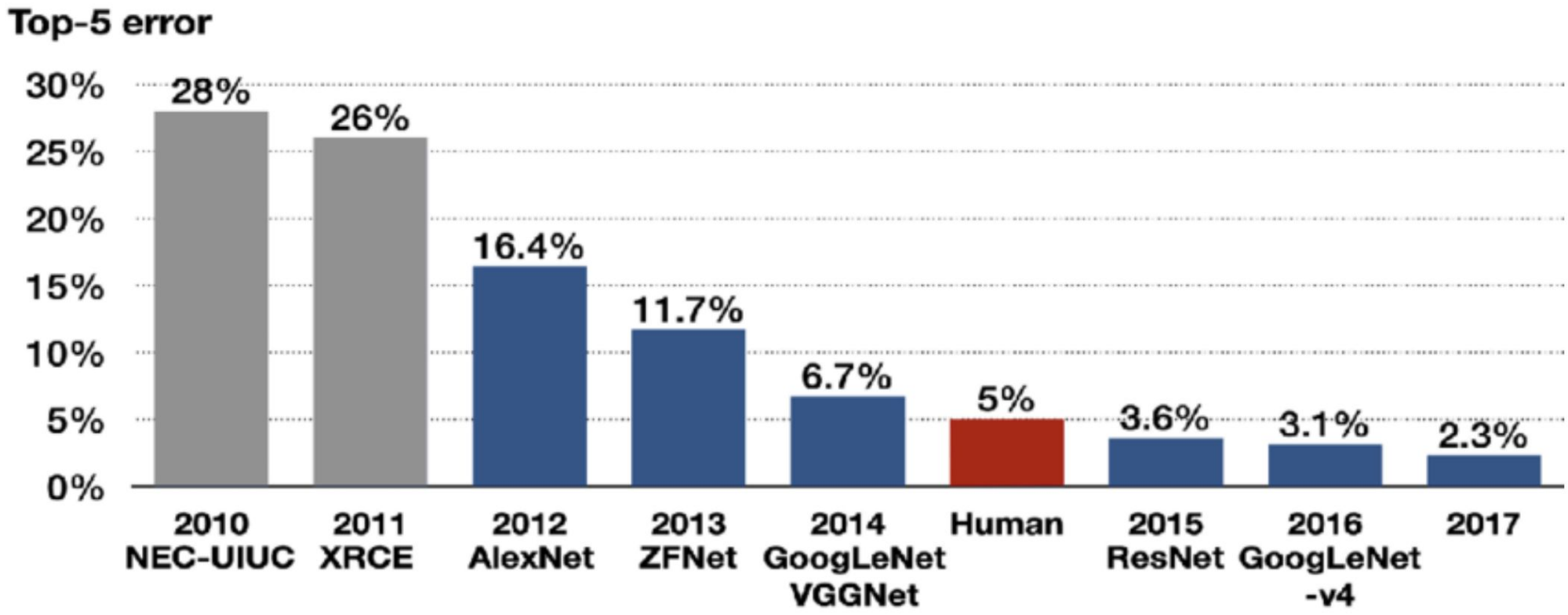
Famous CNNs

- **ResNet - 50 (2015)**
  - ~25.6M trainable params
  - 50 layers!
  - Use skip connections to deal with the vanishing gradient problem

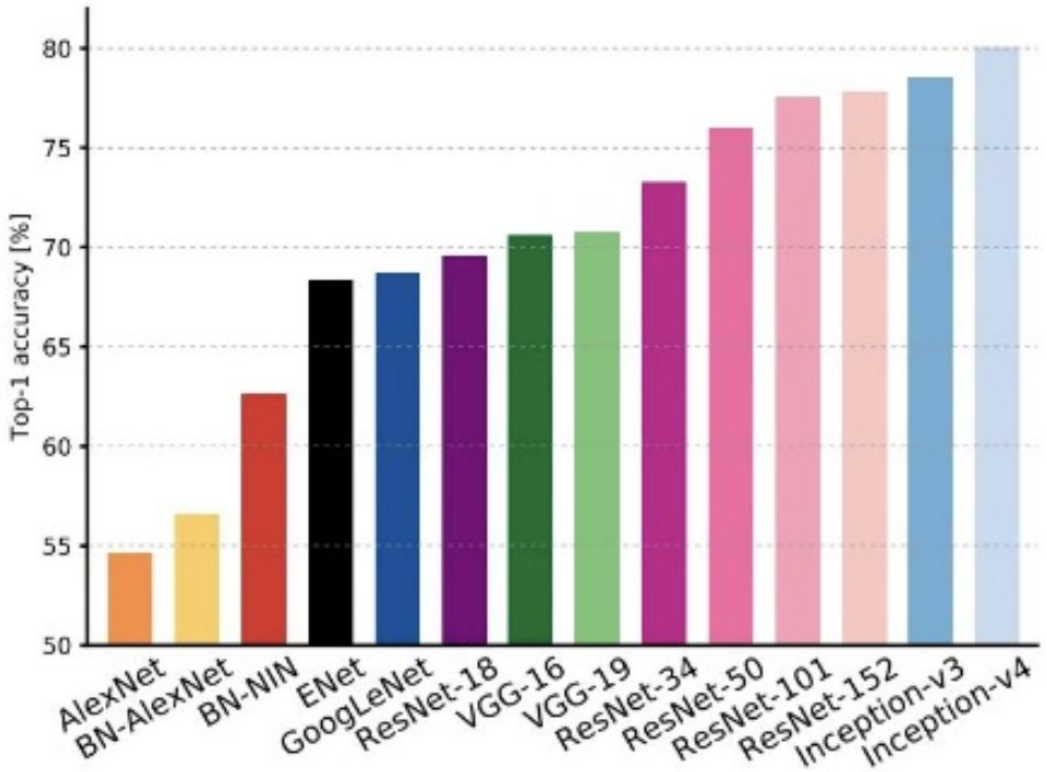


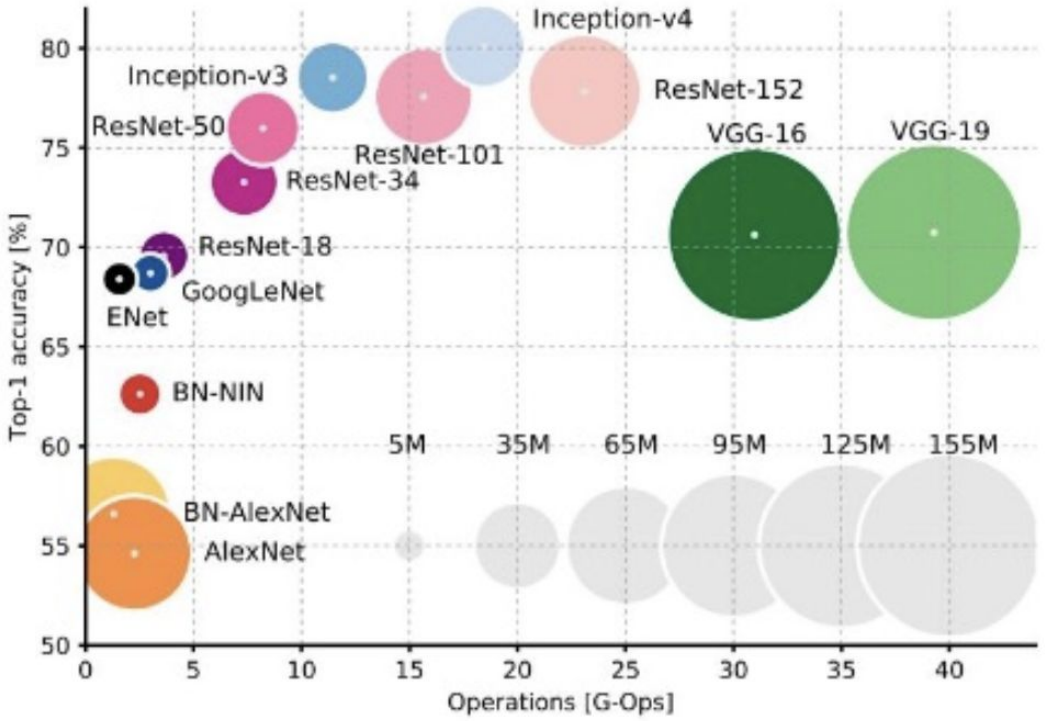


ILSVRC Error Evolution



ILSVRC Error Evolution





Famous CNNs

Inception v3 (2015)

