University of Cyprus

MAI645 - Machine Learning for Graphics and Computer Vision

Andreas Aristidou, PhD

Spring Semester 2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Image Classification: *CNN Architectures*

These notes are based on the work of Fei-Fei Li, Jiajun Wu, Ruohan Gao,

CS231 - Deep Learning for Computer Vision

LELAND STANFORD JUNIOR UNIVERSITY
1891

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of artificial neural network that is commonly used for image recognition and classification. They are designed to automatically and adaptively learn spatial hierarchies of features from raw input data.

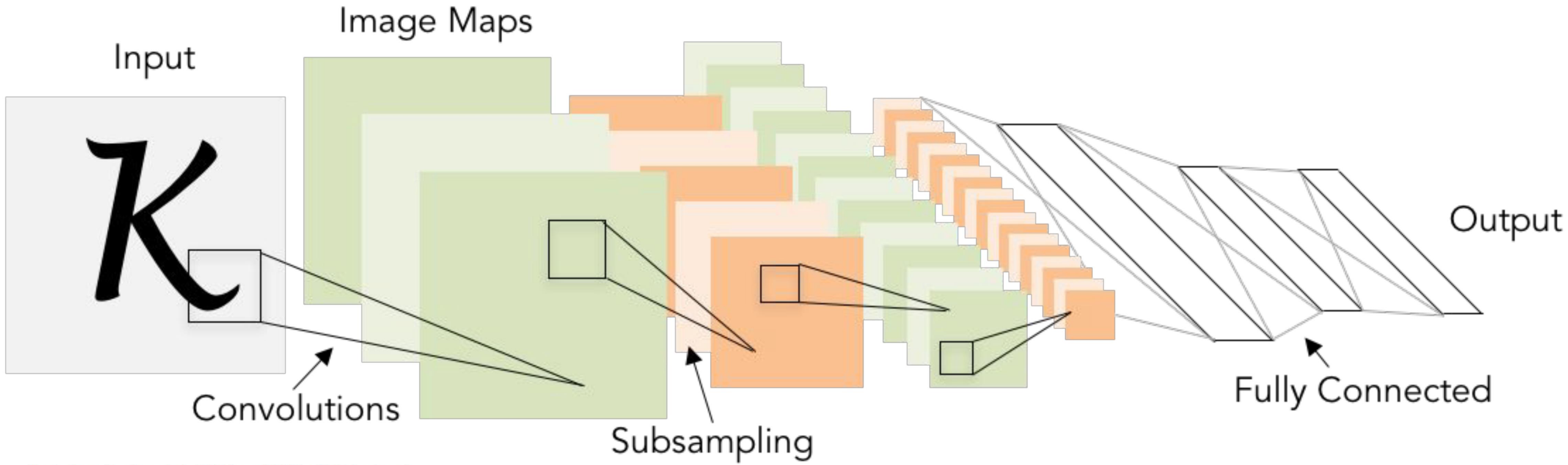


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks

CNNs are comprised of several layers, including **convolutional layers**, **pooling layers**, and **fully connected layers**.

- The convolutional layers use filters (also called kernels) to scan the input image and extract features, which are then passed on to the next layer.
- The pooling layers downsample the output from the convolutional layers, reducing the spatial dimensions of the feature maps.
- The fully connected layers take the output from the previous layers and use it to make predictions about the input data.

One of the key advantages of CNNs is their ability to learn hierarchical representations of features. The lower layers of the network learn simple features such as edges and corners, while higher layers learn more complex features such as object parts and textures. This allows the network to make accurate predictions about the input data, even when it is presented with new and previously unseen examples.

https://www.image-net.org/

Convolutional Neural Networks: *A bit of history*

ImageNet is a large-scale visual recognition challenge, in which researchers compete to build models that can classify images into one of 1,000 categories. The challenge uses a dataset of over one million labeled images, making it one of the largest and most comprehensive datasets of its kind.

The ImageNet challenge has had a significant impact on the field of image processing and computer vision. Prior to the challenge, many researchers were working on relatively small datasets and using handcrafted features to identify objects in images. The ImageNet dataset and challenge helped to shift the focus towards the use of deep learning techniques, particularly Convolutional Neural Networks (CNNs), for image classification.

In 2012, the winning team in the ImageNet challenge used a CNN architecture called AlexNet to achieve a significant improvement in image classification accuracy. This breakthrough demonstrated the power of deep learning for image processing and helped to spark a revolution in the field. Since then, researchers have continued to develop increasingly sophisticated CNN architectures, which have been applied to a wide range of image processing tasks, including object detection, semantic segmentation, and image captioning.

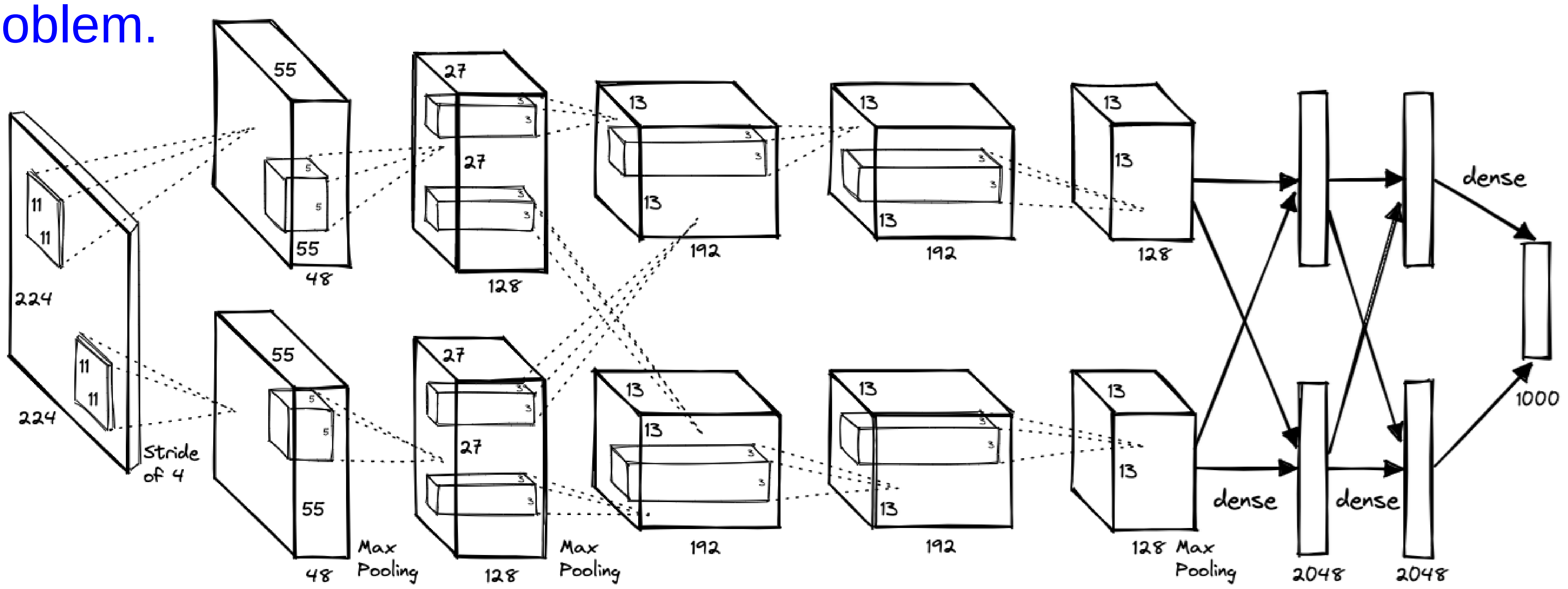
In addition to advancing the state of the art in image processing, the ImageNet challenge has also led to the development of new techniques for data augmentation, regularization, and optimization, which have helped to improve the robustness and generalization of deep learning models.

Convolutional Neural Networks: *A bit of history*

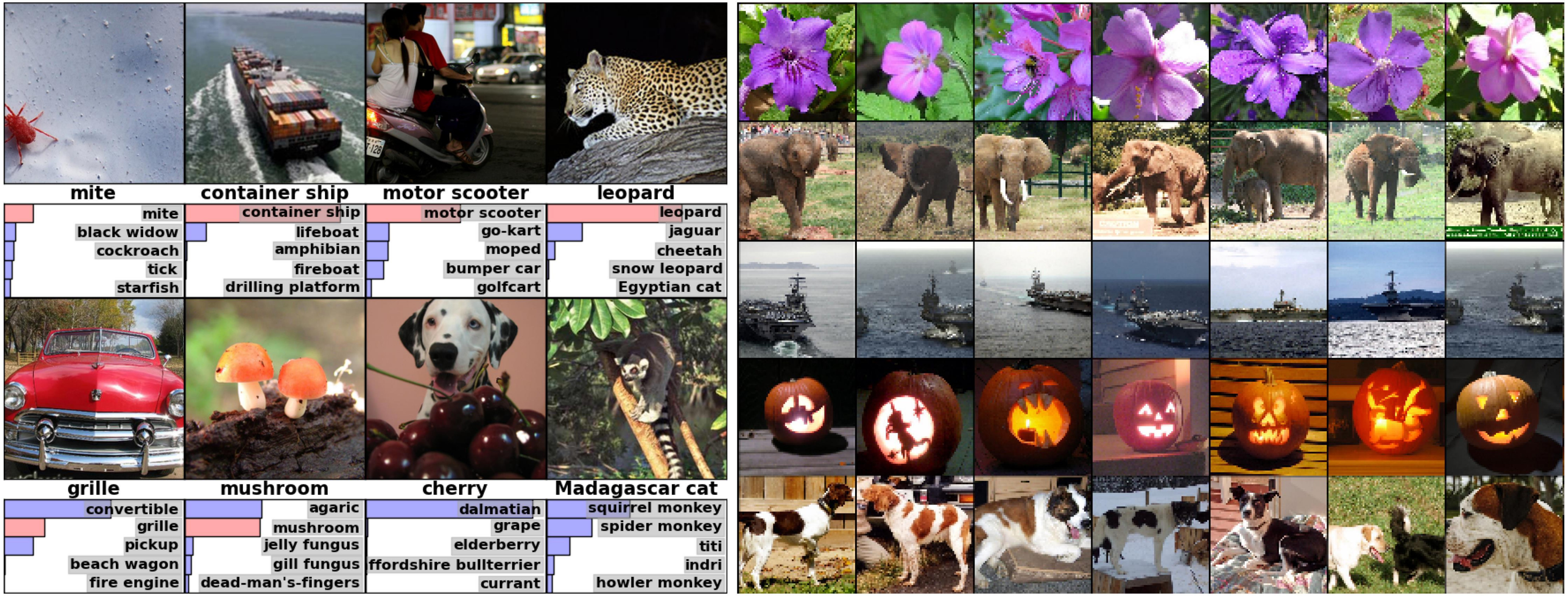
AlexNet is a deep convolutional neural network architecture that was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. At the time it was introduced, AlexNet was a breakthrough in the field of computer vision and image processing, achieving state-of-the-art performance on the ImageNet dataset. The architecture consists of eight layers, including five convolutional layers and three fully connected layers.

The key innovation of AlexNet was the use of **a large number of learnable parameters**. The architecture contained 60 million parameters, which was orders of magnitude larger than previous deep learning models. This enabled the network to learn more complex and abstract features from the input images, which improved its ability to classify objects.

Another important innovation of AlexNet was the use of **Rectified Linear Units** (ReLU) as the activation function. ReLU has been shown to be more effective than traditional activation functions such as sigmoid or tanh, as it enables the network to learn more quickly and avoids the vanishing gradient problem.

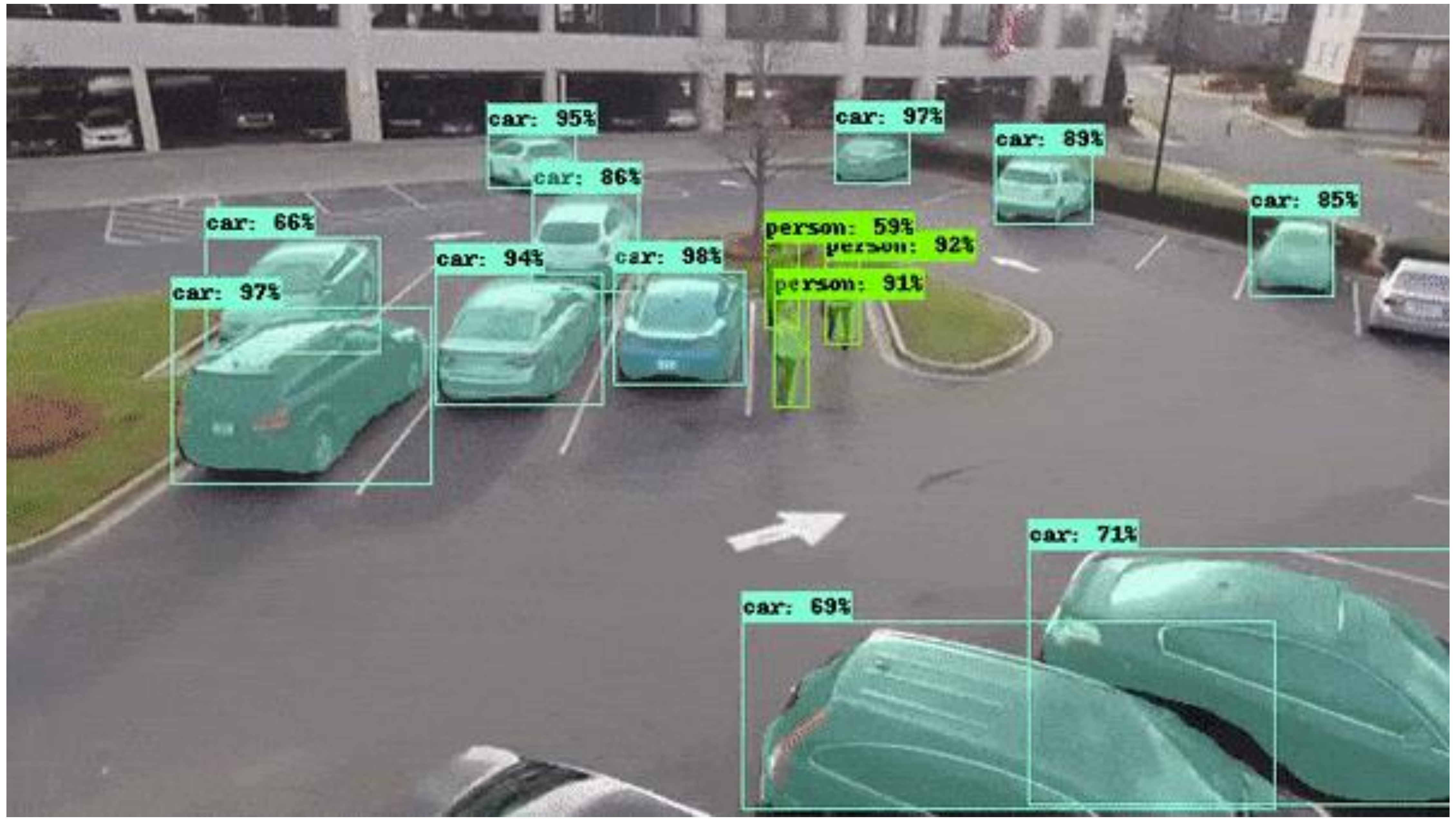


Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*



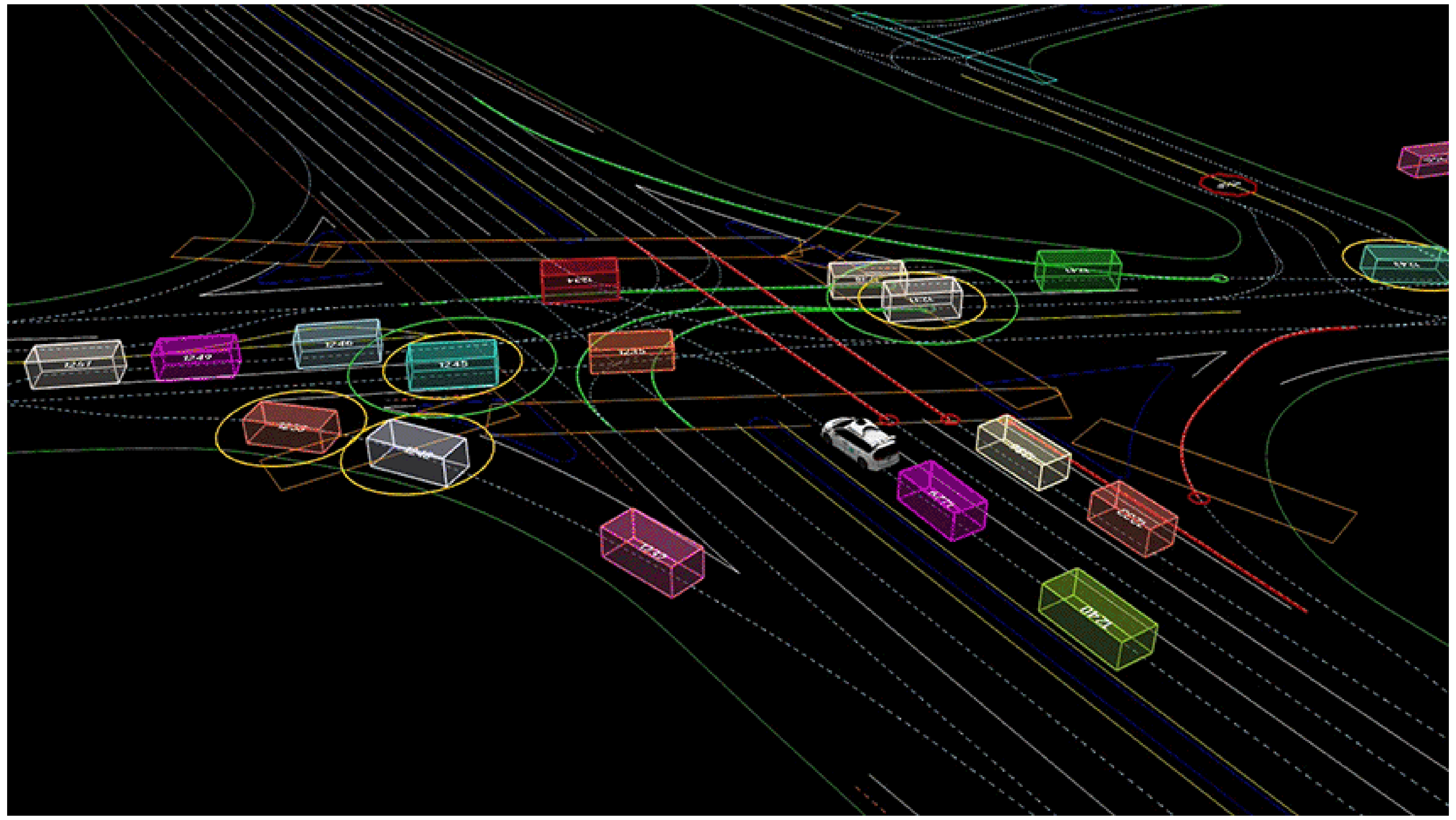
Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012

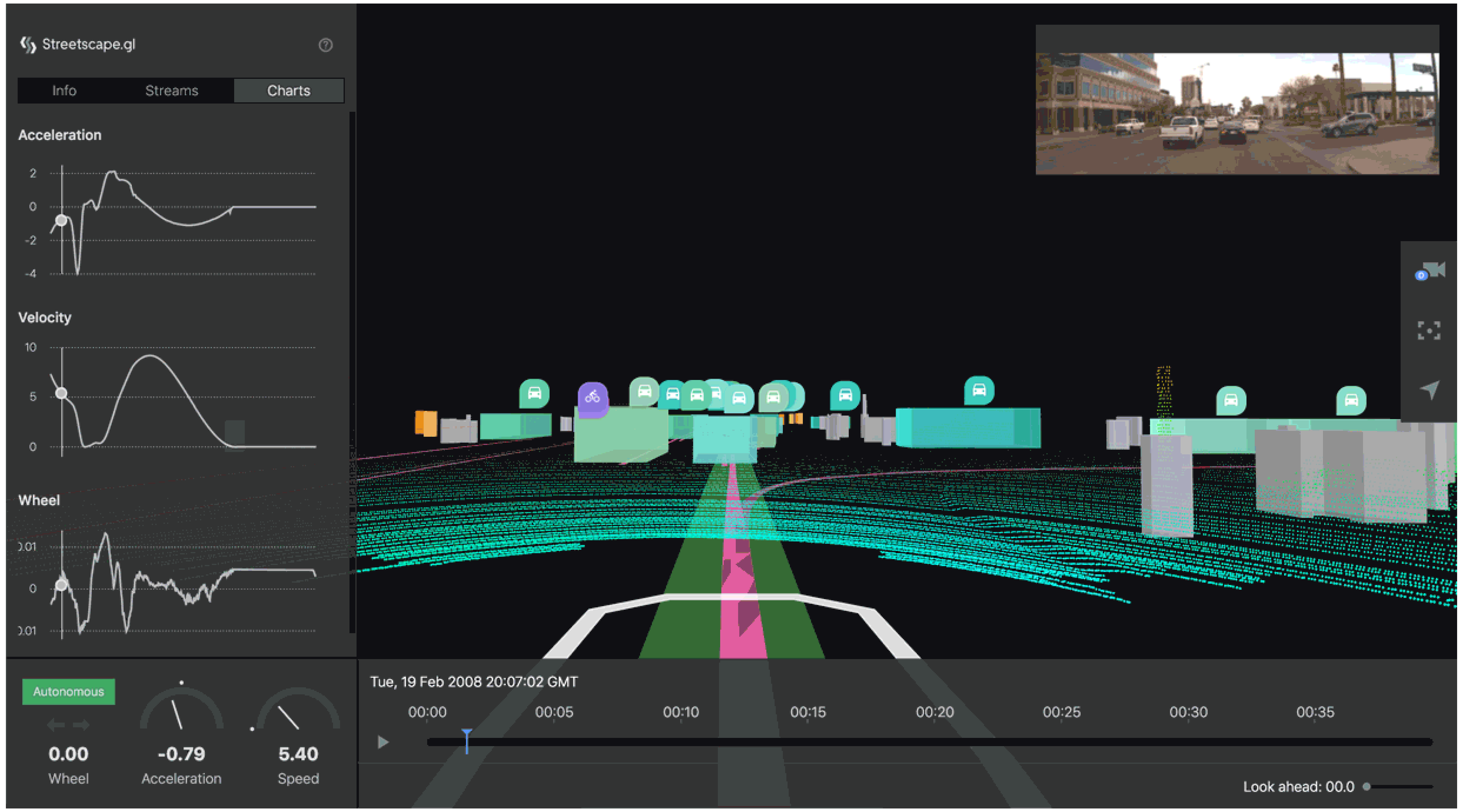
Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*



Object detection

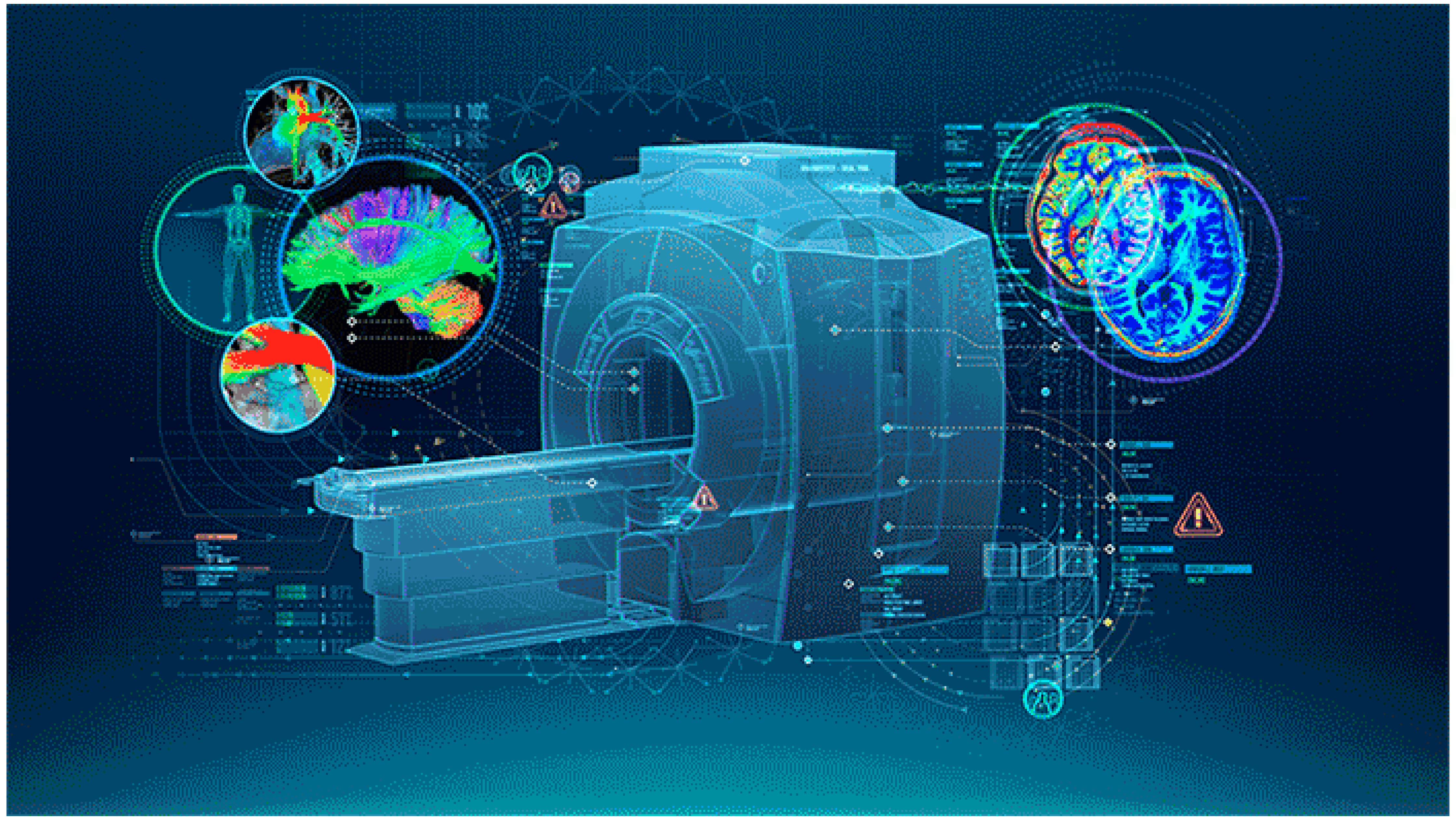
Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*

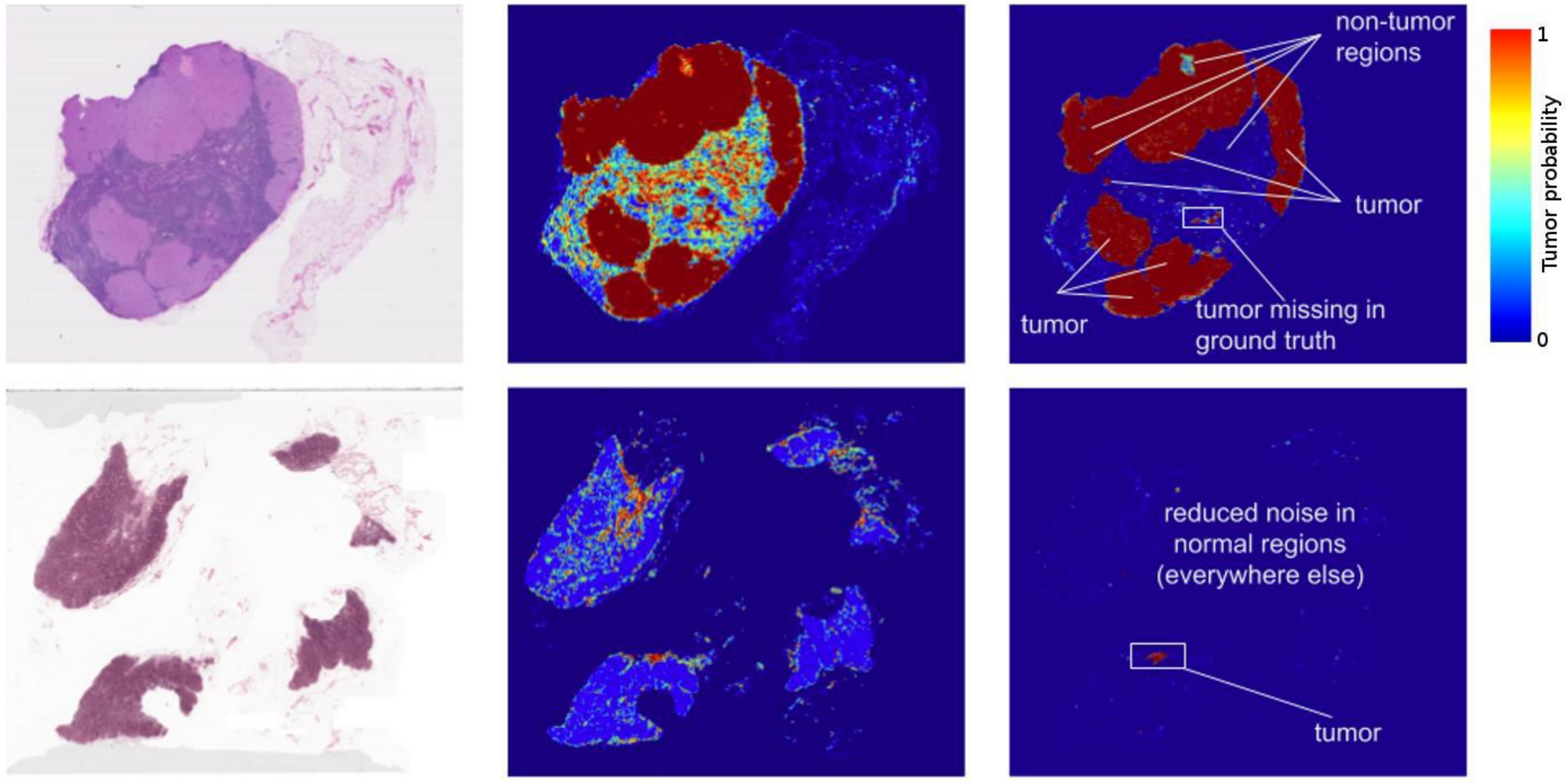




Self-driving cars

Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*





Healthcare, cancer detection

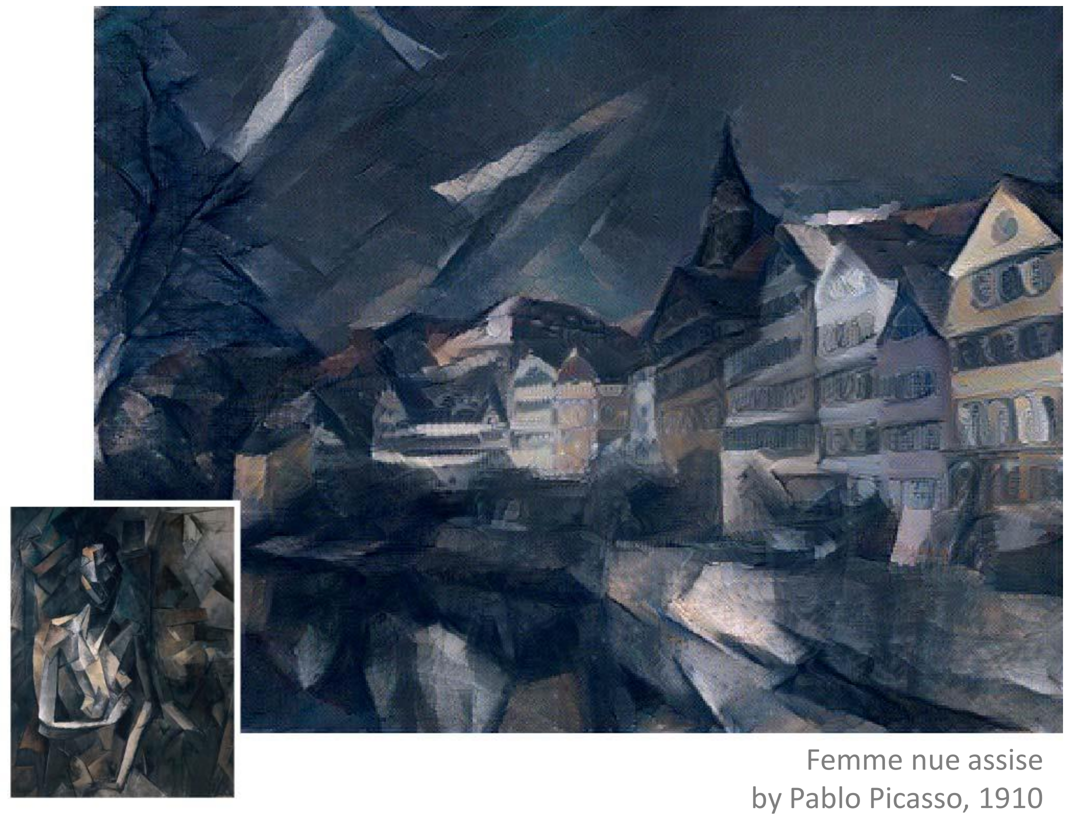
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*

Neckarfront in Tubingen, Germany ©Andreas Praefcke

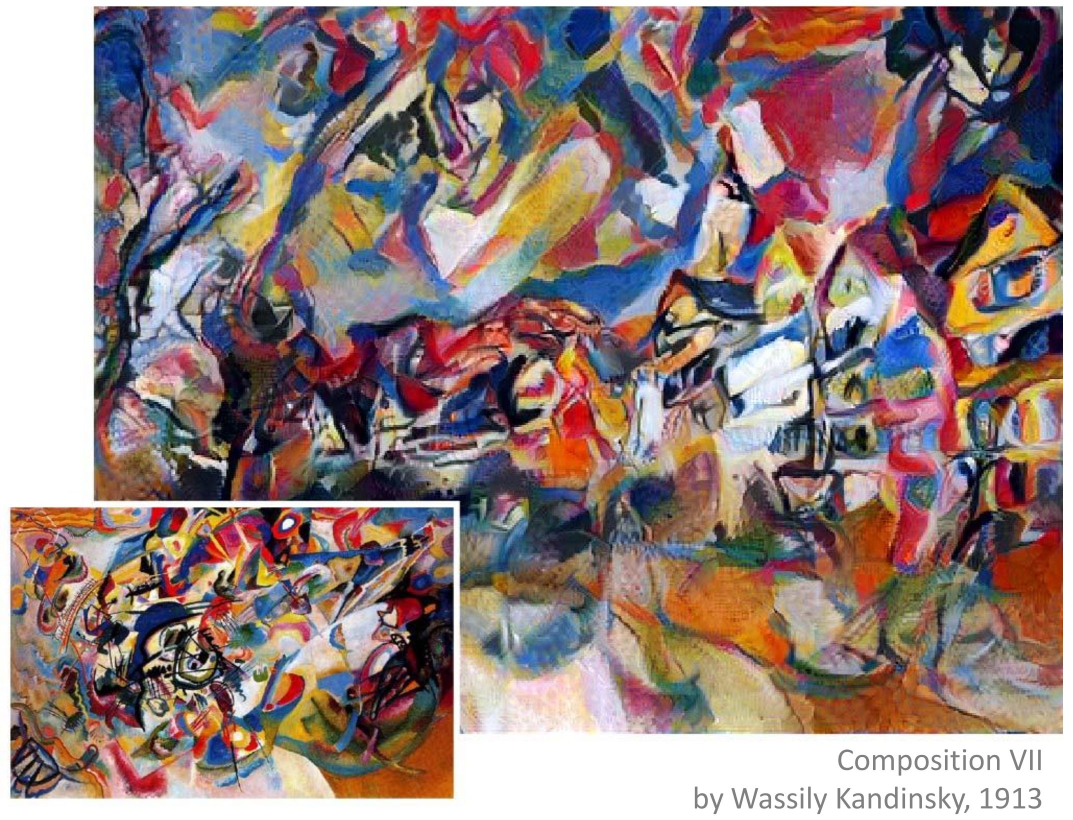
The Shipwreck of the Minotaur
by J.M.W. Turner, 1805



Femme nue assise
by Pablo Picasso, 1910

The Starry Night
by Vincent van Gogh, 1889

Der Schrei
by Edvard Munch, 1893



Composition VII
by Wassily Kandinsky, 1913

Image style transfer

Gatys et al. 2016. Image Style Transfer Using Convolutional Neural Networks. Proc. CVPR 2016.

Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*

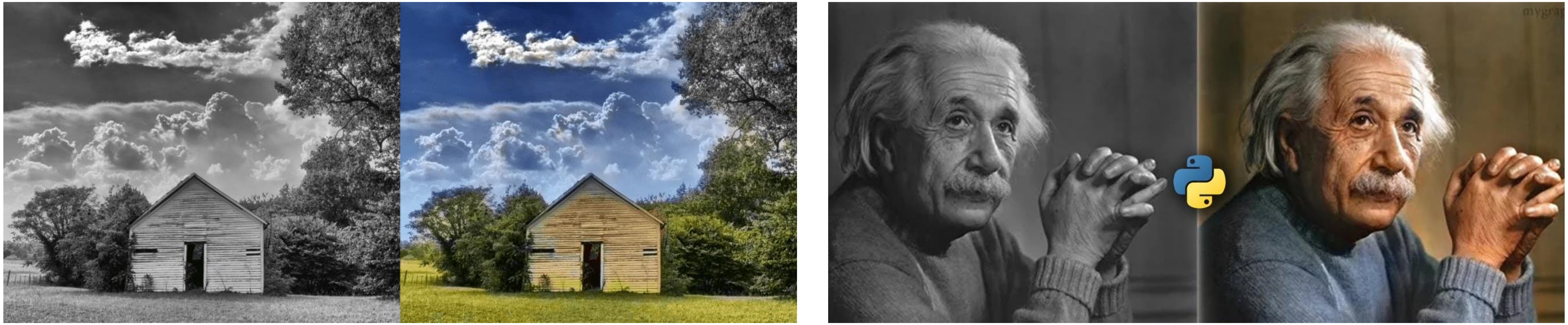
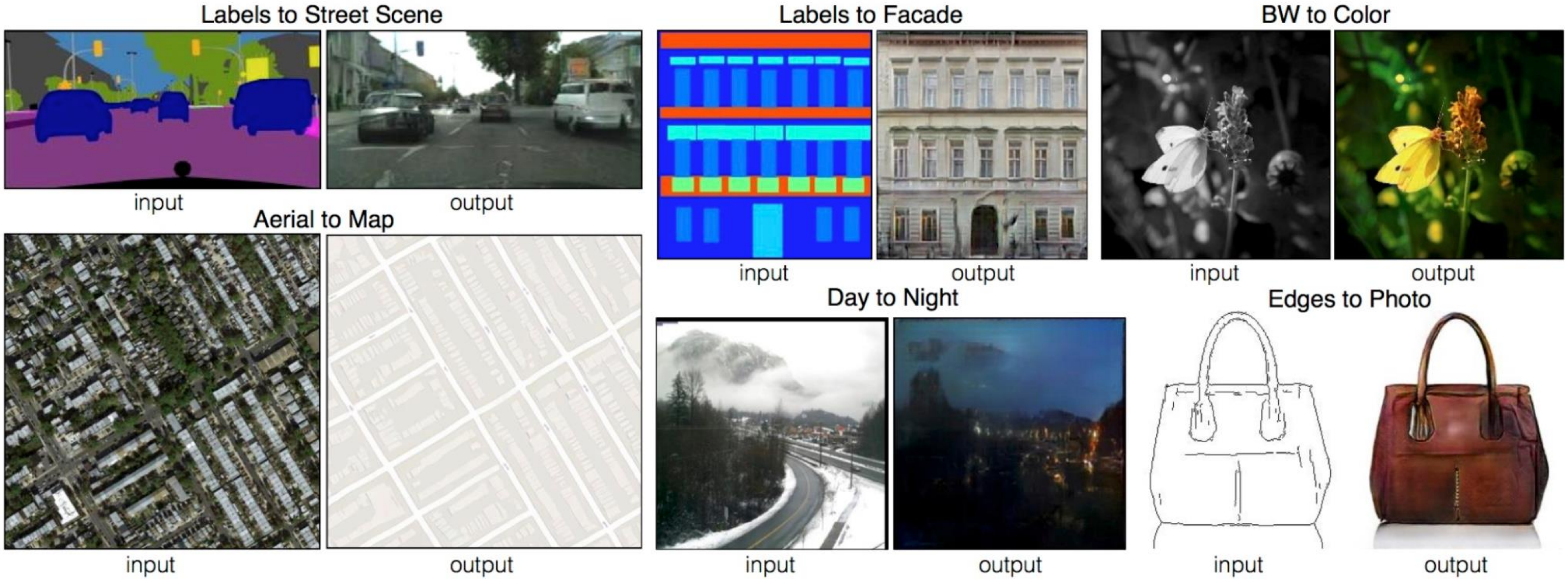


Image coloring

Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*



Convolutional Neural Networks: *Fully Connected Layer VS Convolution Layer*

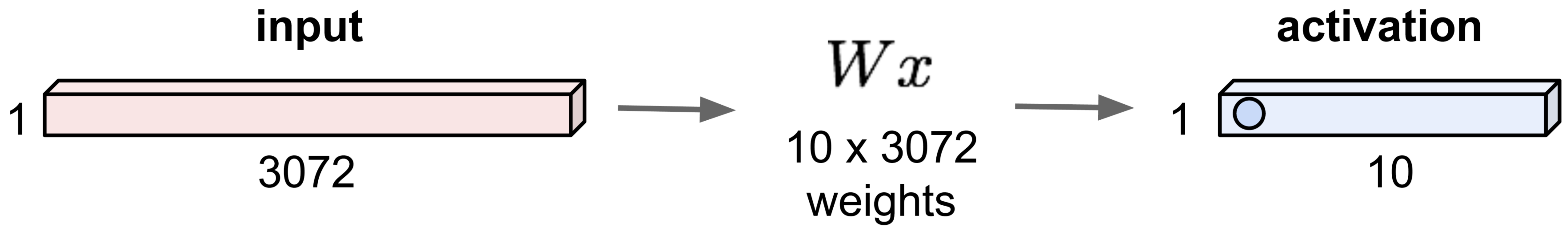
In a **fully connected layer**, also known as a dense layer, each neuron is connected to every neuron in the previous layer. This means that every input feature is processed by every neuron in the layer. The weights and biases of the layer are learned during training, which allows the network to learn complex non-linear relationships between the input and output.

