

CNN Parameters and Training

Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 28th, 2

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

Parameters in a CNN

Convolutions as MLP

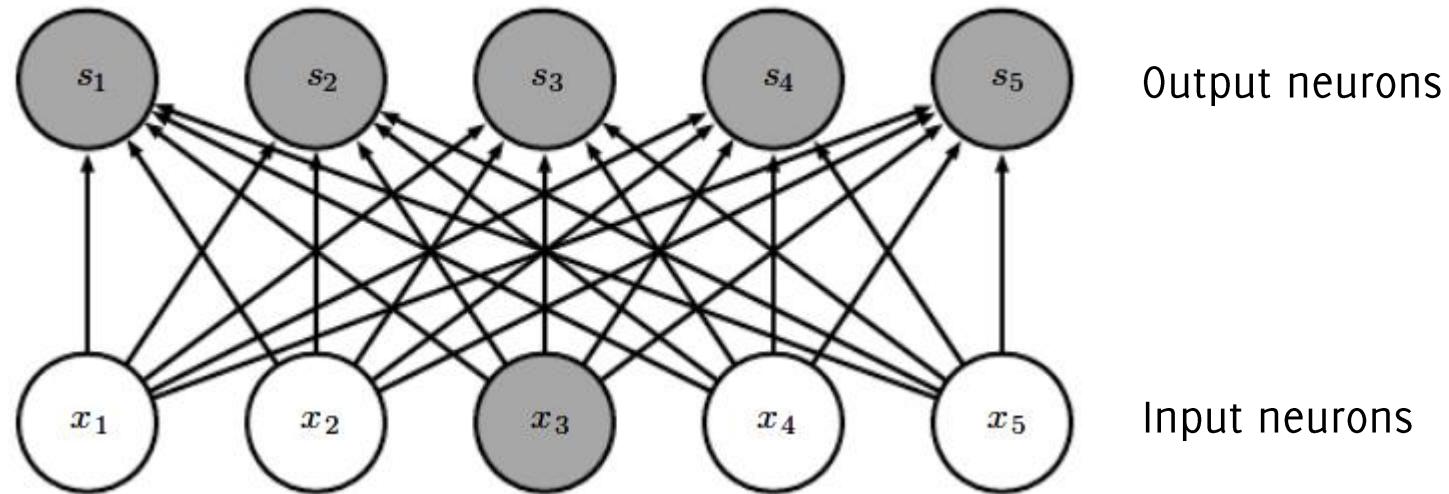
Convolution is a linear operation!

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

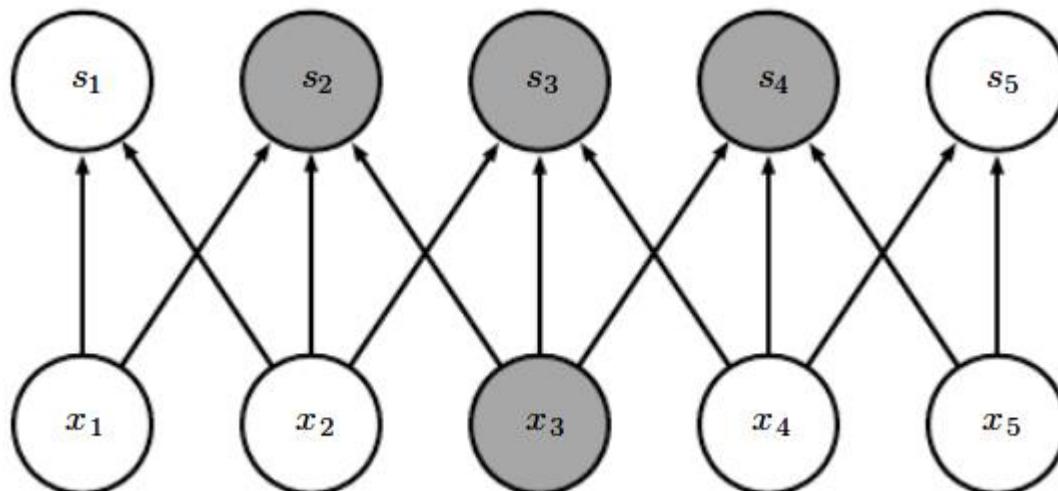
What are the differences between MLP and CNNs then?

CNNs Feature Sparse Connectivity

Fully connected



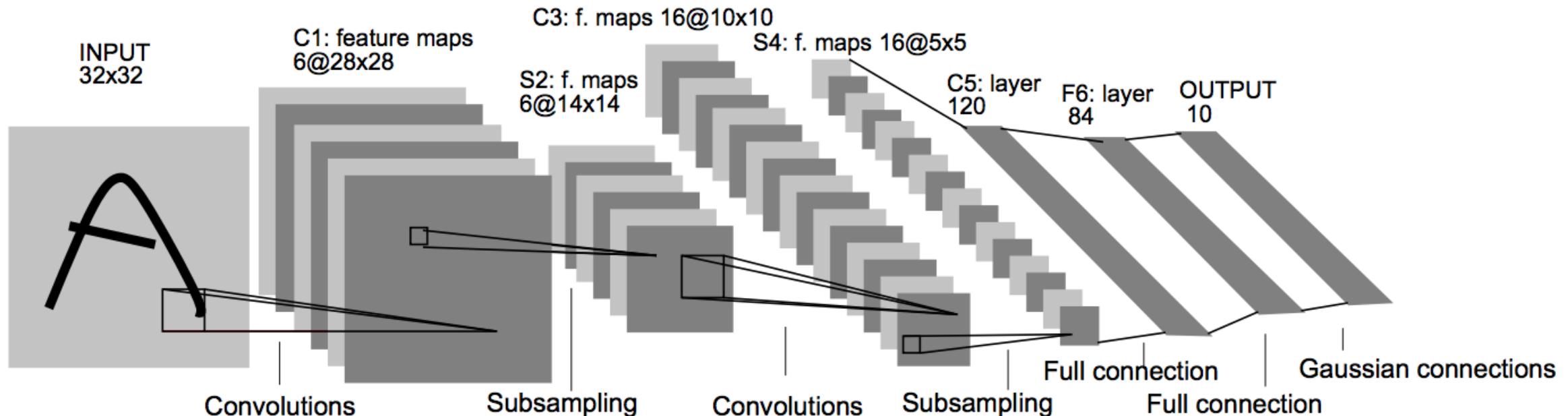
3x1 convolutional



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.

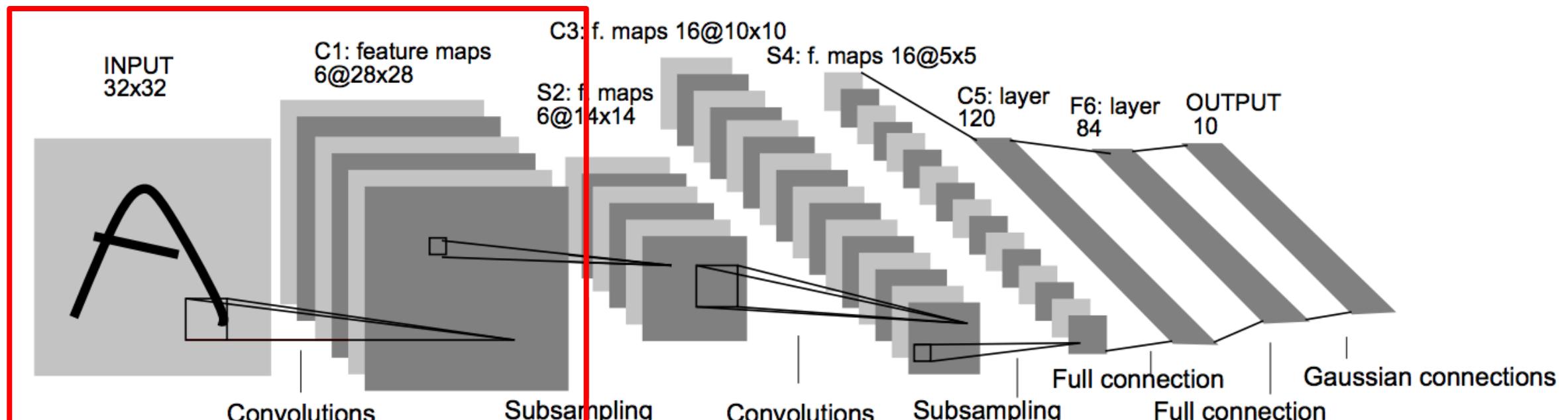
Underlying assumption: **if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x₂,y₂)**



Weight Sharing / Spatial Invariance

If the first layer were a MLP:

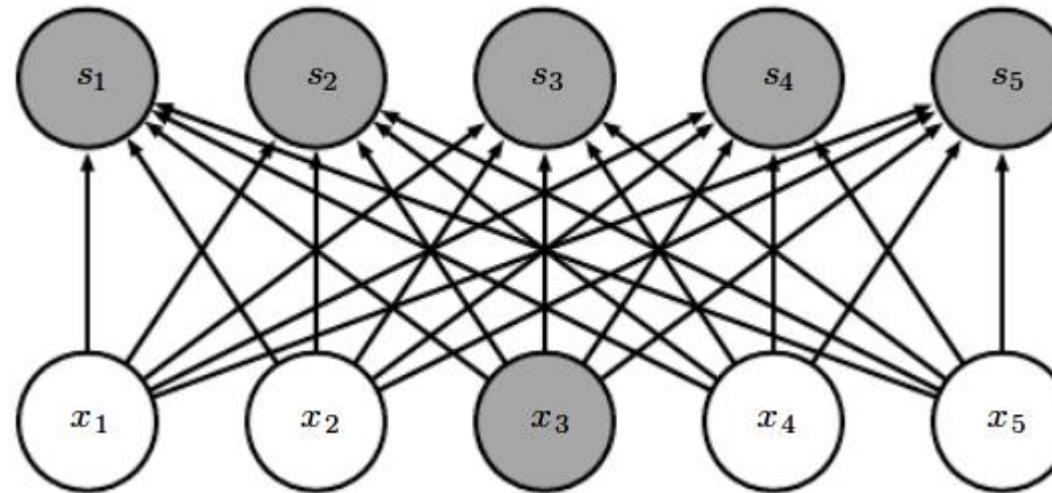
- it should have had $28 \times 28 \times 6$ neurons in the output
 - Consider sparse connectivity: 5×5 nonzero weights each neuron
 - It would be a $28 \times 28 \times 6 \times 25$ weights + $28 \times 28 \times 6$ biases (122 304)
- CNN share the same weights among a few neurons of the same layer**



Parameter sharing

Fully connected

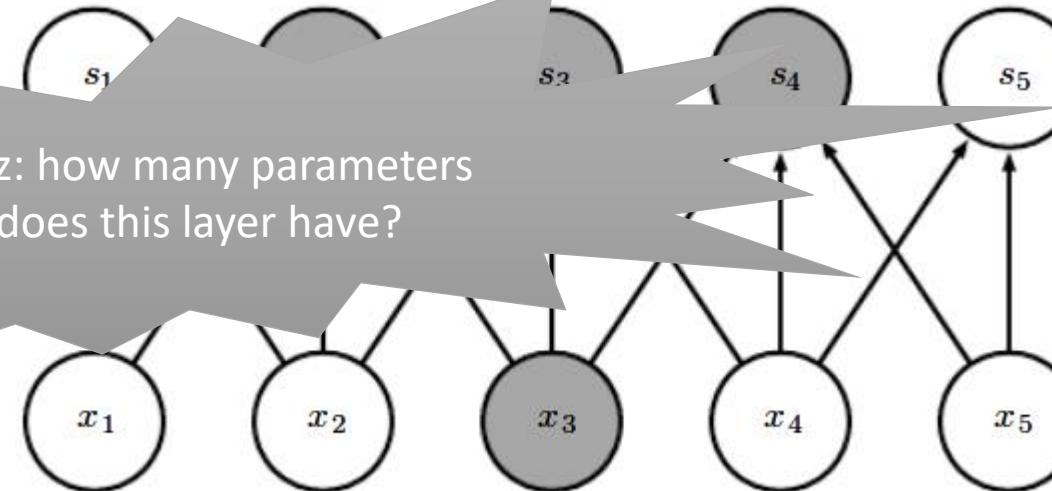
$5 \times 5 = 25$ weights
(+ 5 bias)



3x1 conv

3 weights!
(+ 1 bias)

Quiz: how many parameters does this layer have?



The Receptive Field

A very important aspect in CNNs

The Receptive Field

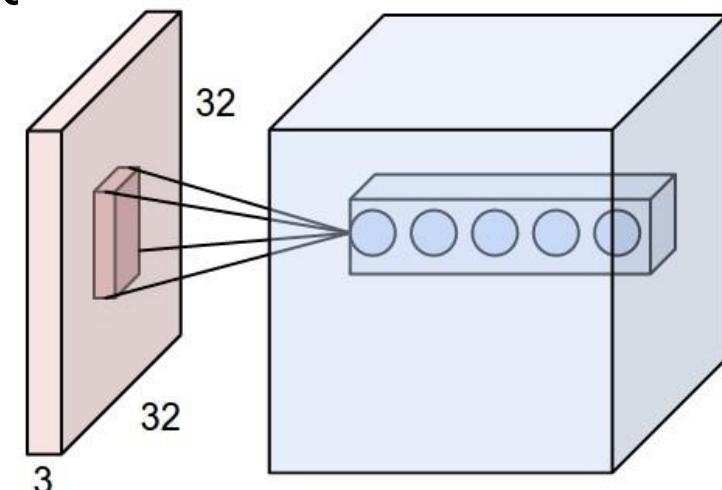
One of the basic concepts in deep CNNs.

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

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

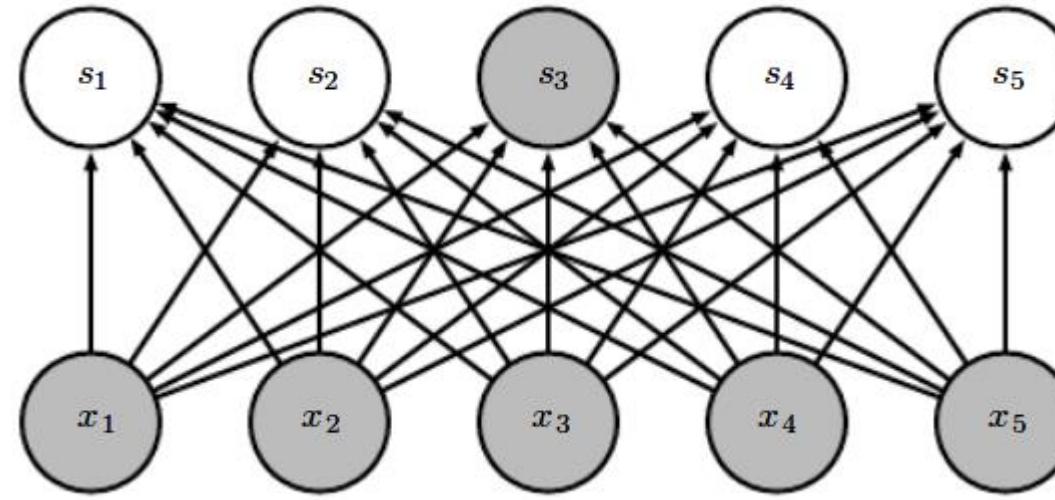
The deeper you go, the wider the receptive field is: maxpooling, convolutions and stride > 1 increase the receptive field

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

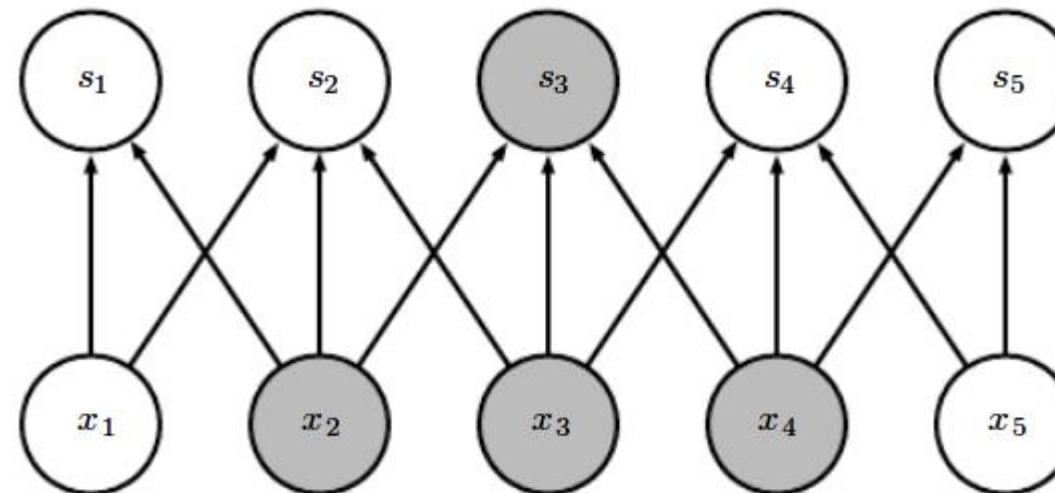


Receptive fields

Fully connected



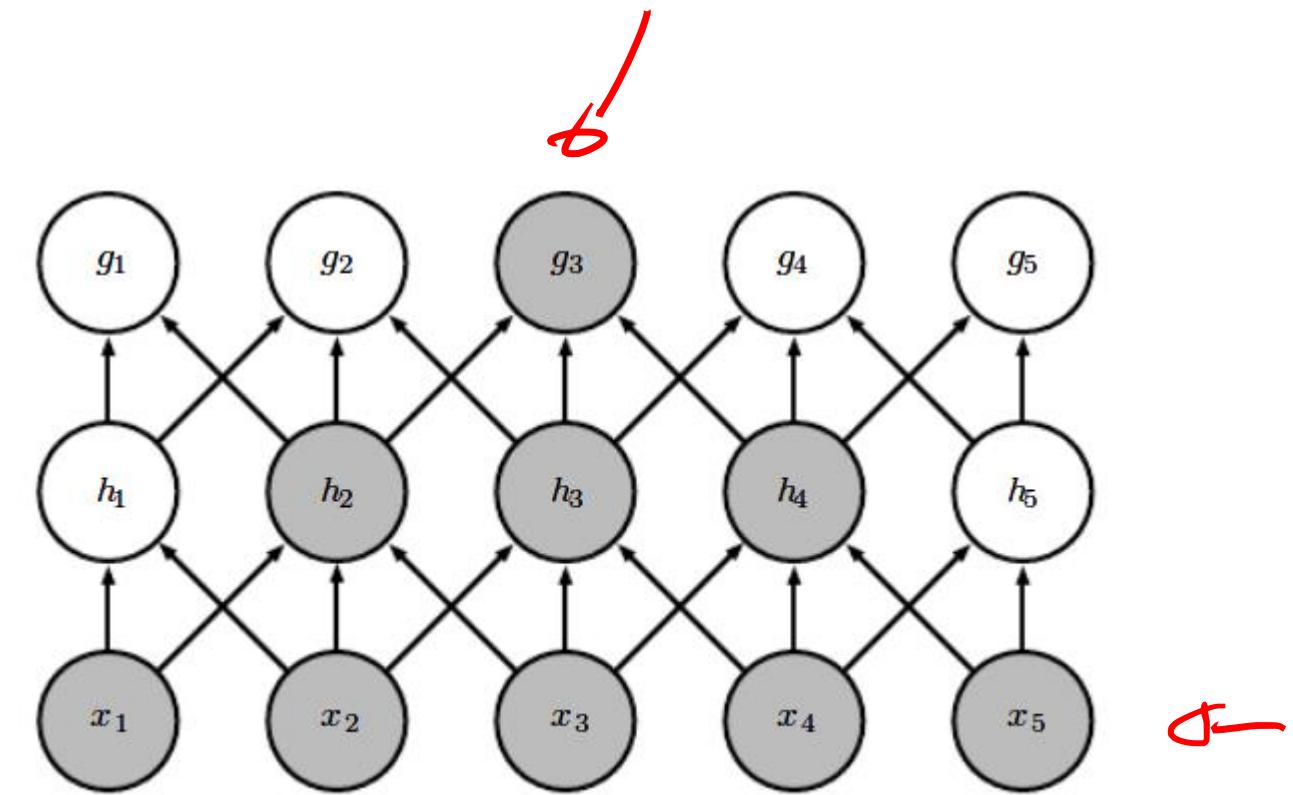
3x1 convolutional



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:



Conv 3x3

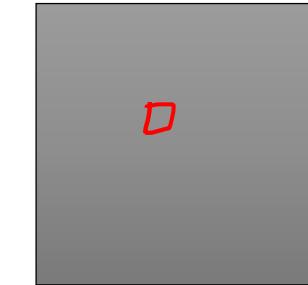
Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3



map



Conv 3x3

MP 2x2

Conv 3x3

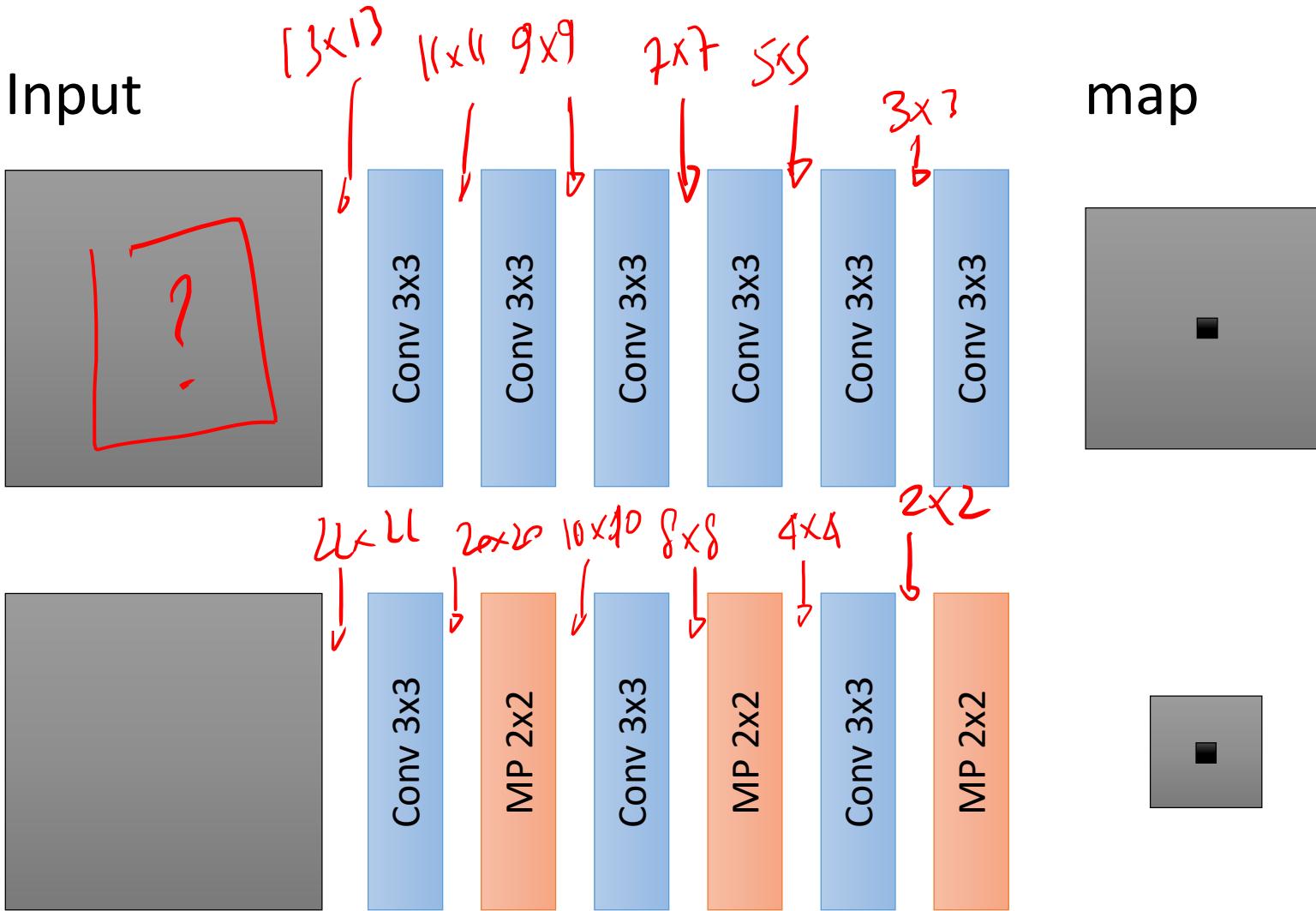
MP 2x2

Conv 3x3

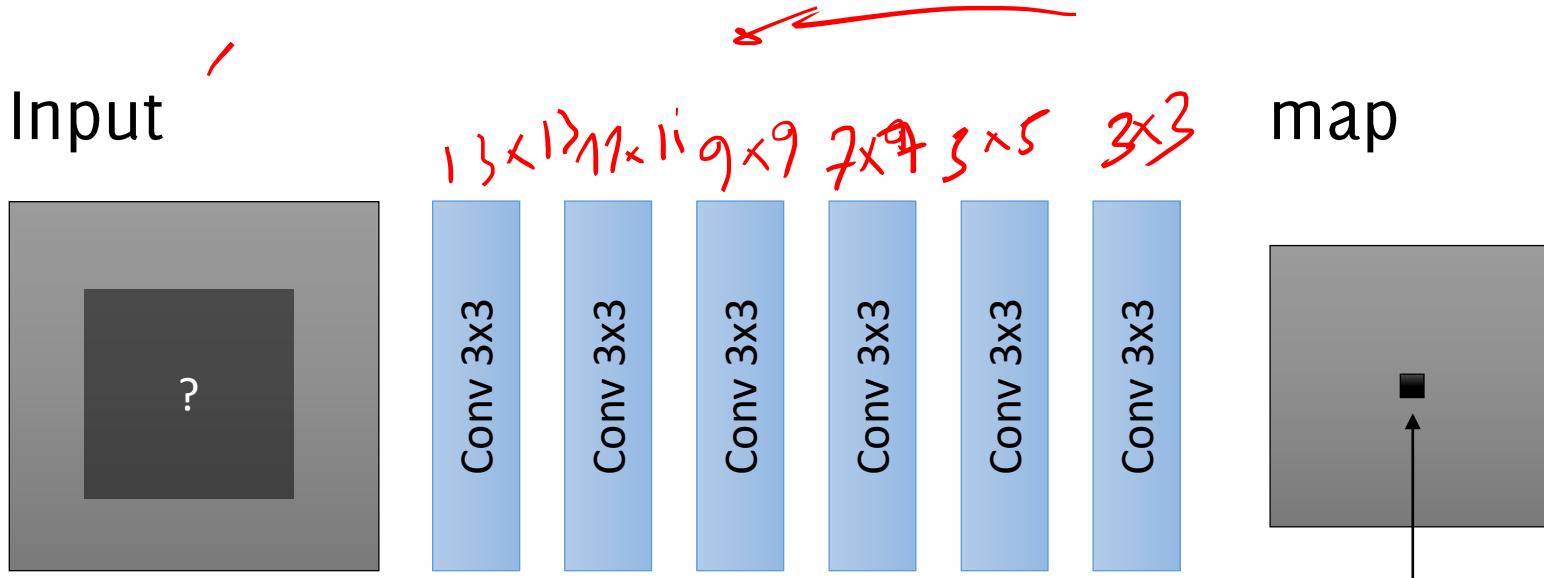
MP 2x2



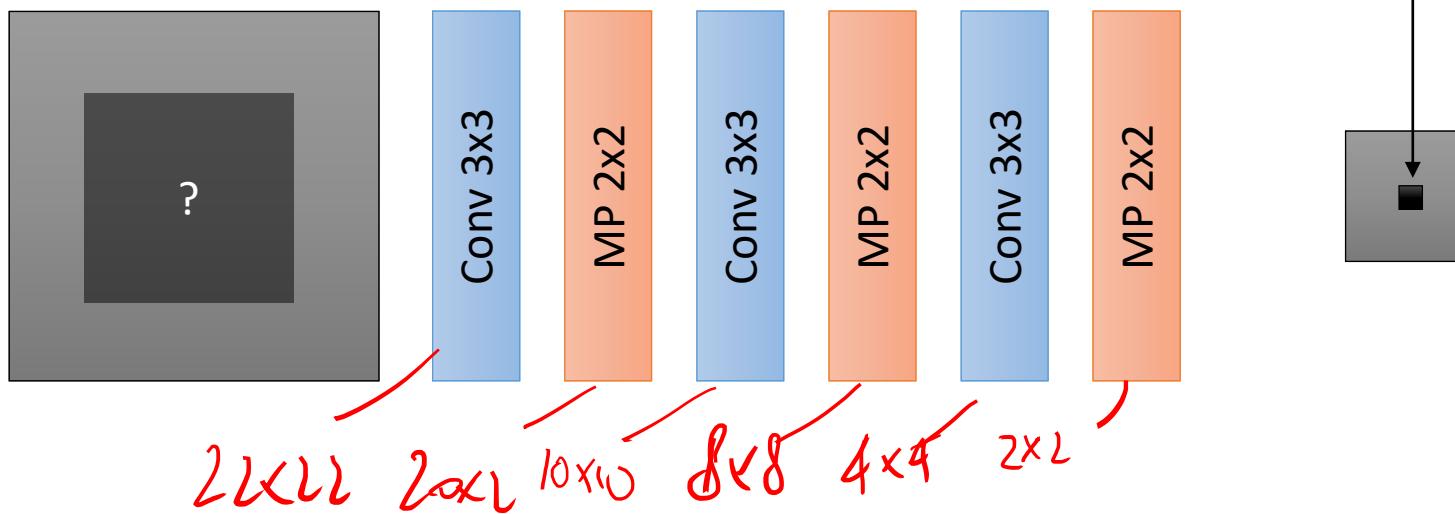
Receptive fields



Receptive fields

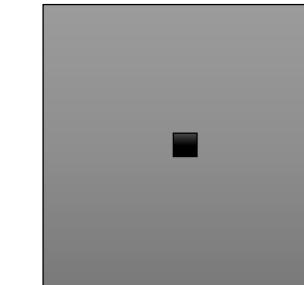
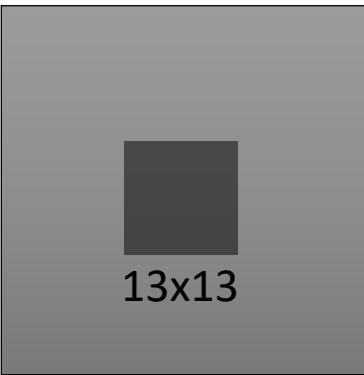


How large is the receptive field of the black neuron?

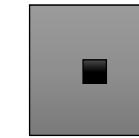


Receptive fields

Input



How large is the receptive field of the black neuron?



no body

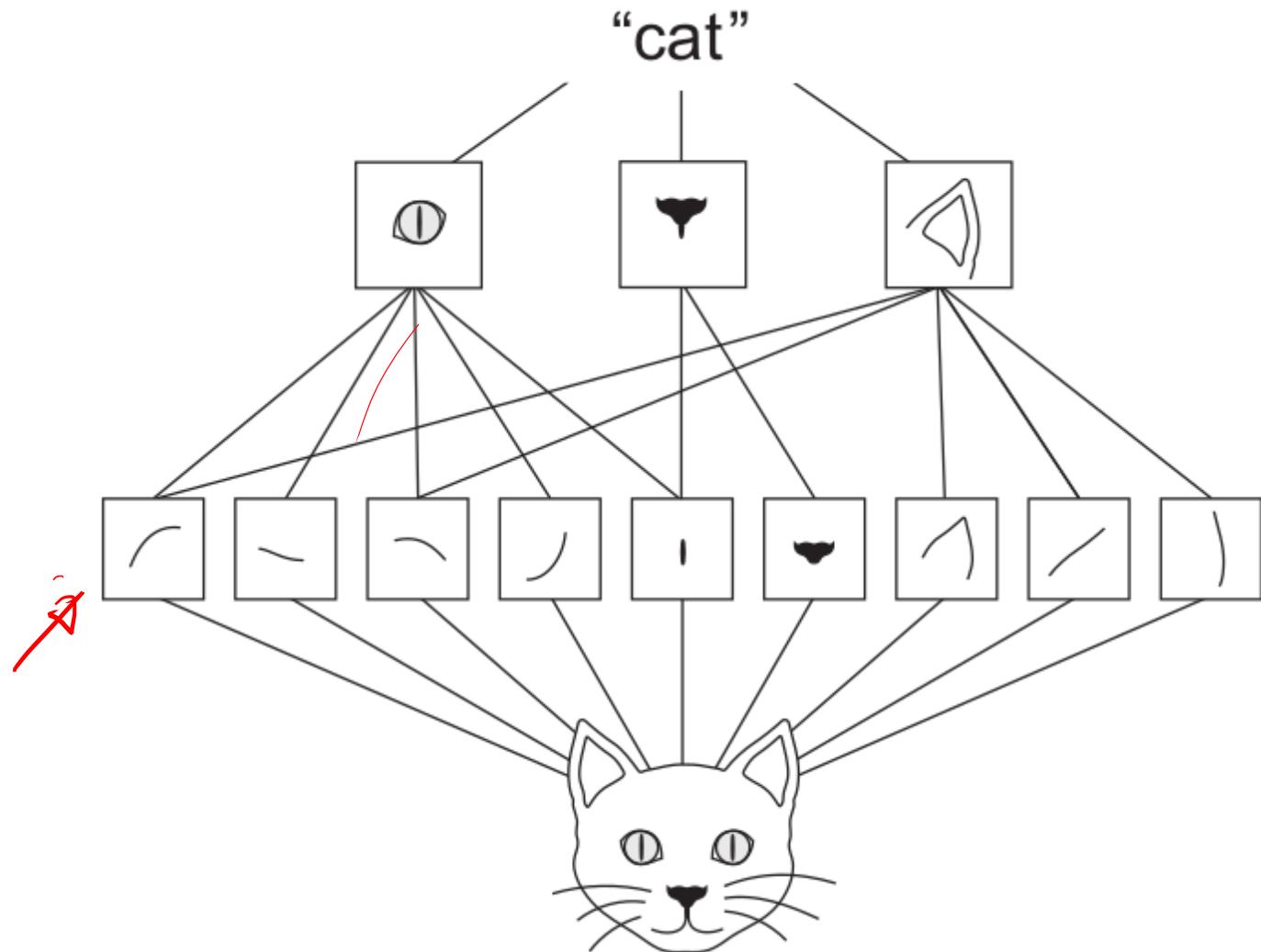
As we move deeper...