- ~23M params
- 47 layers

Inception v4 (2016)

- 55M params
- 159 layers

MobileNet series (2017)

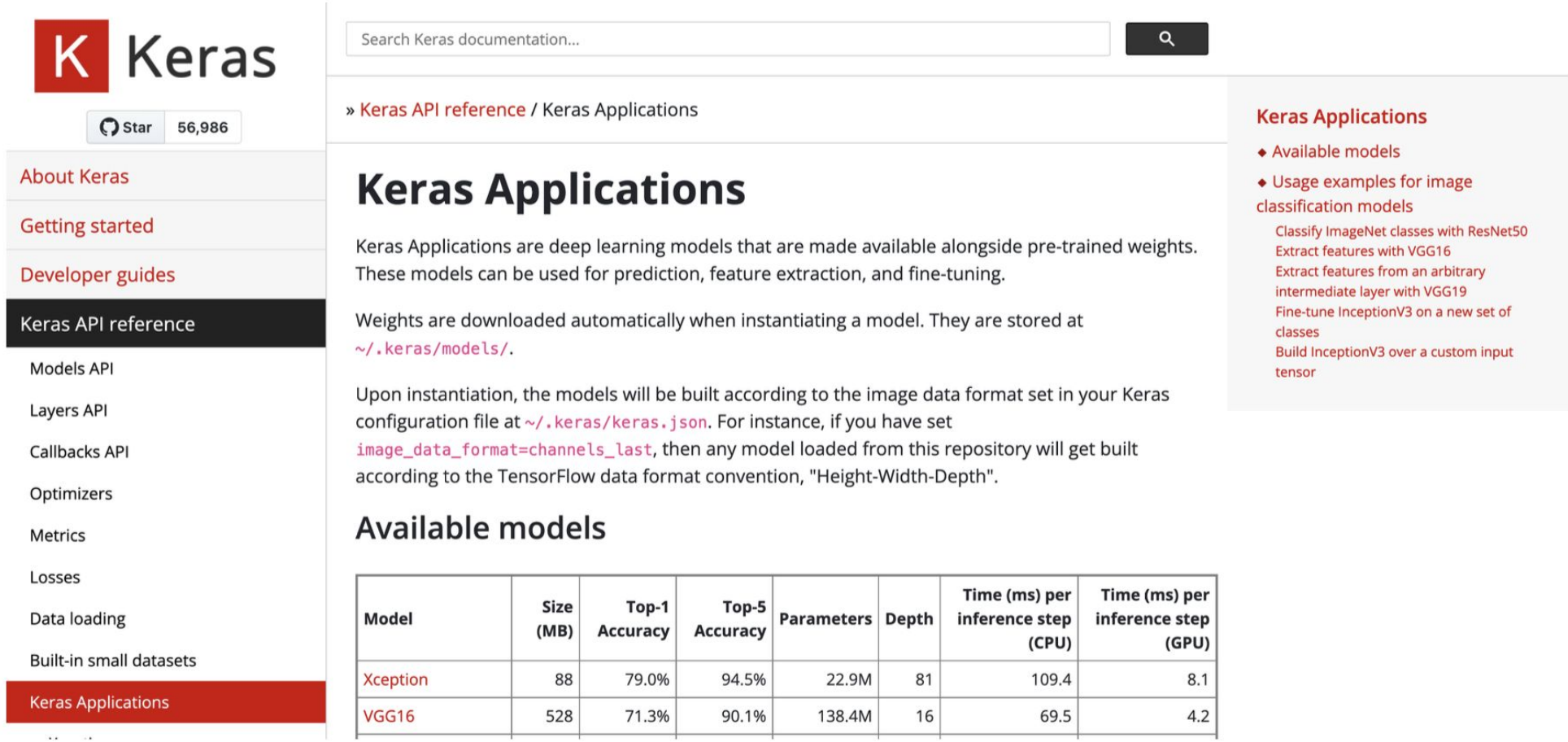
- Use depthwise separable convolutions to significantly reduce the computational cost.
- Designed for efficient on-device inference

EfficientNet series (2019)

- 22 to 155 layers
- 5.3M to 66M params

Famous CNNs

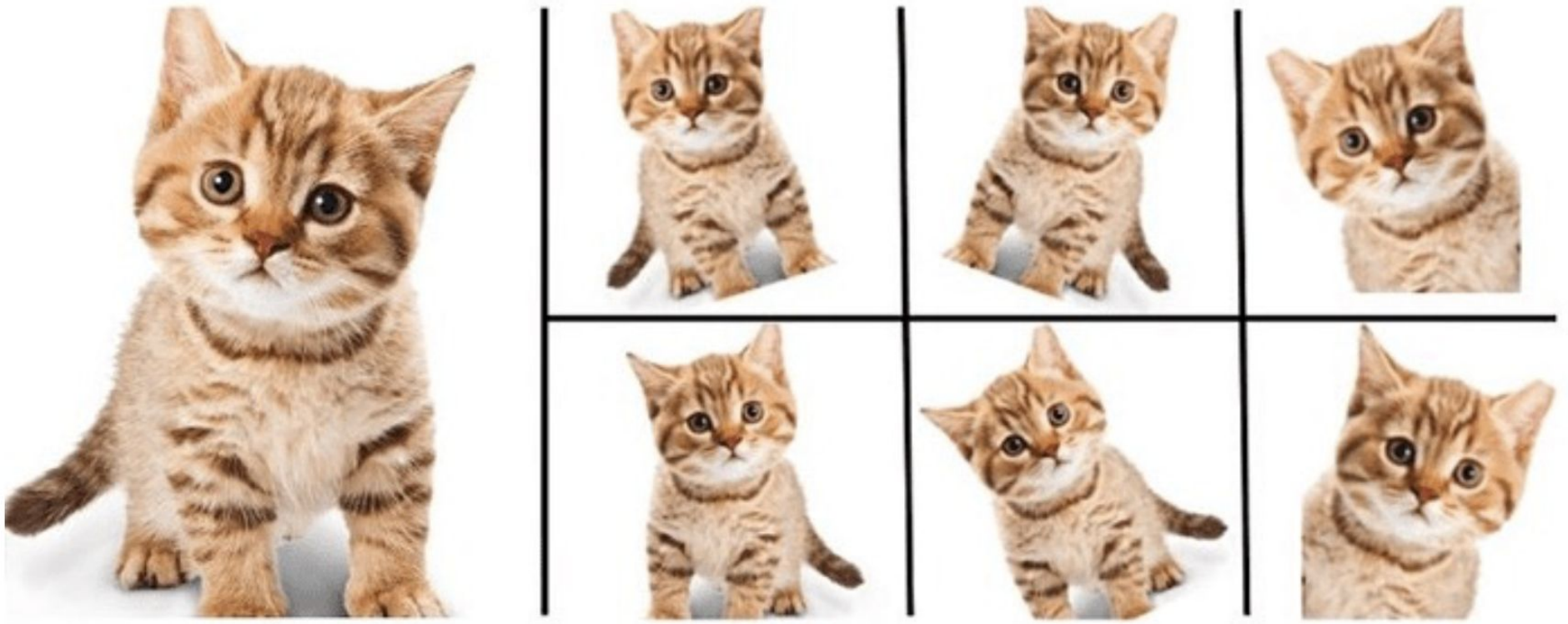
https://keras.io/api/applications/



Data Augmentation

- In computer vision, generally, the more data the better
- Data Augmentation can help you in a high variance situation
- https://www.tensorflow.org/tutorials/images/data_augmentation