In contrast, a **convolutional layer** processes input data using a set of learnable filters (also known as kernels or weights). Each filter is applied to a small region of the input data, and the output from each filter is then combined to create a feature map. This process is repeated for each region of the input data, creating a set of feature maps that represent different aspects of the input.

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Fully Connected Layer*

32x32x3 image -> stretch to 3072 x 1



input

1

3072

Wx

10 x 3072
weights

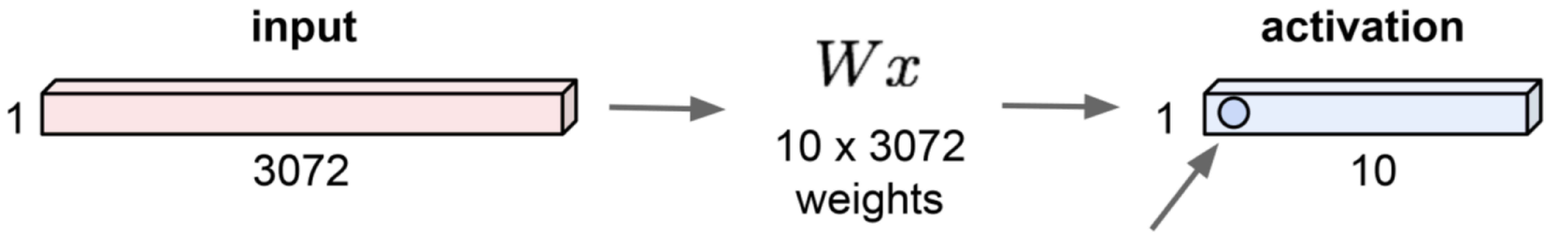
activation

1

10

Convolutional Neural Networks: *Fully Connected Layer*

32x32x3 image -> stretch to 3072 x 1



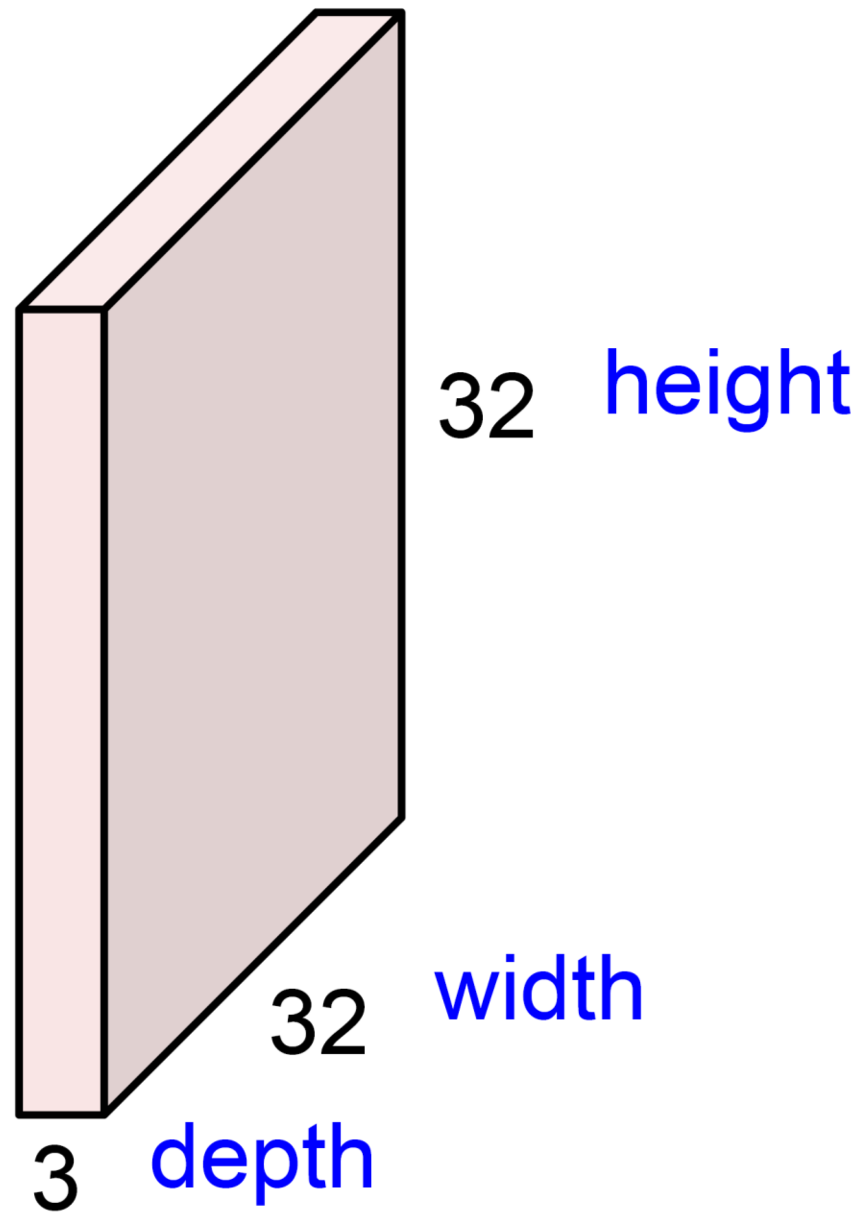
1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*

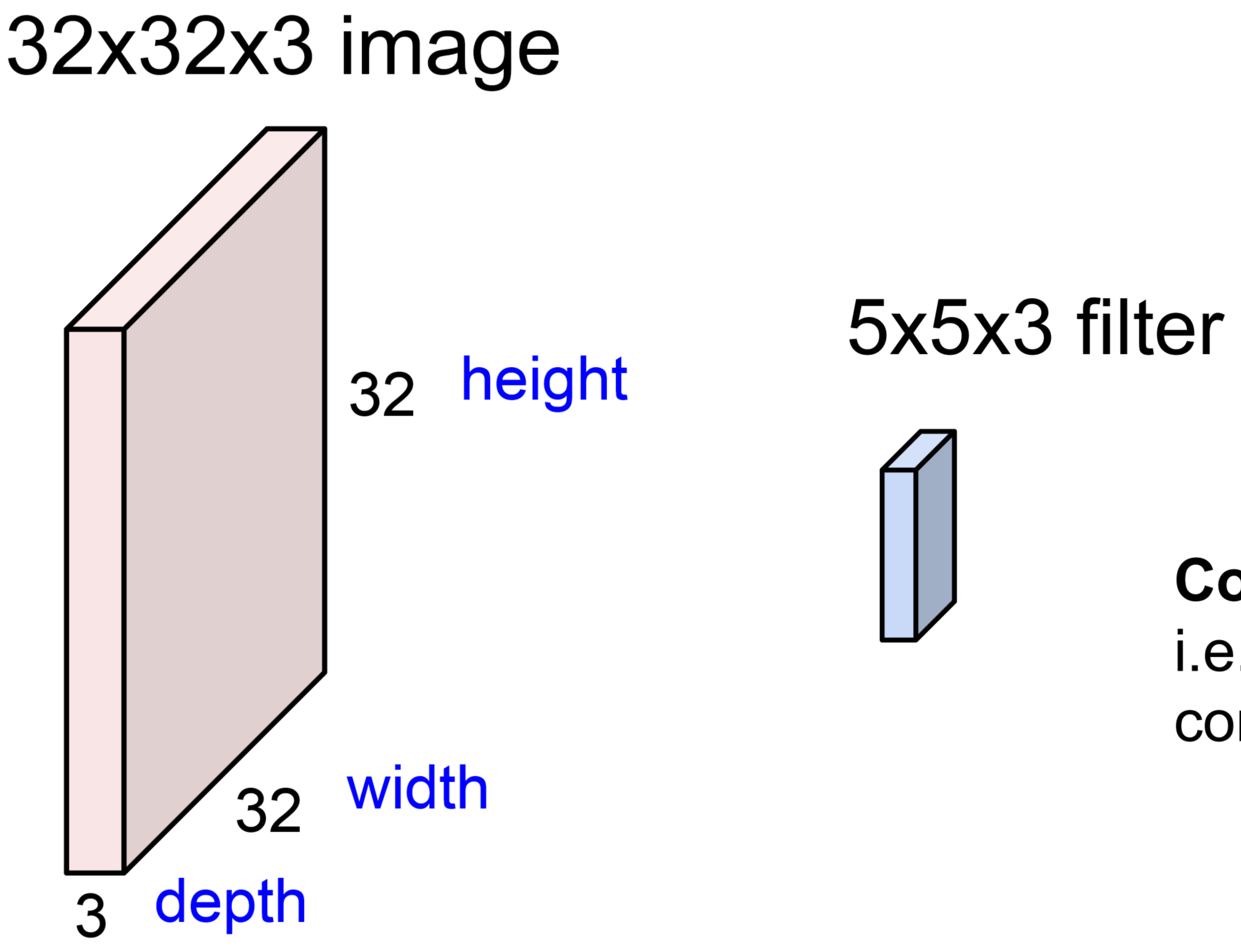
32x32x3 image -> preserve spatial structure



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*

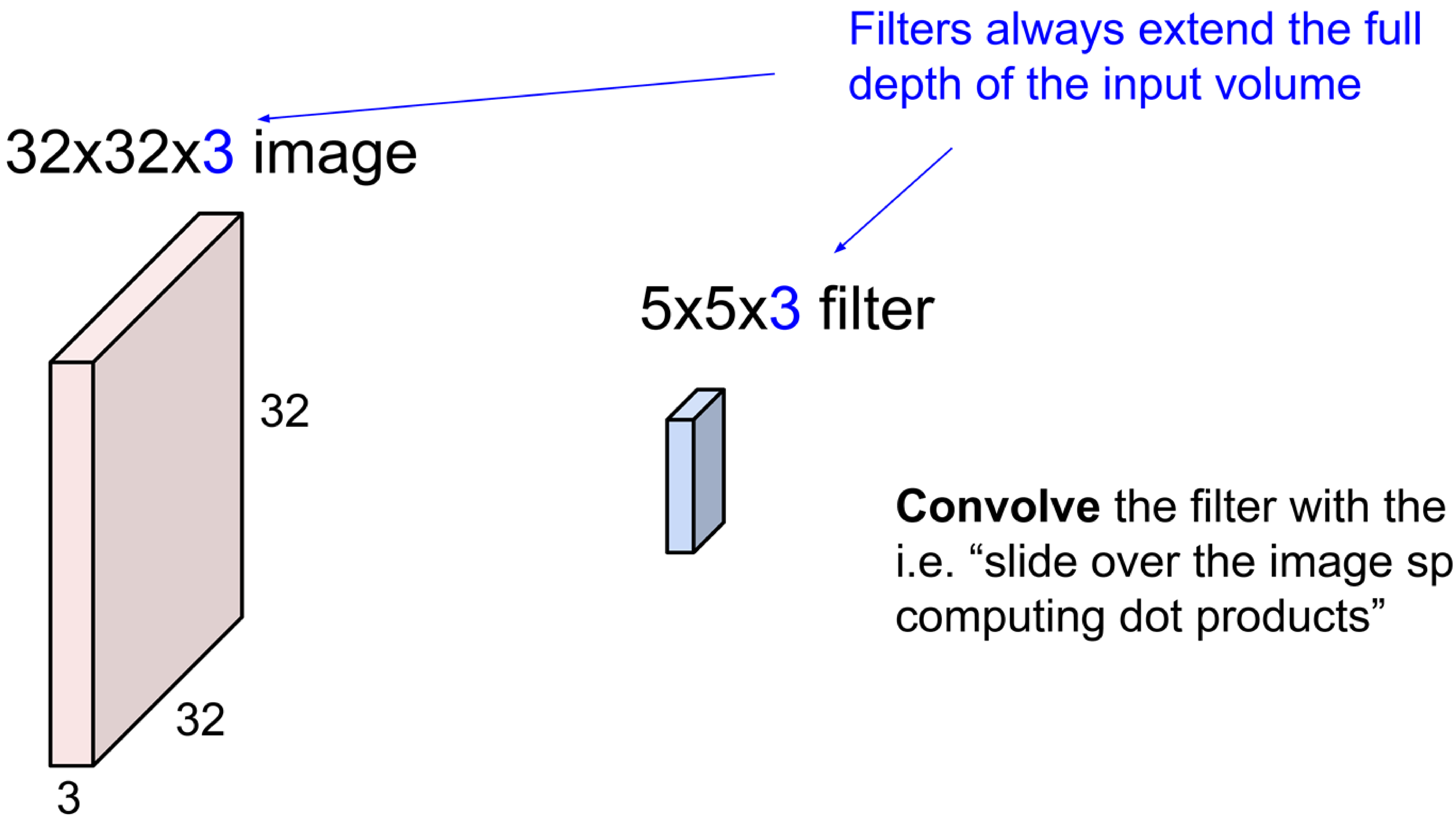


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*

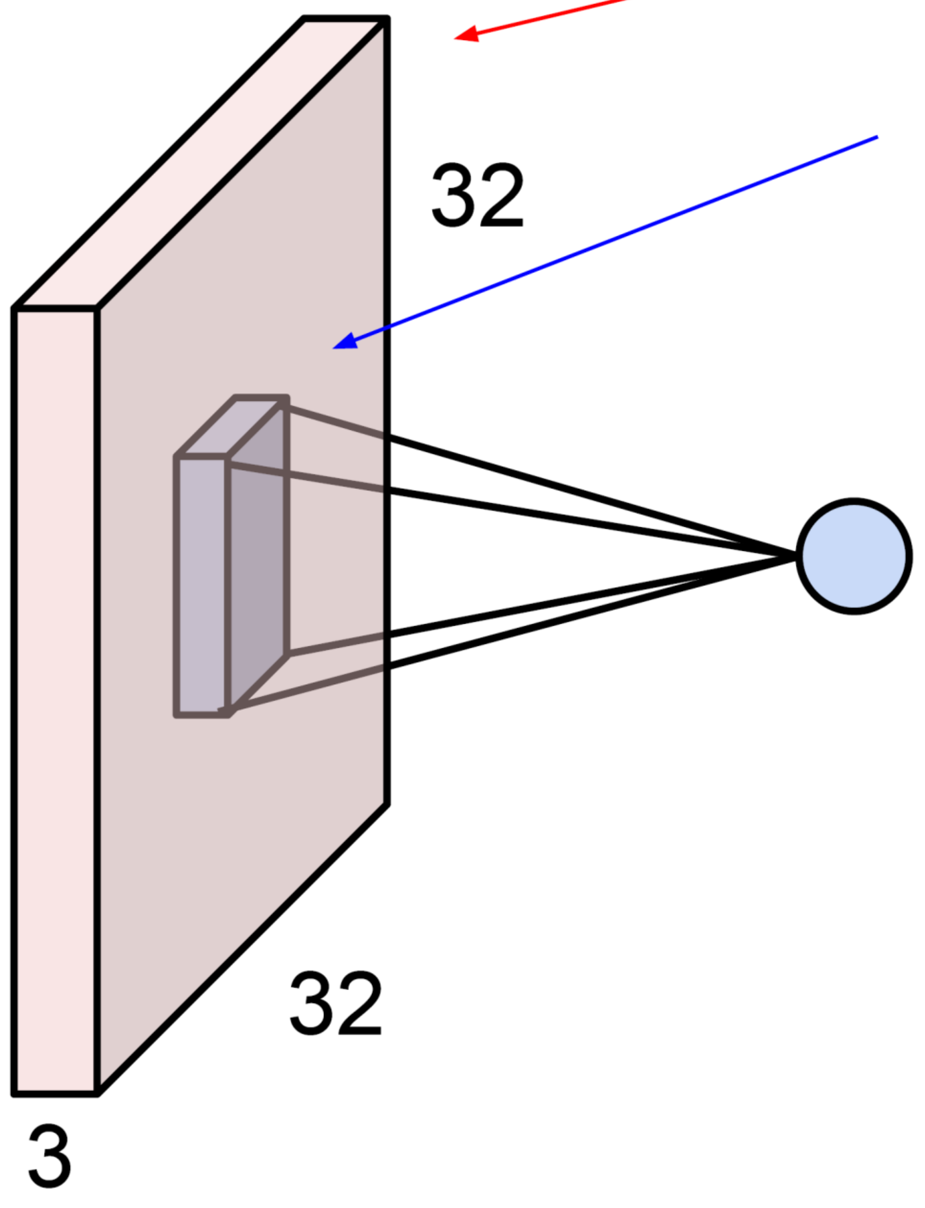


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



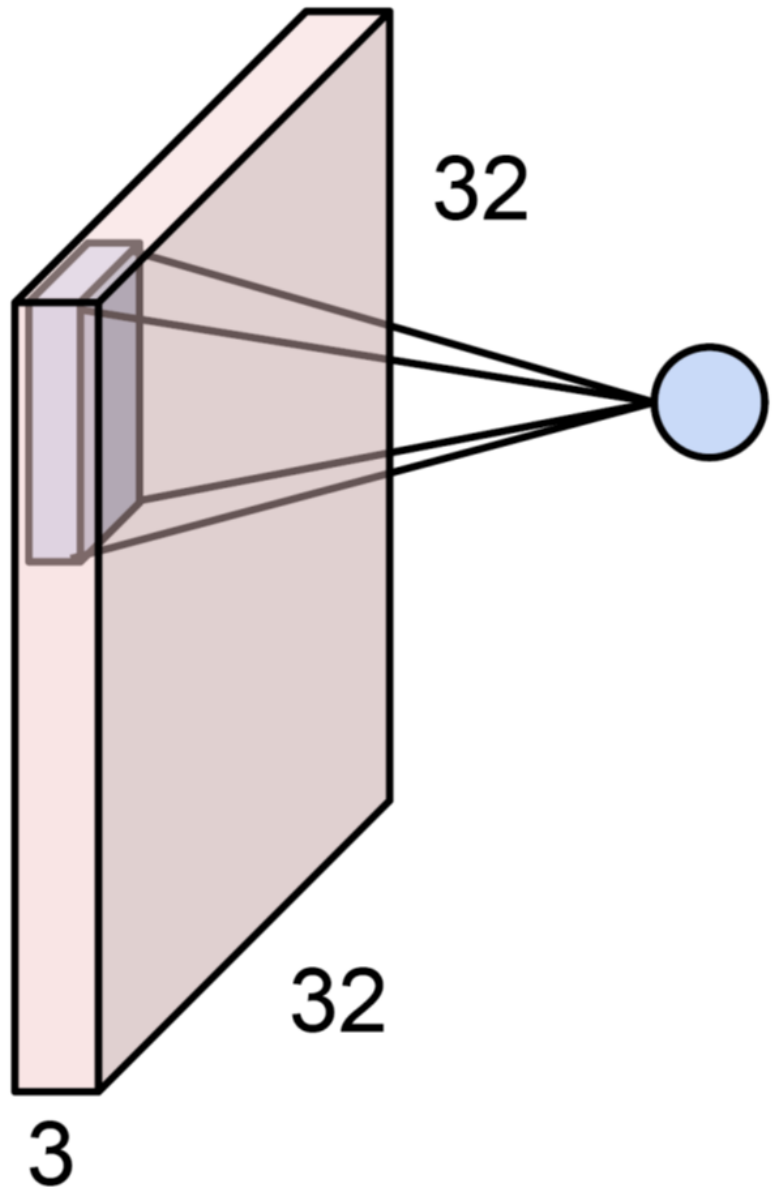
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

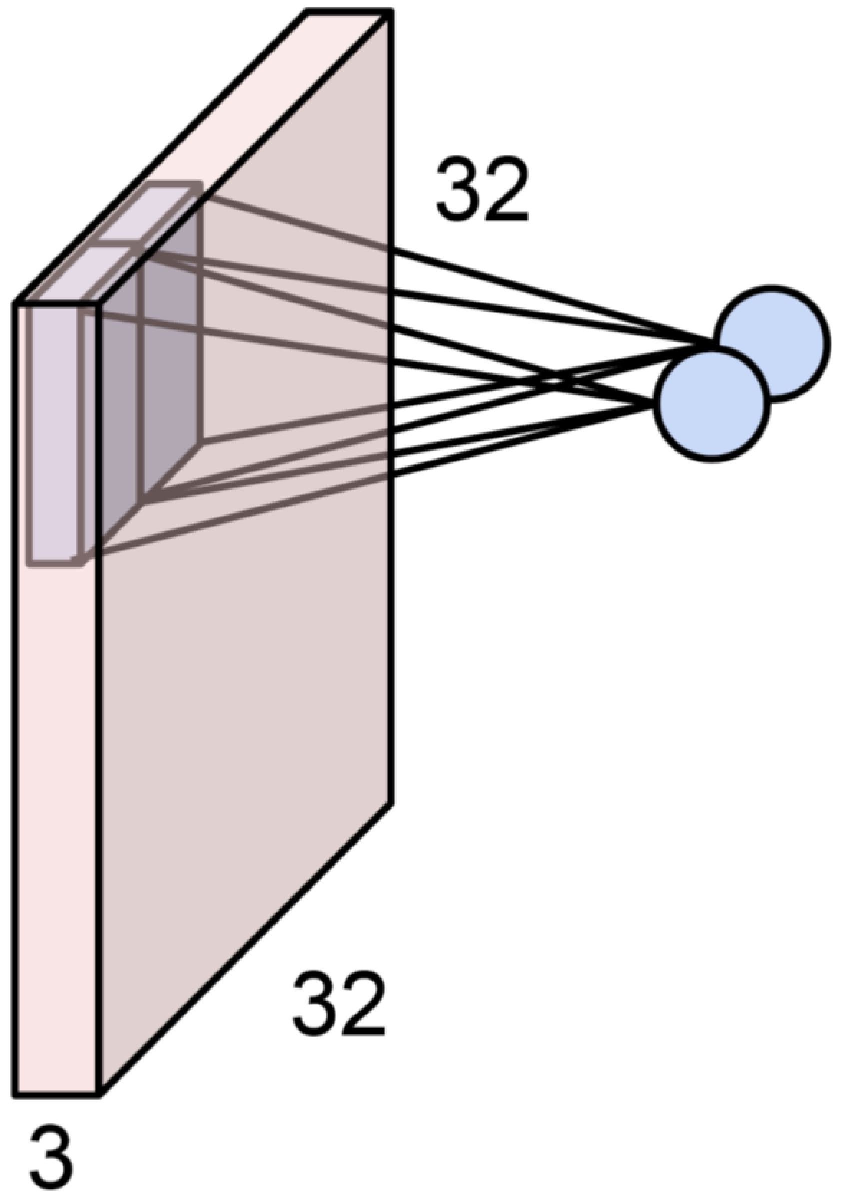
Convolutional Neural Networks: *Convolution Layer*



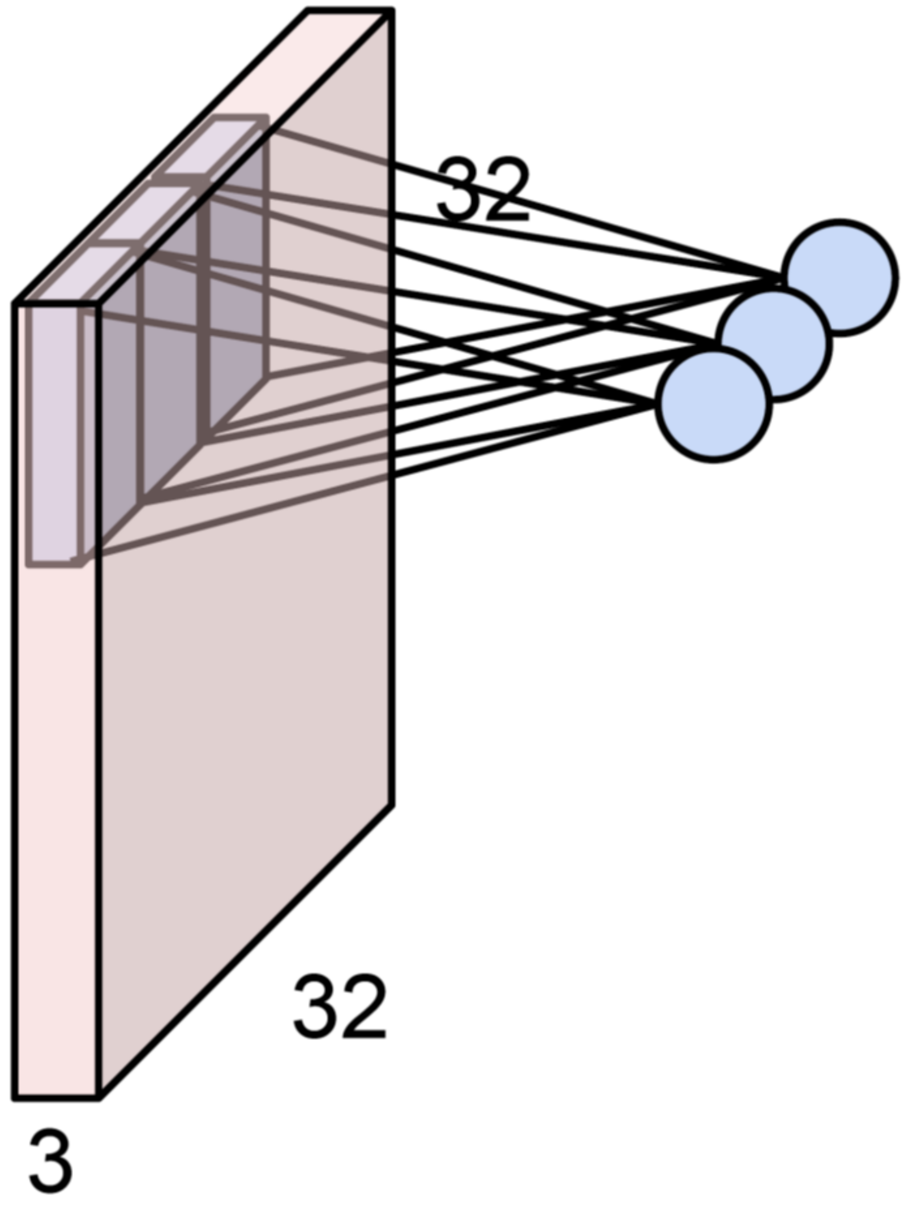
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



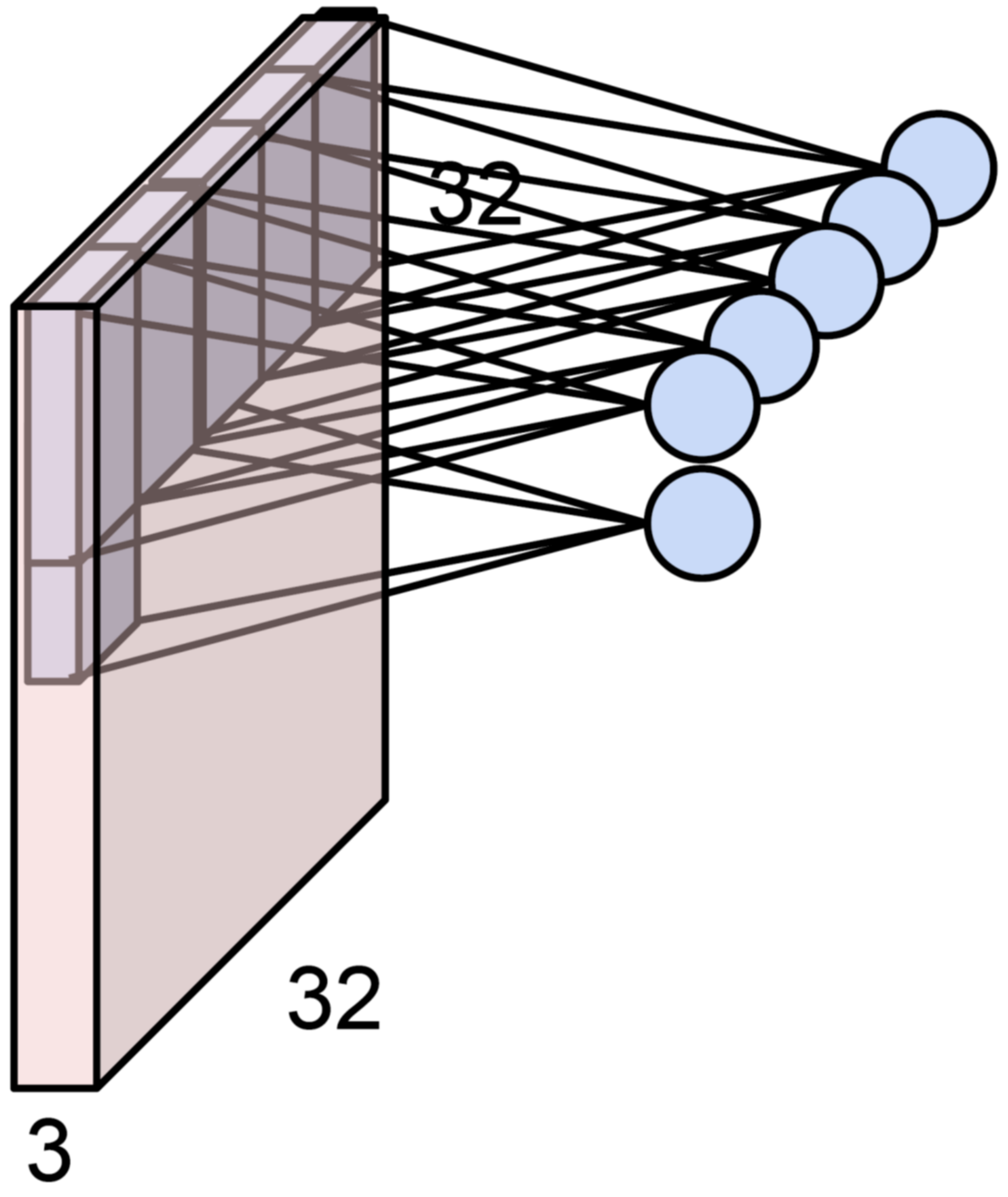
Convolutional Neural Networks: *Convolution Layer*



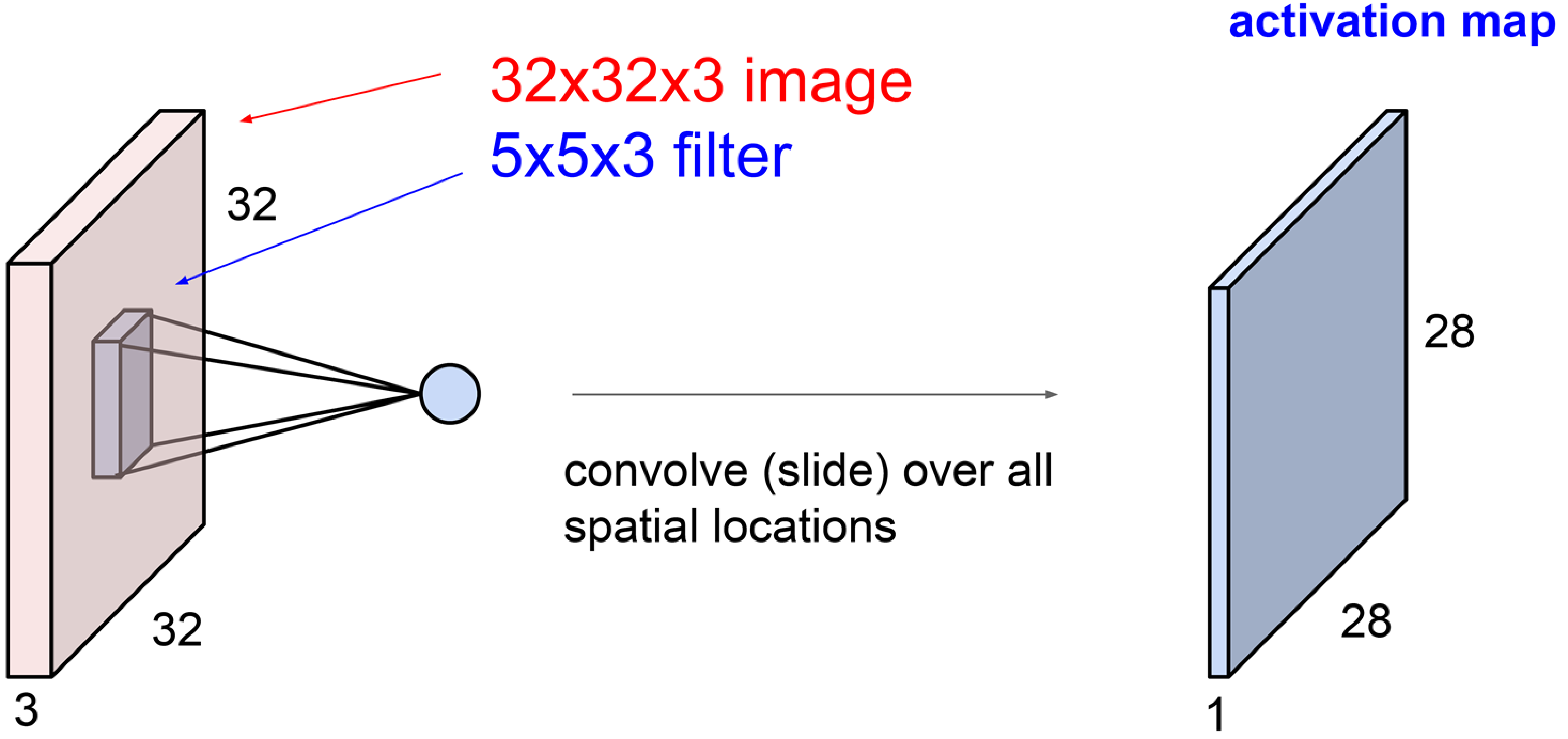
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



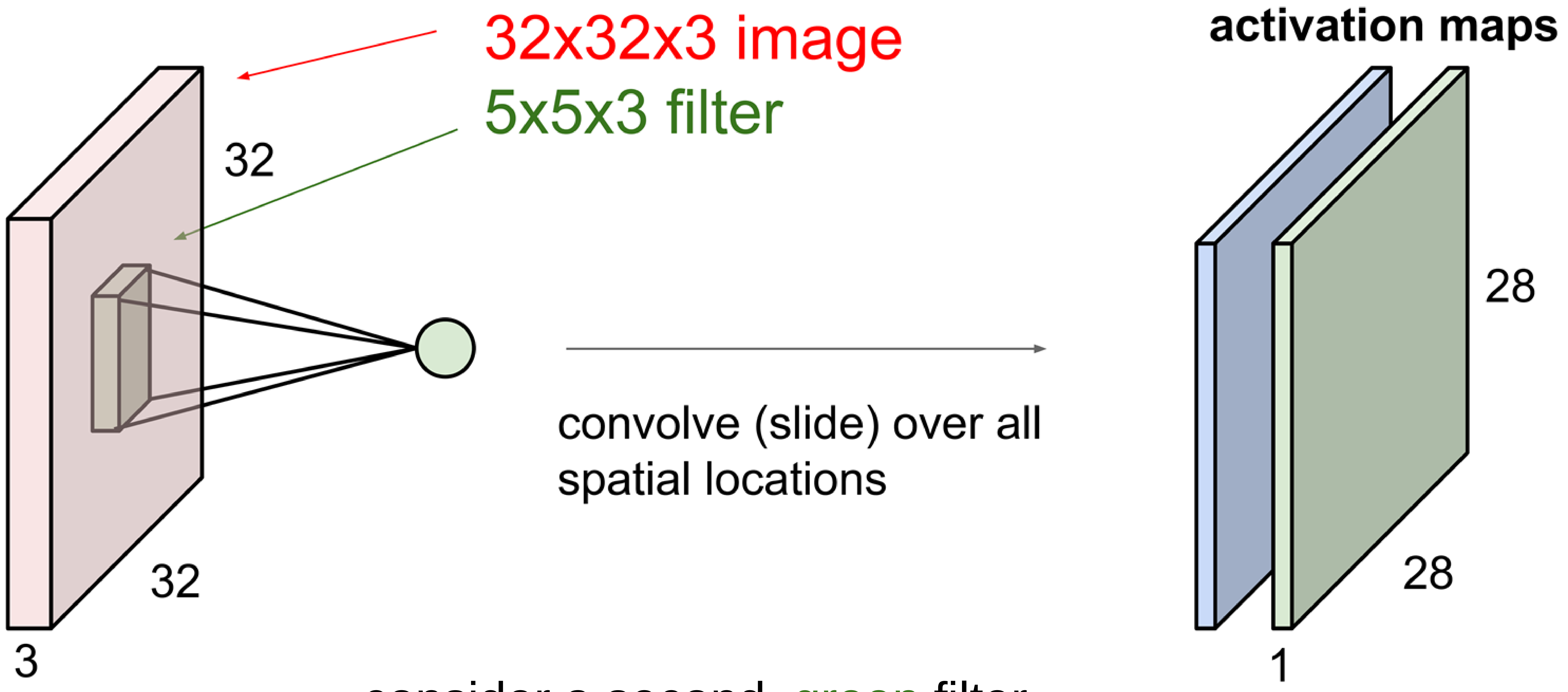
Convolutional Neural Networks: *Convolution Layer*



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

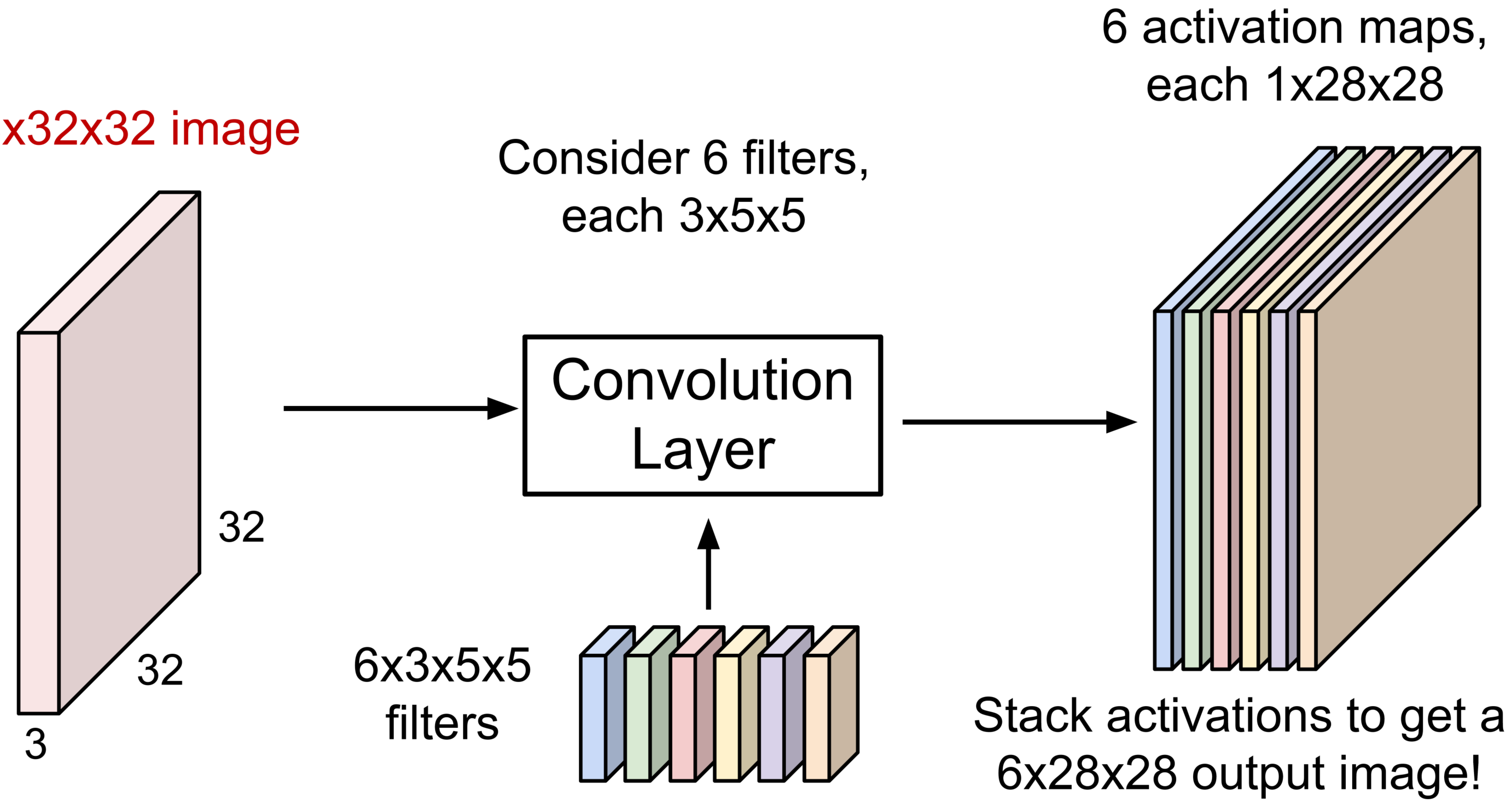
Convolutional Neural Networks: *Convolution Layer*



convolve (slide) over all
spatial locations

consider a second, green filter

Convolutional Neural Networks: *Convolution Layer*



3x32x32 image

32

32

3

Consider 6 filters,
each 3x5x5

Convolution
Layer

6x3x5x5
filters

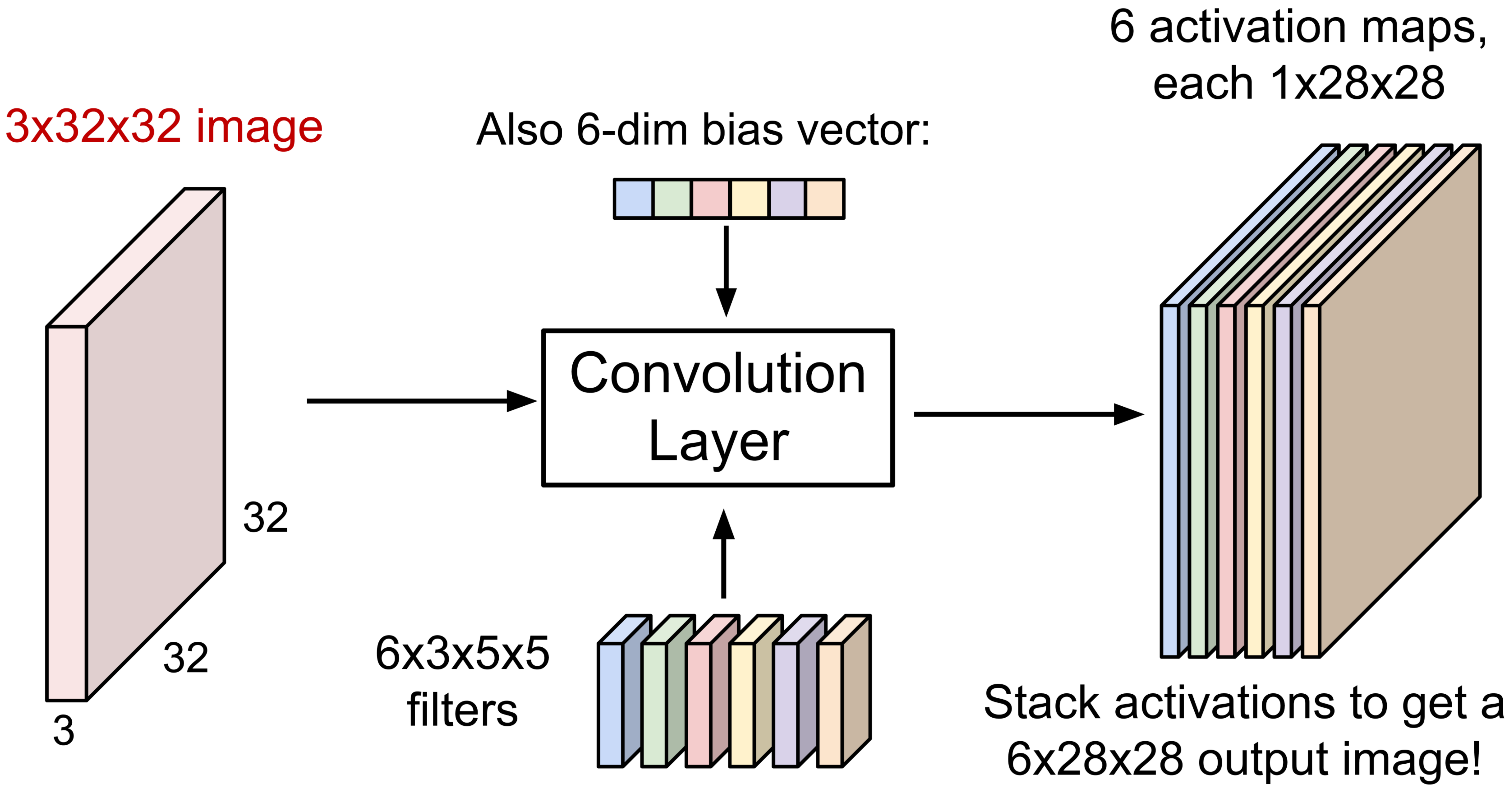
6 activation maps,
each 1x28x28

Stack activations to get a
6x28x28 output image!