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

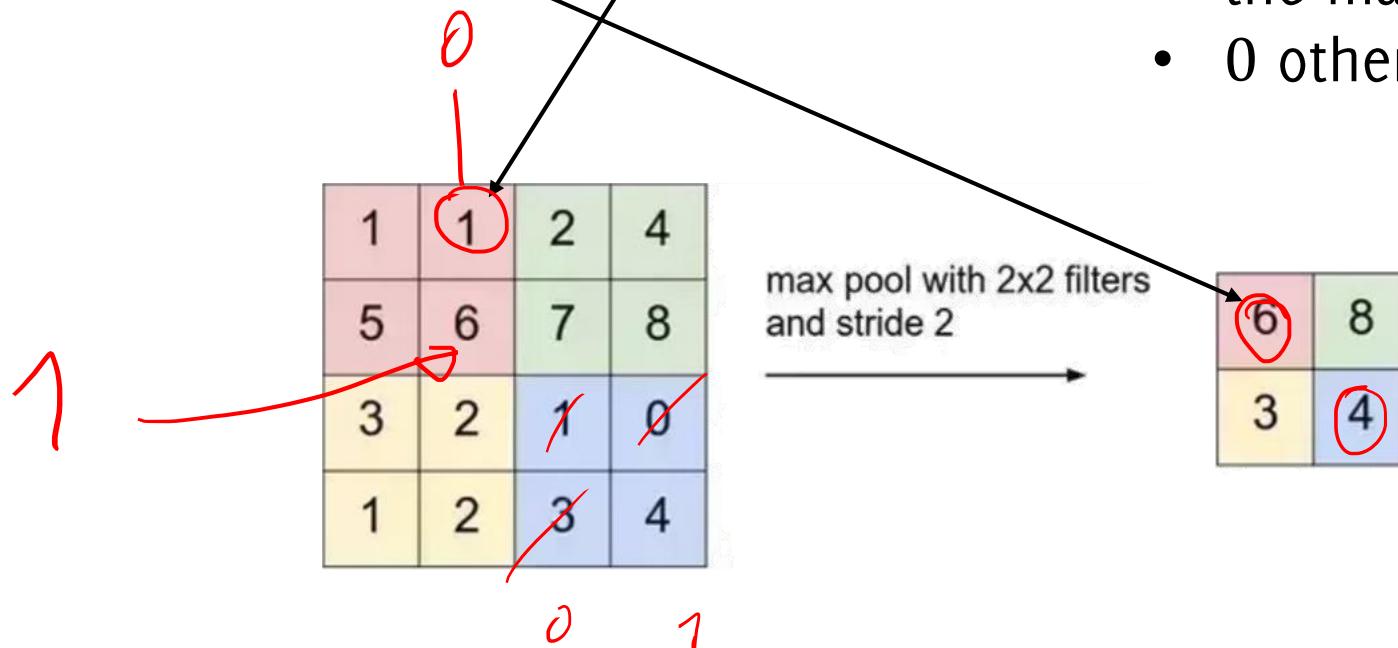
Training a CNN

- The CNN can be seen as a MLP (with sparse and shared connectivities)
- CNN can in principle be 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 \bullet with respect to $\bullet = 0$

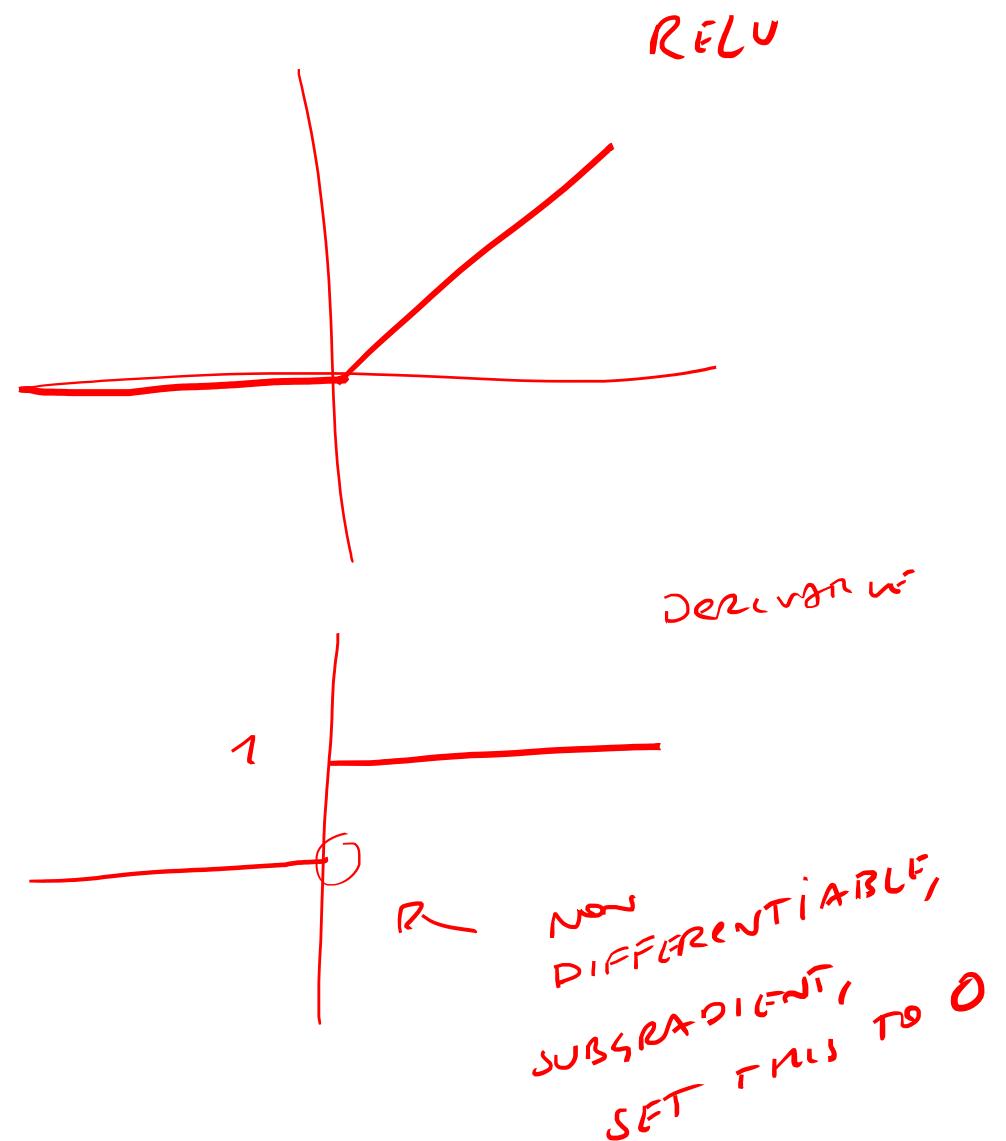


The derivative is:

- 1 at the location corresponding the maximum
- 0 otherwise

Detail: derivative of ReLU

The ReLU derivative is straightforward



Data Scarcity

Training a CNN with Limited Amount of Data

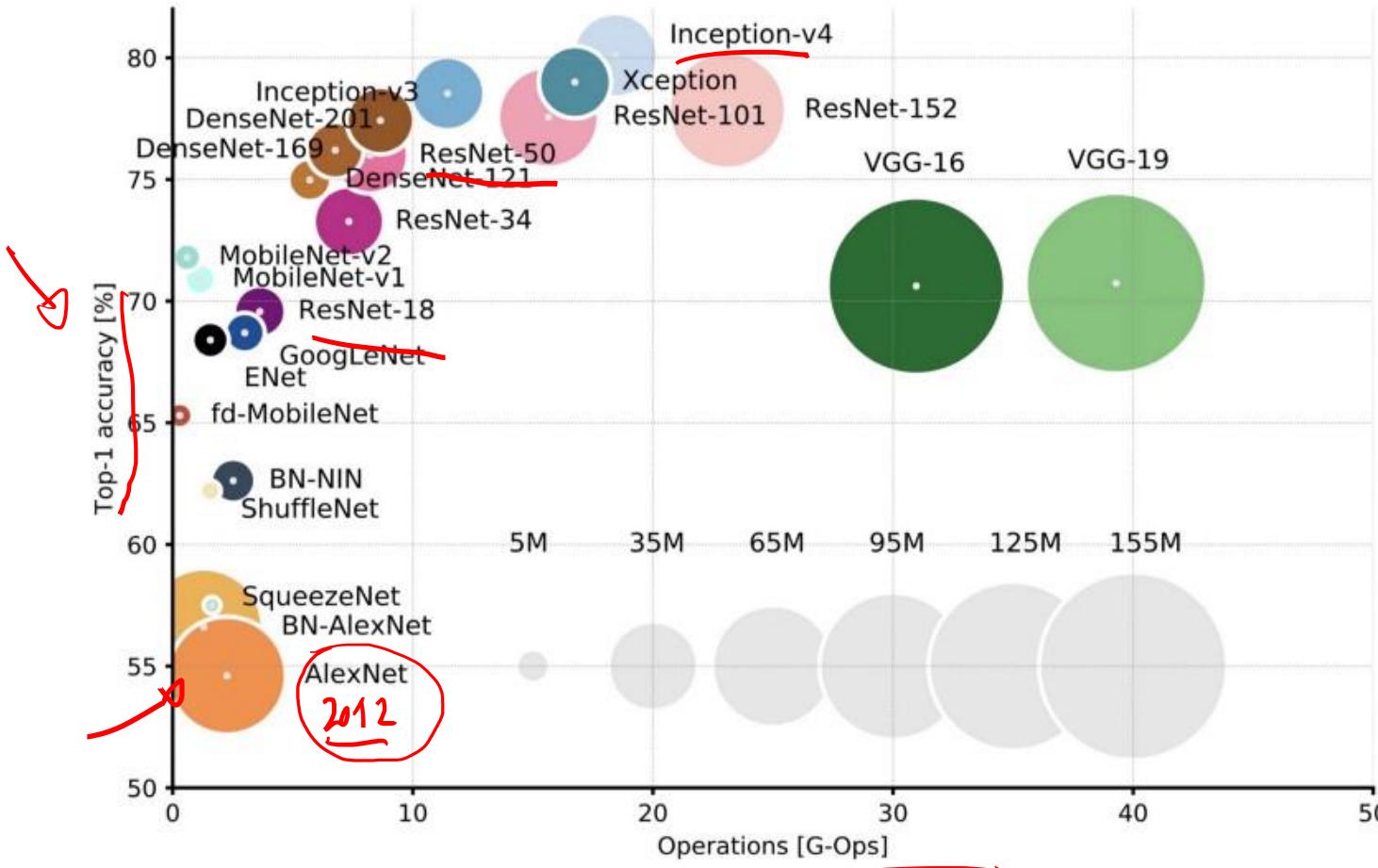
The need of data

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



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



Aleutian Islands

Steller sea lions in the western Aleutian Islands have declined 94% in the last 30 years.



Kaggle in 2017 have opened a competition to develop algorithms which accurately count the number of sea lions in aerial photographs

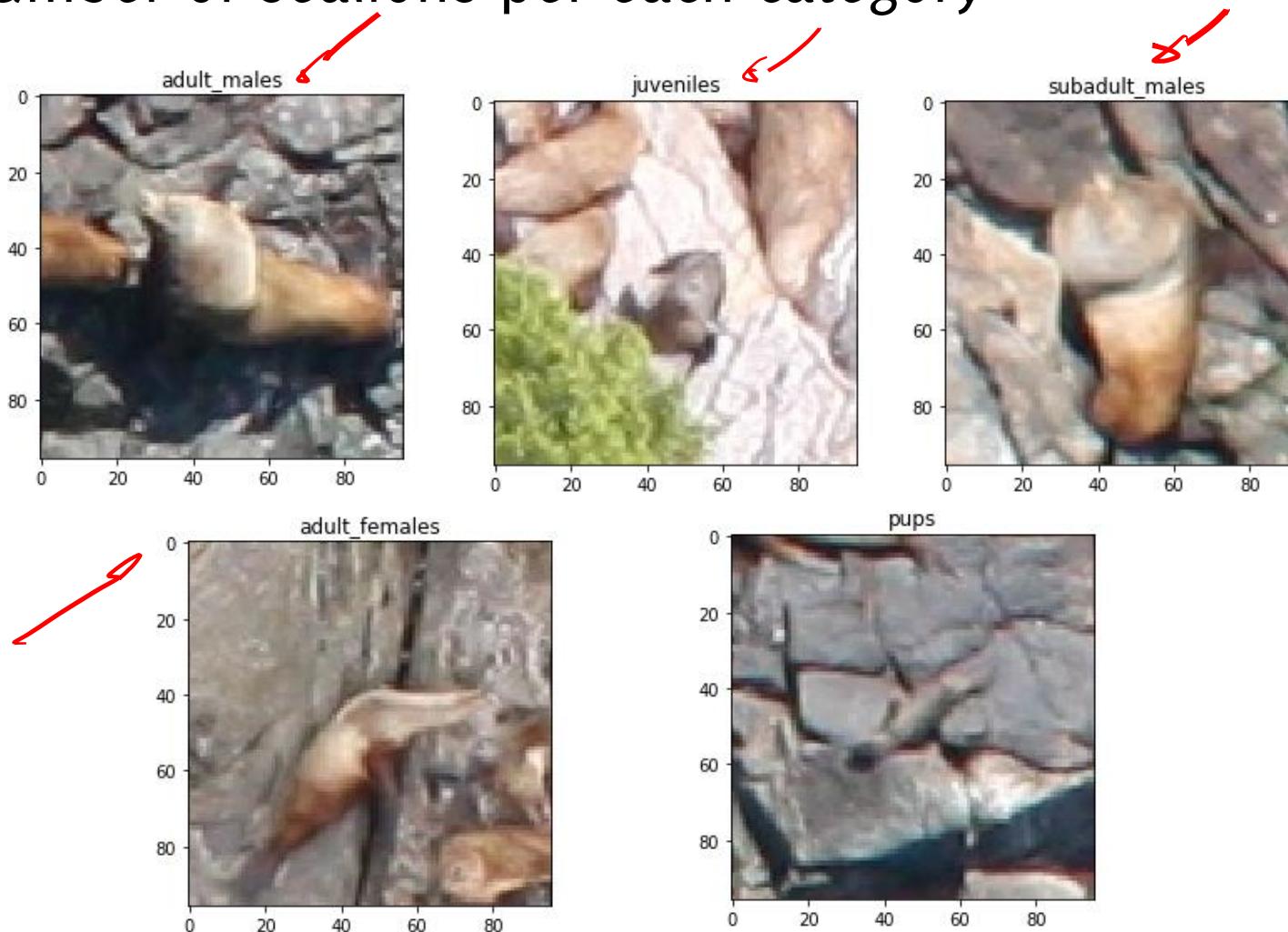


Credits Yinan Zhou

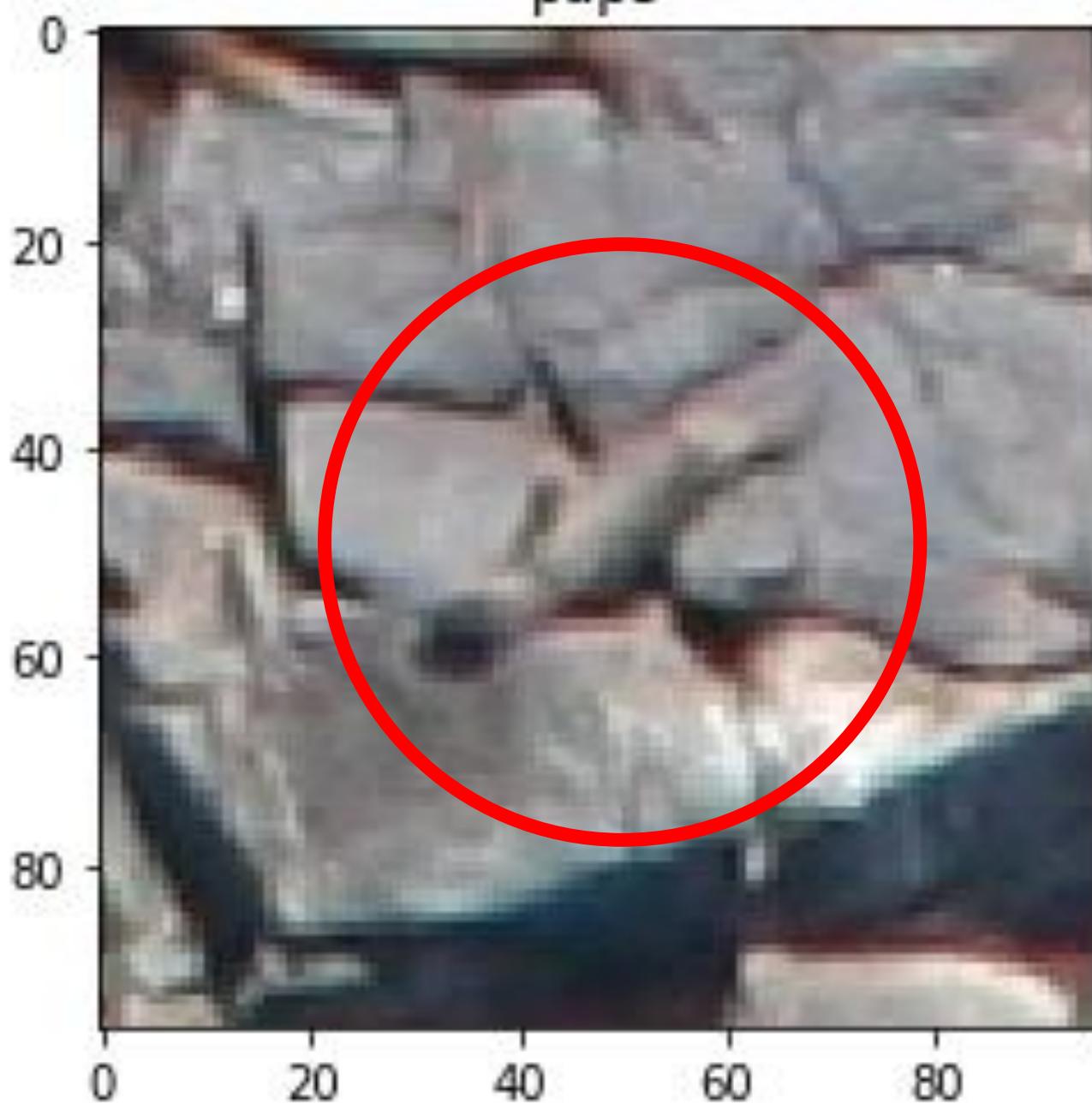
<https://github.com/marioZYN/FC-CNN-Demo>

The Challenge

In very large aerial images ($\approx 5K \times 4K$) shot by drones, automatically count the number of sealions per each category

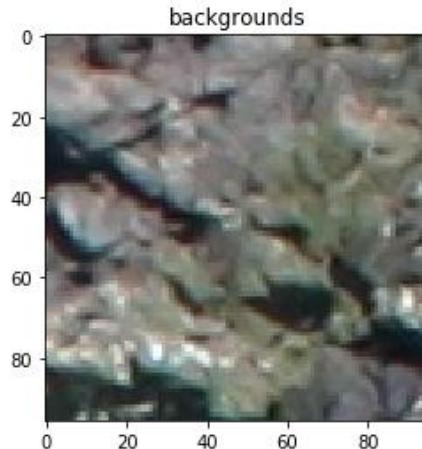
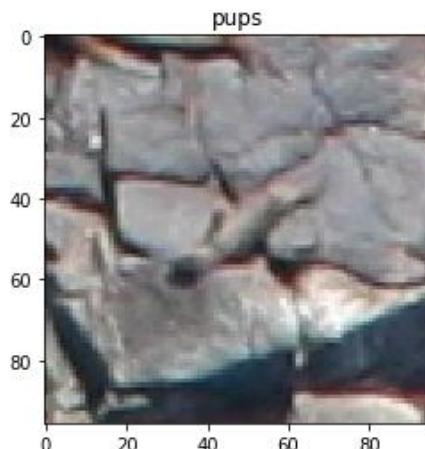
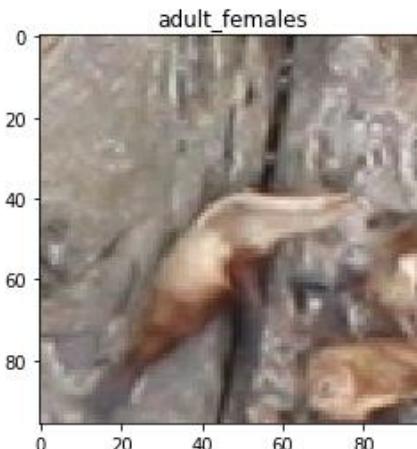
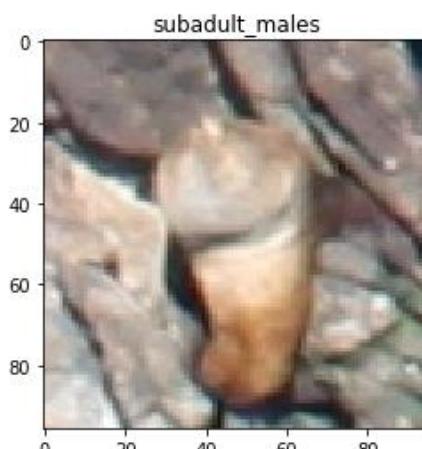
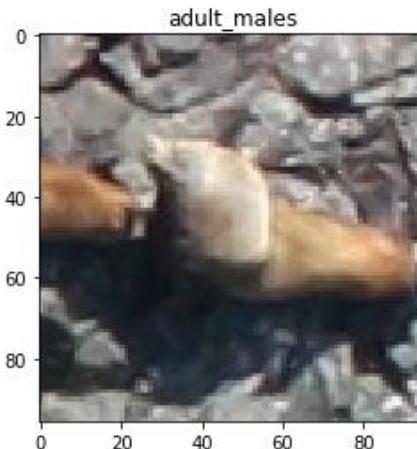


pups

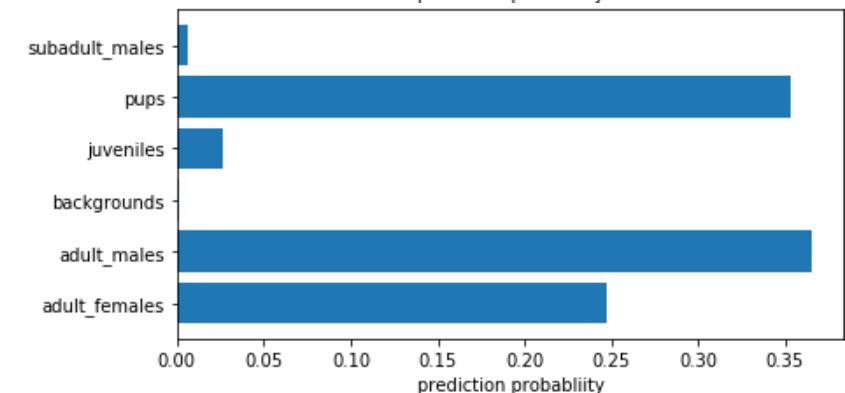
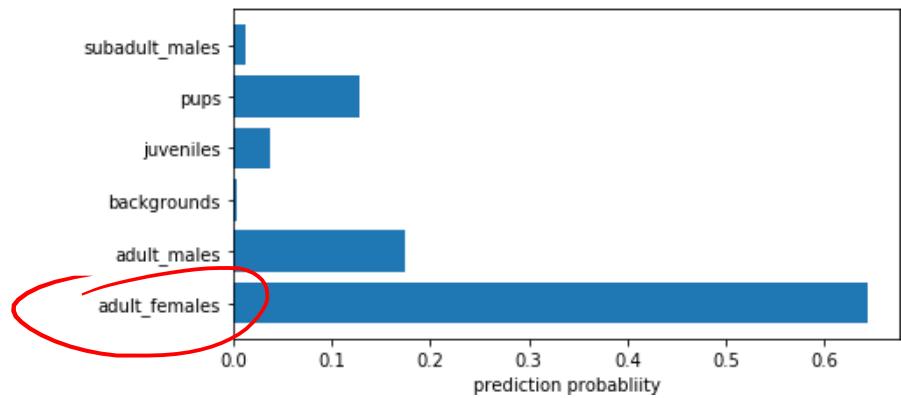
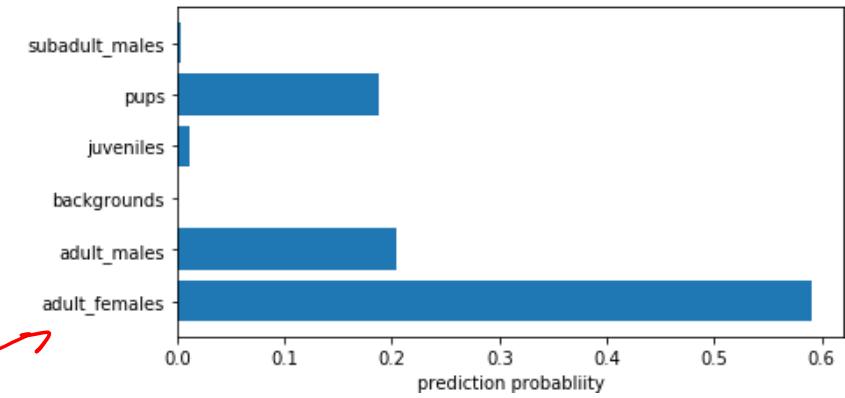
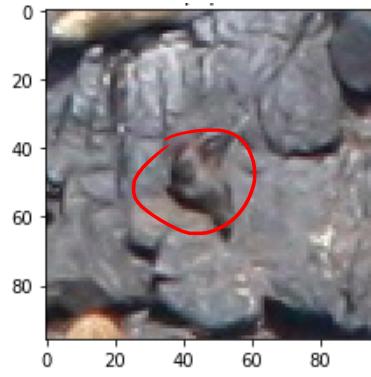
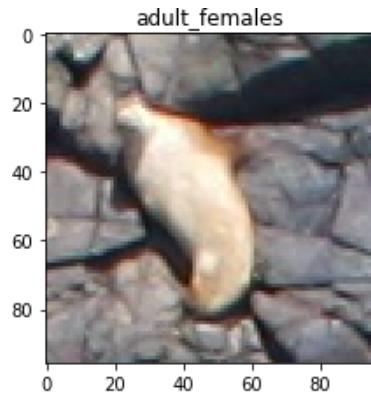
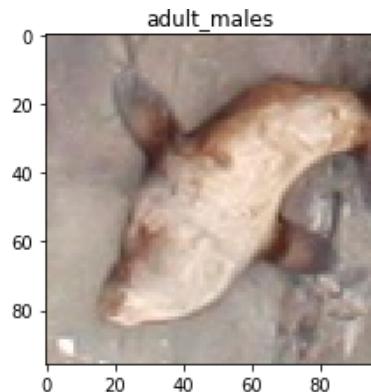


The Challenge

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



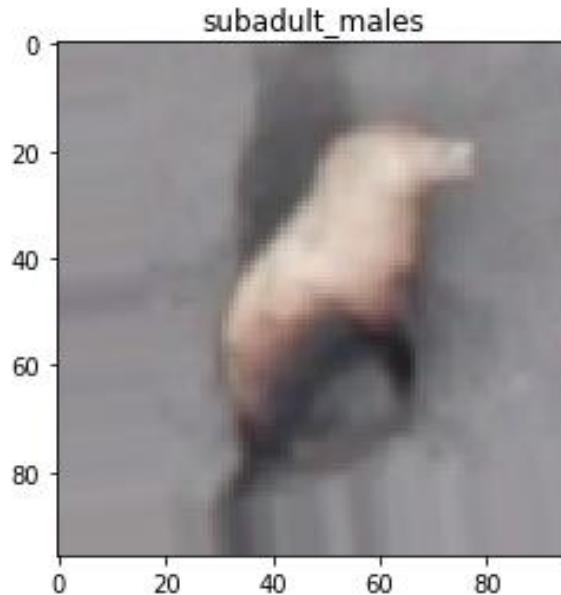
Credits Yinan Zhou

<https://github.com/marioZYN/FC-CNN-Demo>

Data Augmentation

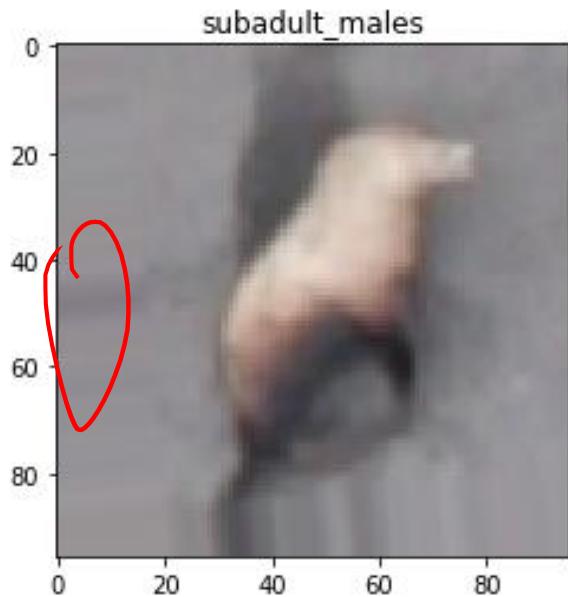
Often, each annotated image represents a class of images that are all likely to belong to the same class

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

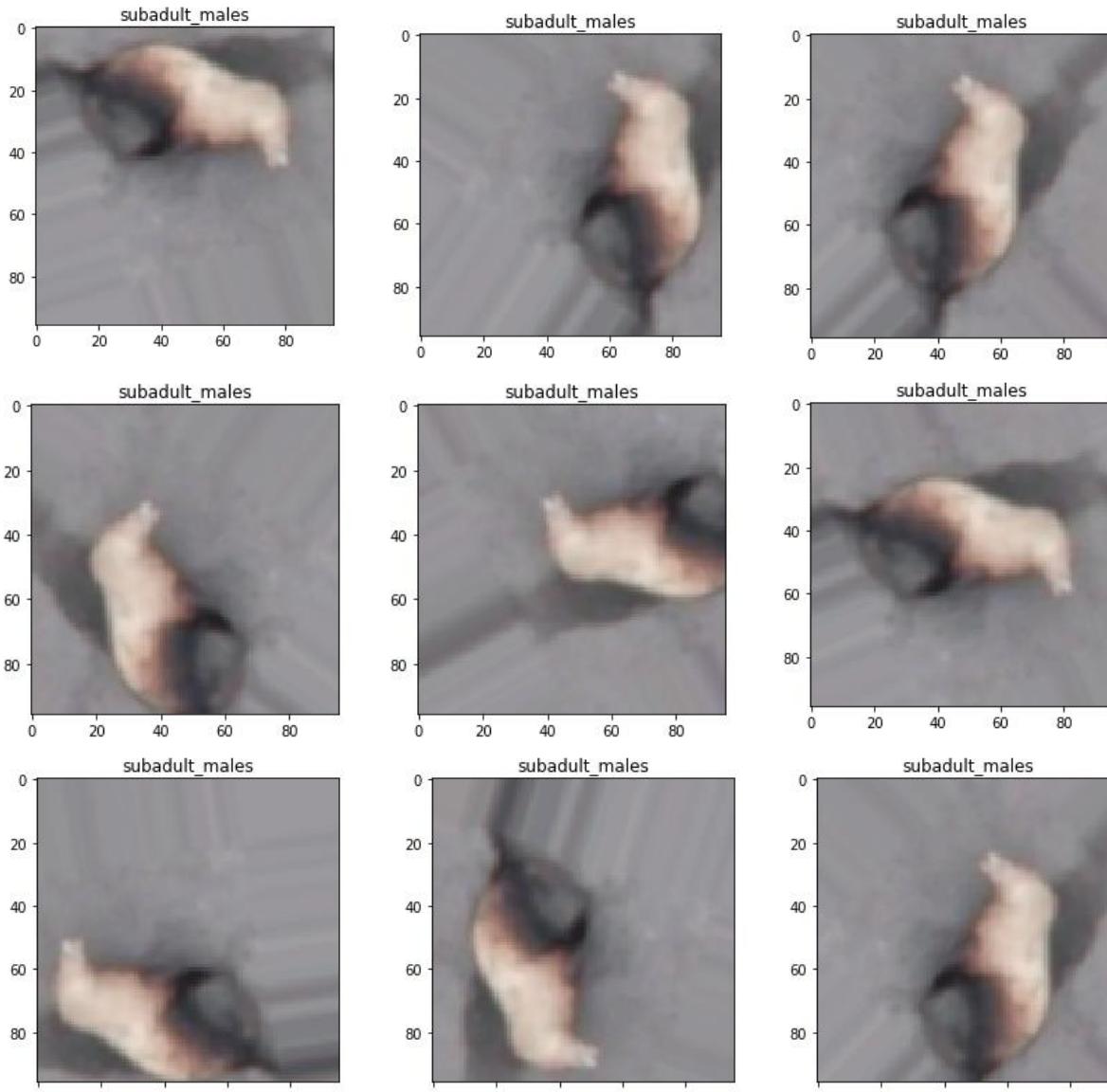


Data Augmentation

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

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

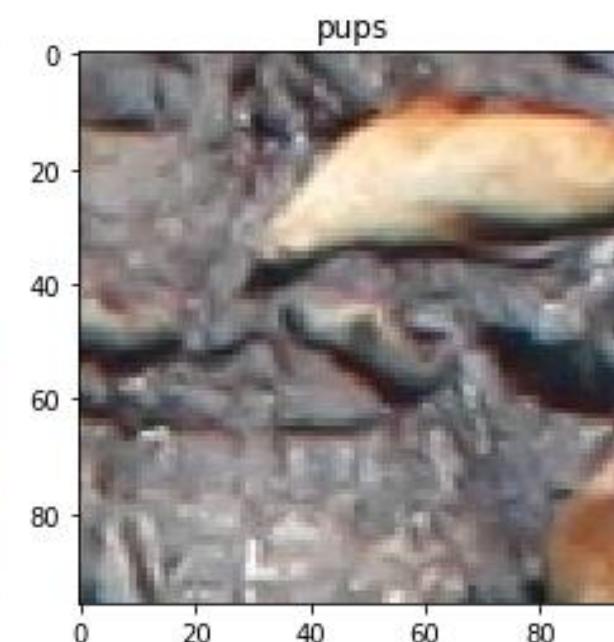
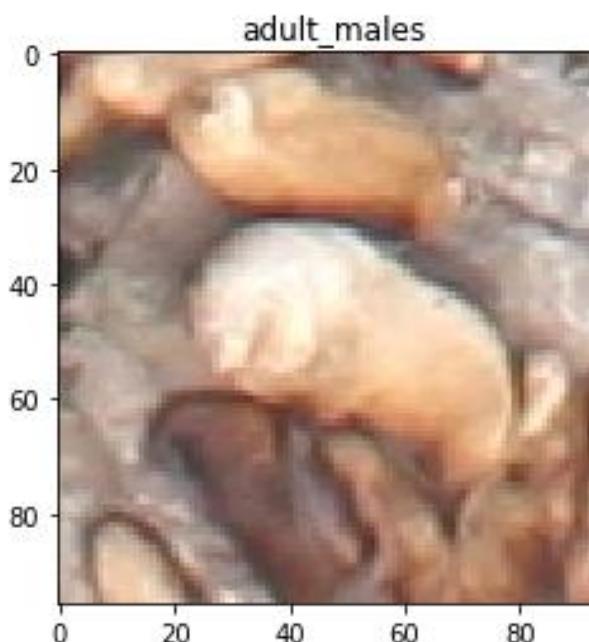
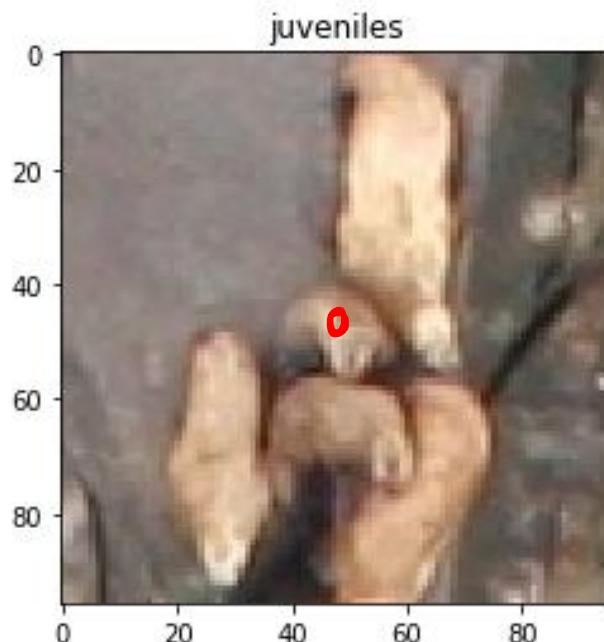
The Image generator has a method `flow_from_directory` that allows to load images in folder where different classes are arranged in subfolders.