Data Augmentation

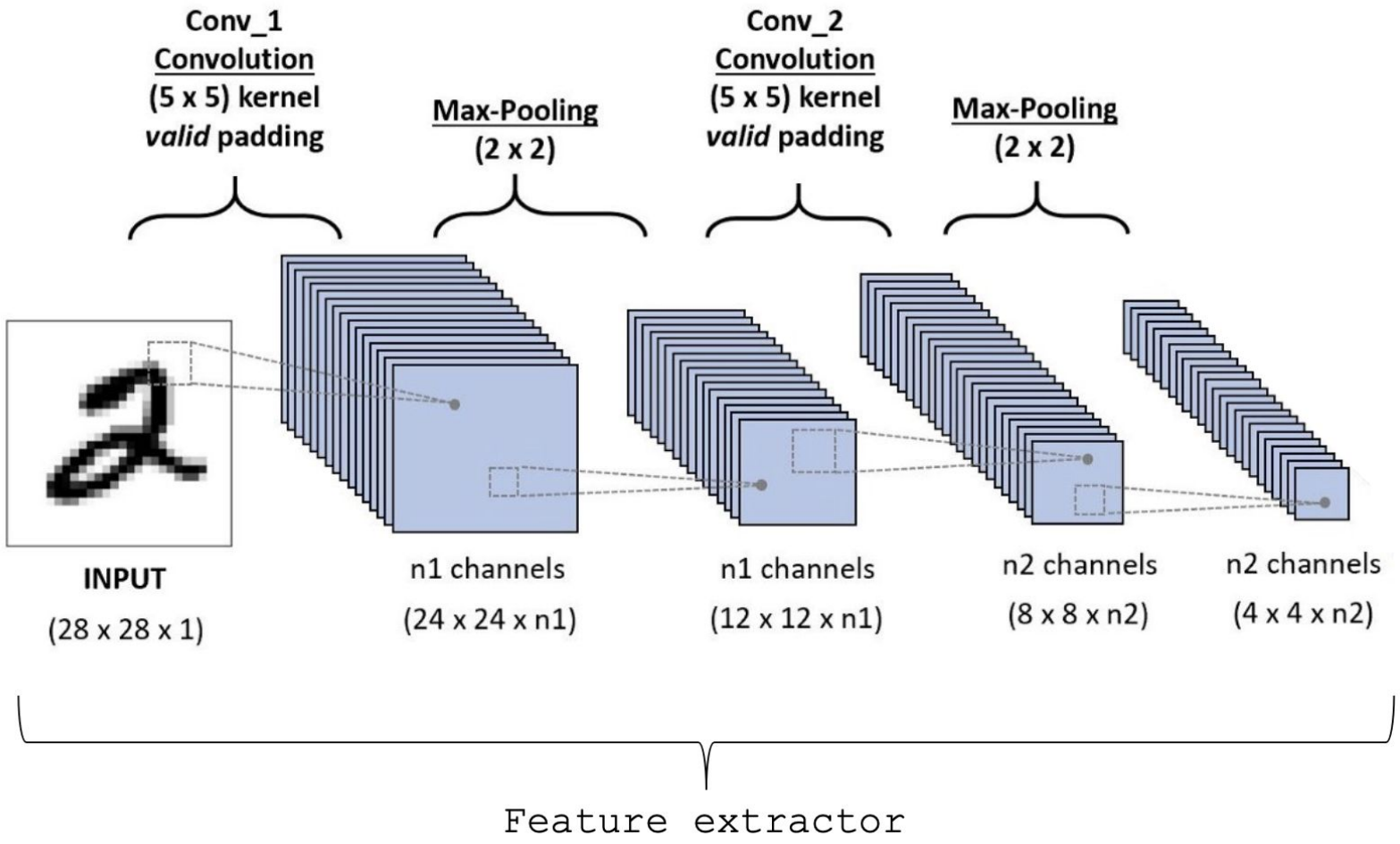


Transfer Learning

- Usually, Large ConvNets need weeks of training with a high computational cost.
- Fortunately, researchers often share the final trained neural nets.
- As convnets, the topology of the net could remain the same even for different input sizes.
- ... obviously, the classifier part might be different for your specific problem.
- One of the things we can do, it is to start from a **pre-trained net**, change the classifier (or part of it) and retrain just that part with our specific problem data.

- This process is called **transfer-learning**.

Transfer Learning: Feature extractor



Transfer Learning

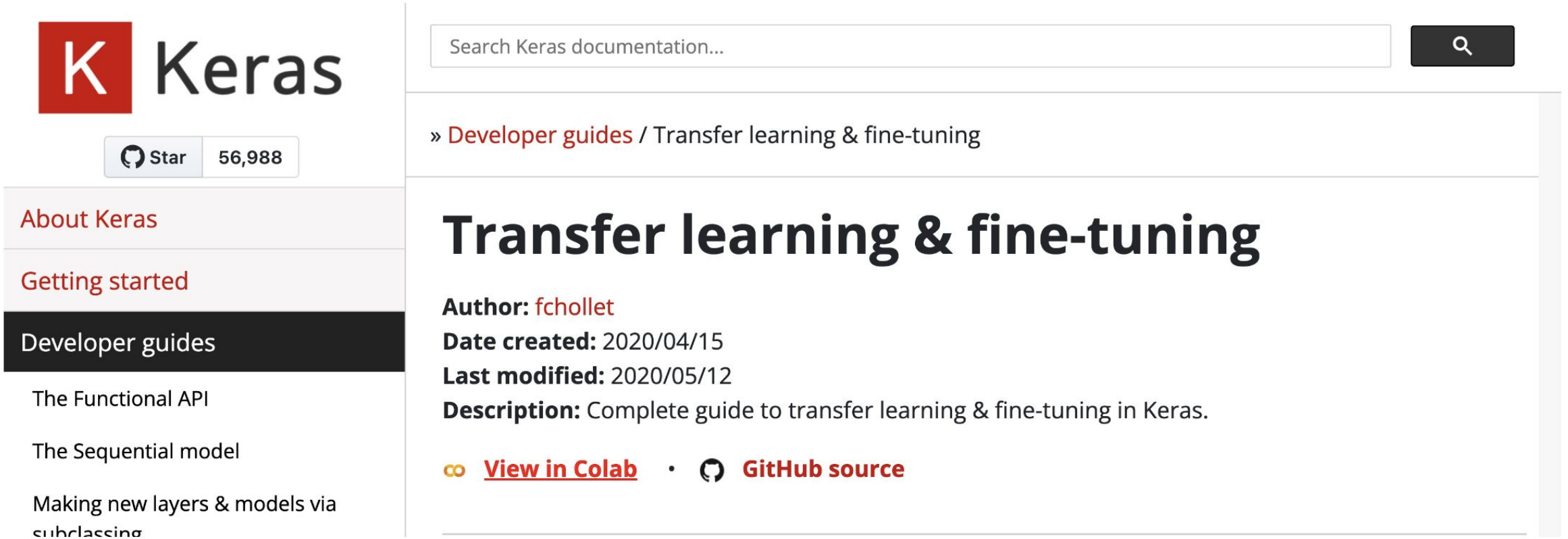
- The pre-trained net will work as a feature extractor

- We will NOT train the weights of those first layers, we will build our classifier on top of the ‘feature extractor’ (the pretrained network without its own classifier)

