CNN Parameters and Training

Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 28th, 2021

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

Parameters in a CNN

Giacomo Boracchi

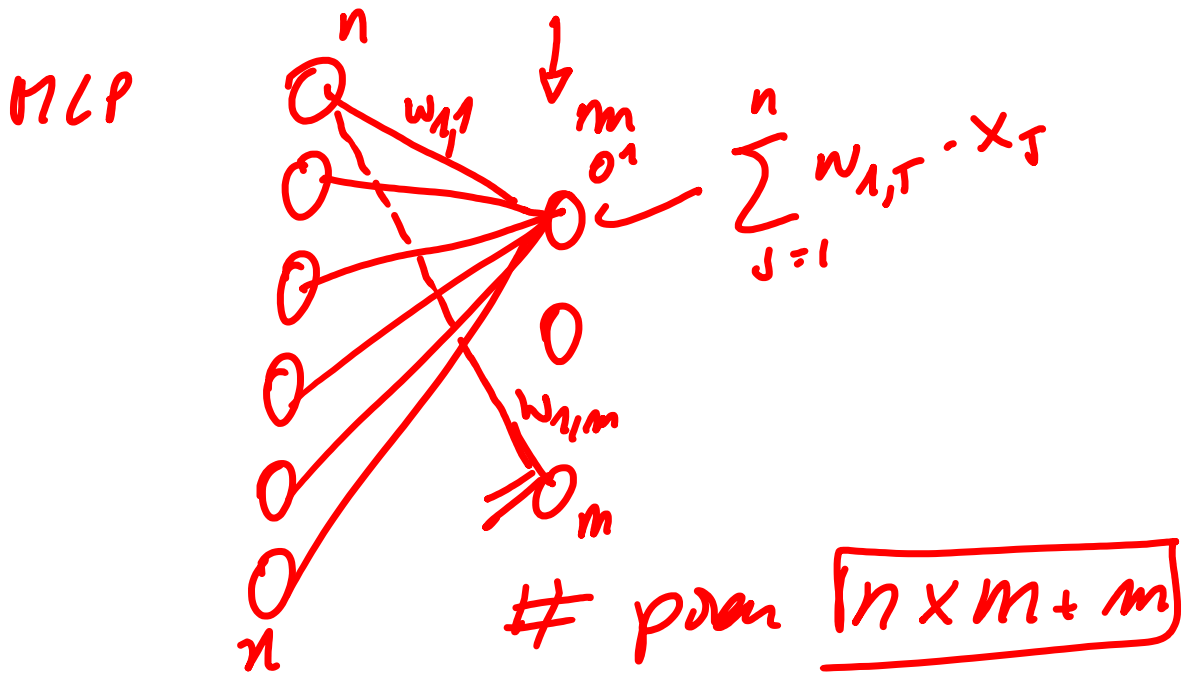
Convolutions as MLP

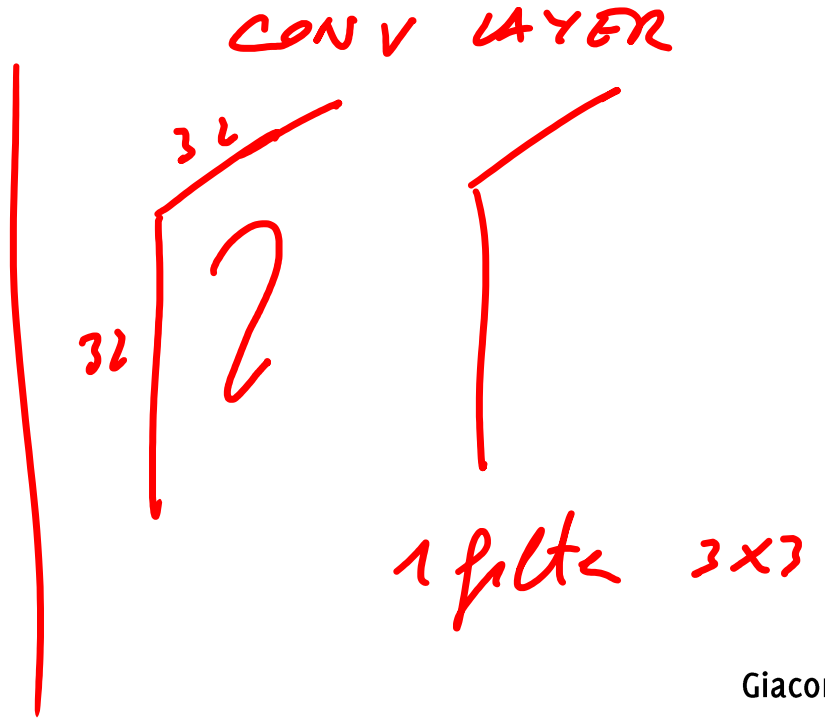
$$(f * g)(r,c) = \sum_{u,v} \underbrace{f(u,v)}_{\text{PARAMS}} \cdot \overbrace{g(r-u, c-v)}^{\text{INPUT}}$$

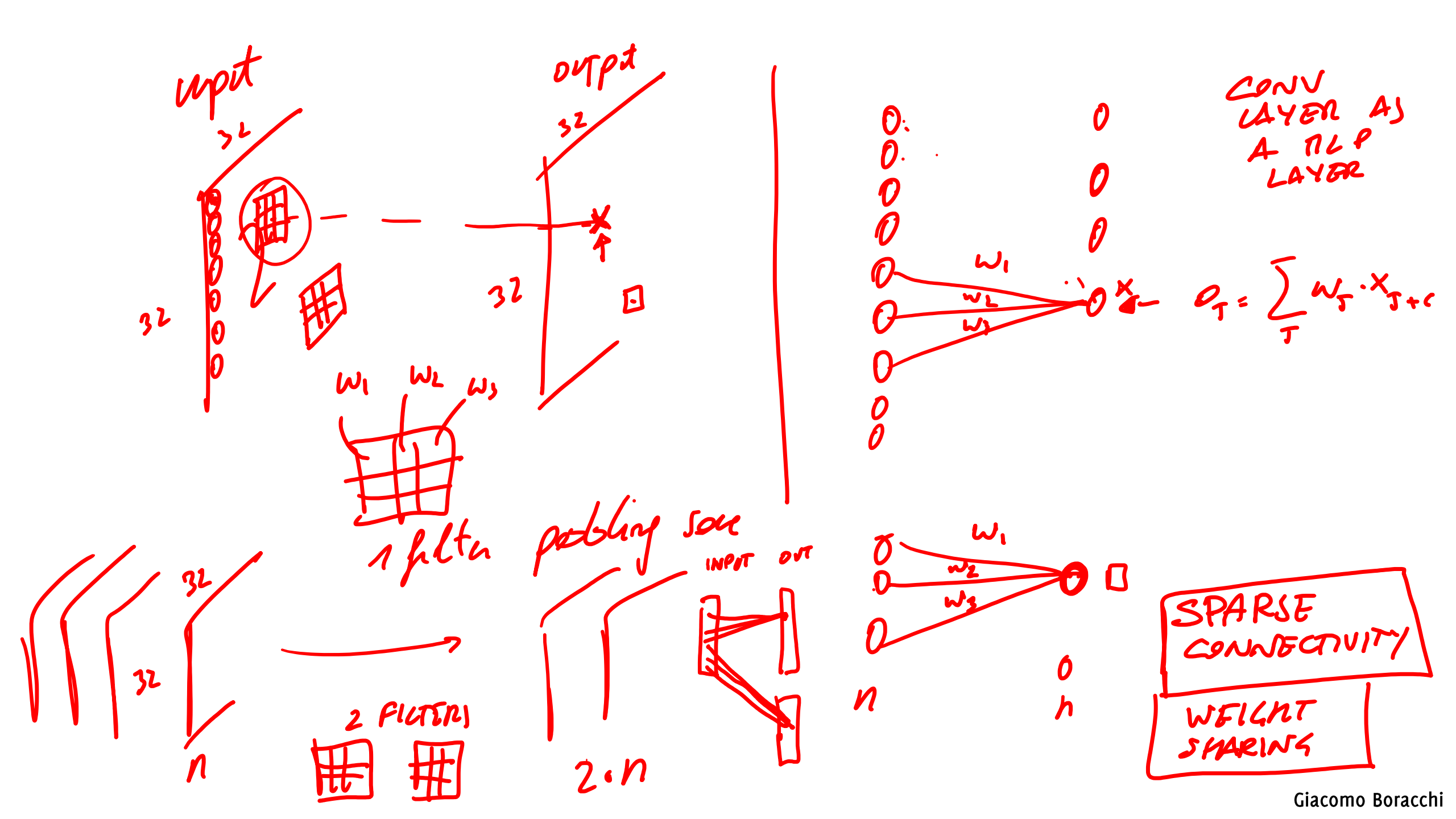
Convolution is a linear operation!

Therefore, if you unroll the input image to a vector, **you can consider convolution weights as the weights of a Multilayer Perceptron Network!**

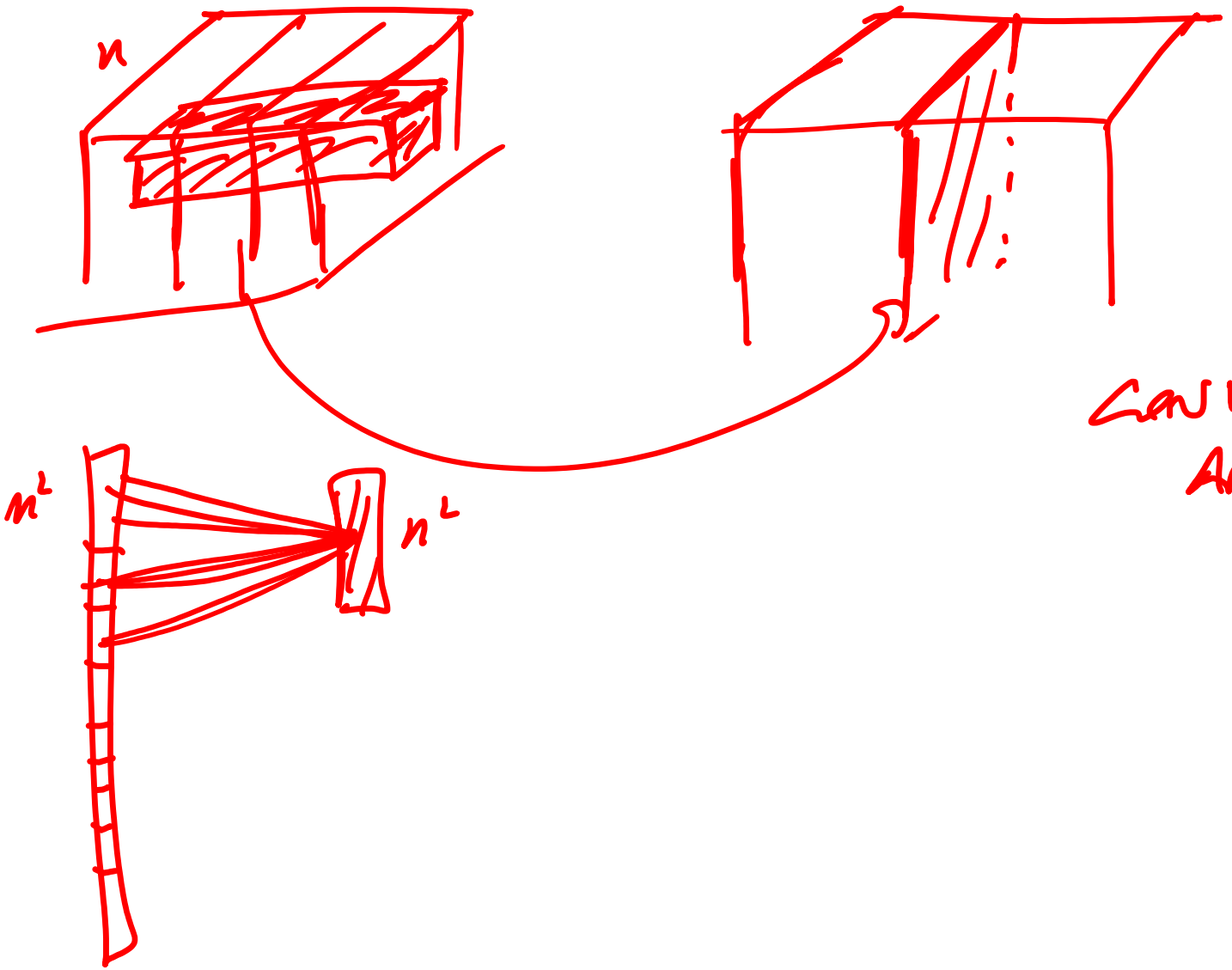
What are the differences between MLP and CNNs then?







Giacomo Boracchi

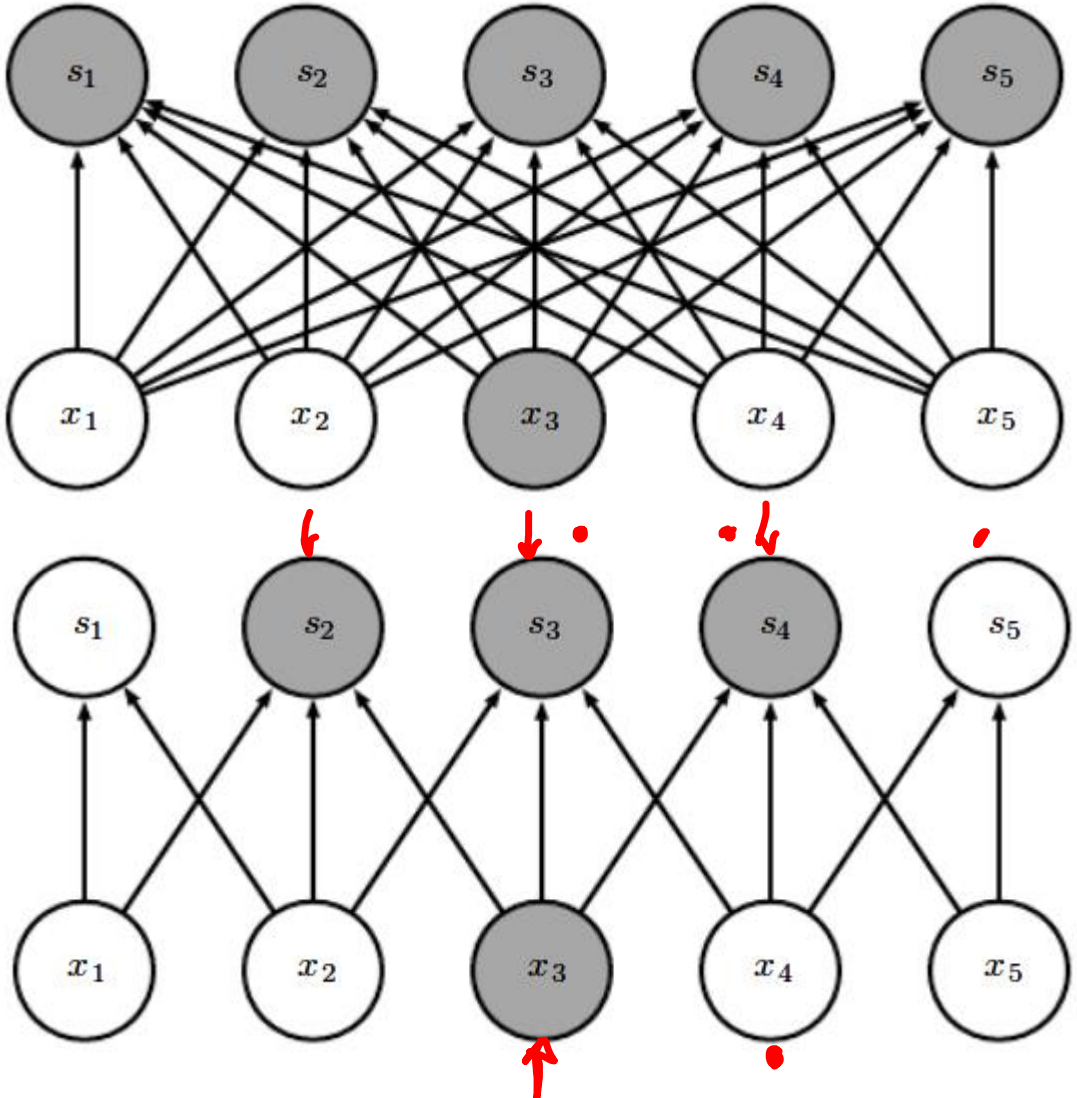


CONV LAYERS
ARE MLP LAYERS
WITH SPARSE CONN.
WEIGHT SHARING

CNNs has Sparse Connectivity

MLP, Fully connected

3x1 convolutional



Output neurons

Input neurons

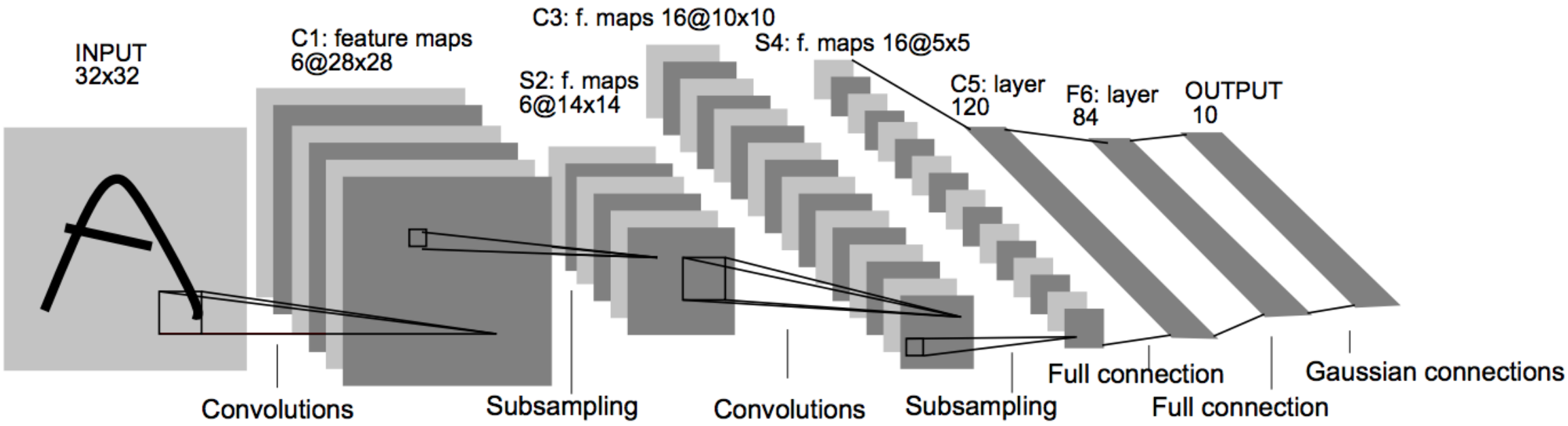
Output neurons (unrolled activations at the first layer)

Input neurons (unrolled image)

Weight Sharing / Spatial Invariance

In a CNN, all the neurons in the same slice of a feature map use the same weights and bias: this reduces the nr. of parameters in the CNN.

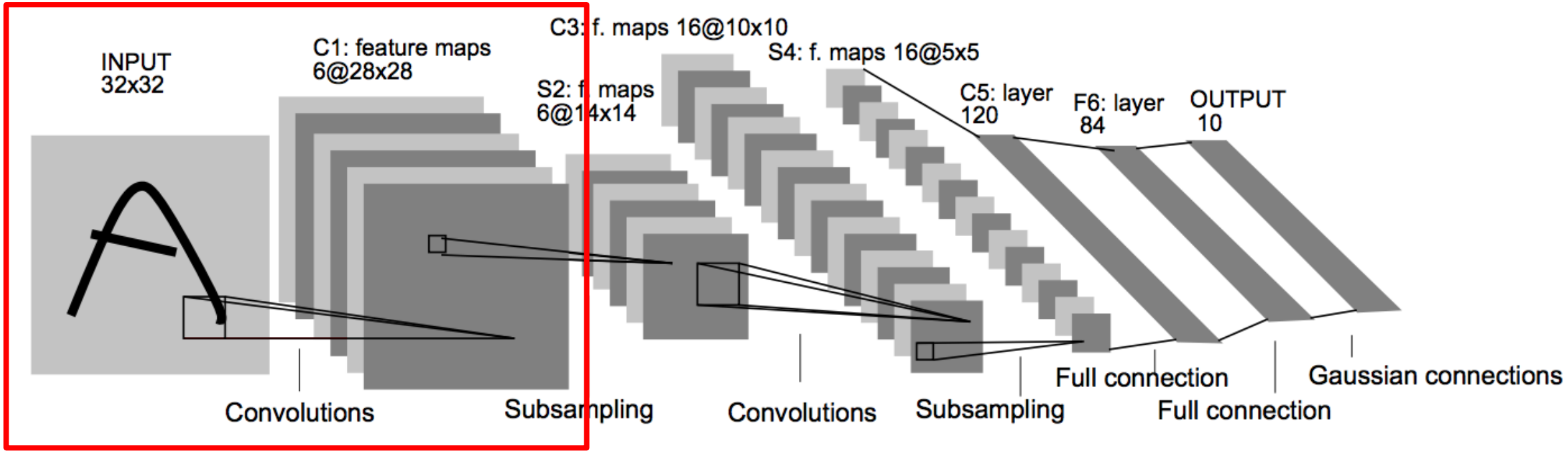
Underlying assumption: **if one feature is useful** to compute **at some spatial position** (x,y), **then it should also be useful** to compute at a **different position** (x2,y2)



Weight Sharing / Spatial Invariance

If the first layer were a MLP:

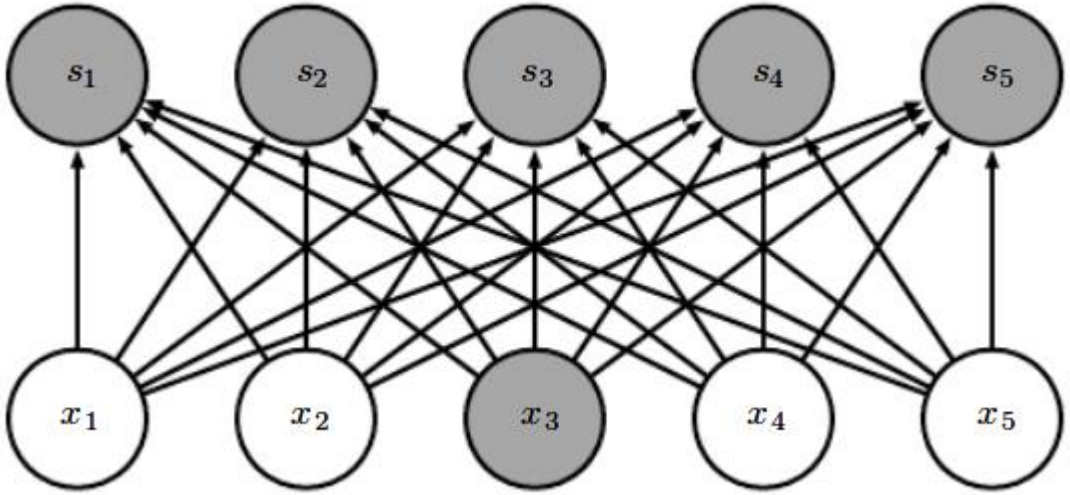
- it should have had 28 x 28 x 6 neurons in the output
- Consider sparse connectivity: 5x5 nonzero weights each neuron
- It would be a 28 x 28 x 6 x 25 weights + 28 x 28 x 6 biases (122 304)
  CNN share the same weights among neurons of the same layer



Parameter sharing

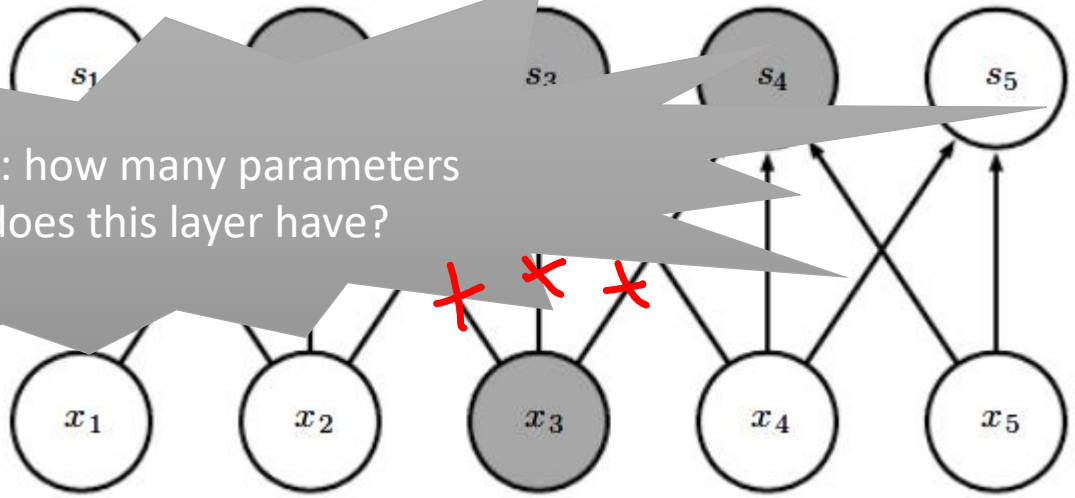
Fully connected

5x5 = 25 weights
(+ 5 bias)



3x1 convo

3 weights!
(+ 1 bias)



Quiz: how many parameters does this layer have?