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

However...

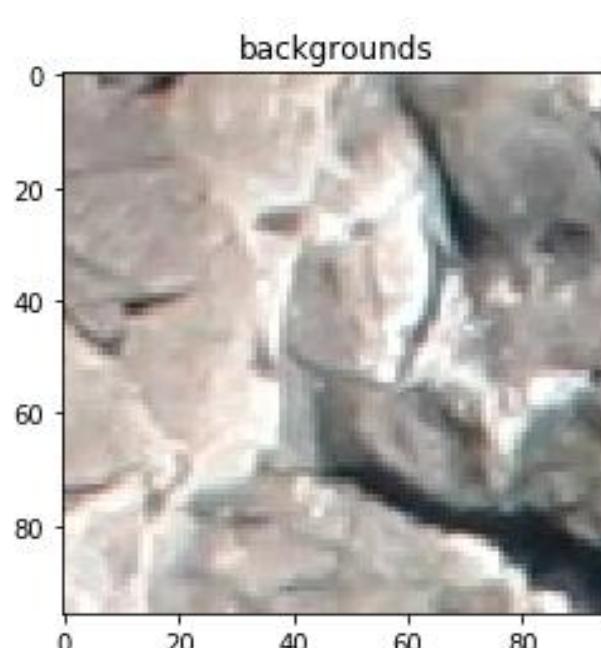
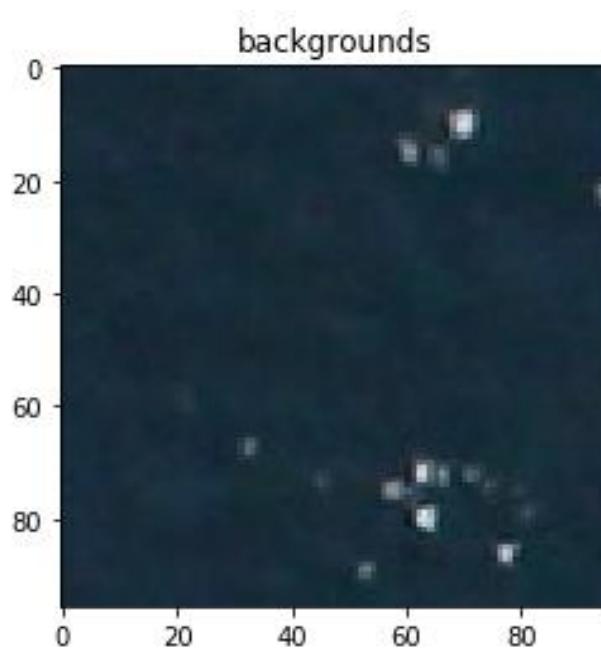
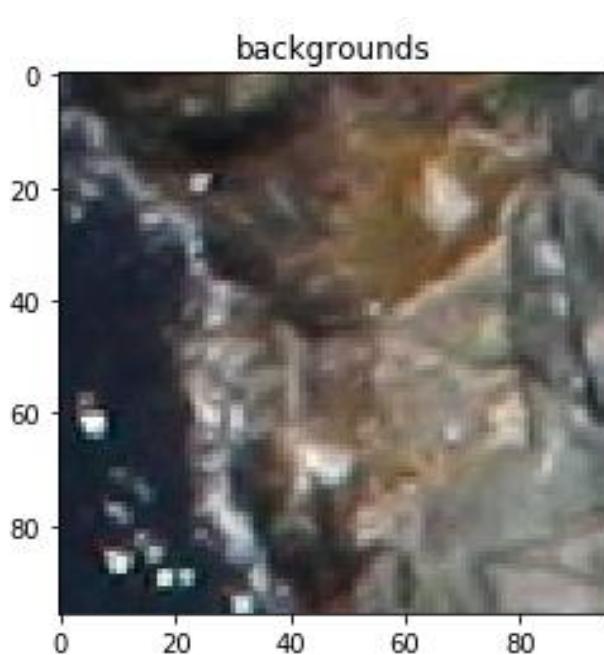
This sort of data augmentation might not be enough to capture the inter-class variability of images...

Superimposition of targets



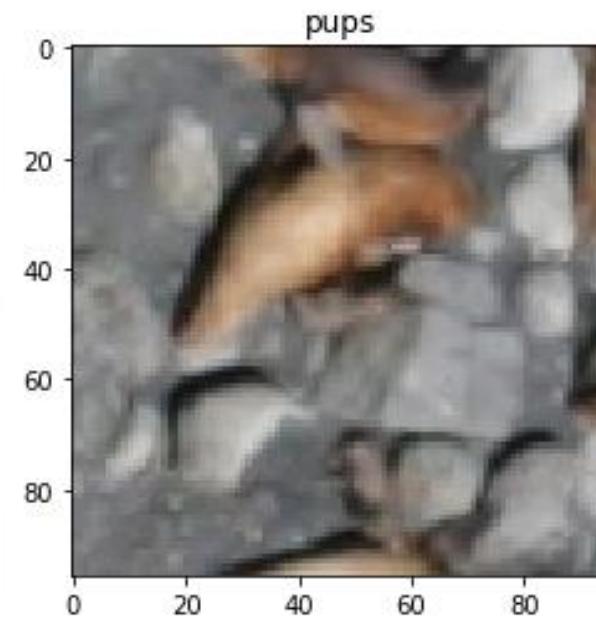
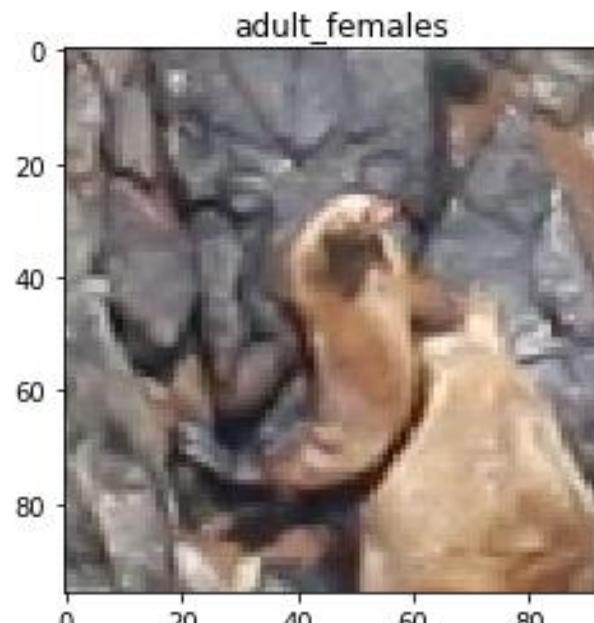
However...

Background variations



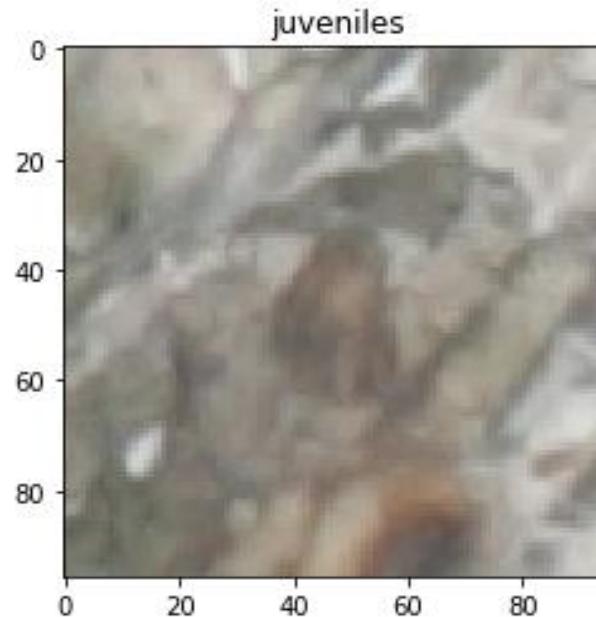
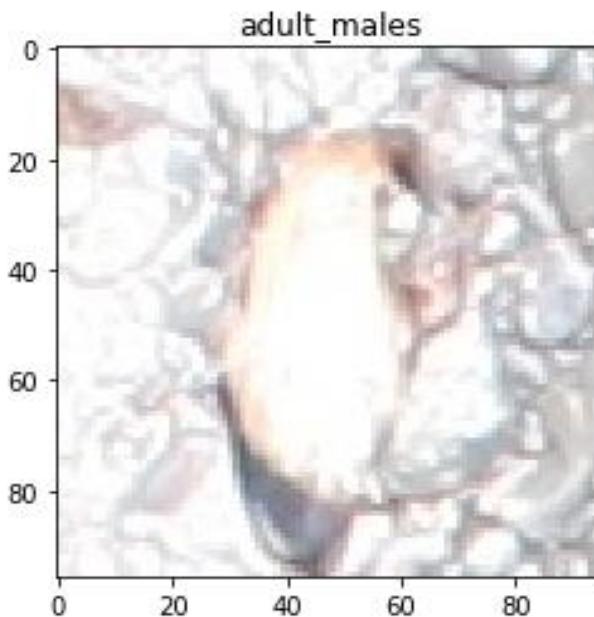
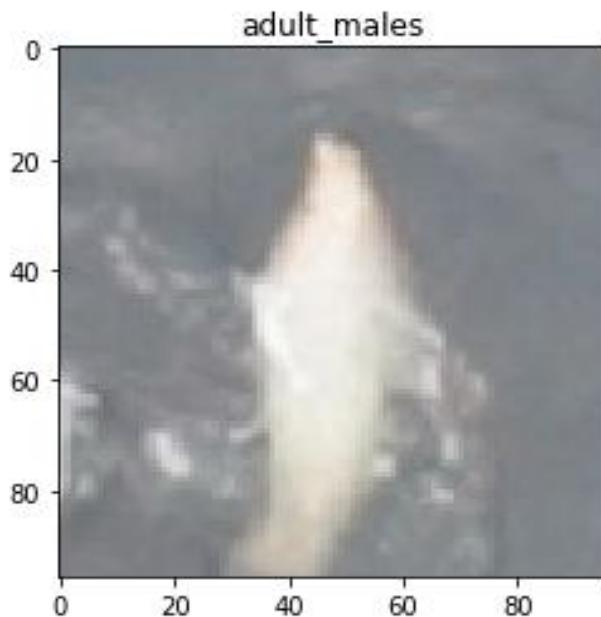
However...