- Then, we can train the on-top classifier with our data

Transfer Learning

https://keras.io/guides/transfer_learning/



Applications

Classification

CAT

Applications

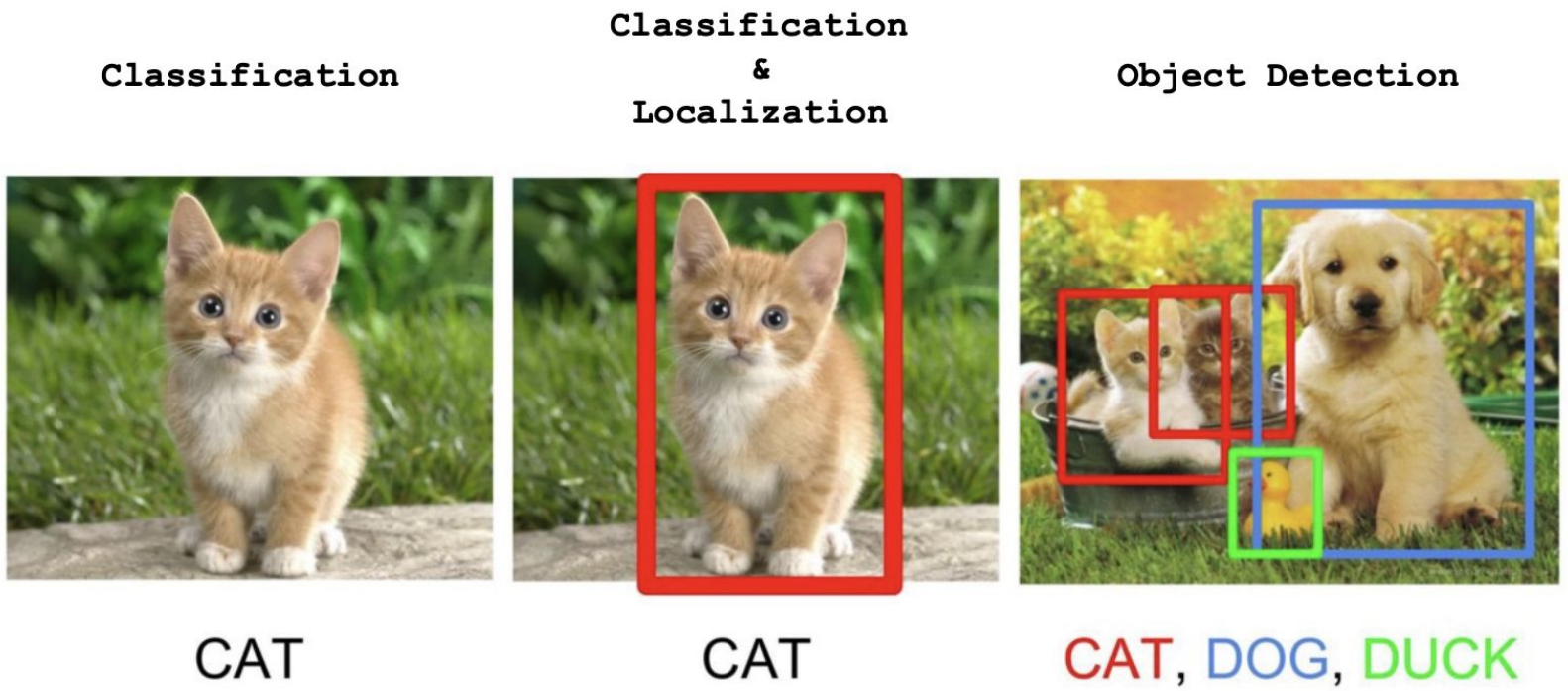
Classification

CAT

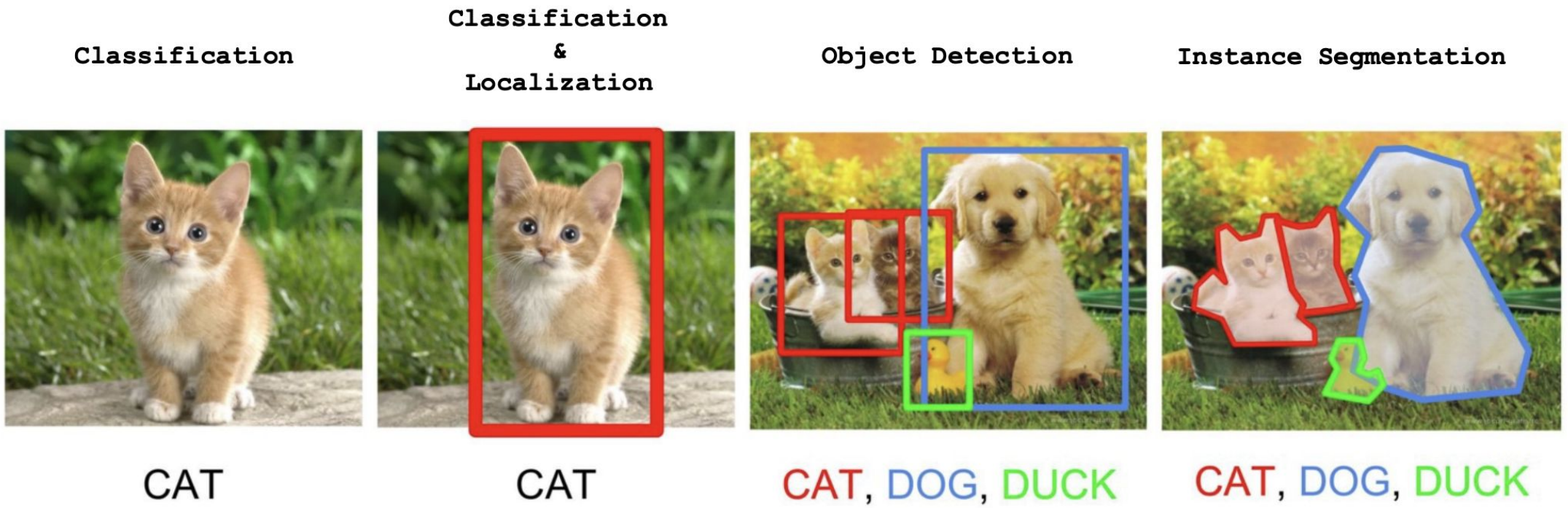
Classification
&
Localization

CAT

Applications



Applications



Apps: Classification and Localization

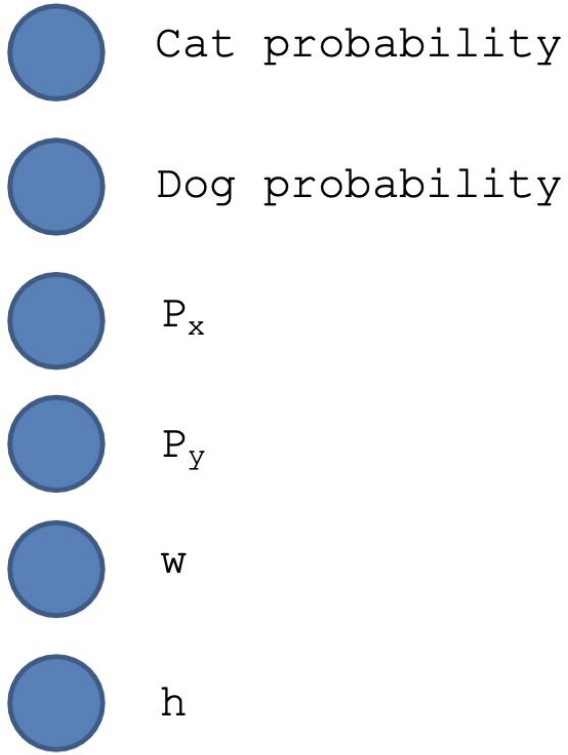
CAT

Apps: Classification and Localization

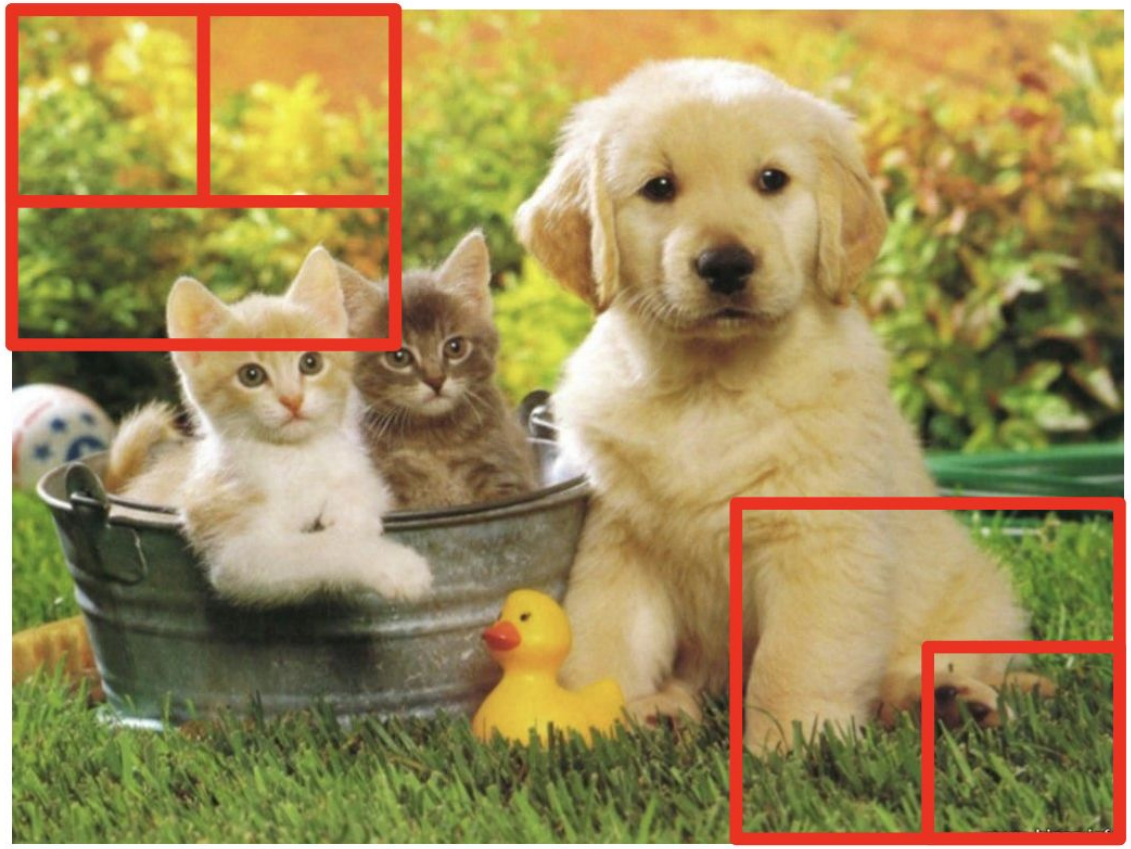
CAT

P_x, P_y, w, h

CNN

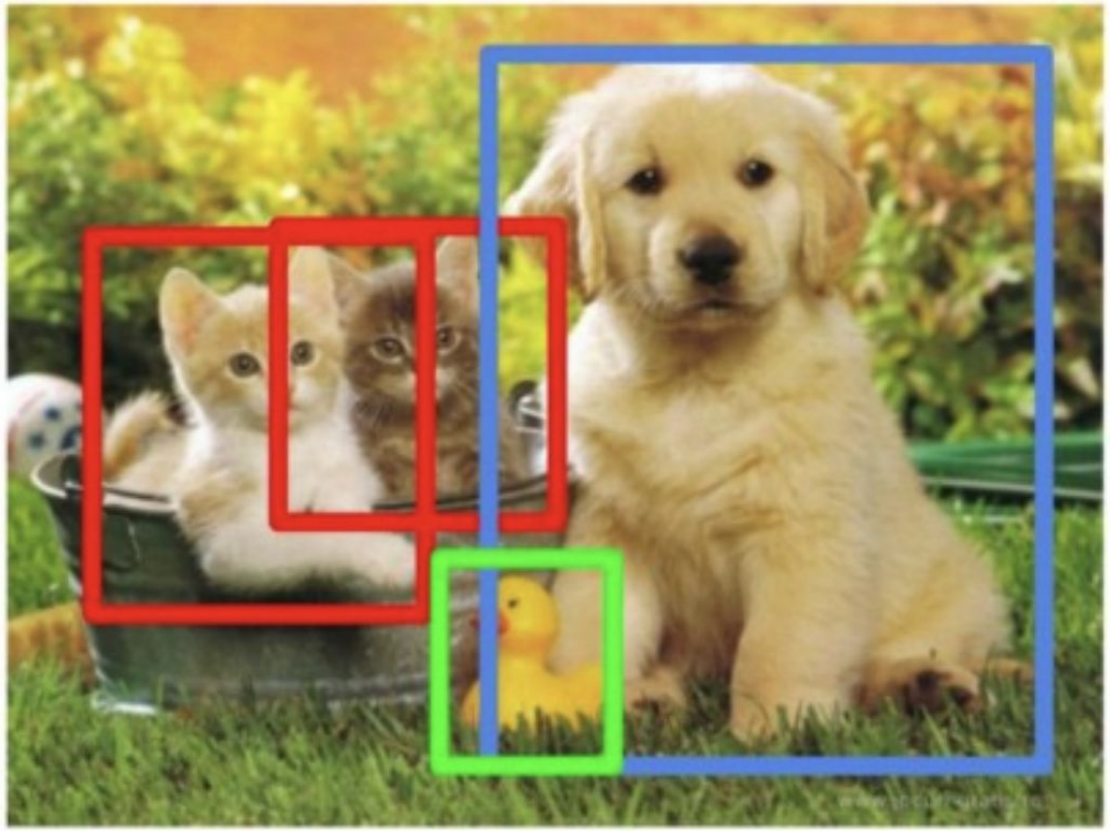


Apps: Object Detection (OD)