+ 5 biases

5 outputs

3 inputs

The Receptive Field
A very important aspect in CNNs

Giacomo Boracchi

The Receptive Field

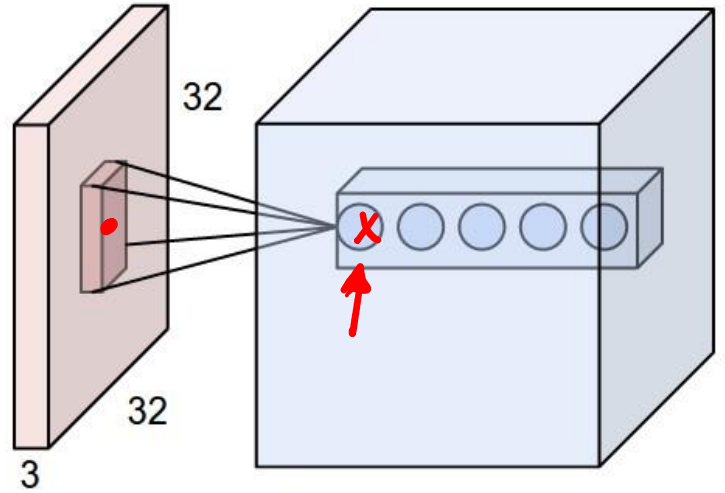
One of the basic concepts in deep CNNs.

Due to sparse connectivity, unlike in FC networks where the value of each output depends on the entire input, **in CNN each output only depends on a specific region in the input.**

This region in the input is the receptive field for that output

The deeper you go, the wider the receptive field is: maxpooling, convolutions and stride > 1 increase the receptive field

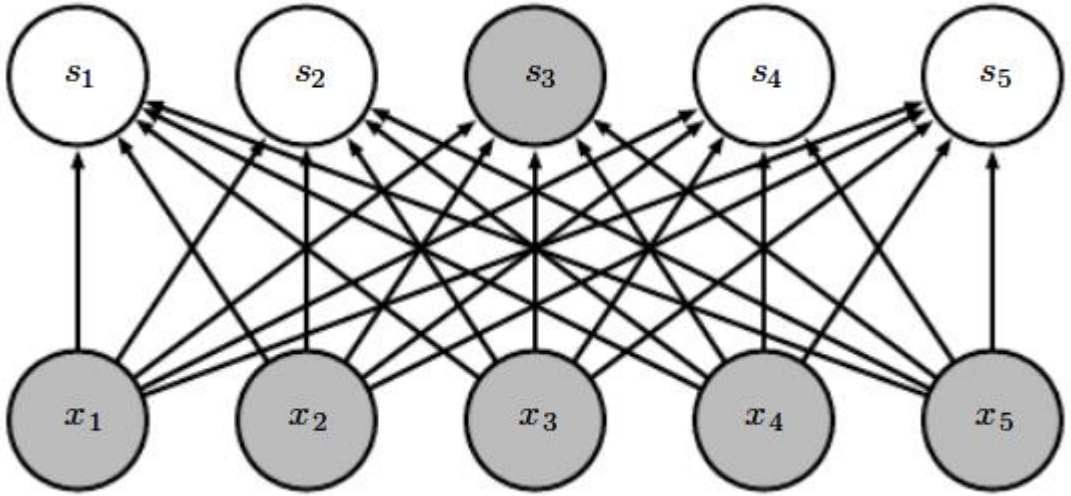
Usually, the receptive field refers to the final **output unit** of the network in relation to the network input, but the same definition holds for intermediate volumes



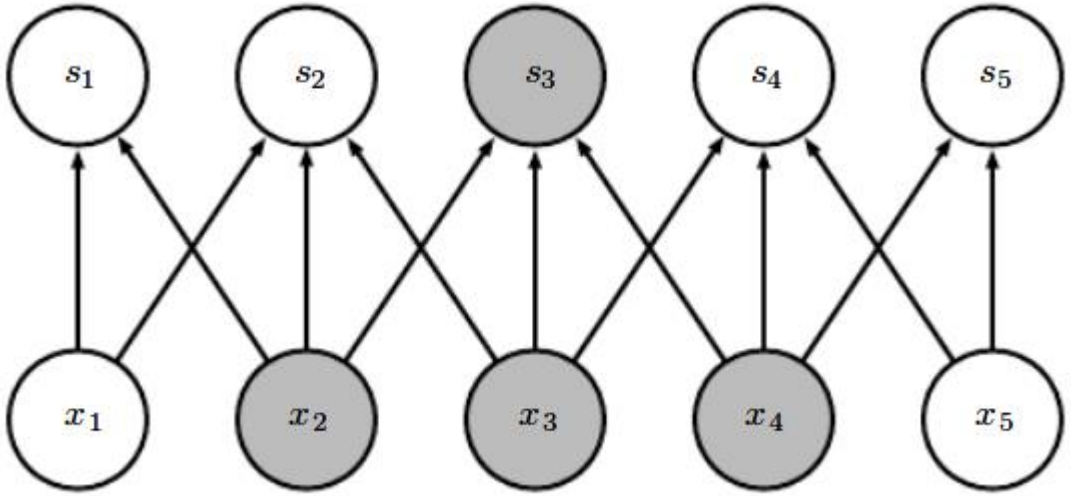
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Receptive fields

MPL, Fully connected



3x1 convolutional

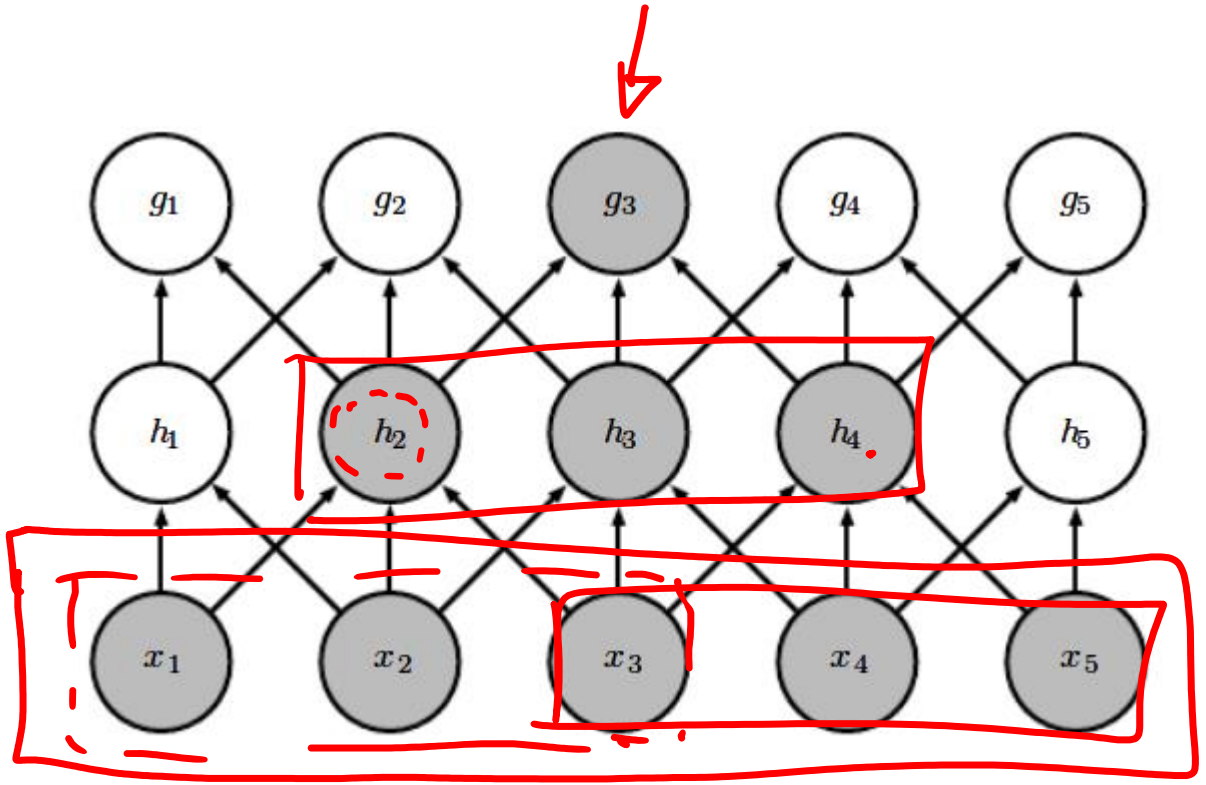


Receptive fields

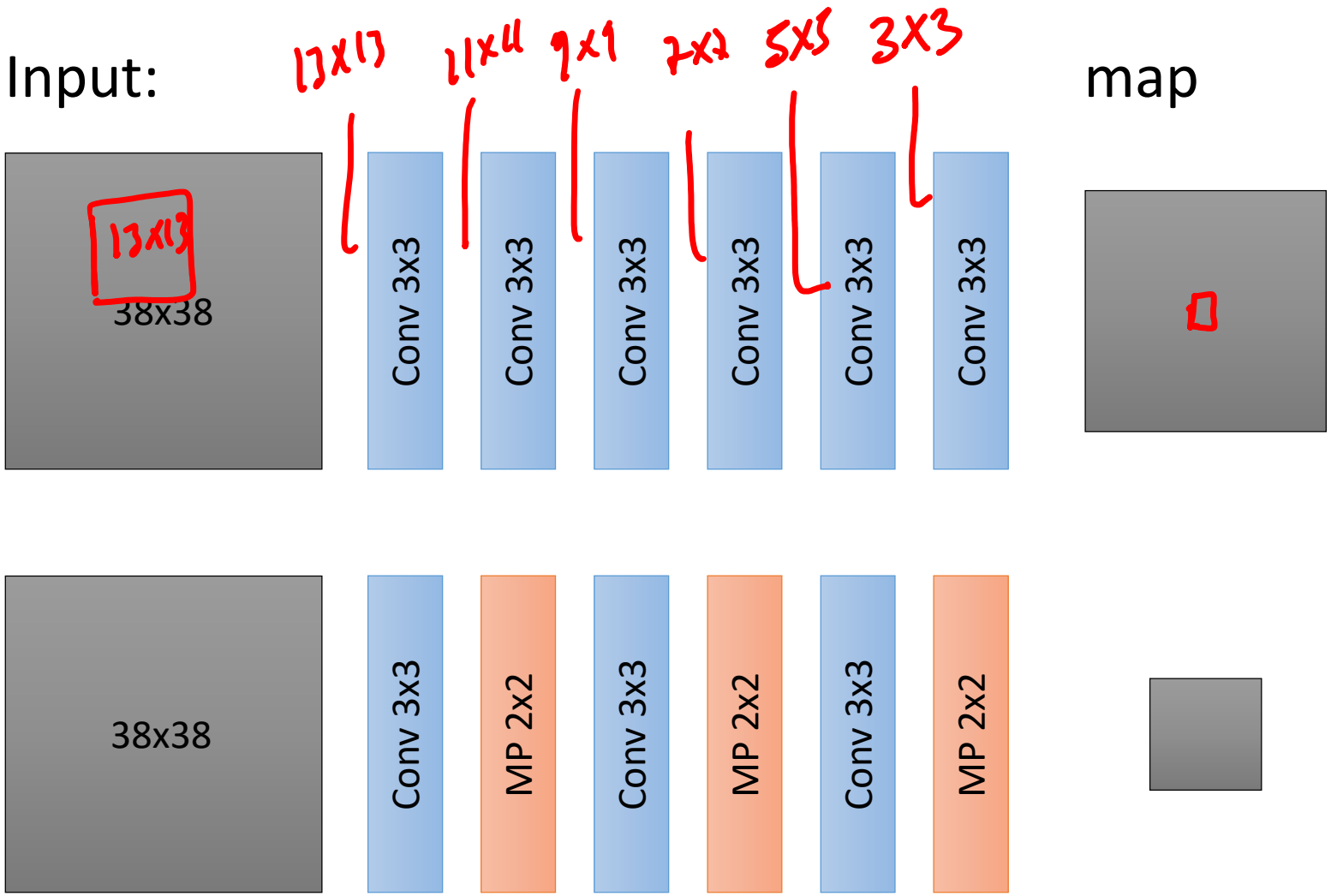
Deeper neurons depend on wider patches of the input (convolution is enough to increase receptive field, no need of maxpooling)

3x1 convolutional

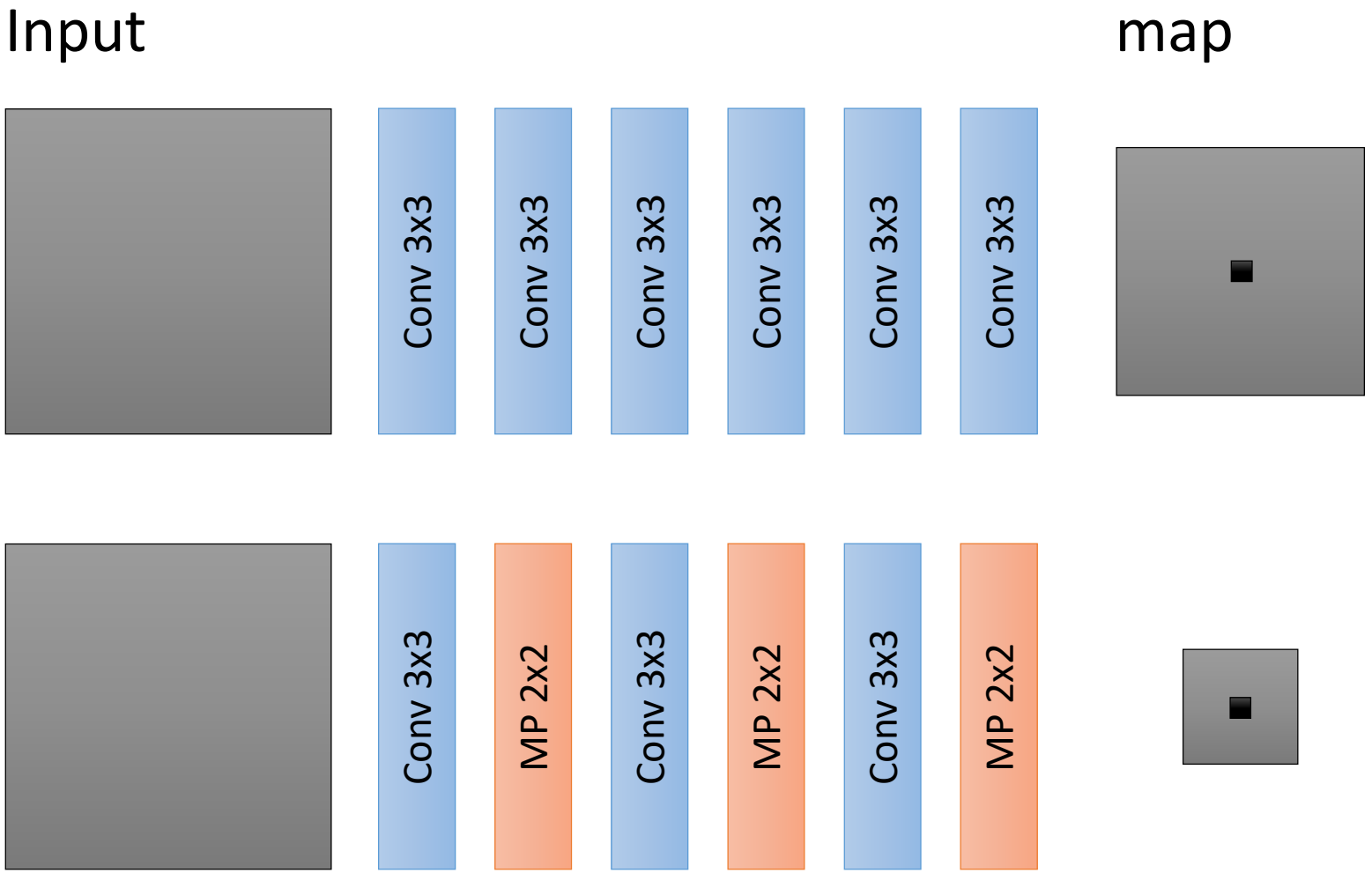
3x1 convolutional



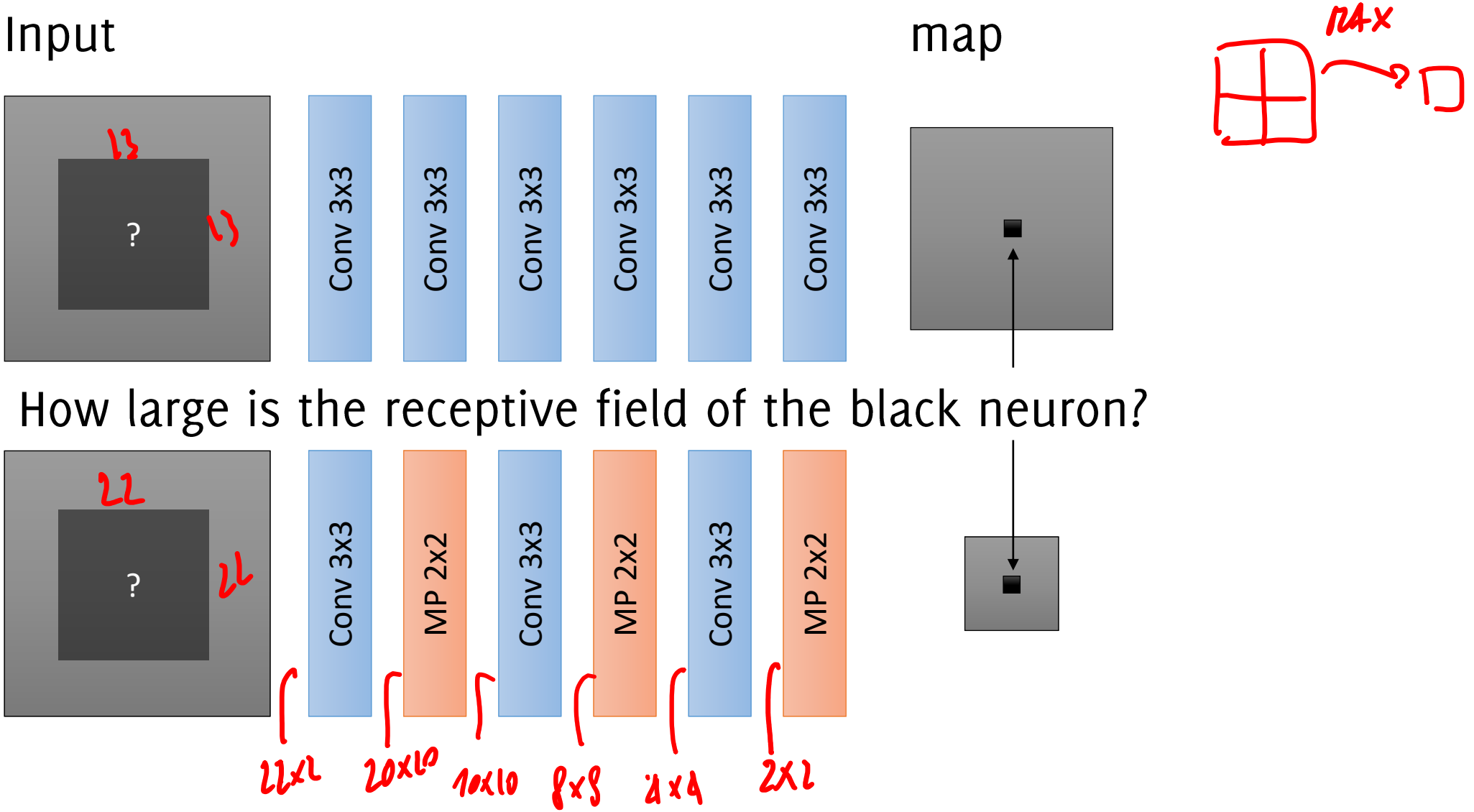
Exercise


Input:
13x13
11x11
9x9
7x7
5x5
3x3
map
13x13
38x38
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
38x38
Conv 3x3
MP 2x2
Conv 3x3
MP 2x2
Conv 3x3
MP 2x2

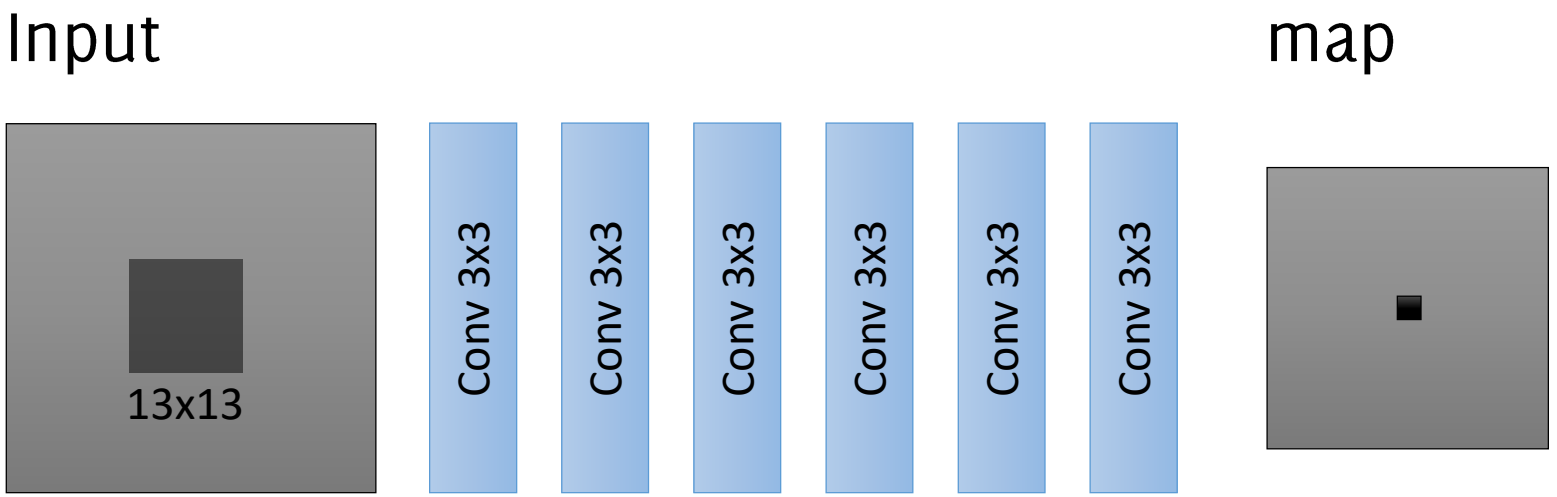
Receptive fields



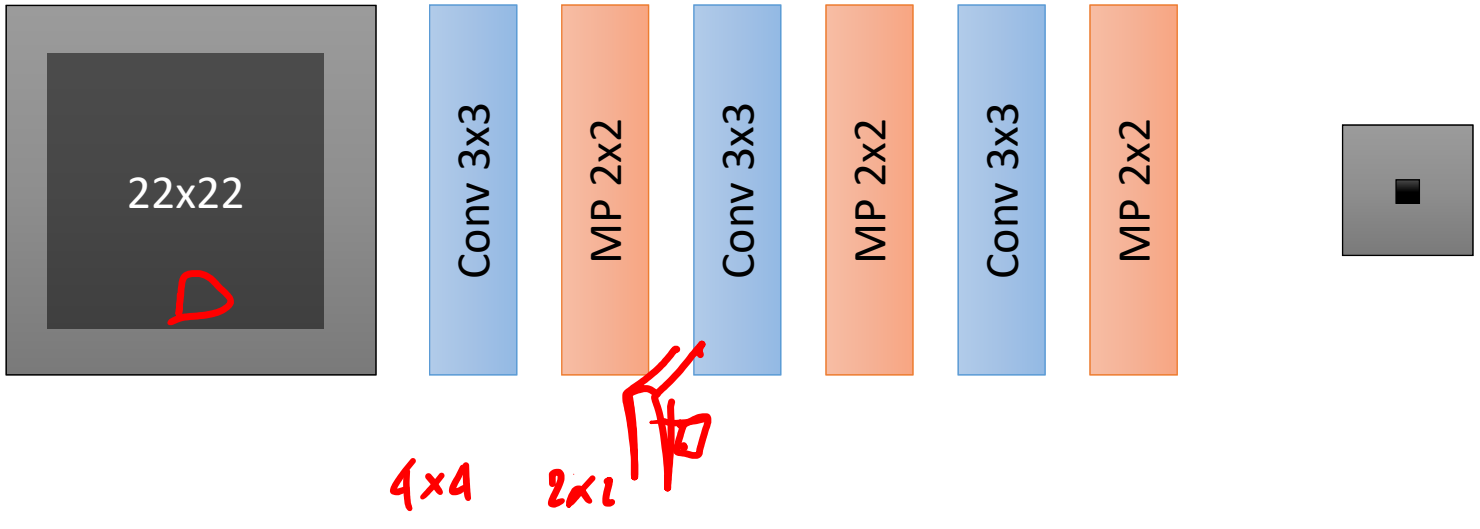
Receptive fields



Receptive fields



How large is the receptive field of the black neuron?