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

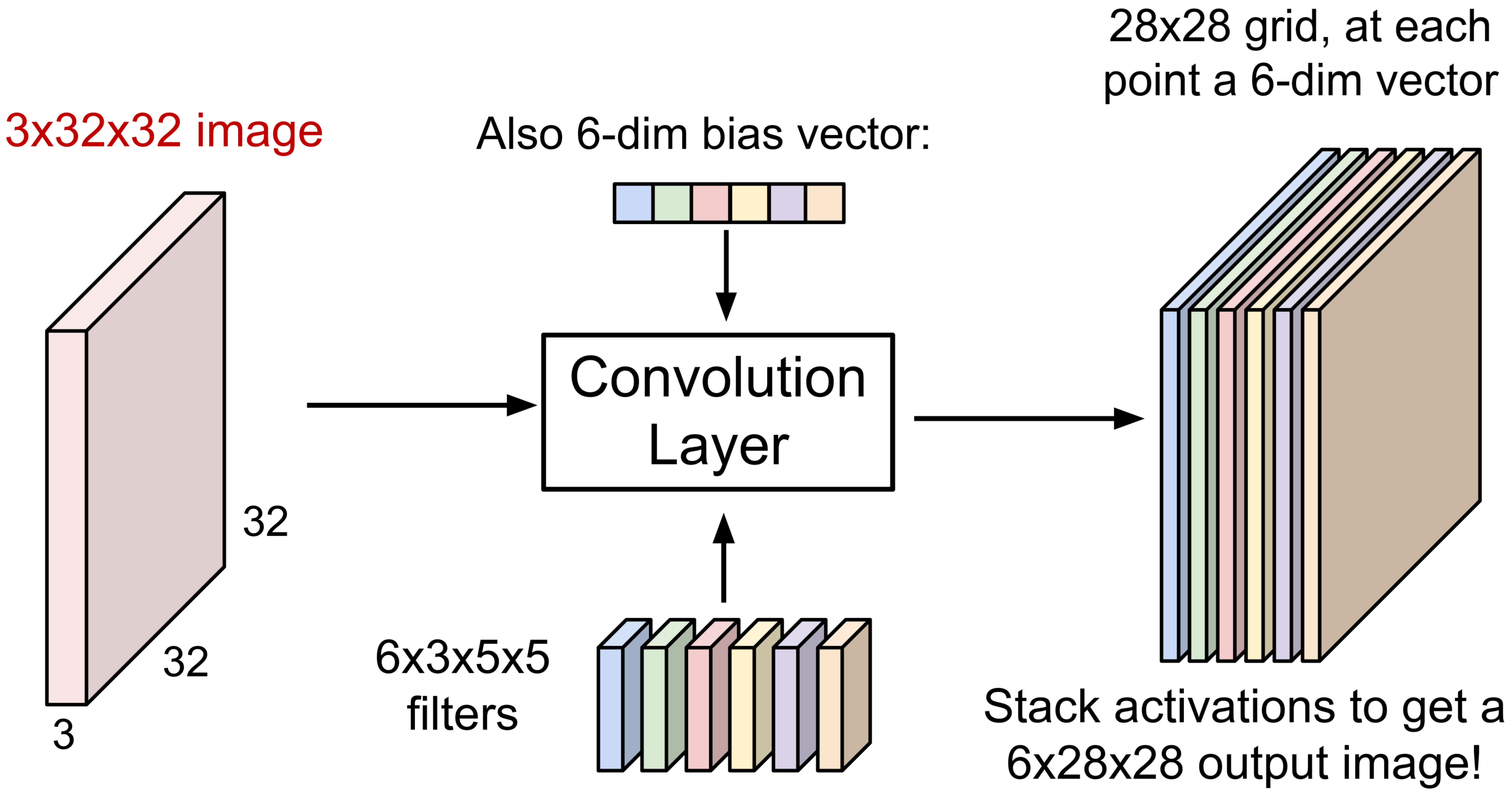
Convolutional Neural Networks: *Convolution Layer*



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



3x32x32 image

Also 6-dim bias vector:

28x28 grid, at each point a 6-dim vector

Convolution Layer

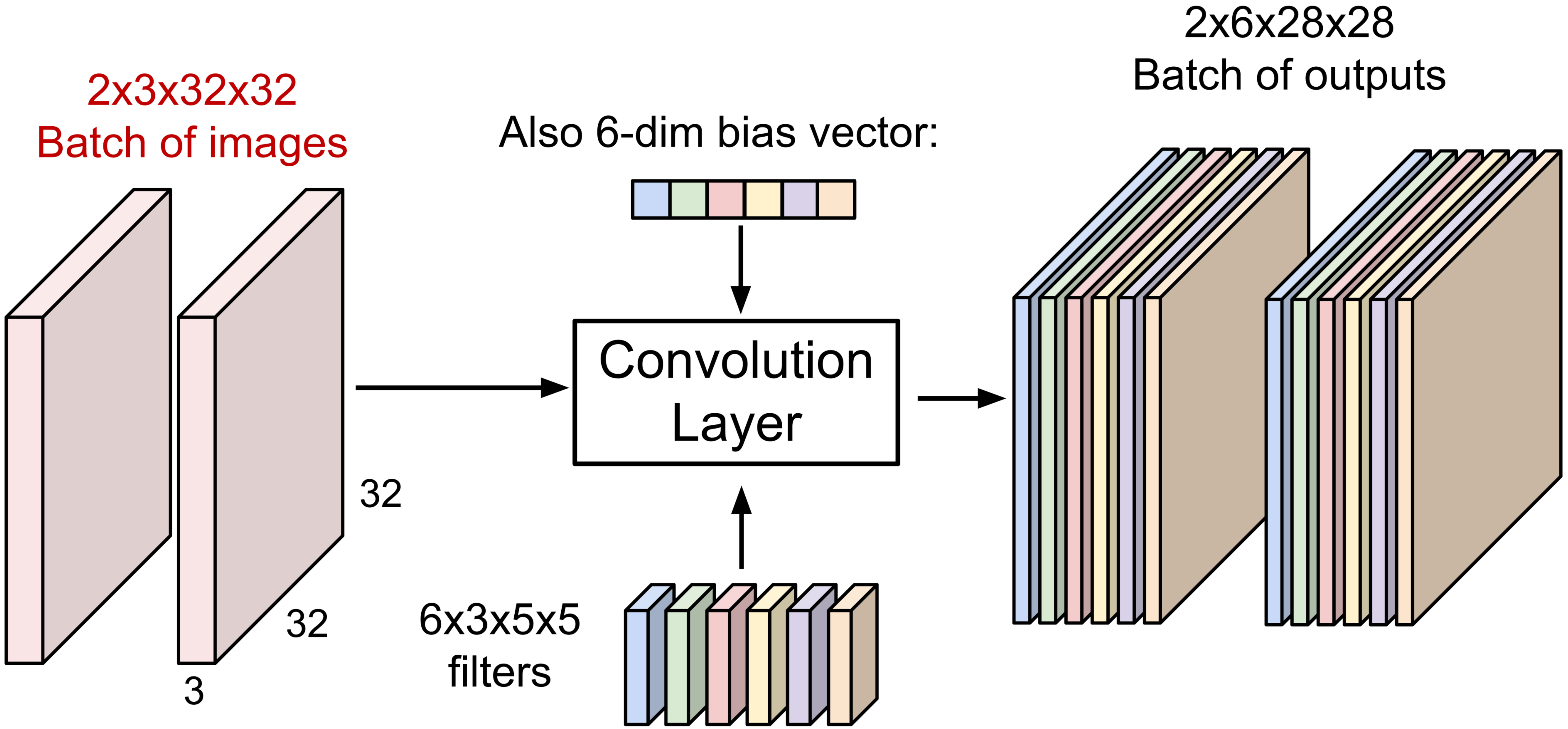
6x3x5x5 filters

Stack activations to get a 6x28x28 output image!

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



2x3x32x32
Batch of images

32

32

3

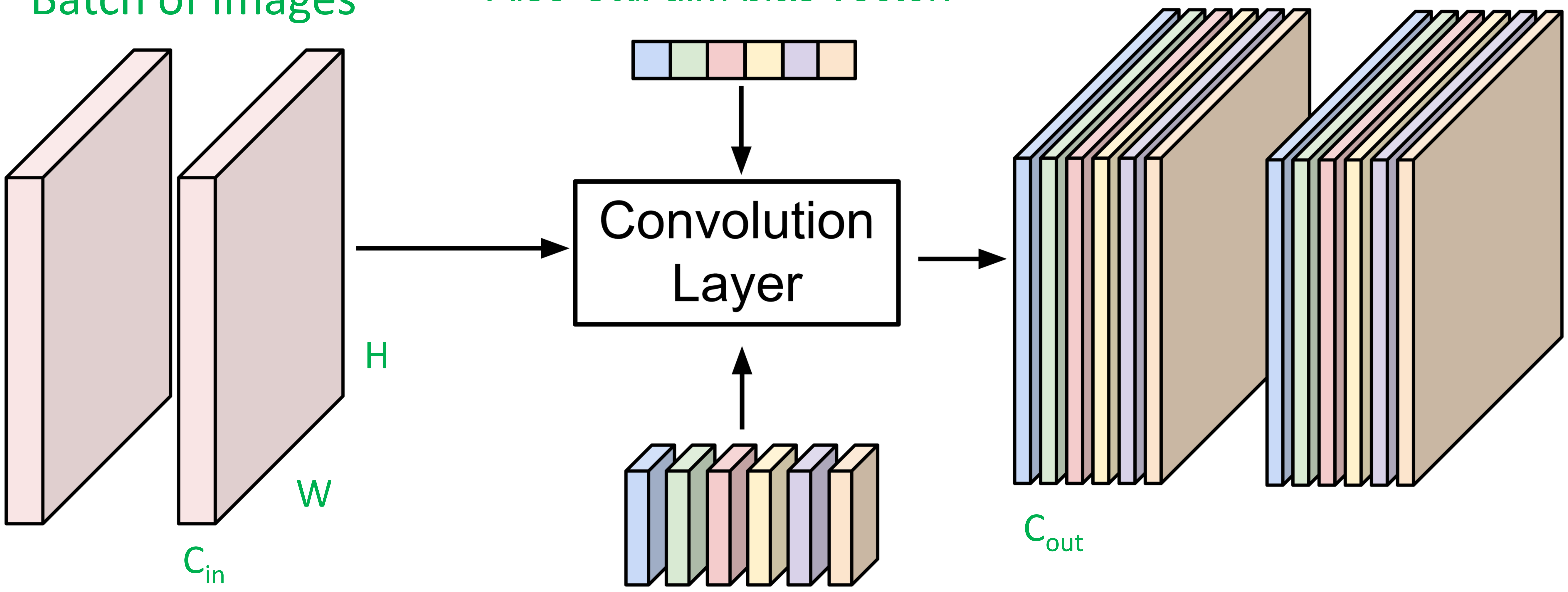
Also 6-dim bias vector:

Convolution Layer

6x3x5x5
filters

2x6x28x28
Batch of outputs

Convolutional Neural Networks: *Convolution Layer*



N x C_{in} x H x W
Batch of images

Also C_{out}-dim bias vector:

N x C_{out} x H' x W'
Batch of outputs

C_{out} x Ci_n x K_w x K_h
filters

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Fully Connected Layer VS Convolution Layer*

The main advantage of convolutional layers is their ability to capture local spatial relationships in the input data. By sharing weights across different regions of the input, convolutional layers are able to learn translation-invariant features that are useful for tasks such as image recognition and object detection.

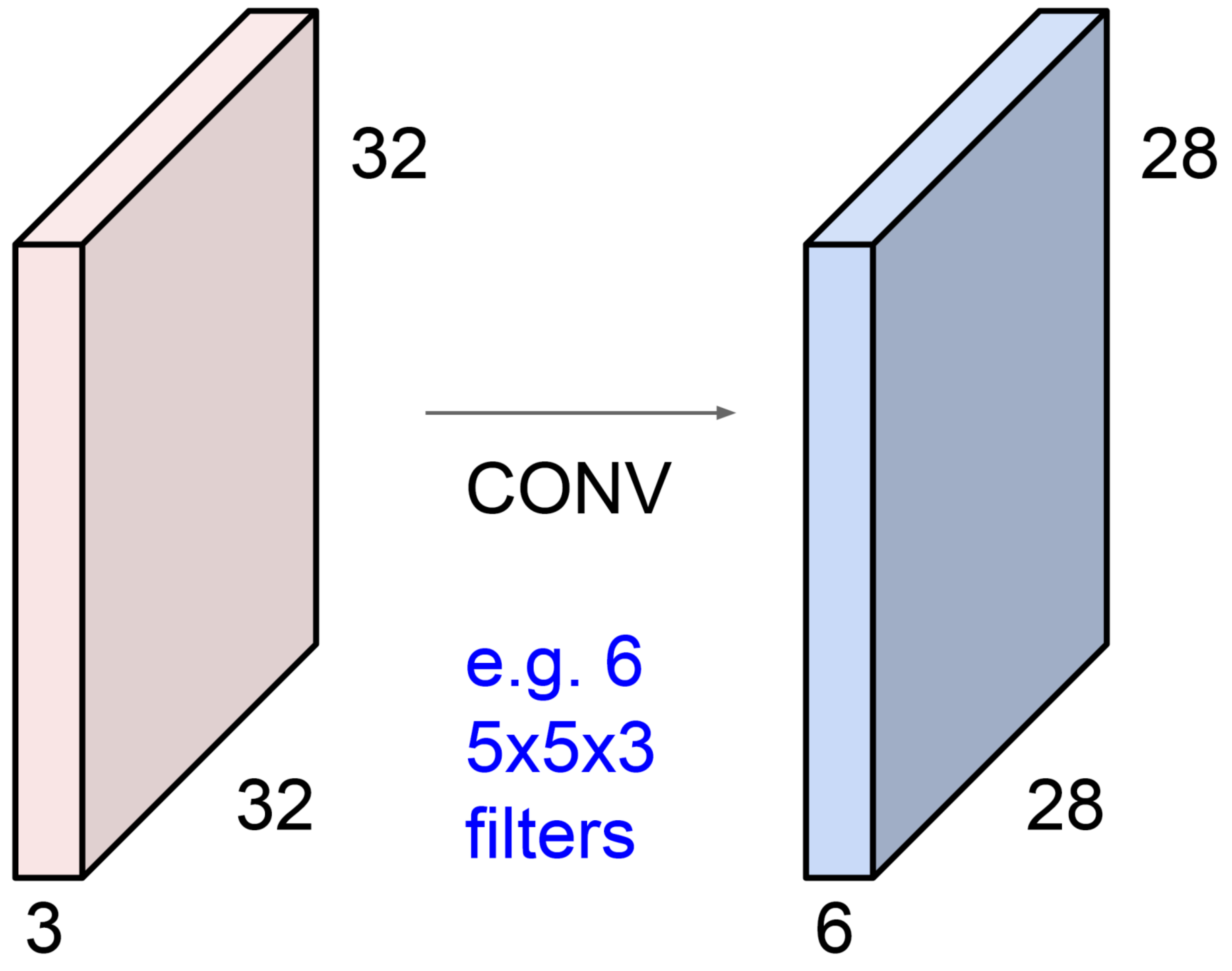
Another important difference between fully connected and convolutional layers is the way they handle input data. Fully connected layers require input data to be flattened into a one-dimensional vector, while convolutional layers can accept input data with multiple dimensions (e.g., height, width, and depth for an image). This makes convolutional layers well-suited for processing high-dimensional data such as images, audio, and video.

In practice, deep learning models typically contain a combination of fully connected and convolutional layers, along with other types of layers such as pooling layers, activation functions, and dropout layers.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

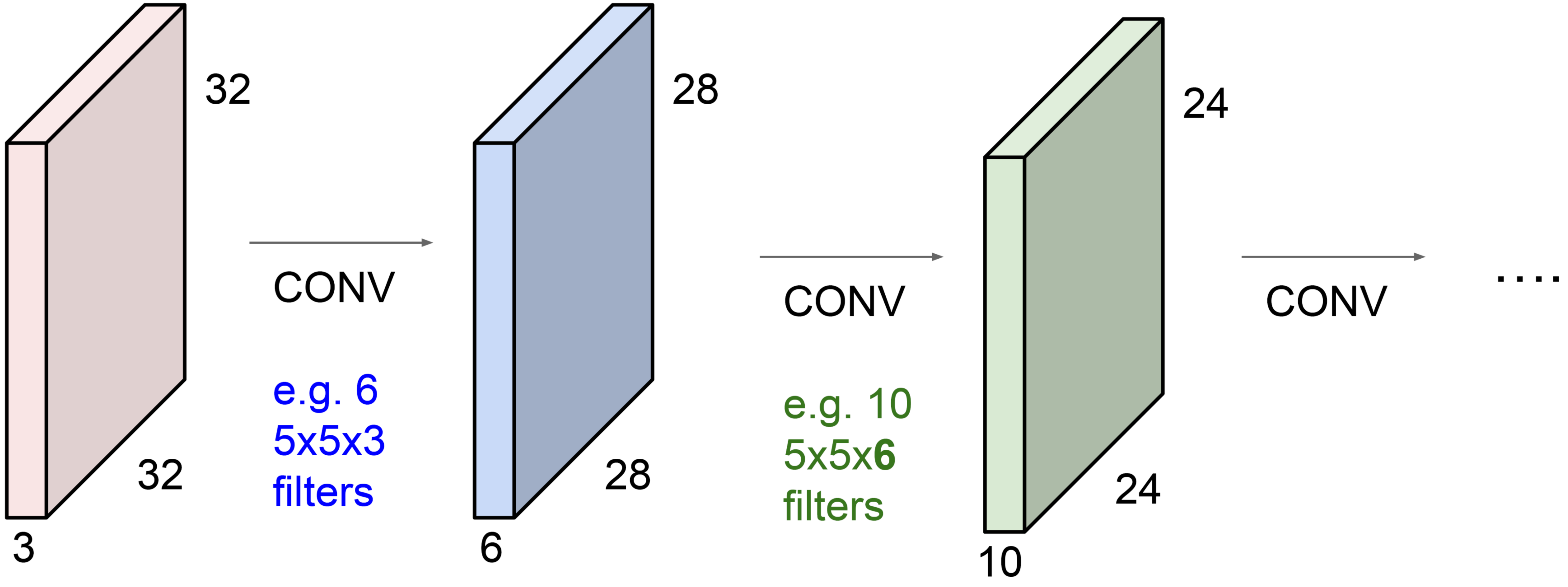
ConvNet is a sequence of Convolution Layers



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

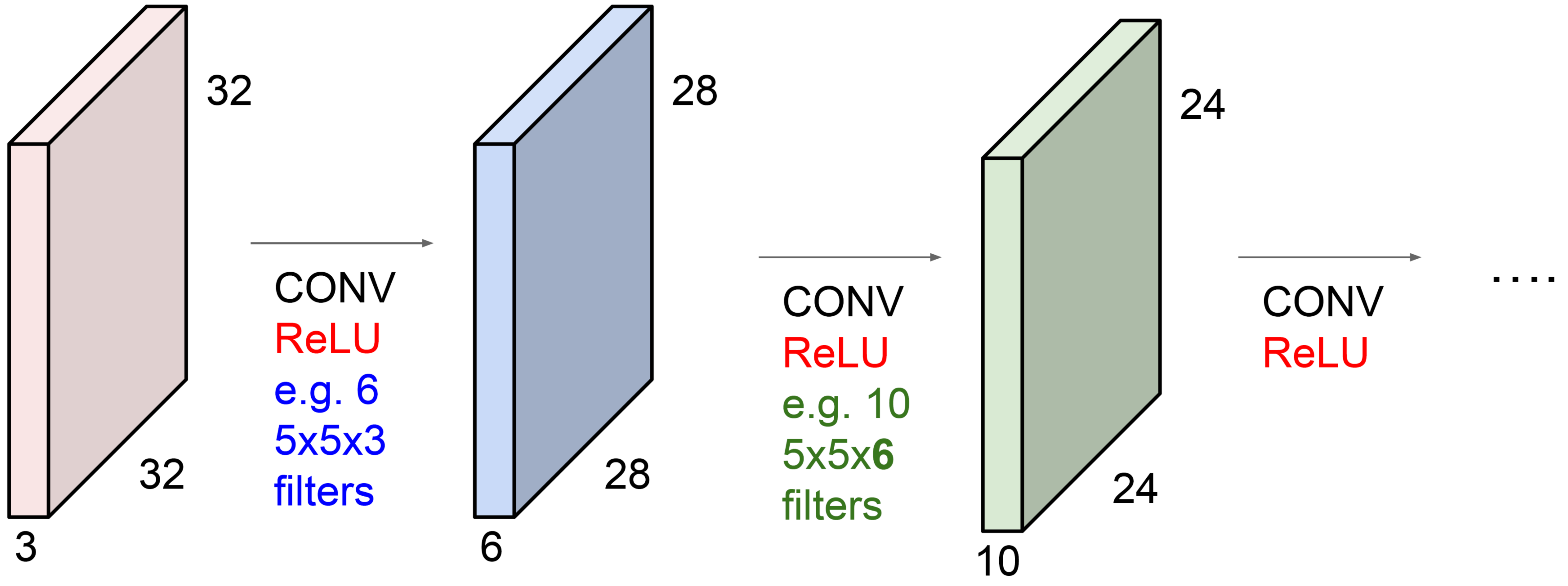
ConvNet is a sequence of Convolution Layers



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

ConvNet is a sequence of Convolution Layers, interspersed with activation functions



CONV
ReLU
e.g. 6
5x5x3
filters

CONV
ReLU
e.g. 10
5x5x**6**
filters

CONV
ReLU

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

What do convolutional filters learn?

Convolutional filters, also known as kernels or weights, learn to recognize local patterns or features in the input data. These patterns can be as simple as edges or corners, or as complex as object parts or textures. During training, the filters are initialized with random values, and their weights are adjusted based on the error between the predicted output and the true output. As the network is trained, the filters learn to recognize different patterns in the input data that are relevant to the task at hand.

The specific patterns that a filter learns to recognize depend on the structure and complexity of the input data, as well as the objective of the network. For example, in an image recognition task, early convolutional filters might learn to recognize basic features such as edges, lines, and corners. As the network becomes deeper, the filters might learn to recognize more complex patterns such as object parts or textures.

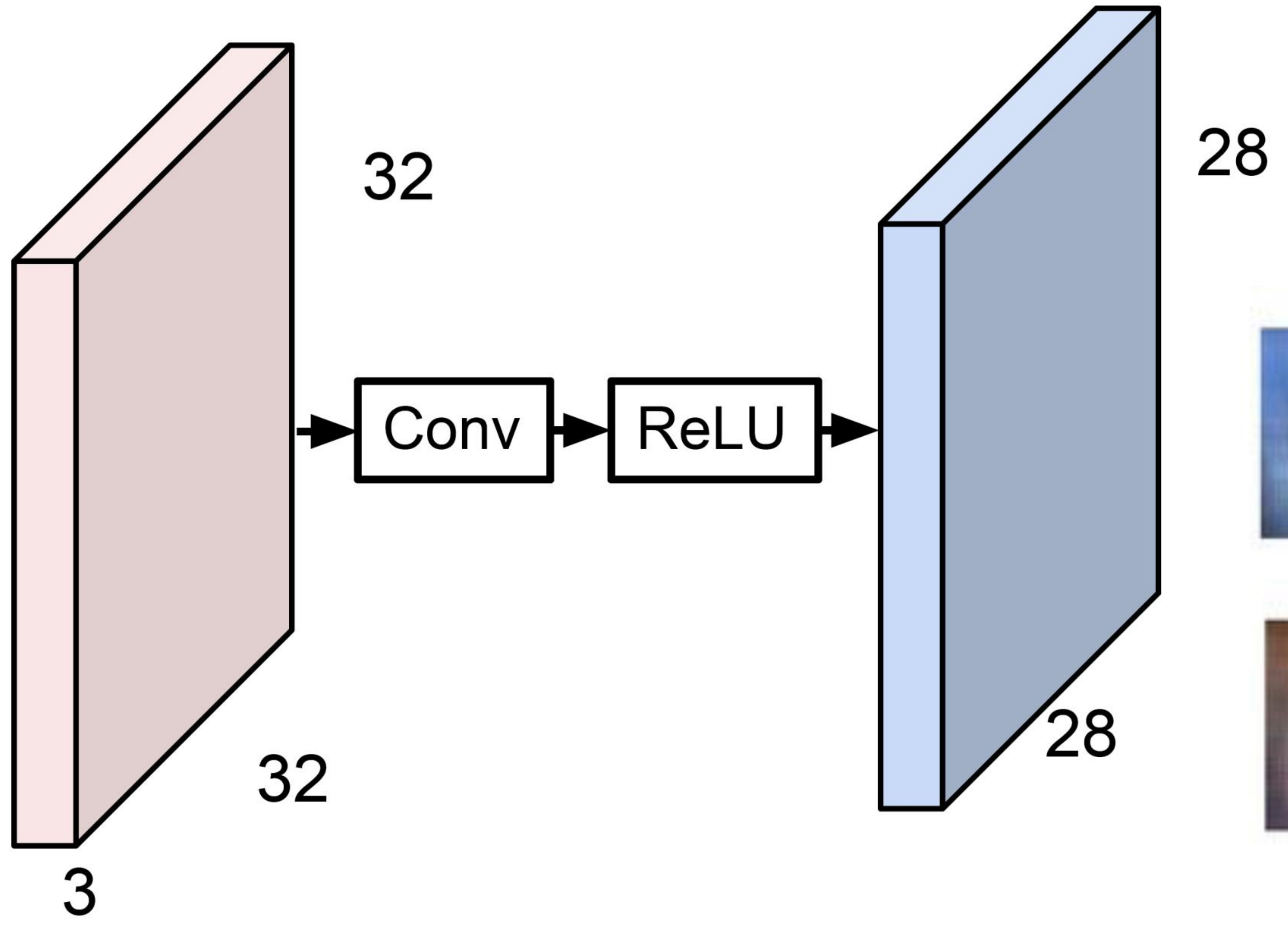
It is important to note that filters in a convolutional layer are designed to be translation-invariant, meaning that they can recognize the same pattern regardless of where it appears in the input data. This is achieved by sharing the same set of weights across different regions of the input.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

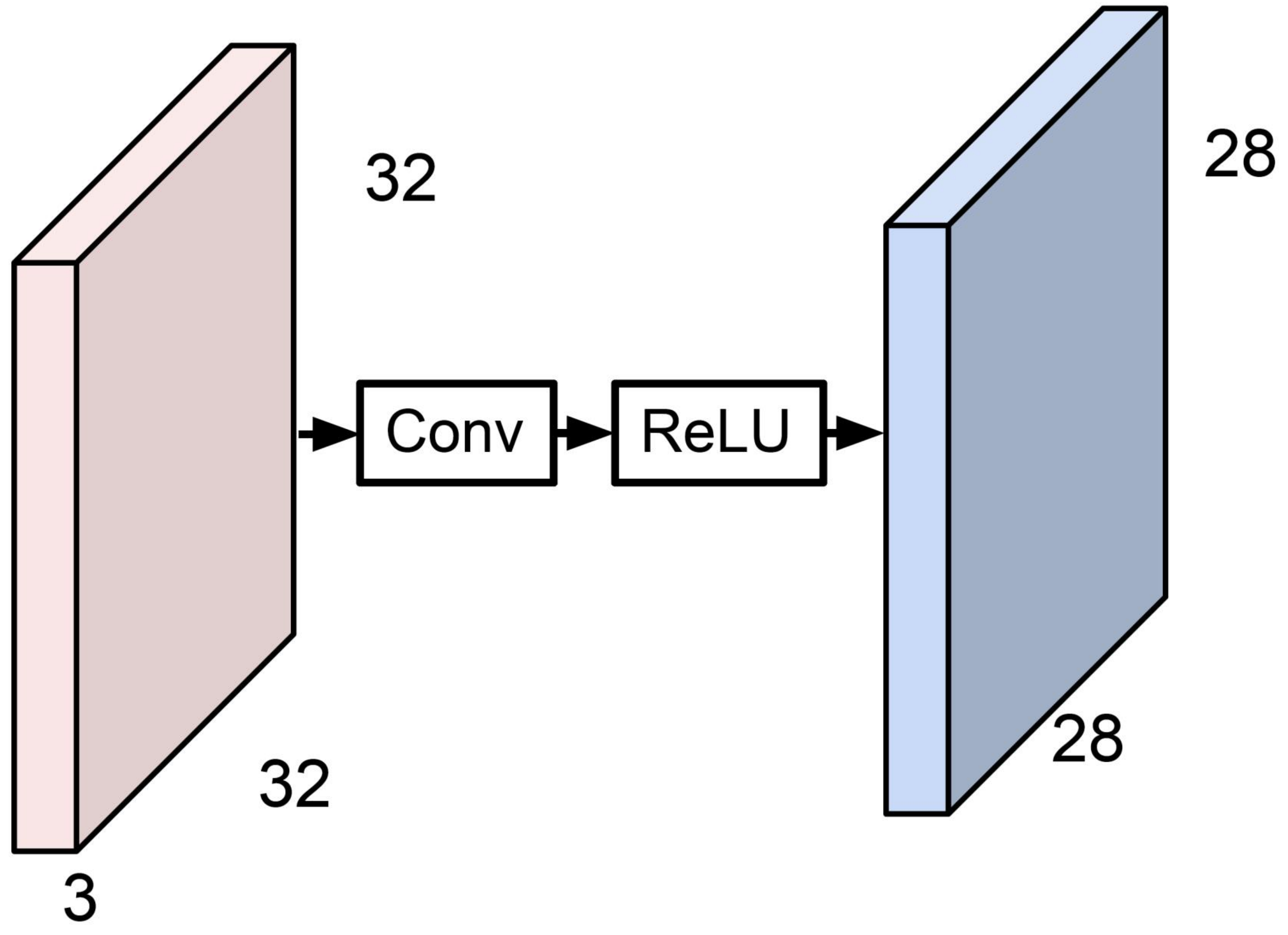
What do convolutional filters learn?

Overall, convolutional filters learn to extract relevant features from the input data, which can be used to make accurate predictions or classifications. By stacking multiple convolutional layers, a deep learning model can learn increasingly complex representations of the input data, which enables it to achieve state-of-the-art performance on a wide range of image processing tasks.



What do convolutional filters learn?

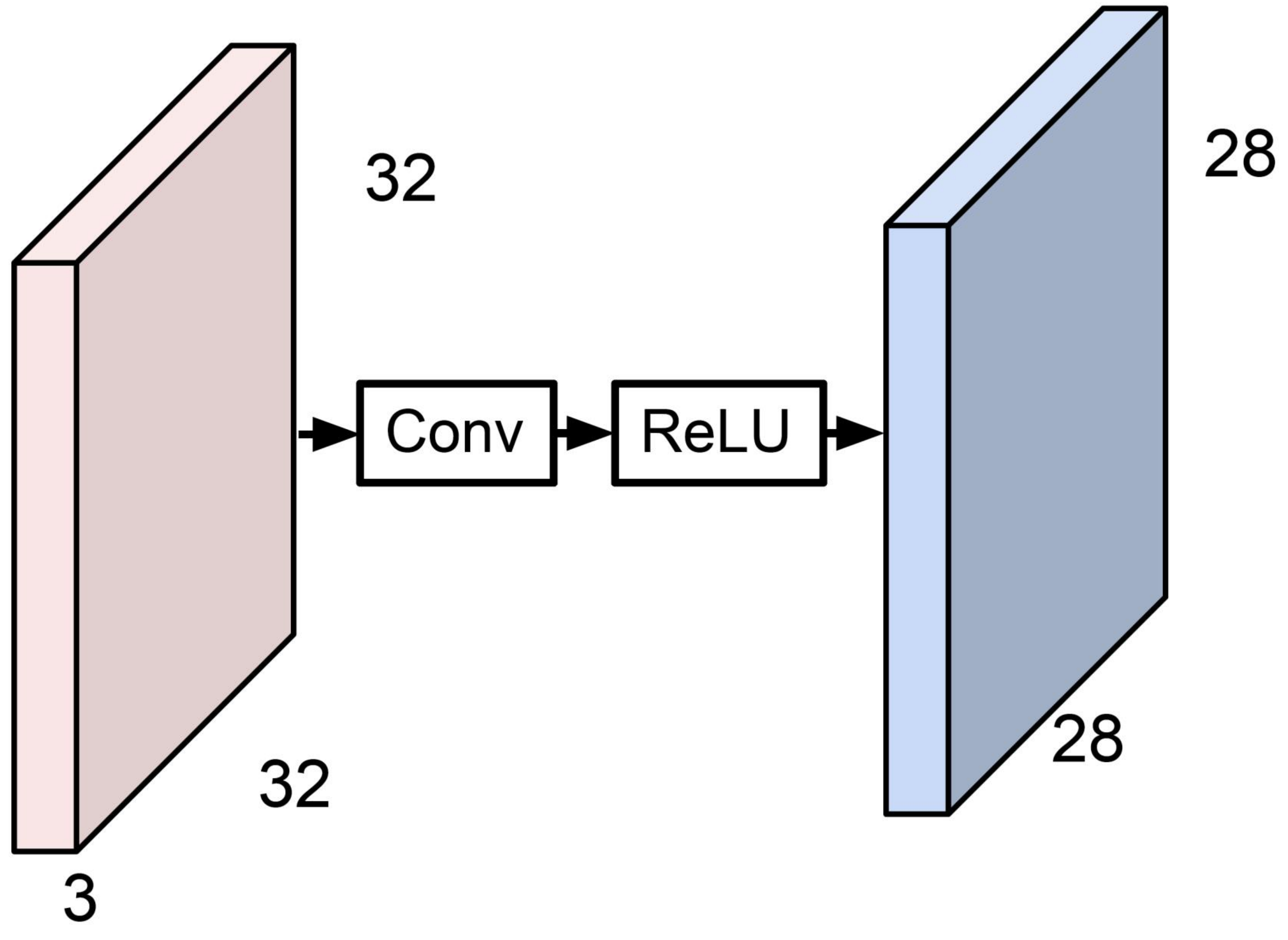
Overall, convolutional filters learn to extract relevant features from the input data, which can be used to make accurate predictions or classifications. By stacking multiple convolutional layers, a deep learning model can learn increasingly complex representations of the input data, which enables it to achieve state-of-the-art performance on a wide range of image processing tasks.



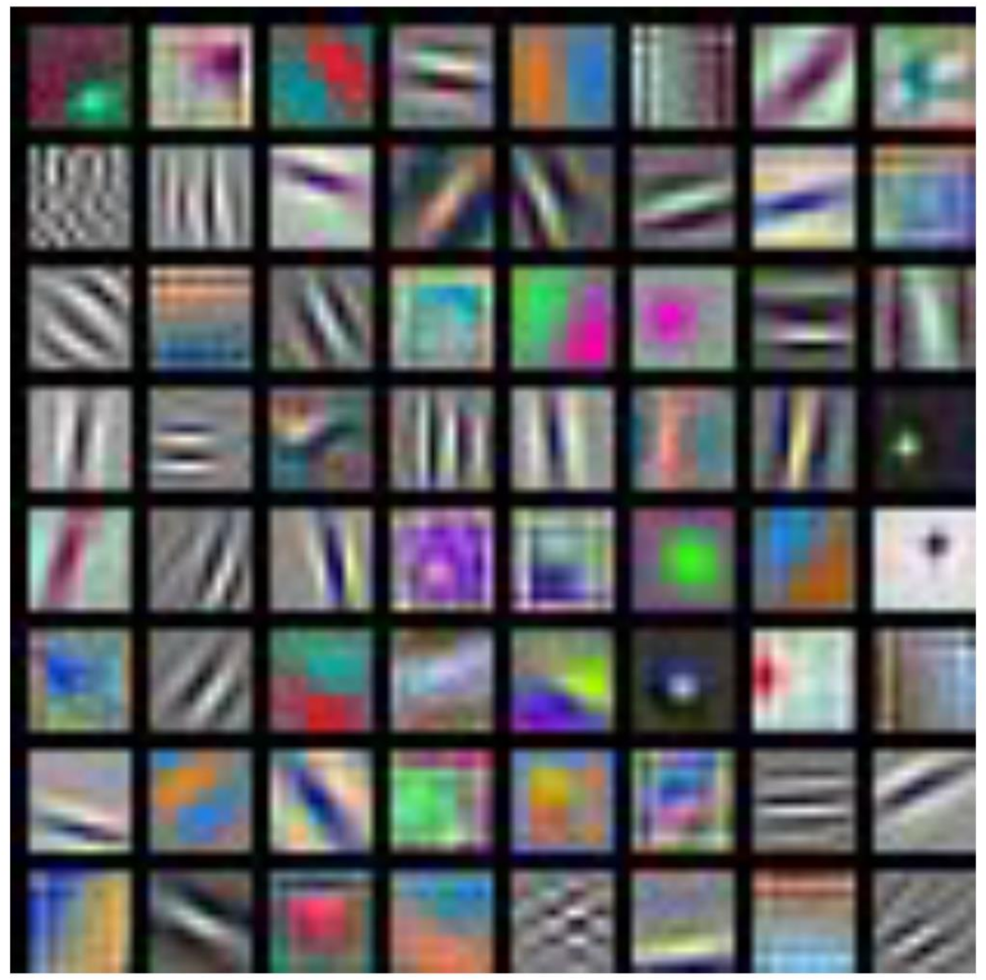
MLP: Bank of whole-image templates

What do convolutional filters learn?

Overall, convolutional filters learn to extract relevant features from the input data, which can be used to make accurate predictions or classifications. By stacking multiple convolutional layers, a deep learning model can learn increasingly complex representations of the input data, which enables it to achieve state-of-the-art performance on a wide range of image processing tasks.



First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11

What do convolutional filters learn?

In a convolutional neural network, a convolution layer works by applying a set of learnable filters to the input data, producing a set of activation maps that represent different features of the input.

Each filter is a small matrix of weights that is convolved (i.e., element-wise multiplied and summed) with a small region of the input data, creating a single value in the output feature map. The filter is then moved across the input data, applying the same computation at every position, to create a complete output feature map.

The number of filters in the convolutional layer determines the number of activation maps in the output. Each filter produces a separate activation map, which represents a different aspect or feature of the input. For example, in an image recognition task, one filter might detect horizontal edges, while another might detect vertical edges.

The size of the output feature maps depends on the size of the input data, the size of the filters, and the stride of the convolution operation (i.e., how much the filter is shifted across the input at each step). By adjusting these parameters, it is possible to control the size and resolution of the output feature maps.