Sliding window detection.

Apps: Object Detection (OD)



Sliding window detection - very slow, infeasible

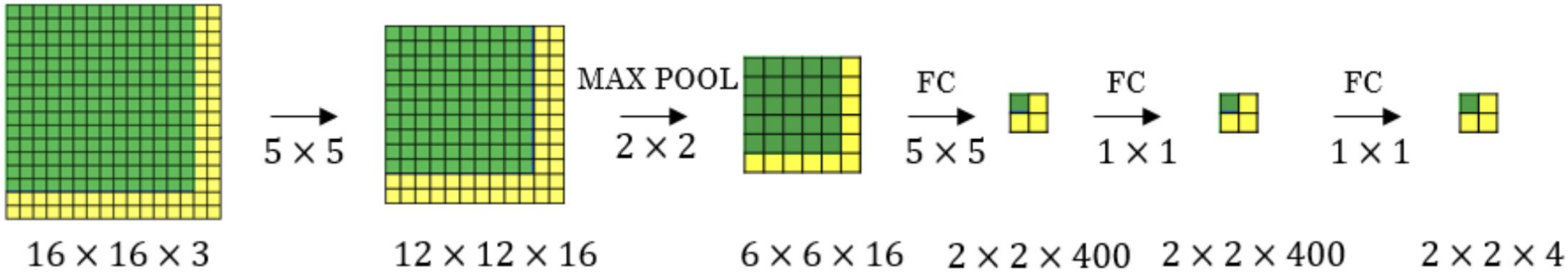
As we run N (classes) classifiers for every window for every size

OD: Sliding Window

- In the sliding window approach, we first train a classifier for every object (or a multi-class classifier)
- Then we use a sliding window to crop the images and pass them through the N classifiers to detect the object
- We use this with different sizes of our windows.
- In the epoch of simpler (linear or shallow) classifiers this was OK, but using conv-nets is very slow...