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)
```

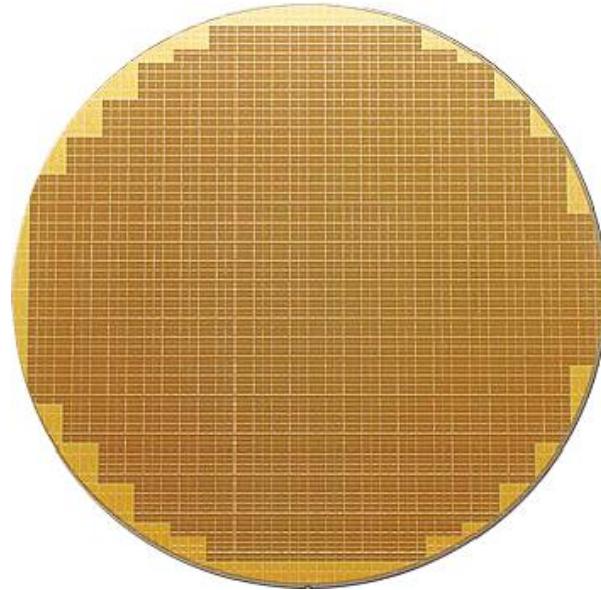
... in case you need some extra flexibility

Data Augmentation is often key..

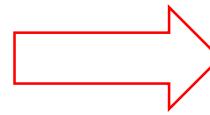
Collaboration with  life.augmented

For instance to monitor wafer manufacturing

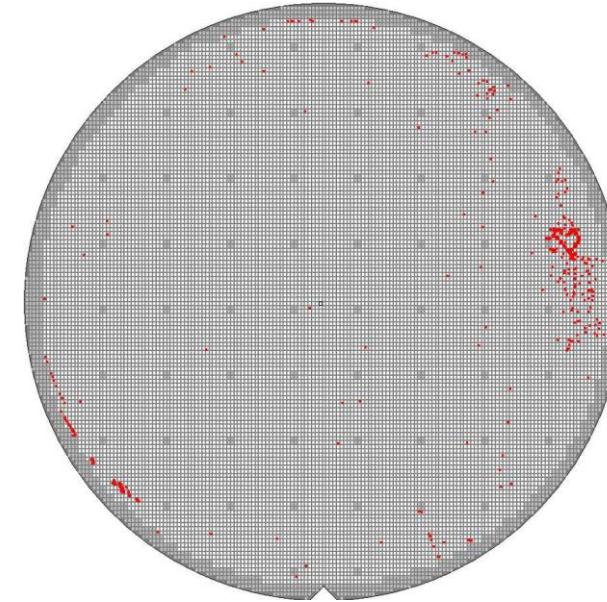
Wafer (containing chips)



Inspection tool



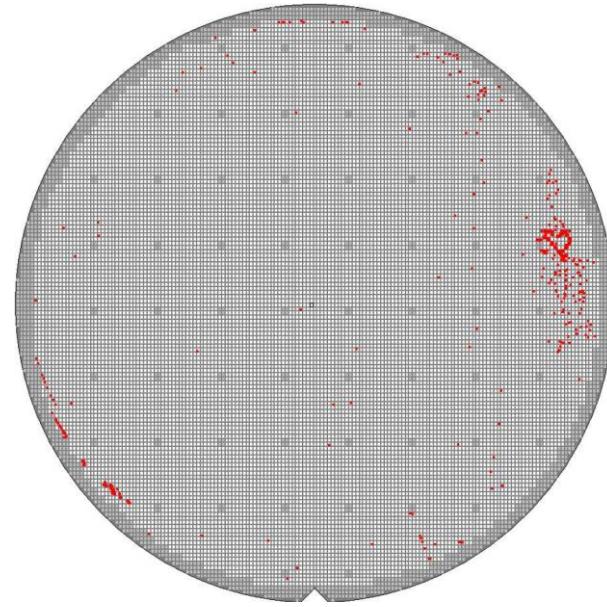
Wafer Defect Map



Data Augmentation is often key..

Collaboration with  life.augmented

Train a deep learning model to identify defective patterns

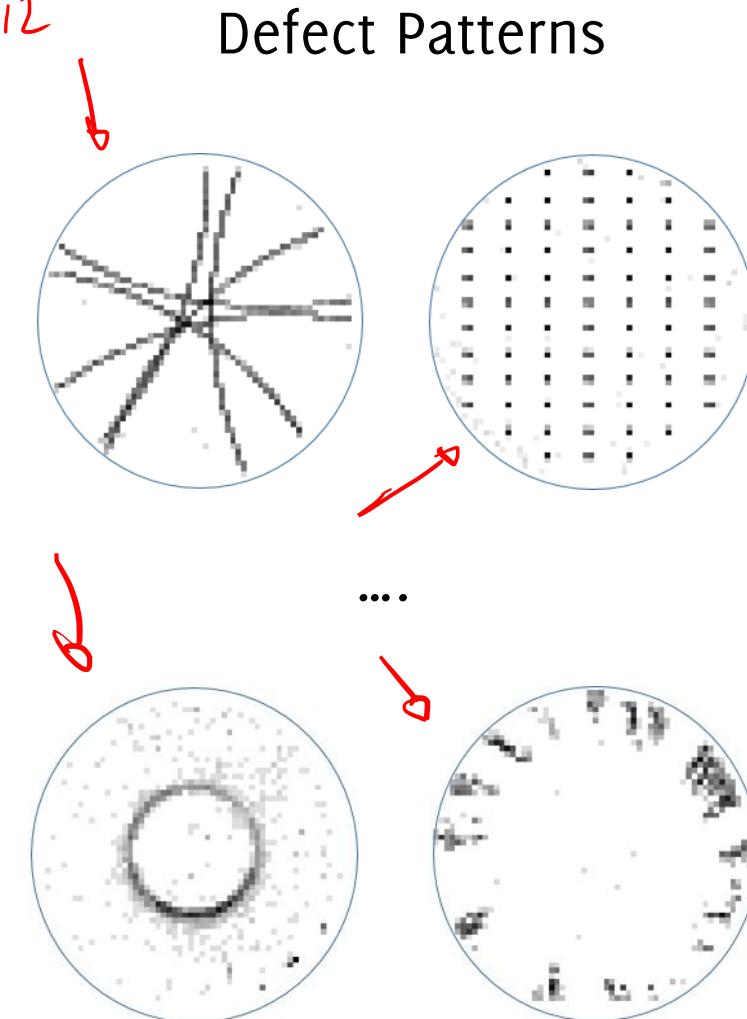


Wafer Defect Map

Deep Learning



12

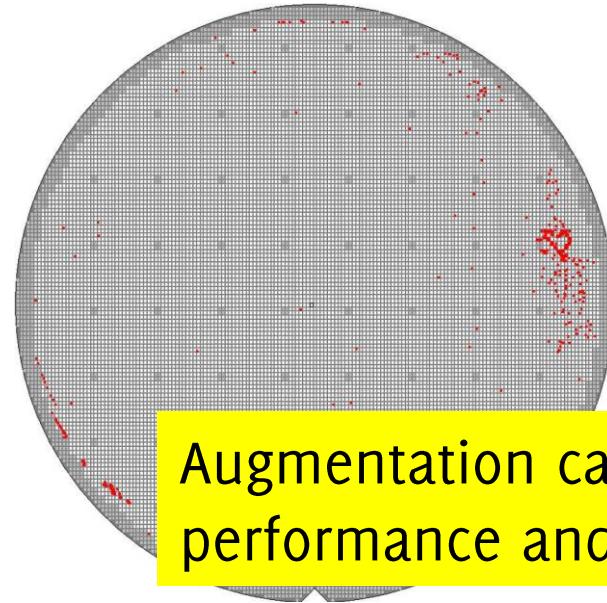


Defect Patterns

Data Augmentation is often key..

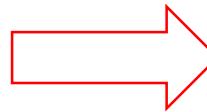
Collaboration with  life.augmented

Train a deep learning model to identify defective patterns



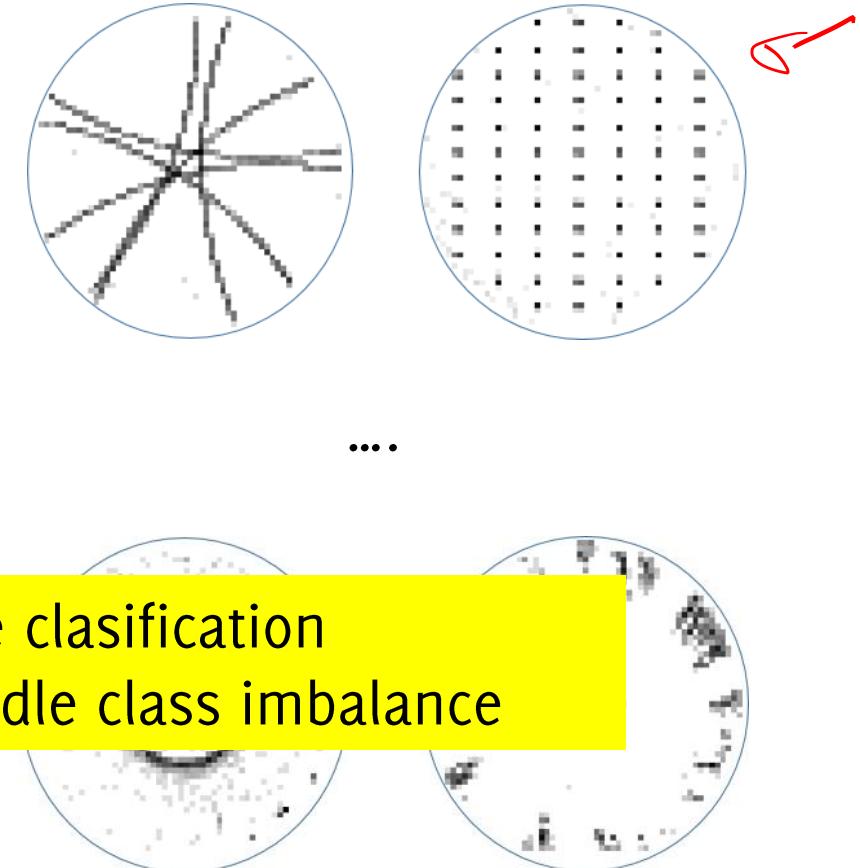
Wafer Defect Map

Deep Learning



Augmentation can greatly improve classification performance and successfully handle class imbalance

Defect Patterns



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.

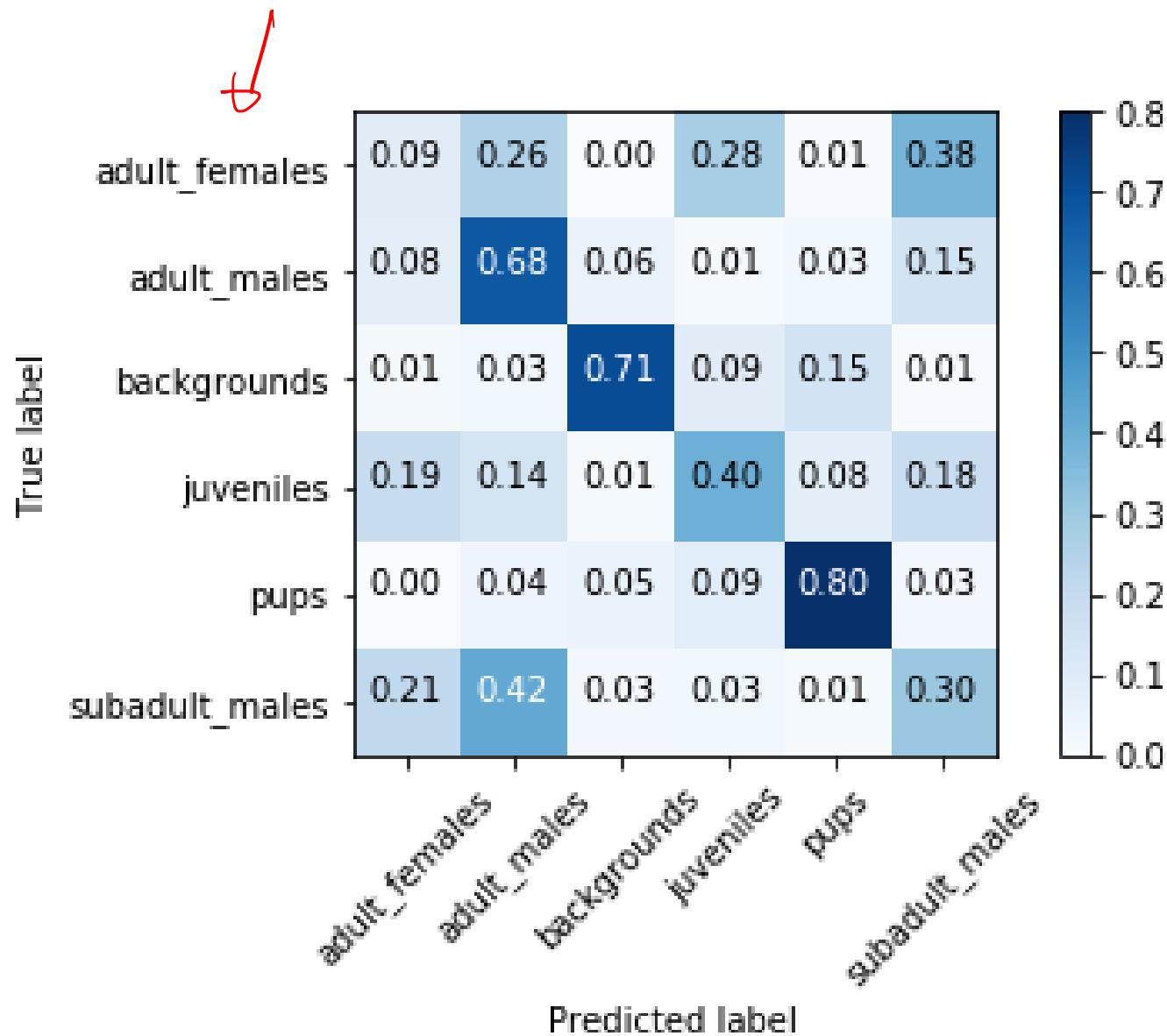
- Perform random augmentation of each test image I
- Average the predictions of each augmented image
- Take the average vector of posterior for defining the final guess

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

Confusion Matrix

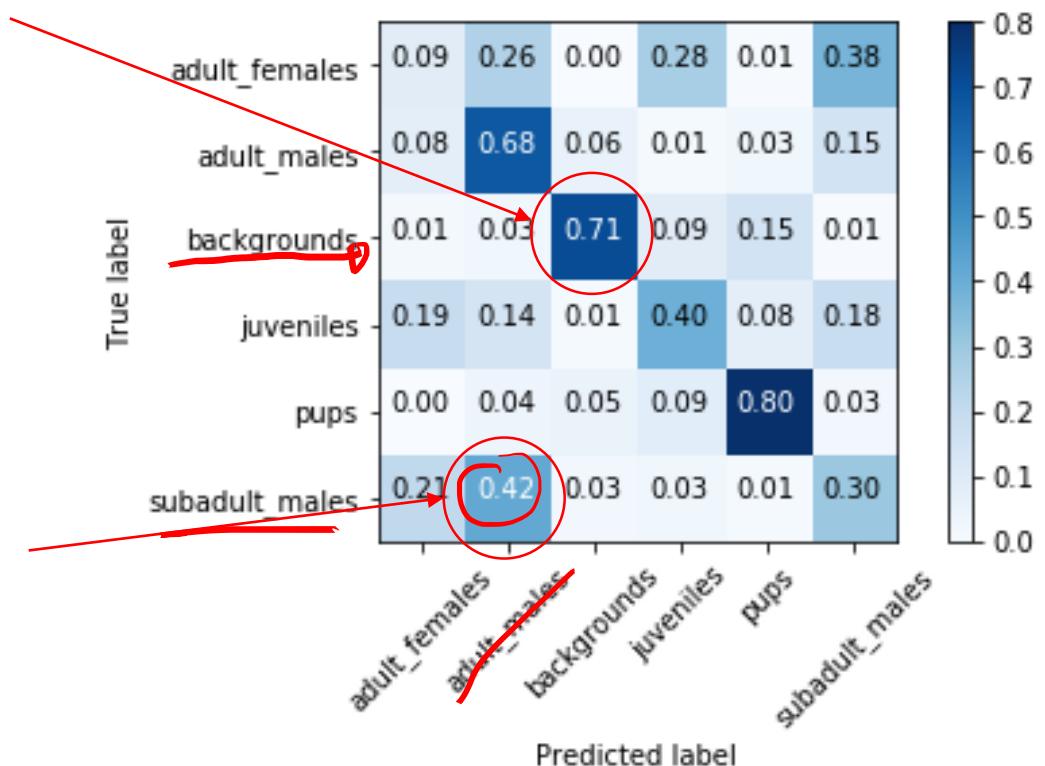


Confusion Matrix

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

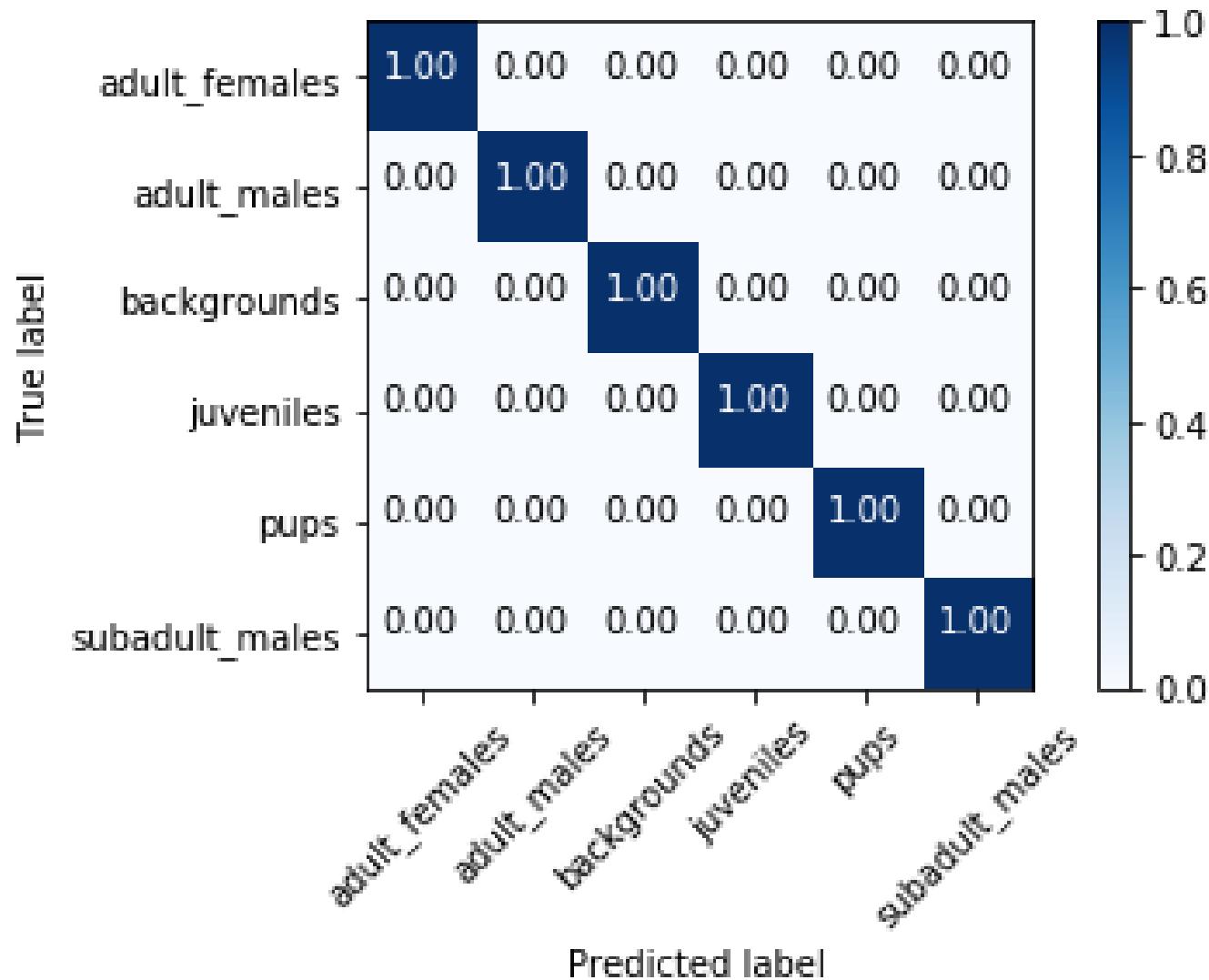
71% of background patches have been correctly classified as background

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

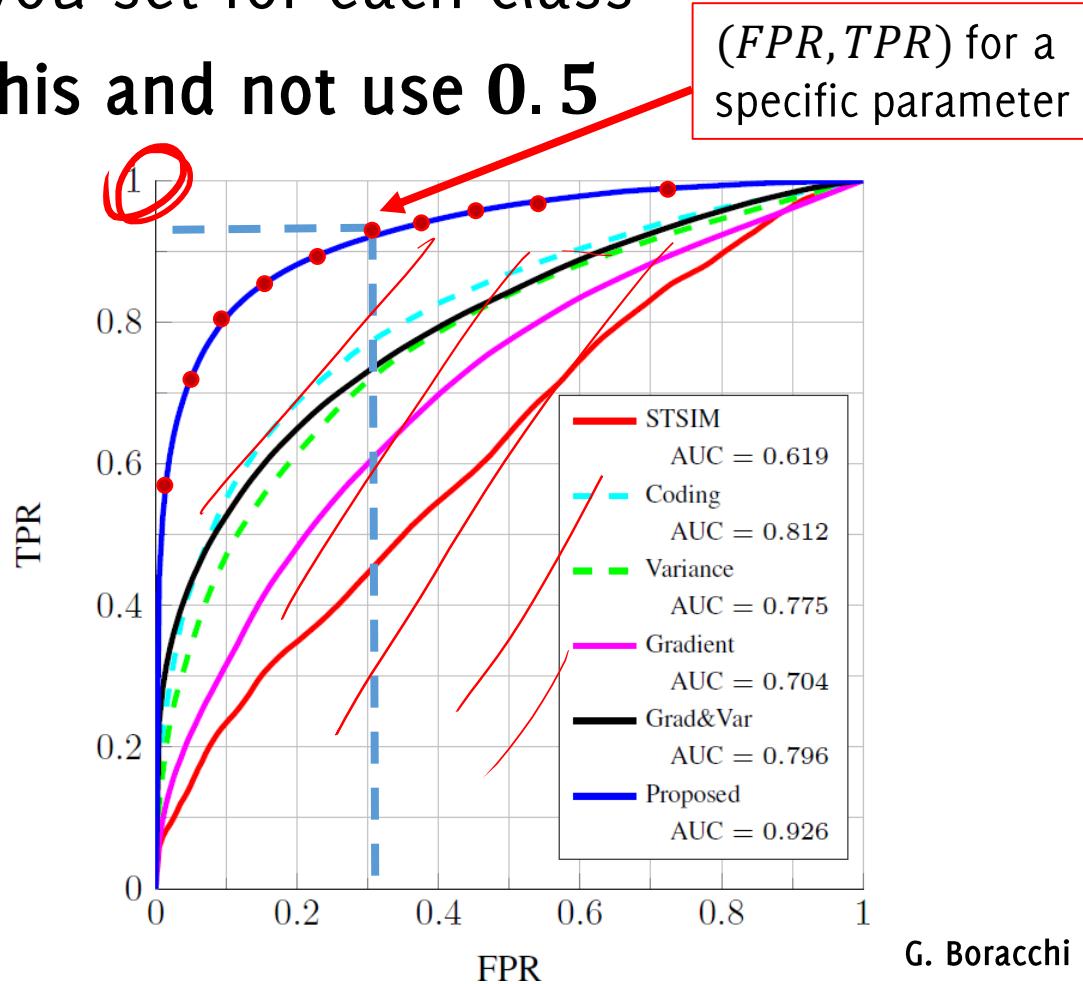
The ideal detector would achieve:

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

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

- Transfer Learning
- CNN Visualization
- (Fully Convolutional CNN)

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

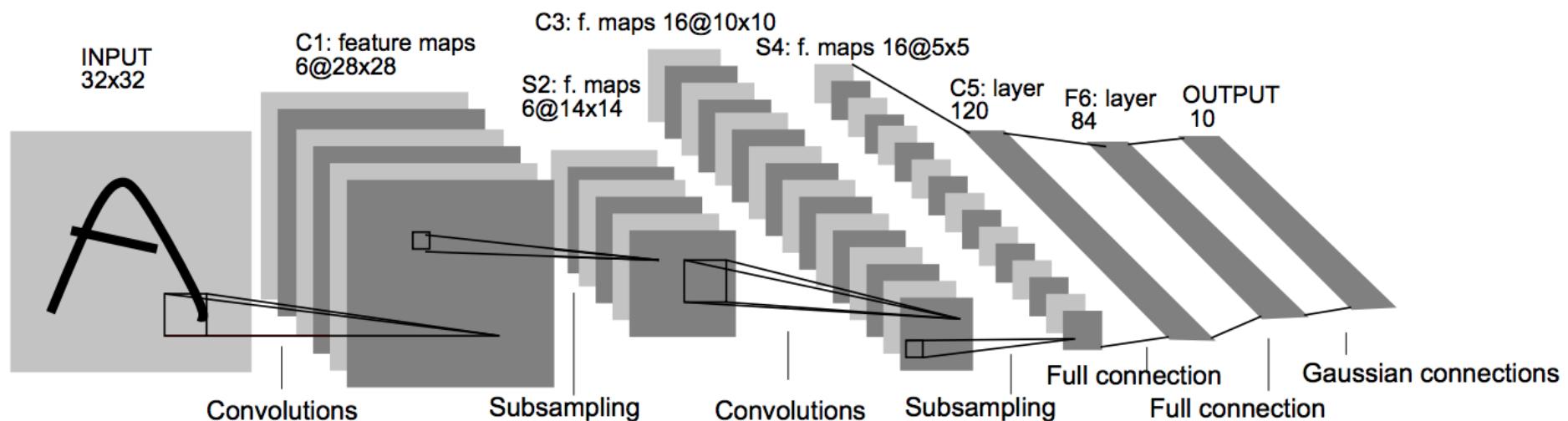
Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent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

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



LeNet-5 (1998)

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

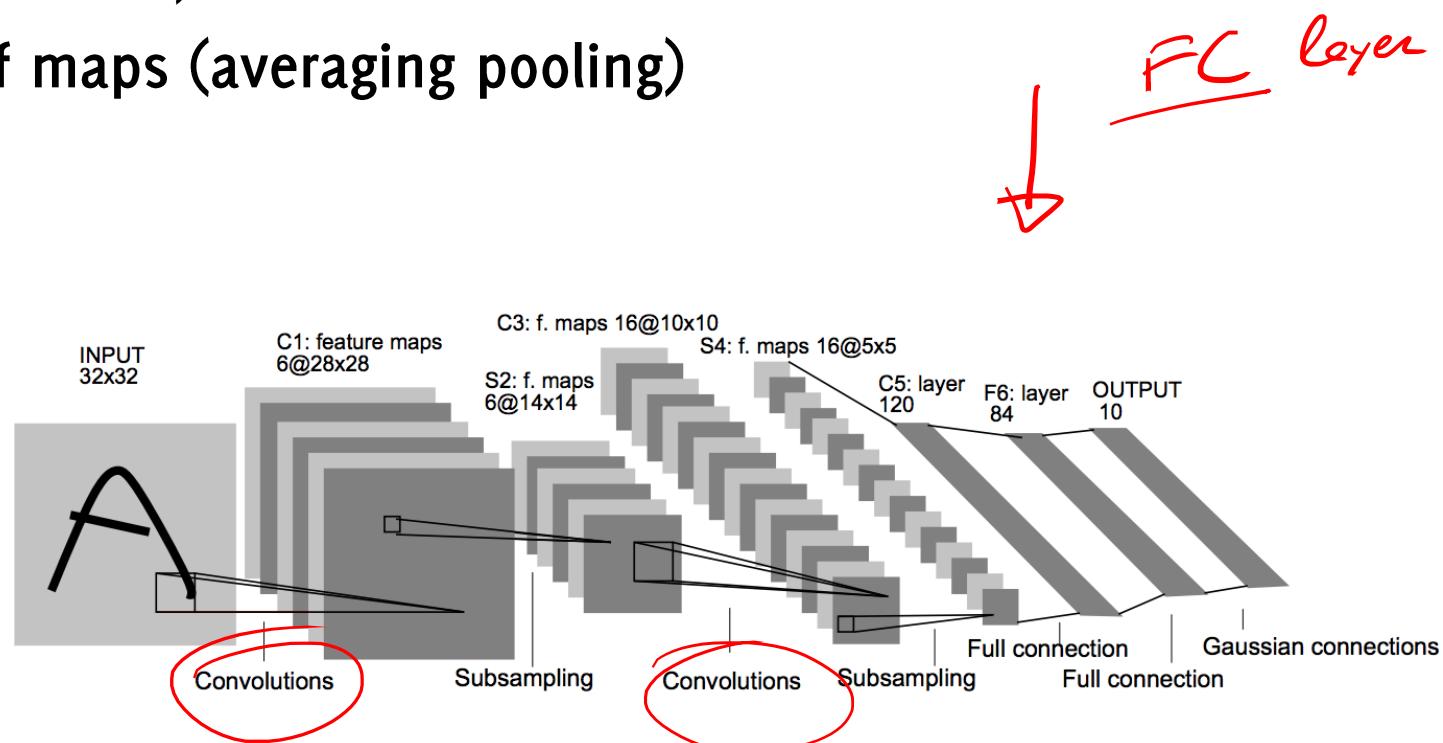
Pixels in an image are highly correlated: there is no point to separately feed each pixel value through a classification network: add learnable convolutional layers

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



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

AlexNet (2012)

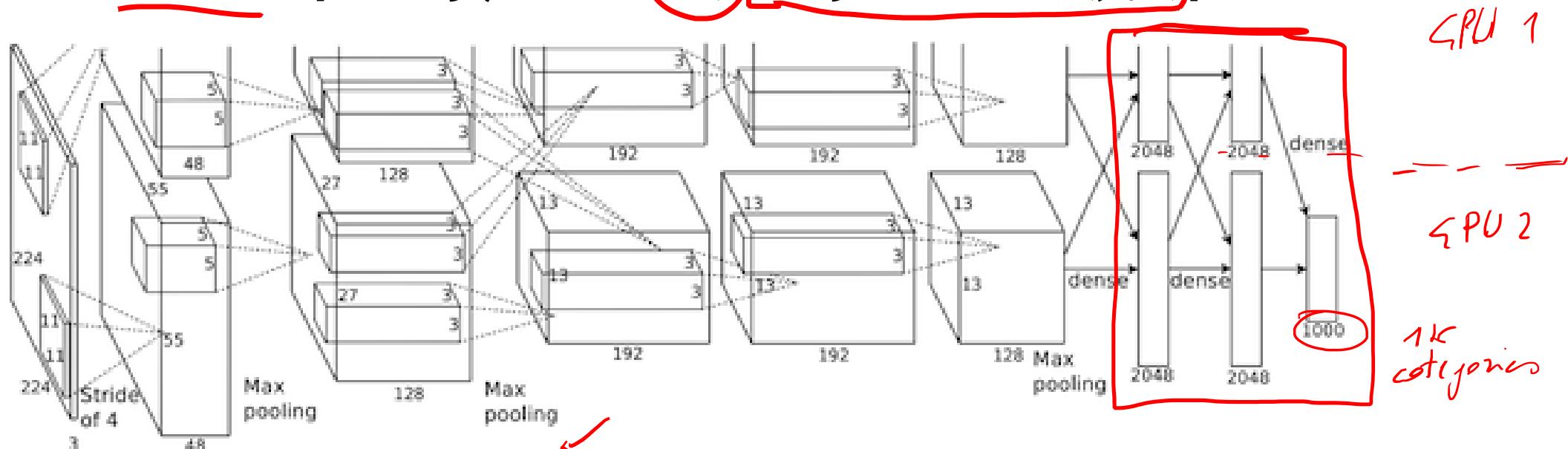
Developed by Alex Krizhevsky et al. in 2012 and won Imagenet competition

Architecture is quite similar to LeNet-5:

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

Input size $224 \times 224 \times 3$ (the paper says $227 \times 227 \times 3$)

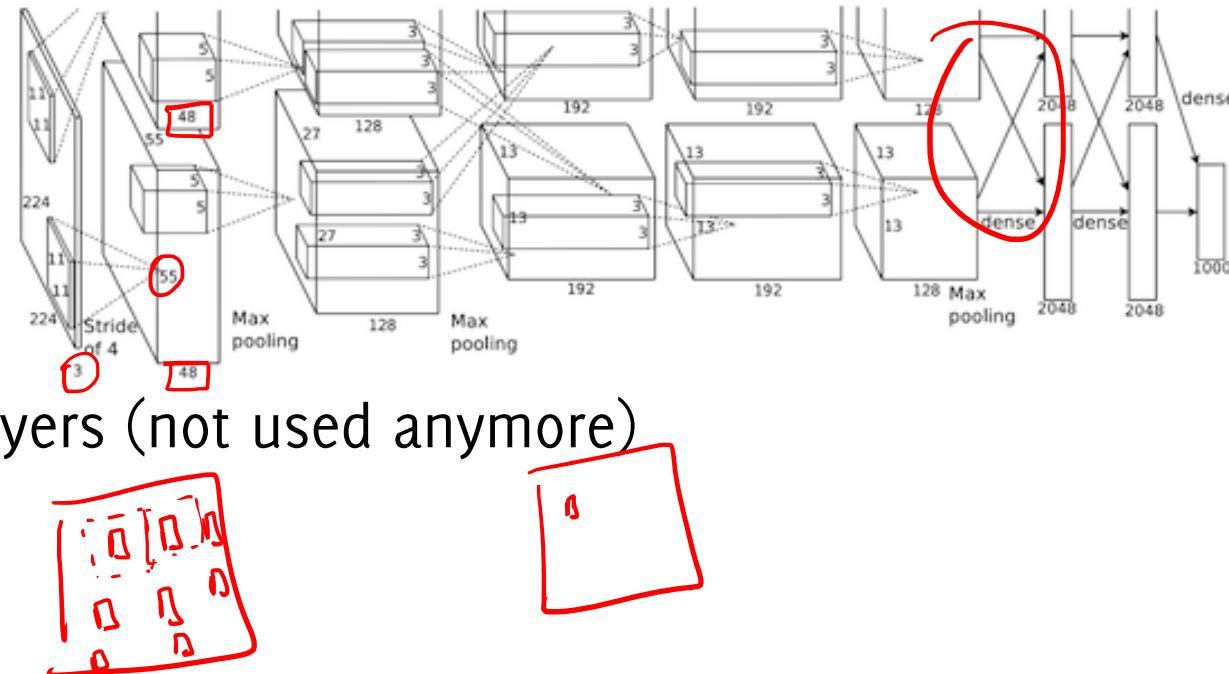
Parameters: 60 million [Conv: 3.7million (6%), FC: 58.6 million (94%)]



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 $55 \times 55 \times 48$ separated over two GTX 580 GPUs (3GB, 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%

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

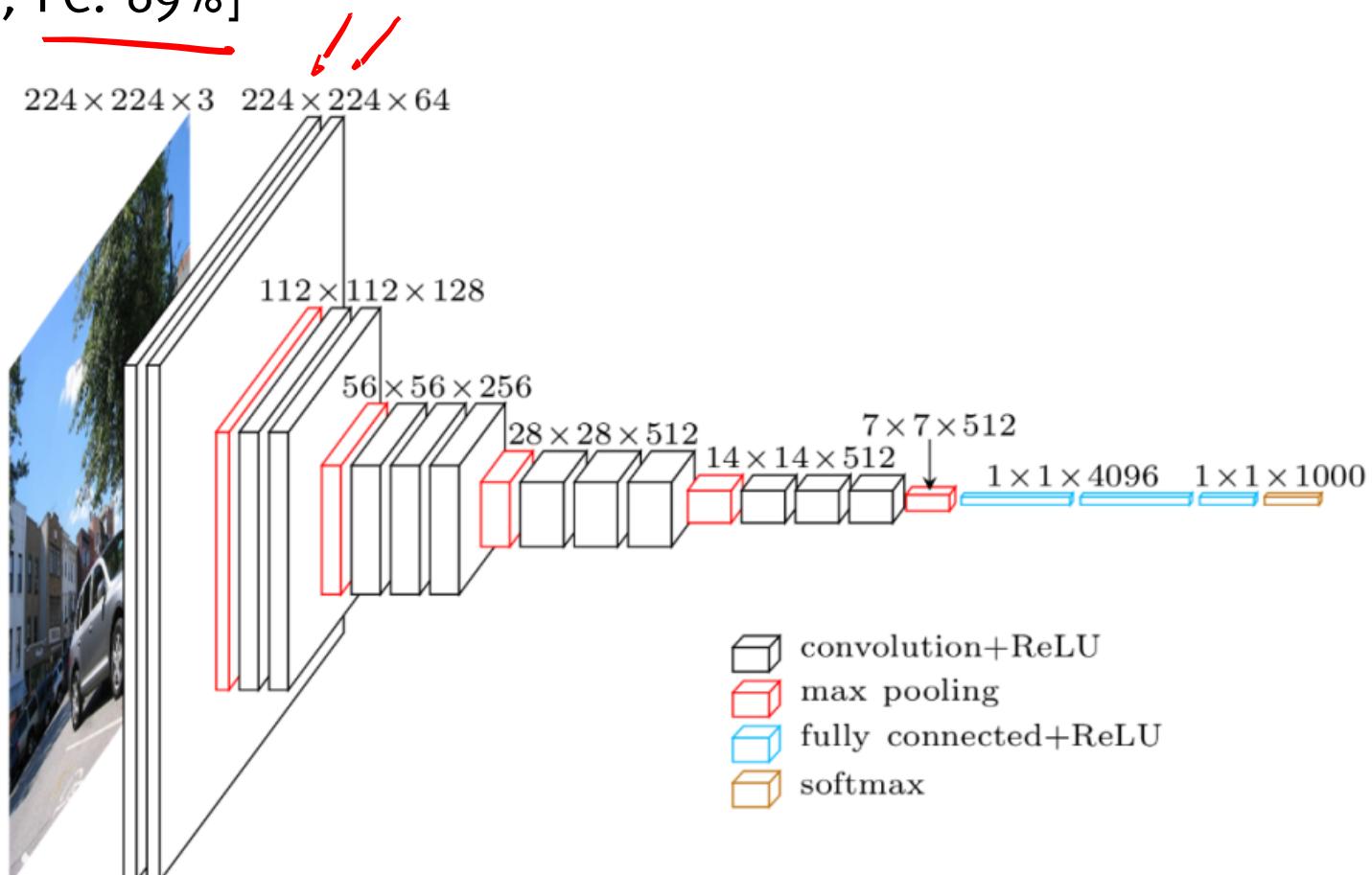
ABSTRACT