What do convolutional filters learn?

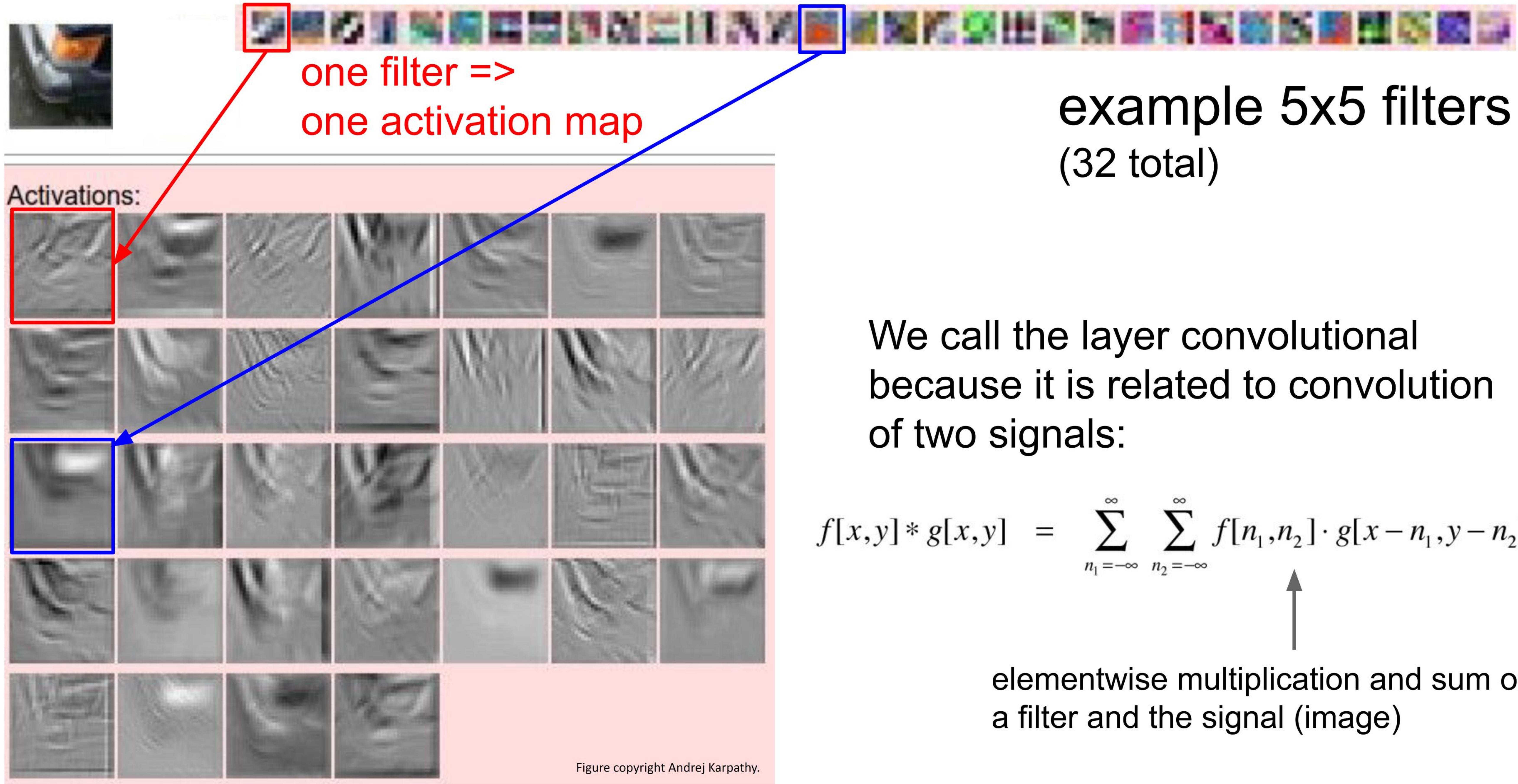


Figure copyright Andrej Karpathy.

example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y]*g[x,y] = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2]\cdot g[x-n_1,y-n_2]$$

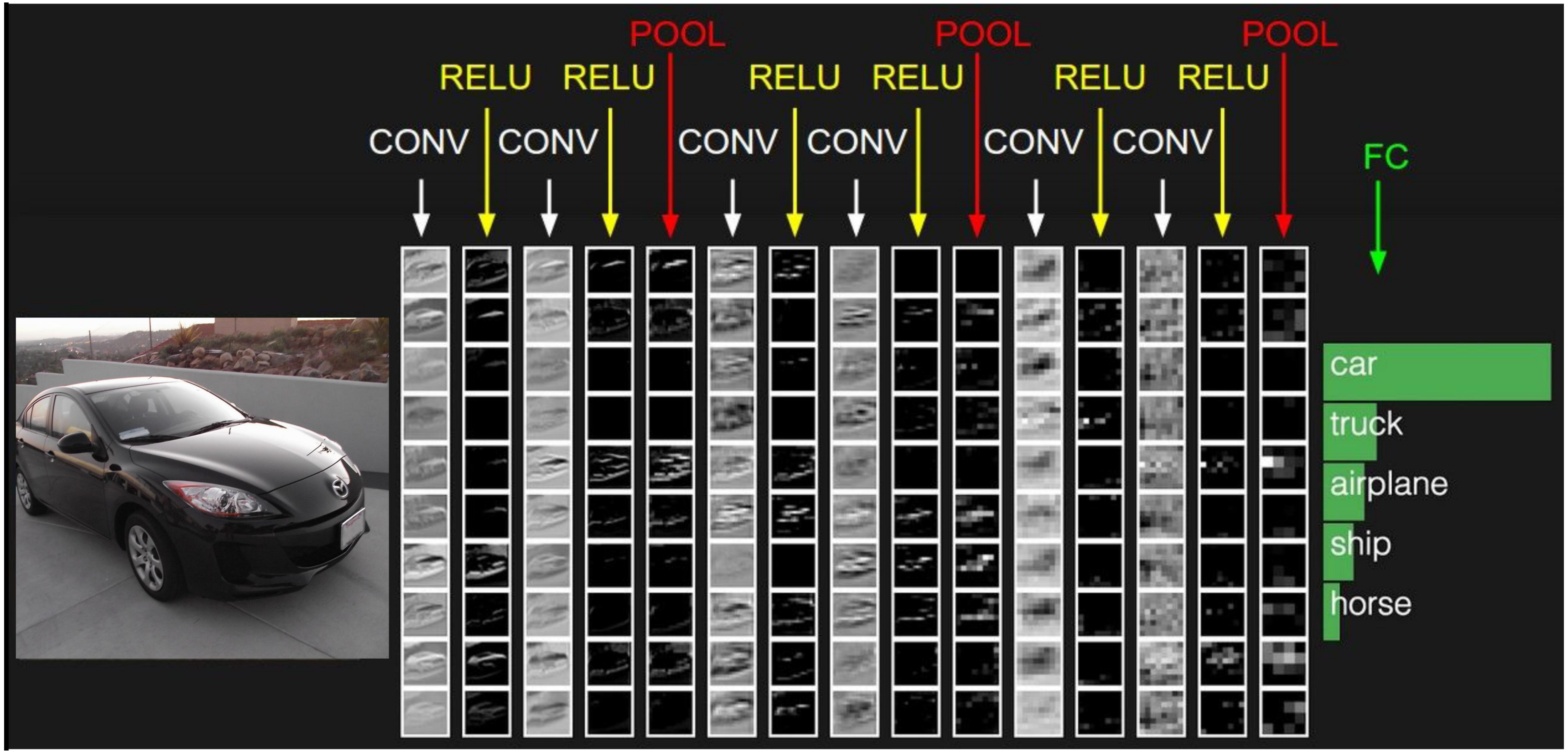
elementwise multiplication and sum of a filter and the signal (image)

It is true that each filter requires one activation map as output, as each filter produces a single feature map.

A convolutional layer can have multiple filters, each producing its own feature map.

In practice, modern convolutional neural networks typically contain many convolutional layers, with hundreds or thousands of filters in each layer, producing a large number of feature maps that are used to extract increasingly complex representations of the input data.

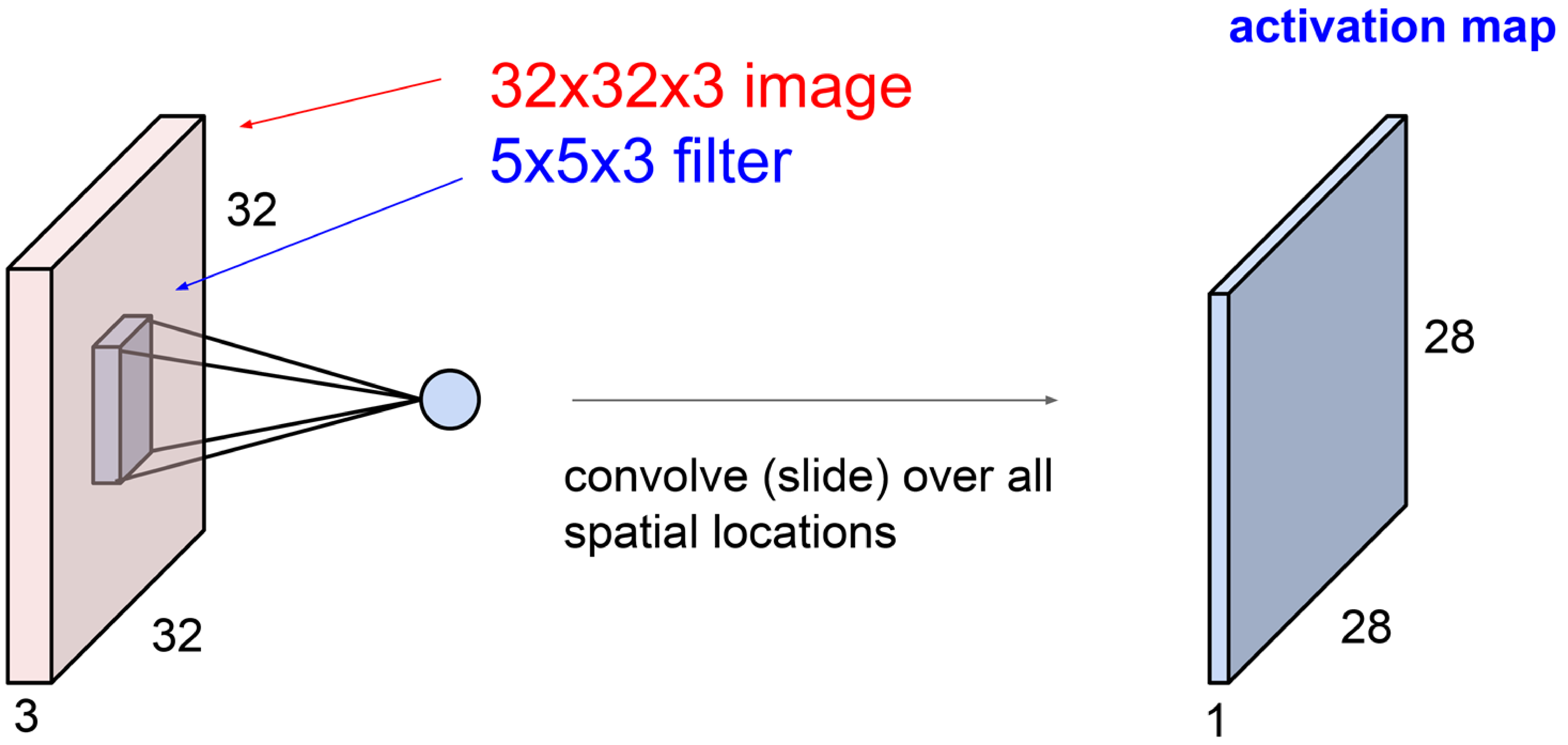
What do convolutional filters learn?



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

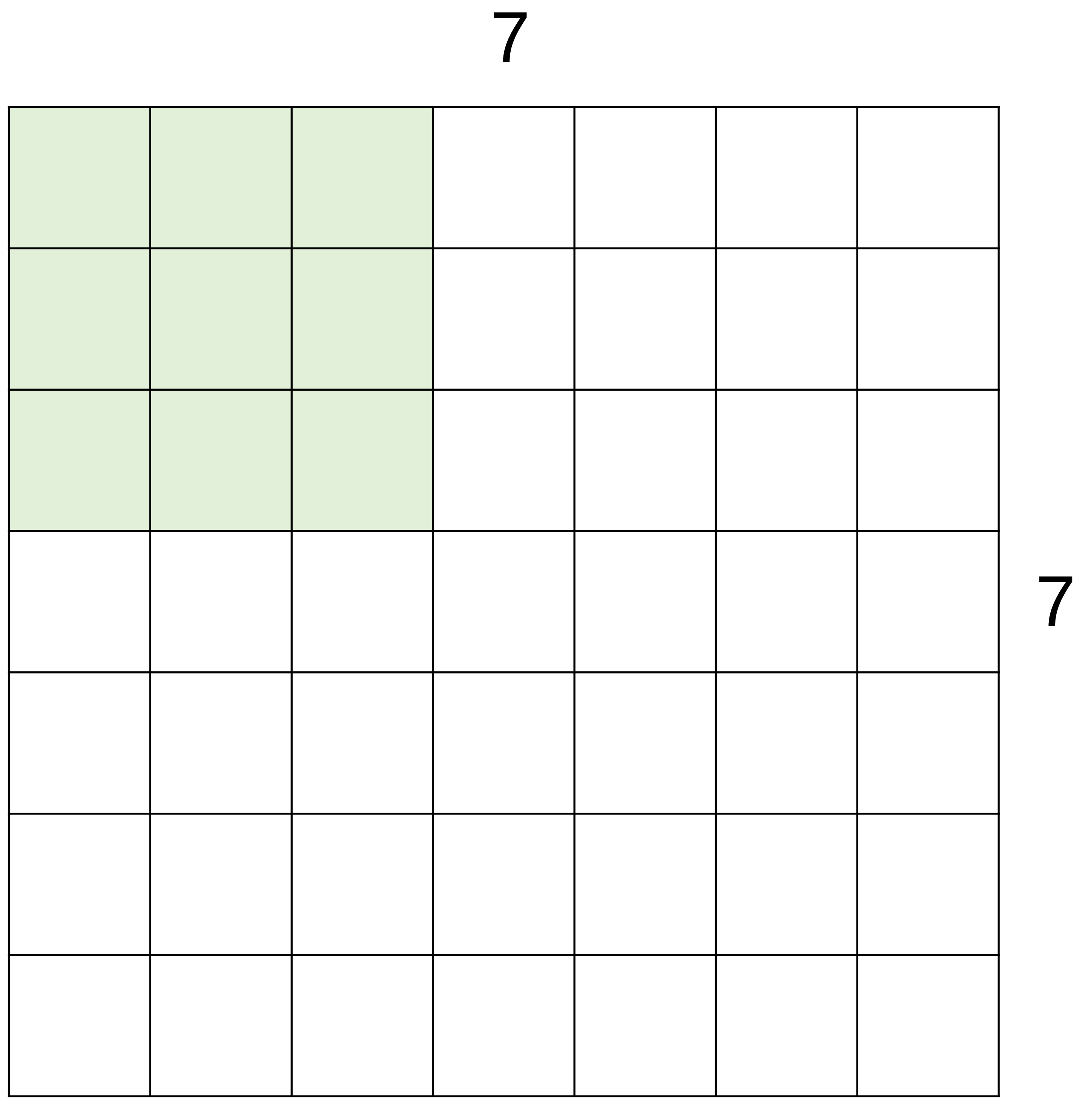
Convolutional Neural Networks: *A closer look at spatial dimensions*



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *A closer look at spatial dimensions*

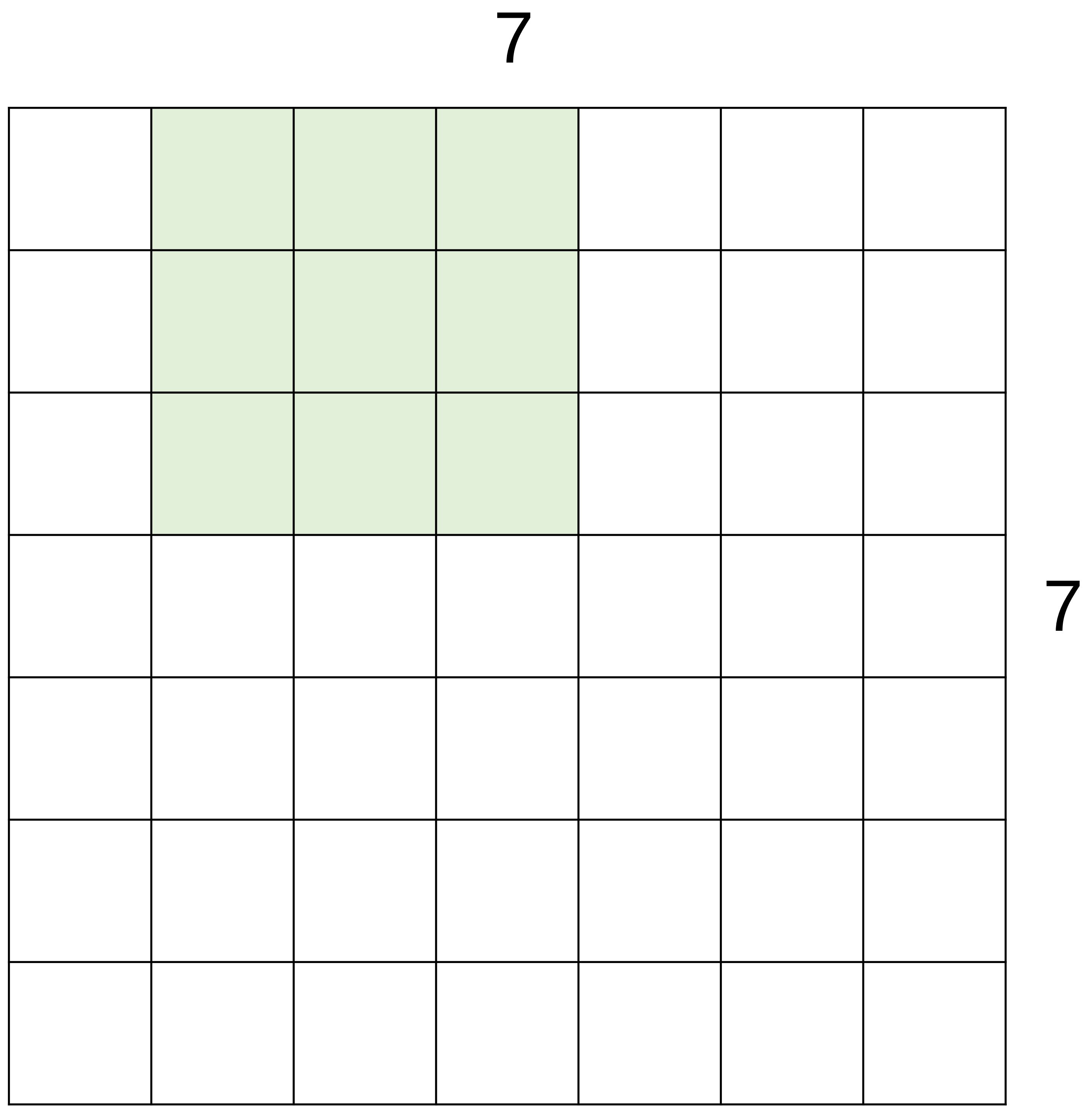


7x7 input (spatially)
assume 3x3 filter

Co-financed by the European Union
Connecting Europe Facility

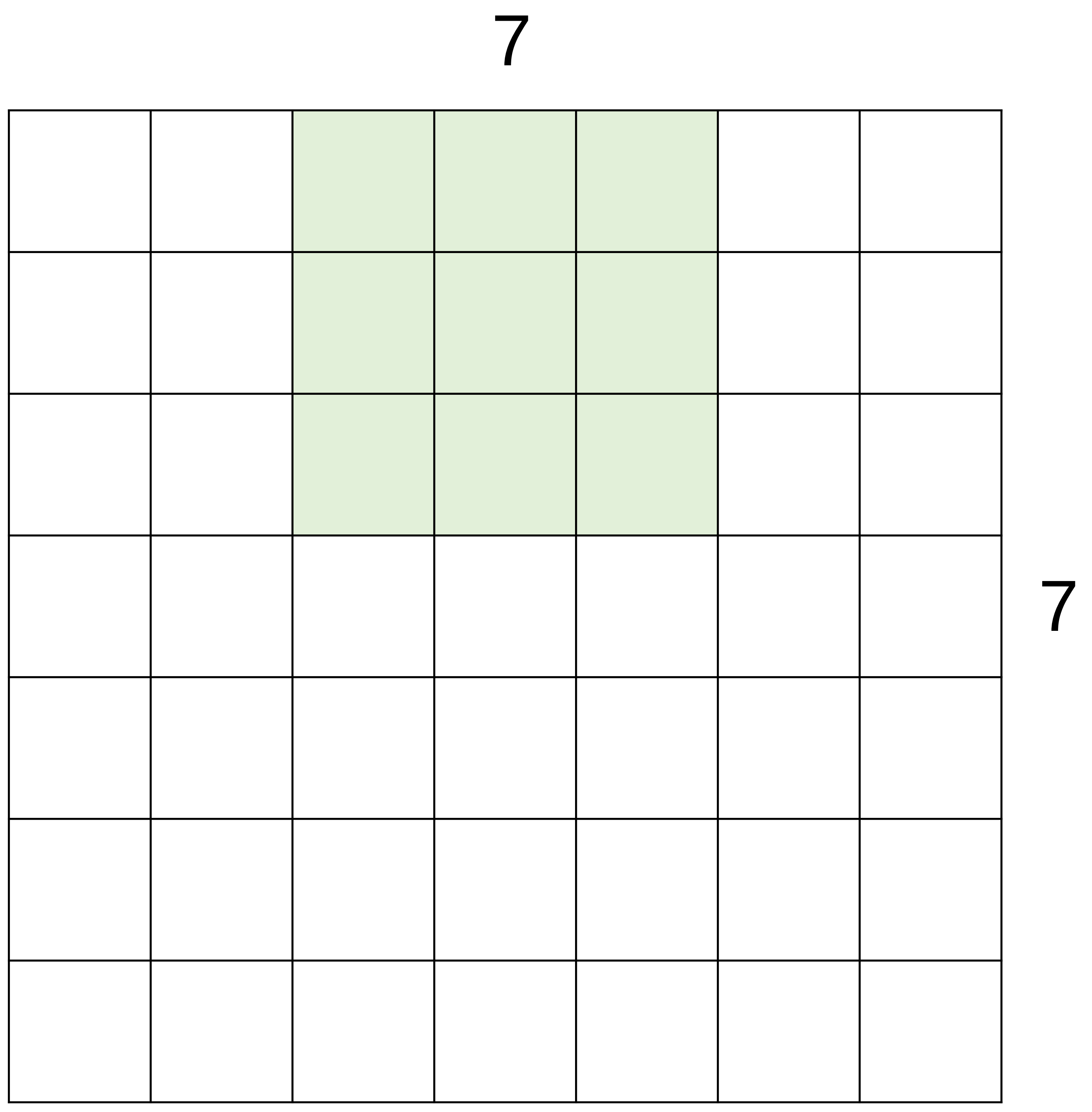
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *A closer look at spatial dimensions*



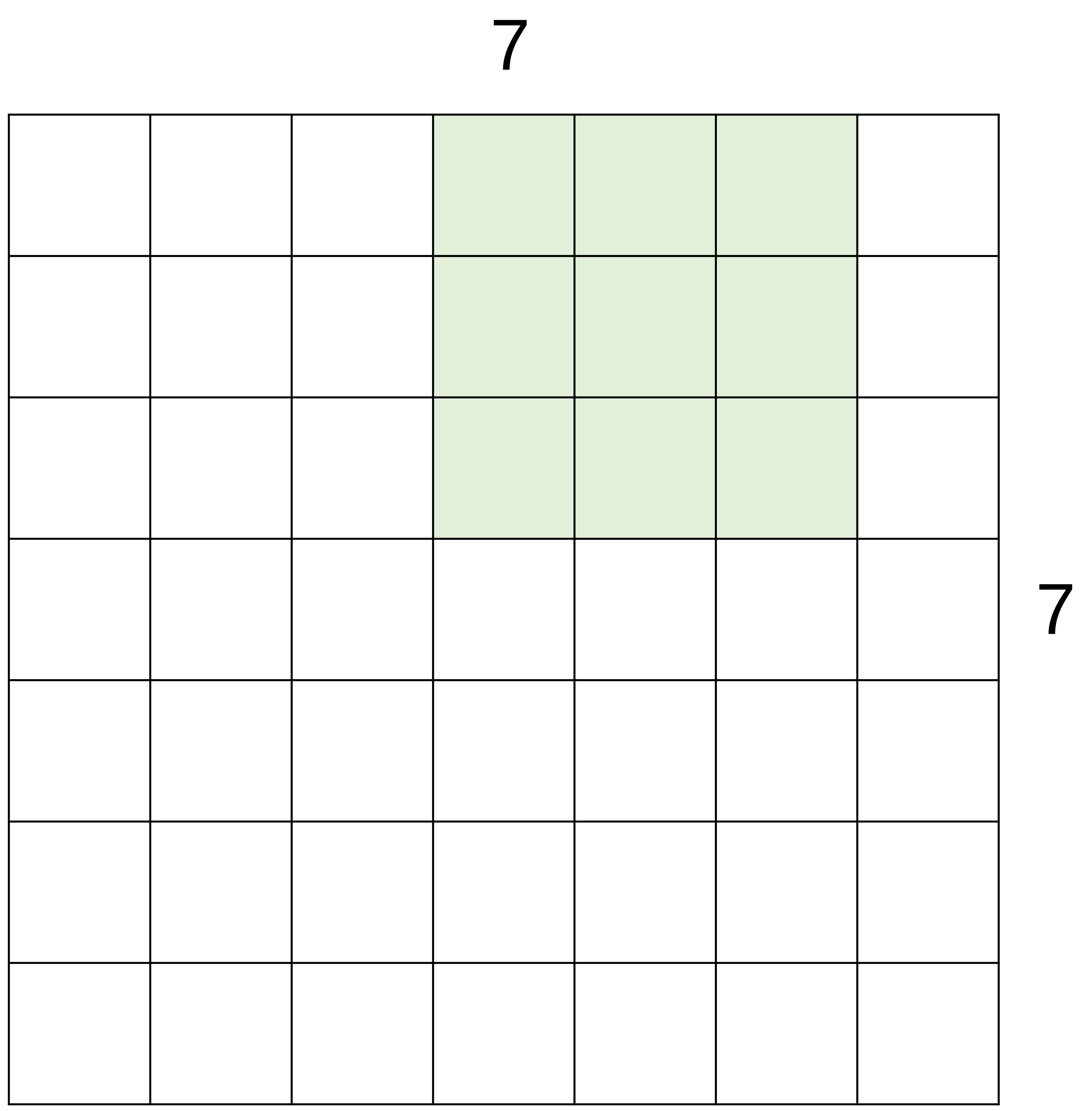
7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks: *A closer look at spatial dimensions*



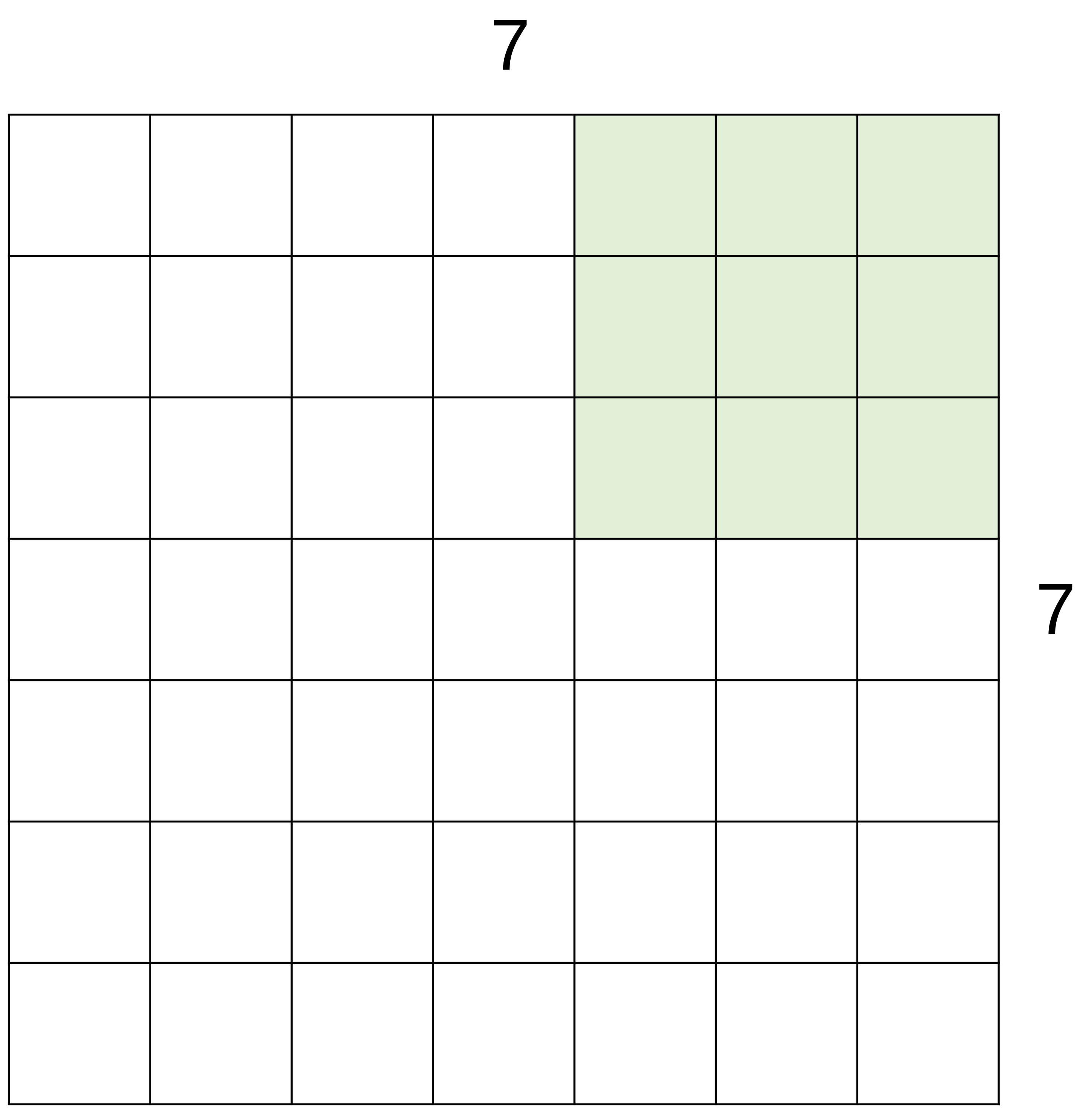
7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

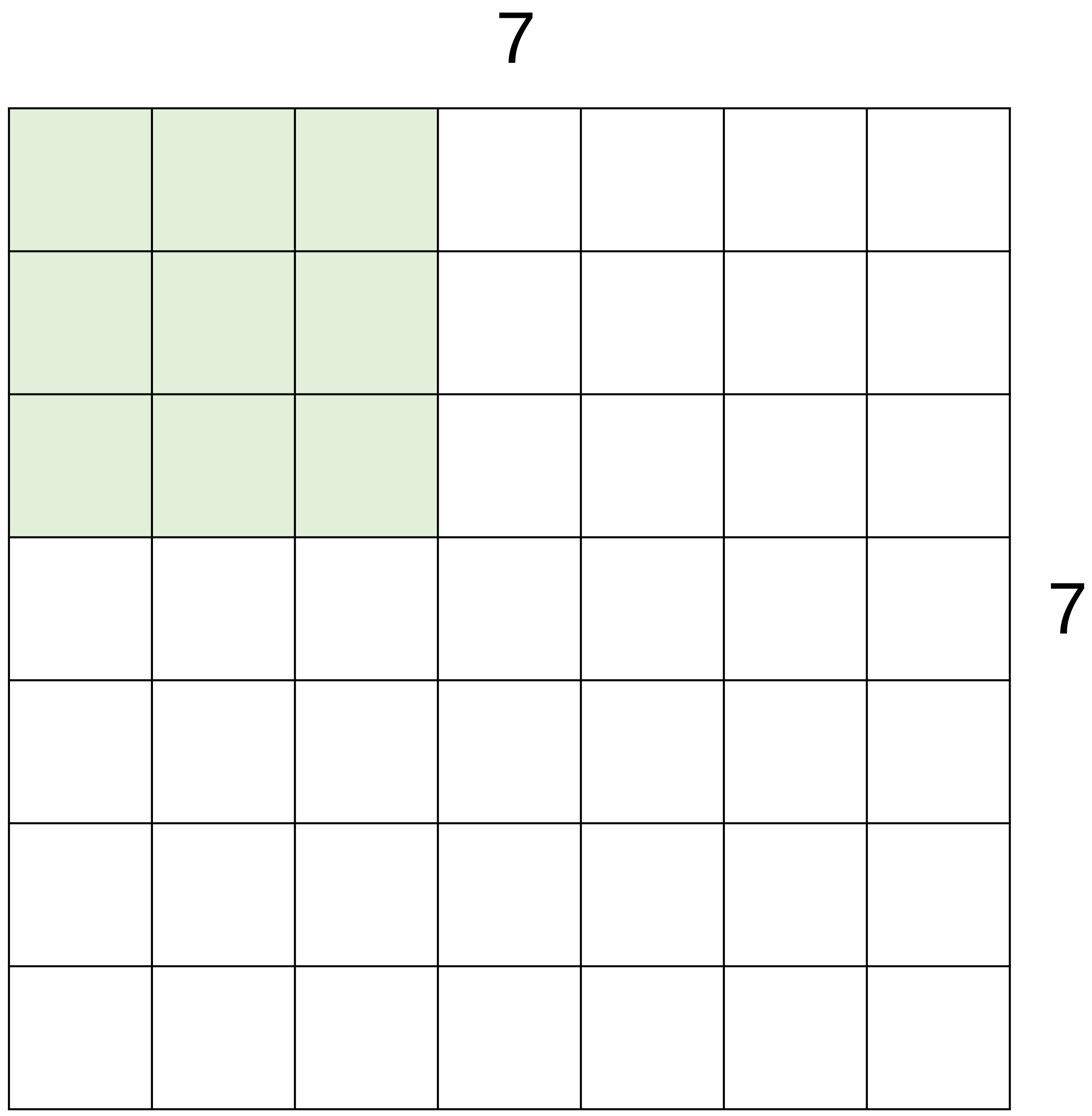
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

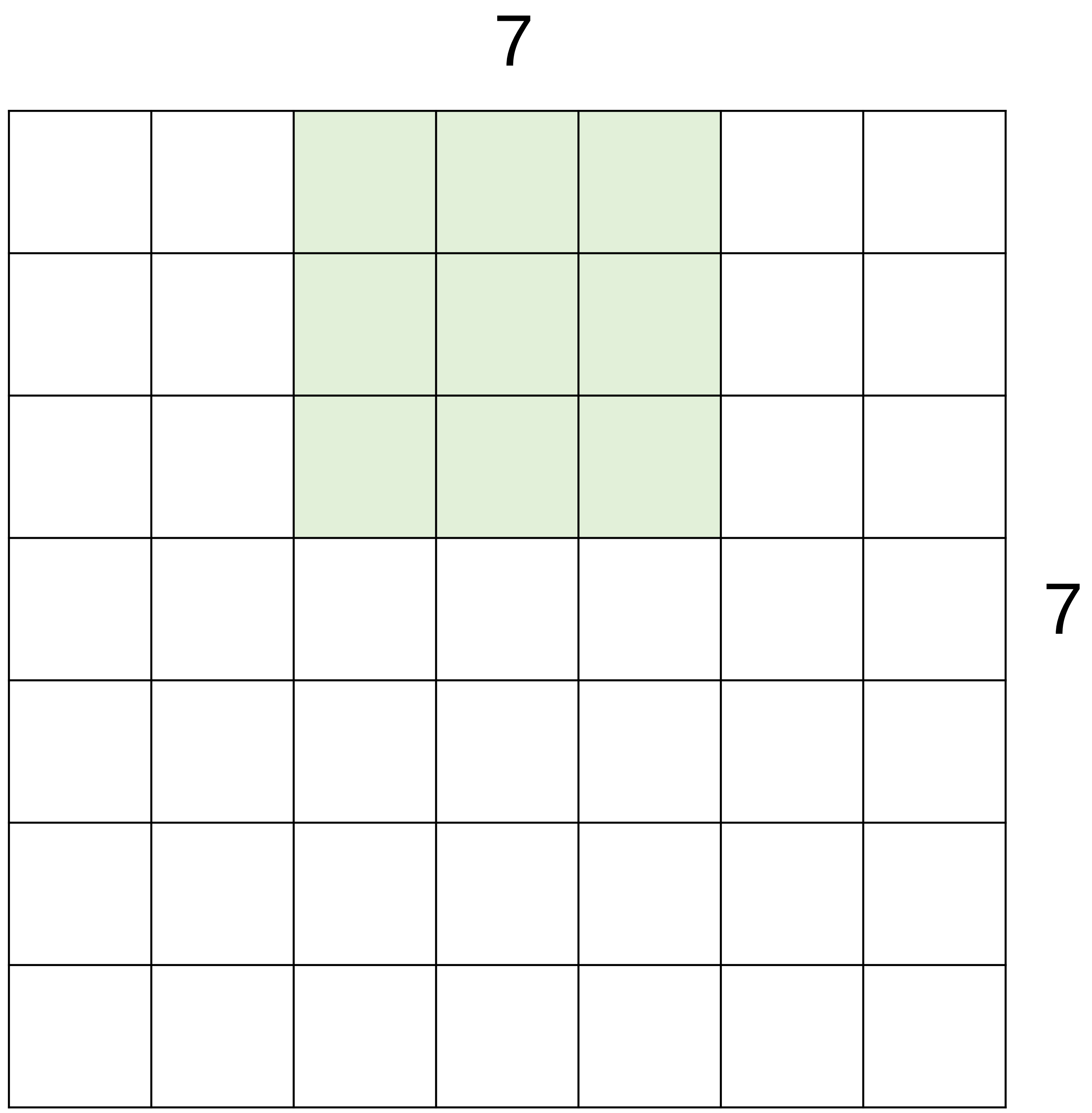
=> 5x5 output

Convolutional Neural Networks: *A closer look at spatial dimensions*



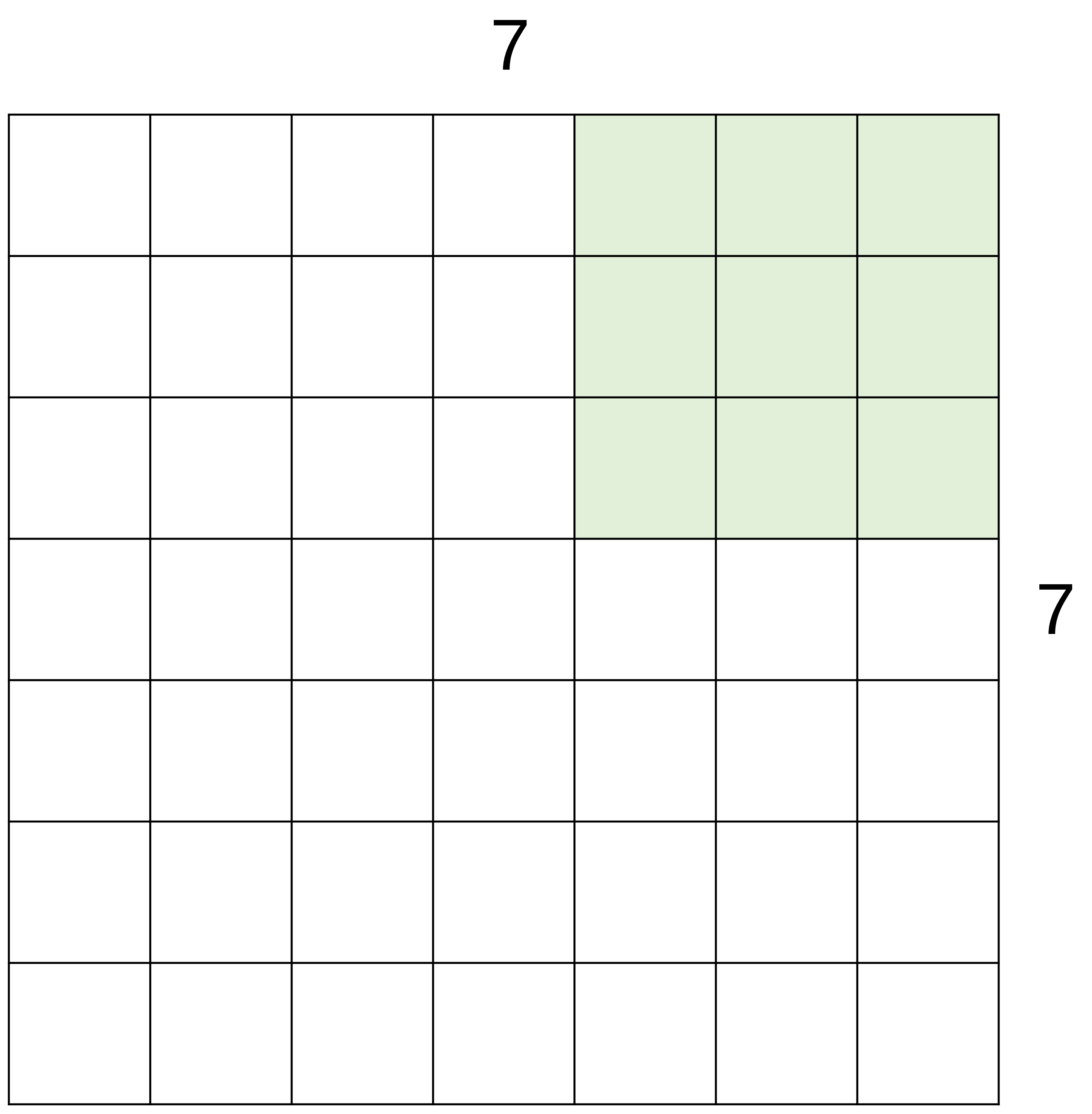
7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

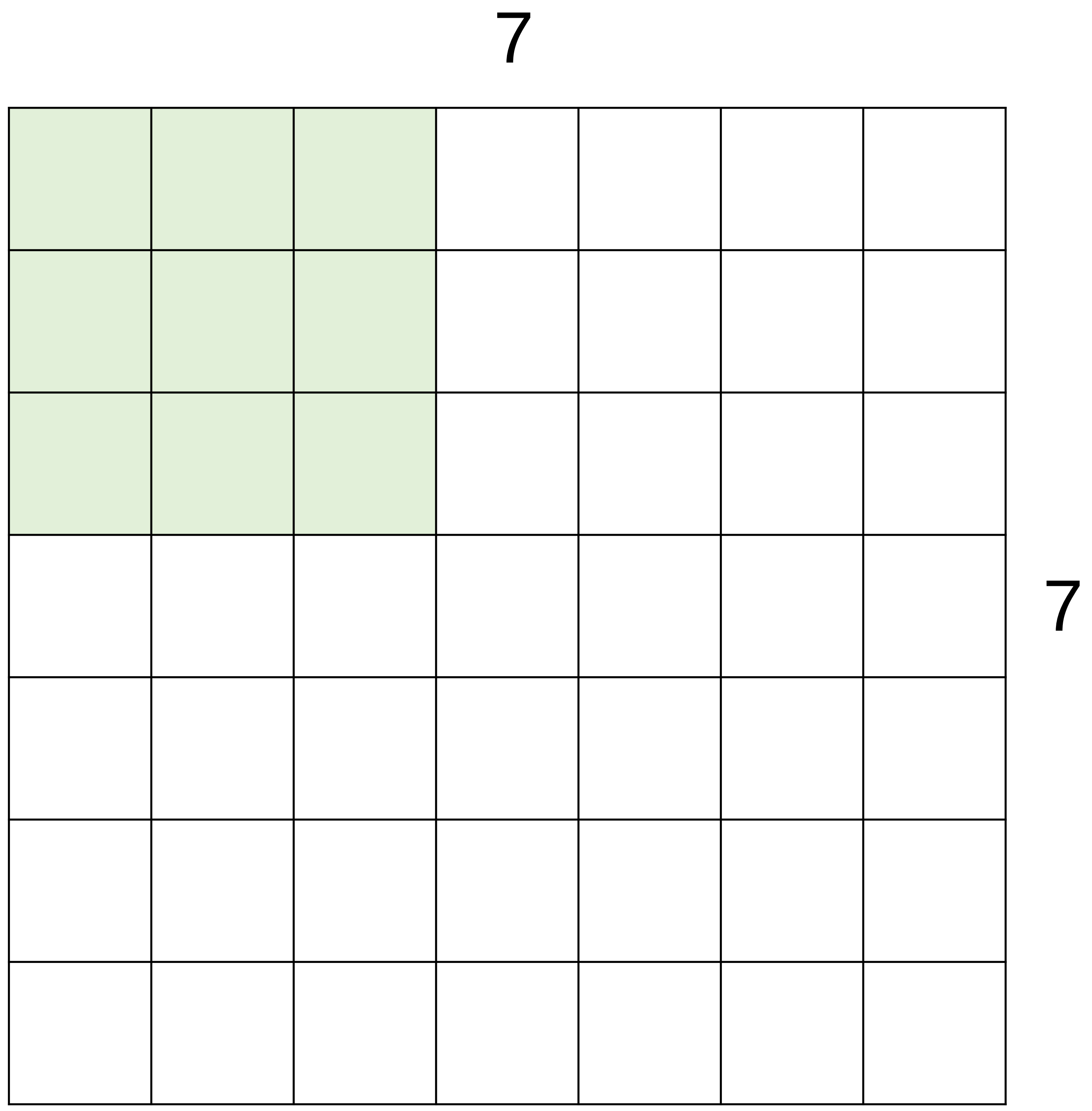
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

=> 3x3 output!