OD: Architecture

Instead of using the sliding window approach we leverage the convolution operator to get the same in a simple attempt. **Convolutional Sliding Window.**



OD: Architecture

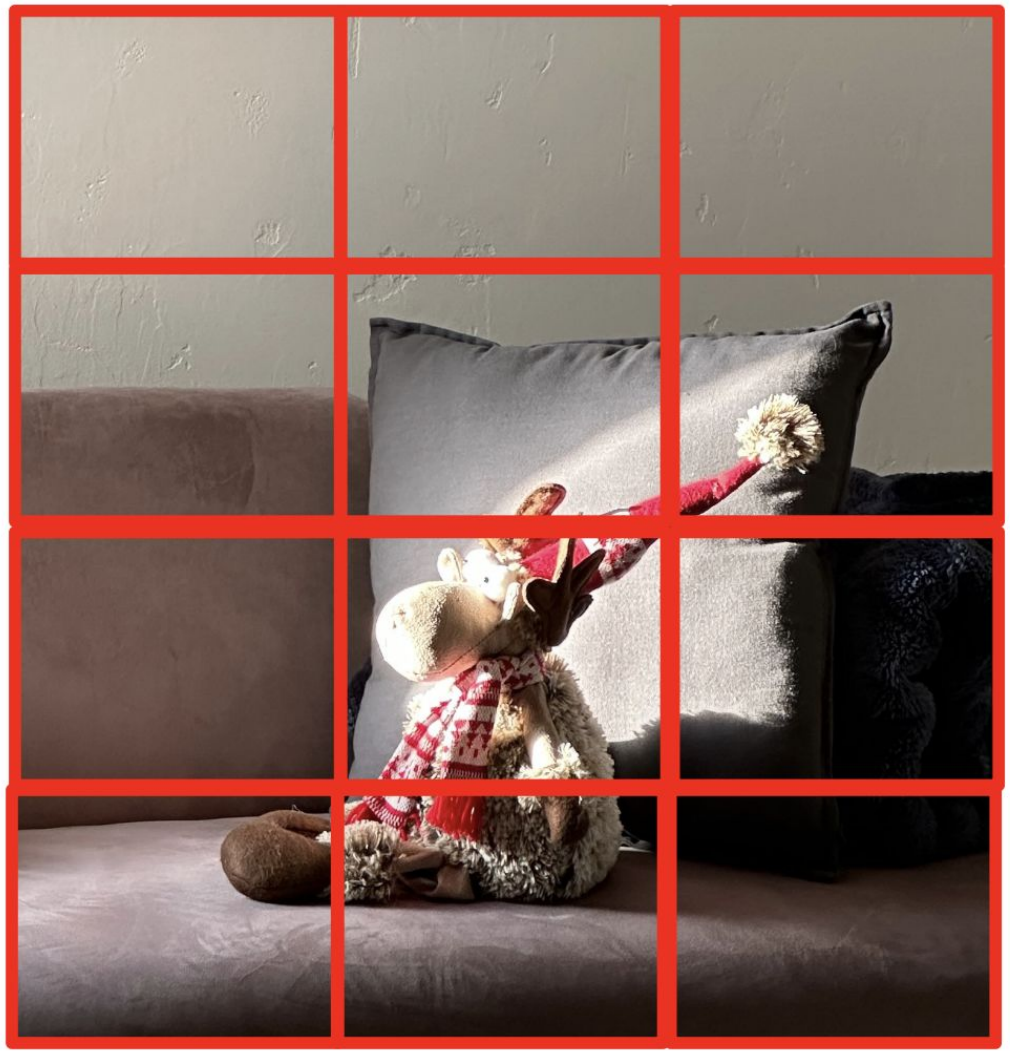
- ... The solution is to use the convolutional sliding window approach
- We leverage the power of the convolution operator
- We apply the multiclass classifier over all the windows that we will use in the sliding window approach but in a single pass
- For doing this the only that we need to do is to convert the fully connected layers into conv layers.

OD: YOLO

2015, You Only Look Once: Unified, Real-Time Object Detection

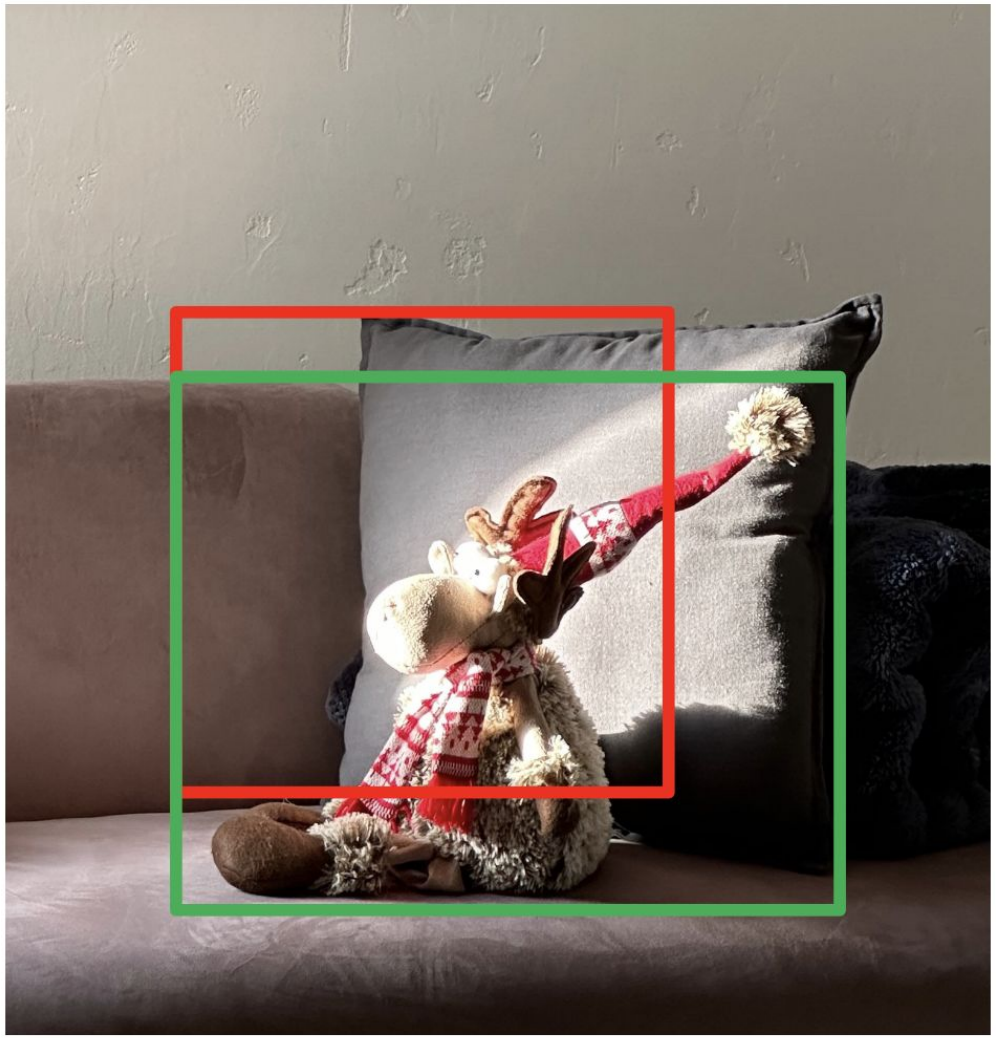
Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