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.

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%]



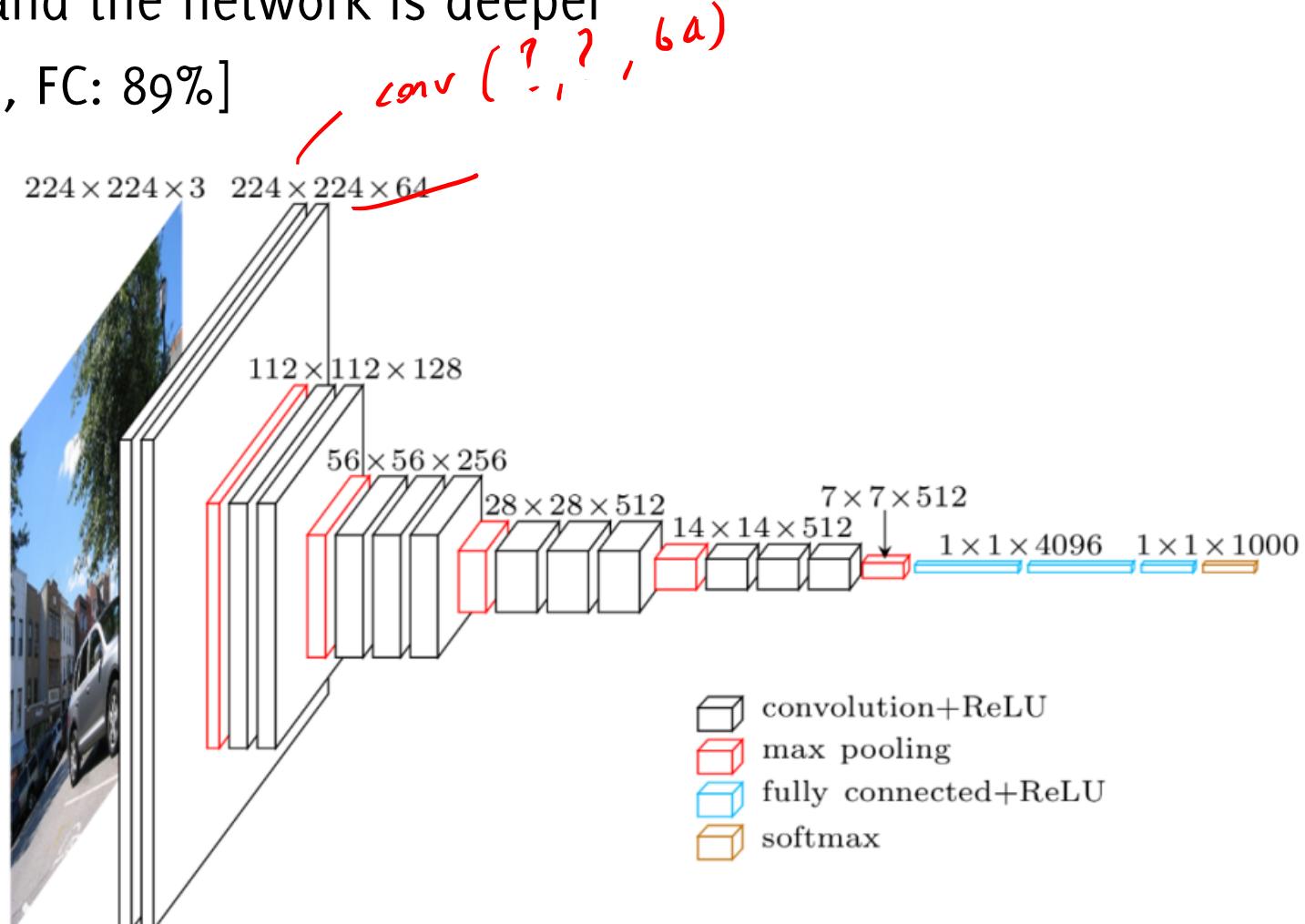
VGG16 (2014)

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 $224 \times 224 \times 3$



VGG16 (2014): Smaller Filter, Deeper Network

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

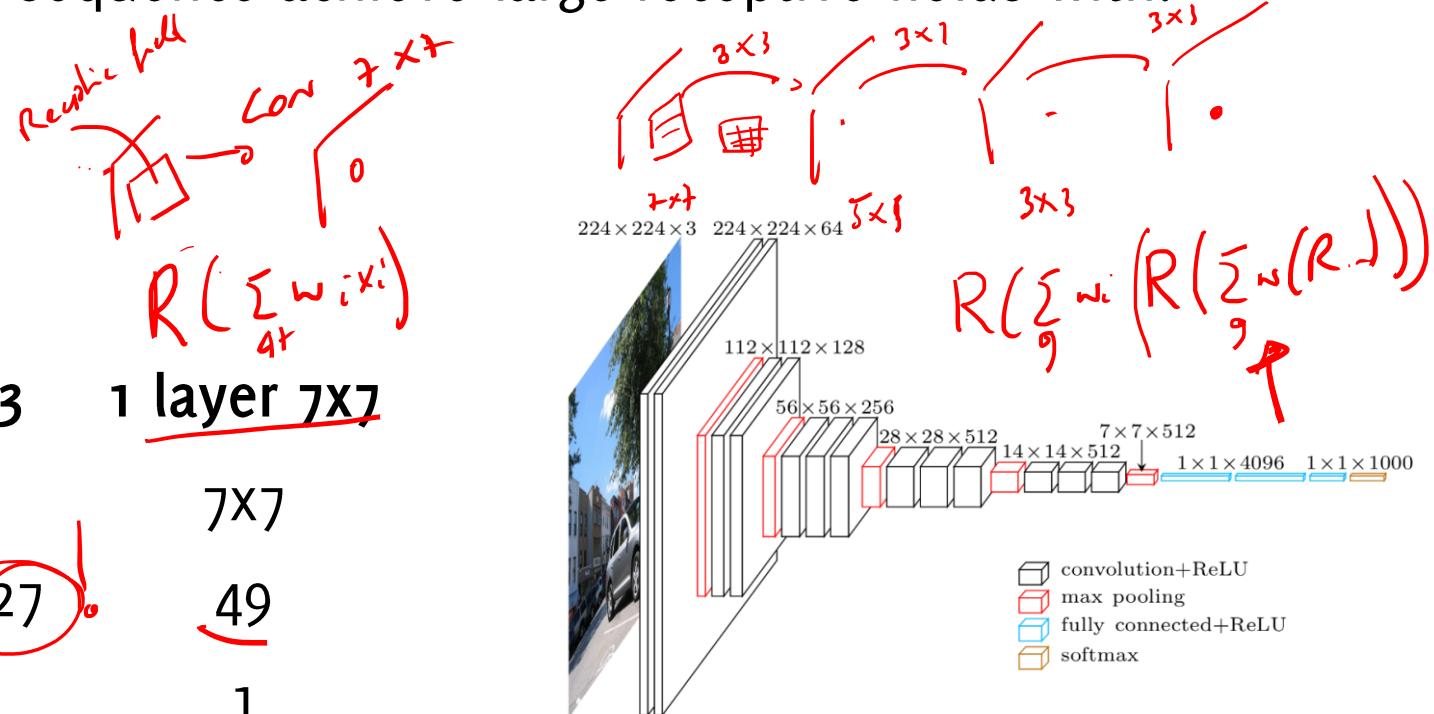
[...] Fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3×3) convolution filters in all layers.

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

- less parameters
 - more nonlinearities

than larger filters in a single layer

	<u>3 layers 3x3</u>	<u>1 layer 7x7</u>
Receptive field	7x7	7x7
Nr of parameters	$3 \times 3 \times 3 = 27$	49
Nr of nonlinearities	3	1



VGG16

Layer (type)	Output Shape	Param #	Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0		[...]	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792	block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928	block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856	block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584	block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	flatten (Flatten)	(None, 25088)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168	fc1 (Dense)	(None, 4096)	102764544
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080	fc2 (Dense)	(None, 4096)	16781312
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080	predictions (Dense)	(None, 1000)	4097000
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	Total params: 138,357,544		
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	Trainable params: 138,357,544		
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808	Non-trainable params: 0		
block4_conv3 (Conv2D)	(None, 28, 28, 512) [...]	2359808			

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

Most parameters in FC layers : 123,642,856

VGG16

Layer (type)	Output Shape	Param #	Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0		[...]	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792	block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928	block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block2_conv1 (Conv2D)				14, 14, 512)	2359808
block2_conv2 (Conv2D)				7, 7, 512)	0
block2_pool (MaxPooling2D)				25088)	0
block3_conv1 (Conv2D)				4096)	102764544
block3_conv2 (Conv2D)				4096)	16781312
block3_conv3 (Conv2D)				1000)	4097000
block3_pool (MaxPooling2D)					
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	Trainable params: 138,357,544 Non-trainable params: 0		
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808			
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808			
	[...]				

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)

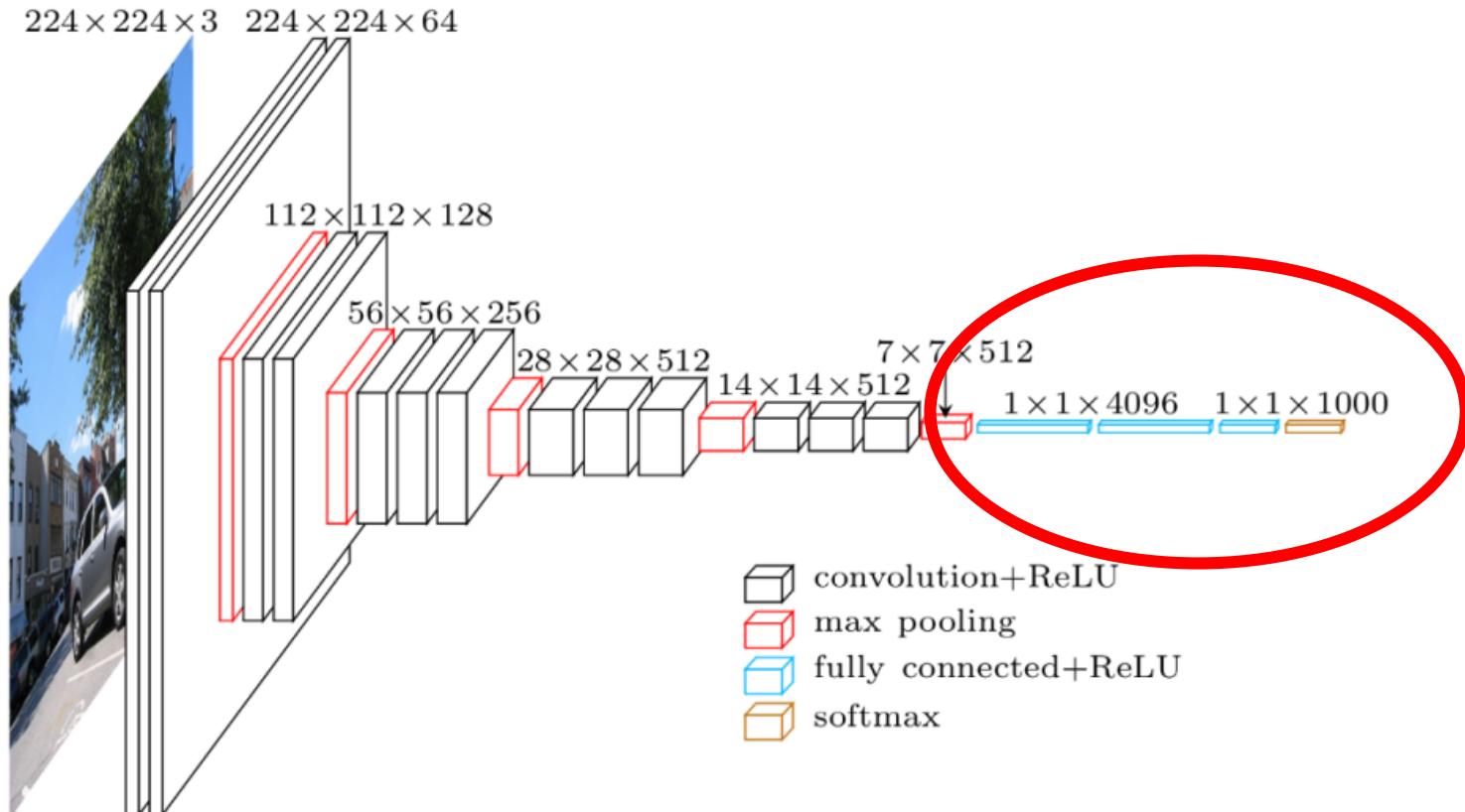
Popular Architectures to be continued...

Limited Amount of Data: Transfer Learning

Training a CNN with Limited Aumont of Data

Transfer Learning

How to use pre-trained models to solve different problems

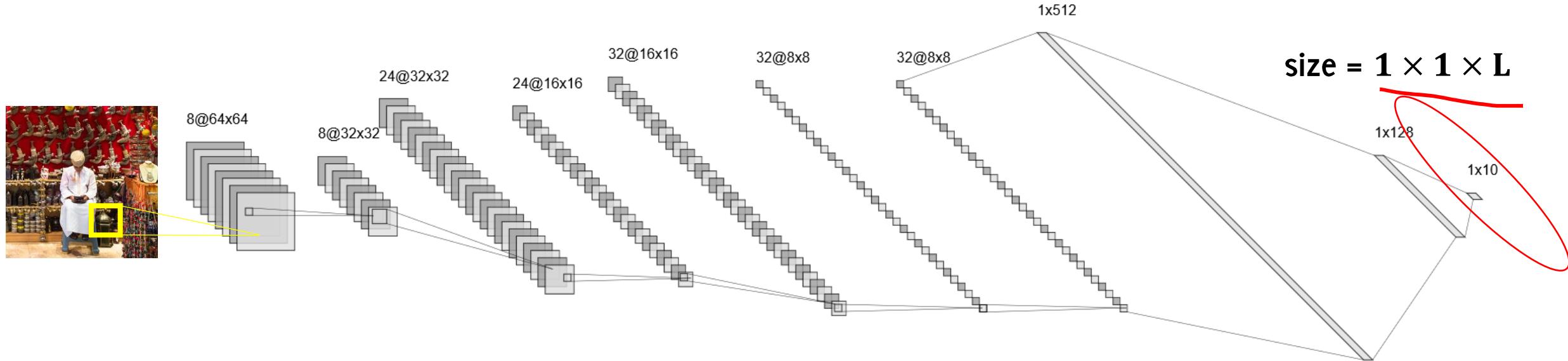


Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

G. Boracchi

Convolutional Neural Networks (CNN)

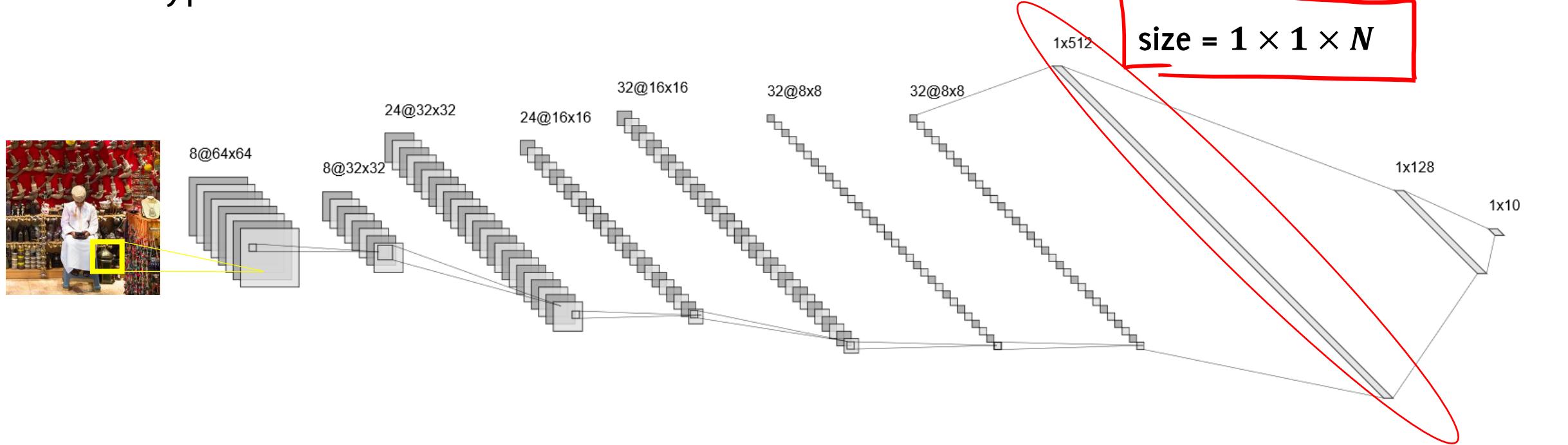
The typical architecture of a convolutional neural network



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

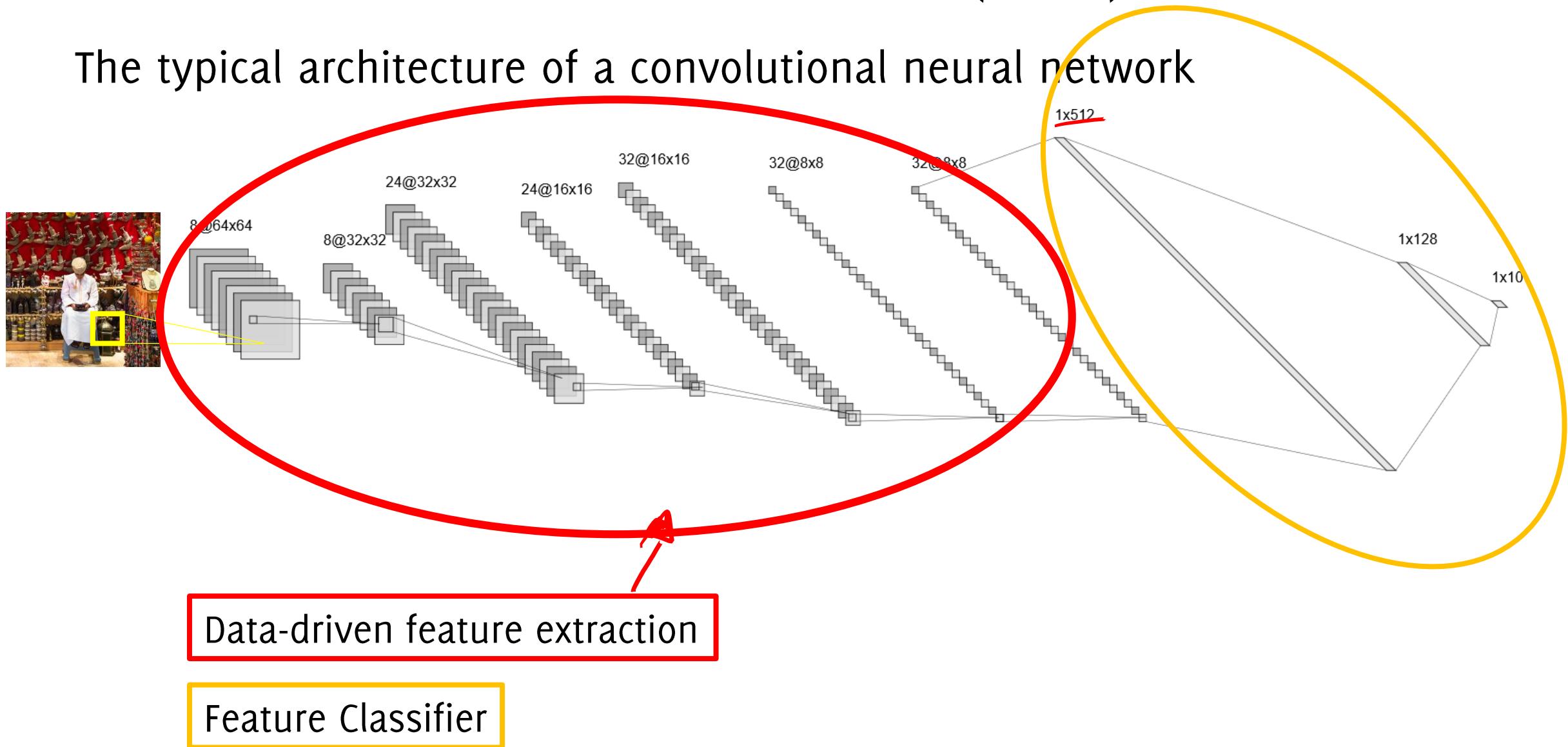