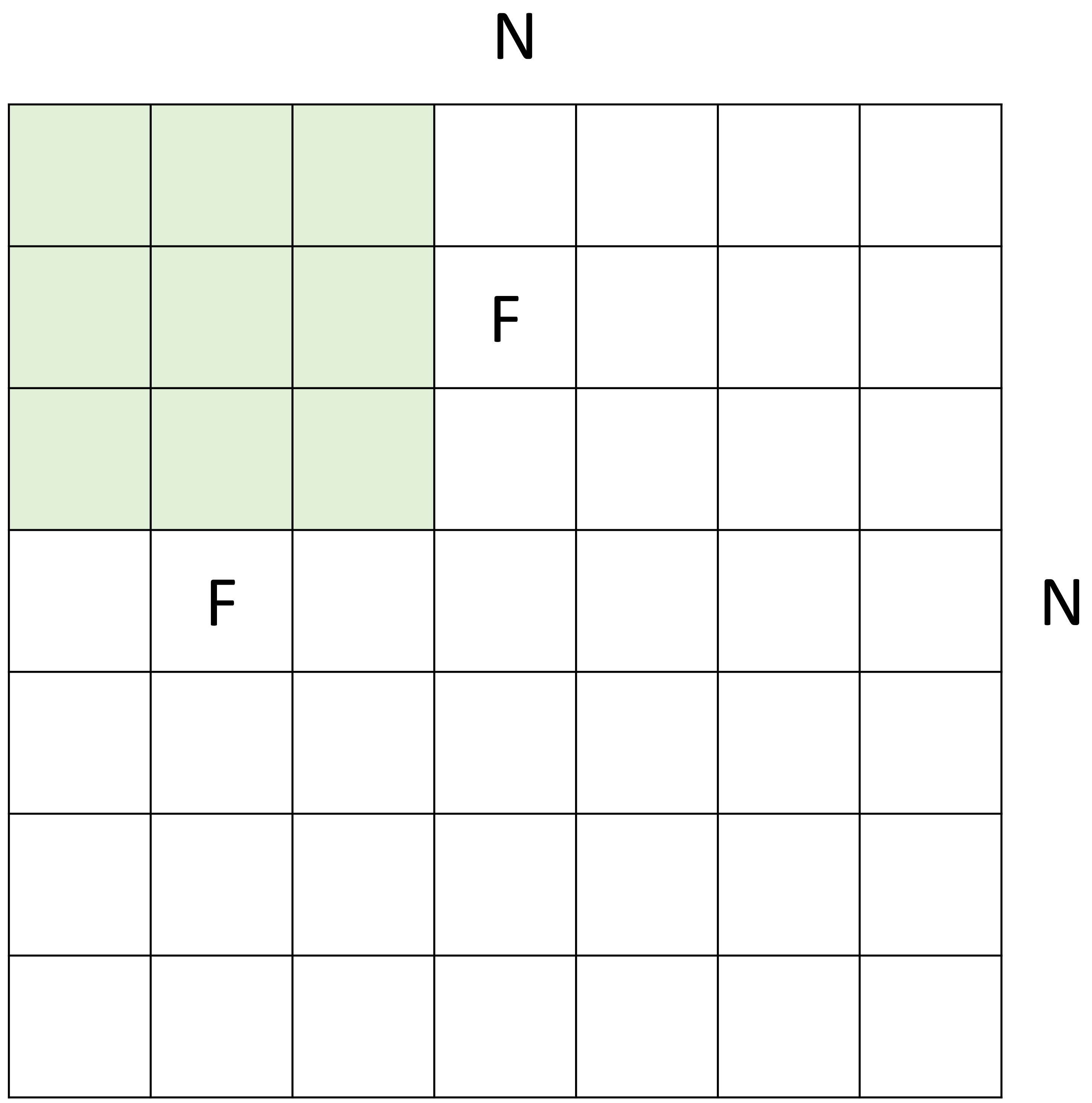
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

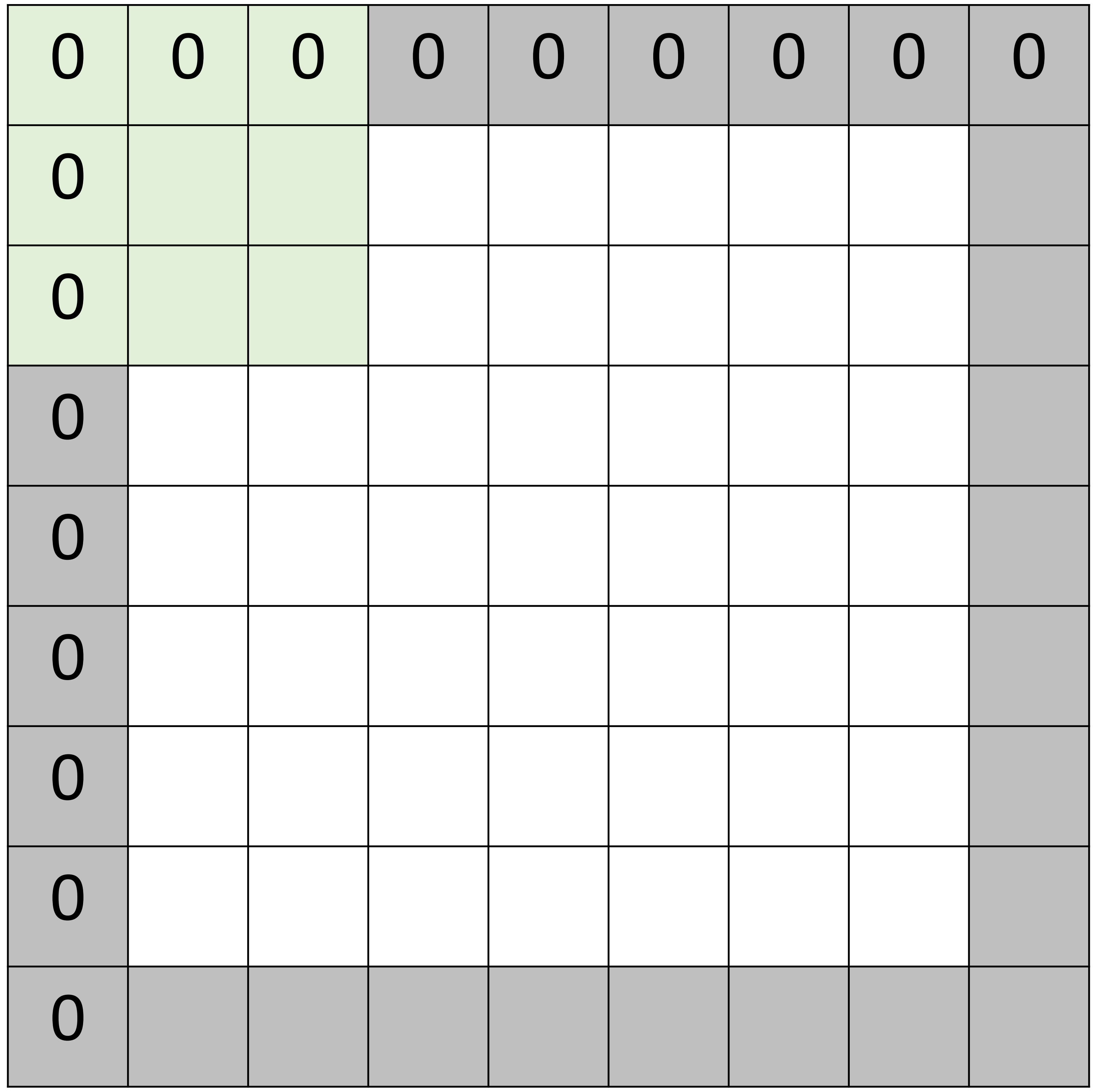
doesn’t fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolutional Neural Networks: *A closer look at spatial dimensions*



Output size: (N - F) / stride + 1
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 ✗

Convolutional Neural Networks: *Common to zero pad the border*



0	0	0	0	0	0	0	0	0
0								
0								
0								
0								
0								
0								
0								
0								

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

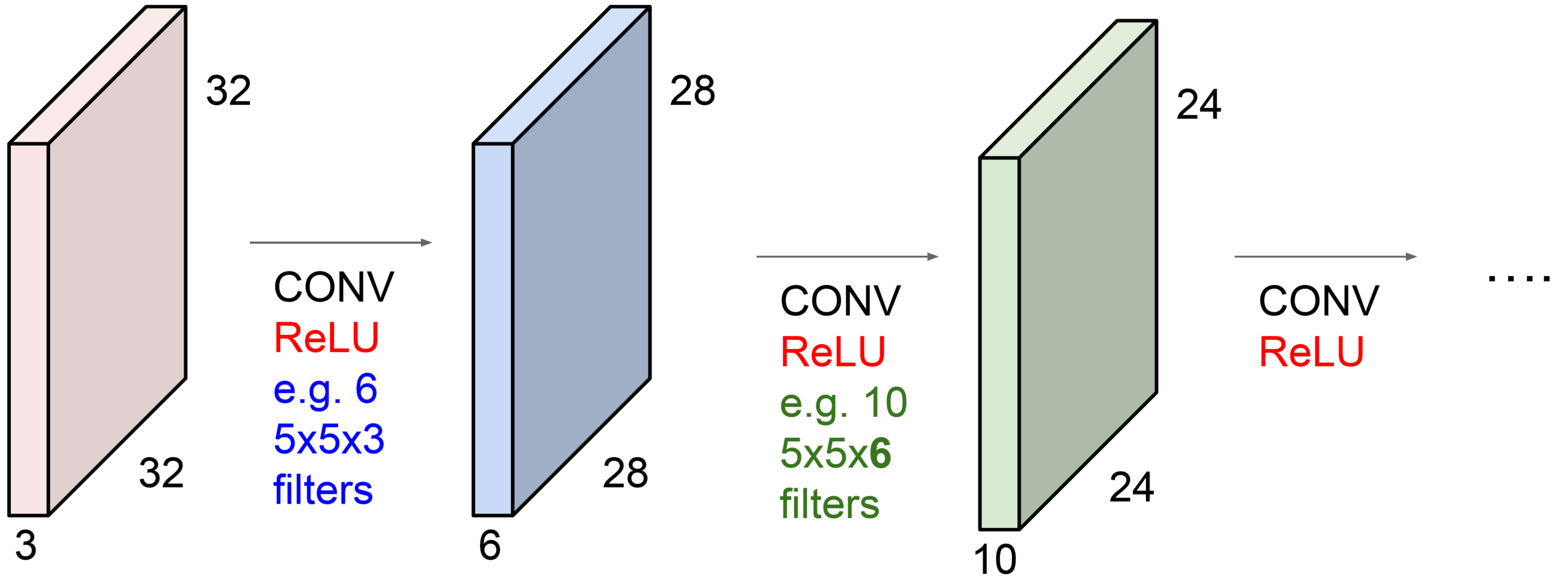
7x7 output!

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

(recall:)
(N - F) / stride + 1

ConvNet

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

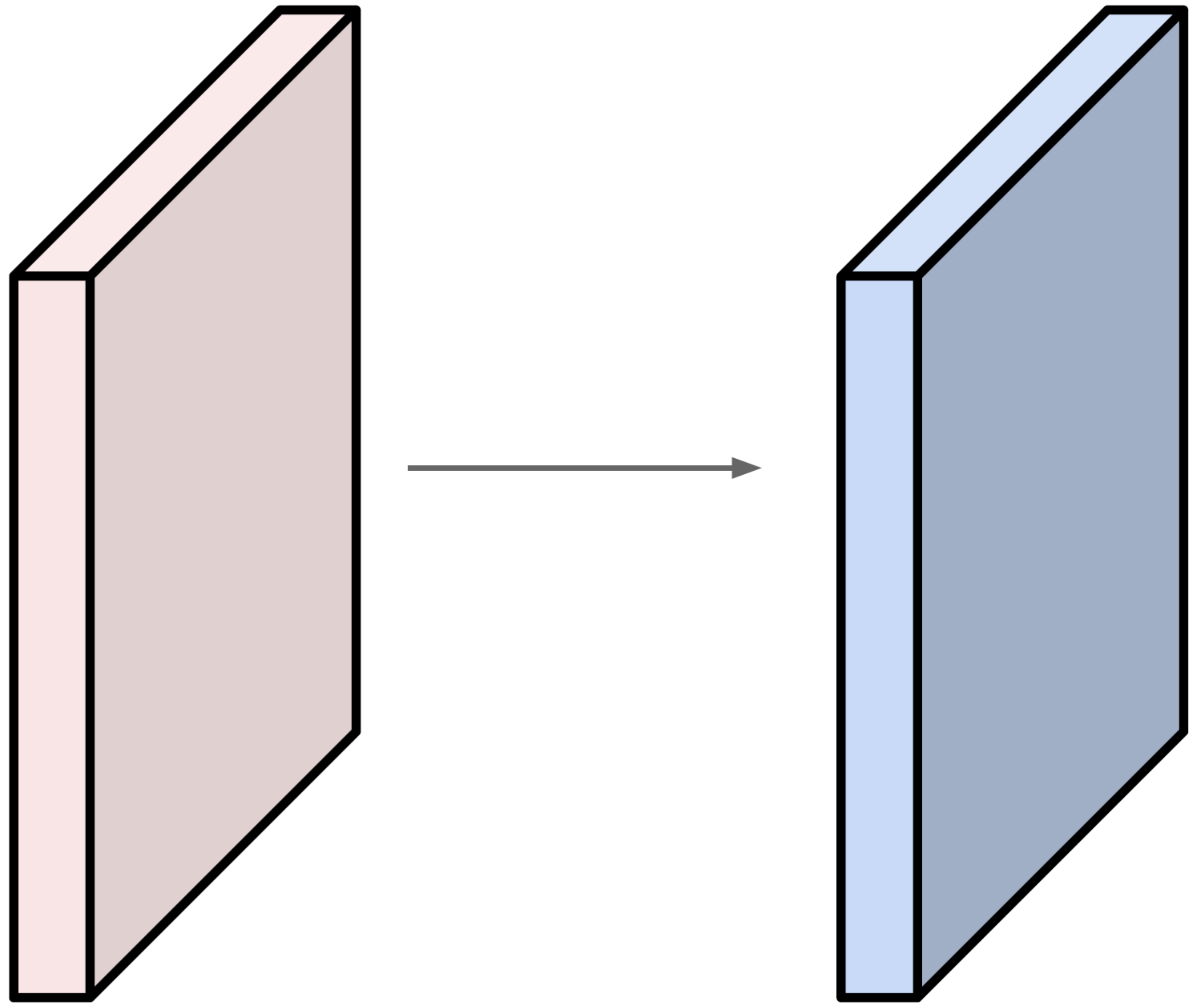


ConvNet: *Example*

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

What is the output volume size: ?



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

ConvNet: *Example*

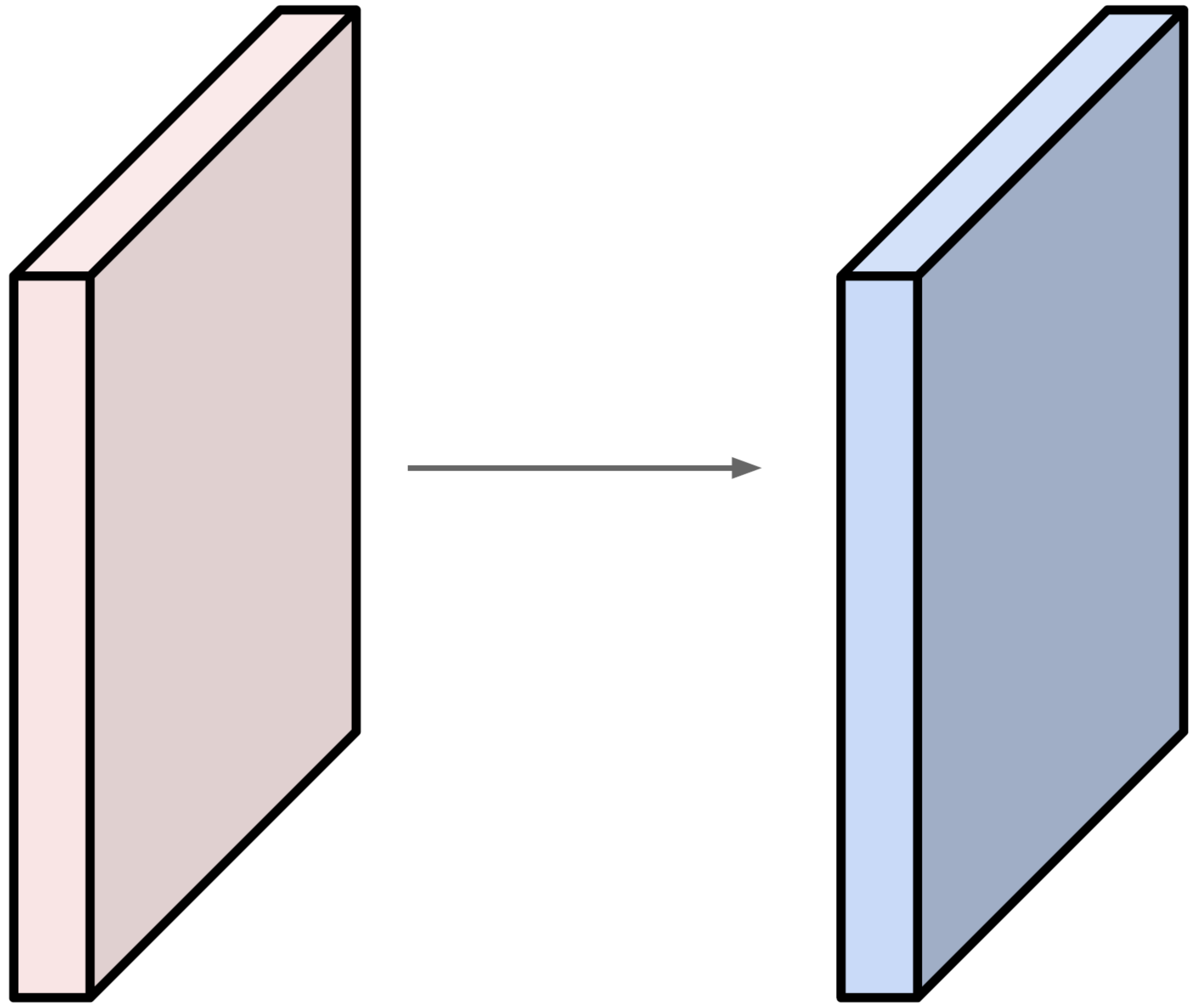
Input volume: **32x32**x3

10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

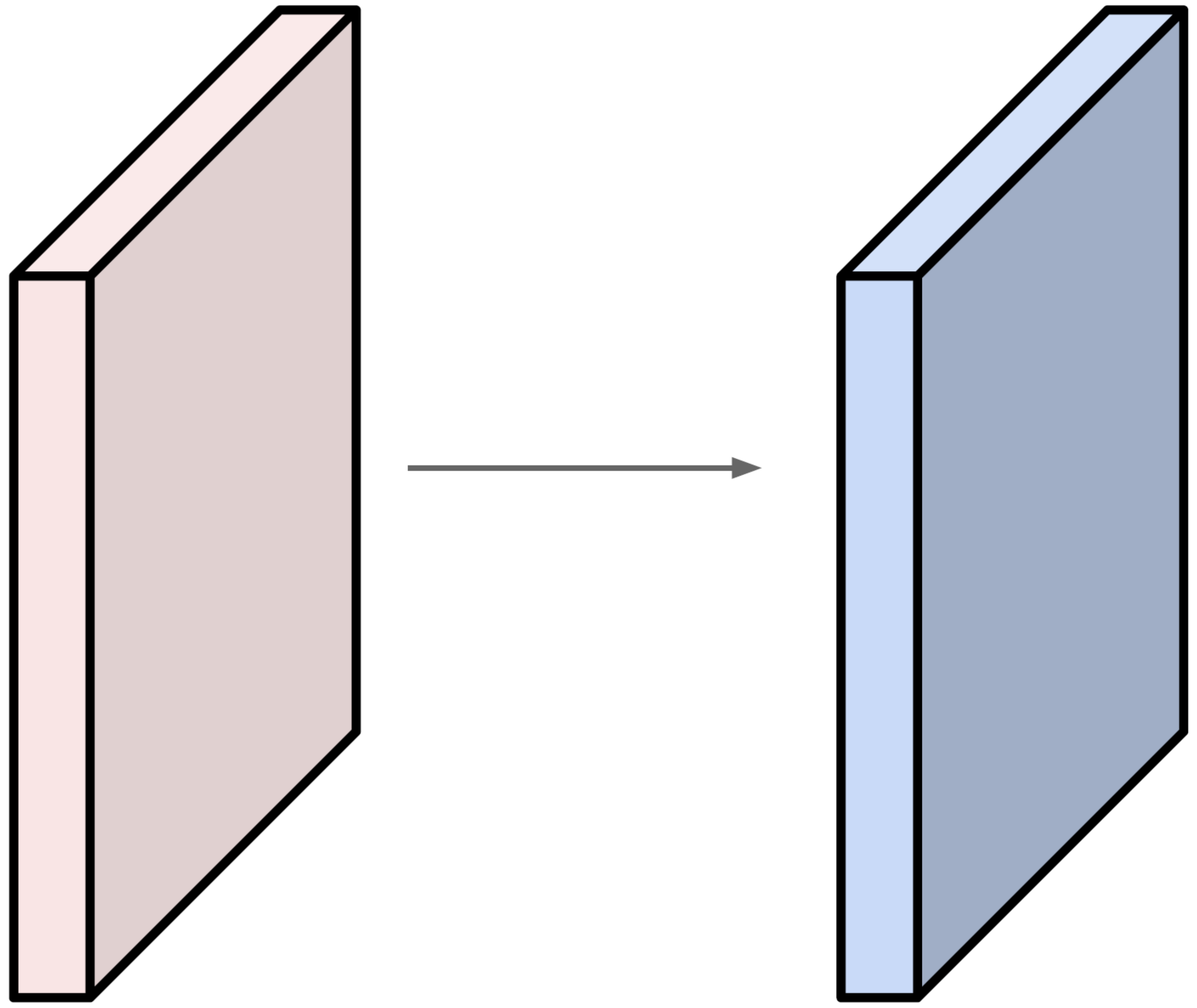


ConvNet: *Example*

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

What is the number of parameters in this layer?



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

ConvNet: *Example*

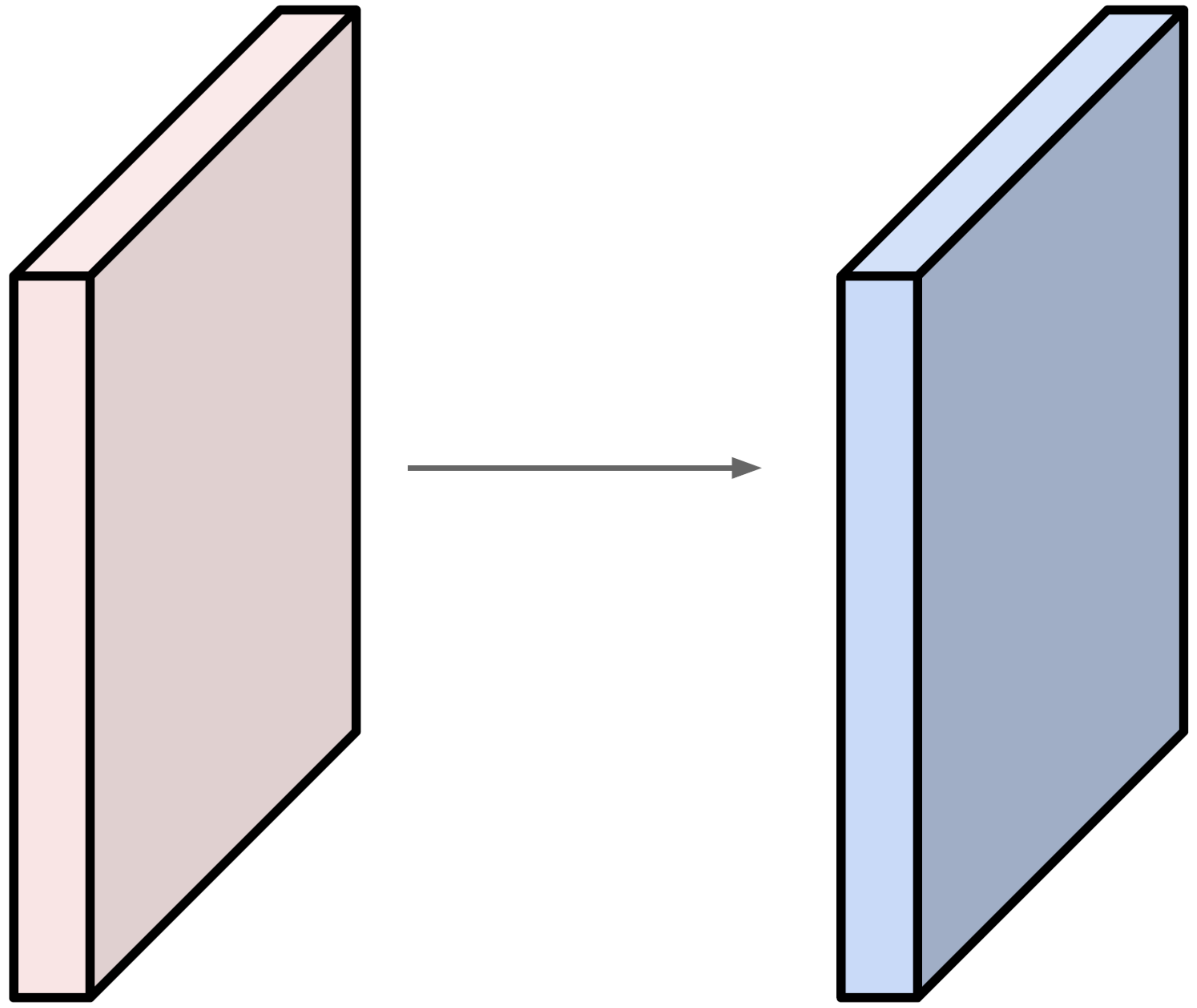
Input volume: **32x32**x3

10 5x5 filters with stride 1, pad 2

Number of parameters:

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = 760



Receptive Fields

In ConvNets, the **receptive field** refers to the region of the input space that a particular neuron in the network is "looking" at. This region is defined by the size of the filters used in the convolutional layers and the stride of the convolution operation.

At a high level, the receptive field of a neuron in a ConvNet is determined by the size of the filters in the convolutional layers that come before it in the network. Each filter has a certain size (e.g. 3x3, 5x5, etc.) and is applied to a certain region of the input data, with the size of the region determined by the stride of the convolution operation. As a result, each neuron in the network has a receptive field that encompasses a region of the input data that is a function of the sizes of the filters and the stride of the convolution operation in the layers that come before it.

The receptive field size of a ConvNet generally increases as you move deeper into the network, due to the way that convolutional layers are stacked on top of one another. This increasing receptive field size allows the network to capture increasingly complex features of the input data.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Receptive Fields

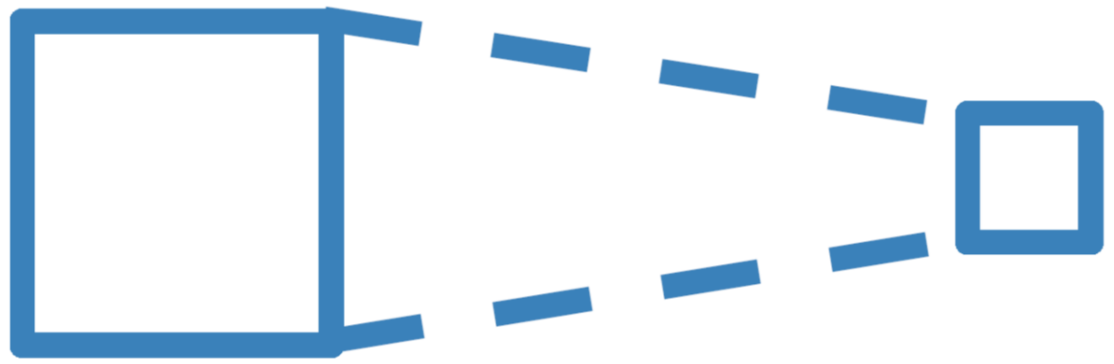
Receptive field size is an important aspect of ConvNets that can affect the network's ability to learn and generalize well to new data. There are a few reasons why receptive field size can be a problem:

1. **Limited receptive field size:** If the receptive field size of the neurons in a ConvNet is too small, the network may not be able to capture all the relevant features of the input data. This can lead to a reduction in the network's ability to recognize patterns and generalize to new data.
2. **Overfitting:** If the receptive field size of the neurons in a ConvNet is too large, the network may be more likely to overfit to the training data. This is because large receptive fields can result in high levels of parameter sharing and spatial pooling, which can cause the network to lose spatial information and capture features that are too specific to the training data.
3. **Computational cost:** Increasing the receptive field size of the neurons in a ConvNet can also increase the computational cost of training the network. This is because larger receptive fields require more parameters and more computation to train and evaluate.

Overall, the choice of receptive field size depends on the specific requirements of the task at hand and the resources available for training the network. Balancing the trade-off between model complexity, overfitting, and computational cost is an important consideration when designing ConvNets with appropriate receptive field size.

Receptive Fields

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input

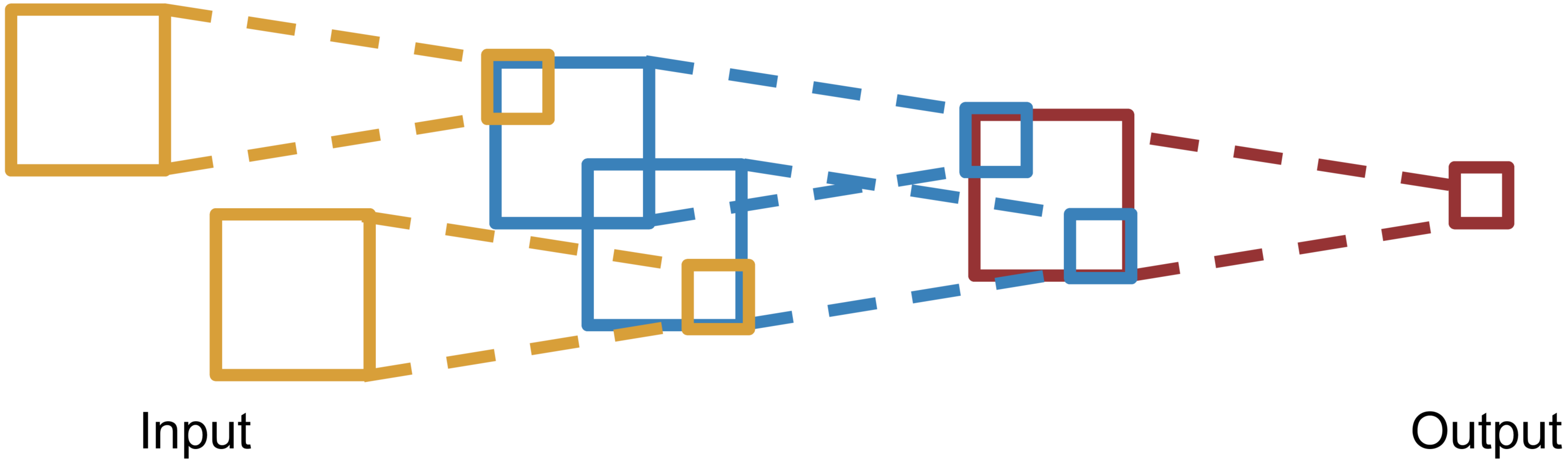


Input

Output

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



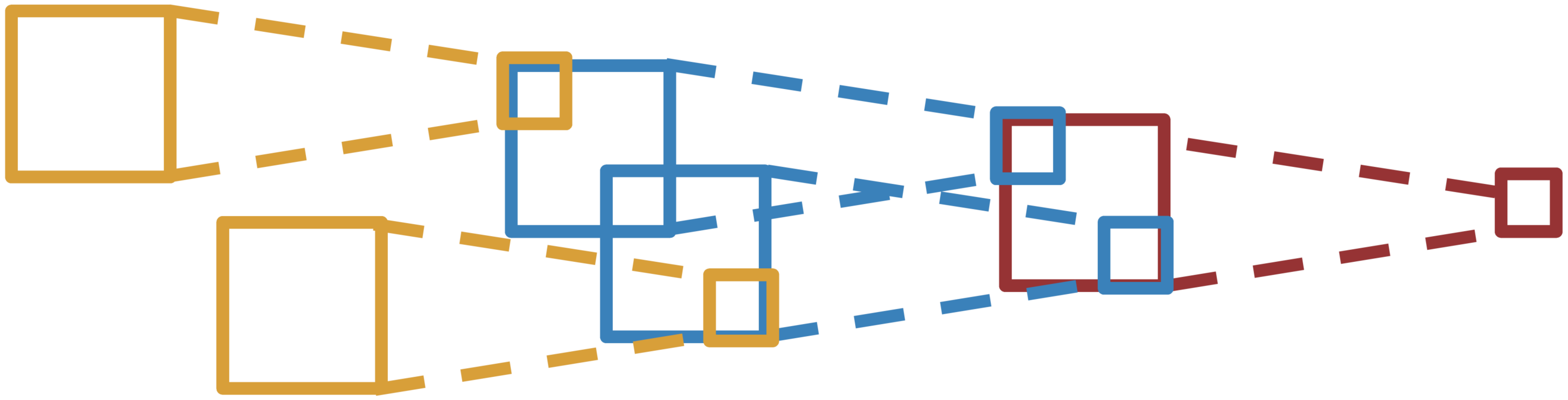
Be careful – ”receptive field in the input” vs. “receptive field in the previous layer”

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Receptive Fields

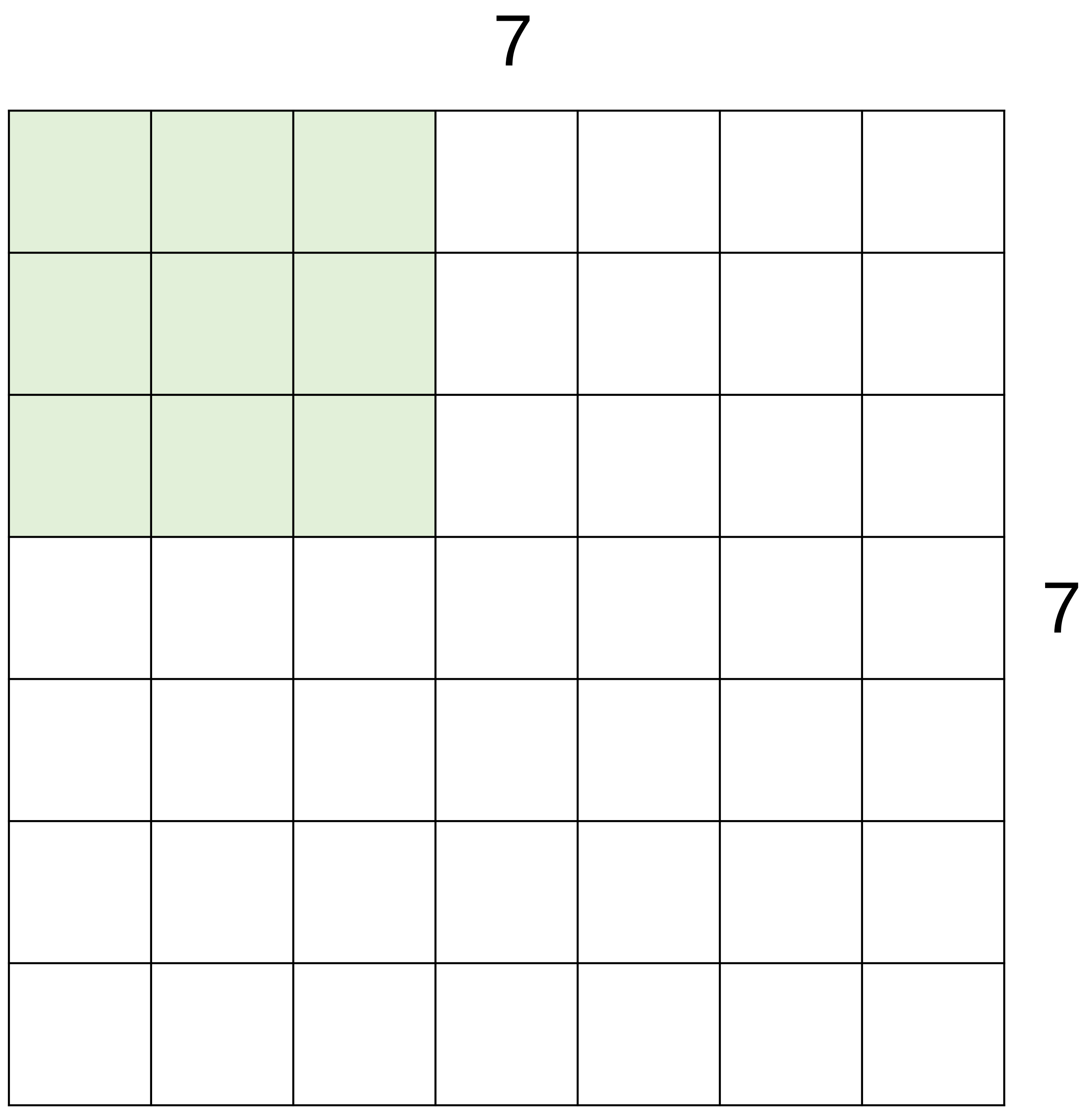
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Solution: Downsample inside the network

Problem: For large images we need many layers for each output to “see” the whole image

Receptive Fields: *Solution - Strided Convolution*

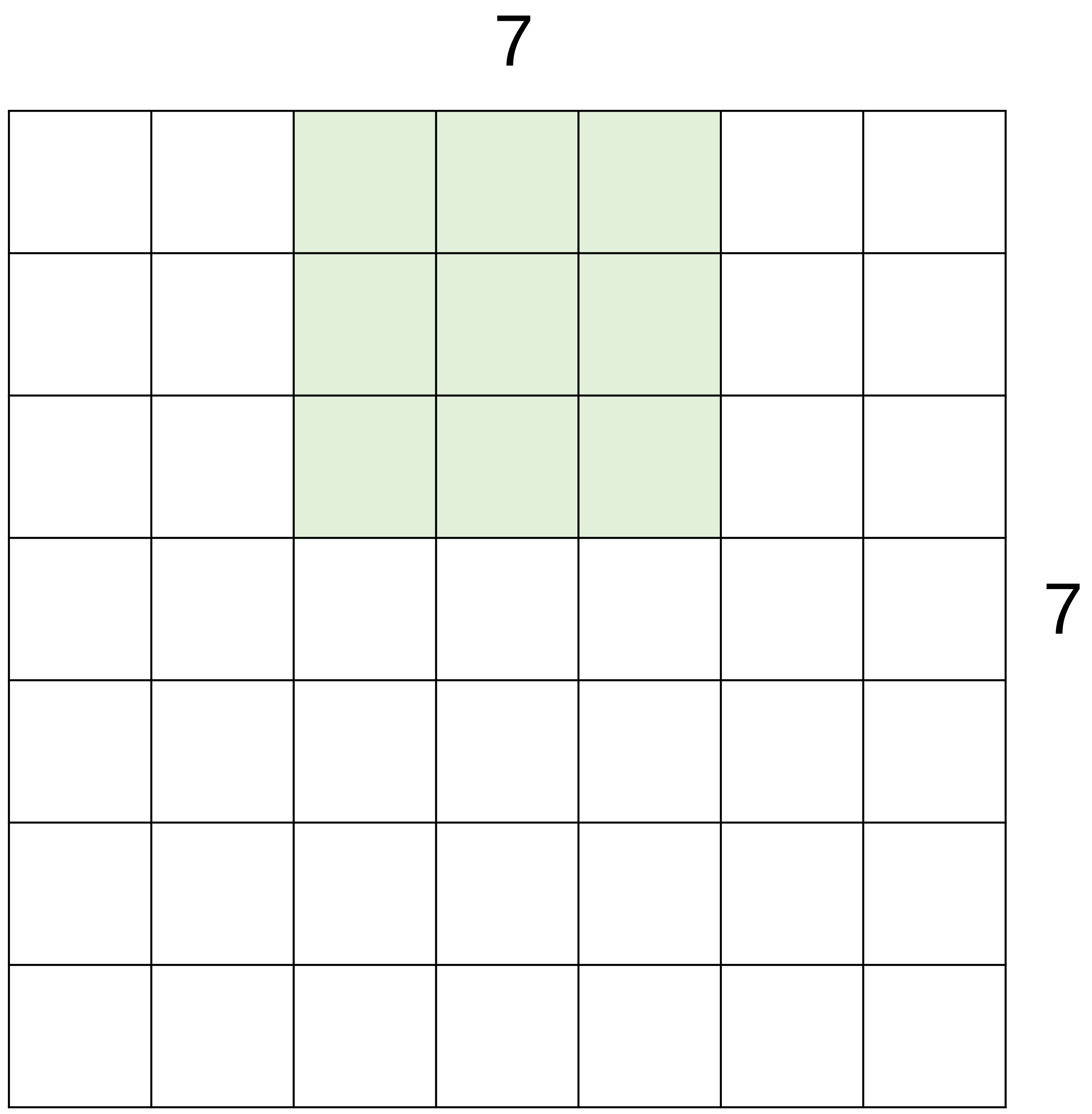


7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Receptive Fields: *Solution - Strided Convolution*



7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

=> 3x3 output!