OD: YOLO



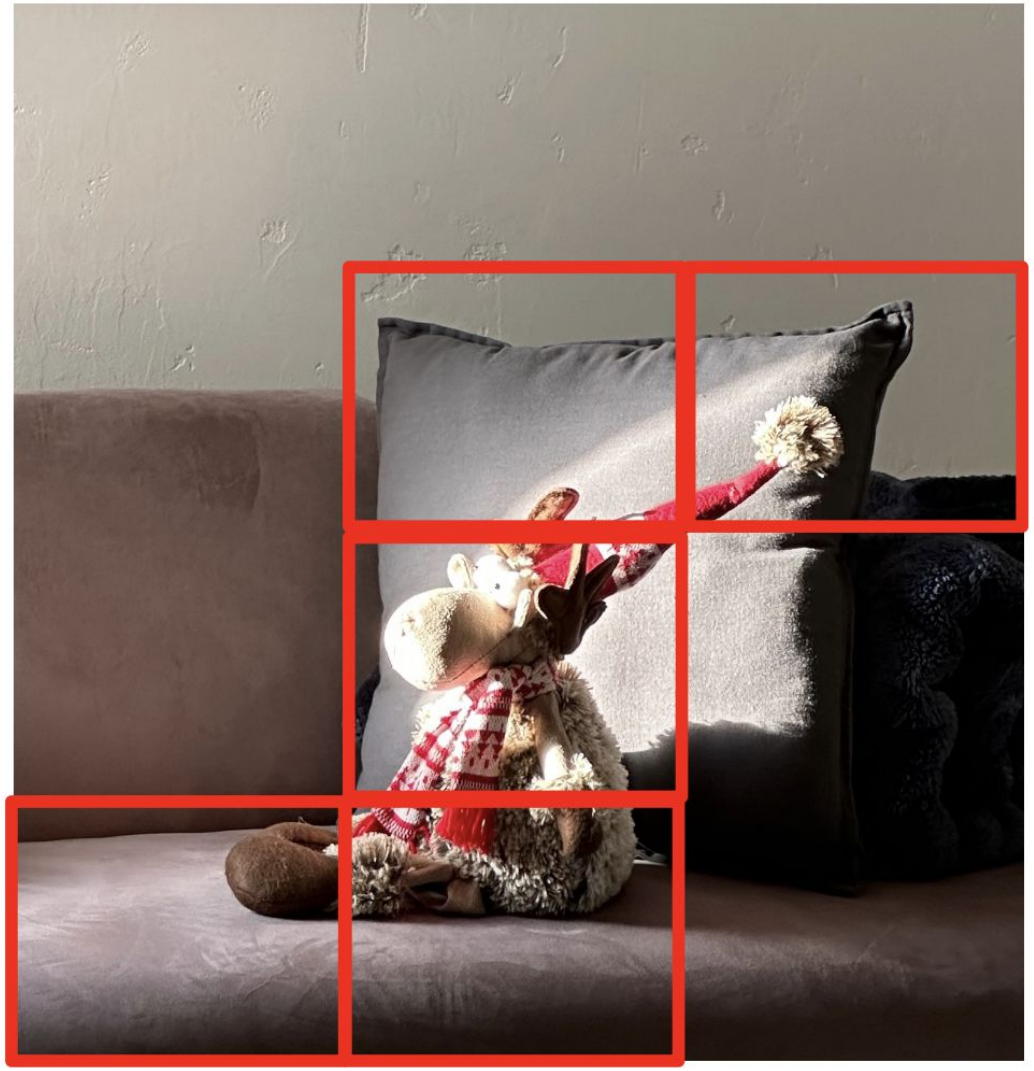
- Allows to get more precise bounding boxes with different sizes.
- We run (using convolution) a classification and localization convnet over a grid
- Labels include the actual bounding boxes
- Engineering:
  - Non-Max Suppression
  - Anchor boxes

OD: YOLO



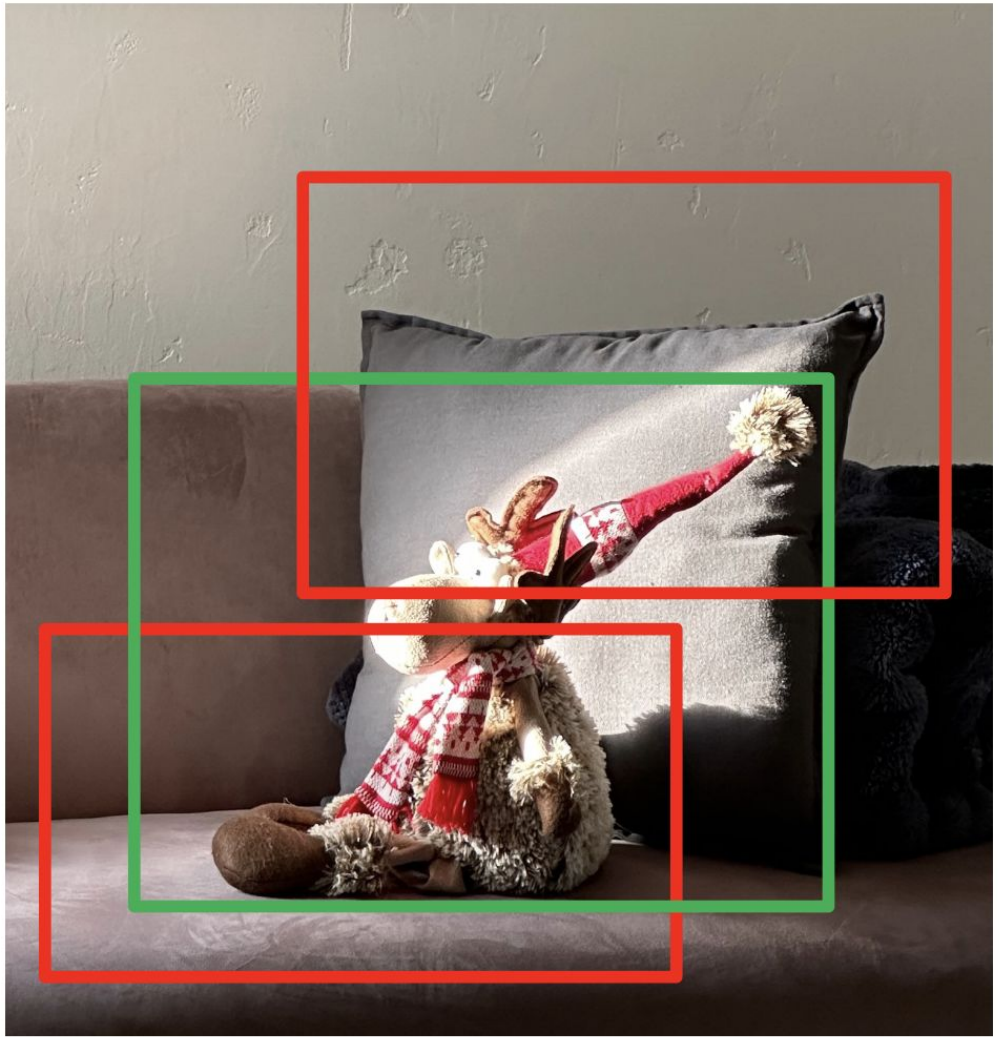
- Intersection over union.
- Size of intersection / size of the union > threshold