The input of the fully connected layer can be seen as a data-driven descriptor of the input image, i.e. a feature vector. These features are:

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

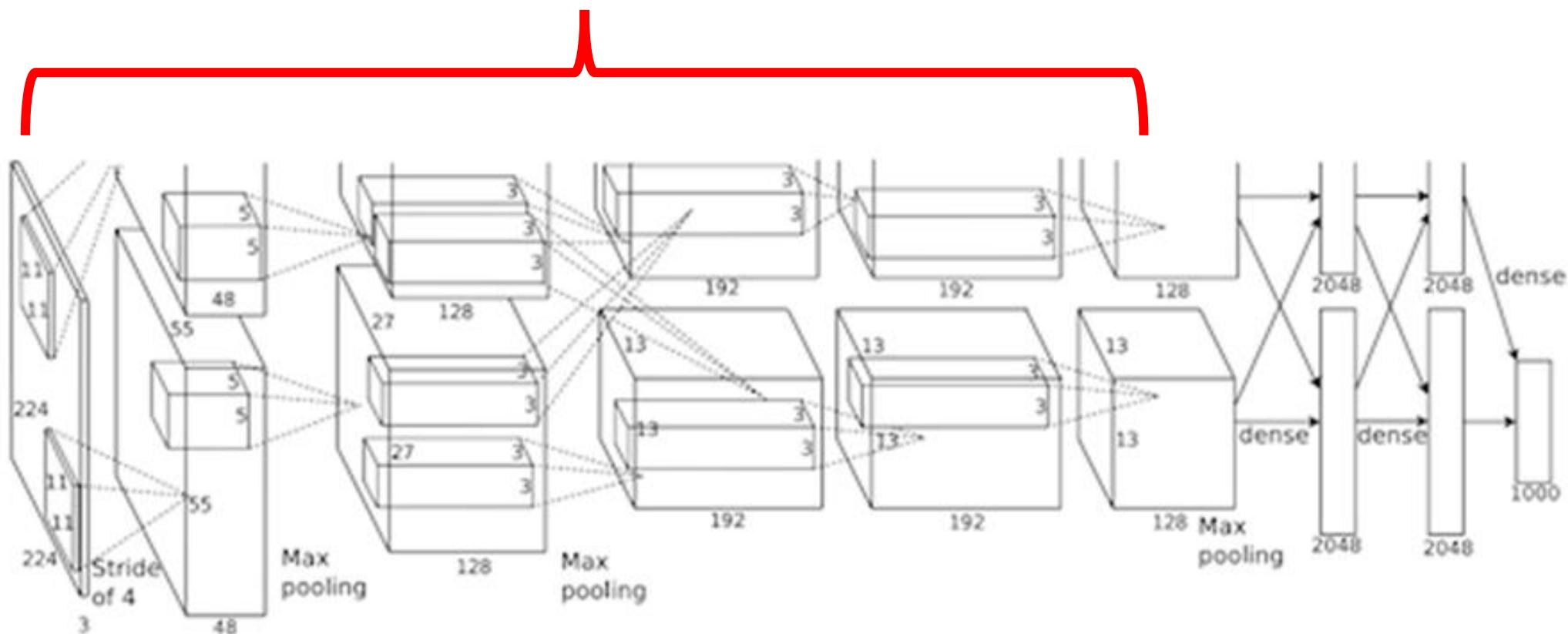
Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



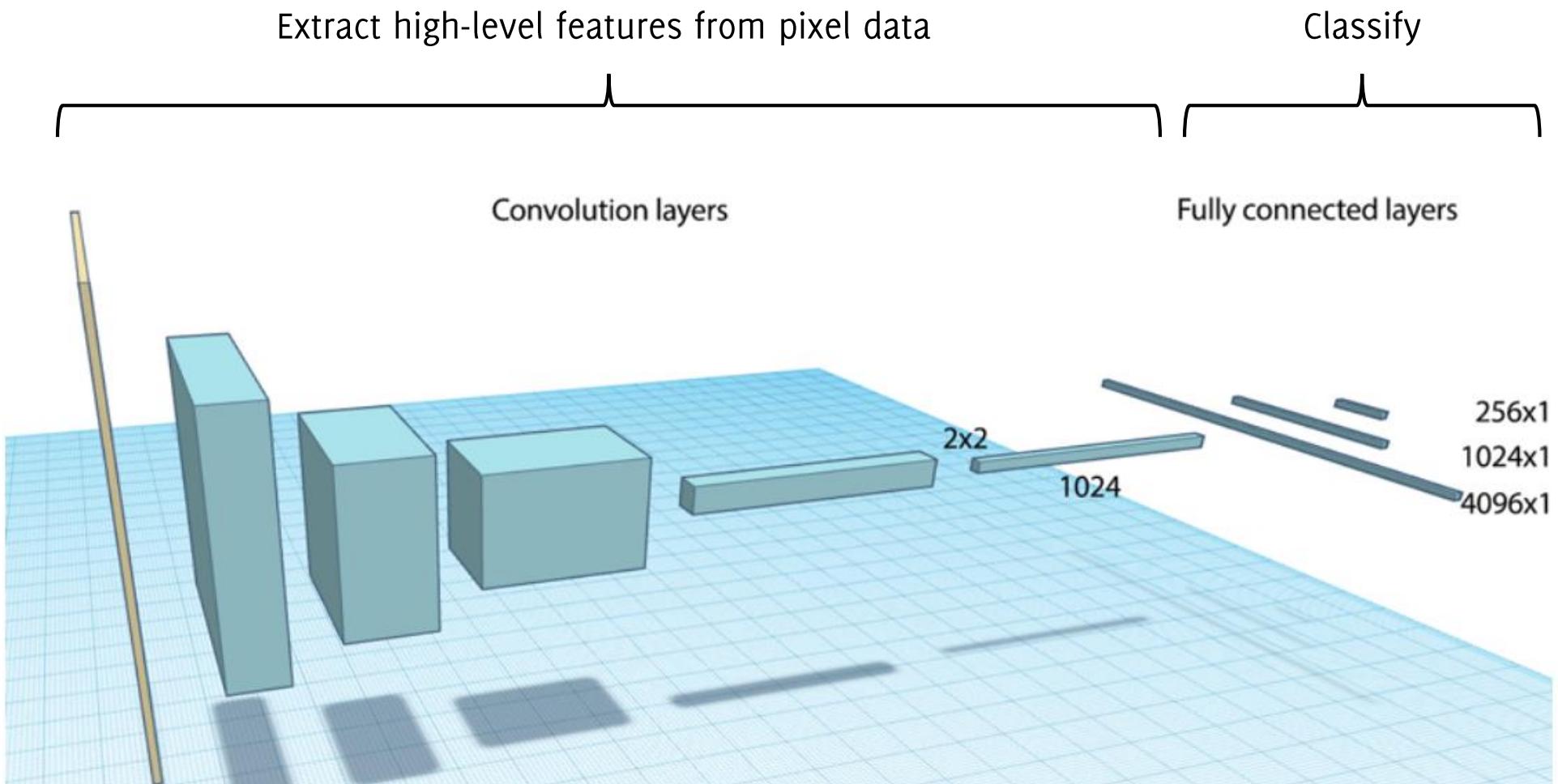
Deep architectures

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)

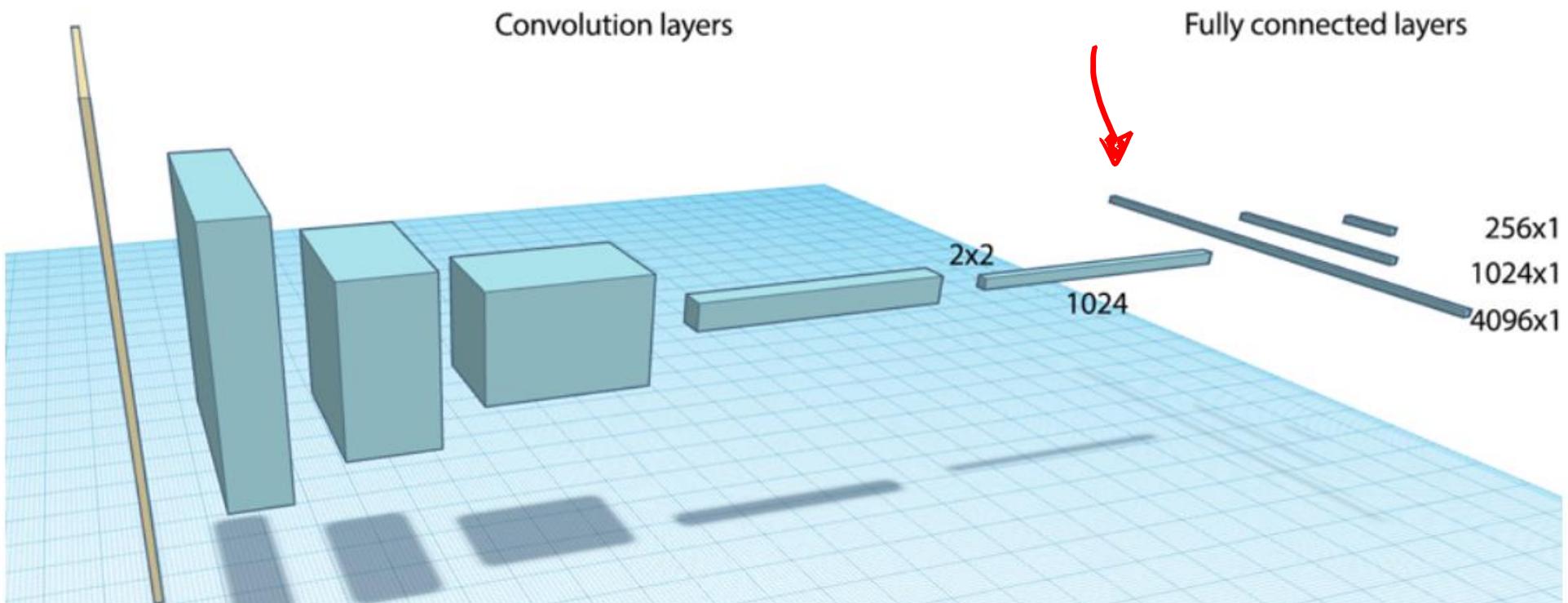
A typical architecture



A typical architecture

As we move to deeper layers:

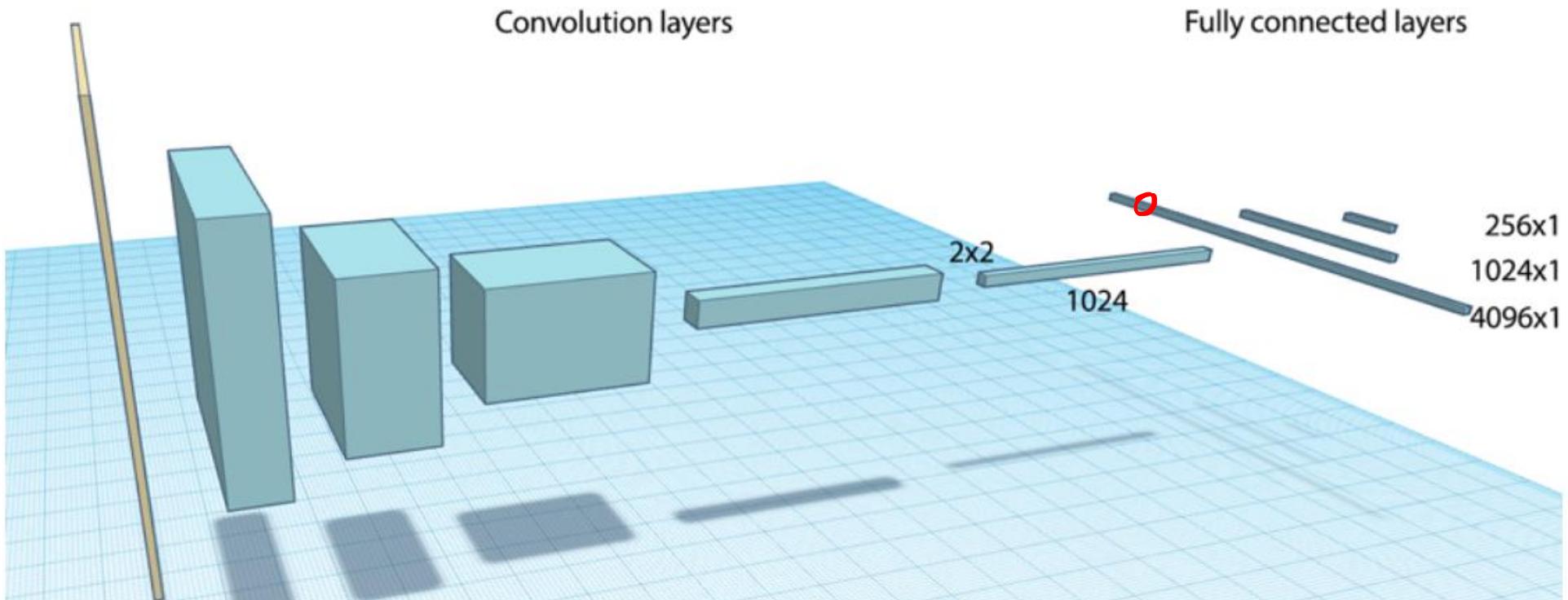
- spatial resolution is reduced
- the number of maps increases



A typical architecture

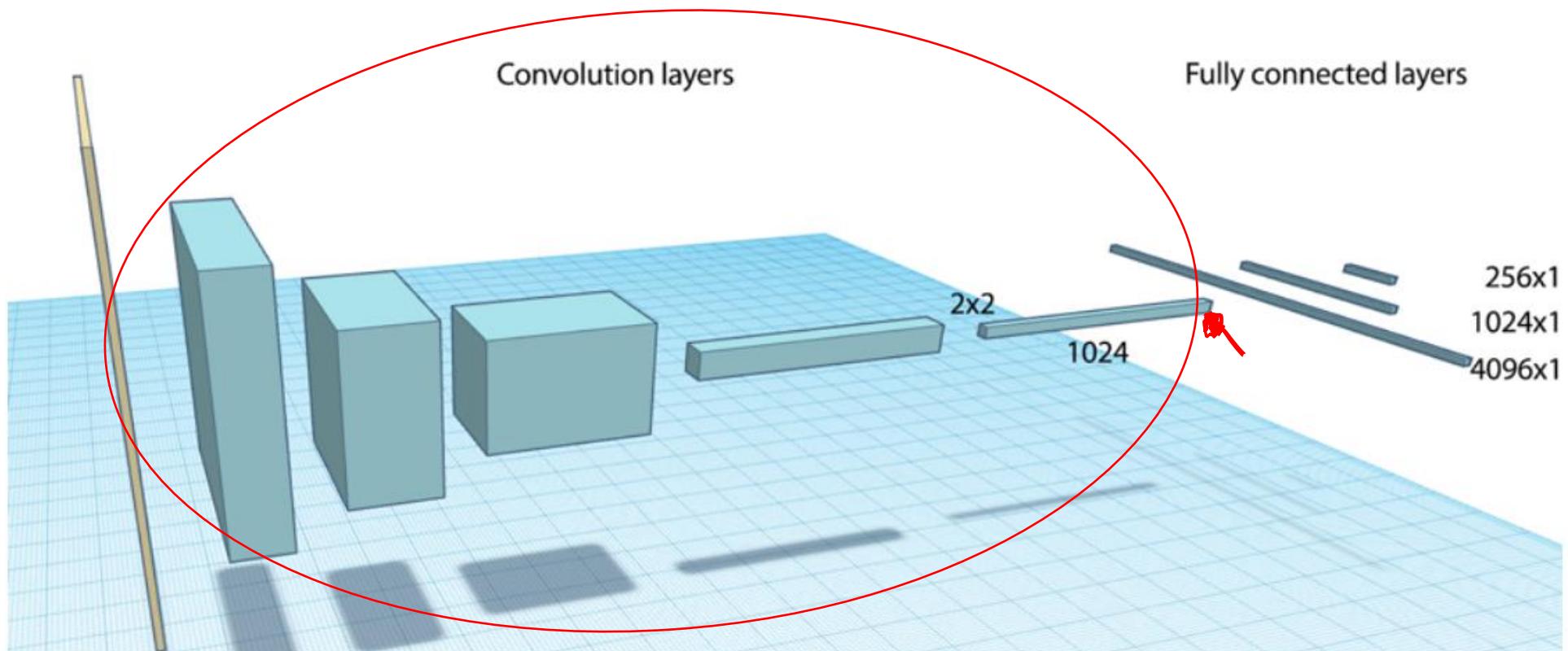
As we move to deeper layers:

- 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!



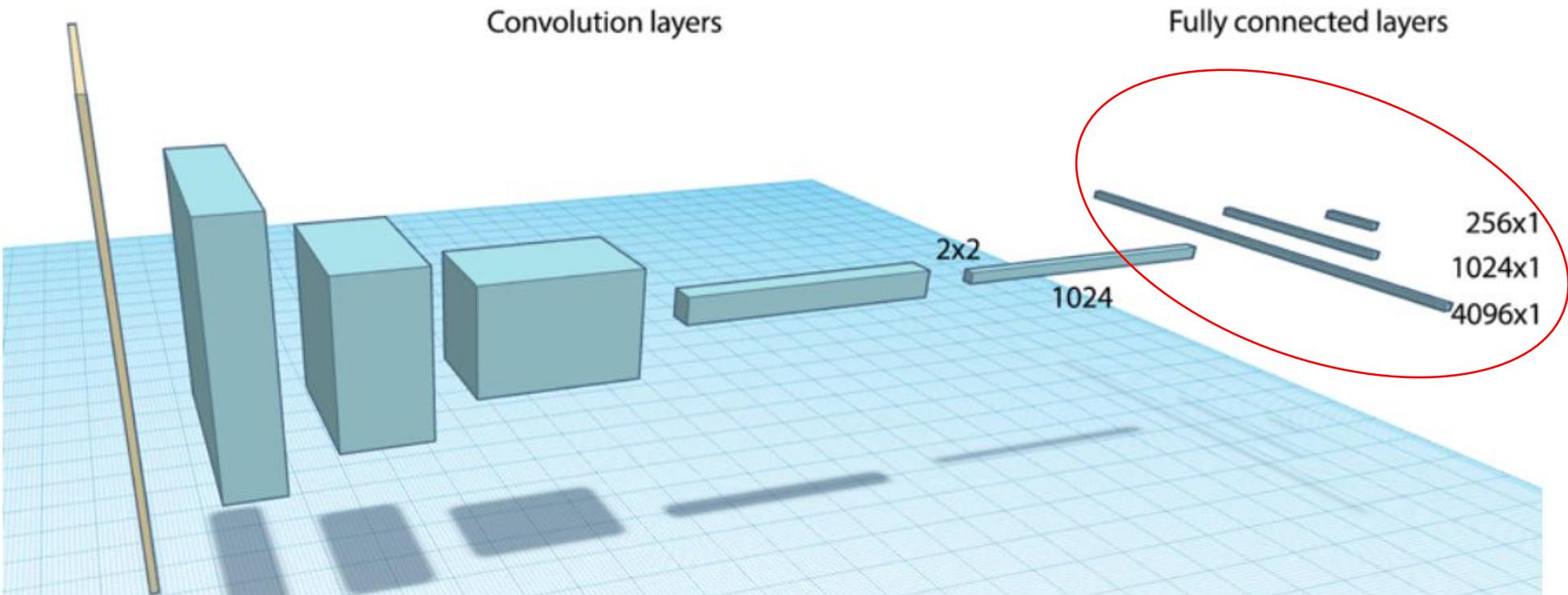
A typical architecture: Convolutional Block

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



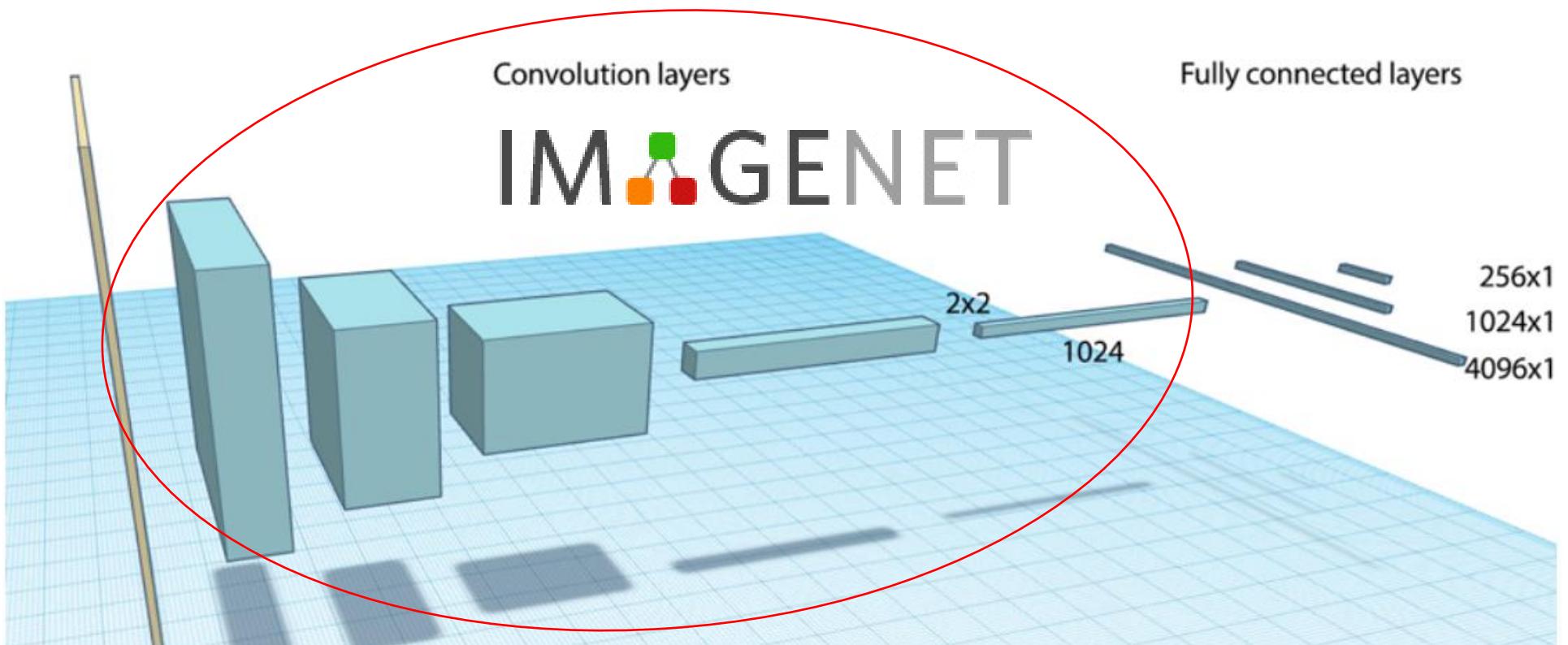
A typical architecture: FC layers

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



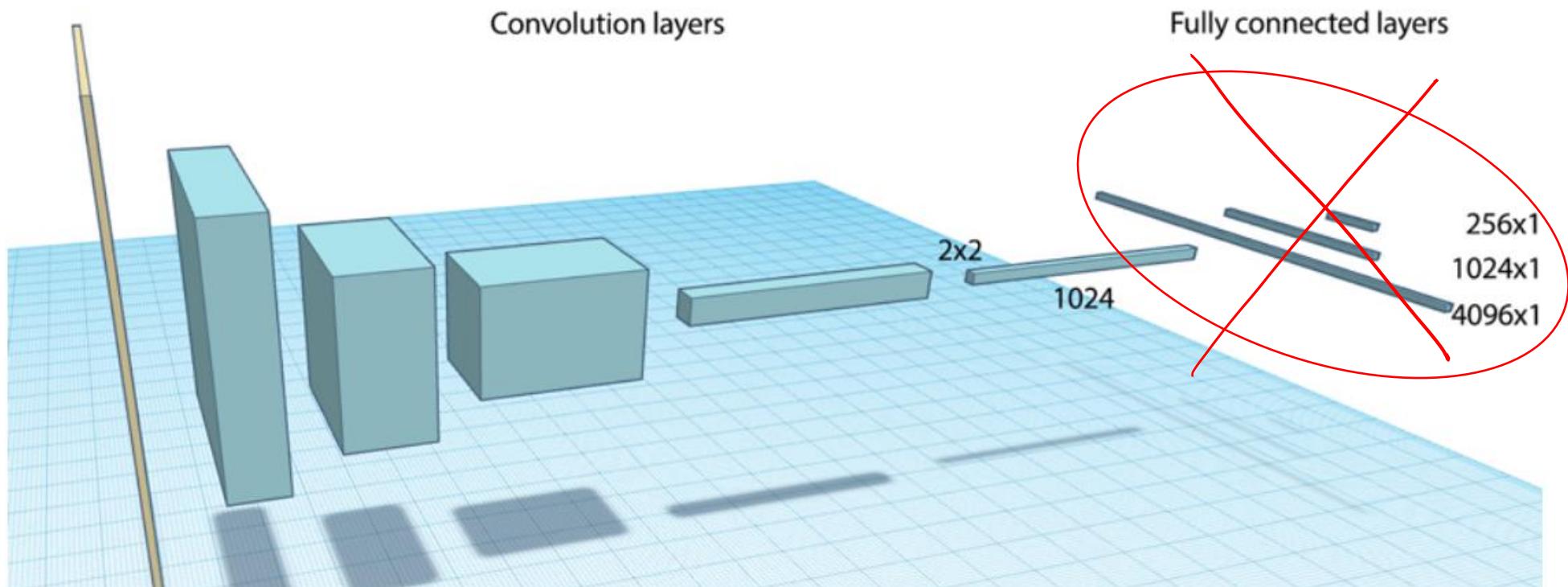
Transfer Learning

- Take a successful pre-trained model such as VGG



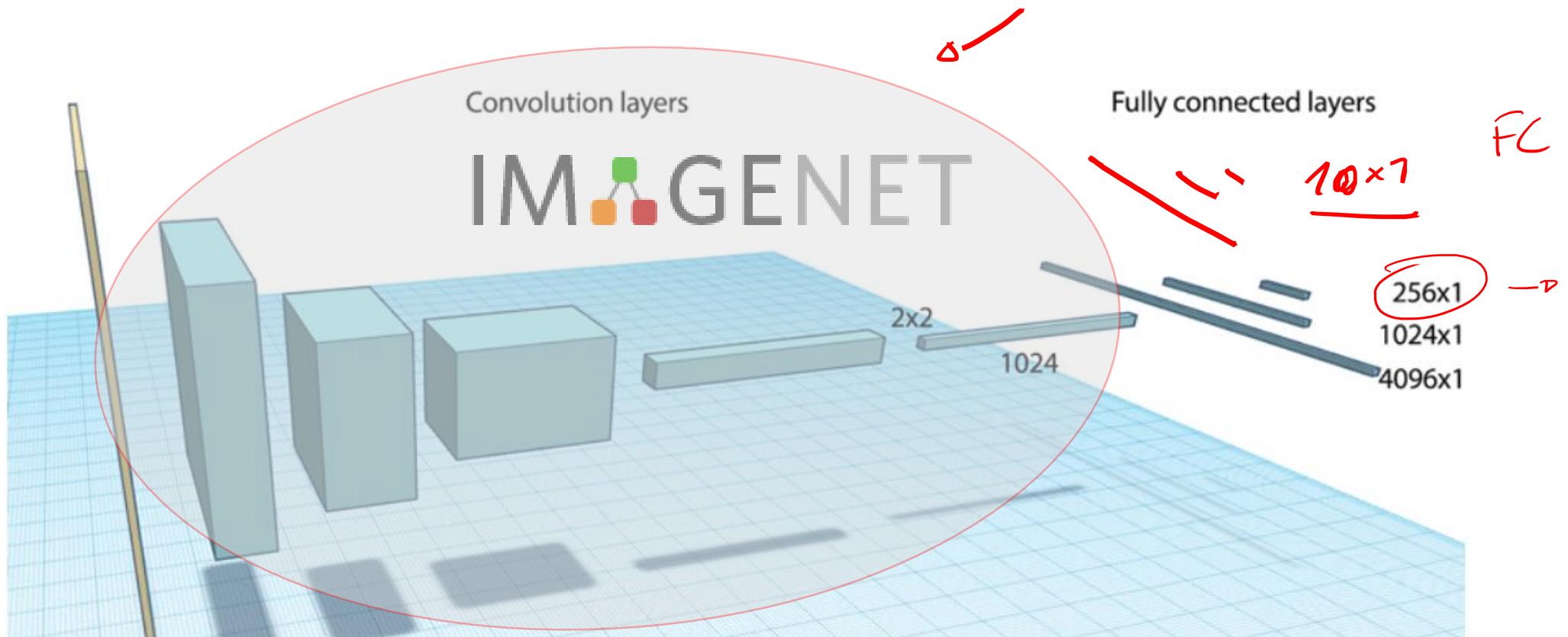
Transfer Learning

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



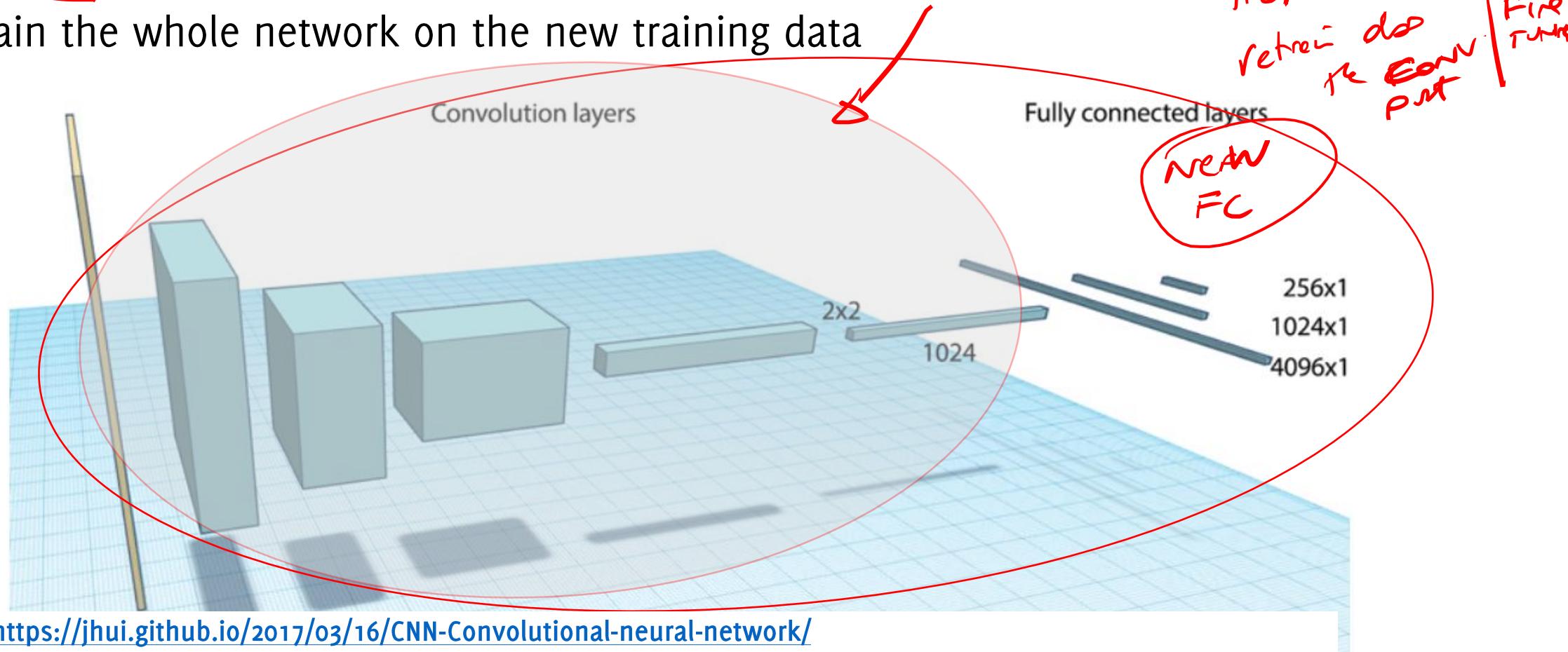
Transfer Learning

- 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



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
- Train the whole network on the new training data



In keras...

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

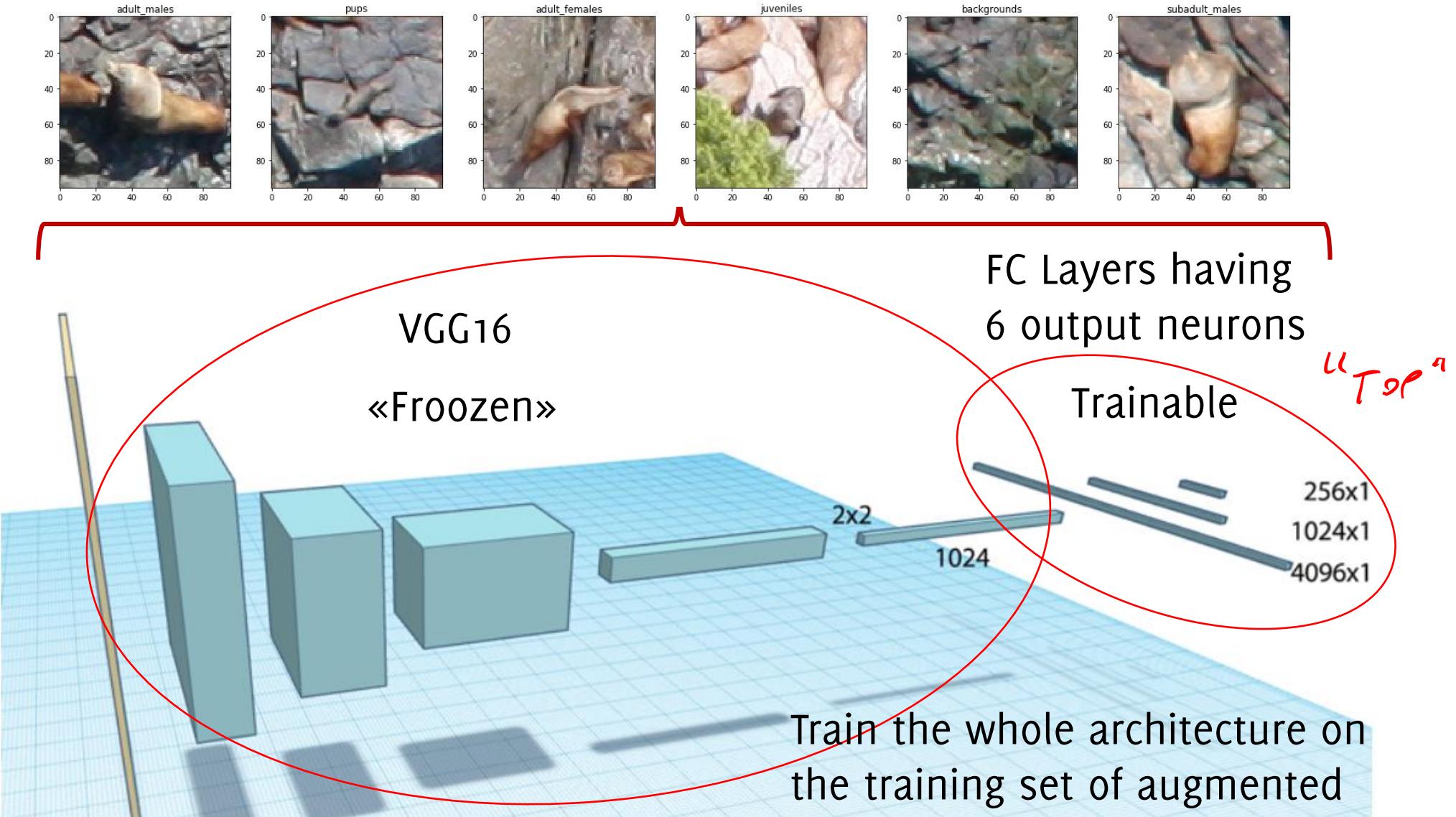
- **include_top = True**: provides the entire network, including the fully convolutional layers. This network can be used to solve the classification problem it was trained for
- **include_top = False**: contains only the convolutional layers of the network, and it is specifically meant for transfer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



Transfer Learning in Keras...

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

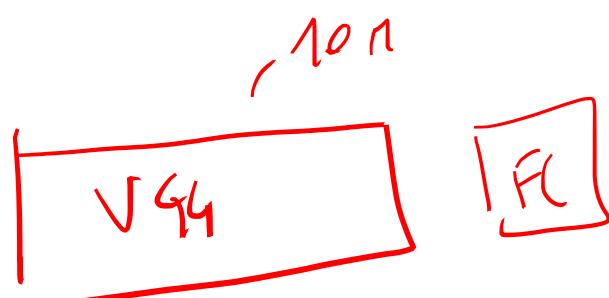
Then, before training it is necessary to loop through the network layers (they are in `model.layers`) and then modify the trainable property

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

Transfer Learning

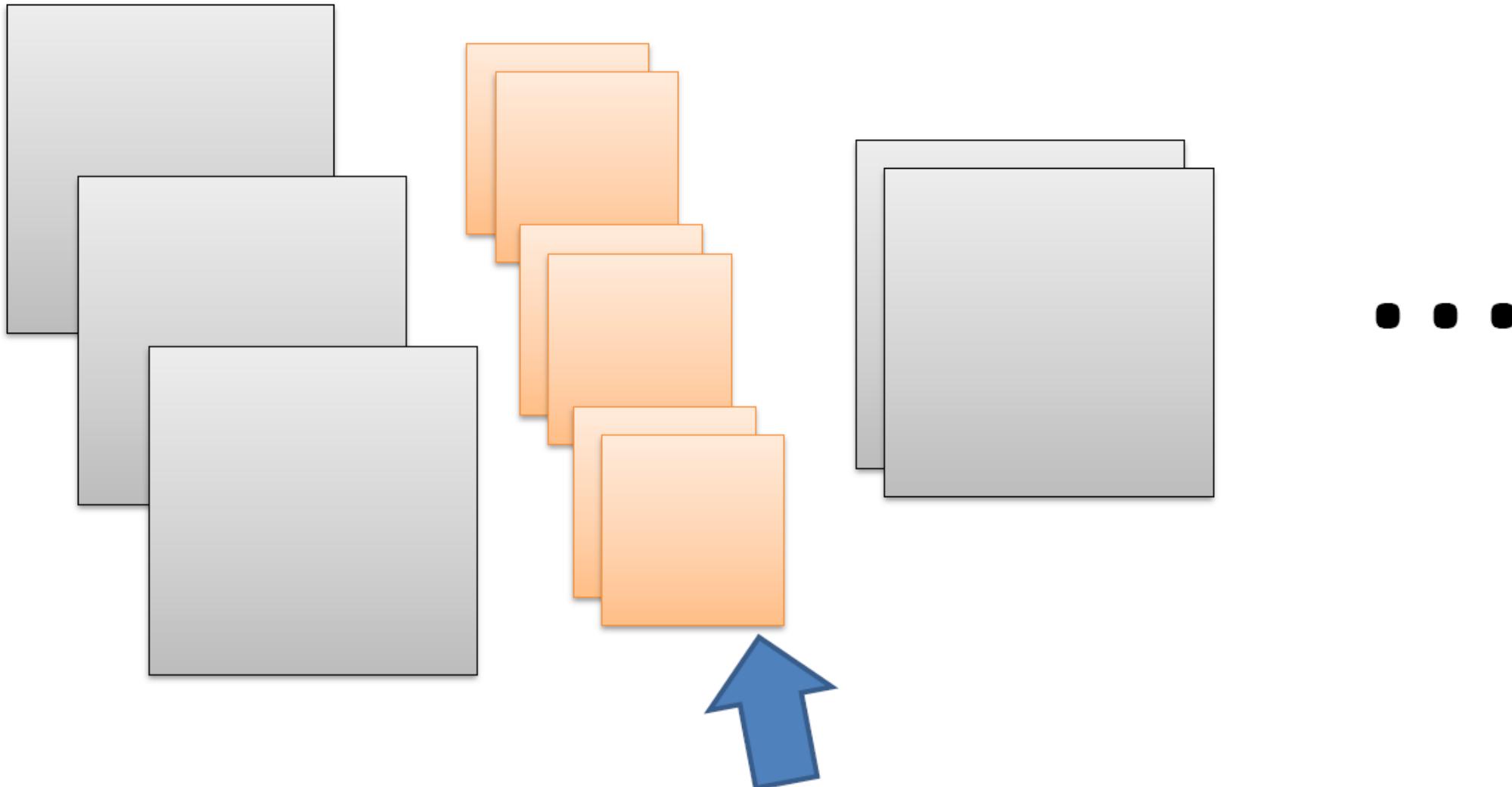
Different Options:

- 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



CNN Visualization

Visualizing CNN Filters



We want to see this

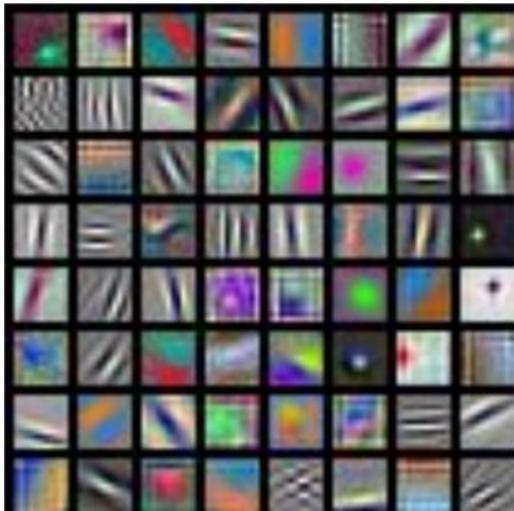
11x11x3 filters (visualized in RGB) extracted from
the first convolutional layer



$11 \times 11 \times 3$ filters (visualized in RGB) extracted from the first convolutional layer

Recall the relation between convolution and template matching:
The first layer seems to match low-level features such as edges
and simple patterns that
are discriminative to describe the data

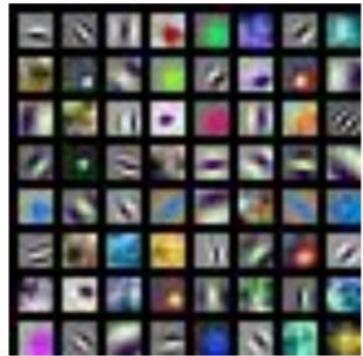
First layer's filters are often like these



AlexNet:
 $64 \times 3 \times 11 \times 11$



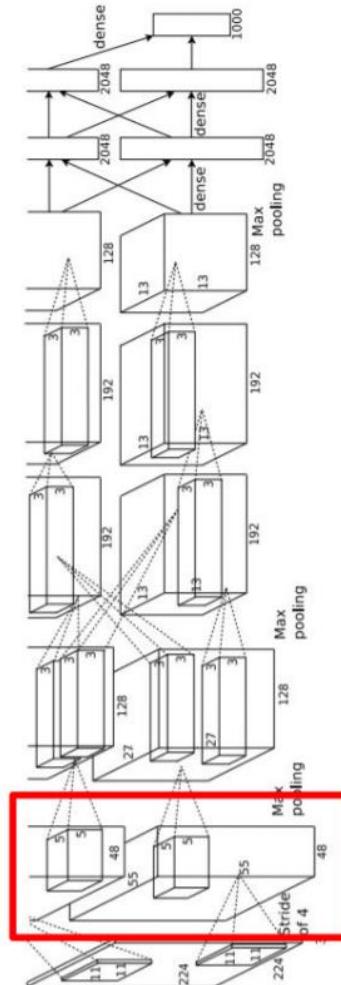
ResNet-18:
 $64 \times 3 \times 7 \times 7$



ResNet-101:
 $64 \times 3 \times 7 \times 7$



DenseNet-121:
 $64 \times 3 \times 7 \times 7$



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

Difficult to interpret deeper layers

Weights:


1°

Weights:

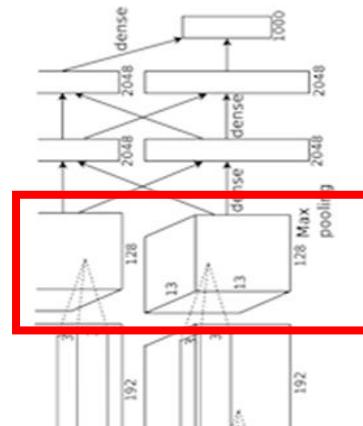
(weights for layer 1)
(weights for layer 1)

Weights:

(weights for layer 2)
(weights for layer 2)

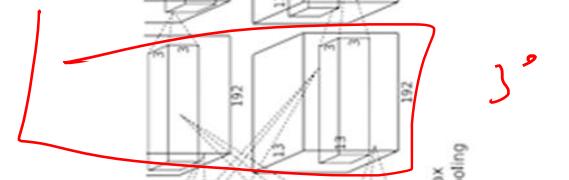
layer 1 weights

$16 \times 3 \times 7 \times 7$



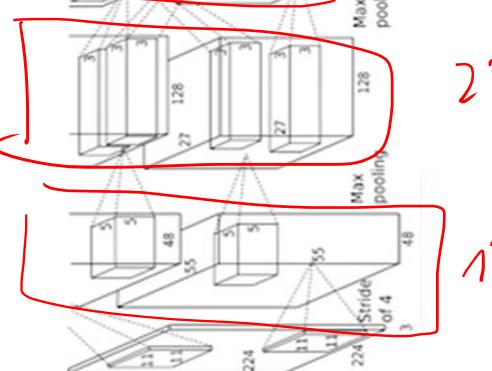
layer 2 weights

$20 \times 16 \times 7 \times 7$



layer 3 weights

$20 \times 20 \times 7 \times 7$



Difficult to interpret deeper layers

Weights:

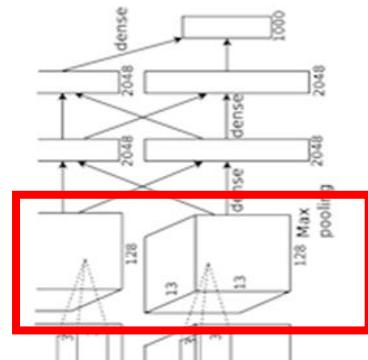


Weights:

(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)

layer 1 weights

$16 \times 3 \times 7 \times 7$



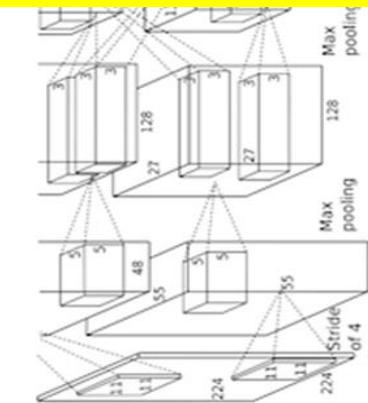
layer 2 weights

Another way to determine «what the deepest layer see» is required

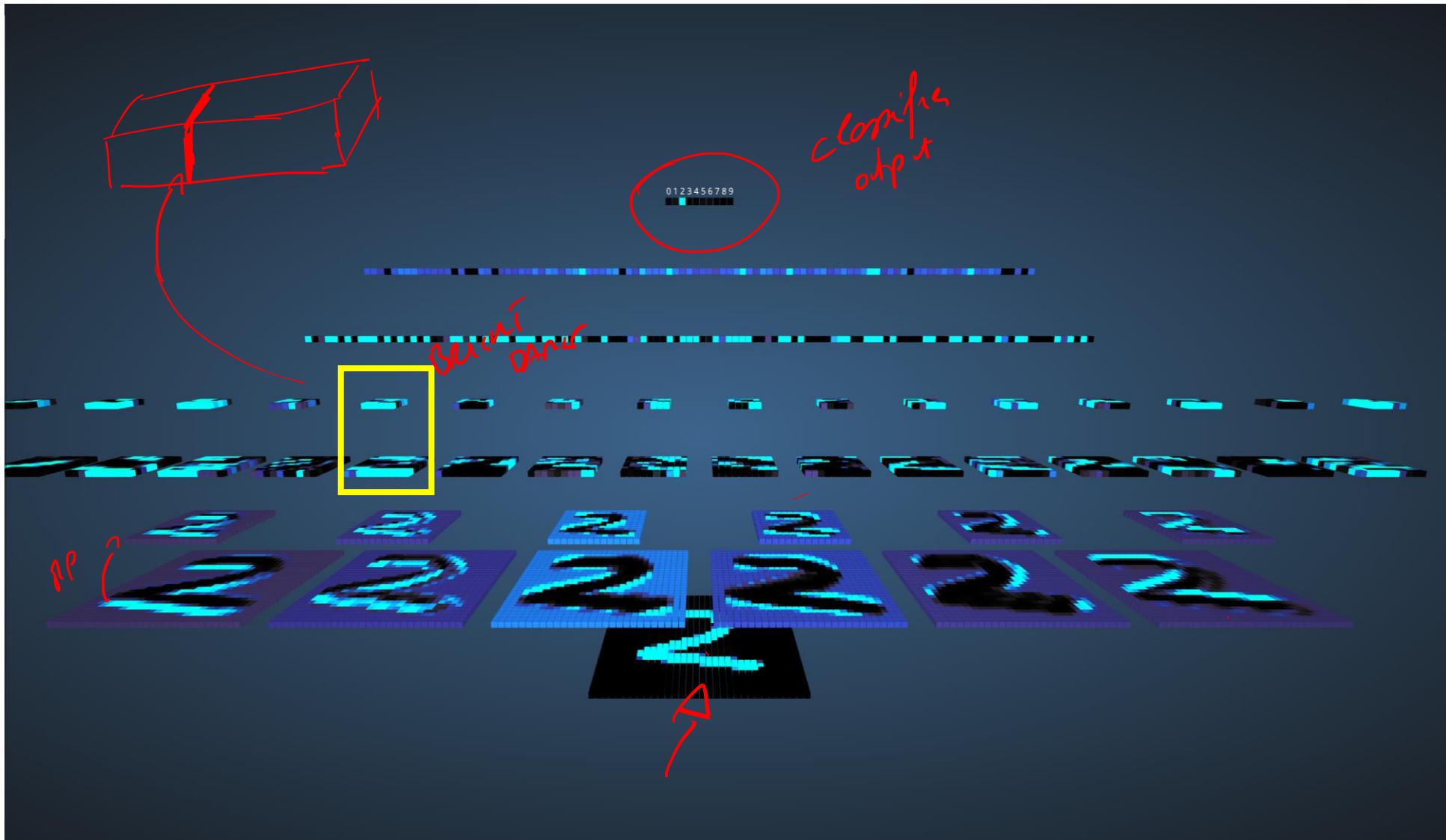
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)

layer 3 weights

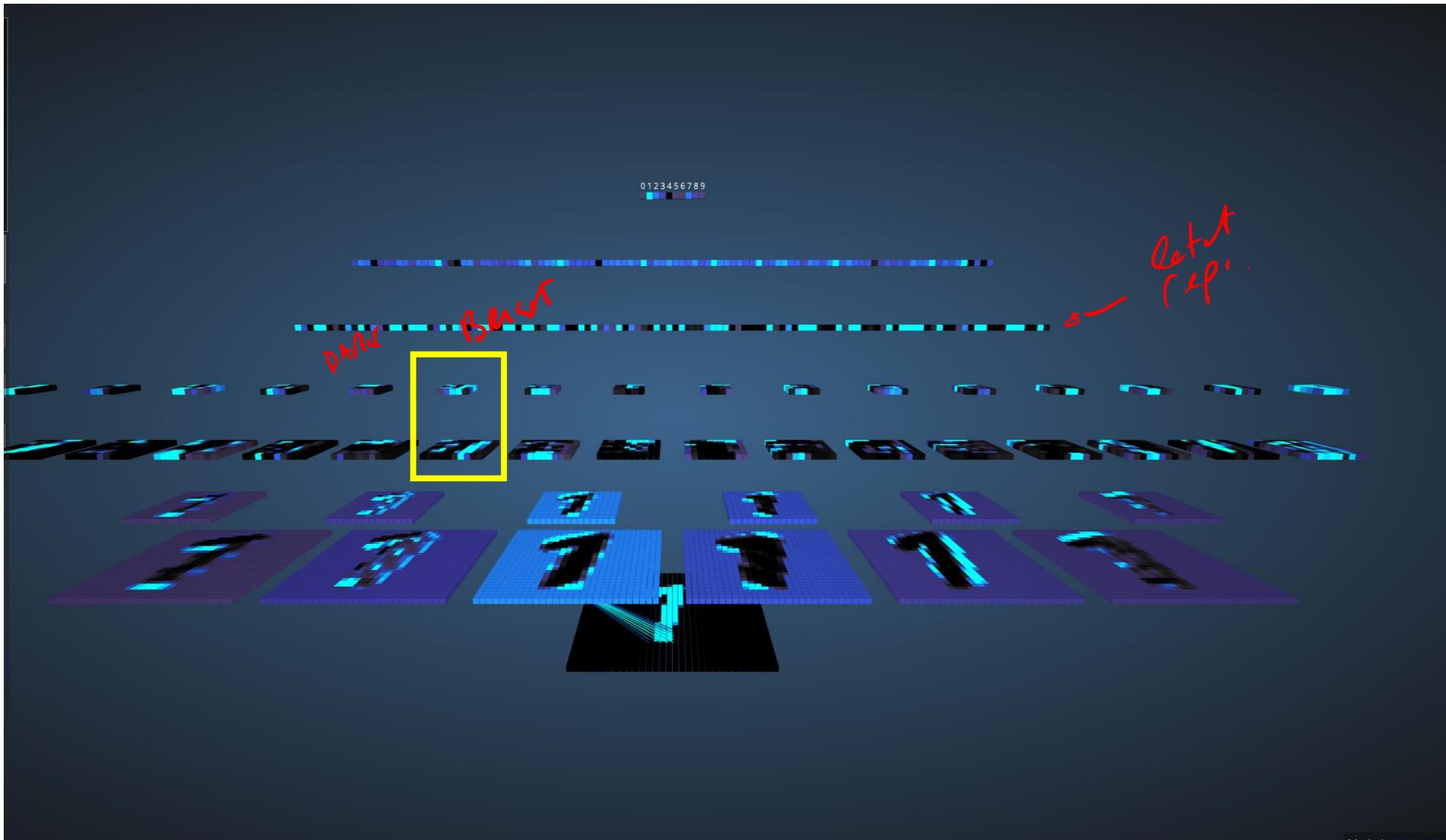
$20 \times 20 \times 7 \times 7$



What if we look at the activations?

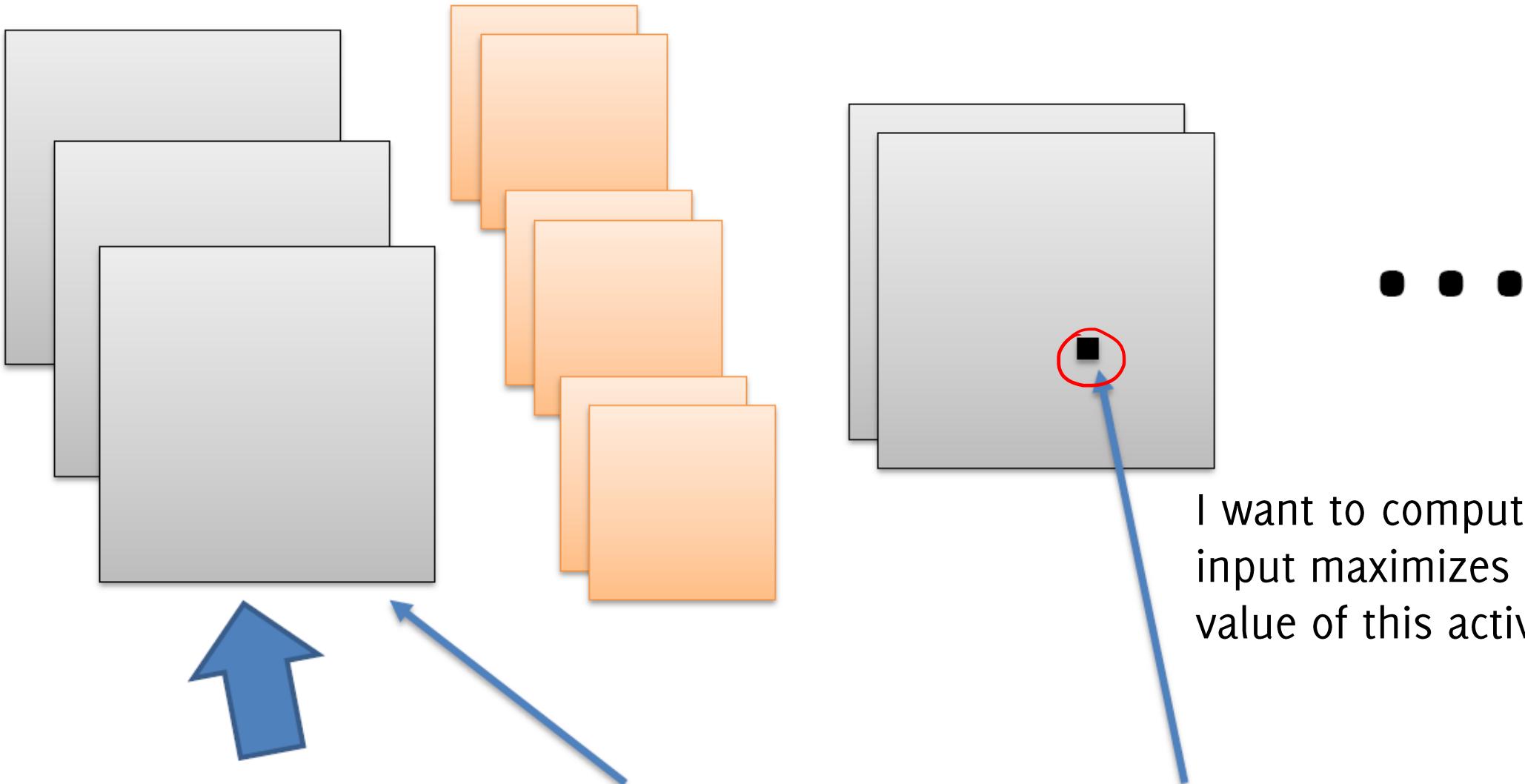


What if we look at the activations?

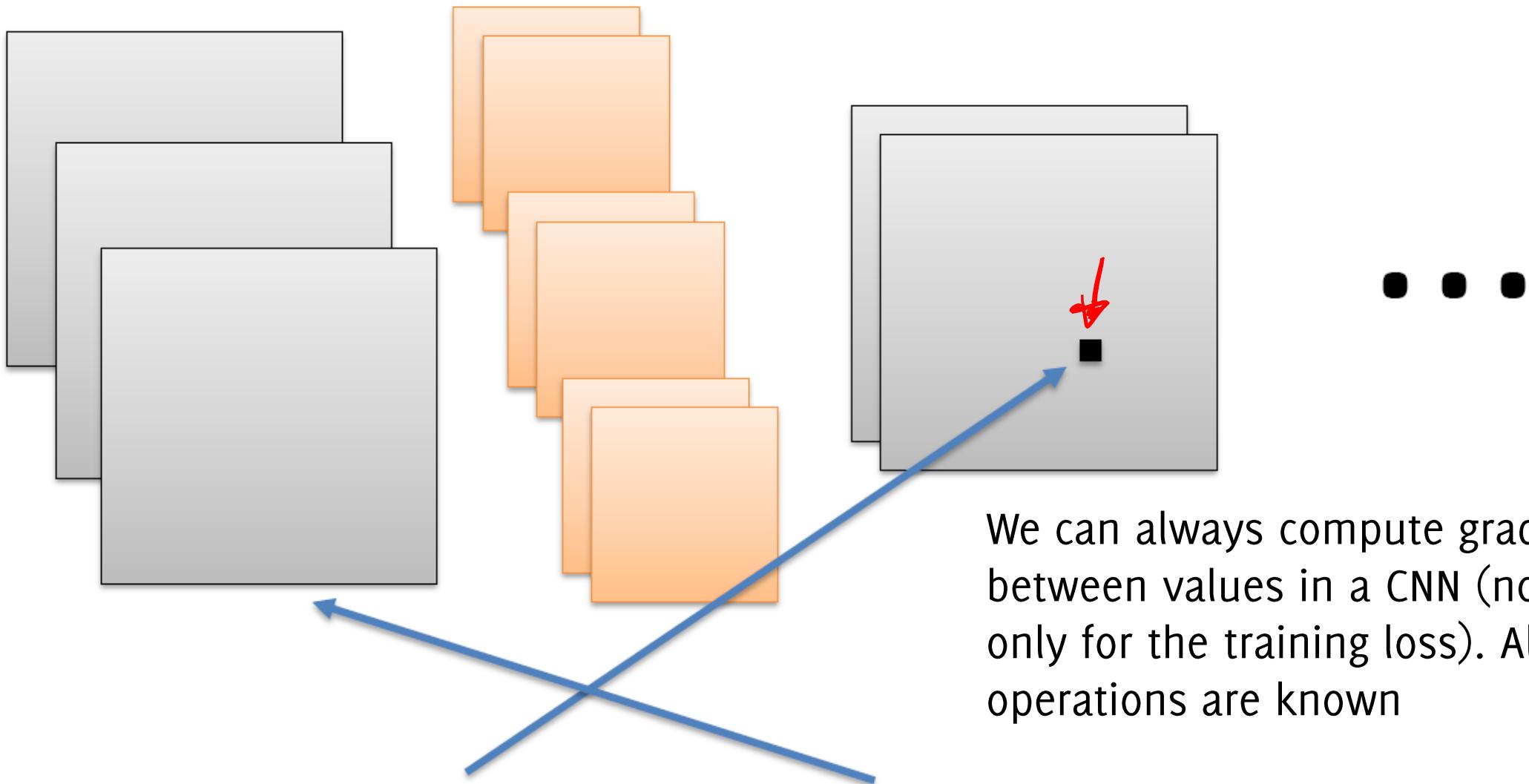


Made by Adam Harley

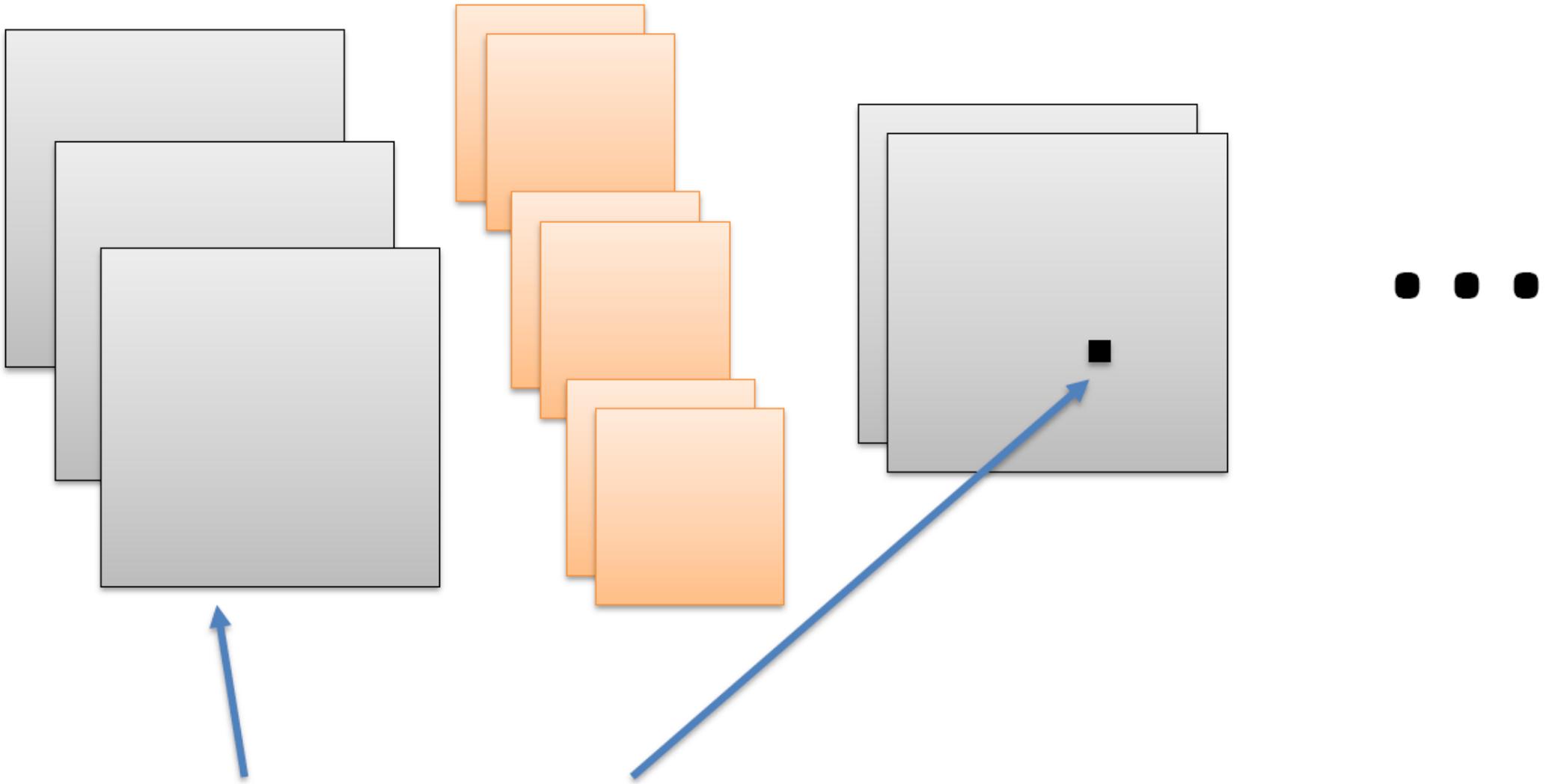
Visualizing the Input that maximally activates some neurons



Step 1

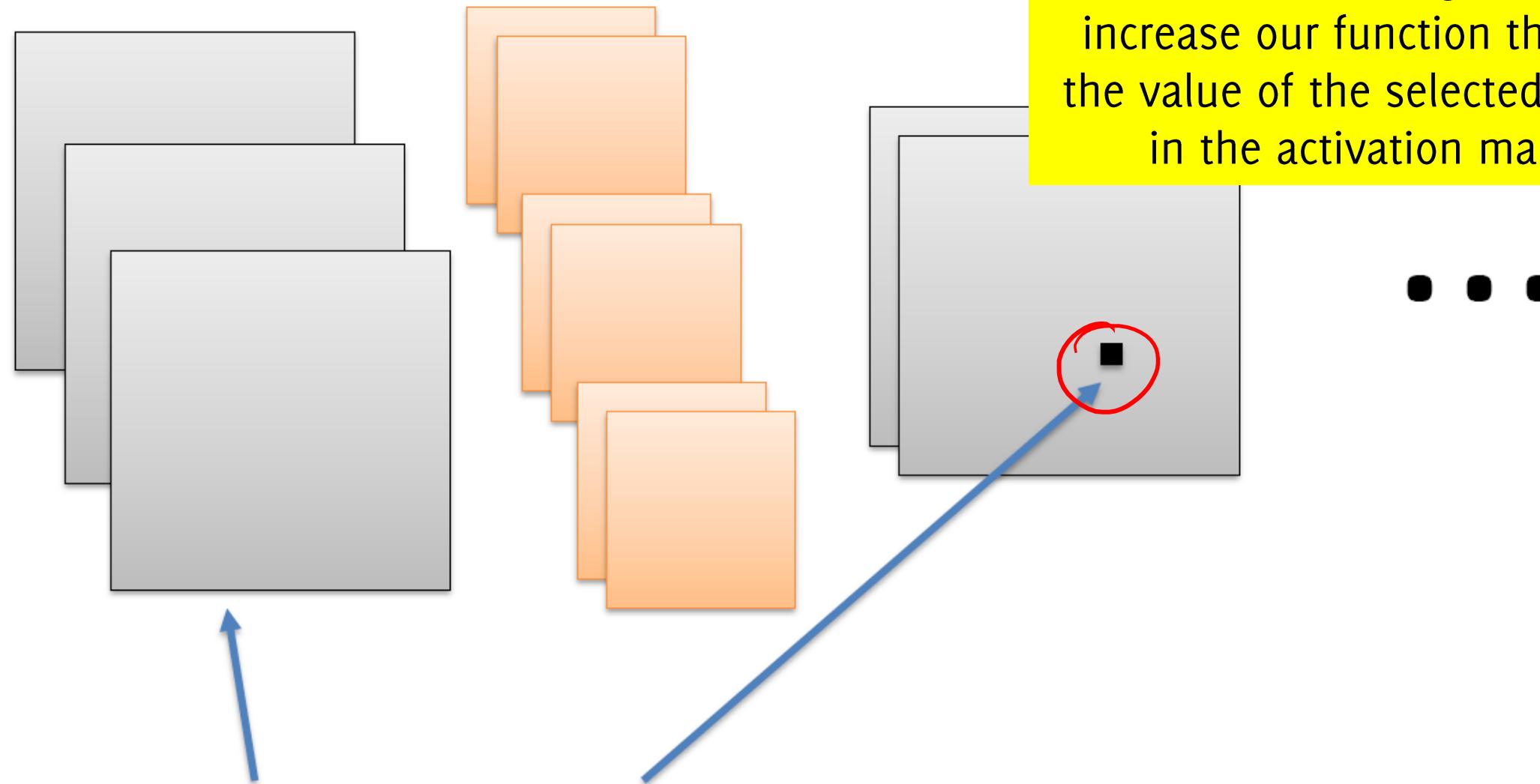


Step 2



Nudge the input accordingly: our guy will increase its activation

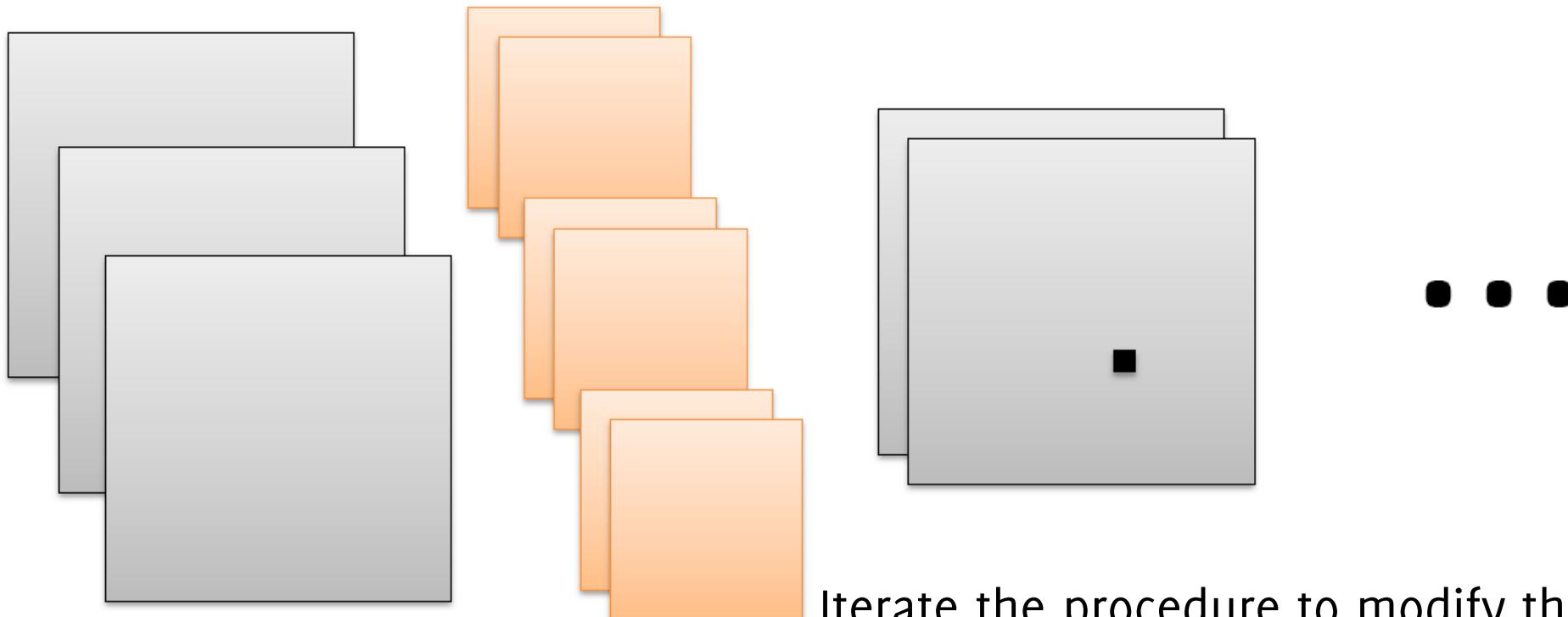
Step 2



Nudge the input accordingly: our guy will increase its activation

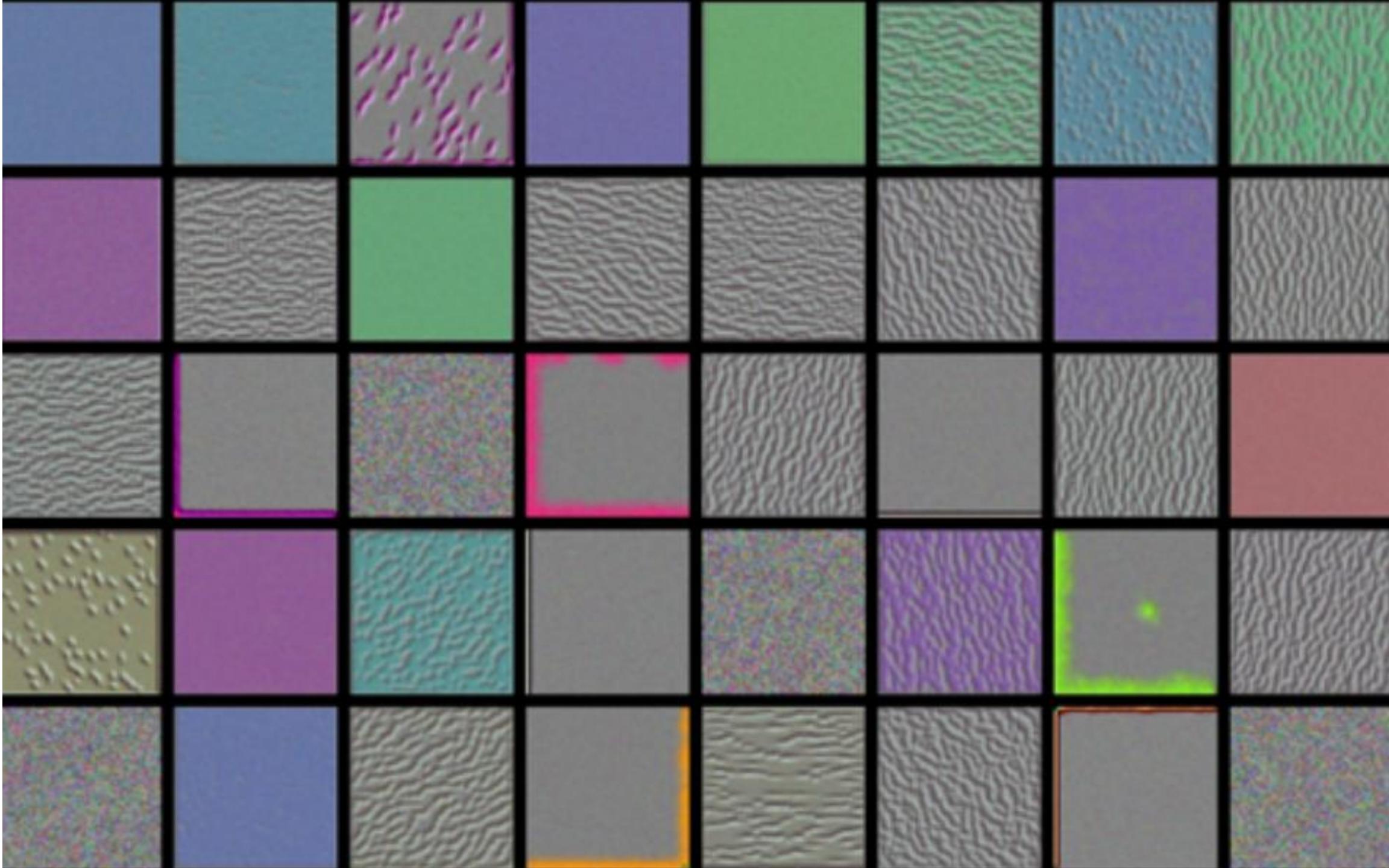
Now we perform gradient ascent, we modify the input in the direction of the gradient, to increase our function that is the value of the selected pixel in the activation map

Back to step 1 and iterate

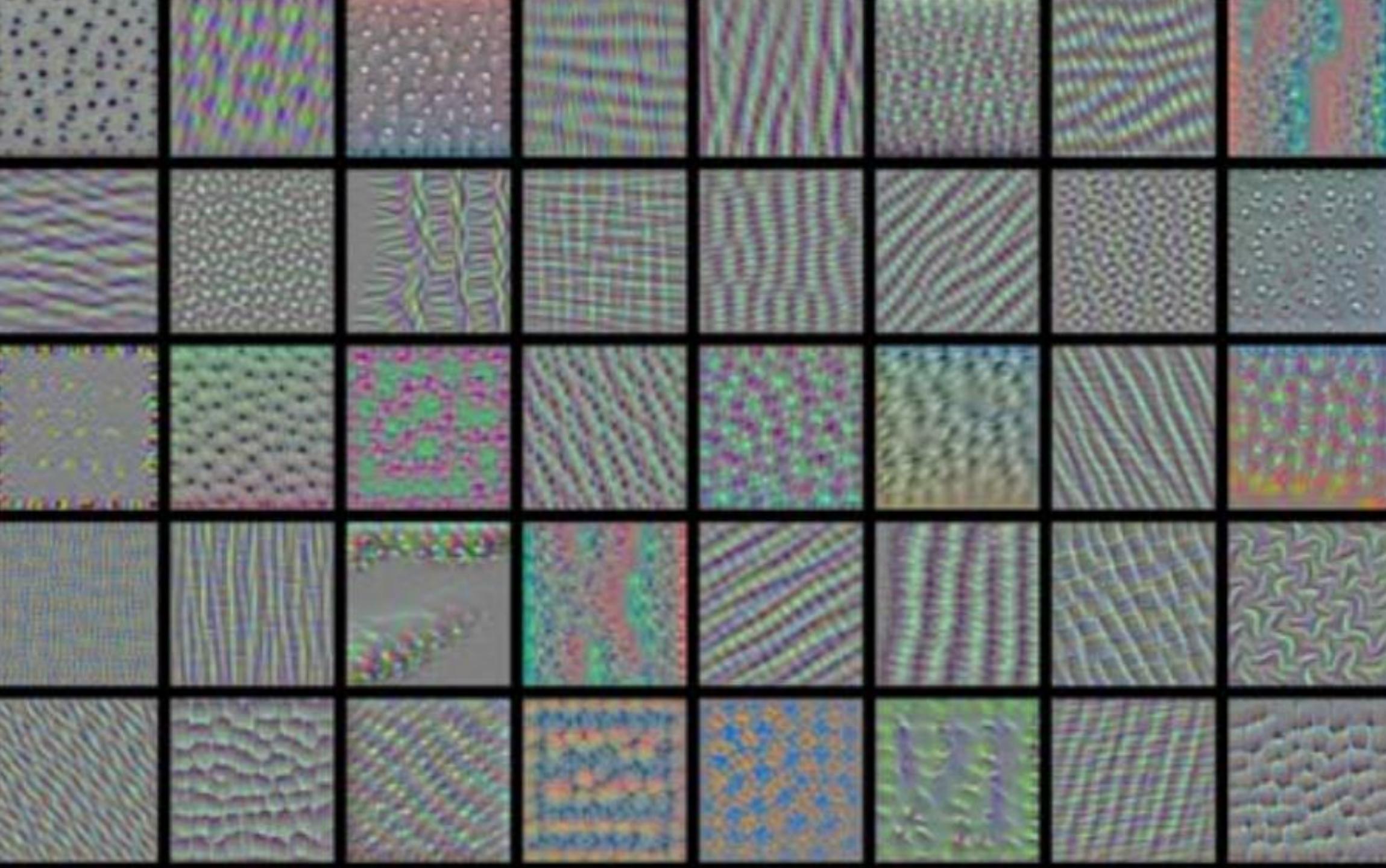


Iterate the procedure to modify the input.
Some form of regularization can be added to the
selected pixel value to steer the input to look
more like a natural image

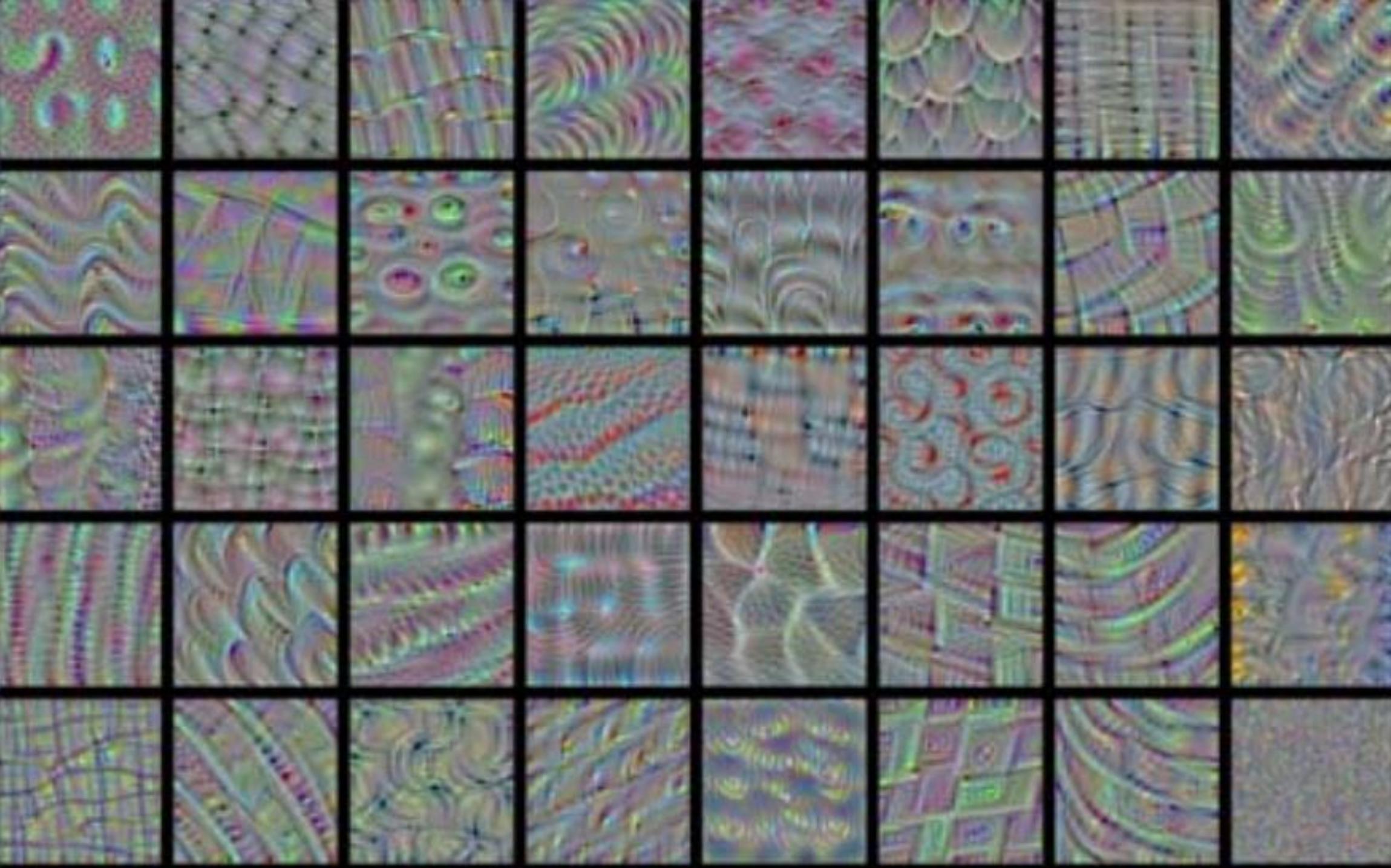
Shallow layers respond to fine, low-level patterns



Intermediate layers ...

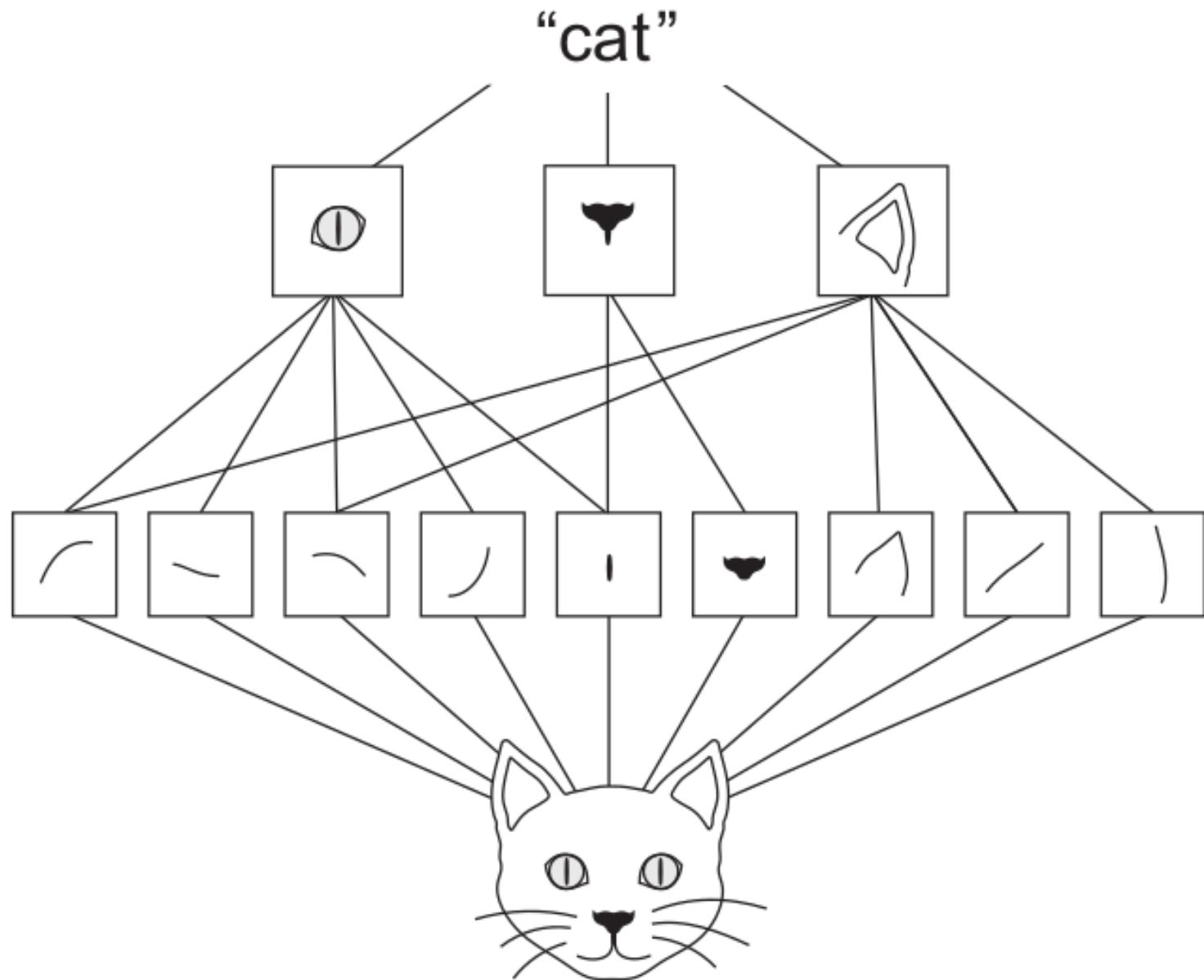


Deep layers respond to complex,
high-level patterns



Why CNNs work

Convnets learn a **hierarchy** of **translation-invariant** spatial pattern detectors

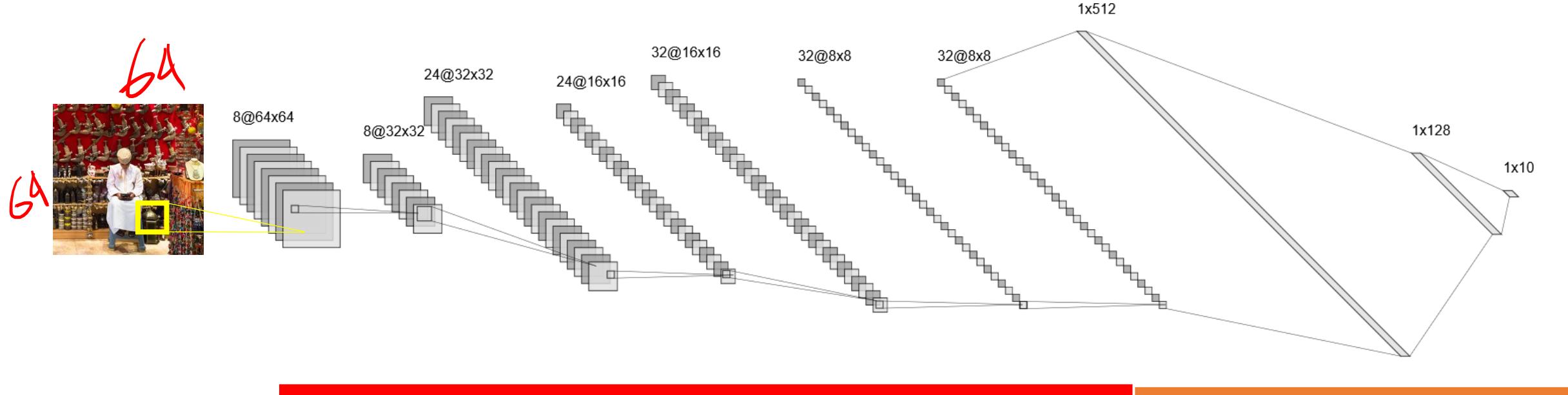


Fully Convolutional Networks

What happens when feeding the network with
an image having different size?

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



CNNs are meant to process input of a fixed size (e.g. 200 x 200).

The **convolutional and subsampling layers** operate in a sliding manner over image having arbitrary size

The **fully connected** layer constrains the input to a fixed size.

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network

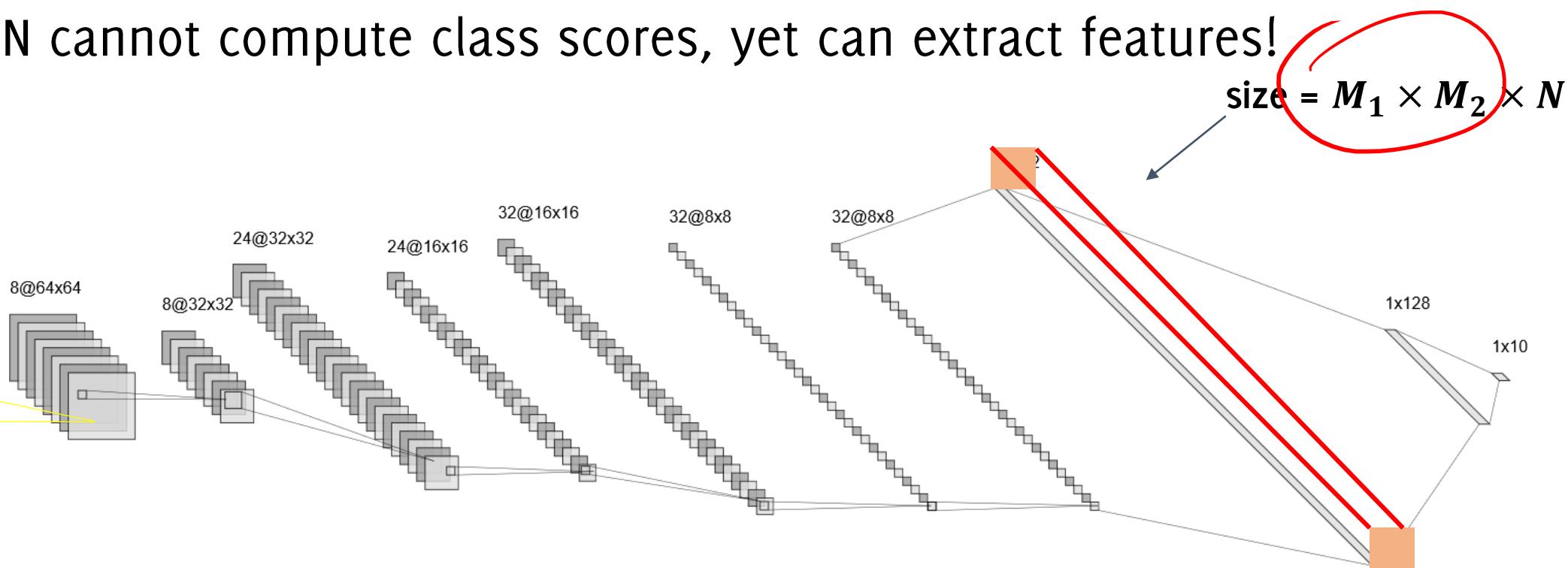


Convolutional Neural Networks (CNN)

Convolutional filters can be applied to volumes of any size, yielding larger volumes in the network until the FC layer.

The FC network however does require a fixed input size

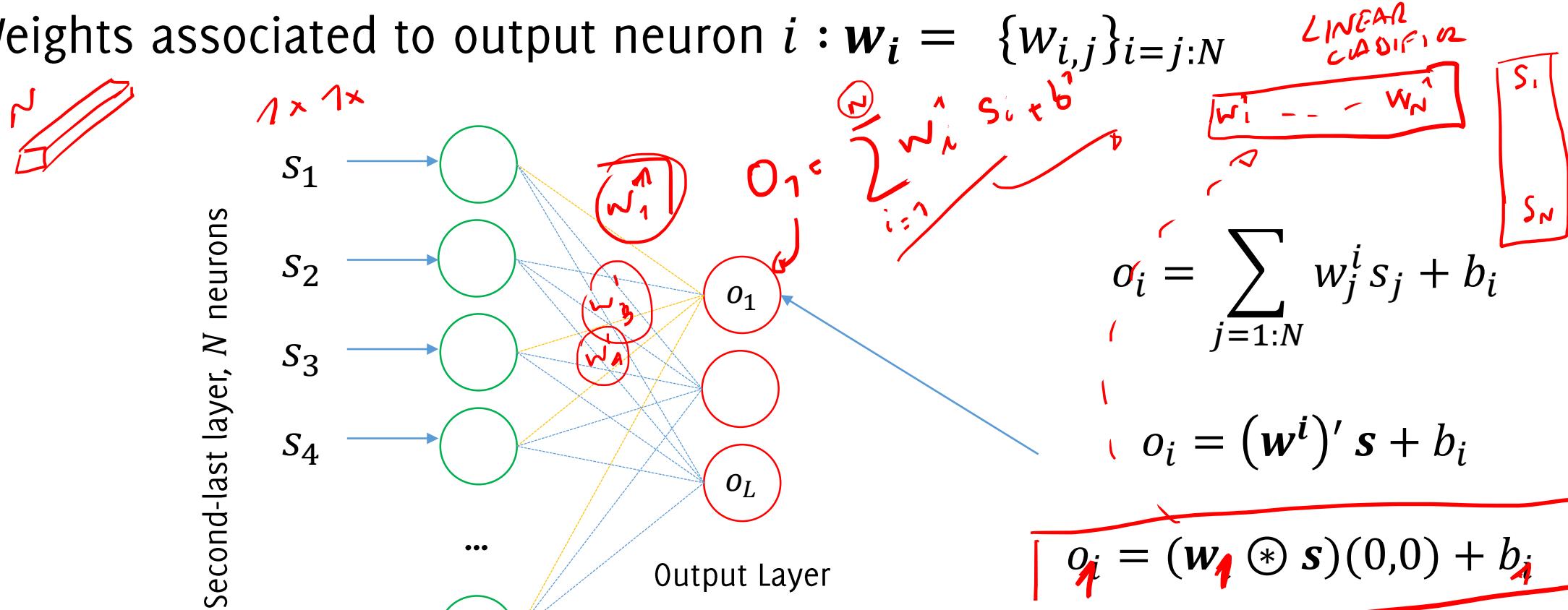
Thus, CNN cannot compute class scores, yet can extract features!



Fully Convolutional Neural Networks (f-CNN)

However, since the FC is linear, it can be represented as convolution!

Weights associated to output neuron i : $\mathbf{w}_i = \{w_{i,j}\}_{j=1:N}$

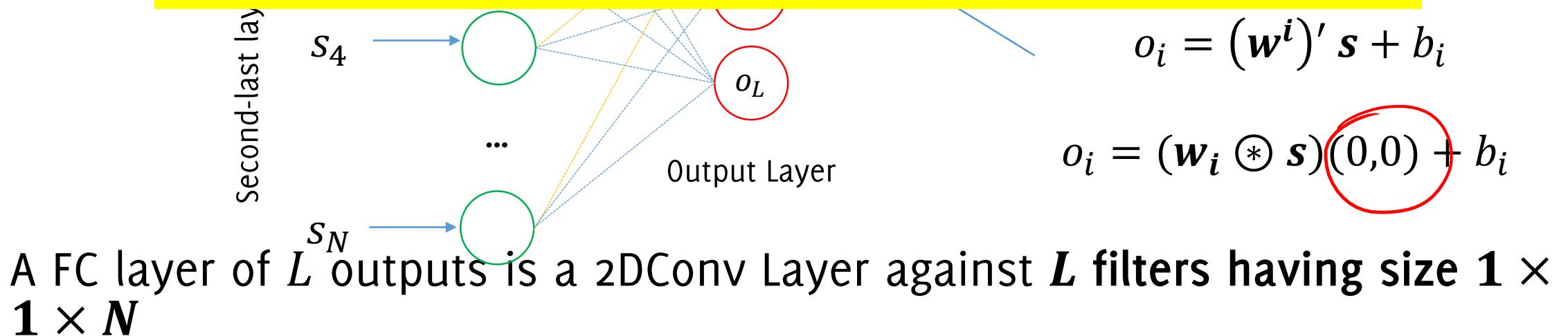