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolution layer: *Summary*

Let's assume input is $W_1 \times H_1 \times C$

Conv. layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$ where:

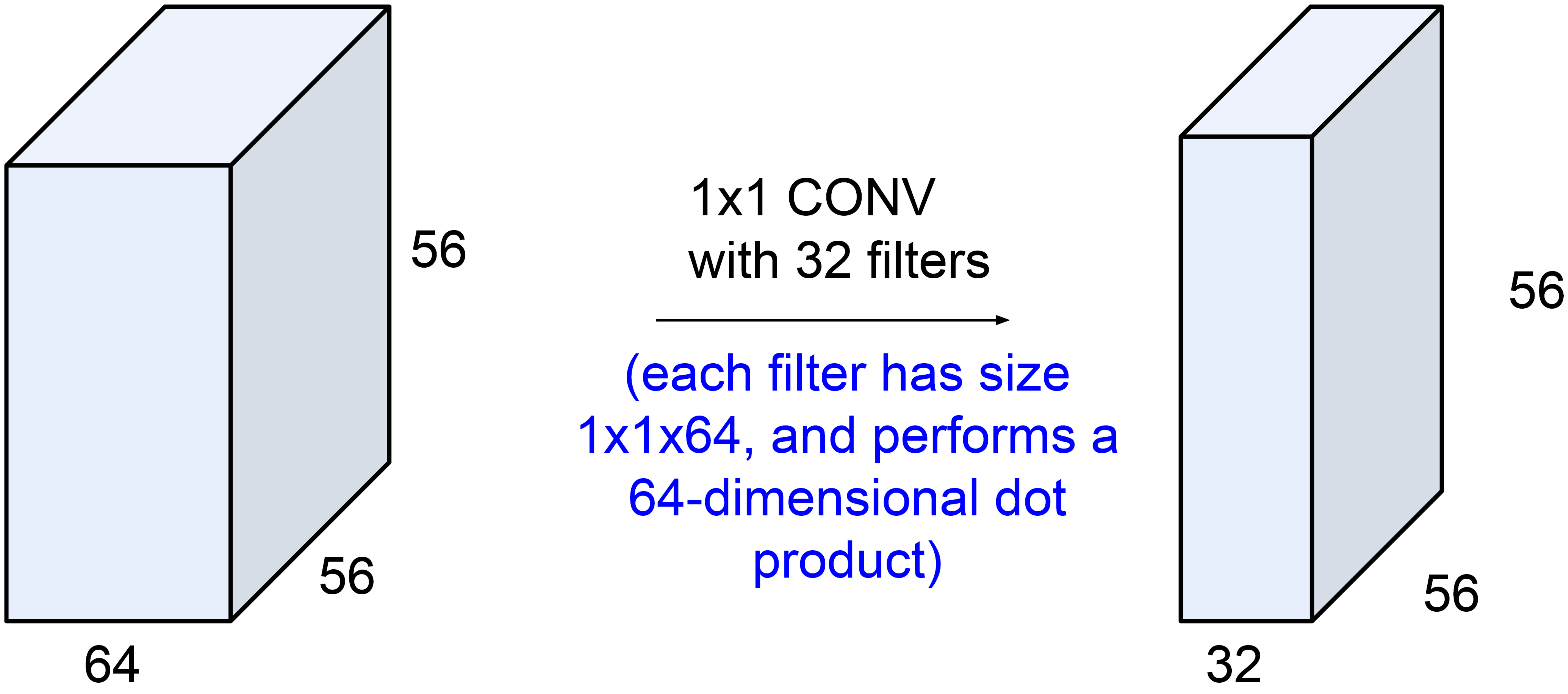
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

1x1 convolution layers make perfect sense



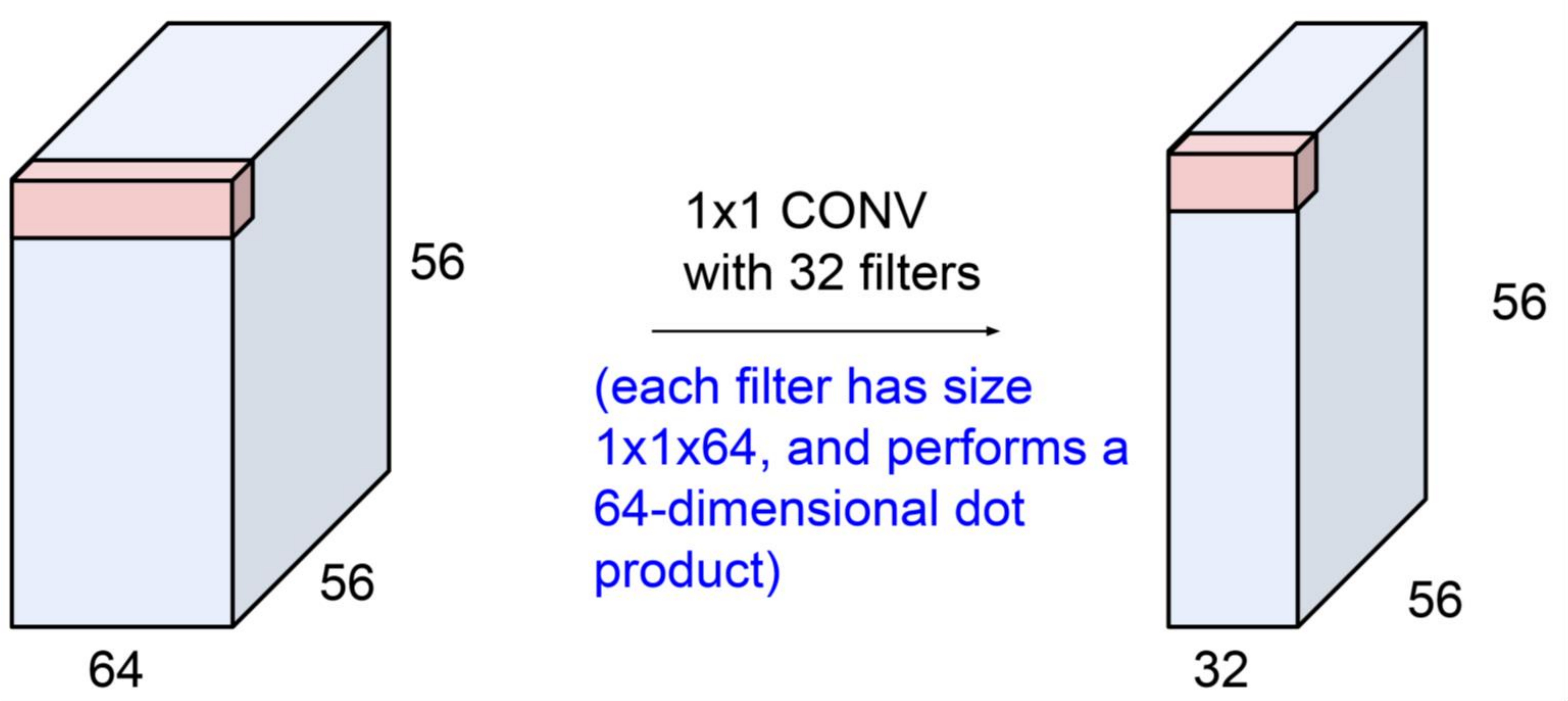
1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

1x1 convolution layers make perfect sense

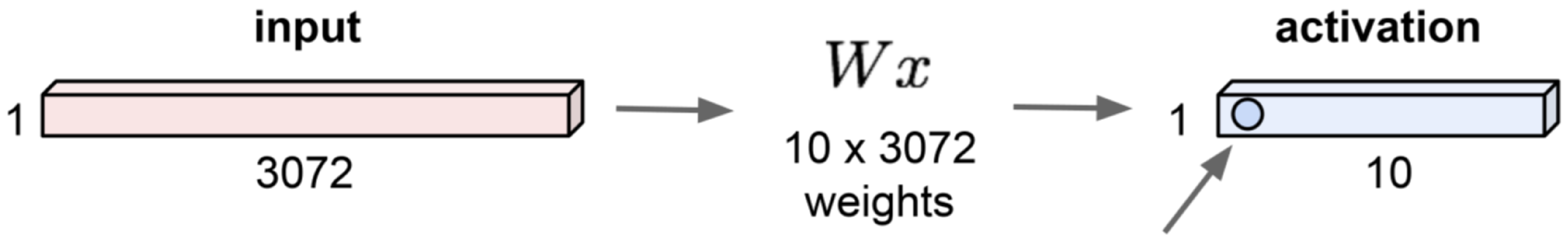


(each filter has size 1x1x64, and performs a 64-dimensional dot product)

Reminder: *Fully Connected Layer*

32x32x3 image -> stretch to 3072 x 1

Each neuron
looks at the full
input volume



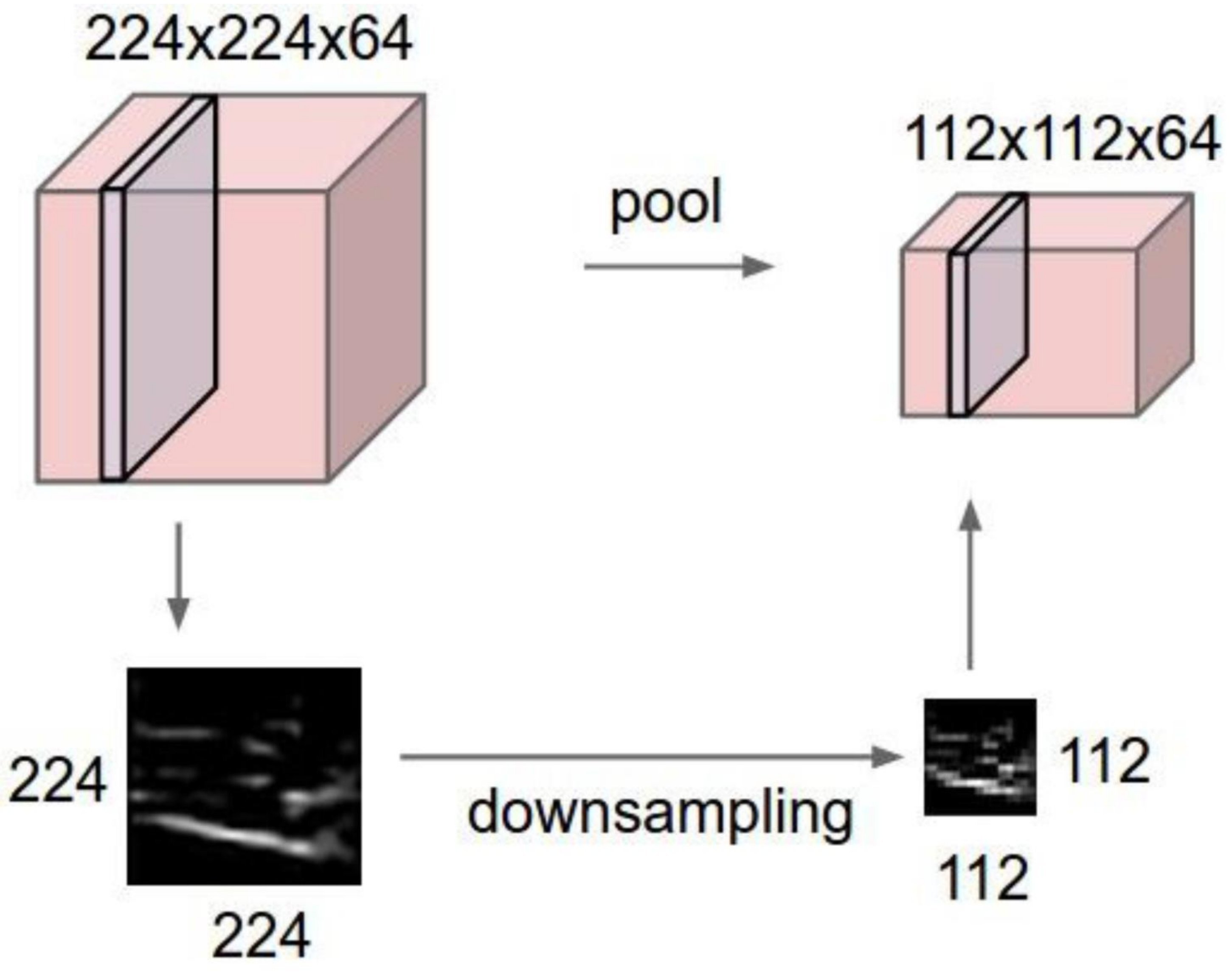
1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

Convolutional Neural Networks: *Pooling Layer*

Pooling layers are a type of layer in ConvNets that are used to reduce the spatial dimensions of the input volume.

- In **max pooling**, the input volume is divided into a set of non-overlapping rectangular regions, and the maximum value within each region is retained, while the other values are discarded. This results in a reduced spatial dimension and an increased level of spatial invariance, meaning that the output feature maps are less sensitive to small spatial translations in the input.
- In **average pooling**, the input volume is divided into the same non-overlapping rectangular regions, and the average value within each region is retained, while the other values are discarded. Like max pooling, this also results in a reduced spatial dimension and an increased level of spatial invariance.

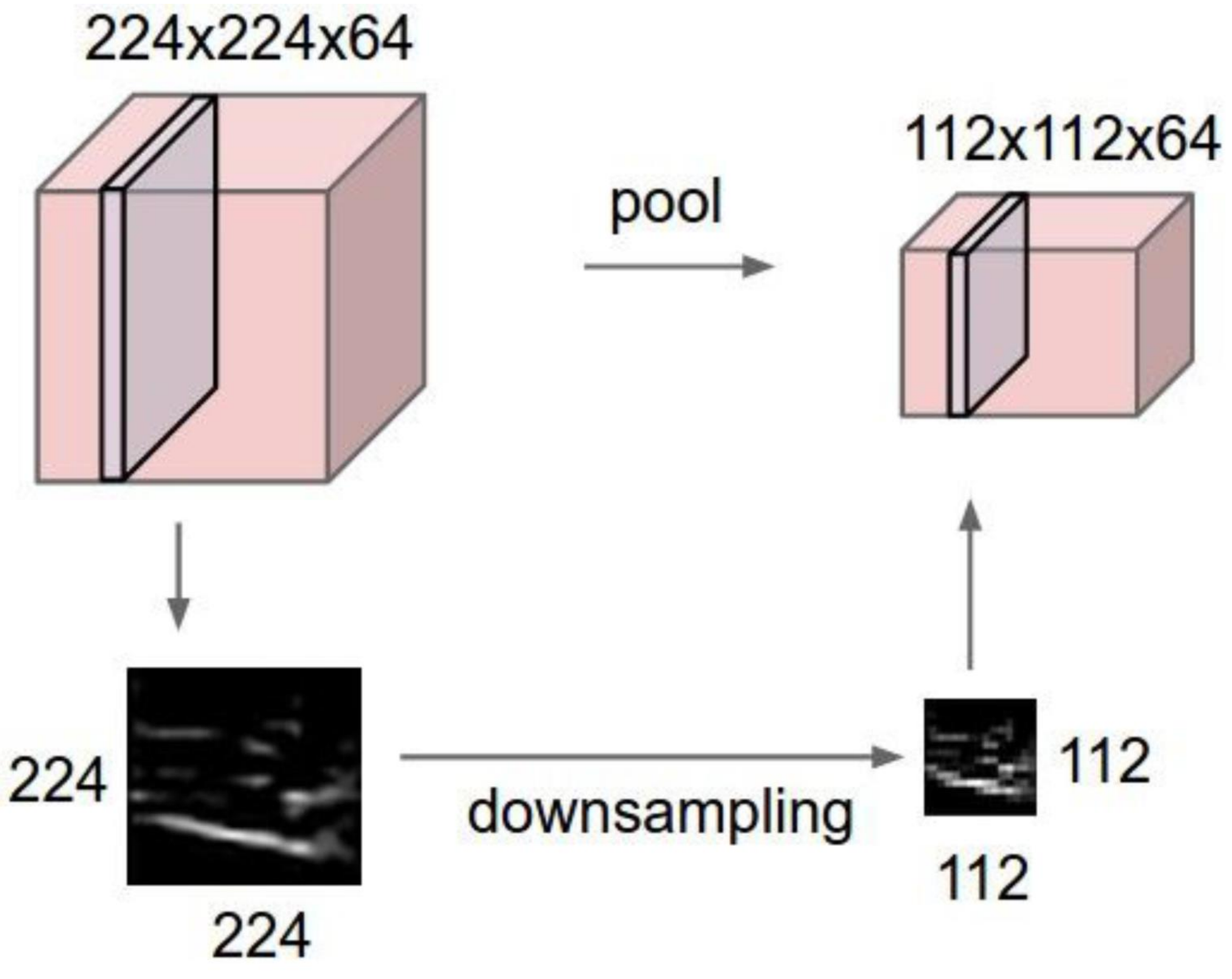
The primary purpose of pooling layers is to help the network to become more robust to small variations in the input. This is achieved by reducing the spatial resolution of the feature maps and summarizing them in a way that retains the most important information. Pooling can also help to reduce the computational cost of training the network by reducing the number of parameters in the network.



Convolutional Neural Networks: *Pooling Layer*

Pooling layers are a type of layer in ConvNets that are used to reduce the spatial dimensions of the input volume.

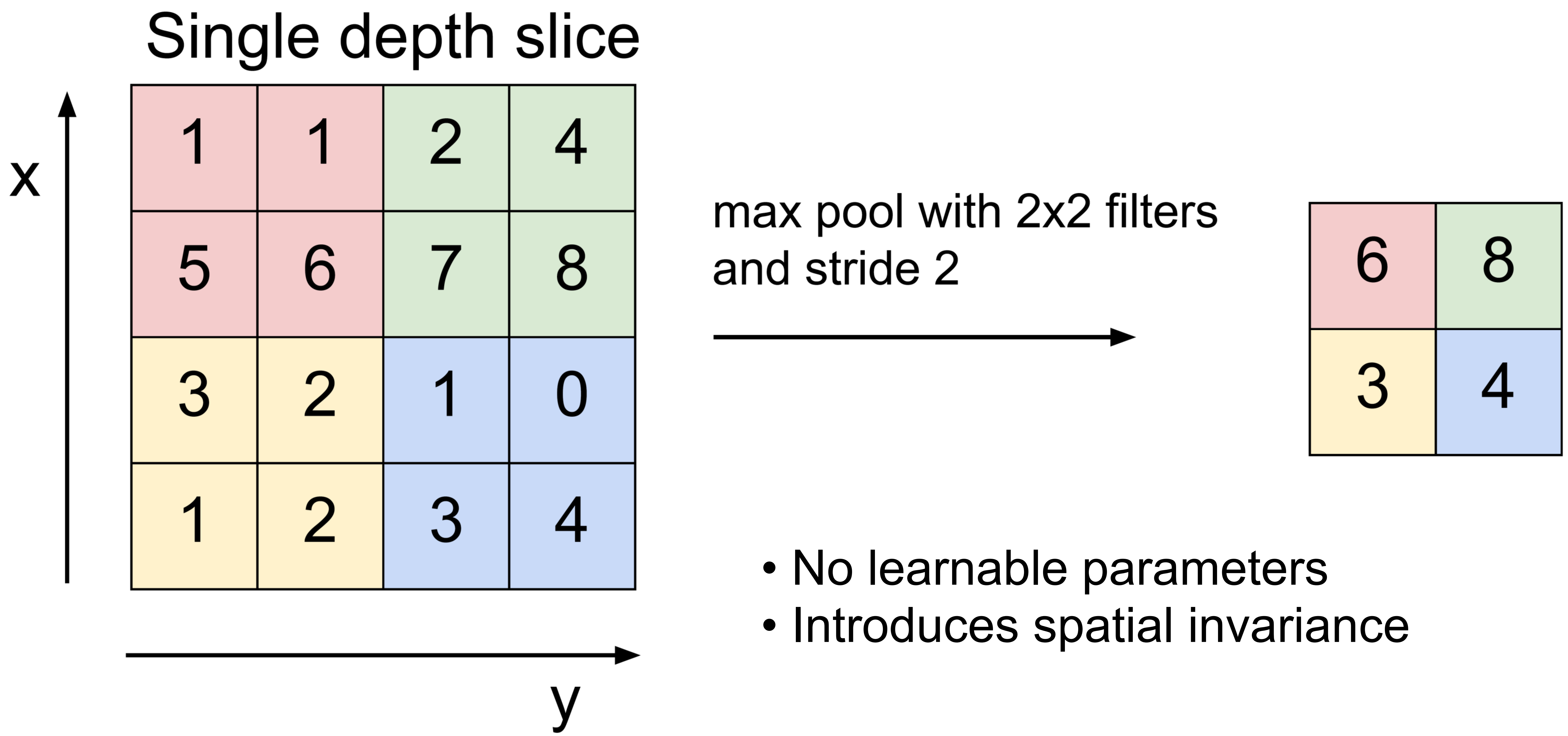
However, pooling layers can also result in a loss of information, since the discarded values do contain some spatial information. This can be mitigated by using smaller pooling regions or using other types of pooling, such as fractional max pooling, which retains more information than traditional max pooling. Overall, the choice of pooling layer and its parameters depends on the specific requirements of the task at hand and the resources available for training the network.



Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Pooling Layer: *Max pooling*



Single depth slice

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

max pool with 2x2 filters
and stride 2

6	8
3	4

- No learnable parameters
- Introduces spatial invariance

Pooling layer: *Summary*

Let’s assume input is W_1 x H_1 x C

Conv layer needs 2 hyperparameters:

- The spatial extent F
- The stride S

This will produce an output of W_2 x H_2 x C where:

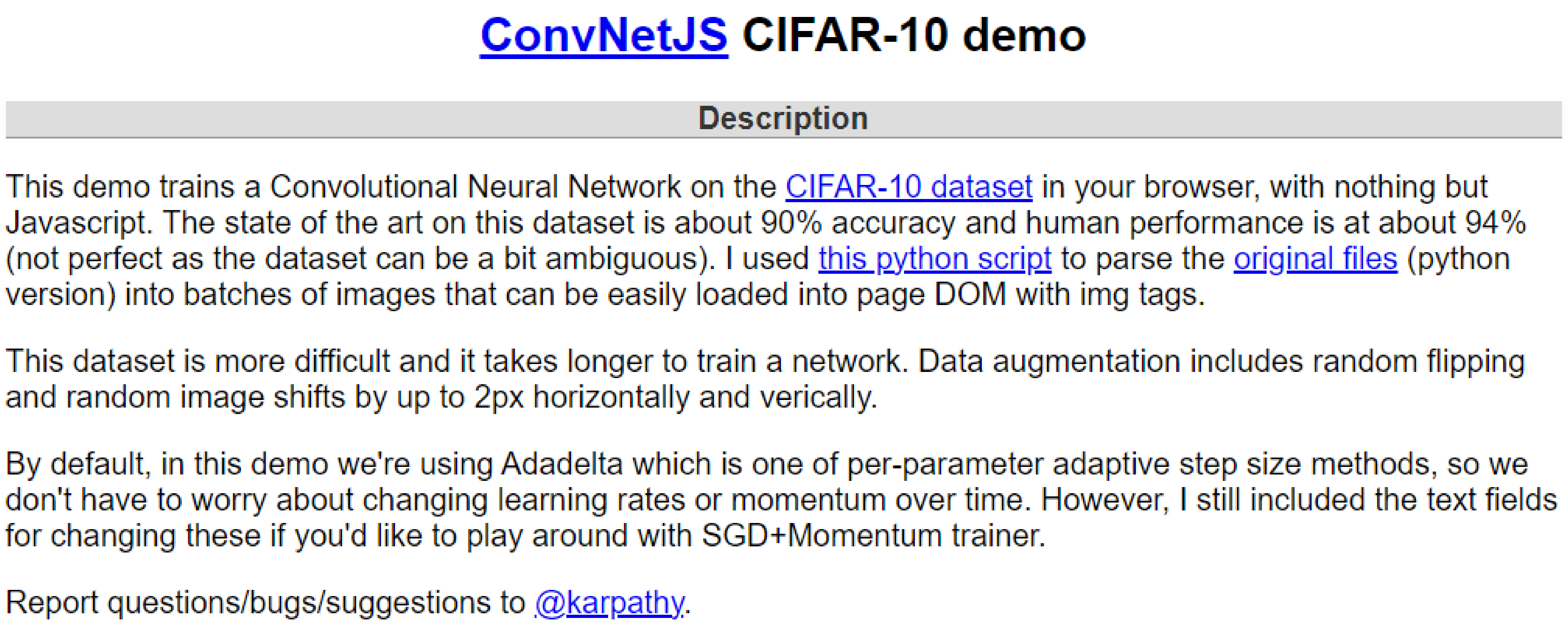
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Pooling layer: *Summary*



ConvNetJS CIFAR-10 demo

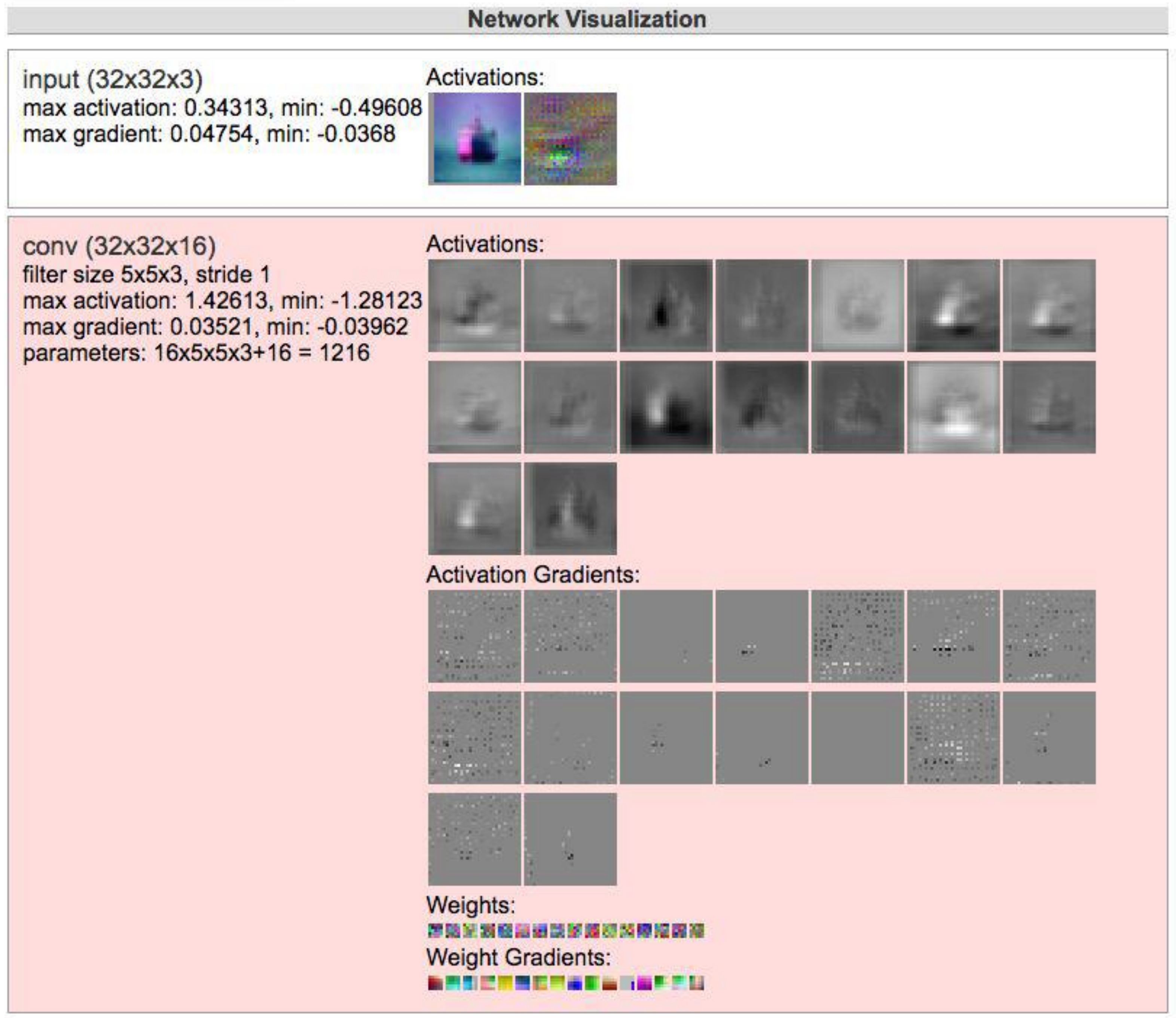
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



Network Visualization

input (32x32x3)
max activation: 0.34313, min: -0.49608
max gradient: 0.04754, min: -0.0368

Activations:

conv (32x32x16)
filter size 5x5x3, stride 1
max activation: 1.42613, min: -1.28123
max gradient: 0.03521, min: -0.03962
parameters: 16x5x5x3+16 = 1216

Activations:

Activation Gradients:

Weights:

Weight Gradients:

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

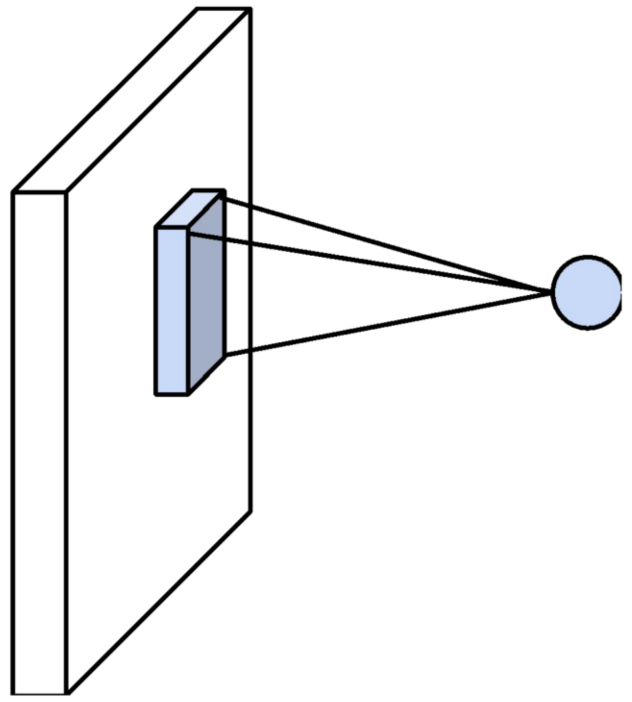
CNN Architectures

Co-financed by the European Union
Connecting Europe Facility

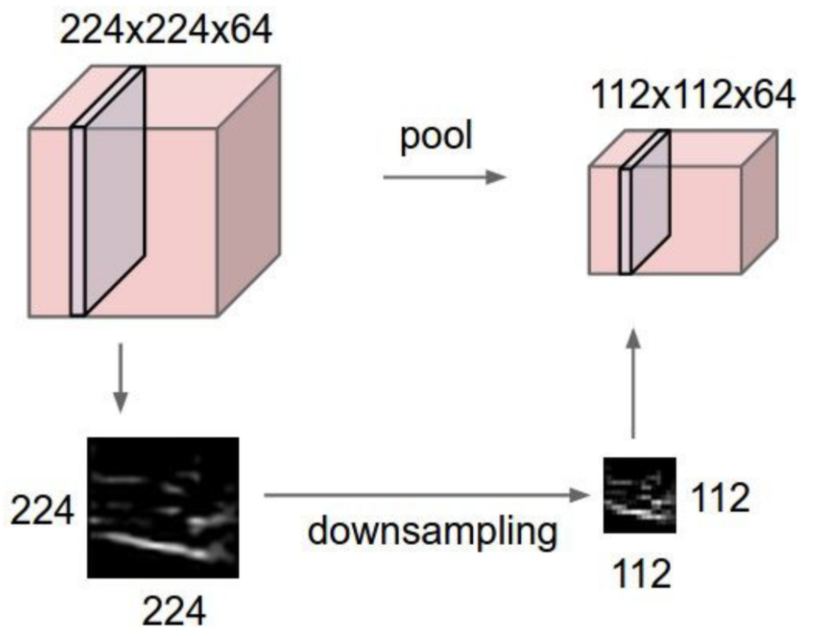
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Components of CNNs

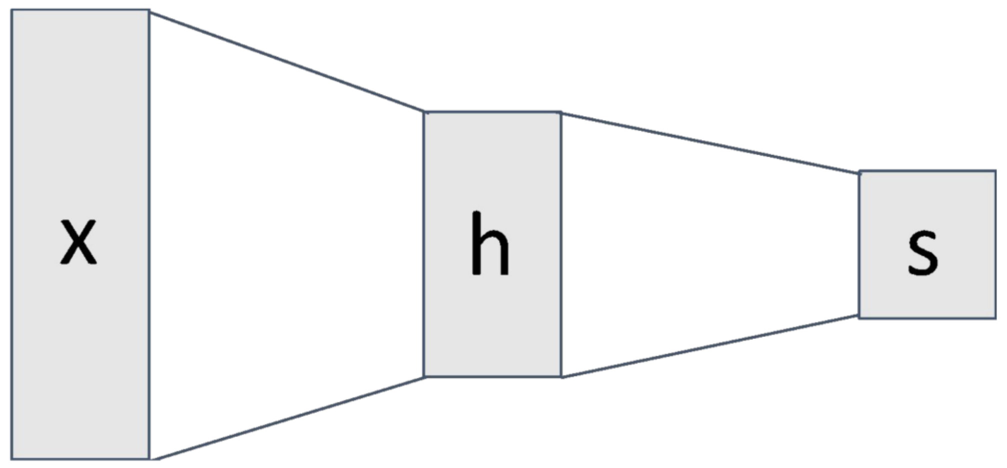
Convolution Layers



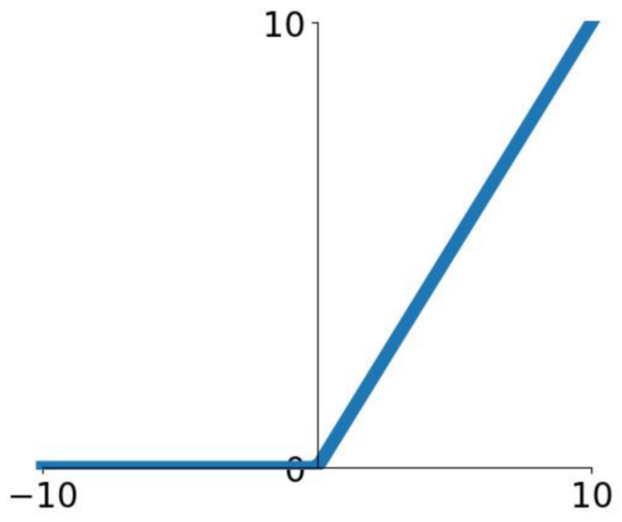
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Batch Normalization

Batch Normalization is a technique used in neural networks to normalize the activations of the previous layer before passing them to the next layer. It involves normalizing the outputs of a layer by subtracting the batch mean and dividing by the batch standard deviation, where the batch refers to the set of examples used in the current forward pass. The resulting normalized values are then scaled and shifted by learnable parameters, which allow the network to undo the normalization if necessary.

Batch Normalization is typically applied after the linear transformation of each layer and before the activation function. It can be used in various types of neural networks, including feedforward networks, convolutional neural networks, and recurrent neural networks. Overall, Batch Normalization has become a common technique for improving the training and performance of neural networks.

Batch Normalization

Batch Normalization has several benefits for neural networks. It can help to reduce the effects of internal covariate shift, which is the change in the distribution of the network's activations due to changes in the distribution of the input data. By normalizing the activations of each layer, Batch Normalization can help to ensure that the subsequent layers receive inputs that are more standardized and less likely to vary widely across different inputs.

Batch Normalization can help to regularize the network and improve its generalization performance by reducing the dependence of each layer on the precise values of the weights in the previous layer. It can also help to mitigate the vanishing and exploding gradient problems that can occur during training, by ensuring that the gradients are centered and have a moderate variance.

Batch Normalization

Consider a single layer $y = Wx$

The following could lead to tough optimization

- Input x are not centered around zero (need large bias)
- Input x have different scaling per-element (entries in W will need to vary a lot)

Solution: Force inputs to be nicely scaled at each layer

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Batch Normalization

“you want zero-mean unit-variance activations? just make them so.”

Consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

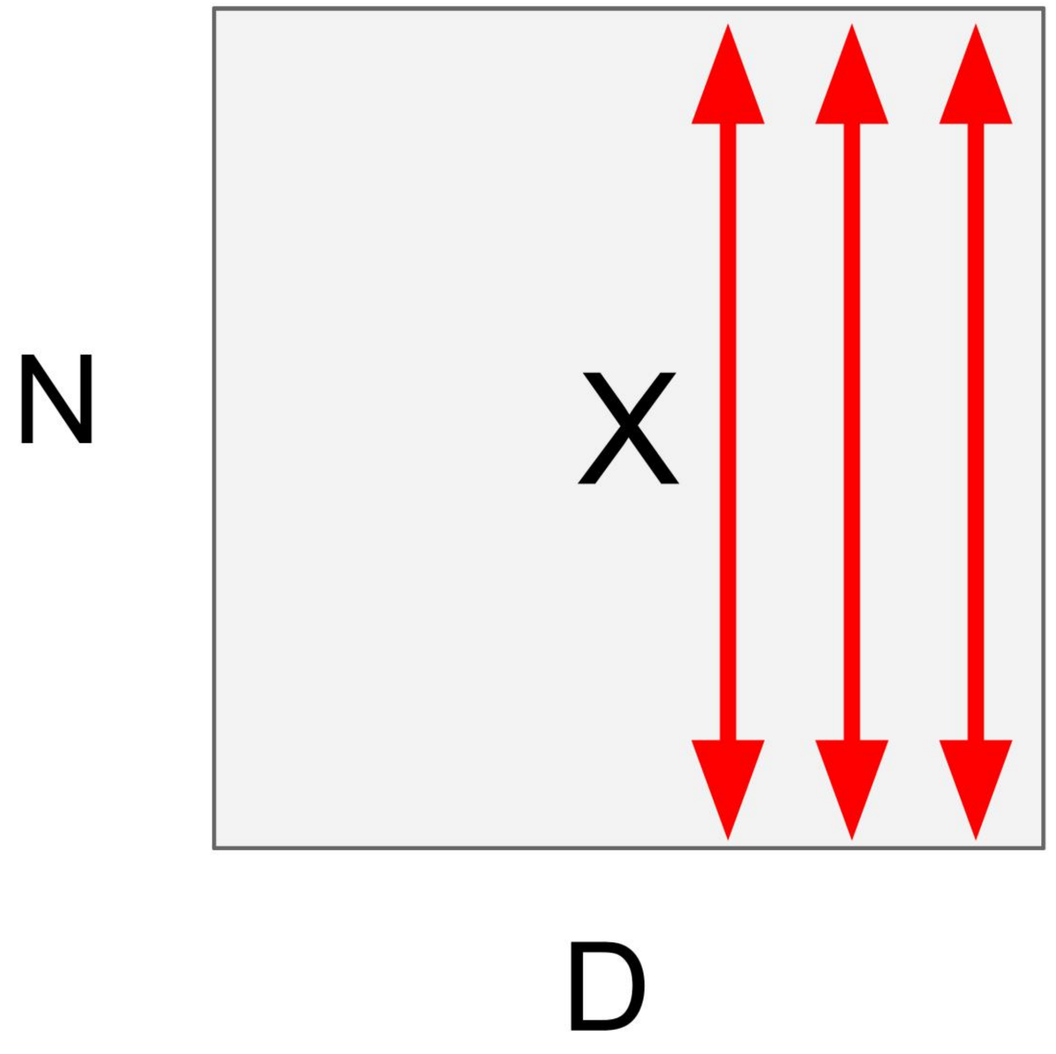
[Ioffe and Szegedy, 2015]

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Batch Normalization

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

[Ioffe and Szegedy, 2015]

Problem: What if zero-mean, unit variance is too hard of a constraint?