OD: YOLO



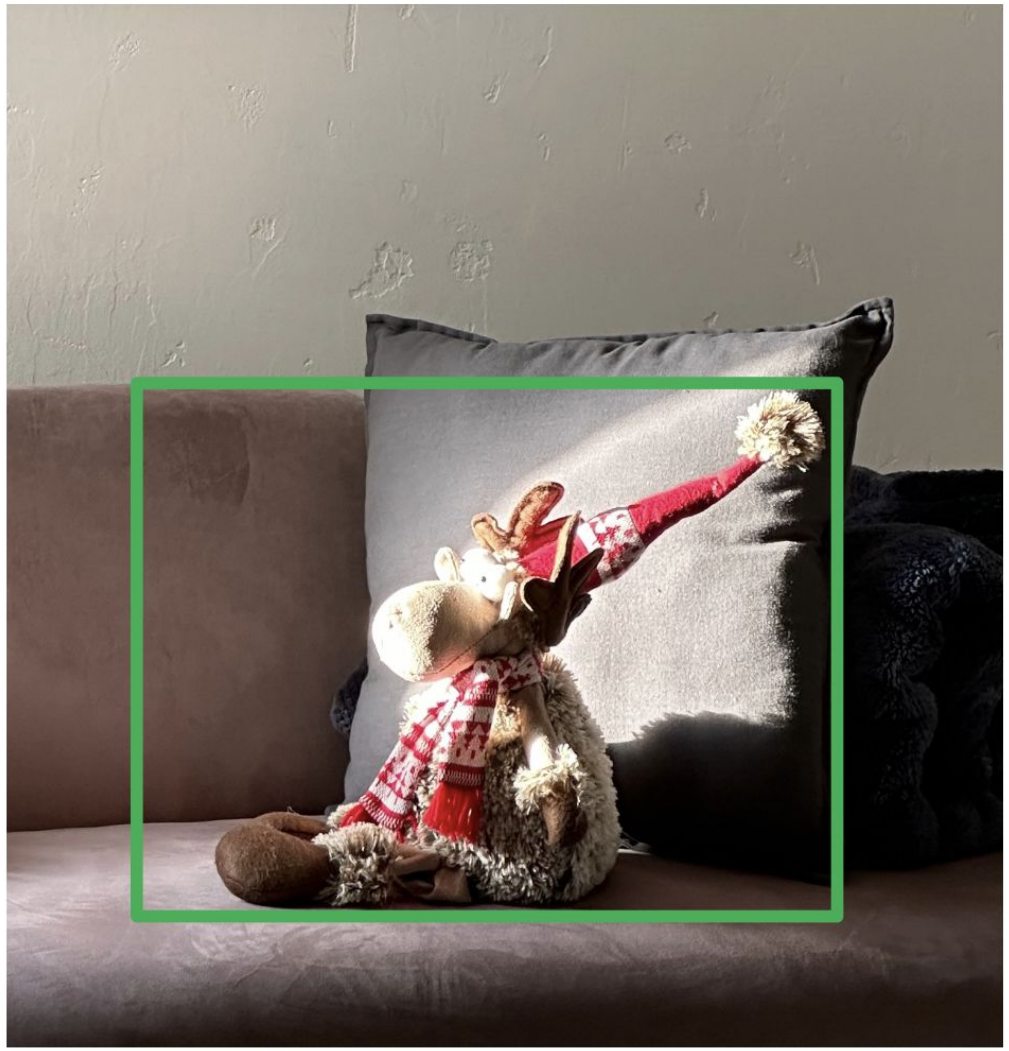
- We use Non-max suppression to solve the problem to get the same object in close grid cells.

OD: YOLO



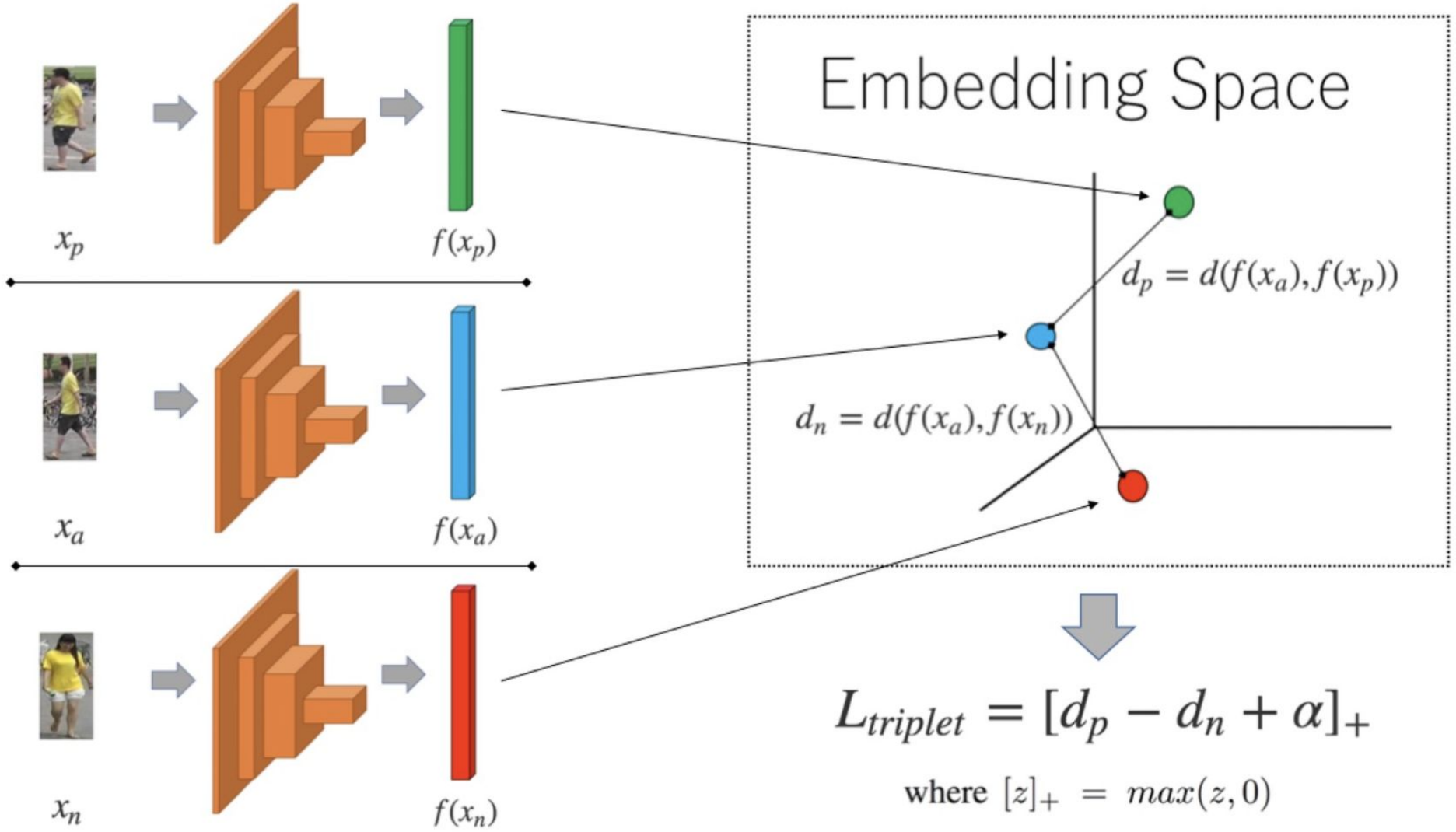
- We use Non-max suppression to solve the problem to get the same object in close grid cells.
- We can end up with different candidates to represent the object
  - We chose the one with highest probability
  - We discard the ones which highly overlap with the selected one.

OD: YOLO



- We use Non-max suppression to solve the problem to get the same object in close grid cells.
- We can end up with different candidates to represent the object
  - We chose the one with highest probability
  - We discard the ones which highly overlap with the selected one.

Face Verification



$$L_{triplet} = [d_p - d_n + \alpha]_+$$

where $[z]_+ = max(z, 0)$

Instance Segmentation