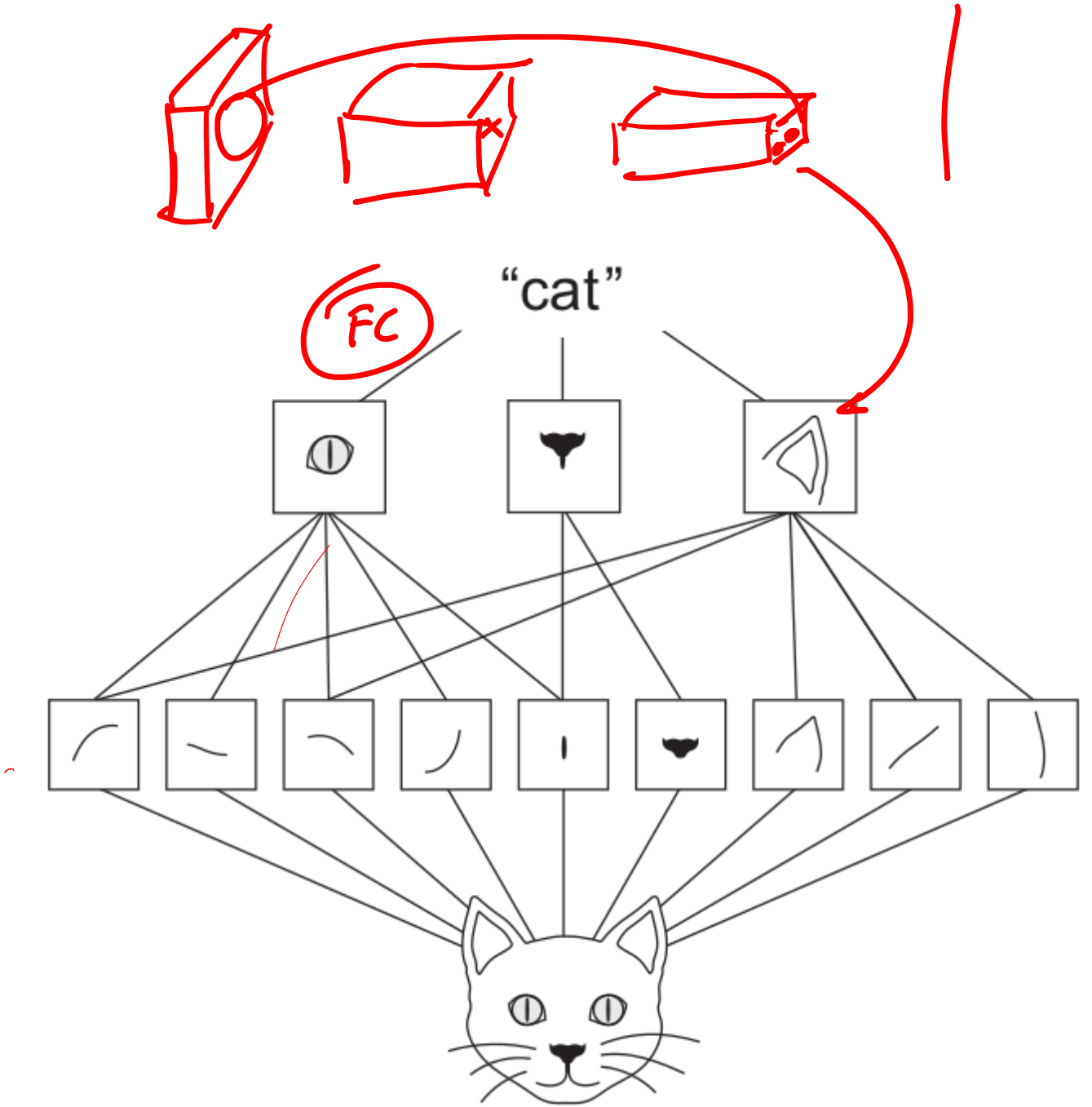
As we move deeper…

As we move to deeper layers:

- spatial resolution is reduced
- the number of maps increases

We search for **higher-level patterns**, and **don't care too much about their exact location**.

There are more high-level patterns than low-level details!



CNN Training

Giacomo Boracchi

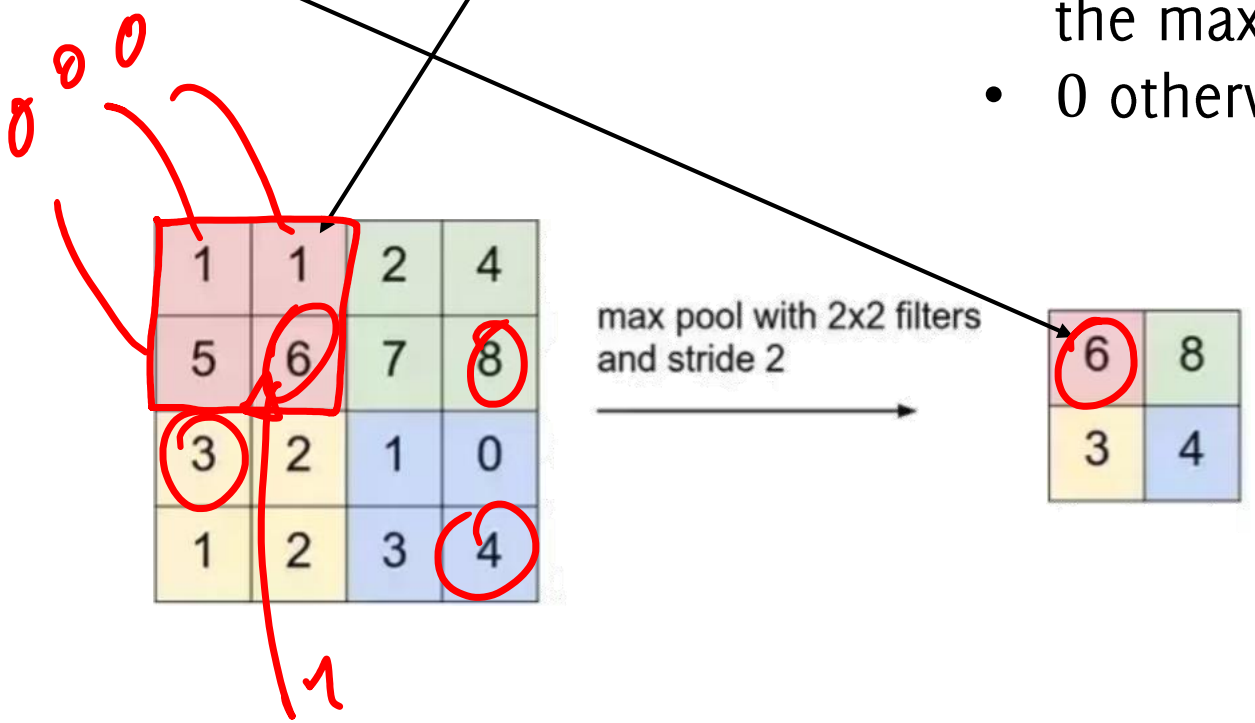
Training a CNN

- **Each CNN can be seen as a MLP**, with sparse and shared connectivities)
- CNN can be in principle **trained by gradient descent** to minimize a loss function over a batch (e.g. binary cross-entropy, RMSE, Hinge loss..)
- Gradient can be computed by **backpropagation** (chain rule) as long as we can derive each layer of the CNN
- **Weight sharing needs to be taken into account** (fewer parameters to be used in the derivatives) while computing derivatives
- There are just a few details missing

Detail: backprop with max pooling

The gradient is only routed through the input pixel that contributes to the output value; e.g.:

Gradient of • with respect to • = 0

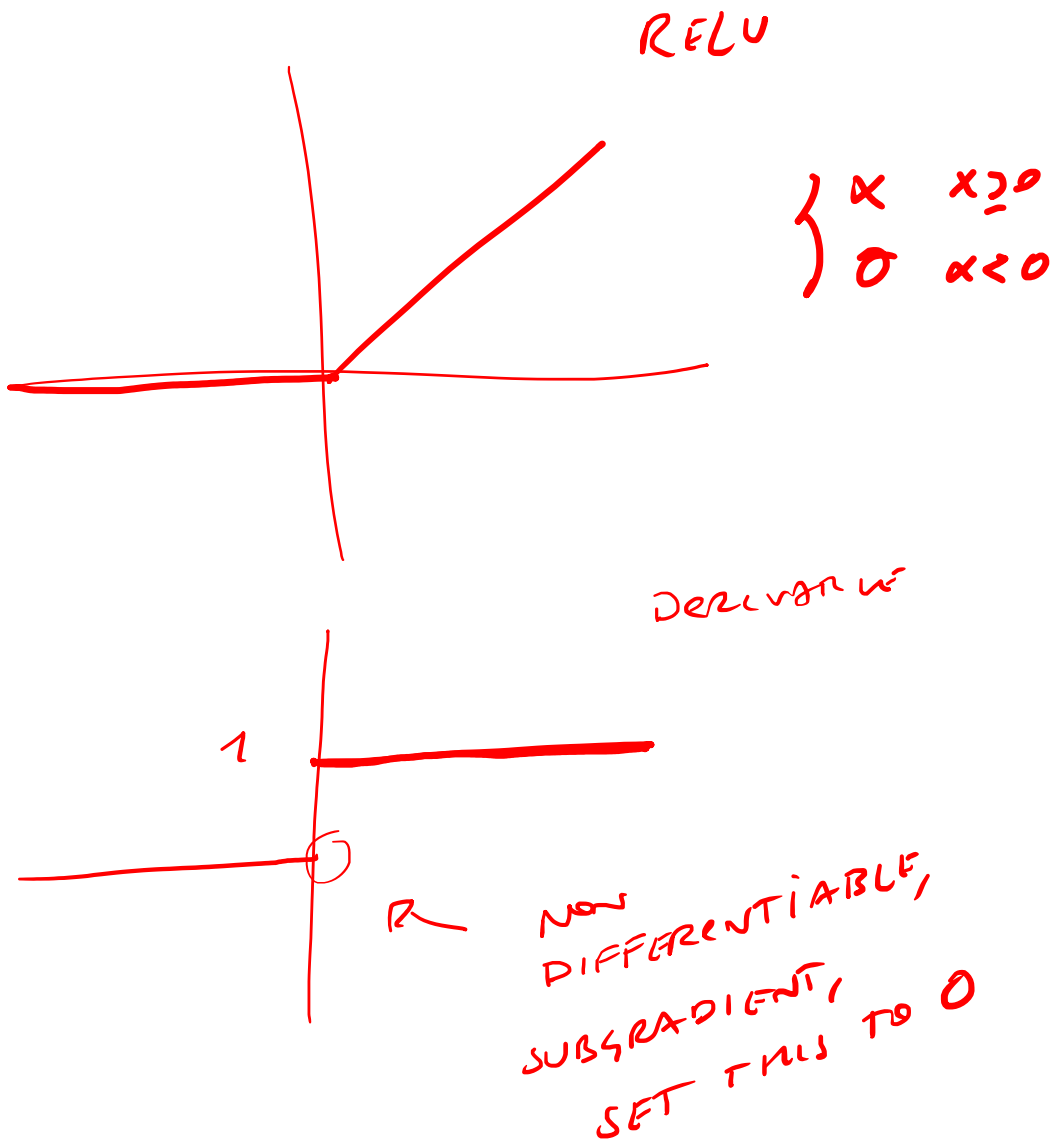


The derivative is:

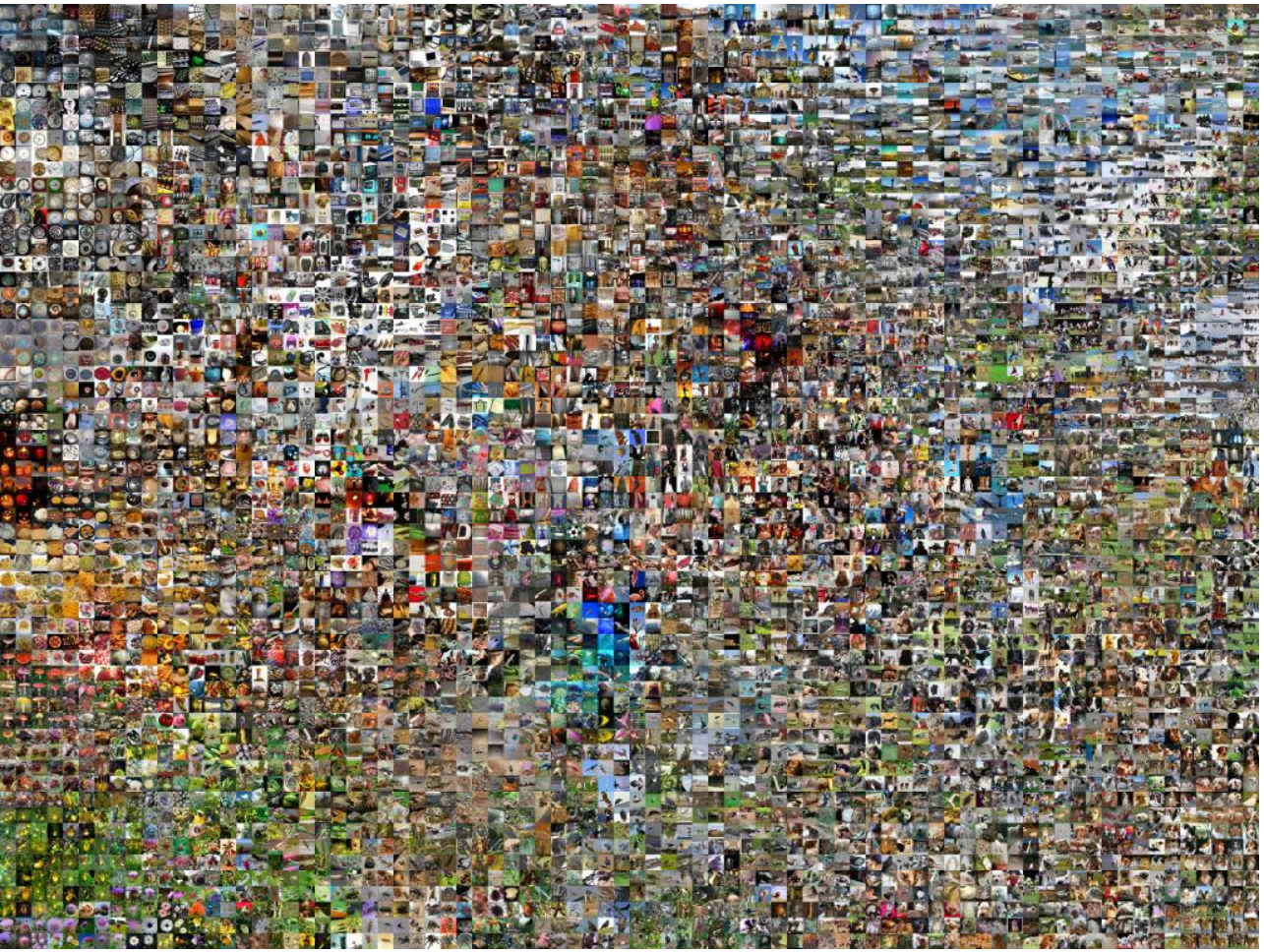
- 1 at the location corresponding the maximum
- 0 otherwise

Detail: derivative of ReLU

The ReLU derivative is straightforward



ImageNet



The ImageNet project is a large visual database designed for use in visual object recognition software research. ***More than 14 million images*** *have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.[3]* ***ImageNet contains more than 20,000 categories***

From Wikipedia October 2021

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, **ImageNet: A Large-Scale Hierarchical Image Database.** *CVPR, 2009.*

Parallel Computing Architectures

Data Scarcity

Training a CNN with Limited Aumont of Data

Giacomo Boracchi

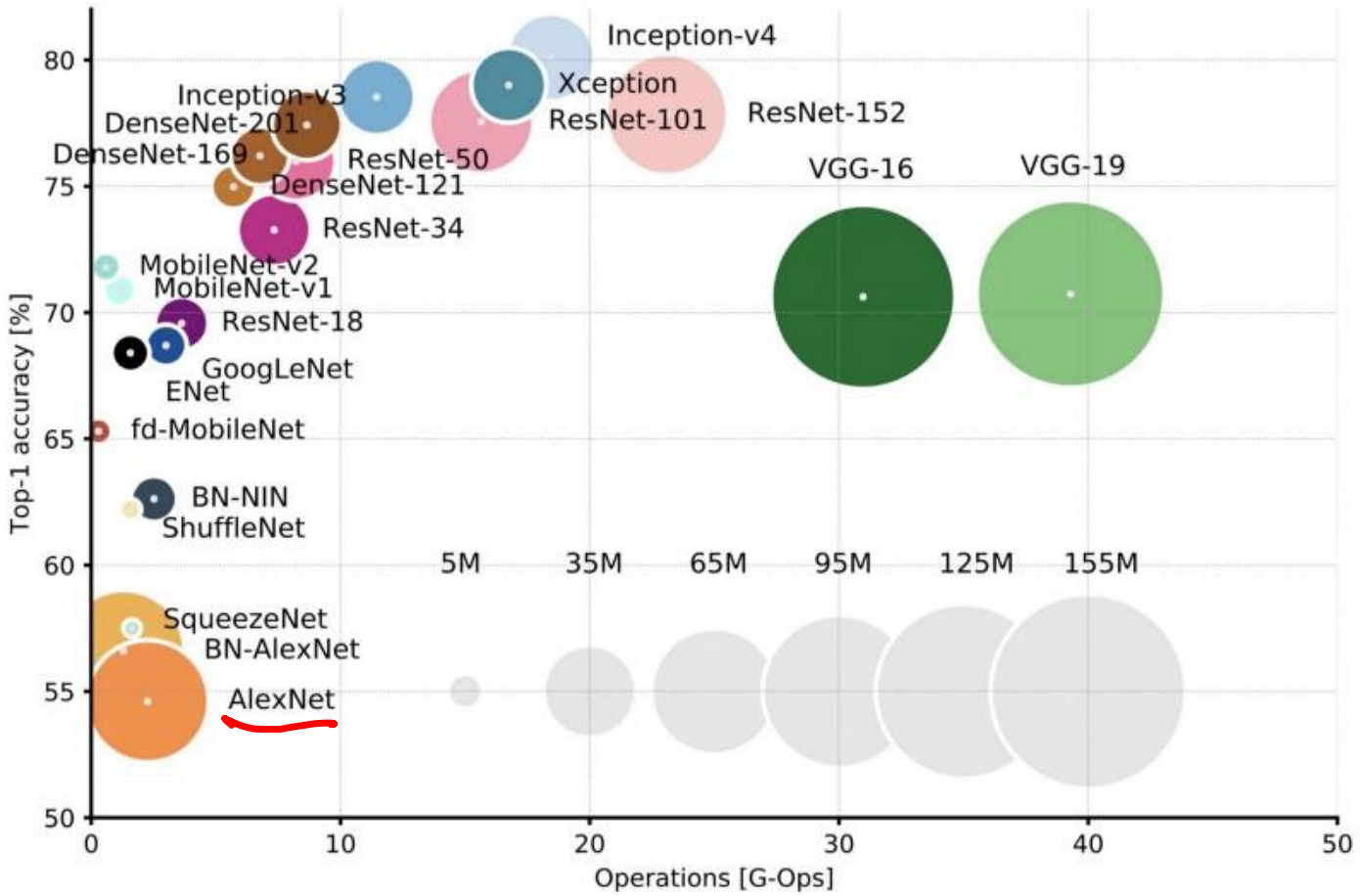
The need of data

Deep learning models are very data hungry.

Networks such as AlexNet have been trained on ImageNet datasets containing tens of thousands of images over hundreds of classes

The need of data

This is necessary to define millions of parameters characterizing these networks



Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).

The need of data

Deep learning models are very data hungry.

… watch out: each image in the training set have to be annotated!

How to train a deep learning model with a few training images?

- Data augmentation
- Transfer Learning

Limited Amount of Data: Data Augmentation

Training a CNN with Limited Aumont of Data

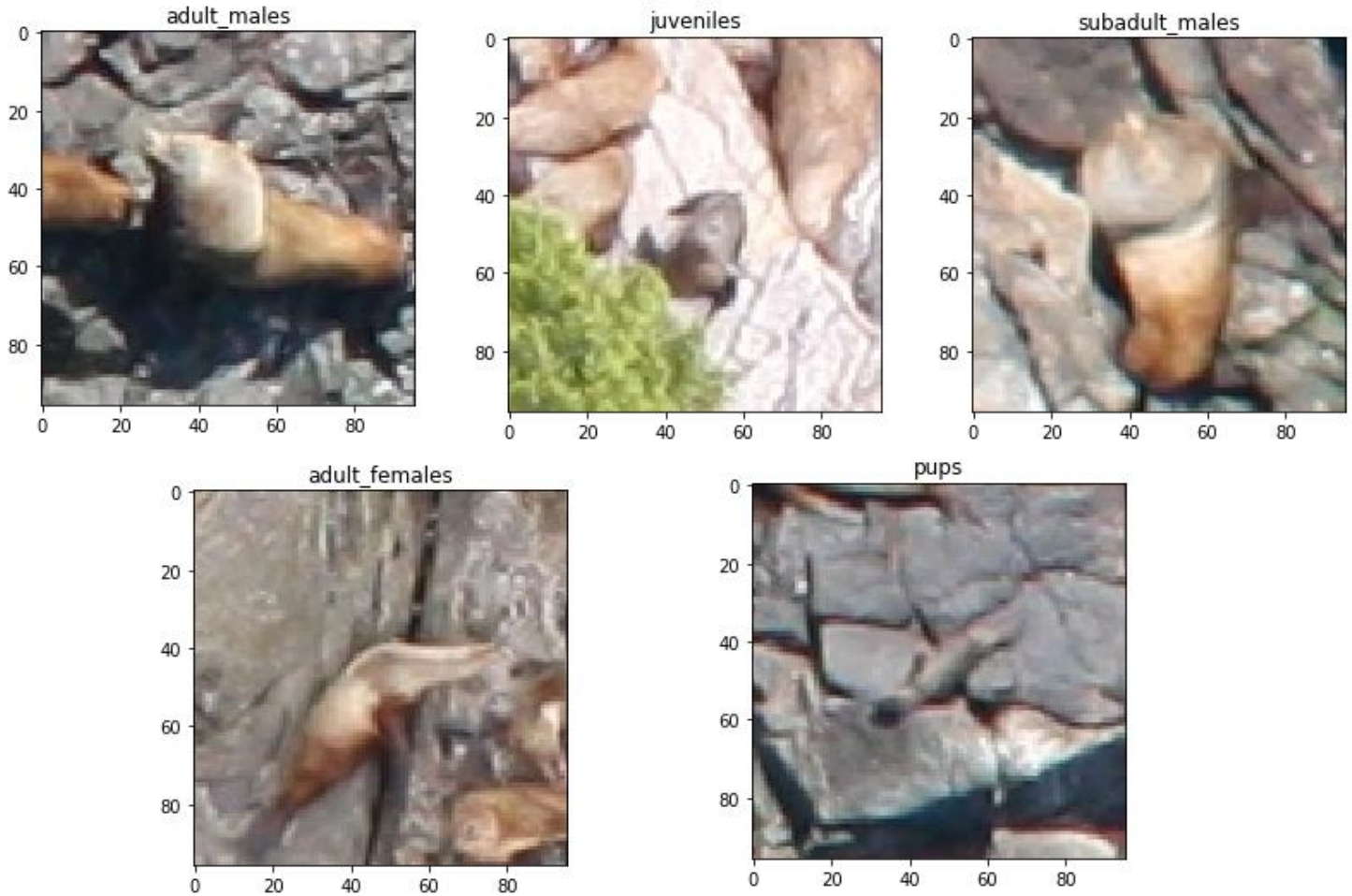
Giacomo Boracchi

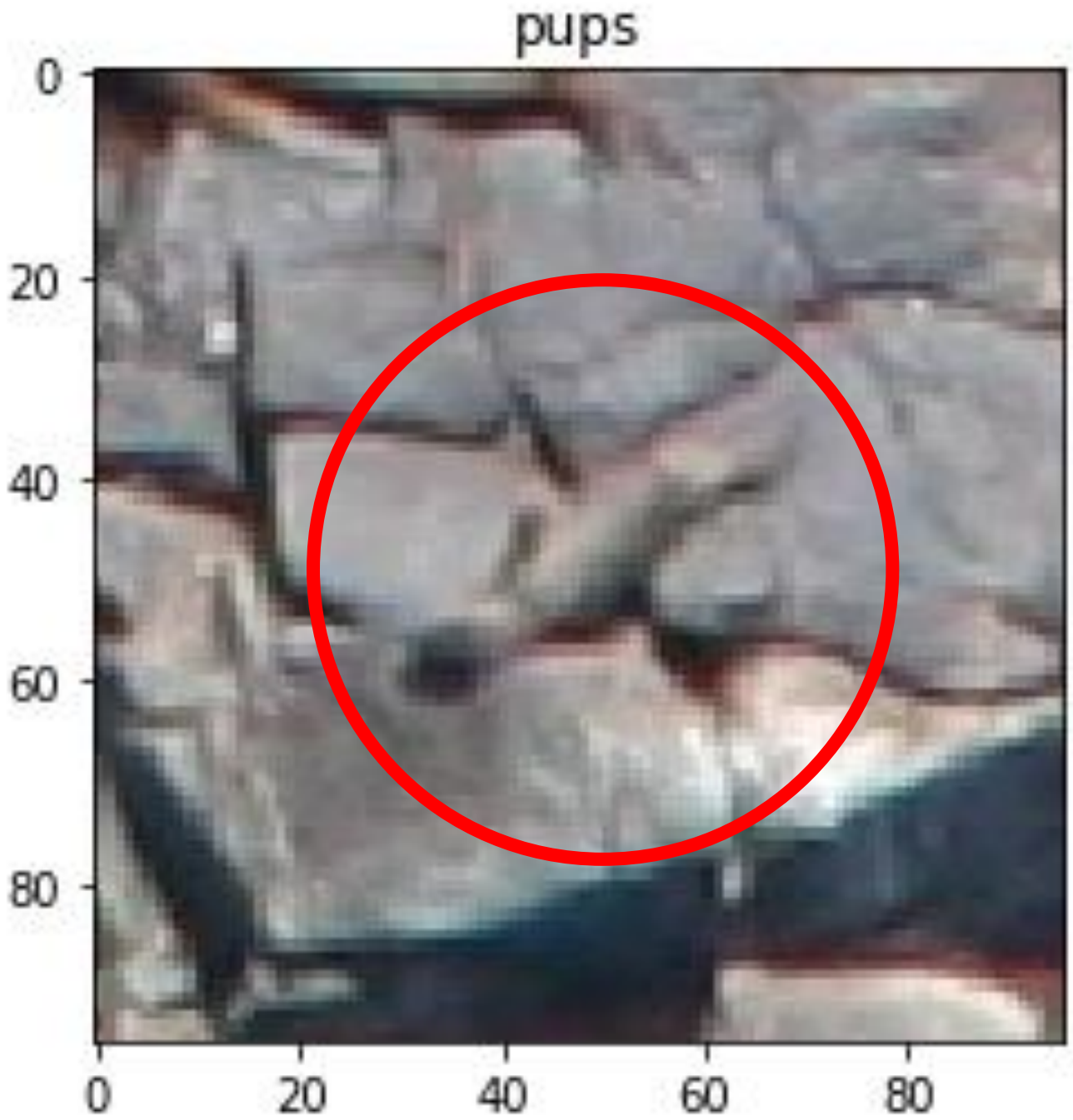

Canada
Stati Uniti
Messico
Aleutian Islands
Steller sea lions in the western Aleutian Islands have declined 94% in the last 30 years.
3D