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Batch Normalization

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

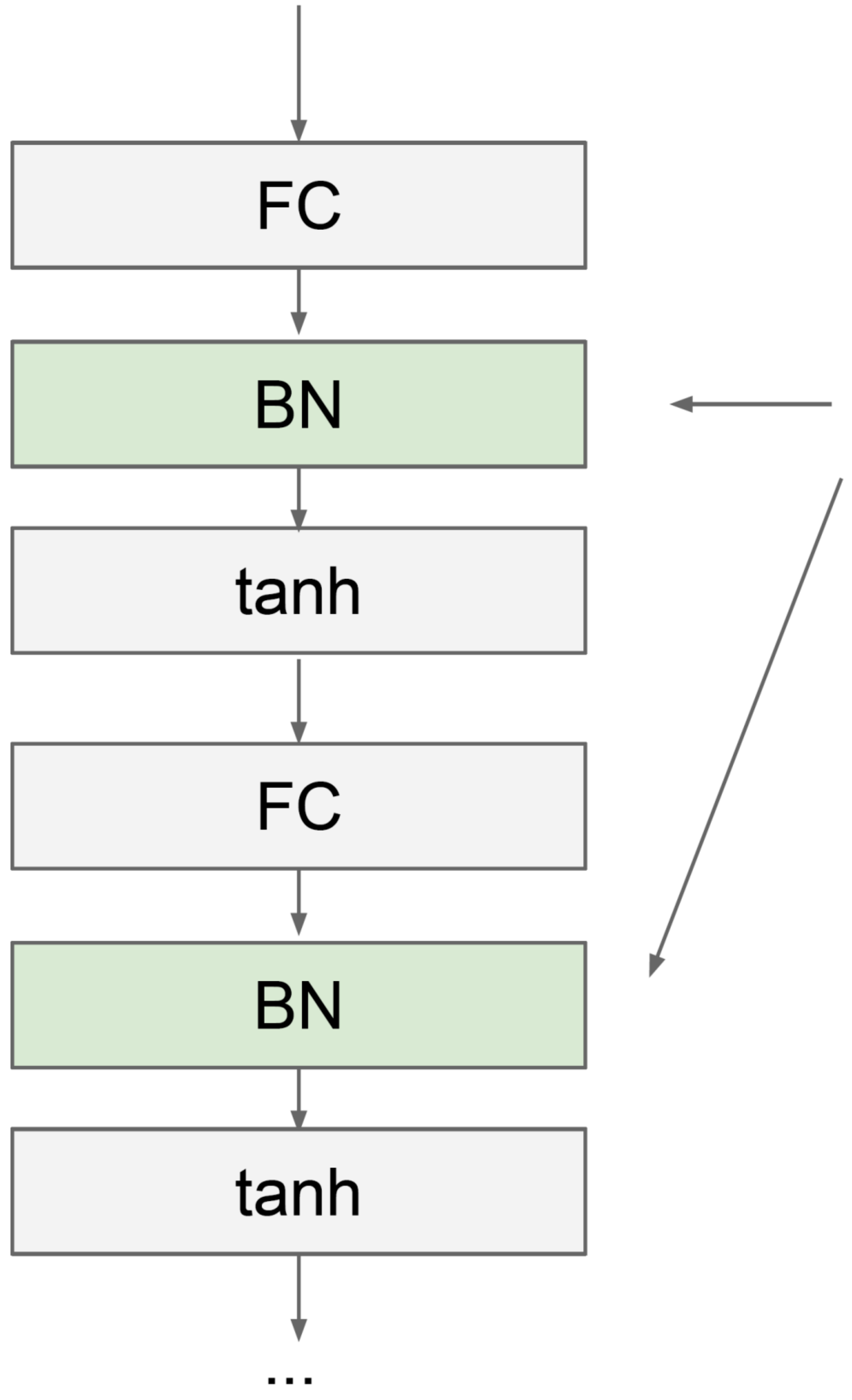
Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

[Ioffe and Szegedy, 2015]

Batch Normalization

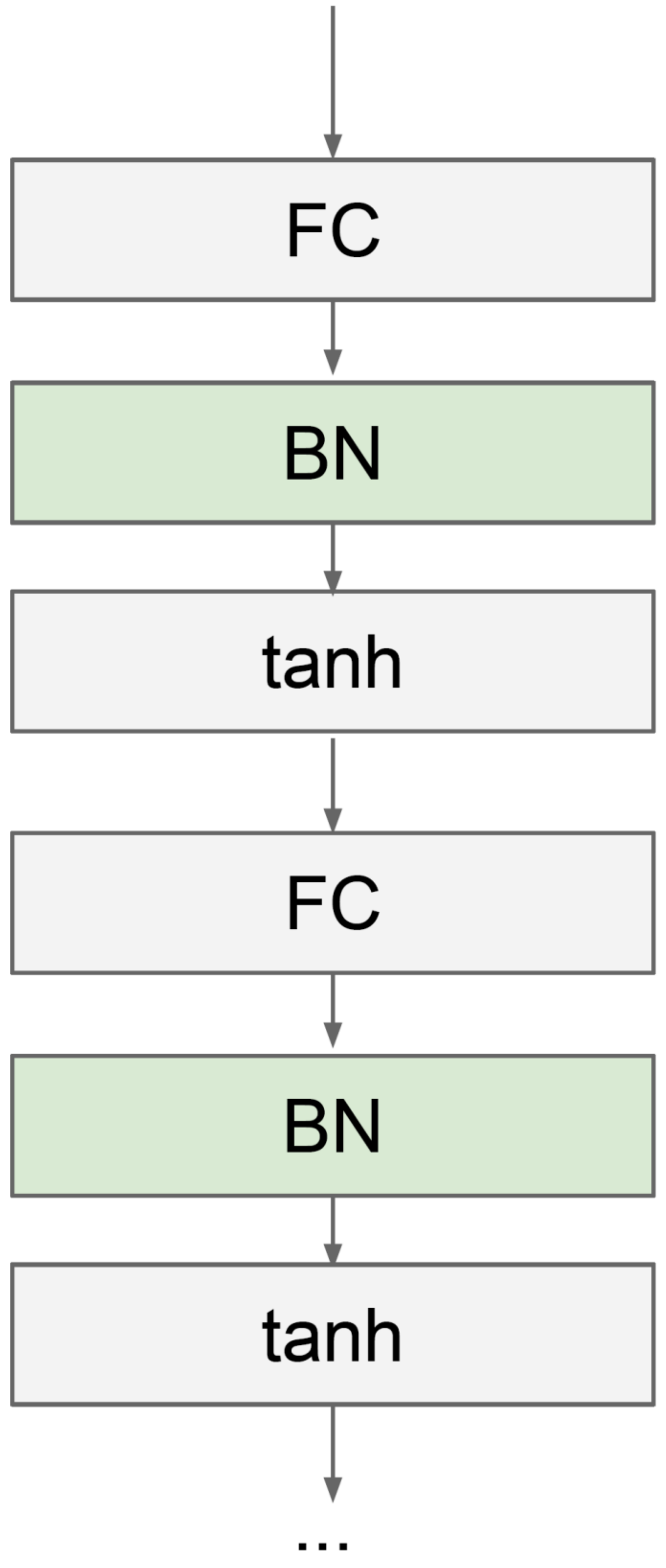


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

[Ioffe and Szegedy, 2015]

Batch Normalization



- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

[Ioffe and Szegedy, 2015]

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

BatchNorm, LayerNorm, and InstanceNorm

Batch normalization is a technique that normalizes the activations of the previous layer for each batch in the training process, so that the mean is zero and the standard deviation is one. This helps to alleviate the problem of internal covariate shift, which can cause the model to converge more slowly or lead to overfitting.

Layer normalization is similar to batch normalization, but it normalizes the activations across all of the inputs for a given layer, rather than just across the batch. This makes it more suitable for recurrent neural networks and other models that don't process inputs in batches, as it helps to reduce the sensitivity to the order of the inputs.

Instance normalization, on the other hand, normalizes the activations across each channel in the input, which is especially useful for style transfer and other image-related tasks. It can also be used in convolutional neural networks to normalize the activations across the spatial dimensions of the input.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

BatchNorm, LayerNorm, and InstanceNorm

The main difference between these normalization techniques lies in the scope of the normalization. BatchNorm normalizes the activations over the entire batch, LayerNorm normalizes the activations over all of the inputs for a given layer, and InstanceNorm normalizes the activations across each channel in the input. The choice of normalization technique depends on the specific characteristics of the problem being solved and the structure of the model.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$$x: \underset{\downarrow}{N} \times D$$

Normalize

$$\mu, \sigma: 1 \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$x: \underset{\downarrow}{N} \times C \times \underset{\downarrow}{H} \times \underset{\downarrow}{W}$$

Normalize

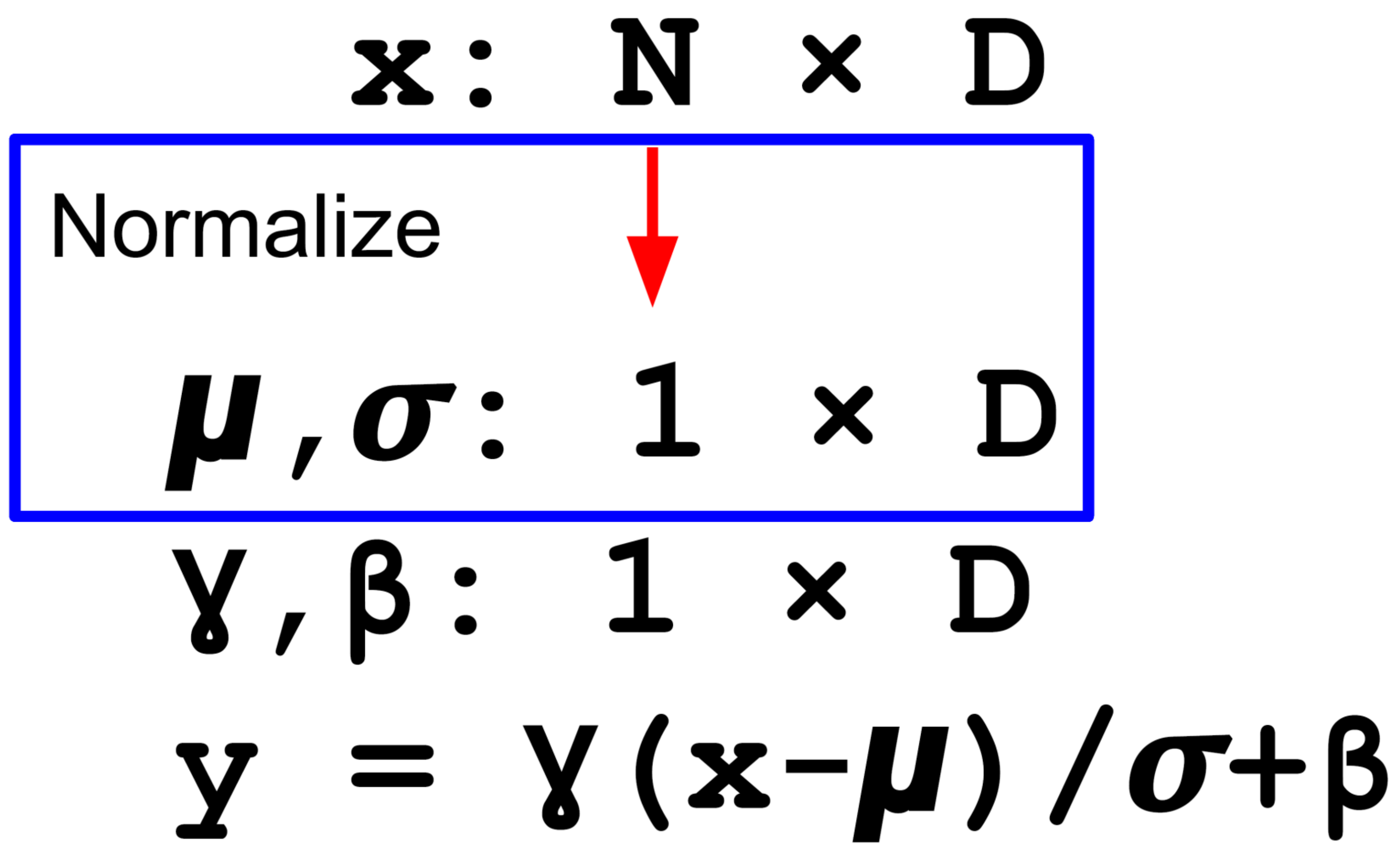
$$\mu, \sigma: 1 \times C \times 1 \times 1$$

$$\gamma, \beta: 1 \times C \times 1 \times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Layer Normalization

Batch Normalization for
fully-connected networks



$$x: N \times D$$

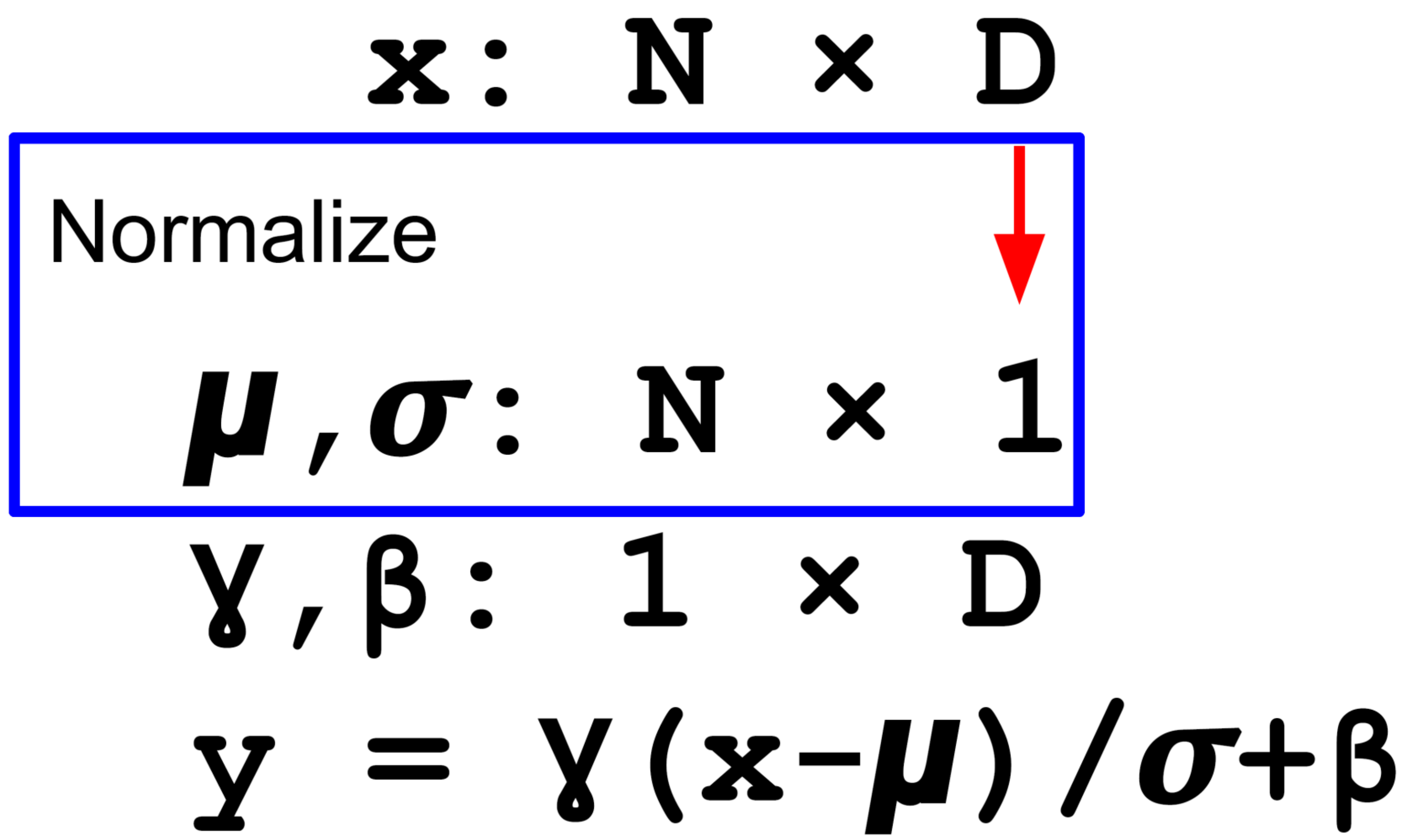
Normalize

$$\mu, \sigma: 1 \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks



$$x: N \times D$$

Normalize

$$\mu, \sigma: N \times 1$$

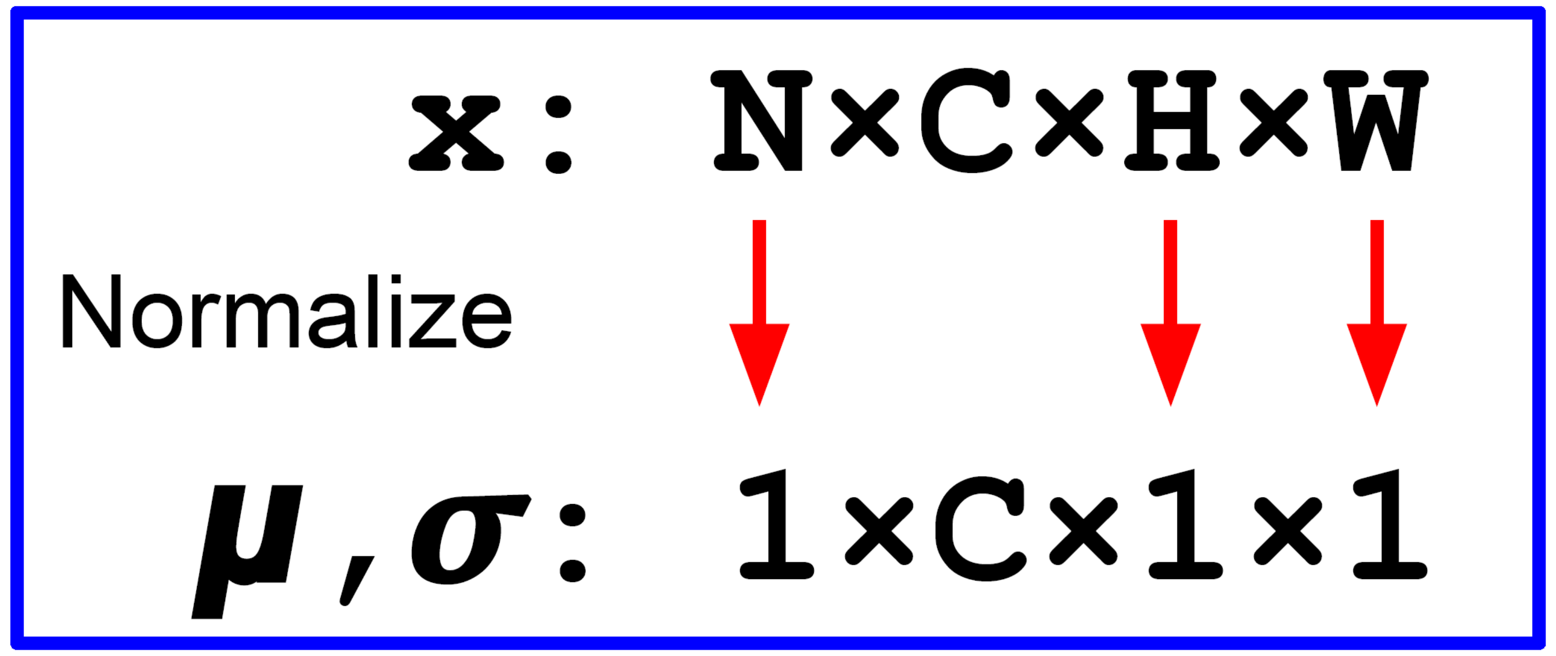
$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Ba, Kiros, and Hinton, “Layer Normalization”, arXiv 2016

Instance Normalization

Batch Normalization for
convolutional networks



$$x:\ N\times C\times H\times W$$

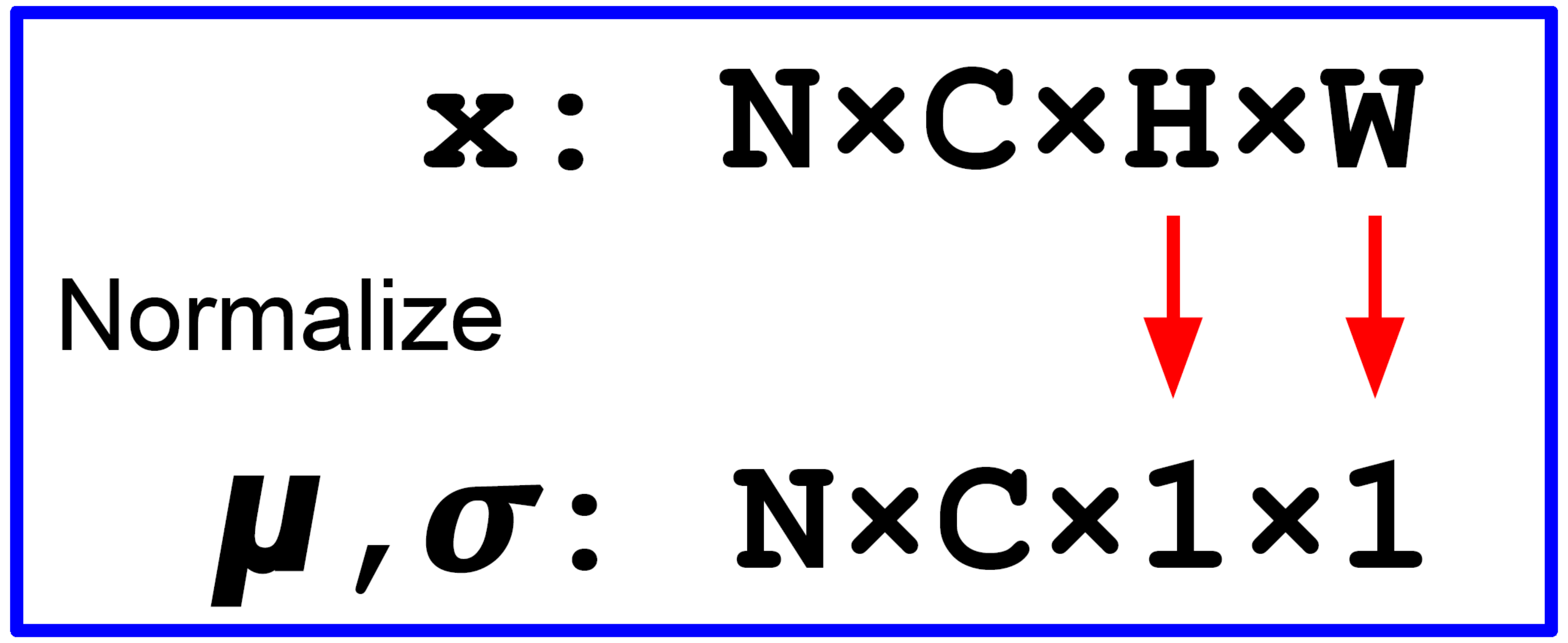
Normalize

$$\mu,\sigma:\ 1\times C\times 1\times 1$$

$$\gamma,\beta:\ 1\times C\times 1\times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Instance Normalization for
convolutional networks
Same behavior at train / test!



$$x:\ N\times C\times H\times W$$

Normalize

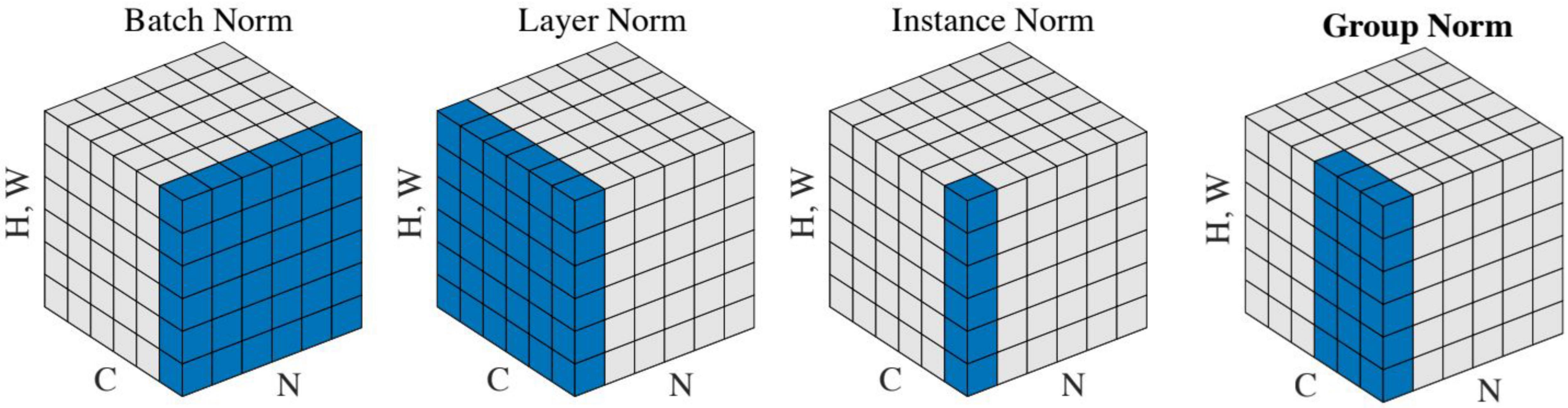
$$\mu,\sigma:\ N\times C\times 1\times 1$$

$$\gamma,\beta:\ 1\times C\times 1\times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Comparison of Normalization Layers



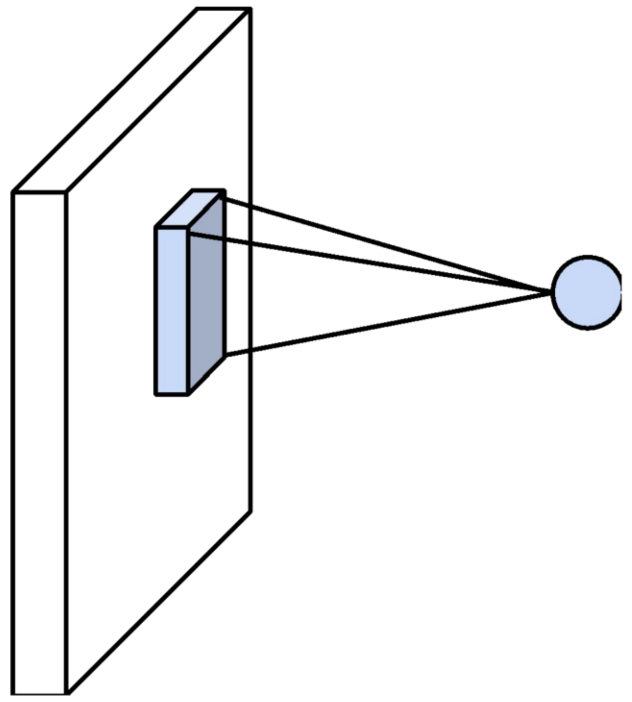
Wu and He, “Group Normalization”, ECCV 2018

Co-financed by the European Union
Connecting Europe Facility

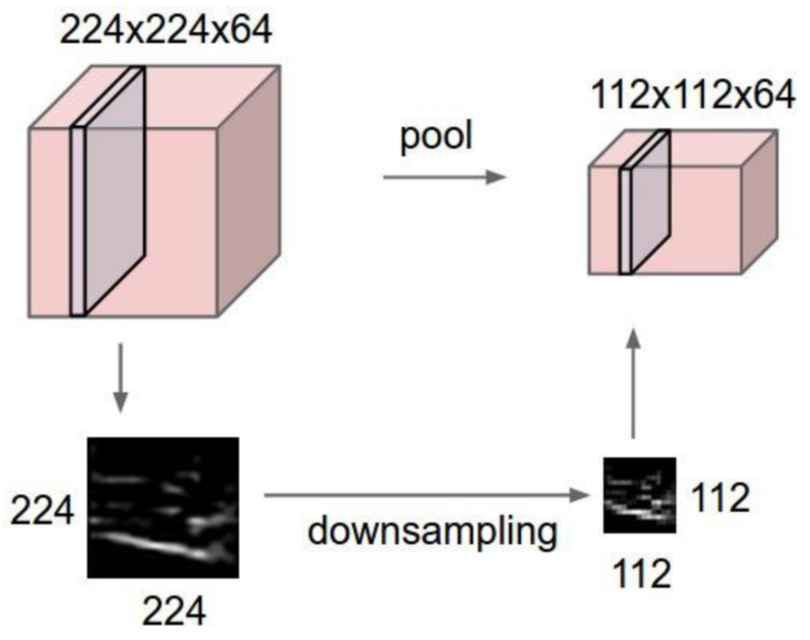
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Components of CNNs

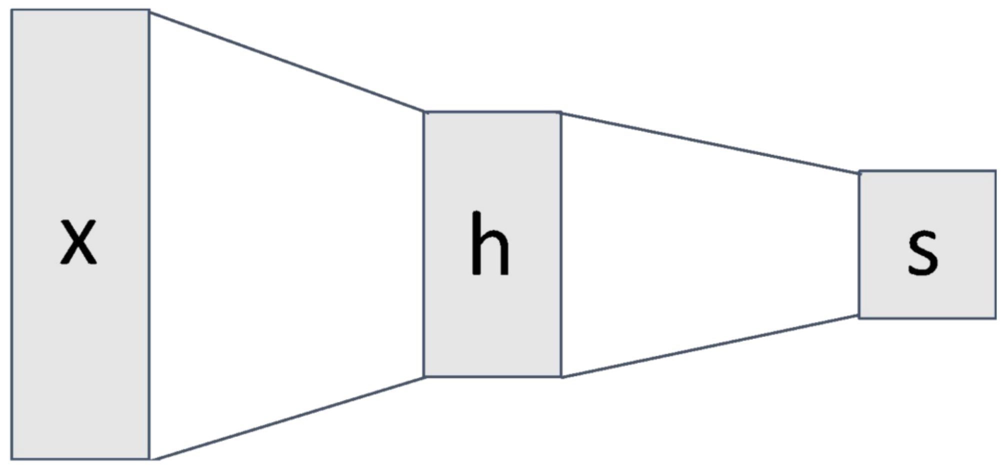
Convolution Layers



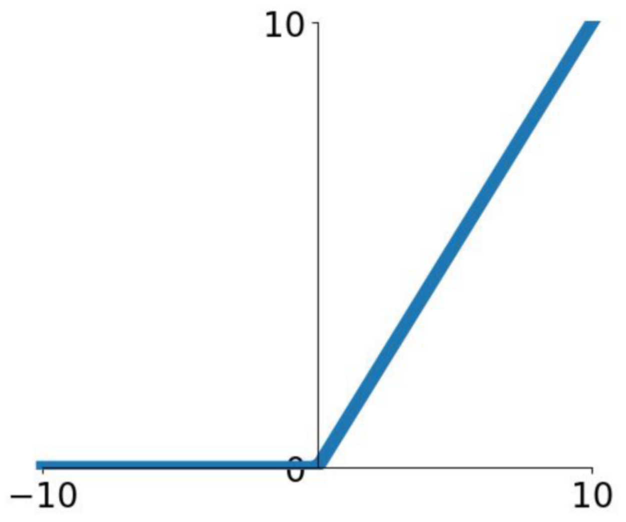
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Question: How should we put them together?

CNN Architectures: *LeNet-5*

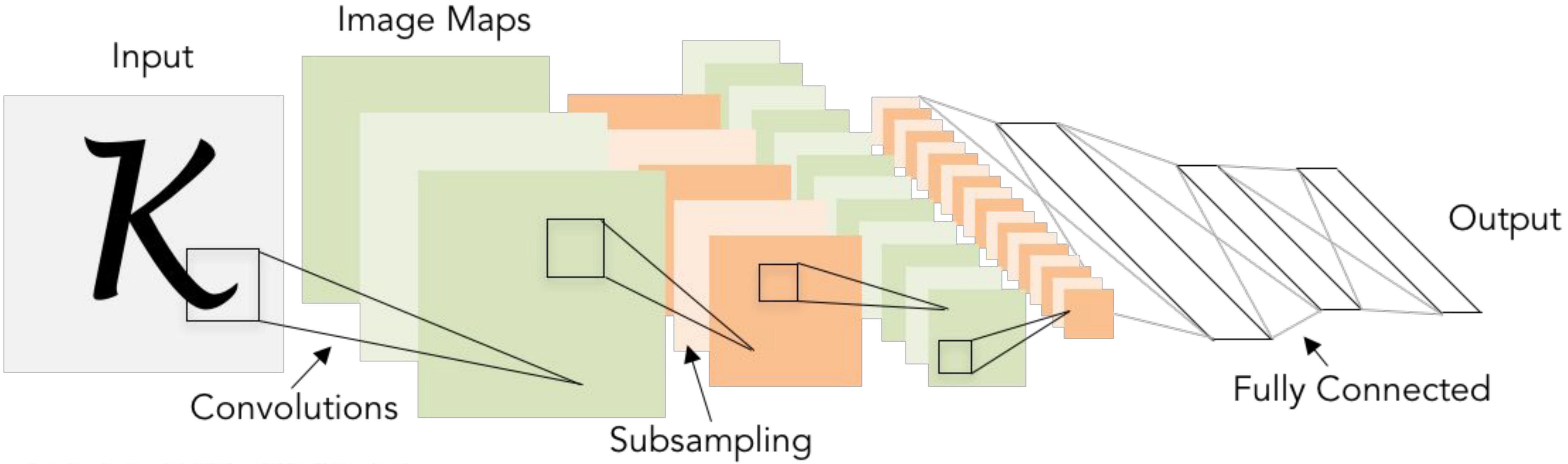


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

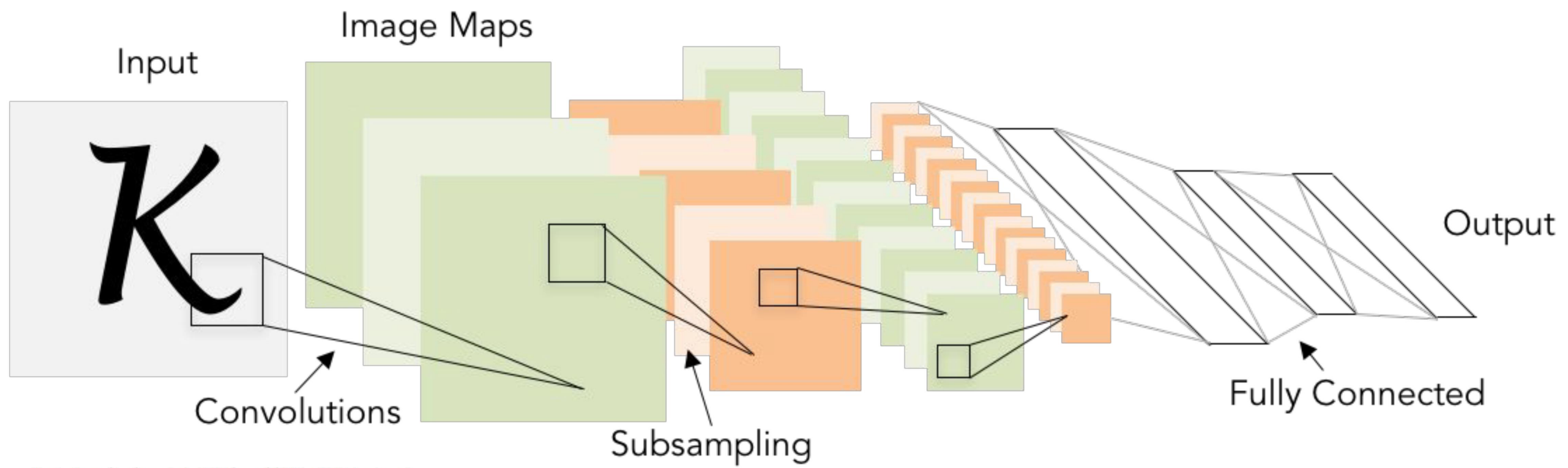


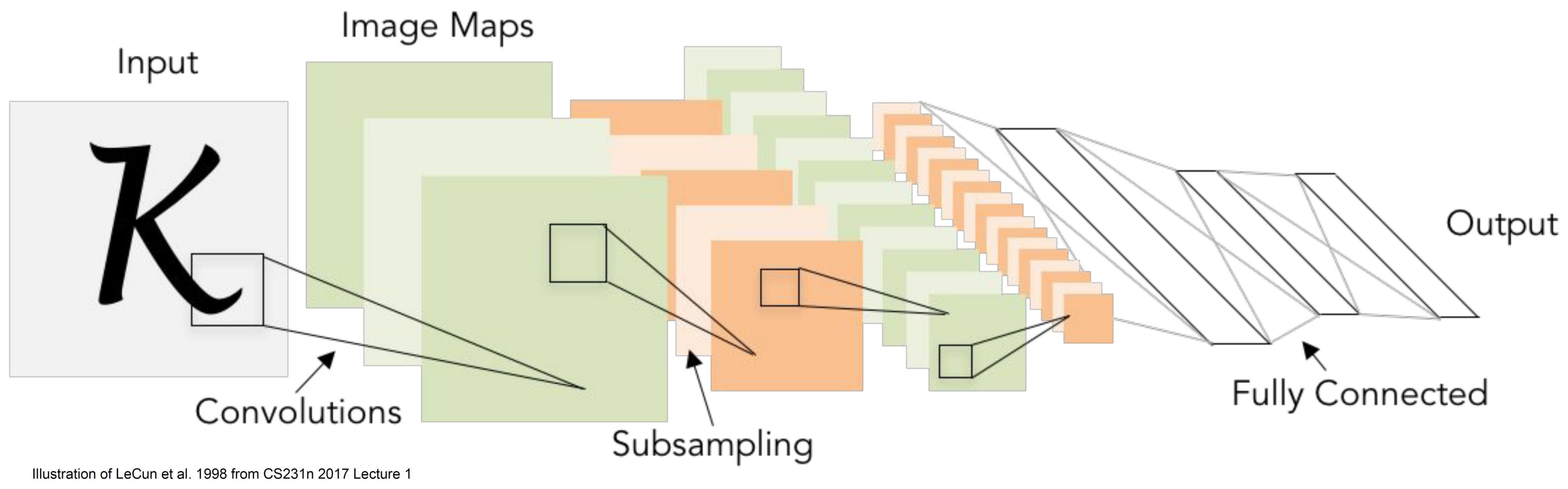
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

CNN Architectures: *LeNet-5*

LeNet-5 is a convolutional neural network (CNN) designed for handwritten digit recognition, proposed by. It consists of seven layers, including three convolutional layers and two fully connected layers, with an input size of 32 x 32 grayscale images. The architecture of LeNet-5 can be summarized as follows:

- Convolutional Layer: 6 filters of size 5x5 with a stride of 1 and a sigmoid activation function
- Average Pooling Layer: non-overlapping 2x2 window with a stride of 2
- Convolutional Layer: 16 filters of size 5x5 with a stride of 1 and a sigmoid activation function
- Average Pooling Layer: non-overlapping 2x2 window with a stride of 2
- Fully Connected Layer: 120 units with a sigmoid activation function
- Fully Connected Layer: 84 units with a sigmoid activation function
- Output Layer: 10 units with a softmax activation function, representing the 10 possible digits (0-9)

The LeNet-5 architecture introduced several key concepts in deep learning, including the use of convolutional layers and pooling layers, as well as the efficient training of neural networks using backpropagation and stochastic gradient descent. It was also one of the first successful models to use weight sharing and local connections, which reduces the number of parameters and improves the generalization ability of the model.