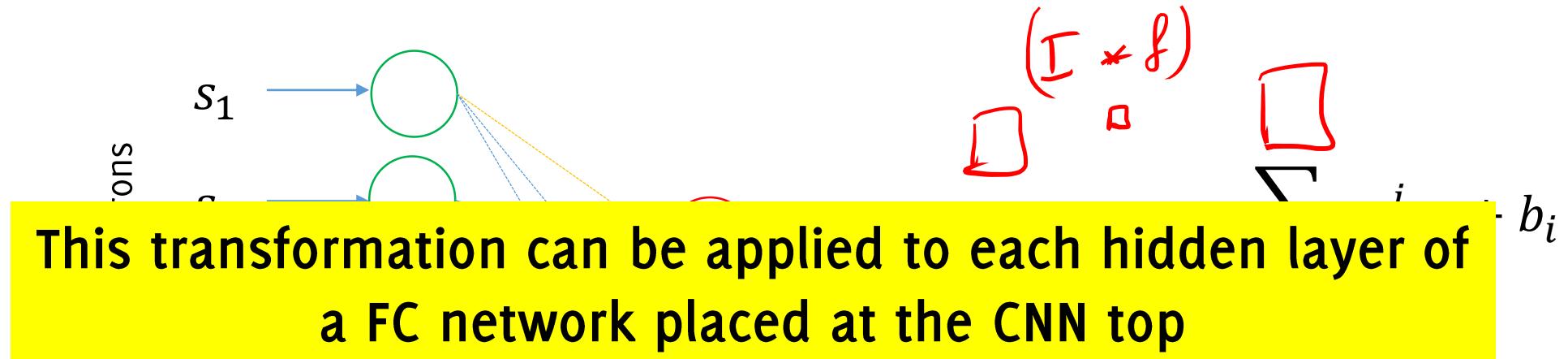


A FC layer of L outputs is a 2DConv Layer against L filters having size $1 \times 1 \times N$

Fully Convolutional Neural Networks (f-CNN)

However, since the FC is linear, it can be represented as convolution!

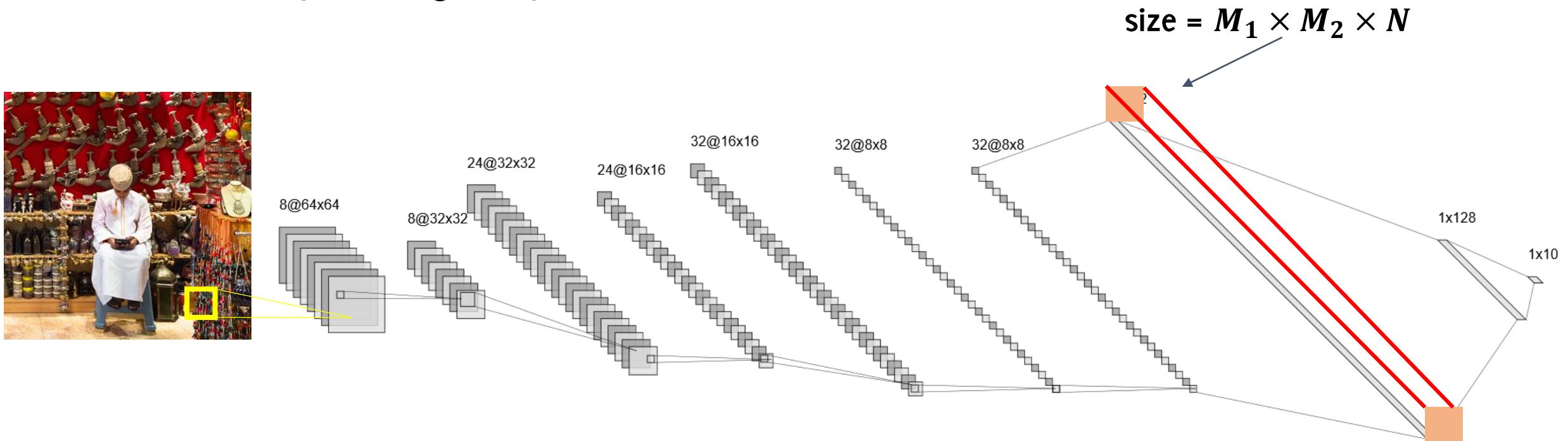
Weights associated to output neuron i : $\mathbf{w}_i = \{w_{i,j}\}_{j=1:N}$



Fully Convolutional Neural Networks (f-CNN)

However, since the FC is linear, it can be represented as convolution against L filters of size $1 \times 1 \times N$

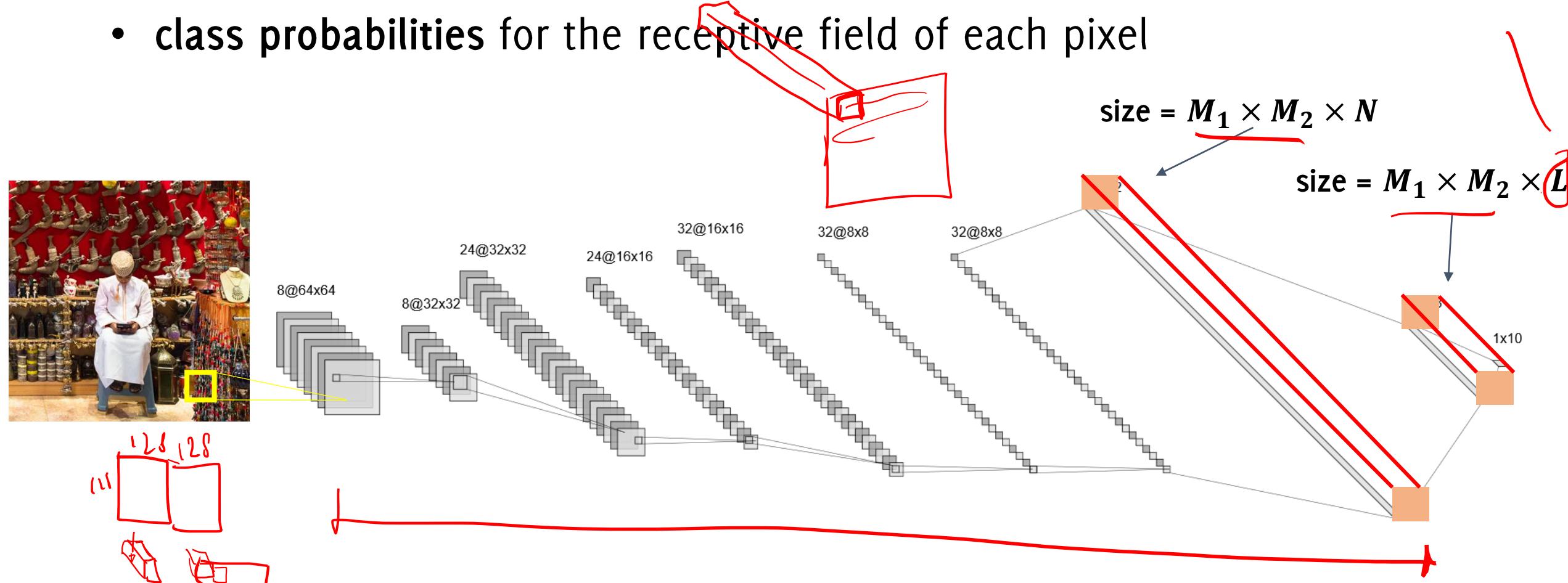
Each of these convolutional filters contains the weights of the FC for the corresponding output neuron



Fully Convolutional Neural Networks (f-CNN)

For each output class we obtain an image, having:

- Lower resolution than the input image
- class probabilities for the receptive field of each pixel



~~FCN~~ Output of a FCNN as heatmaps

The diagram illustrates the transition from a fully connected (FC) layer to a fully convolutional (CONV) layer. It starts with an input of size 12×12 entering a FC layer, which then outputs to a CONV layer of size 10×10 . The output of the CONV layer is labeled "Fully Convolutional CONV".



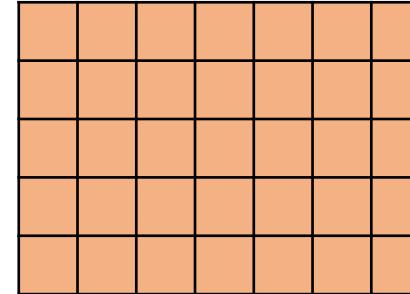
A larger image than those used for training the network

"Castle" probability

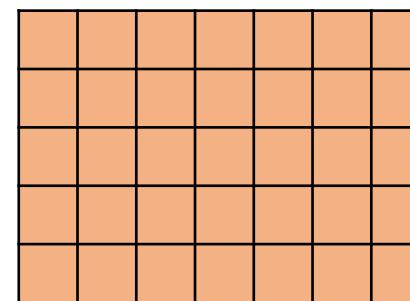
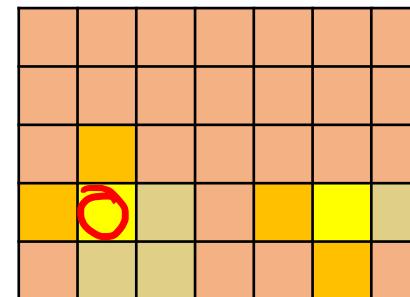
Conv TK
where FC Layer was moved to
a set of convolutional filters



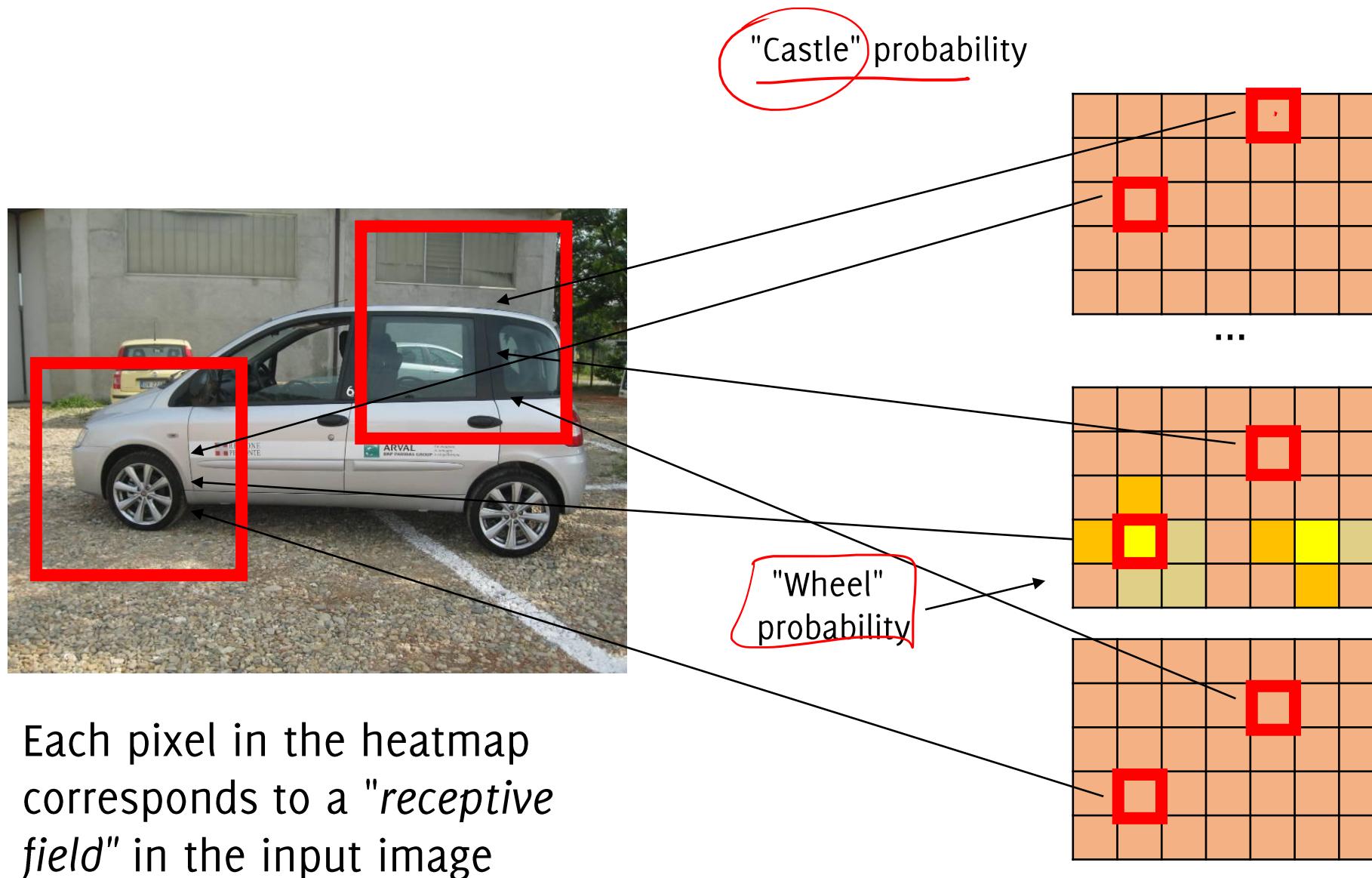
"Wheel" probability



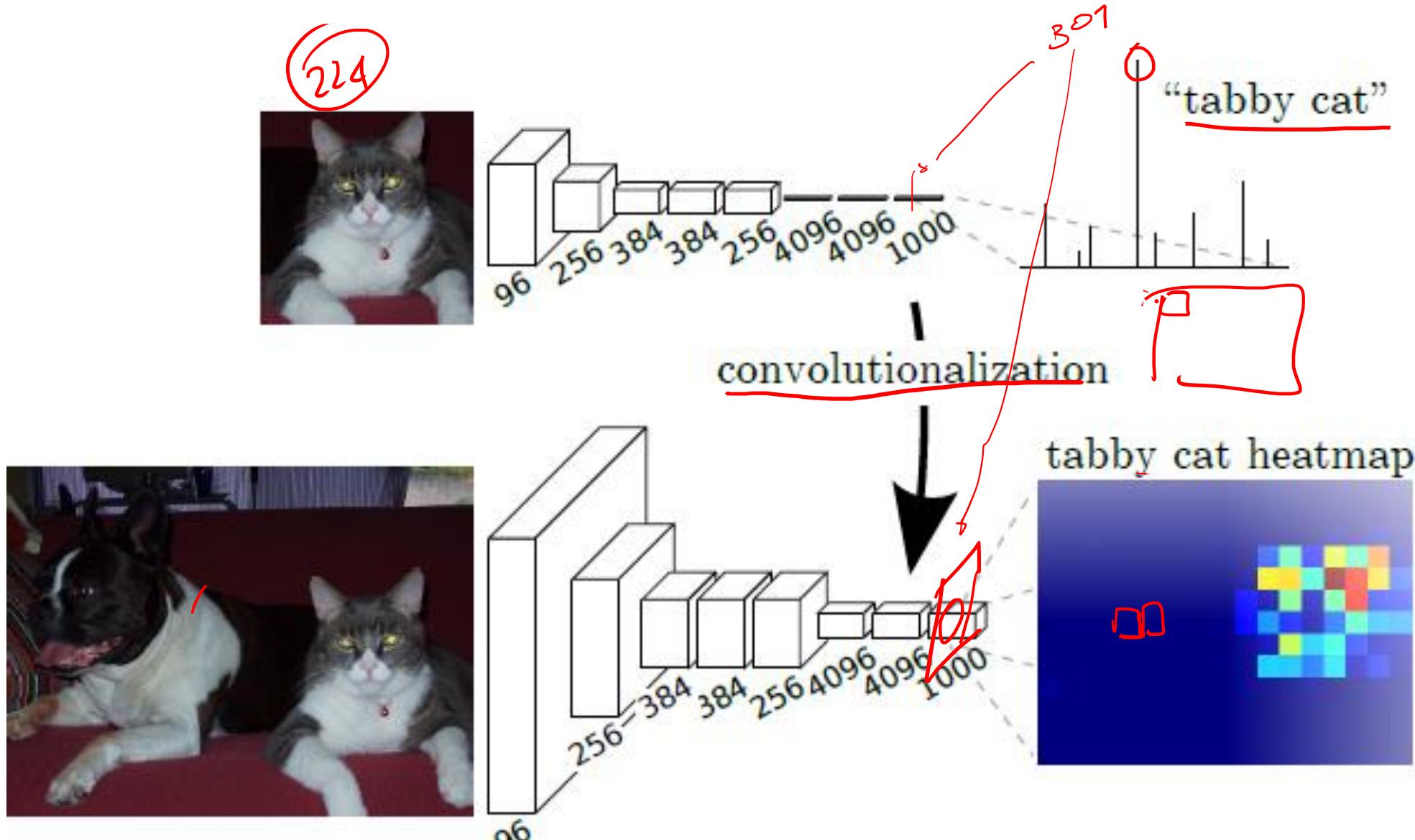
1



Output of a FCNN as heatmaps



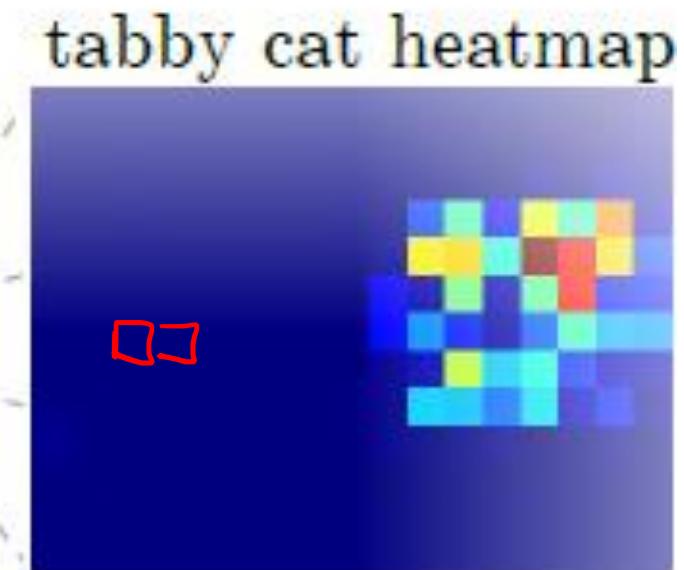
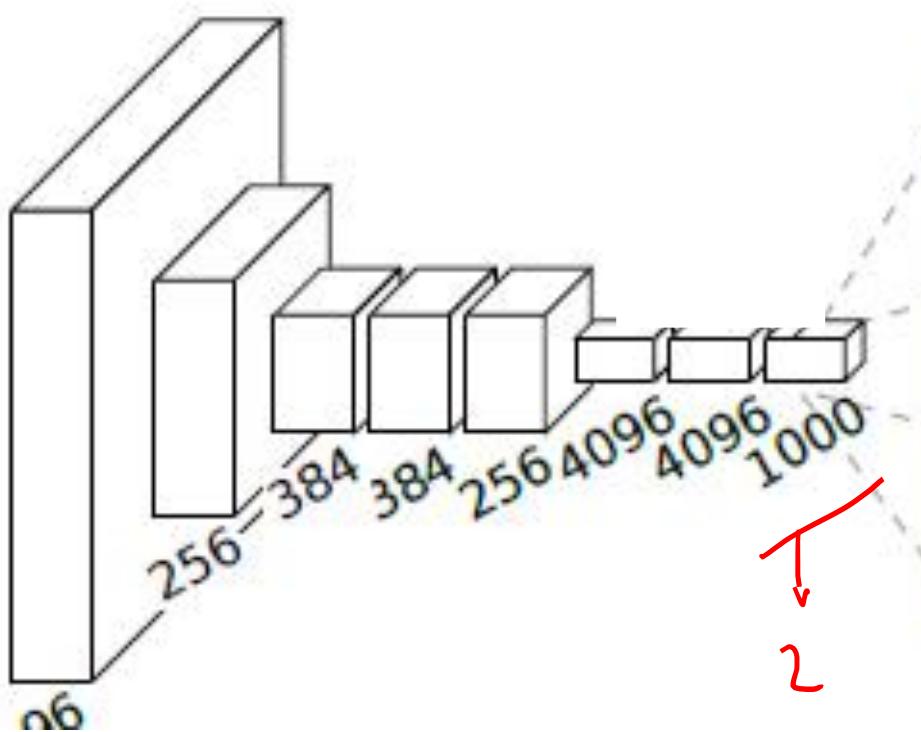
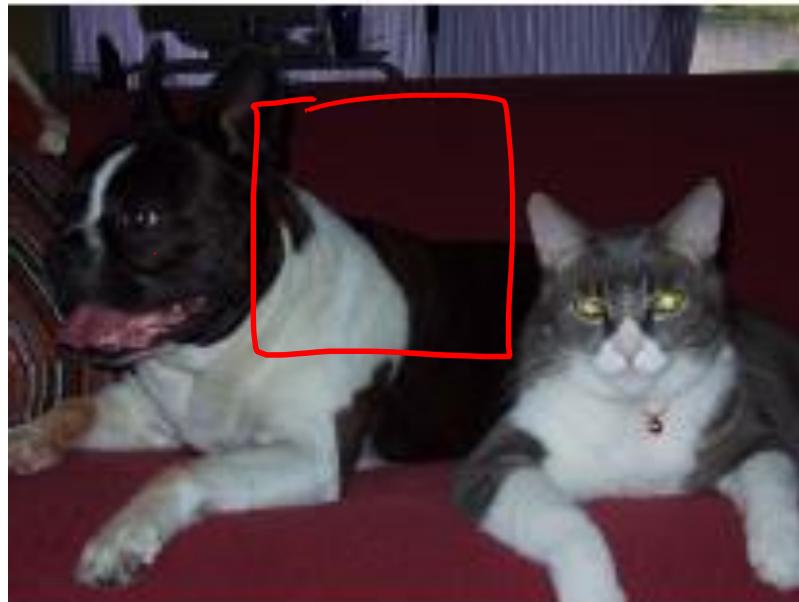
Migration to FCNN of a pretrained model



Migration to FCNN of a pretrained model

This stack of convolutions operates on the whole image as a filter.

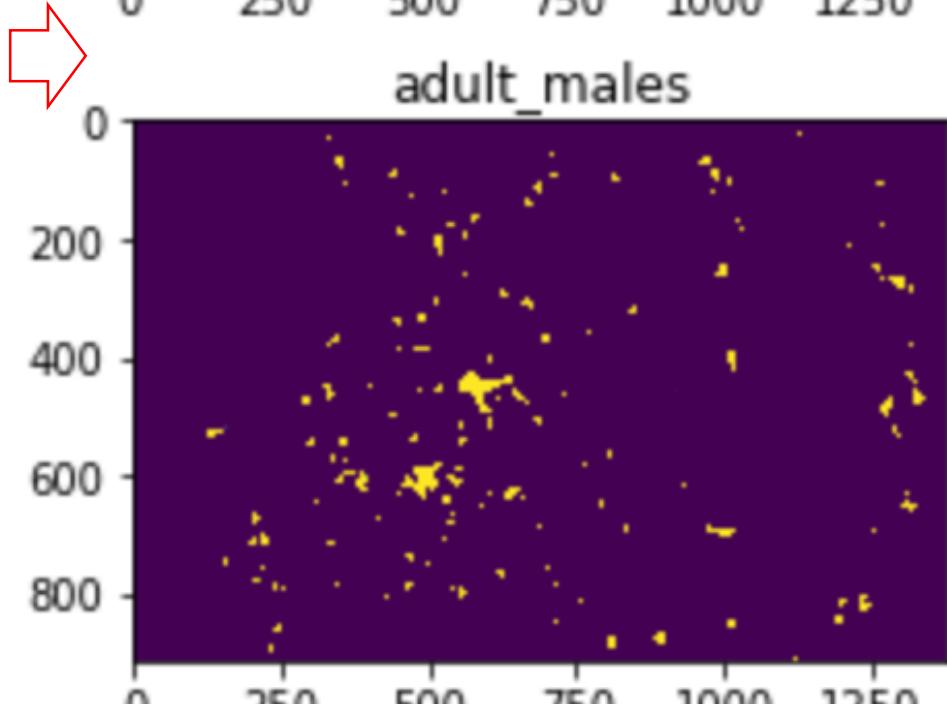
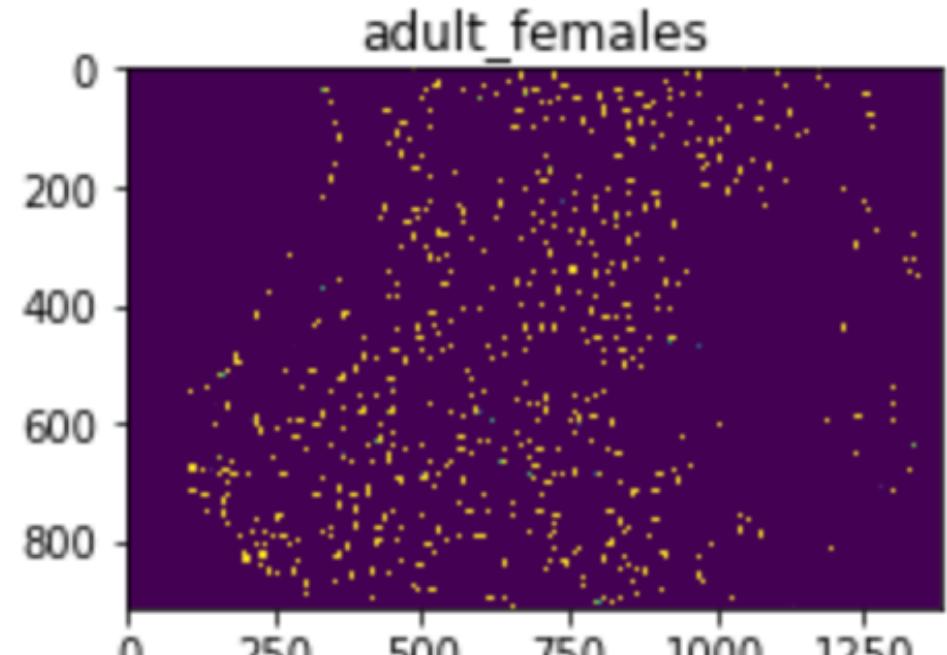
Significantly more efficient than patch extraction and classification (avoids multiple repeated computations within overlapping patches)



Sealion HeatMaps



FCNN



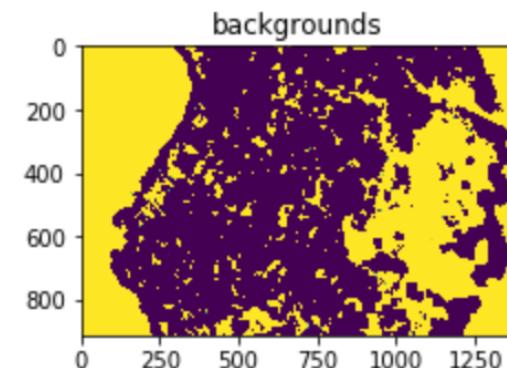
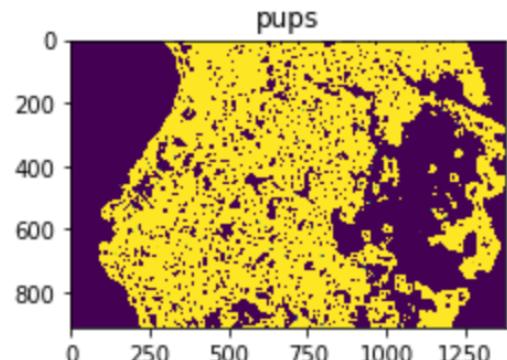
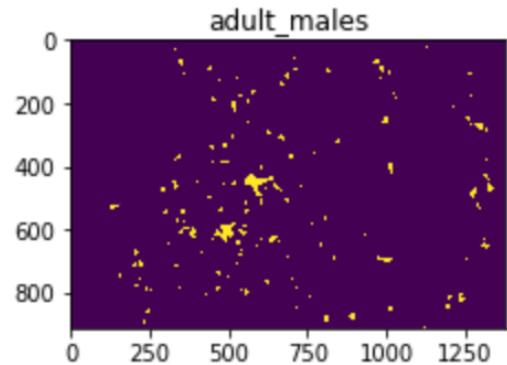
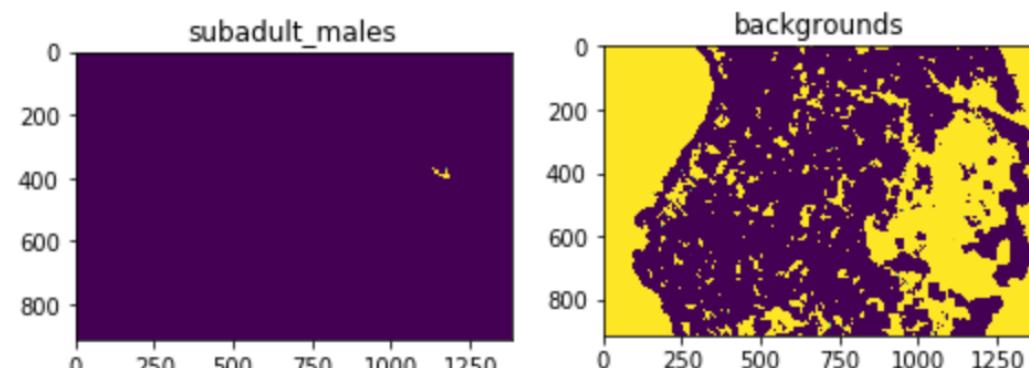
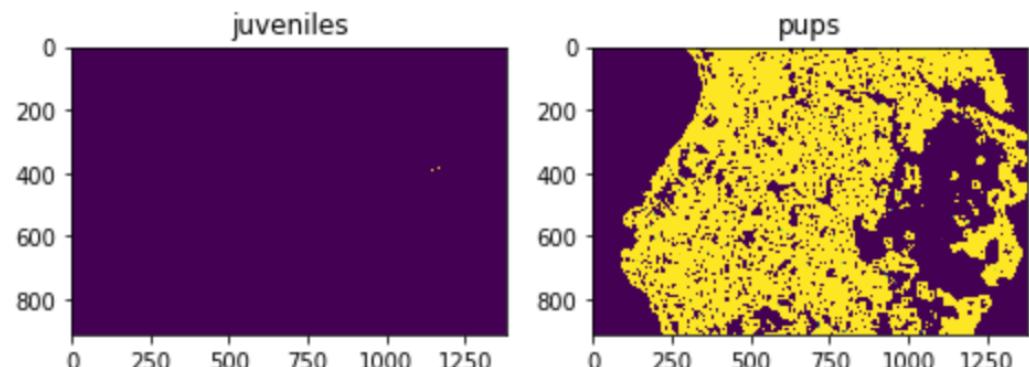
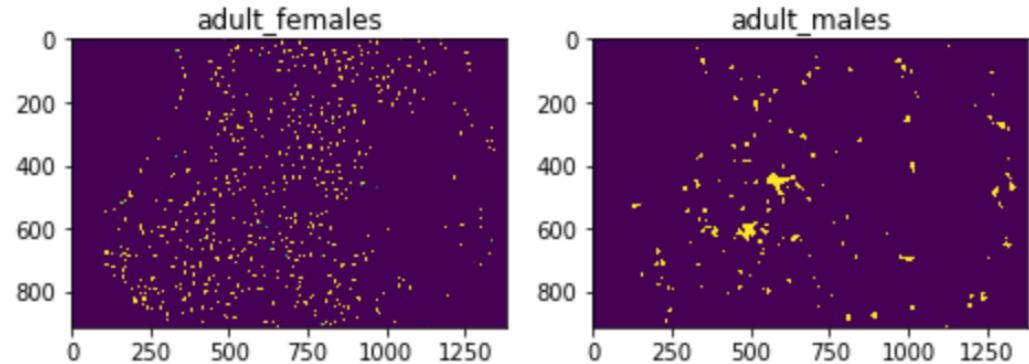
Credits Yinan Zhou

<https://github.com/marioZYN/FC-CNN-Demo>

Sealion HeatMaps



→ FCNN →

A sequence of three red arrows pointing from left to right. The first arrow points from the aerial photograph to a red-outlined rectangular box containing the text "FCNN". The second arrow points from the "FCNN" box to the heatmaps.

Credits Yinan Zhou

<https://github.com/marioZYN/FC-CNN-Demo>

G. Boracchi

Fully convolutional networks in keras

It is necessary to get and set the weights of networks by means of the methods **get_weights** and **set_weights**

- get the weights of the trained CNN

```
w7, b7 = model.layers[7].get_weights()
```

- reshape these weights to become ten 1x1 convolutional filters having depth equal to 256 (these sizes are dictated by the sizes of the dense layer in the “traditional” CNN)

```
w7.reshape(1, 1, 256, 10)
```

- assign these weights to the FCNN architecture

```
model2.layers[7].set_weights(w7, b7)
```