Kaggle in 2017 have opened a competition to develop algorithms which accurately count the number of sea lions in aerial photographs
Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

The Challenge

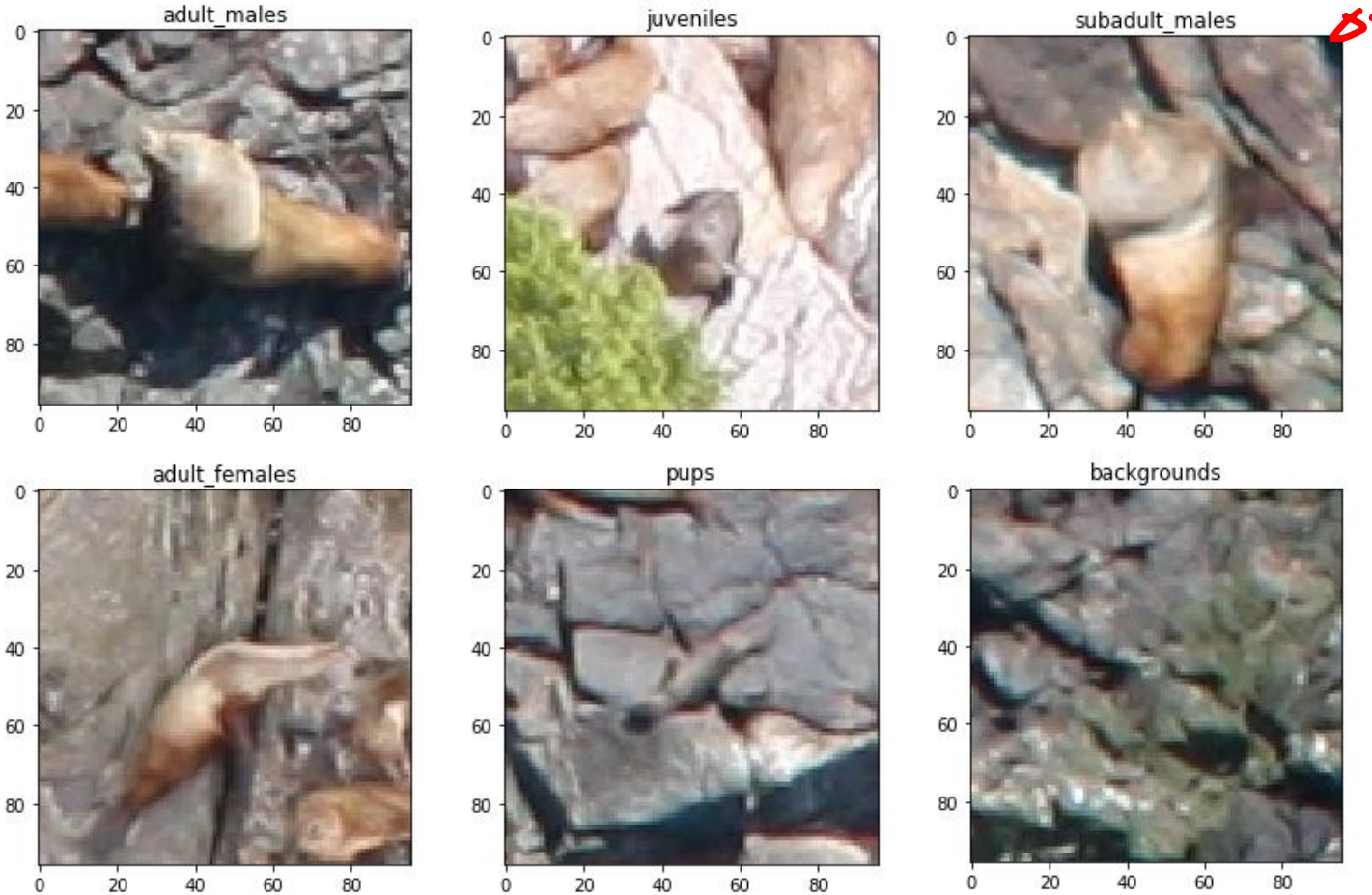
In very large aerial images (≈ 5K x 4K) shot by drones, automatically count the number of sealions per each category



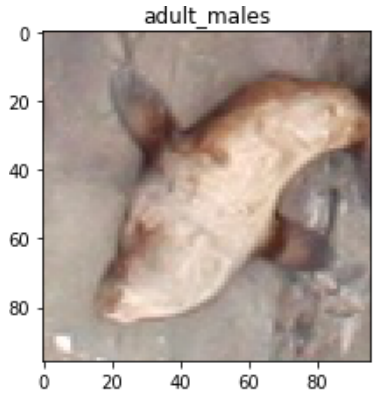

pups
0
20
40
60
80
0
20
40
60
80

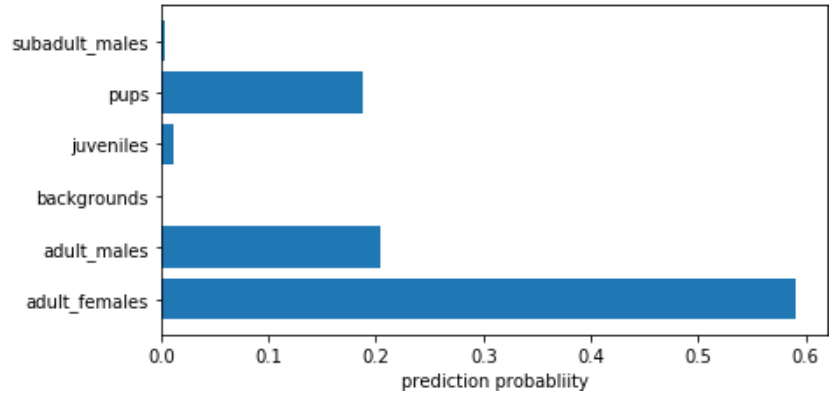
The Challenge

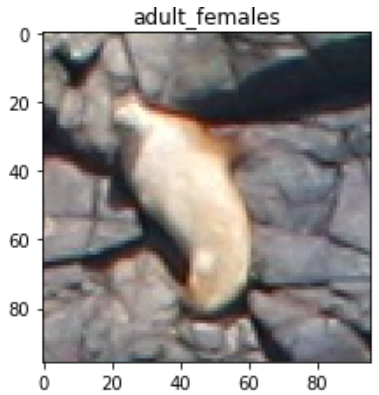
This problem can be naively casted in a patch-by-patch 6-class classification problem, where we include also background

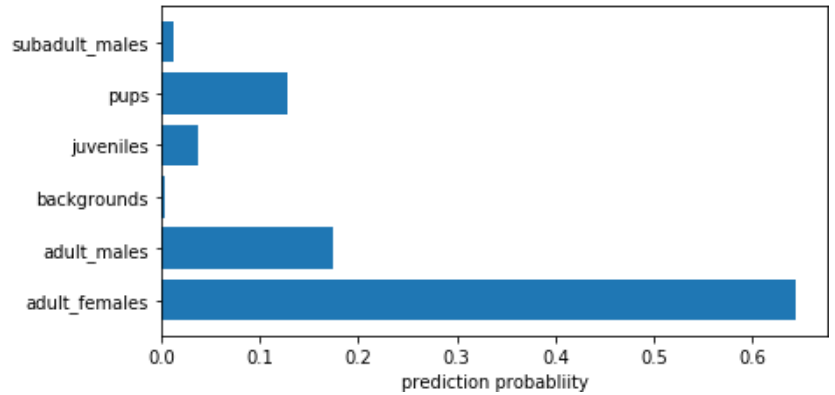


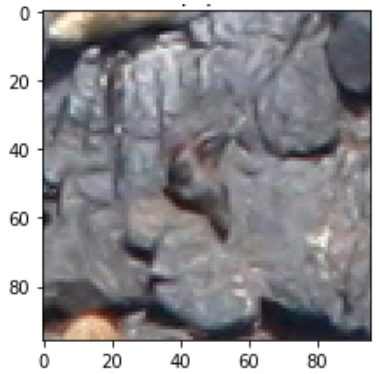
An Example of CNN predictions

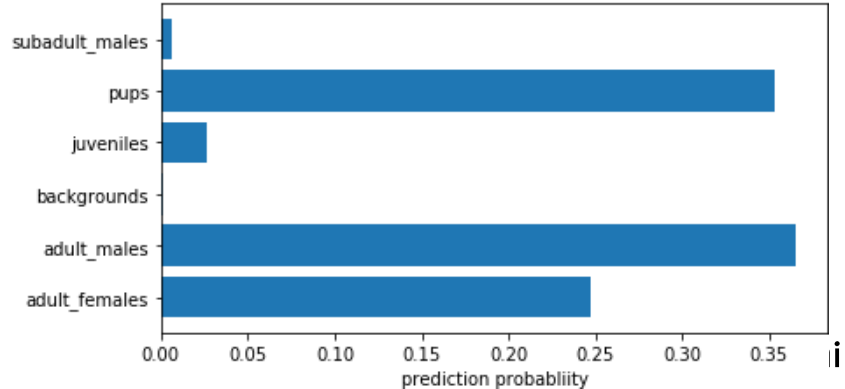












Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

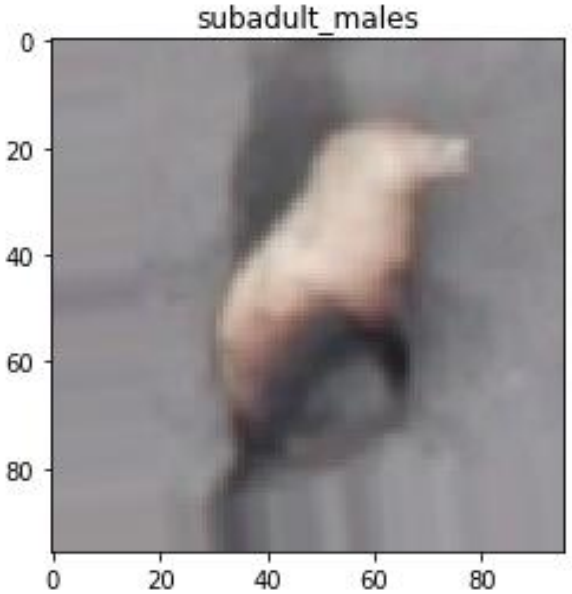
Data Augmentation

Often, each annotated image represents a class of images that are all likely to belong to the same class

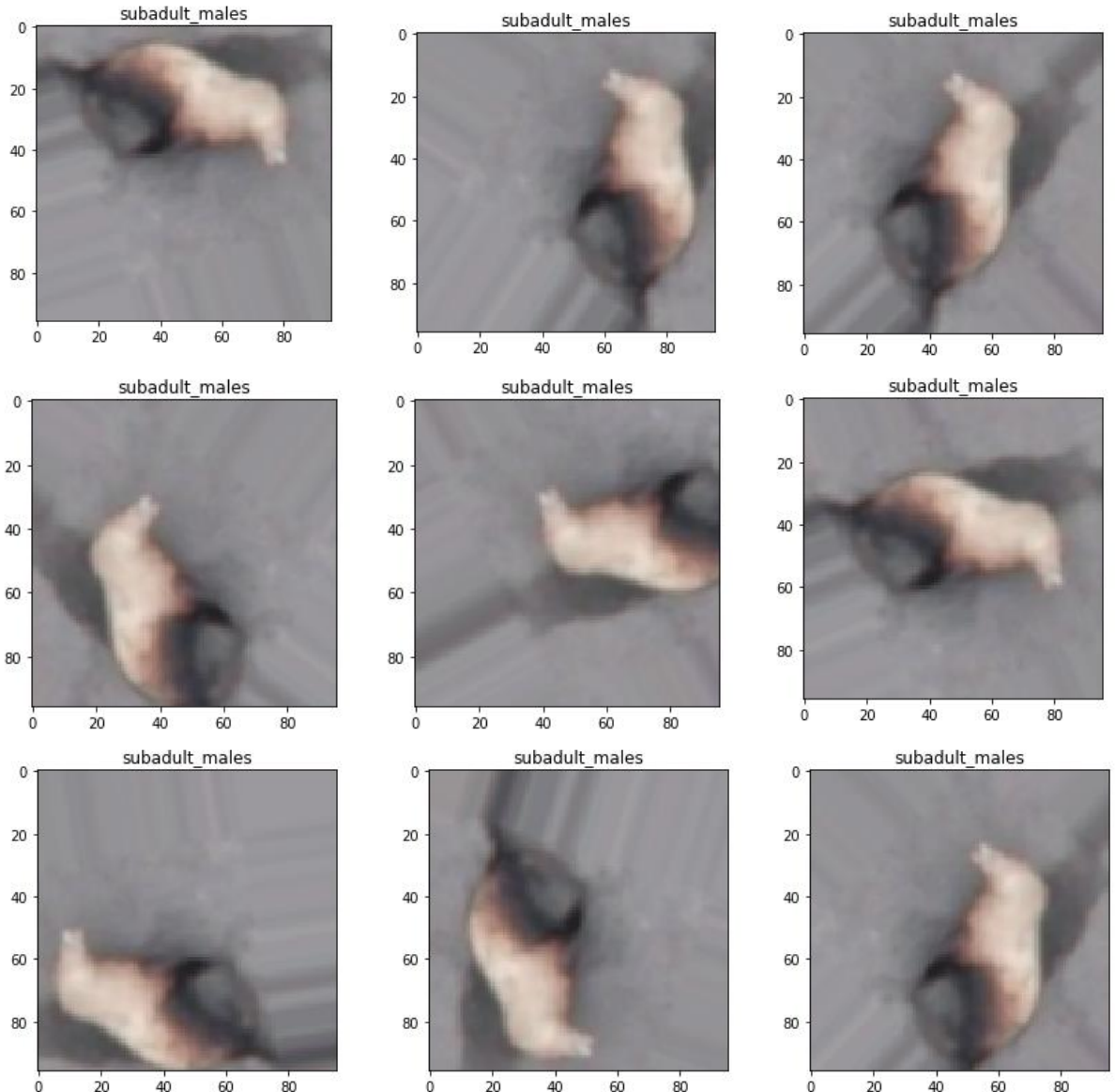
In aereal photograps, for instance, it is normal to have rotated, shifted or scaled images without changing the label

Data Augmentation

Original image



Augmented Images



Data Augmentation

Data augmentation is typically performed by means of

Geometric Transformations:

- Shifts /Rotation/Affine/perspective distortions
- Shear
- Scaling
- Flip

Photometric Transformations:

- Adding noise
- Modifying average intensity
- Superimposing other images
- Modifying image contrast

DO NOT CHANGE IMAGE LABEL

STEER YOUR NETWORK TRAINING TOWARDS INVARIANCE W.R.T. TRANSFORMATIONS

Augmentation in Keras

```
from keras.preprocessing.image import
ImageDataGenerator

ImageDataGenerator(
rotation_range=0,
width_shift_range=0.0, height_shift_range=0.0,
brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0,
fill_mode='nearest',
horizontal_flip=False, vertical_flip=False,
rescale=None,
preprocessing_function=None)
```

Augmentation in Keras: flow from images

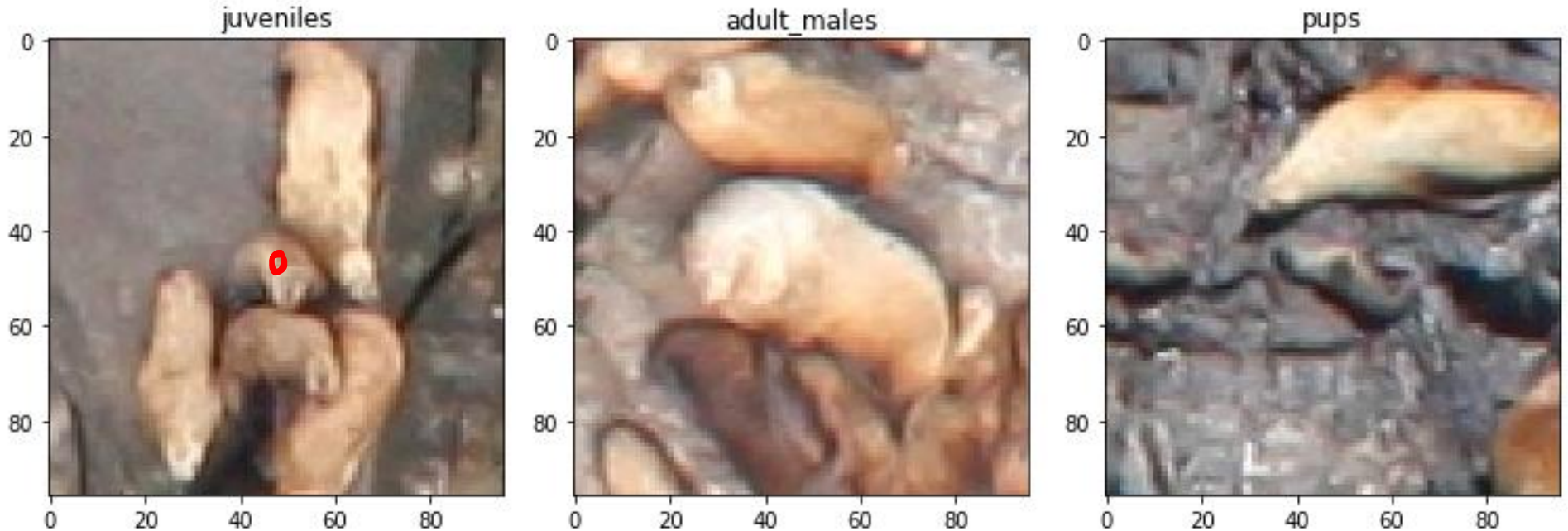
The Image generator has a method **`flow_from_directory`** that allows to load images in folder where different classes are arranged in subfolders.

```
ImageDataGenerator.flow_from_directory(
    directory=PATCH_PATH + 'train/',
    target_size=(img_width, img_width),
    batch_size=batch_size,
    shuffle=True)
```

However...

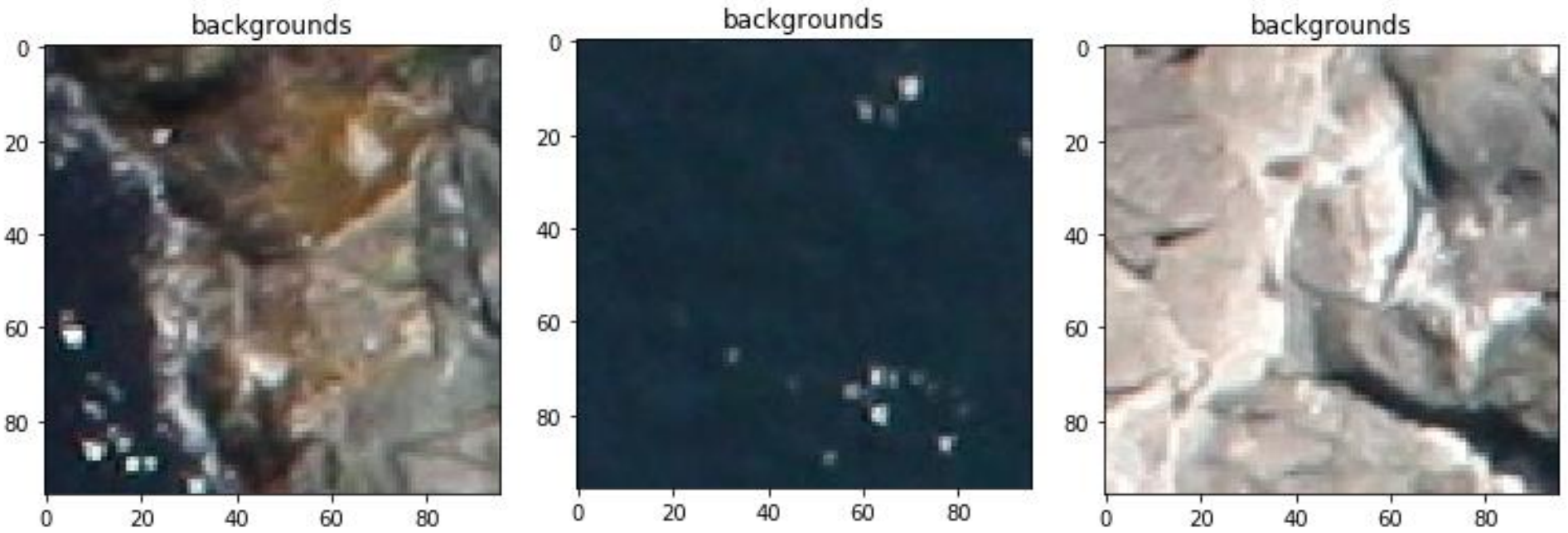
This sort of data augmentation might not be enough to capture the inter-class variability of images...

Superimposition of targets



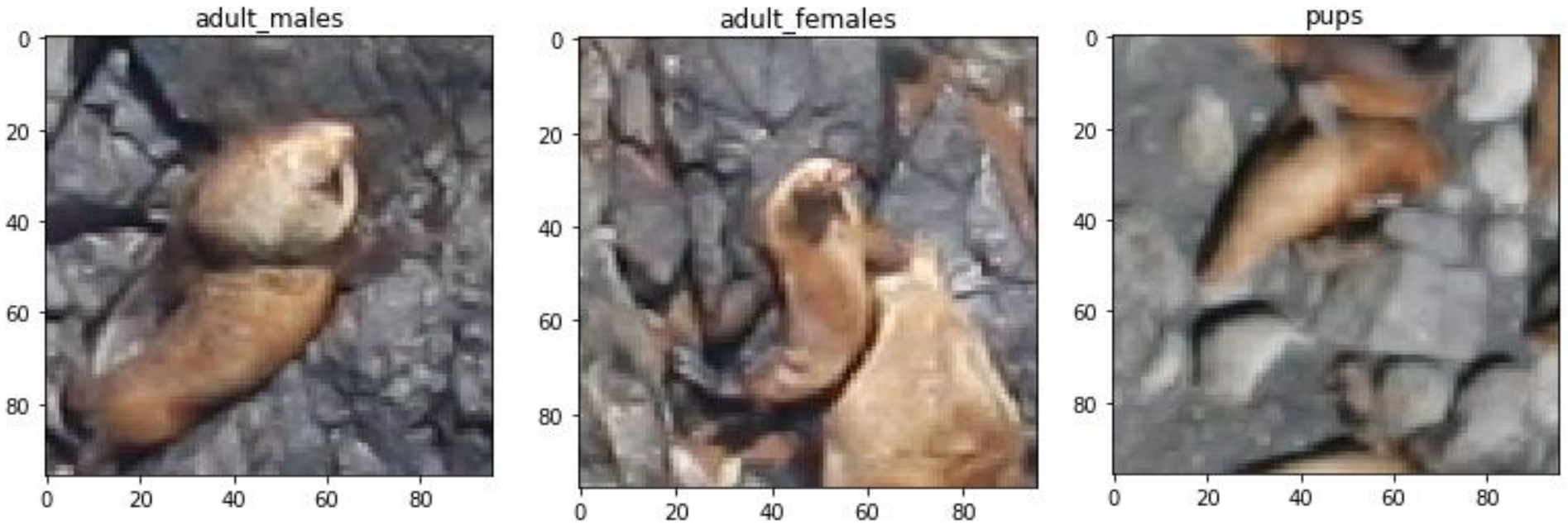
However…

Background variations



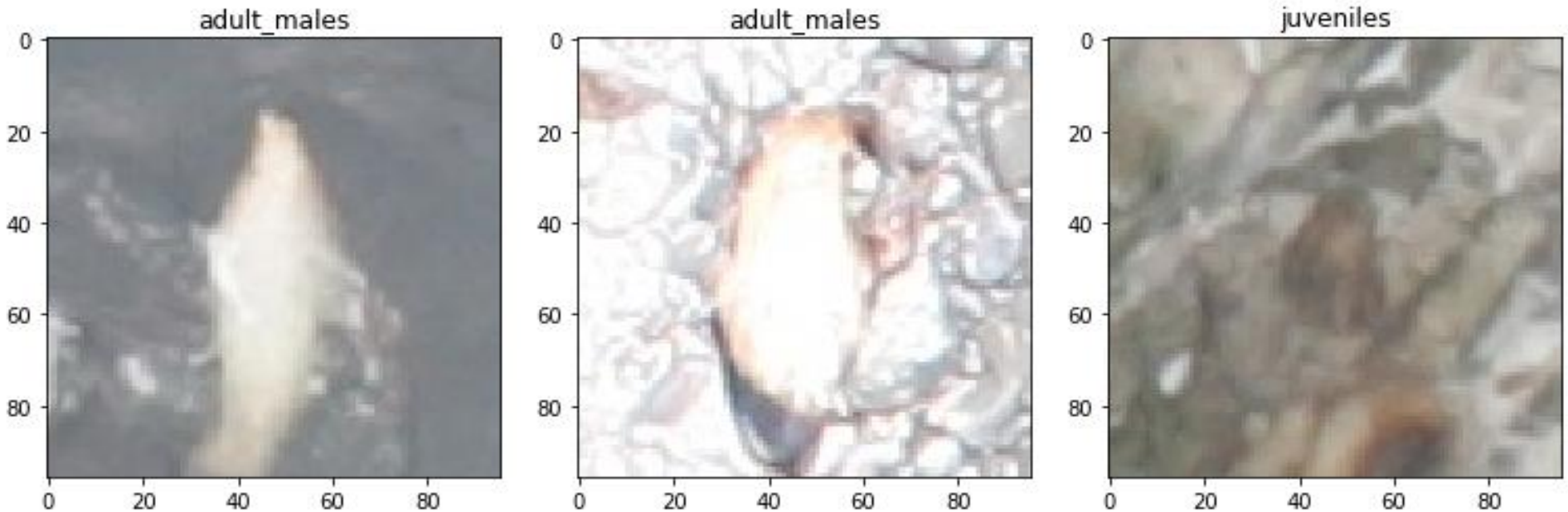
However…

Background variations



However…

Out of focus, bad exposure



Augmentation in Keras

```
from keras.preprocessing.image import
ImageDataGenerator

ImageDataGenerator(
rotation_range=0,
width_shift_range=0.0, height_shift_range=0.0,
brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0,
fill_mode='nearest',
horizontal_flip=False, vertical_flip=False,
rescale=None,
preprocessing_function=None)
```

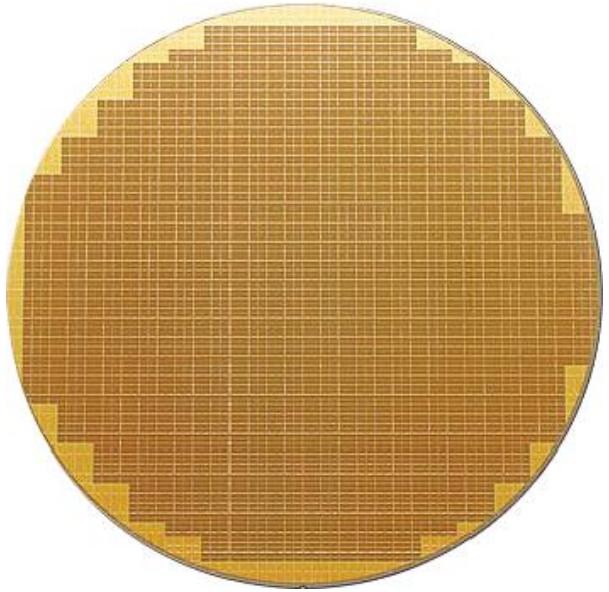
… in case you need some extra flexibility

Data Augmentation is often key..

Collaboration with

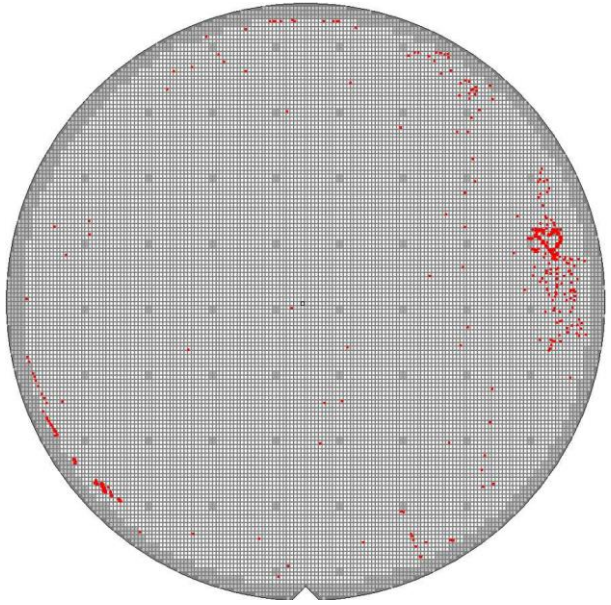
For instance to monitor wafer manufacturing

Wafer (containing chips)



Inspection tool

Wafer Defect Map



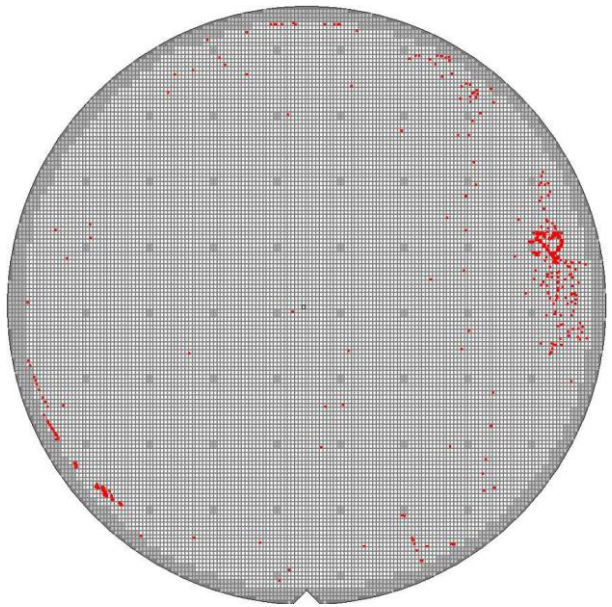
20 K × 20 K

R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks" ICIAP 2019

Data Augmentation is often key..

Collaboration with life.augmented

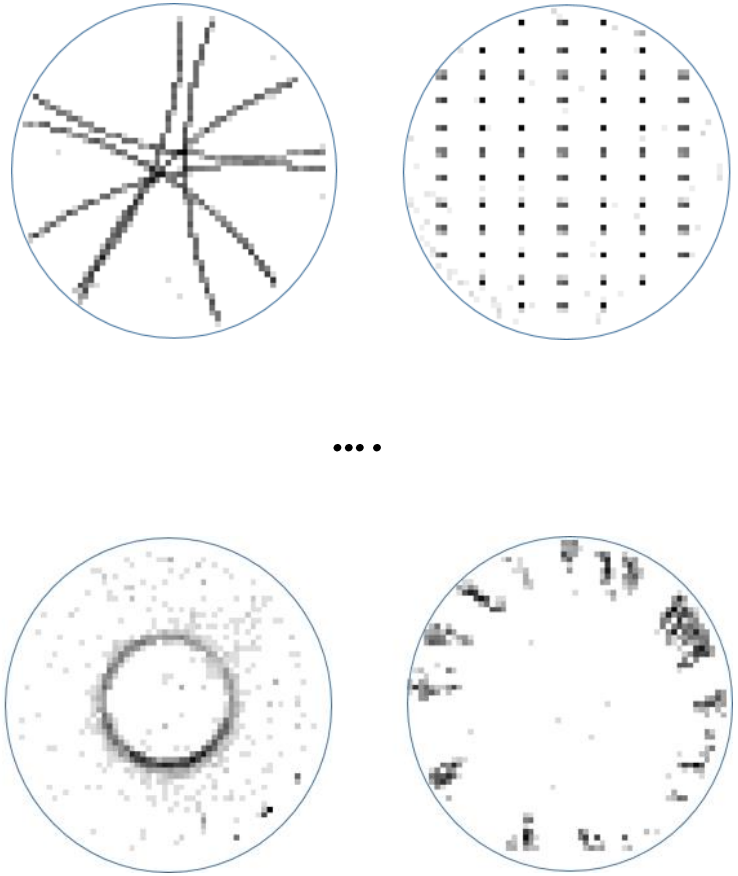
Train a deep learning model to identify defective patterns



Wafer Defect Map

Deep Learning

Defect Patterns



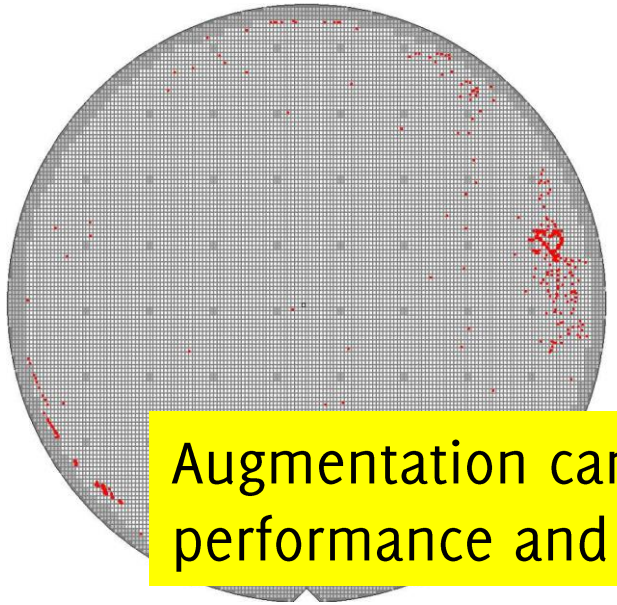
R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks" ICIAP 2019

Data Augmentation is often key..

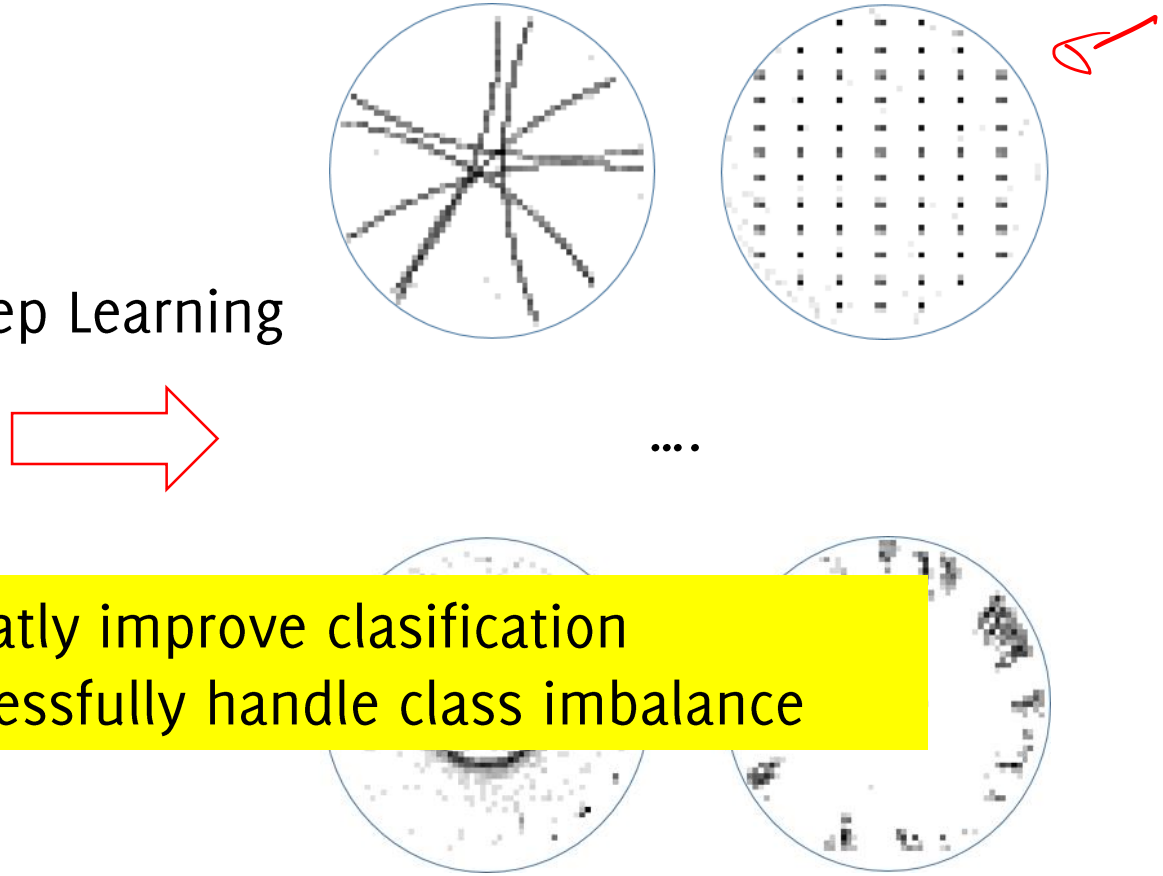
Collaboration with life.augmented

Train a deep learning model to identify defective patterns

Defect Patterns



Deep Learning



Augmentation can greatly improve clasification performance and successfully handle class imbalance

Wafer Defect Map

R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks" ICIAP 2019

Test Time Augmentation (TTA)

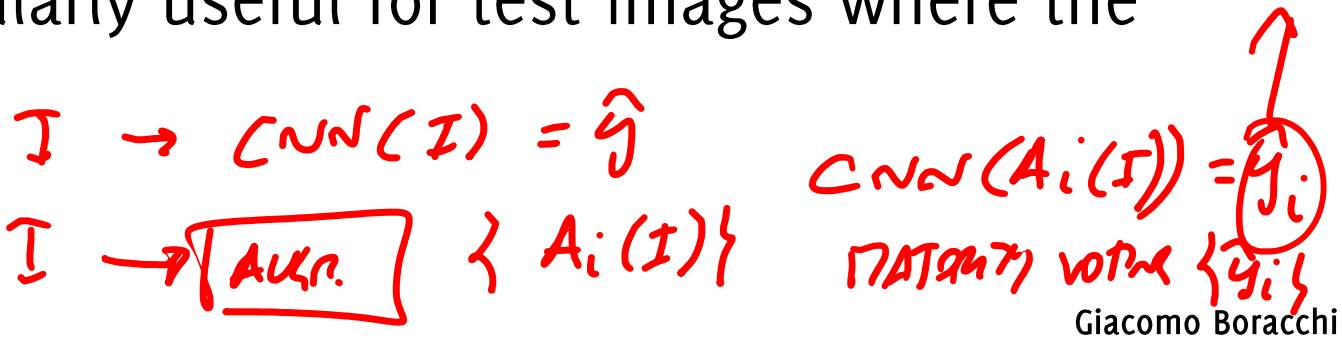
Even if the CNN is trained using augmentation, **it won't achieve perfect invariance** w.r.t. considered transformations

Test time augmentation (TTA): augmentation can be also performed at test time to improve prediction accuracy.

"self ensemble"

- Perform random augmentation of each test image I
- Average the predictions of each augmented image
- Take the average vector of posterior for defining the final guess

Test Time Augmentation is particularly useful for test images where the model is pretty unsure.



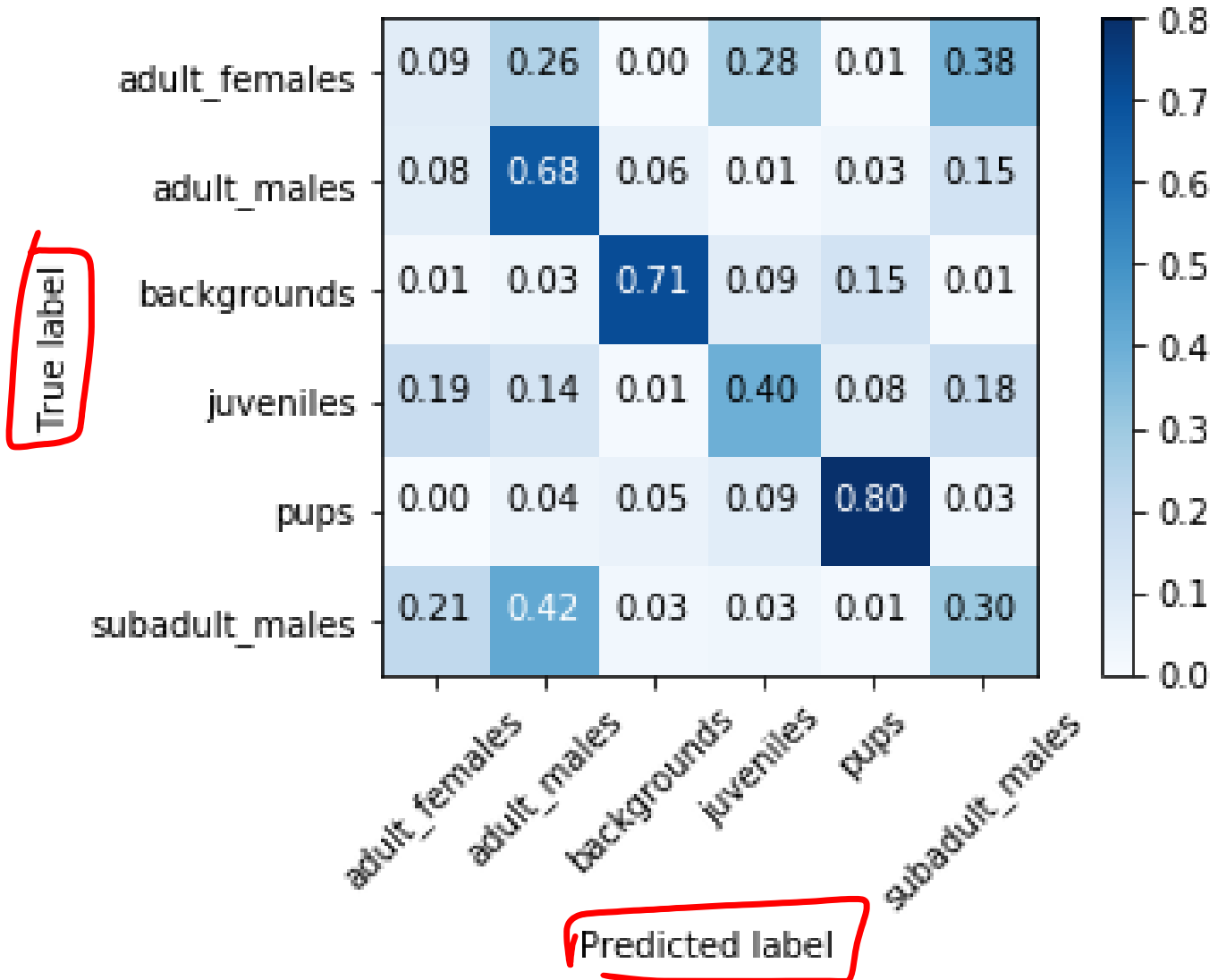
A bit more of background

Performance measures

and an overview of successful architectures

Giacomo Boracchi

Confusion Matrix

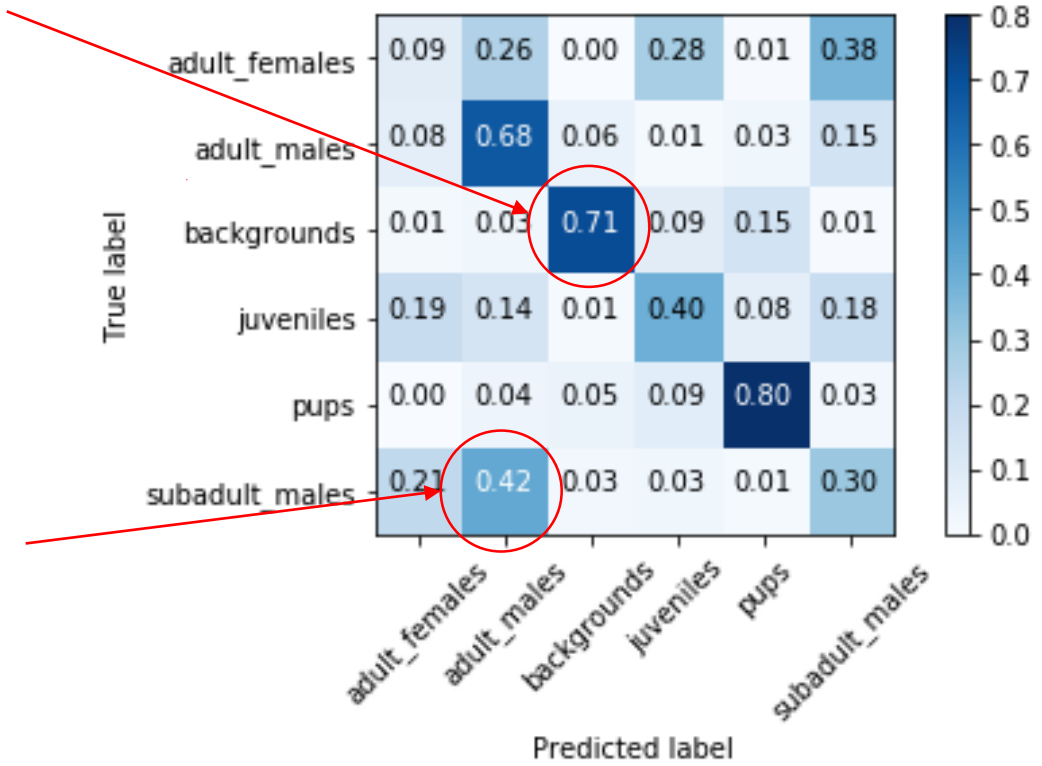


Confusion Matrix

The element $C(i,j)$ i.e. at the i-th row and j-th column corresponds to the percentage of elements belonging to class i classified as elements of class j

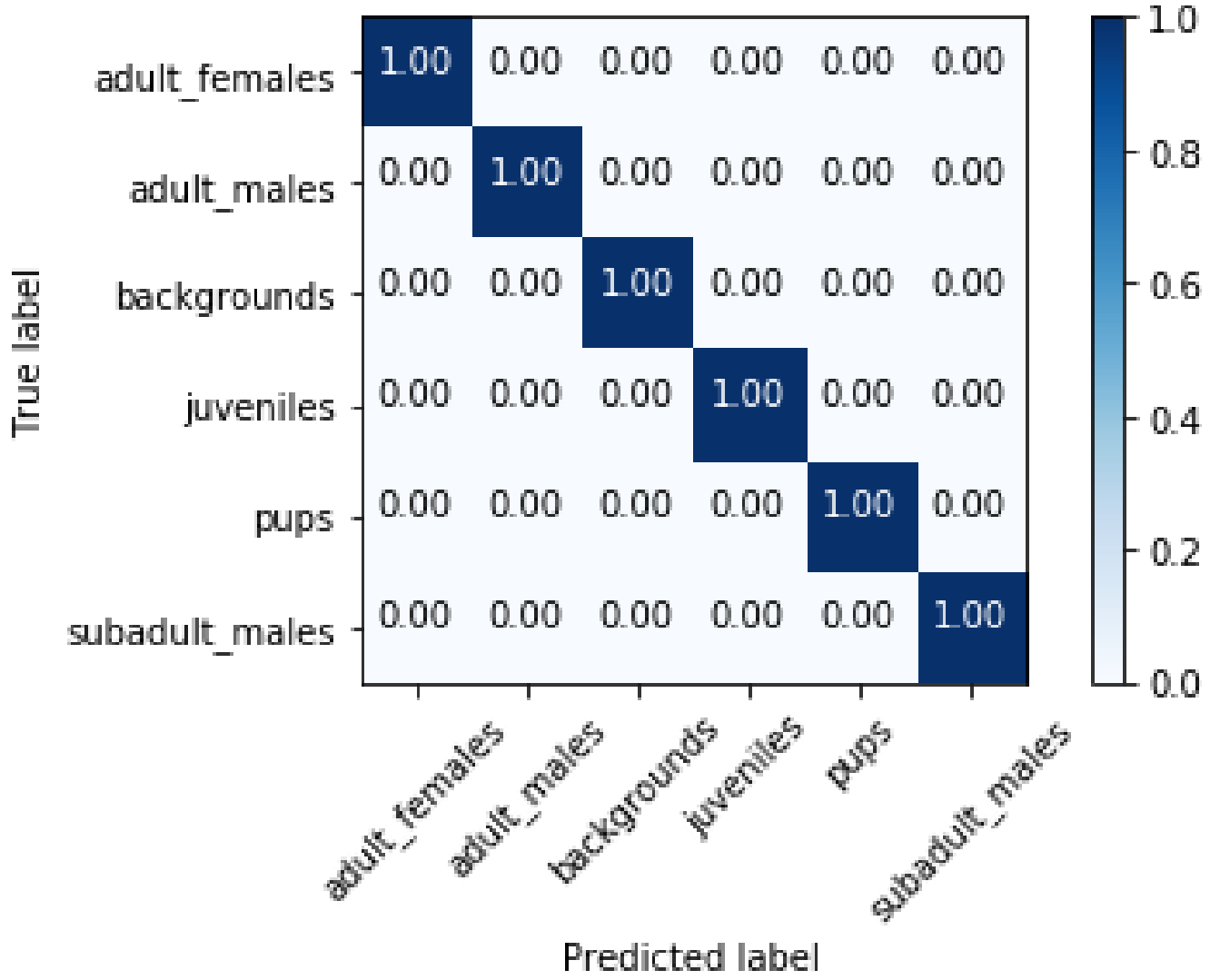
71% of background patches have been correctly classified as background

42% of sub-adult males patches have been wrongly classified as adult-males



… so, the ideal confusion matrix

Which rarely happens



Two-Class Classification

Classification performance in case of binary classifiers can be also measured in terms of the **ROC** (receiver operating characteristic) **curve**, which does not depend on the threshold you set for each class

This is useful in case you plan to modify this and not use 0.5

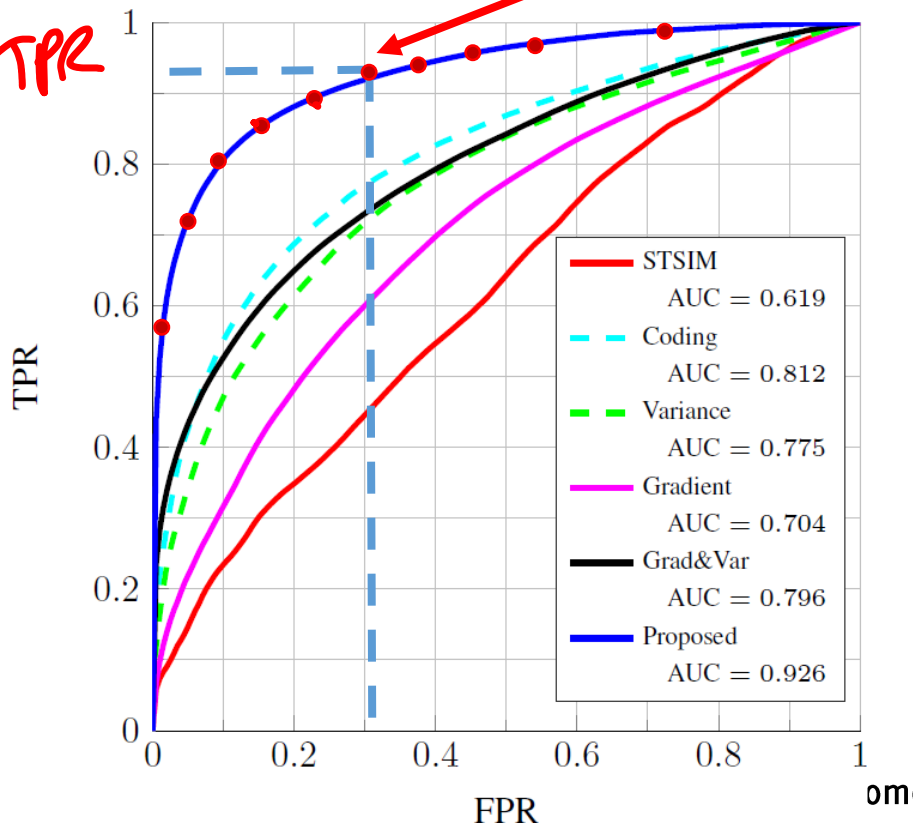
The ideal detector would achieve:

- $FPR = 0\%$,
- $TPR = 100\%$

Thus, the closer to (0,1) the better

The largest the **Area Under the Curve** (AUC), the better

The optimal parameter is the one yielding the point closest to (0,1)



A few popular architectures

AlexNet

VGG

Outline

Lecture inspired to:

Notes accompanying the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

… and papers cited in here!

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

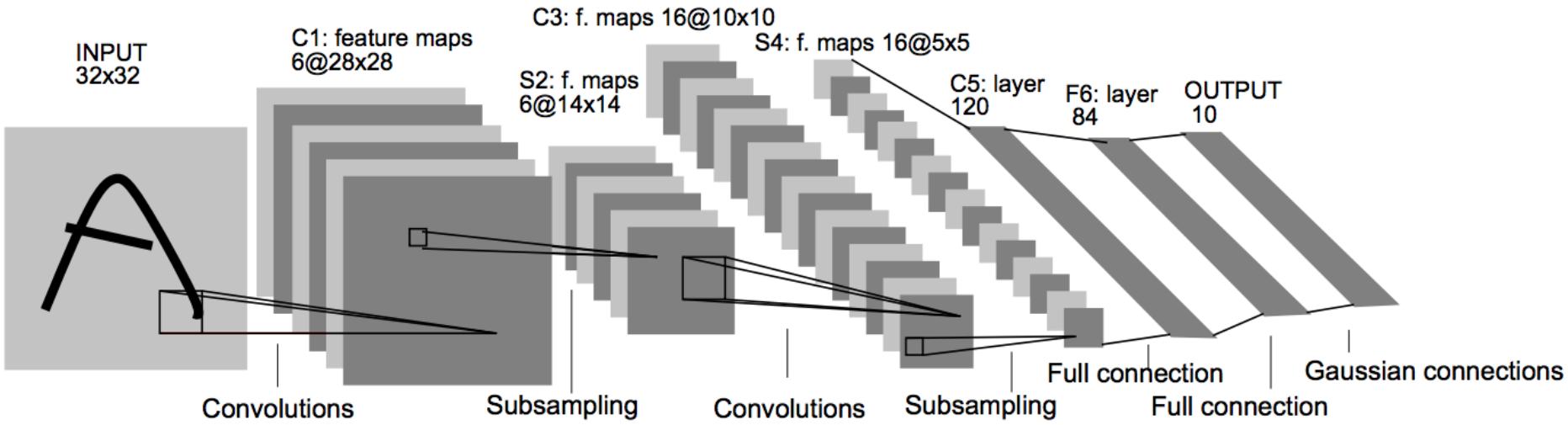
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

LeNet-5 (1998)

Yann Lecun's LeNet-5 model was developed in 1998 to identify handwritten digits for zip code recognition in the postal service.

This pioneering model largely introduced the convolutional neural network as we know it today

Parameters: 60.000



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

LeNet-5 (1998)

The idea of using a sequence of 3 layers: **convolution, pooling, non-linearity.**

Pixels in an image are highly correlated: there is no point to separately feed each pixel value through a classification network: add learnable convolutional layers

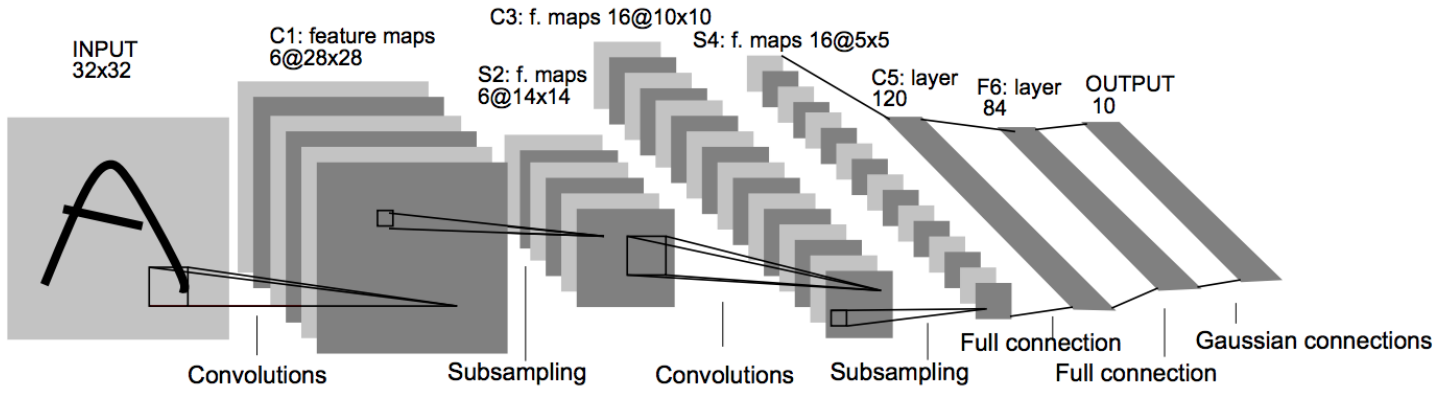
Convolution as a way to extract spatial features and to have sparse connections (thus reducing the number of parameters).

Subsample using spatial average of maps (averaging pooling)

Non-linearities: tanh or sigmoids

MLP as classifier for the final layer

60k



LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 1998*

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

Fathers of the Deep Learning Revolution Receive ACM A.M. Turing Award

Bengio, Hinton and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

https://awards.acm.org/about/2018-turing

AlexNet (2012)

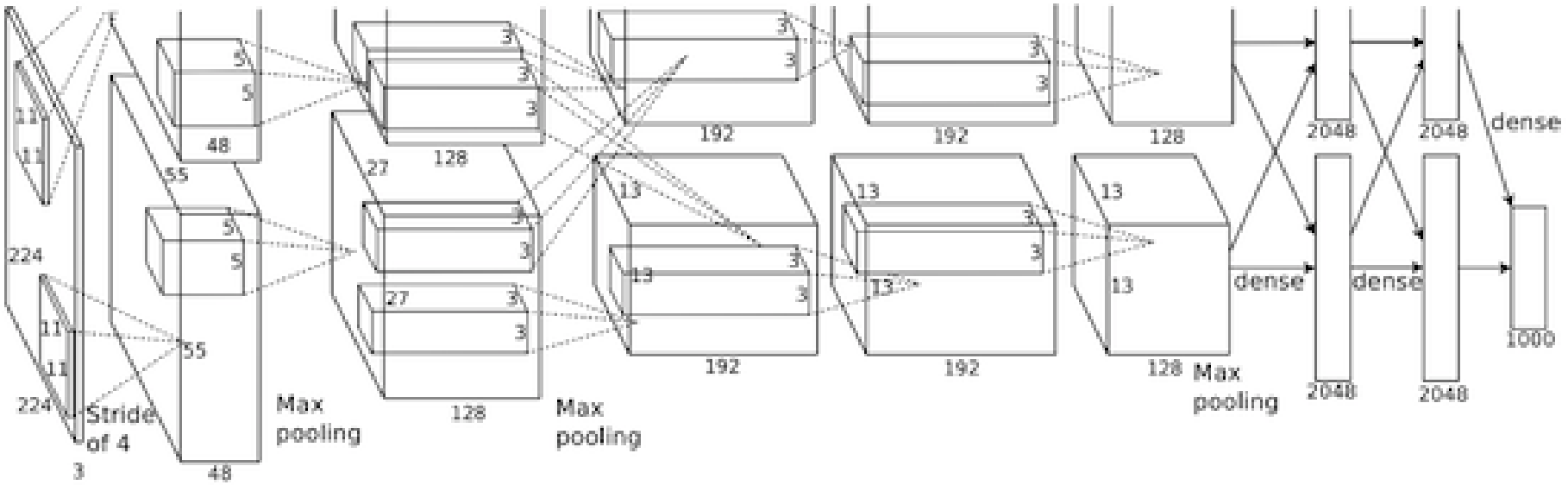
Developed by Alex Krizhevsky et al. in 2012 and won Imagenet competition

Architecture is quite similar to LeNet-5:

- 5 convolutional layers (rather large filters, 11x11, 5x5),
- 3 MLP

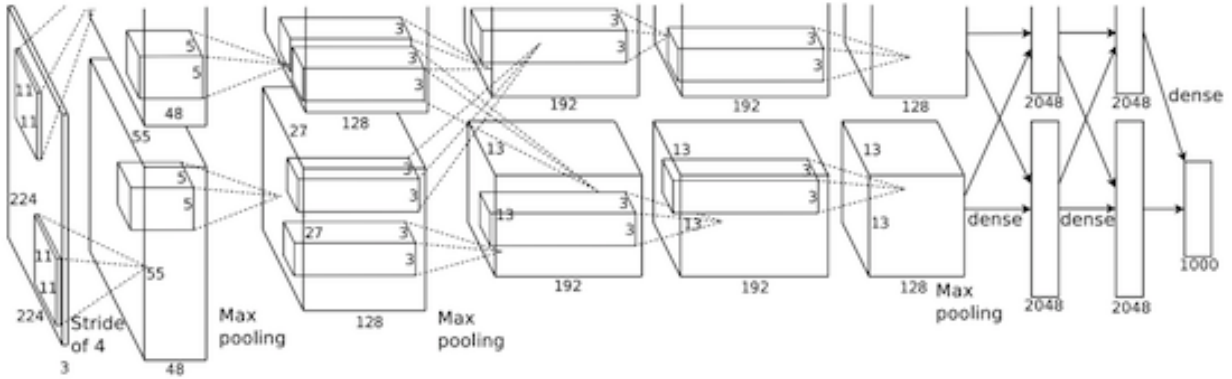
Input size 224 × 224 x 3 (the paper says 227 x 227 x 3)

Parameters: 60 million [**Conv**: 3.7million (6%), **FC**: 58.6 million (94%)]



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

AlexNet (2012)



To counteract overfitting, they introduce:

- RELU (also faster than tanh)
- Dropout (0.5), weight decay and norm layers (not used anymore)
- Maxpooling

The first conv layer has 96 **11x 11 filters, stride 4.**

The output are **two volumes of 55 x 55 x 48 separated over two GTX 580 GPUs** (1.5GB each GPU, 90 epochs, 5/6 days to train).

Most **connections are among** feature maps **of the same GPU**, which will be mixed at the last layer.

Won the ImageNet challenge in 2012

At the end they also trained an **ensemble of 7 models** to drop error: 18.2%->15.4%

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

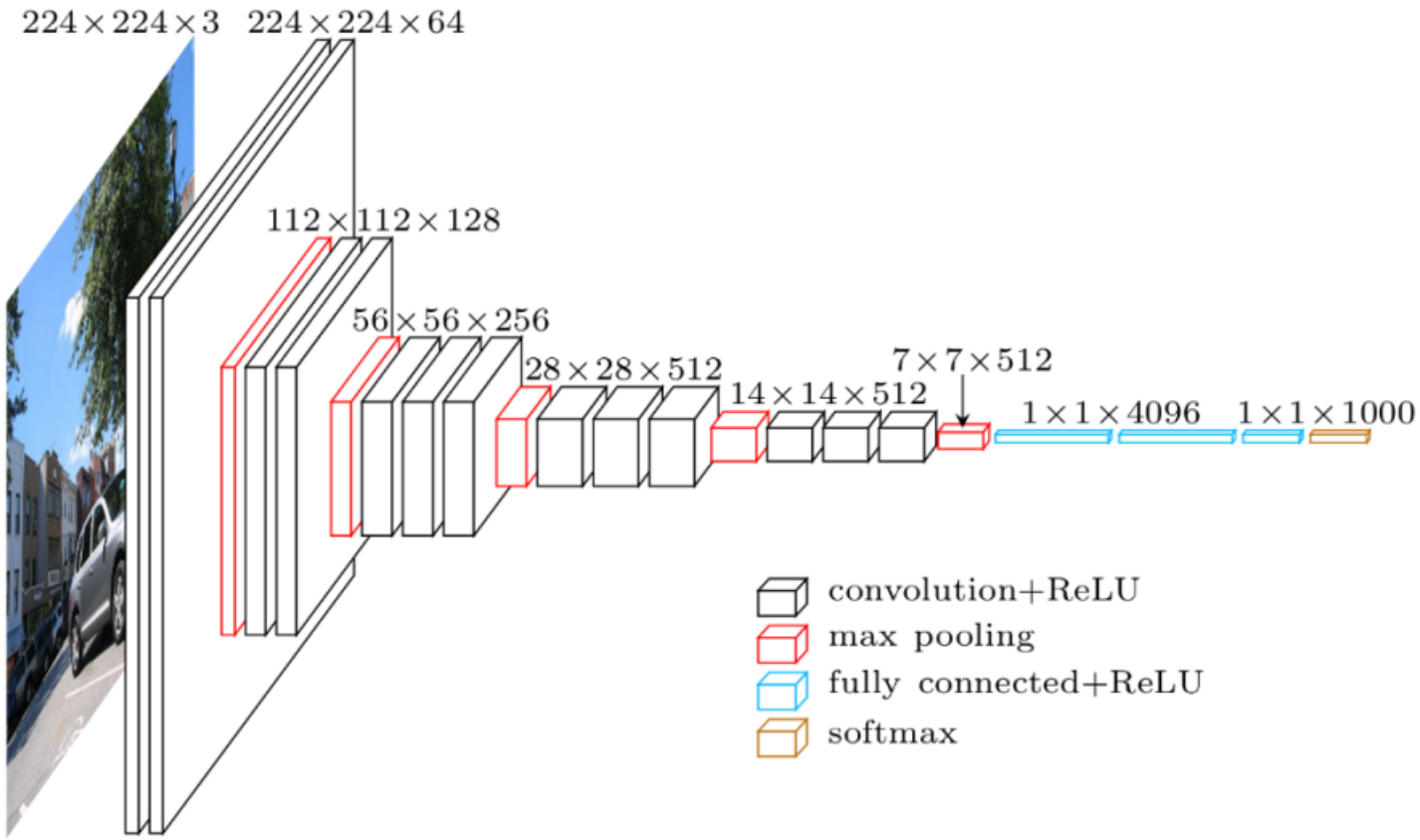
In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

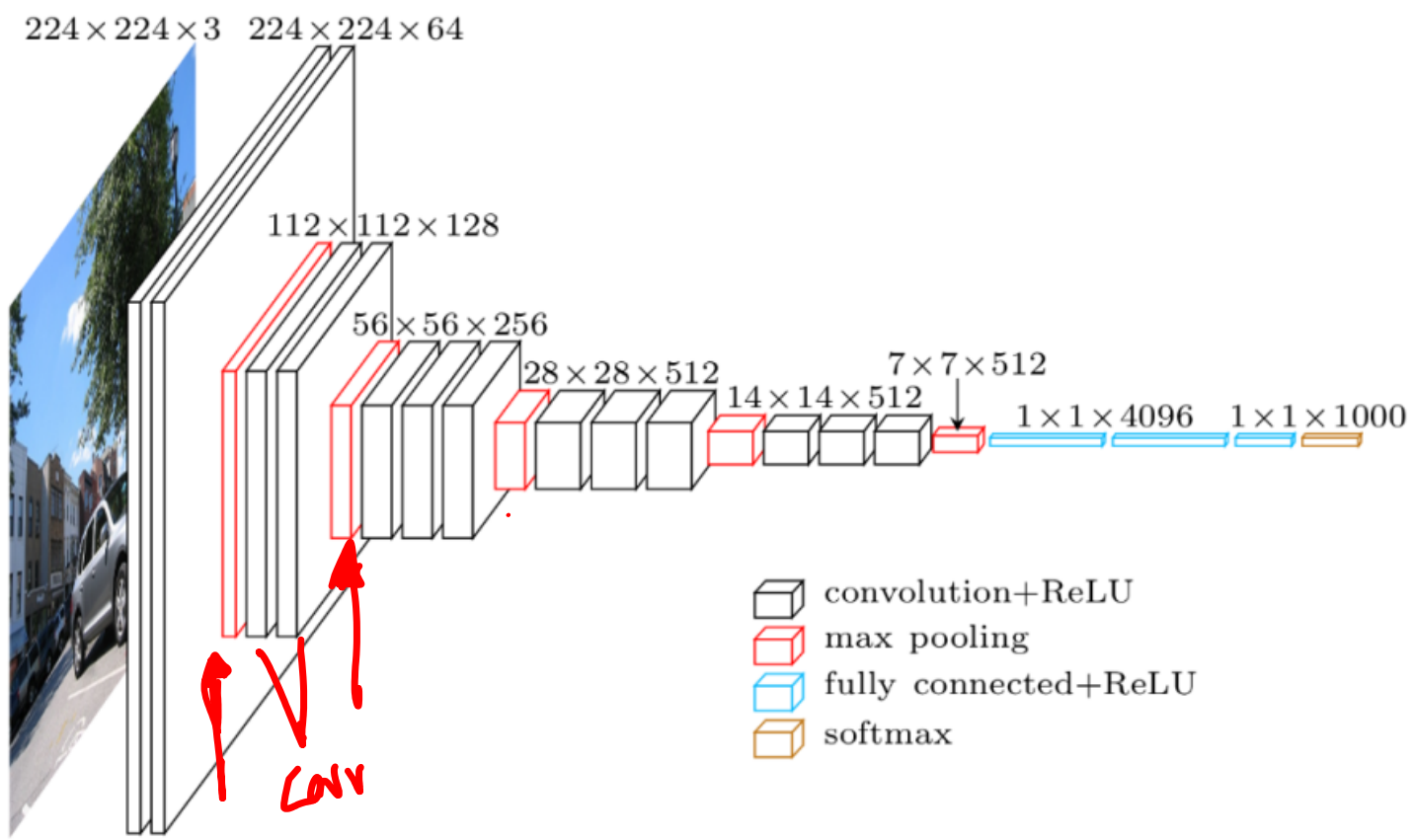
VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]

These architecture **won the first place places (localization) and the second place (classification) tracks in ImageNet Challenge 2014**

Input size 224 × 224 x 3



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014): Smaller Filter, Deeper Network

The paper actually present a thorough **study on the role of network depth**.

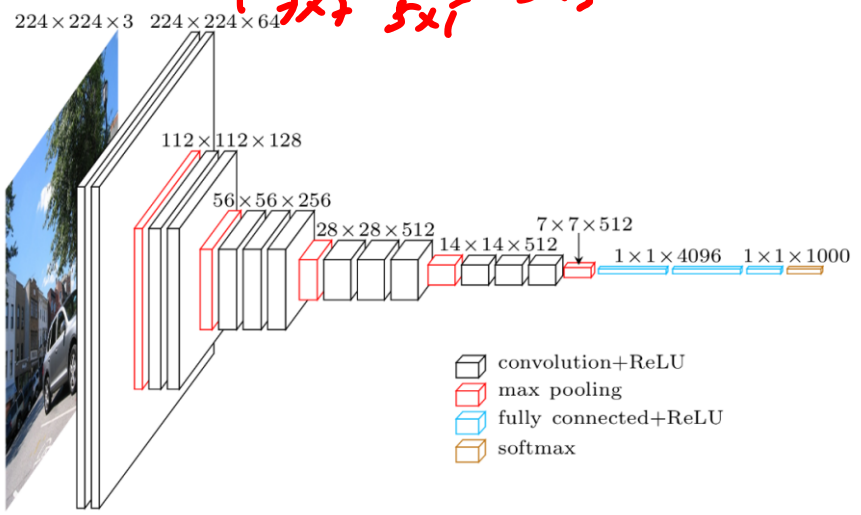
[…]Fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due ***to the use of very small (3. × 3) convolution filters in all layers.***

Idea: Multiple 3×3 convolution in a sequence achieve large receptive fields with:

- less parameters
- more nonlinearities

than larger filters in a single layer

	3 layers 3x3	**1 layer 7x7**
Receptive field	7X7	7X7
Nr of parameters	3 x 3 x 3 = 27	49
Nr of nonlinearities	3	1



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Many convolutional blocks without maxpooling

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

THESE FEATURES ARE VERY GOOD!

Most parameters in FC layers : **123,642,856**

Giacomo Boracchi

VGG16

224 x 224 x 3

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(	
block2_conv2 (Conv2D)	(	
block2_pool (MaxPooling2D)	(	
block3_conv1 (Conv2D)	(	
block3_conv2 (Conv2D)	(	
block3_conv3 (Conv2D)	(	
block3_pool (MaxPooling2D)	(	
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
	14, 14, 512)	2359808
	7, 7, 512)	0
	25088)	0
	4096)	102764544
	4096)	16781312
	1000)	4097000

Trainable params: 138,357,544
Non-trainable params: 0

High memory request, about 100MB per image to be stored in all the activation maps, only for the forward pass (for the backward it's about twice)

Popular Architectures to be continued...

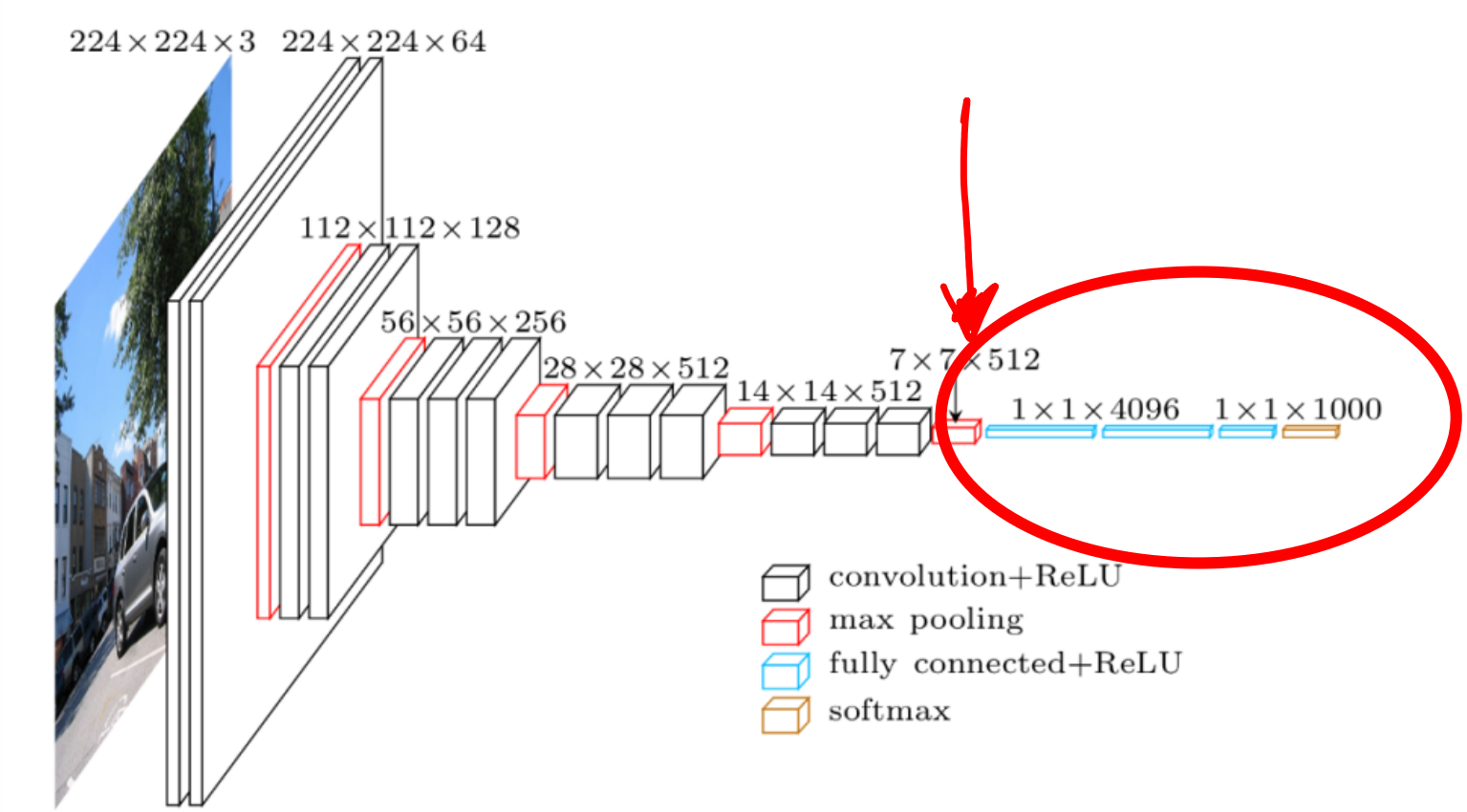
Limited Amount of Data: Transfer Learning

Training a CNN with Limited Aumont of Data

Giacomo Boracchi

Transfer Learning

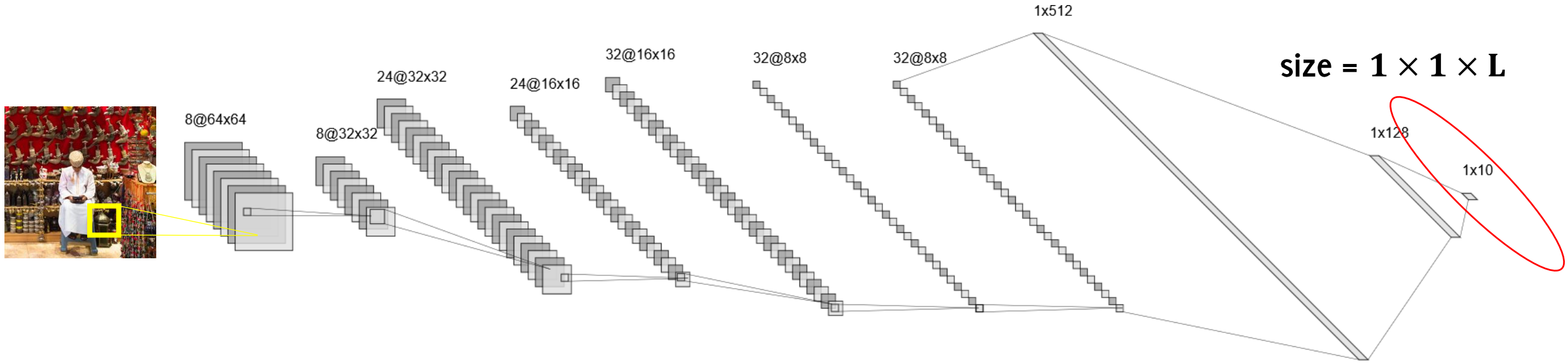
How to use pre-trained models to solve different problems



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Convolutional Neural Networks (CNN)

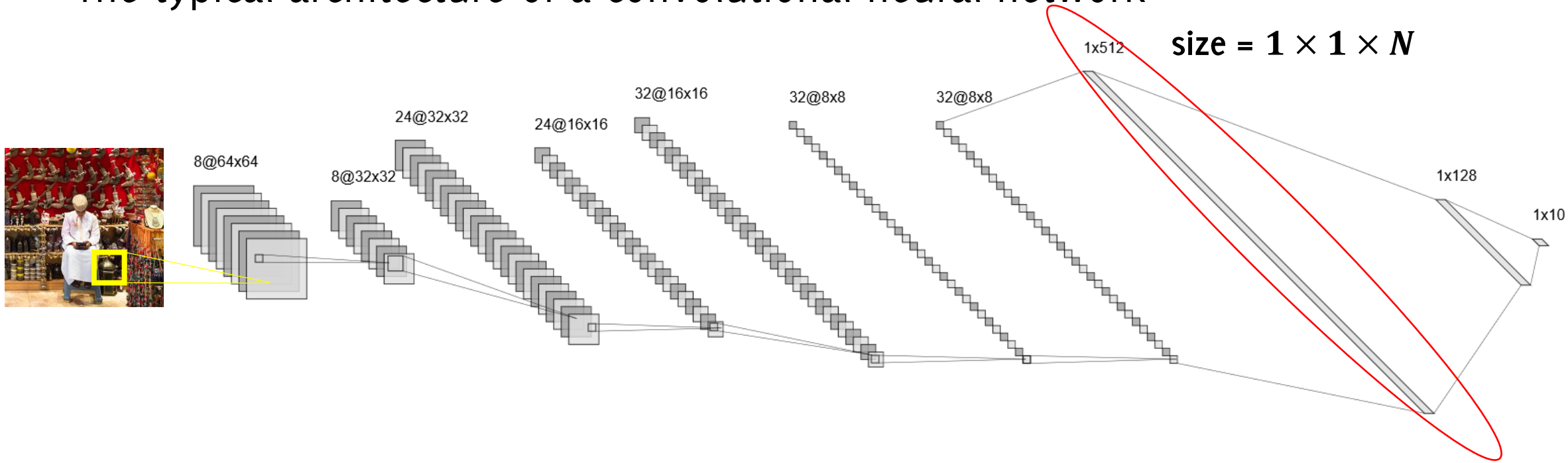
The typical architecture of a convolutional neural network



The output of the fully connected layer has the same size as the number of classes L, and each component provide a score for the input image to belong to a specific class

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



The input of the fully connected layer can be seen as a data-driven descriptor of the input image, i.e. a feature vector. These features are:

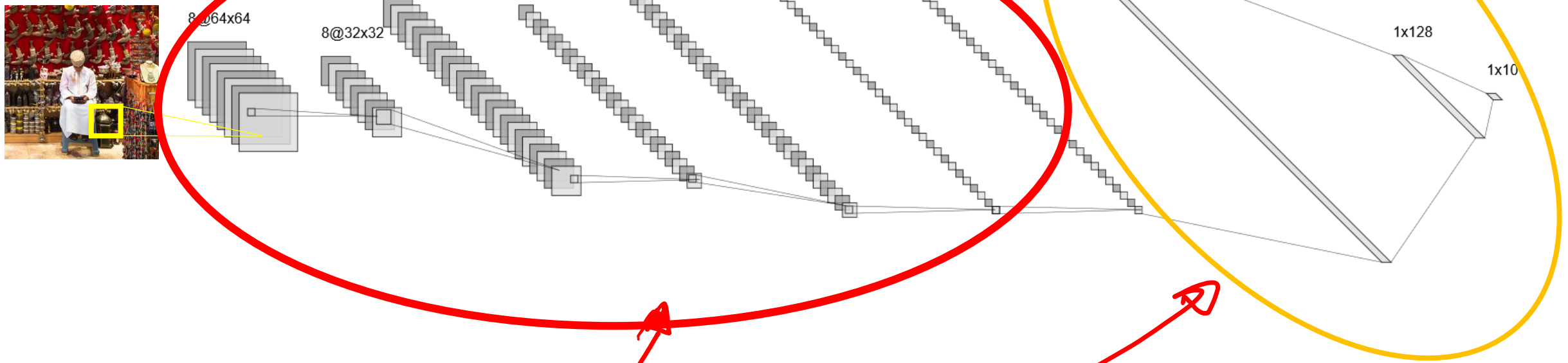
- defined to maximize classification performance
- Trained via backpropagation as the NN they are fed

LATENT REPRESENTATION

By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



Data-driven feature extraction

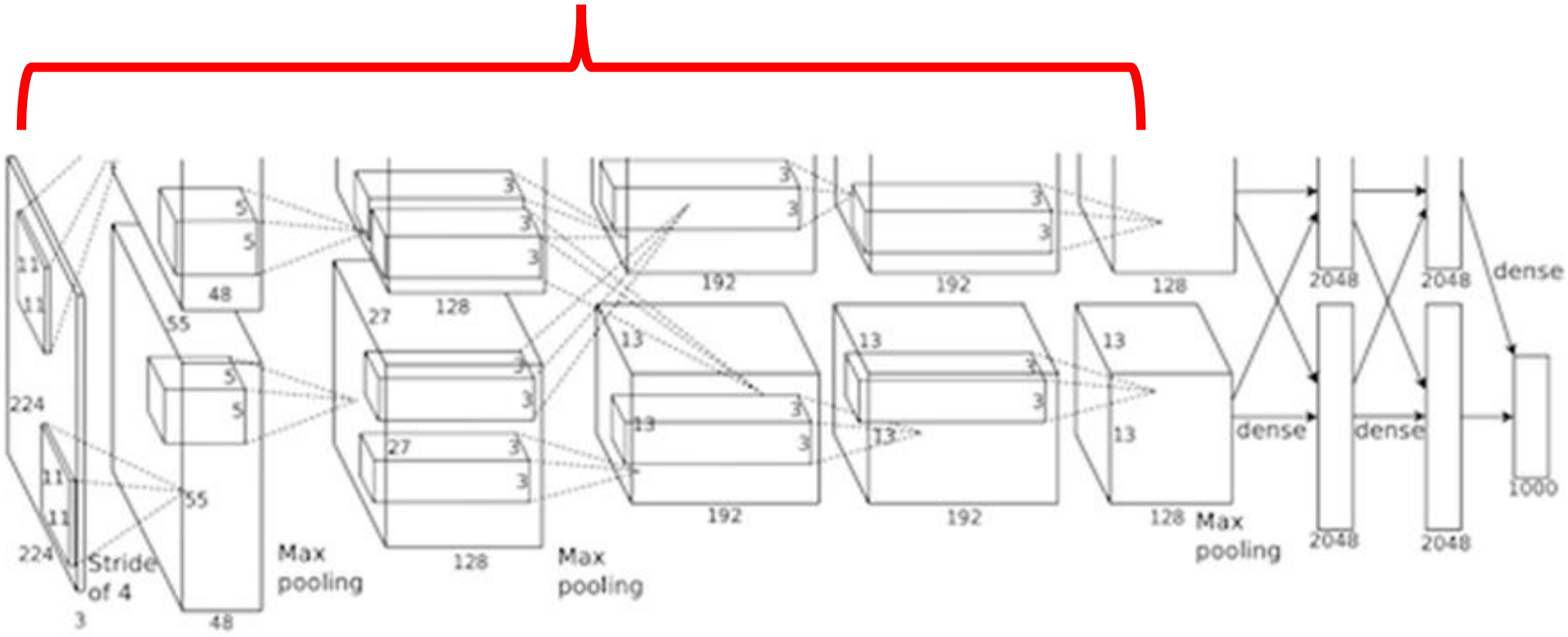
Feature Classifier

By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

Giacomo Boracchi

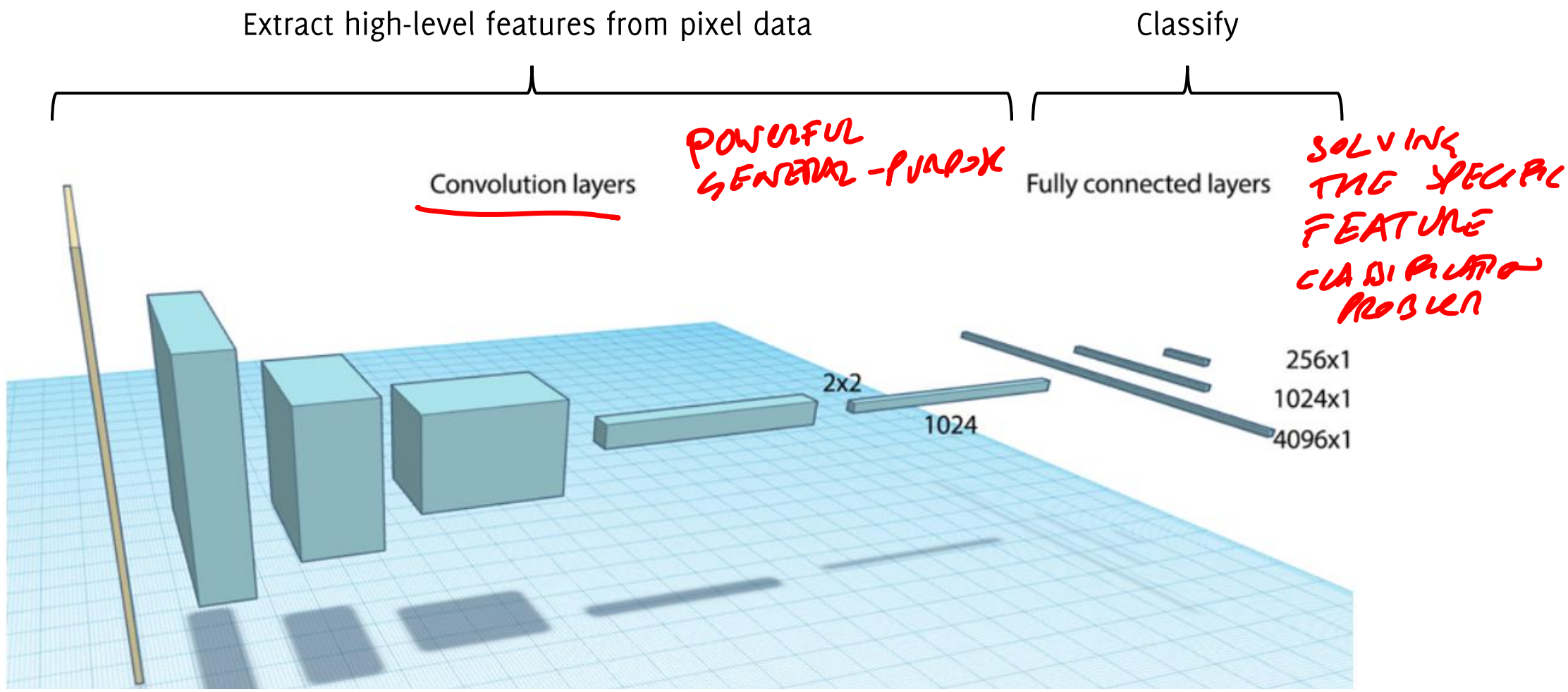
Deep architectures

Deep Networks are great at extracting
(learned) features from images



AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

A typical architecture

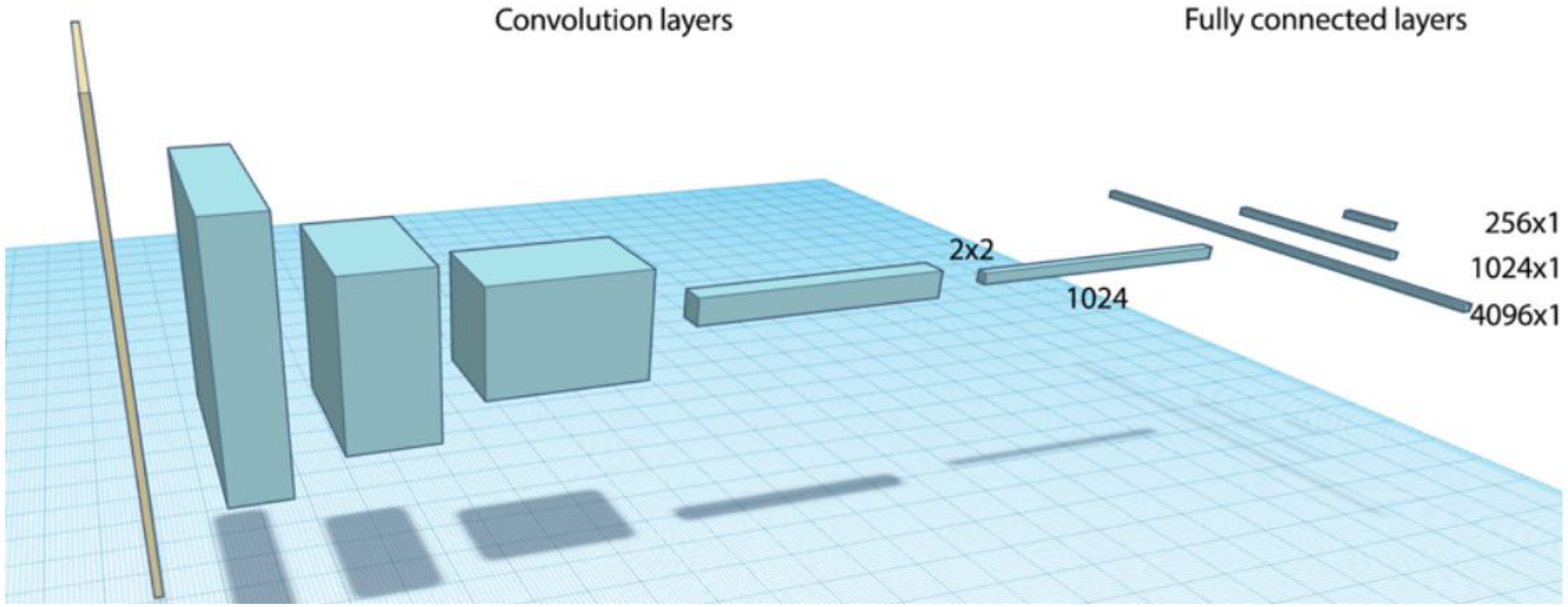


https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture

As we move to deeper layers:

- spatial resolution is reduced
- the number of maps increases

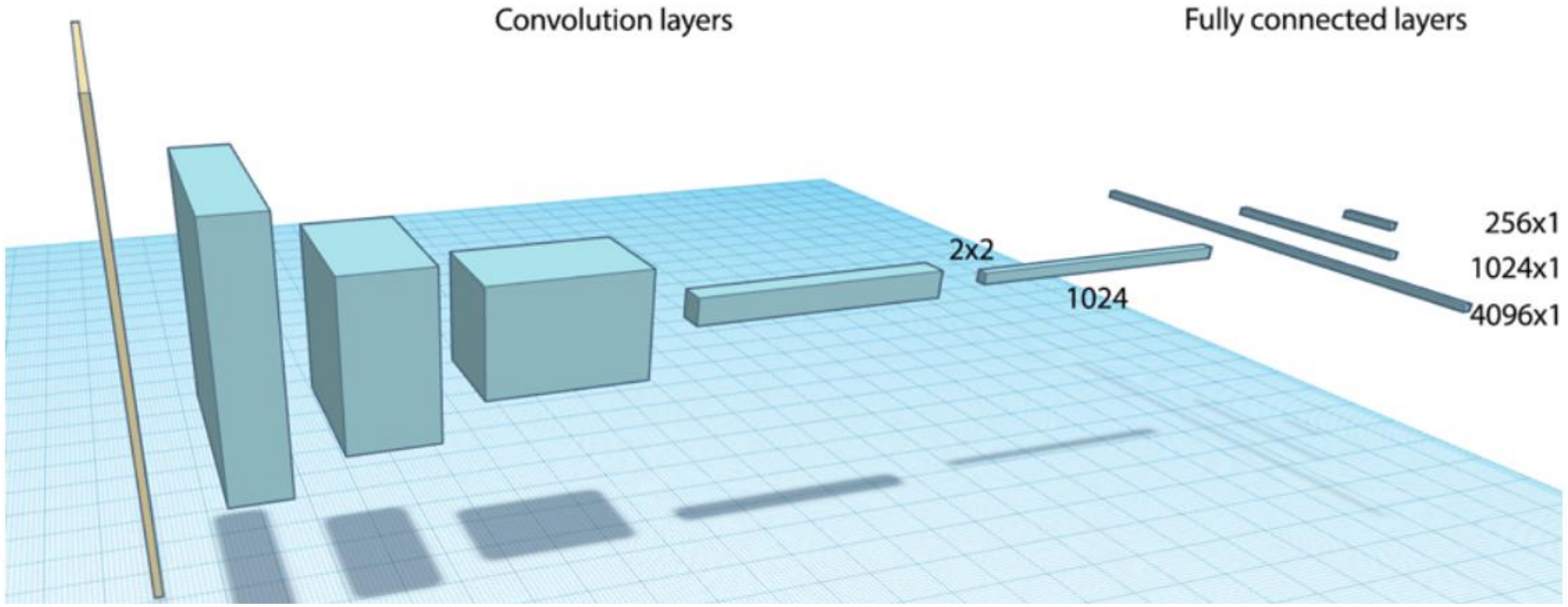


https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture

As we move to deeper layers:

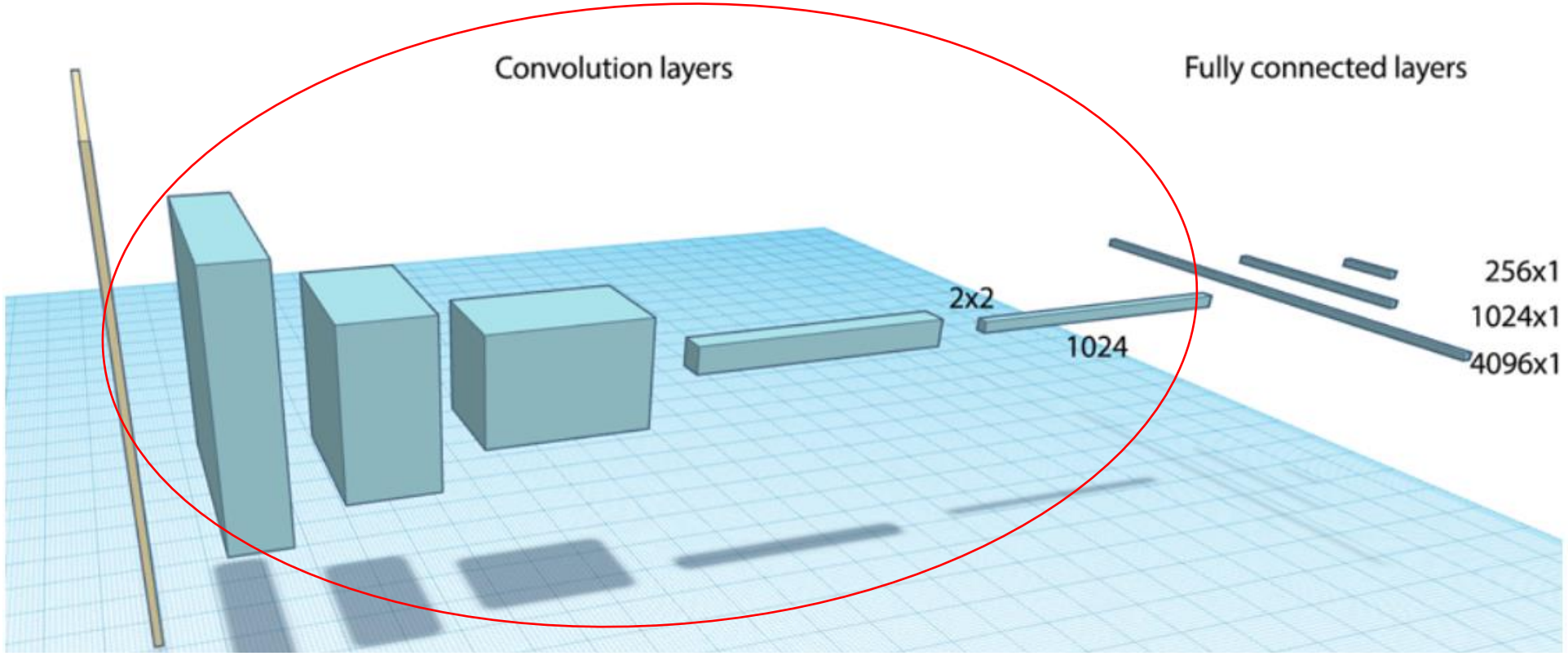
- We search for higher-level patterns, and **don't care too much about their exact location.**
- There are **more high-level patterns** than low-level details!



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture: Convolutional Block

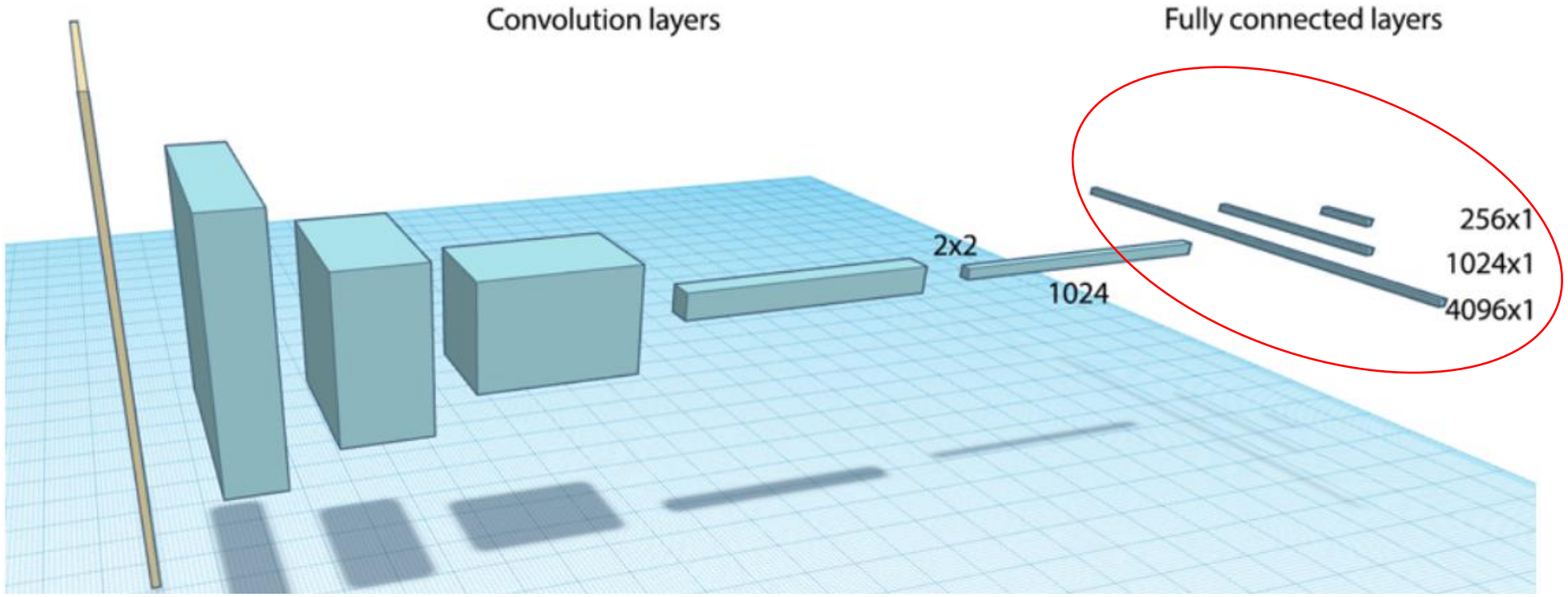
In VGG or Alexnet network that was trained to classify 1000 classes from Imagenet…
the **convolutional part should be a very general feature extractor**



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture: FC layers

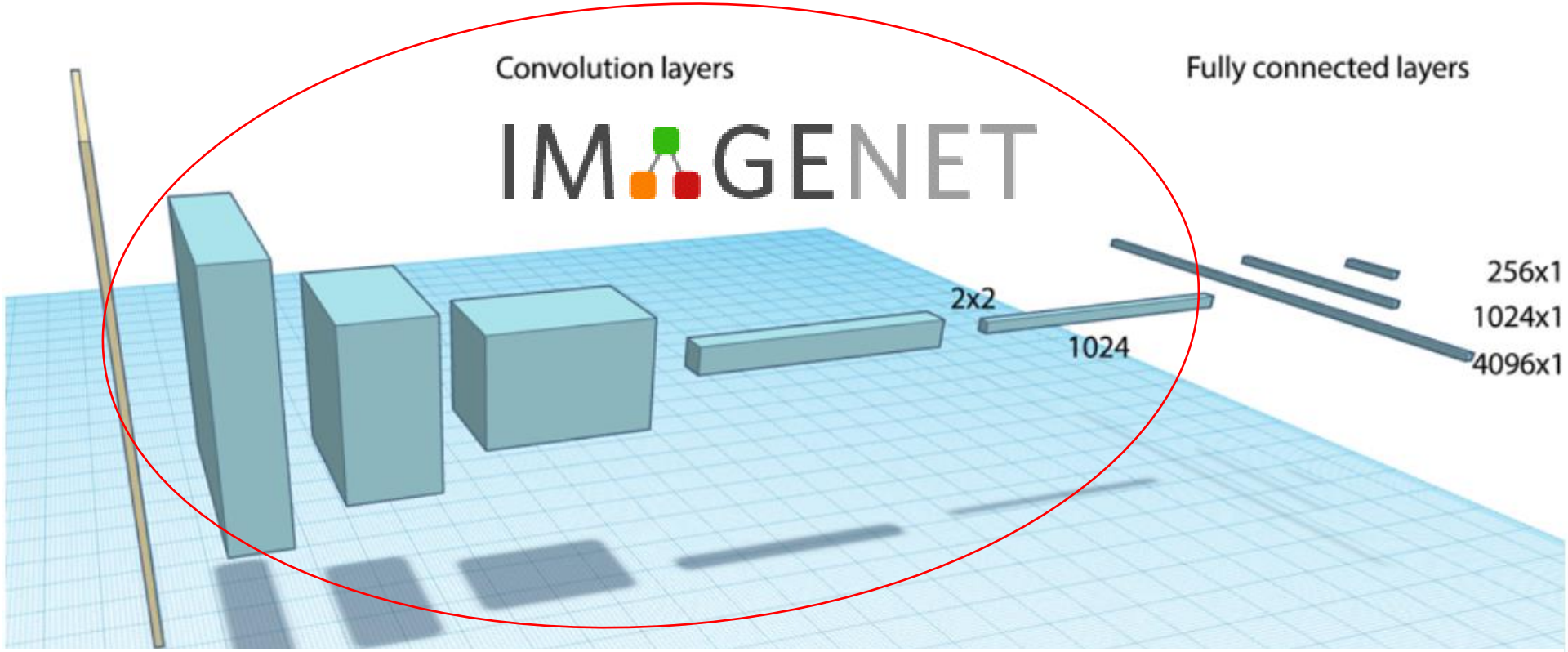
The **FC layers are instead very custom**, as they are meant to solve the specific classification task at hand



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

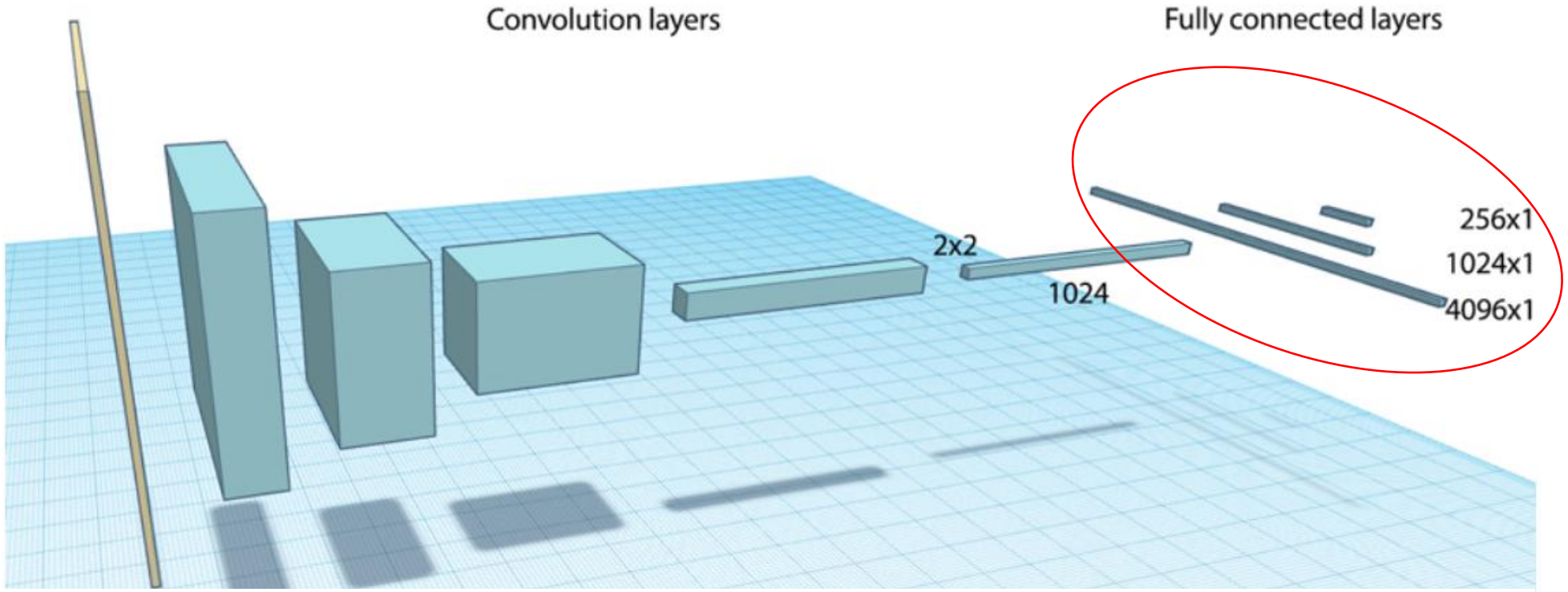
- Take a successful pre-trained model such as VGG



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

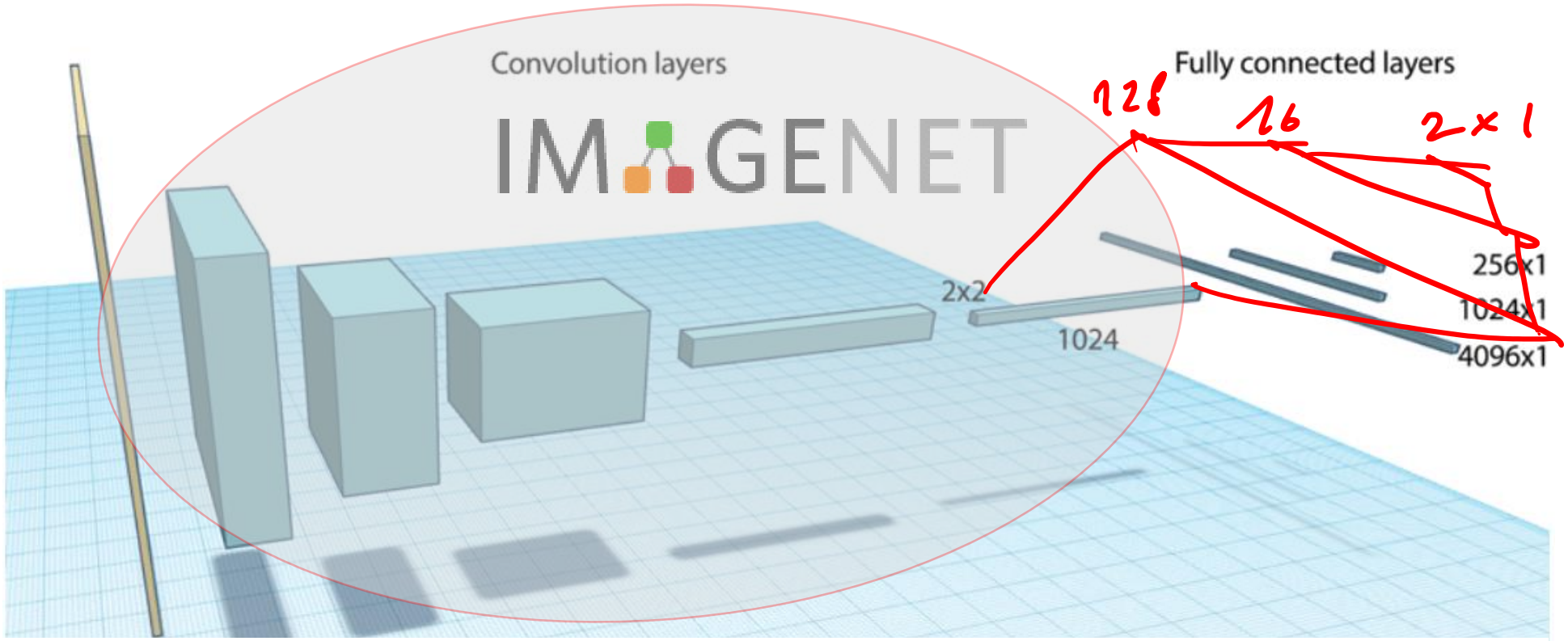
- Take a successful pre-trained model such as VGG
- Remove and modify the fully connected layers for the problem at hand



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

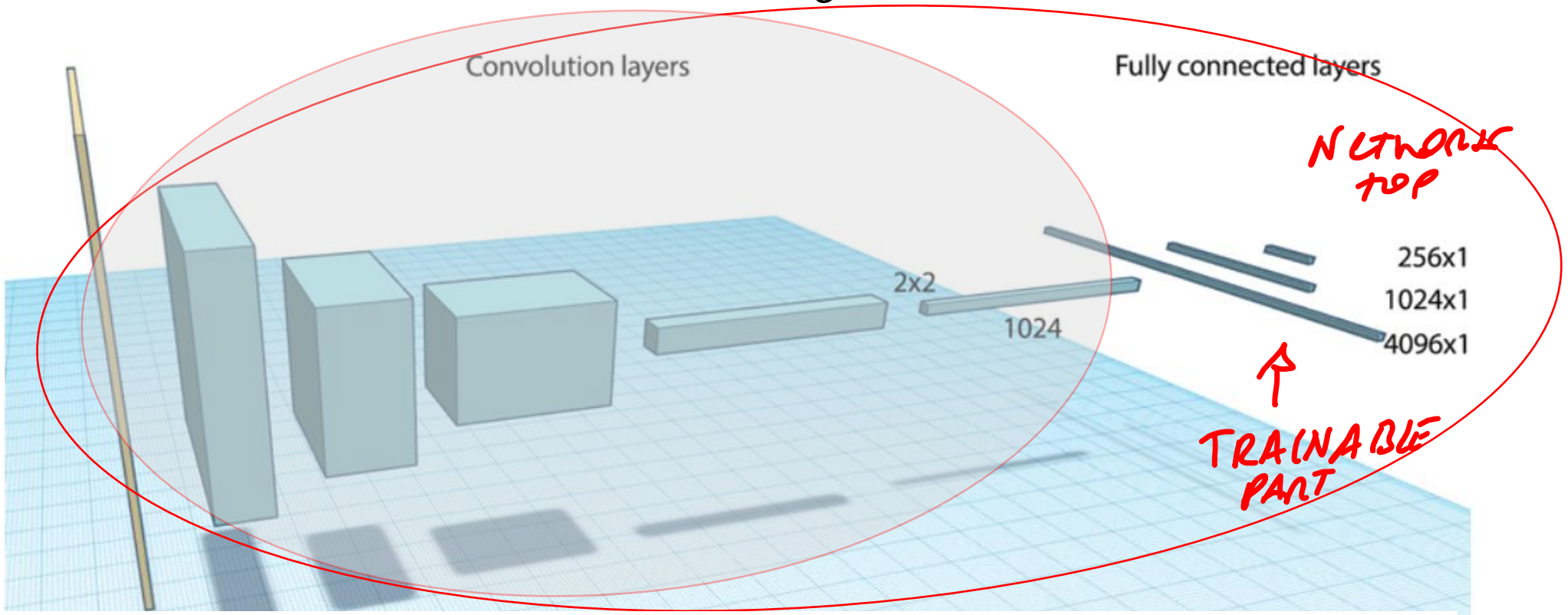
- Take a successful pre-trained model such as VGG
- Remove and modify the fully connected layers for the problem at hand
- «Freeze» the weights in the convolutional layers. **Possibly modify FC architecture**



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

- Take a successful pre-trained model such as VGG
- Remove and modify the fully connected layers for the problem at hand
- «Freeze» the weights in the convolutional layers. **Possibly modify FC architecture**
- Train the whole network on the new training data



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

In keras…

Pre-trained models are available, typically in two ways:

- **`include_top = True`:** provides the entire network, including the fully convolutional layers. This network can be used to solve the classification problem it was trained for
- **`include_top = False`:** contains only the convolutional layers of the network, and it is specifically meant for transfer learning.

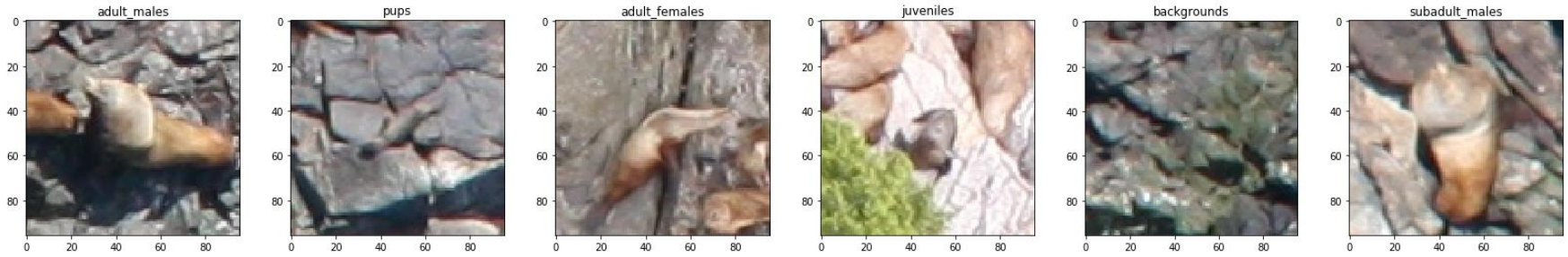
Have a look at the size of these models in the two options!

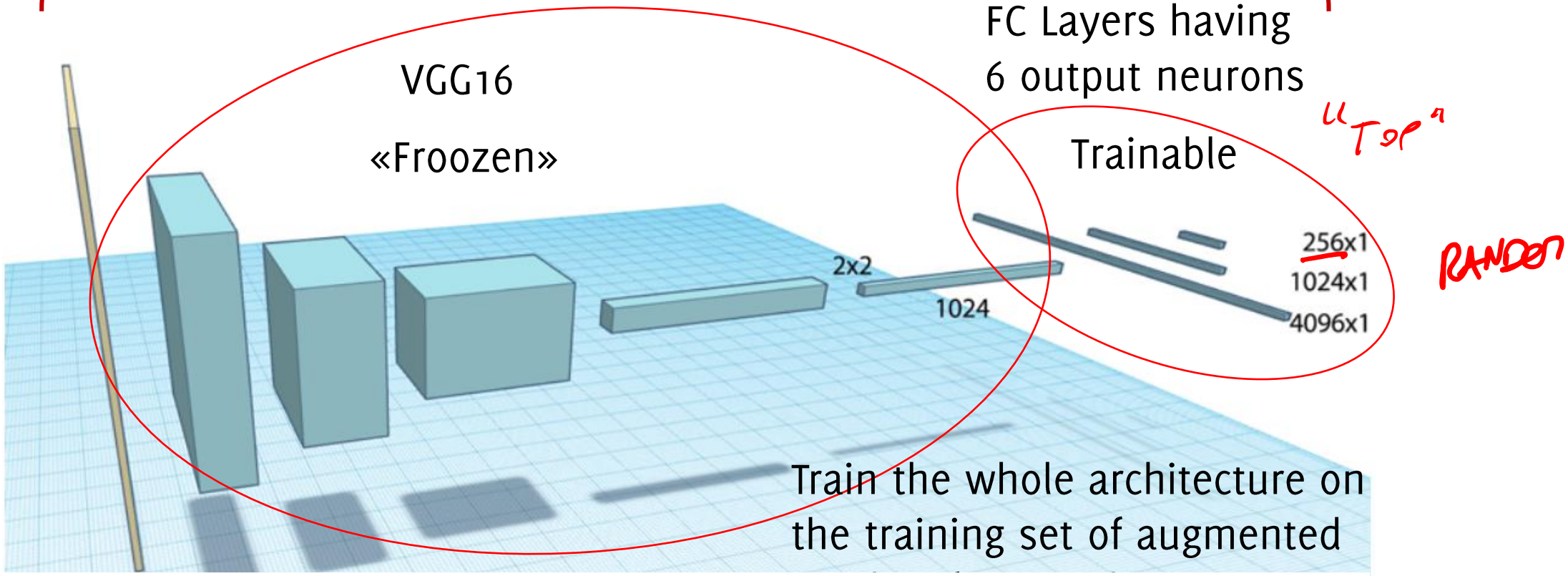
VGG16 in keras...

```
from keras import applications

base_model = applications.VGG16(weights =
"imagenet", include_top=False, input_shape =
(img_width, img_width, 3))
```

Transfer Learning in the Sealion Case





https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning in Keras…

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
for layer in model.layers[: lastFrozen]:
    layer.trainable=False
```

Transfer Learning

Different Options:

- **Transfer Learning**: **only the FC** layers are being trained. A good option when **little training data** are provided and the **pre-trained model** is expected to **match** the problem at hand
- **Fine tuning:** the whole **CNN is retrained**, but the **convolutional layers** are **initialized to the pre-trained model**. A good option when **enough training data** are provided or when the **pre-trained model is not expected to match** the problem at hand.

Typically, for the same optimizer, **lower learning rates** are used when performing fine tuning than when training from scratches