CNN Architectures: *Case studies*

- AlexNet
- VGG
- GoogLeNet
- ResNet

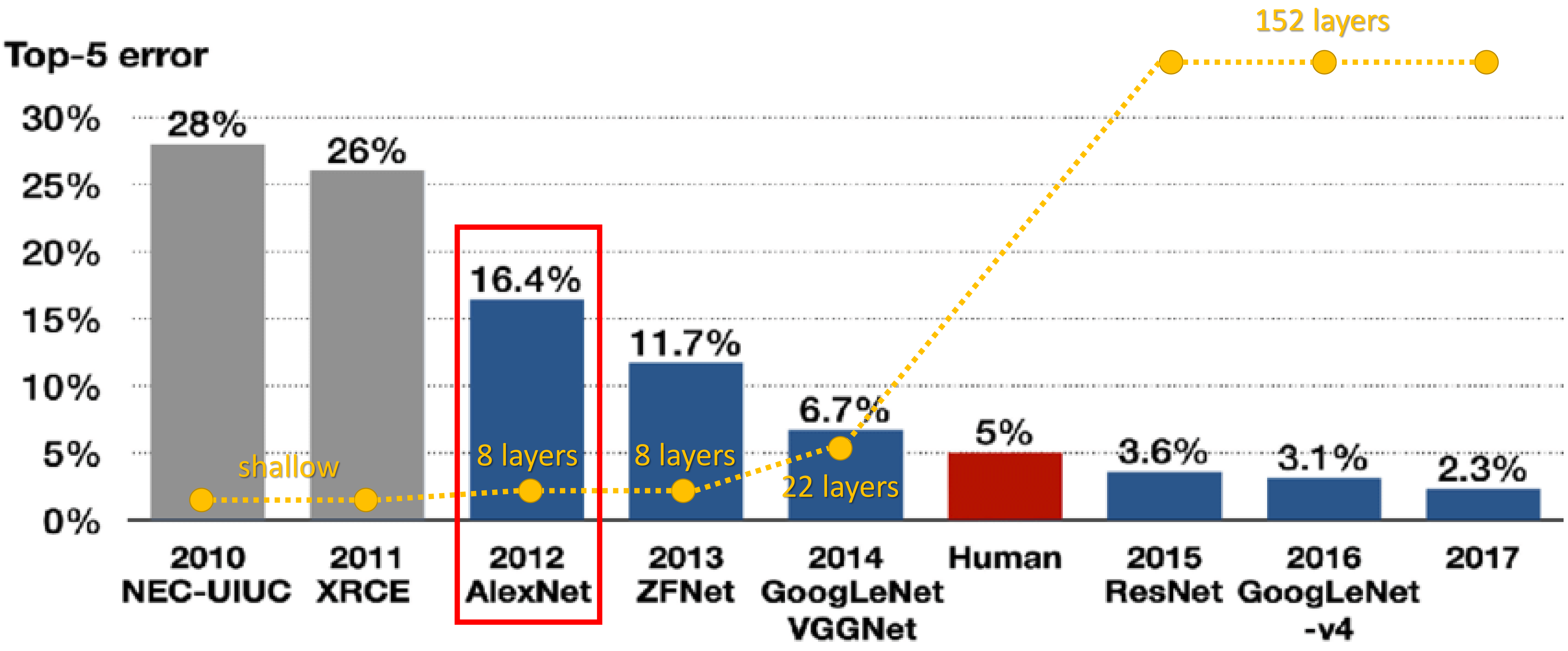
Also....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



First CNN-based winner

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *AlexNet*

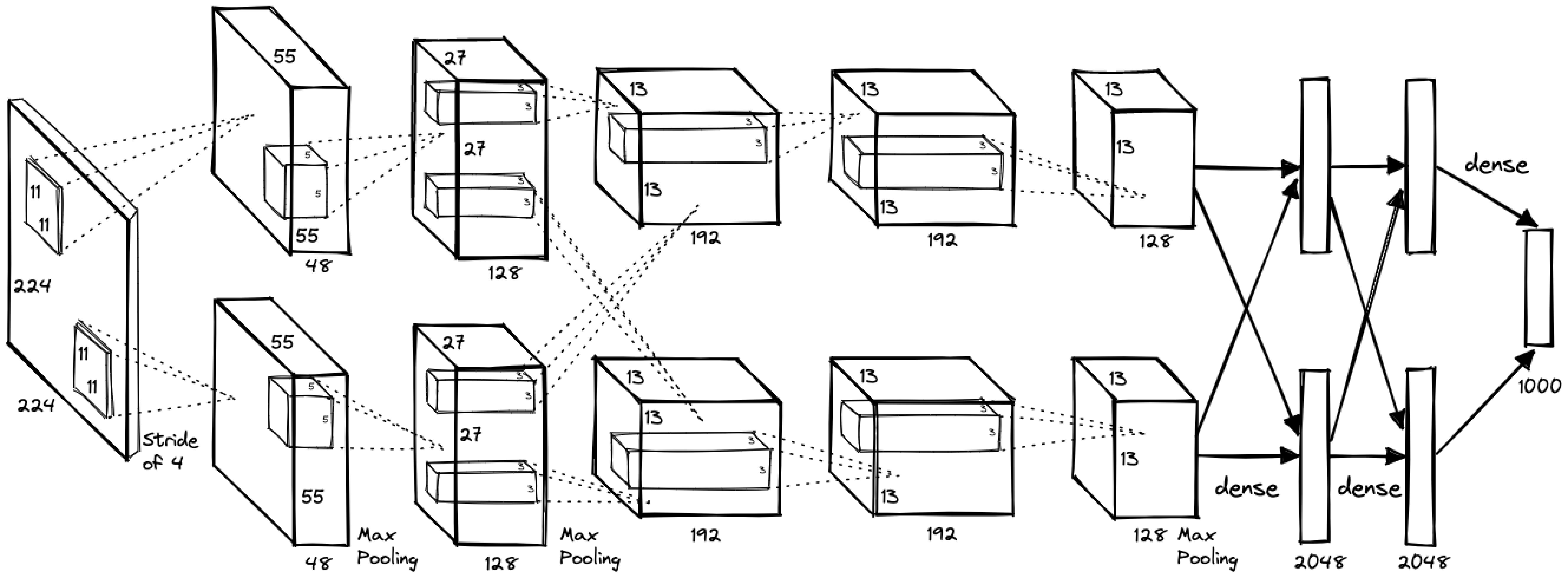
The architecture of AlexNet is composed of eight layers, including five convolutional layers, two fully connected layers, and a softmax output layer. The network takes an input image of size 227 x 227 pixels and produces a probability distribution over 1000 different classes. The main contributions of AlexNet are as follows:

- Use of ReLU activation function: ReLU (Rectified Linear Unit) activation function was used instead of the traditional sigmoid activation function, which significantly reduces the training time and improves the model accuracy.
- Use of overlapping pooling: Max pooling was used with an overlapping window of size 3x3 and a stride of 2, which helped to reduce the spatial dimension of the input and extract useful features.
- Use of dropout regularization: Dropout regularization was applied to the fully connected layers to prevent overfitting, by randomly dropping out some of the units during training.
- Use of data augmentation: Data augmentation techniques such as cropping, flipping, and color shifting were used to increase the diversity of the training data, which improved the generalization ability of the model.

The architecture of AlexNet was much deeper and wider than previous convolutional neural networks, with more parameters and a larger number of neurons in each layer. It was trained on a large-scale dataset of 1.2 million images, which made it possible to learn a rich set of hierarchical features and achieve state-of-the-art accuracy on the ILSVRC challenge.

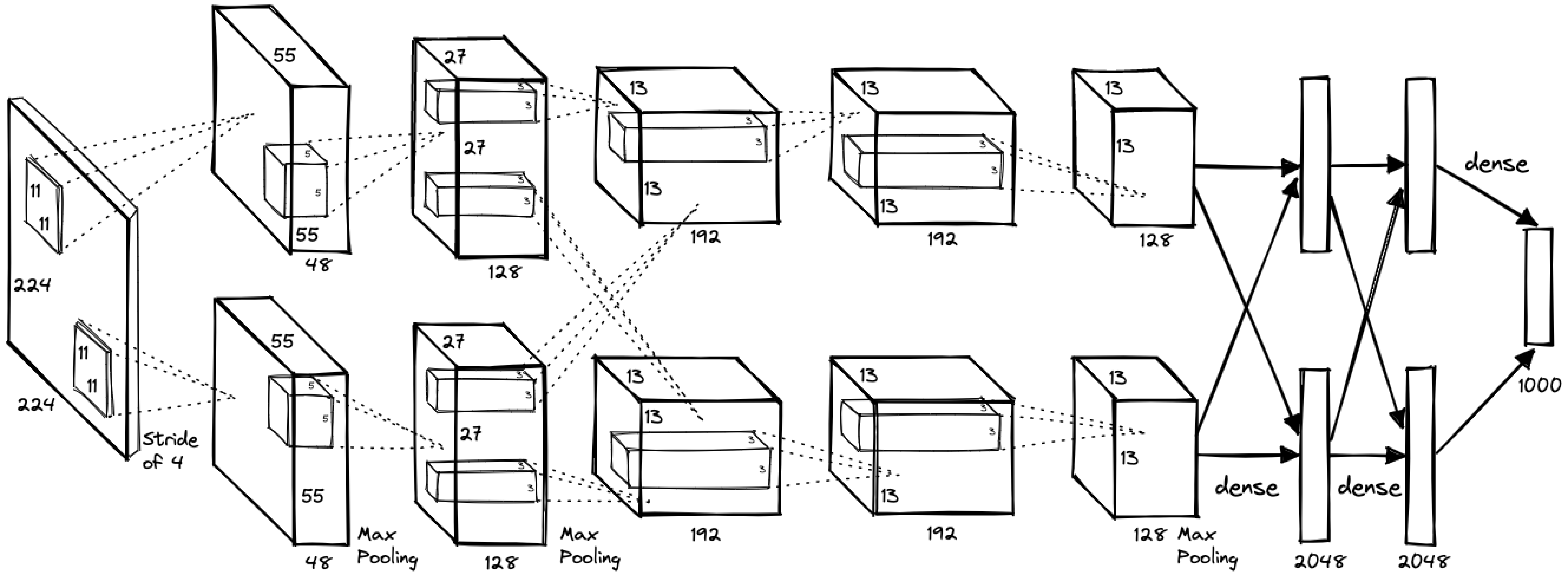
CNN Architectures: *AlexNet*

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



CNN Architectures: *AlexNet*

W’ = (W - F + 2P) / S + 1



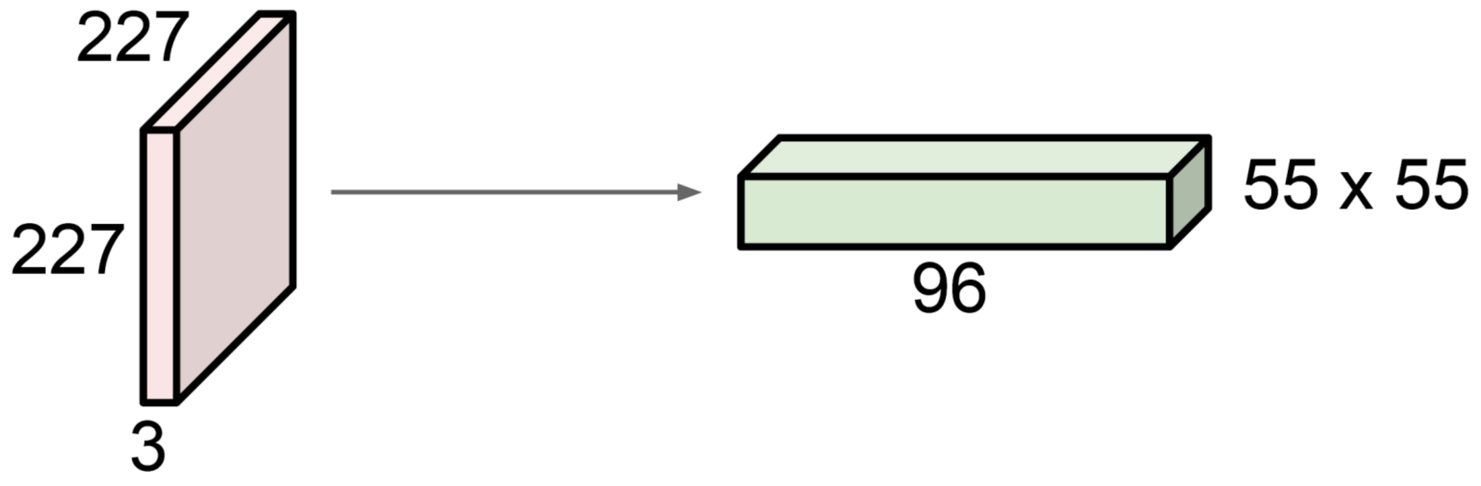
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

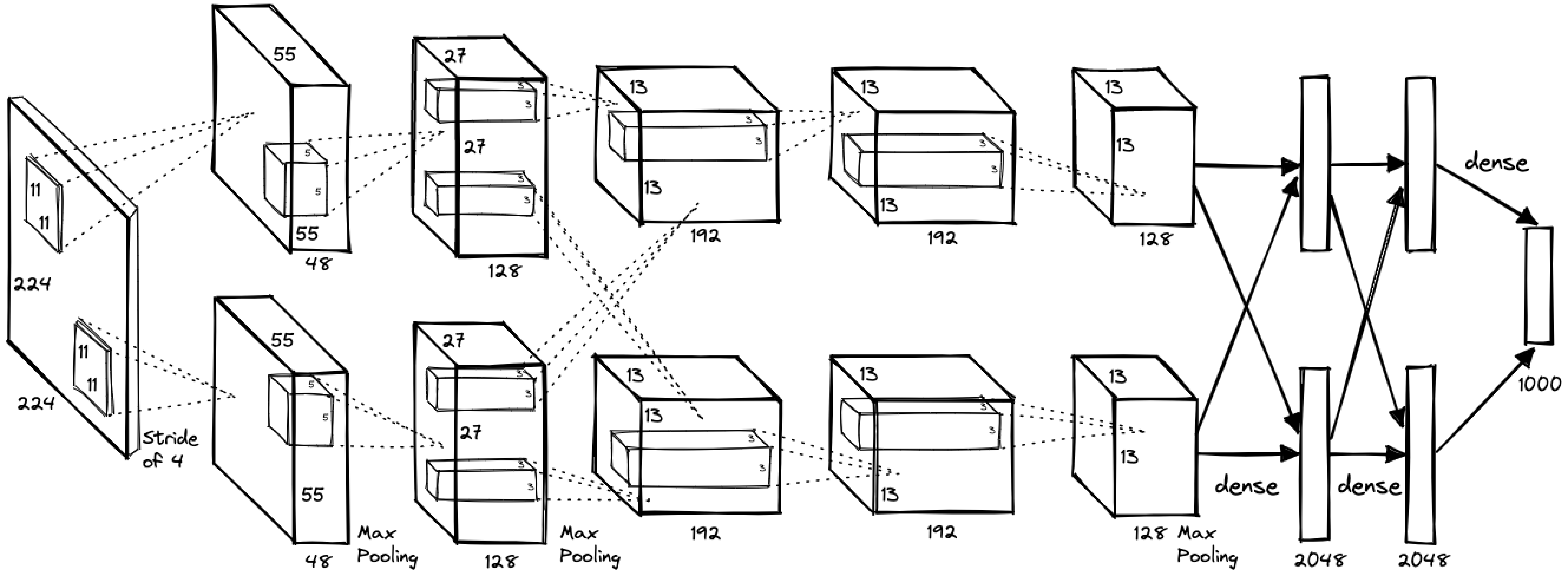
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Output volume **[55x55x96]**



CNN Architectures: *AlexNet*

W’ = (W - F + 2P) / S + 1



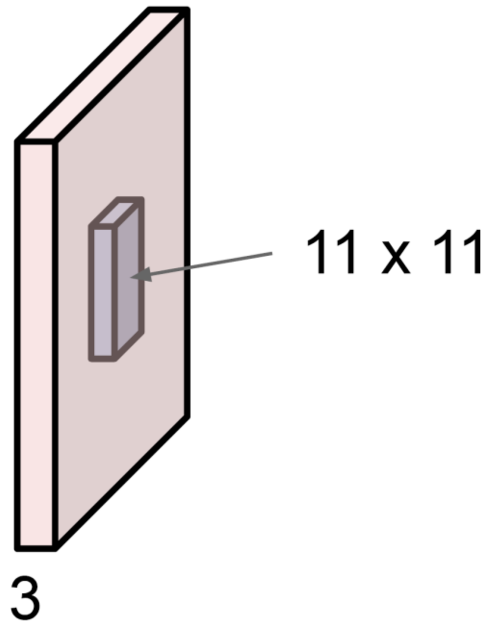
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=> Output volume **[55x55x96]**

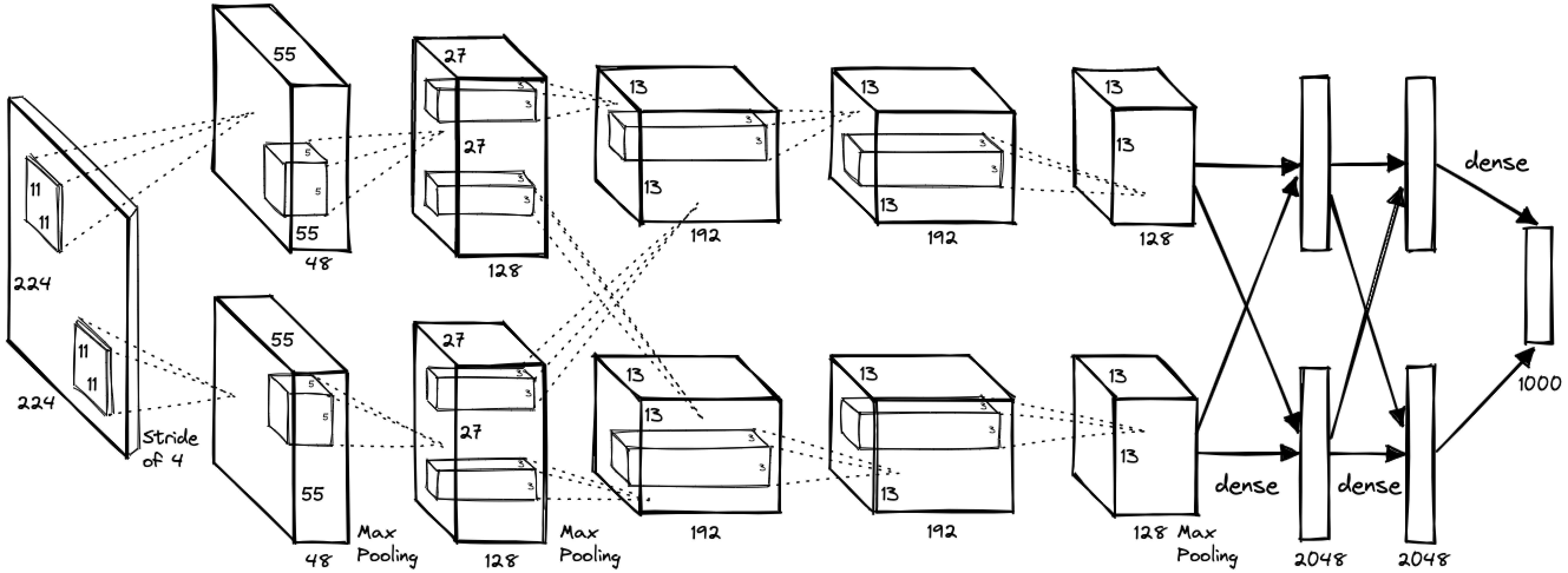
Q: What is the total number of parameters in this layer?

Parameters: (11*11*3 + 1)*96 = **35K**



CNN Architectures: *AlexNet*

W’ = (W - F + 2P) / S + 1

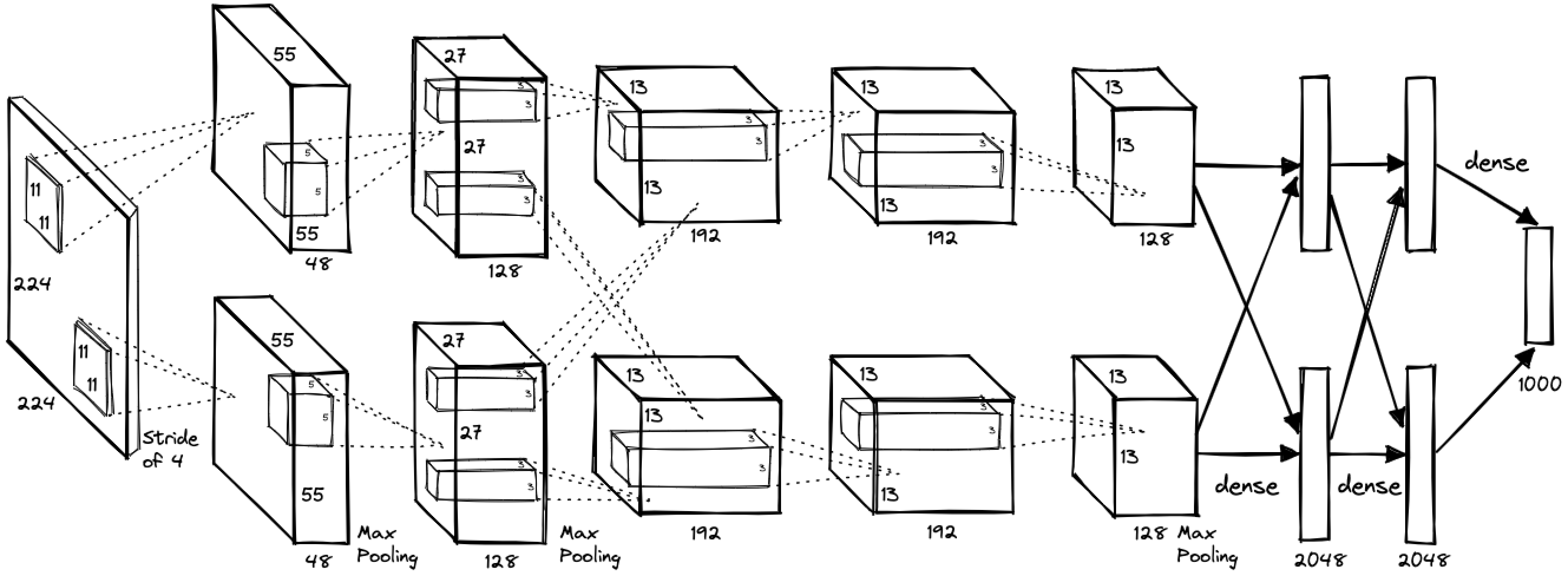


Input: 227x227x3 images
After CONV1: 55x55x96
Second layer (POOL1): 3x3 filters applied at stride 2
Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Output volume: 27x27x96

CNN Architectures: *AlexNet*

$$W' = (W - F + 2P) / S + 1$$



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Q: what is the number of parameters in this layer?

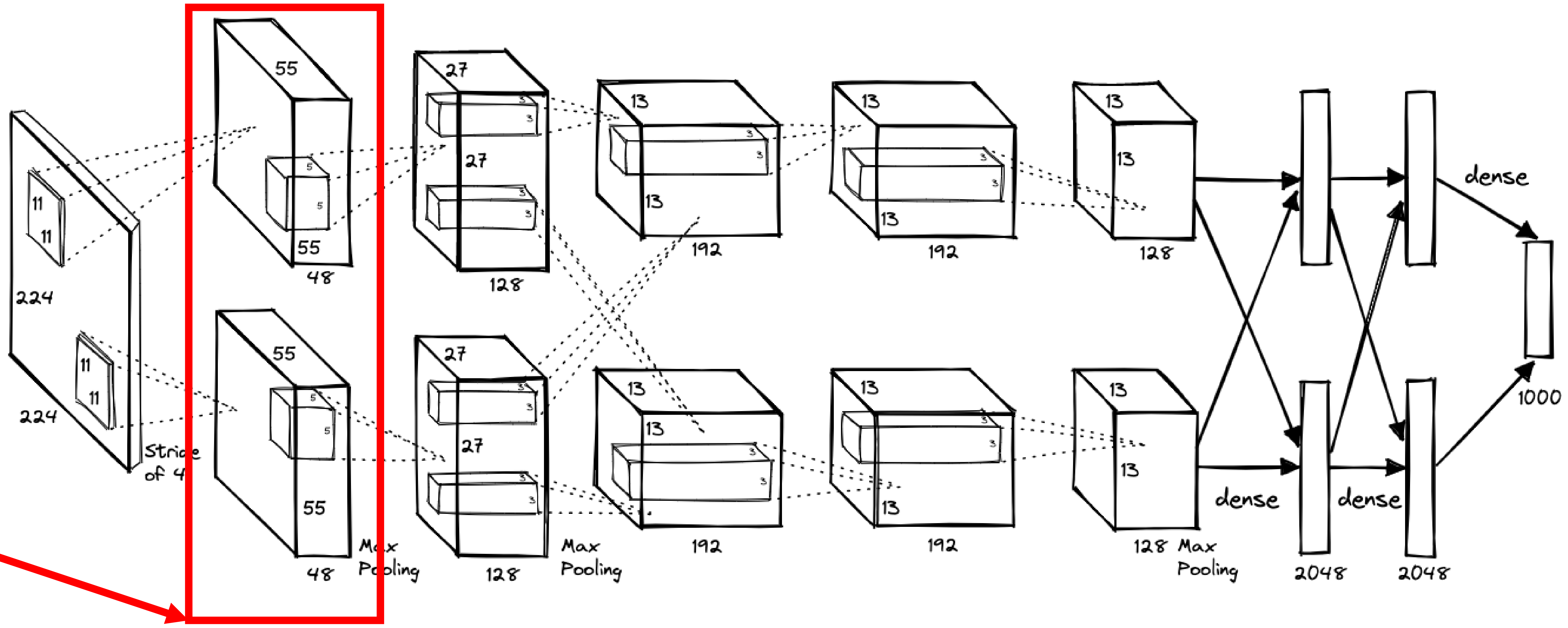
Parameters: 0!

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

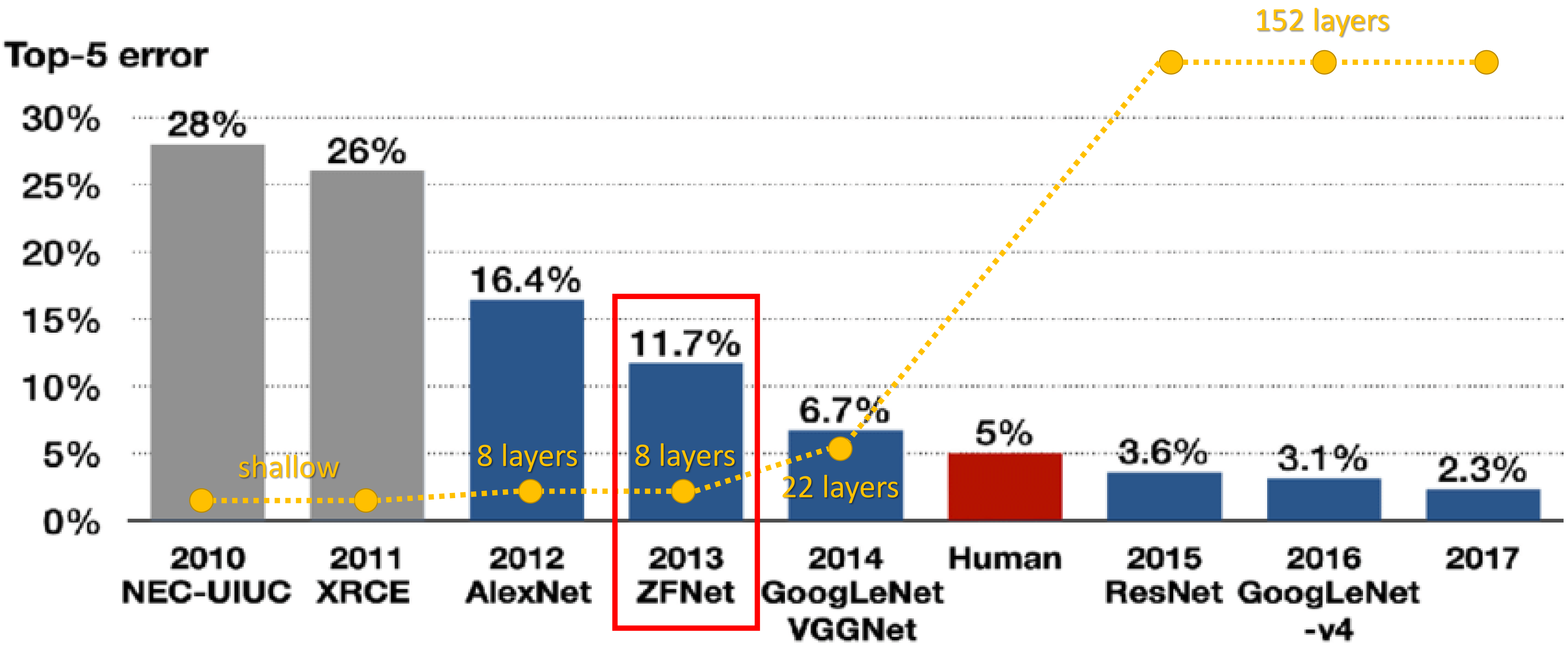
CNN Architectures: *AlexNet*

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



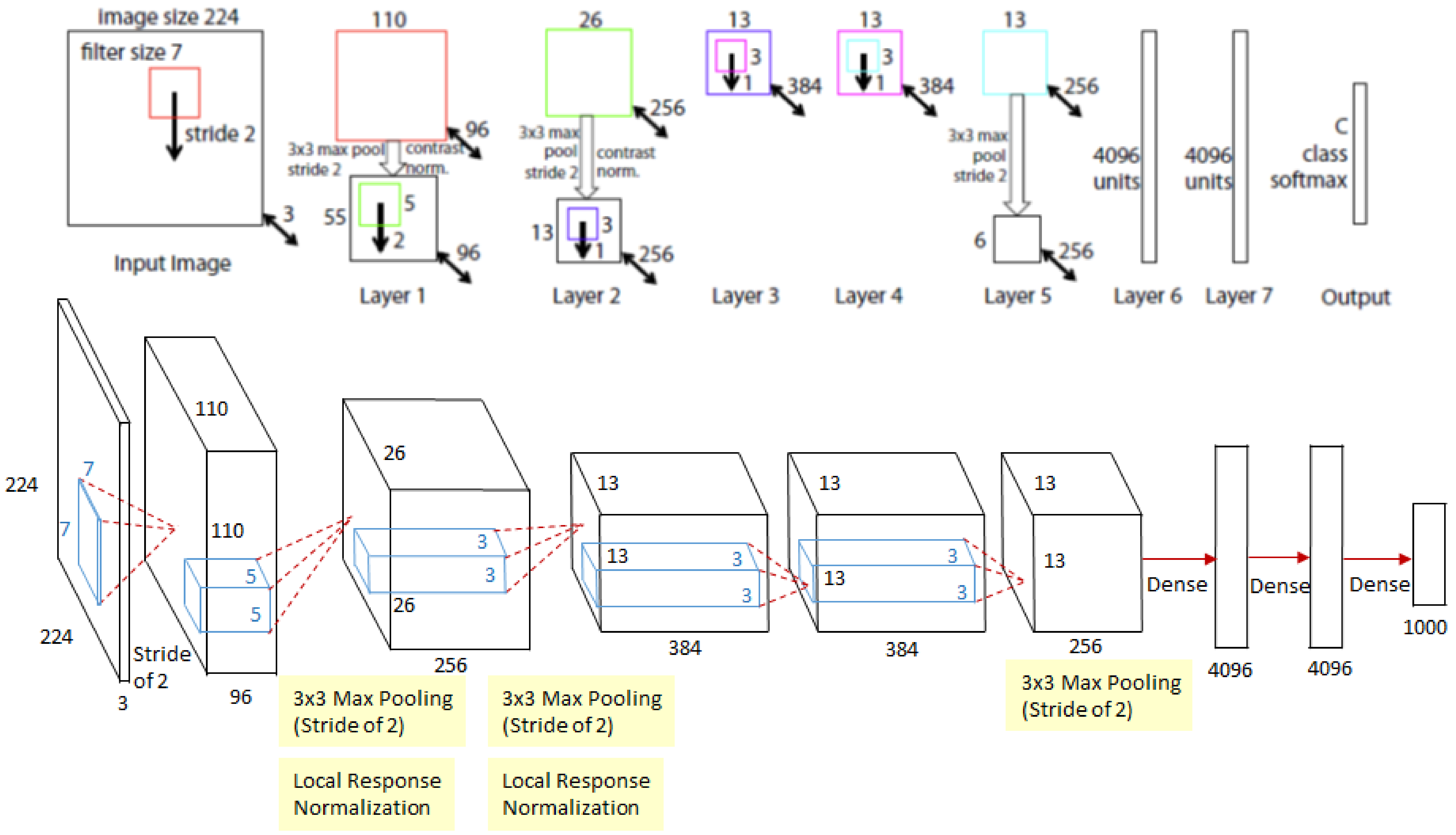
[55x55x48] x 2

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet: Improved hyperparameters over AlexNet

CNN Architectures: *ZFNet*



AlexNet but:

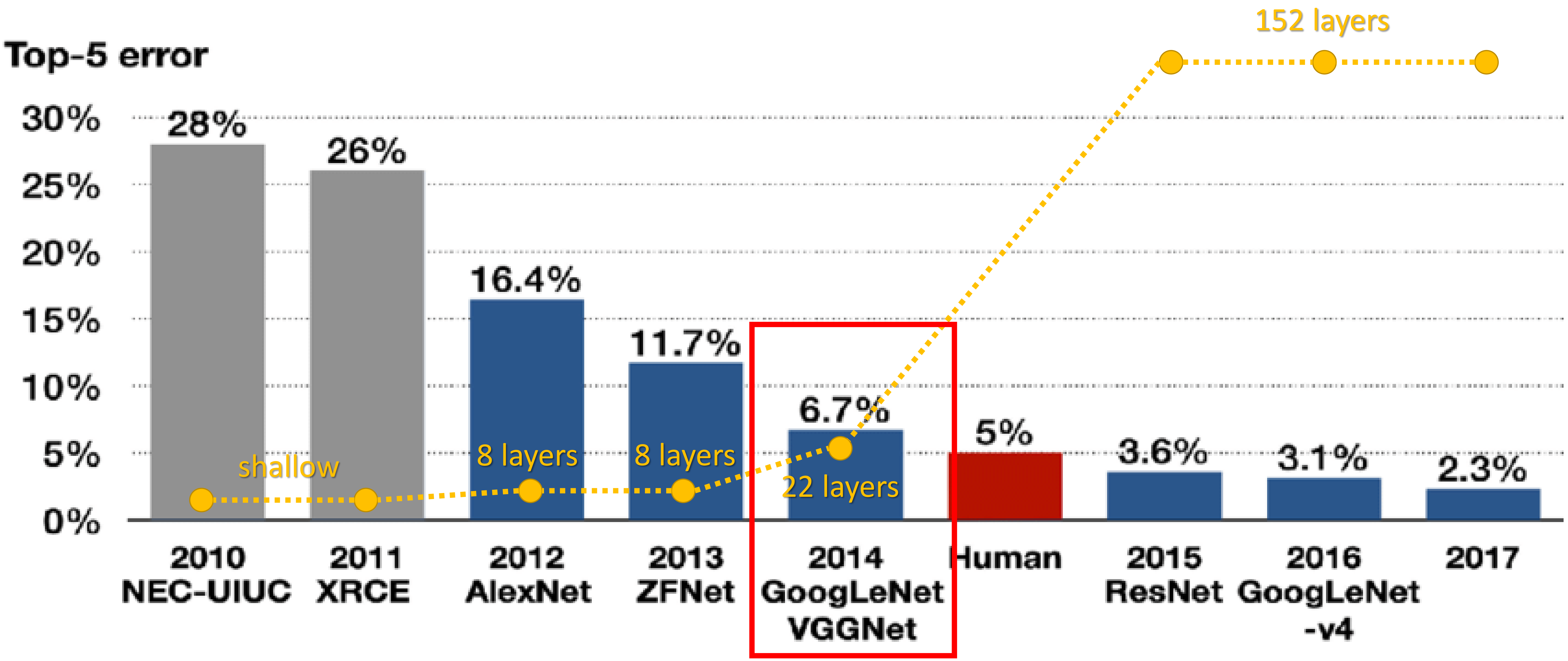
CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

[Zeiler and Fergus, 2013]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Deeper Networks

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *VGGNet*

AlexNet
- Softmax
- FC 1000
- FC 4096
- FC 4096
- Pool
- 3x3 conv, 256
- 3x3 conv, 384
- Pool
- 3x3 conv, 384
- Pool
- 5x5 conv, 256
- 11x11 conv, 96
- Input

VGG16
- Softmax
- FC 1000
- FC 4096
- FC 4096
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- Pool
- 3x3 conv, 128
- 3x3 conv, 128
- Pool
- 3x3 conv, 64
- 3x3 conv, 64
- Input

VGG19
- Softmax
- FC 1000
- FC 4096
- FC 4096
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- Pool
- 3x3 conv, 128
- 3x3 conv, 128
- Pool
- 3x3 conv, 64
- 3x3 conv, 64
- Input

Small filters, Deeper networks
8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2
11.7% top 5 error in ILSVRC’13 (ZFNet)
-> 7.3% top 5 error in ILSVRC’14

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

[Simonyan and Zisserman, 2014]

CNN Architectures: *VGGNet*

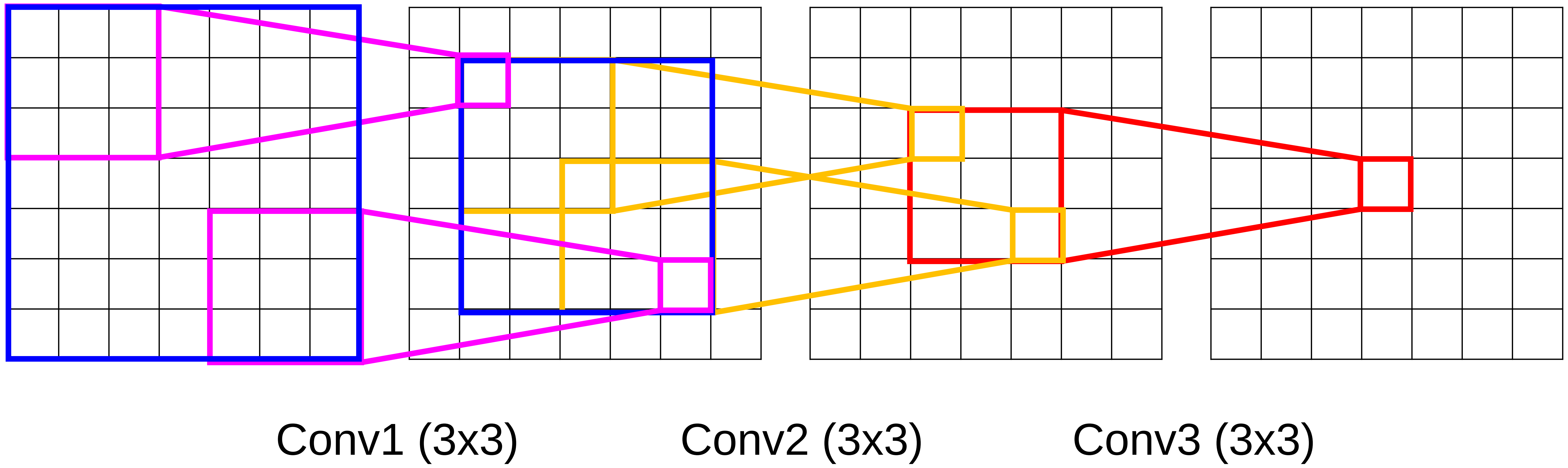
VGG16

- Softmax
- FC 1000
- FC 4096
- FC 4096
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- Pool
- 3x3 conv, 128
- 3x3 conv, 128
- Pool
- 3x3 conv, 64
- 3x3 conv, 64
- Input

VGG19

- Softmax
- FC 1000
- FC 4096
- FC 4096
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- Pool
- 3x3 conv, 128
- 3x3 conv, 128
- Pool
- 3x3 conv, 64
- 3x3 conv, 64
- Input

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



[Simonyan and Zisserman, 2014]

CNN Architectures: *VGGNet*

Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input

VGG16

Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input

VGG19

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities
And fewer parameters: $3 * (3_2 C_2)$ vs. $7_2 C_2$ for C channels per layer

[Simonyan and Zisserman, 2014]

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *VGGNet*

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0

CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

CNN Architectures: *VGGNet*

Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input

VGG16

Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input

VGG19

Details:

- ILSVRC’14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

[Simonyan and Zisserman, 2014]

CNN Architectures: *GoogLeNet*

- ILSVRC’14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
    - 12x less than AlexNet
    - 27x less than VGG-16
- Efficient “Inception” module
- No FC layers

[Szegedy et al., 2014]

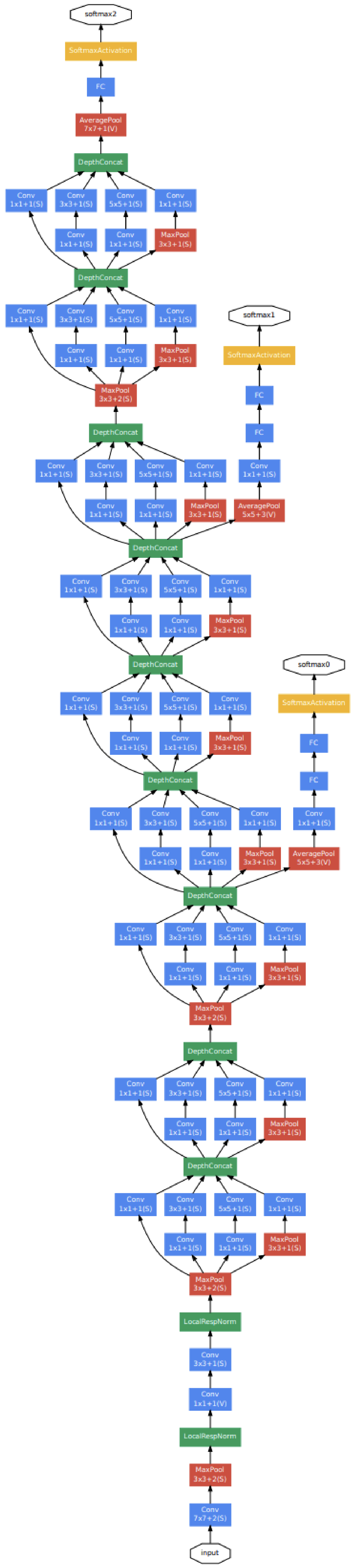
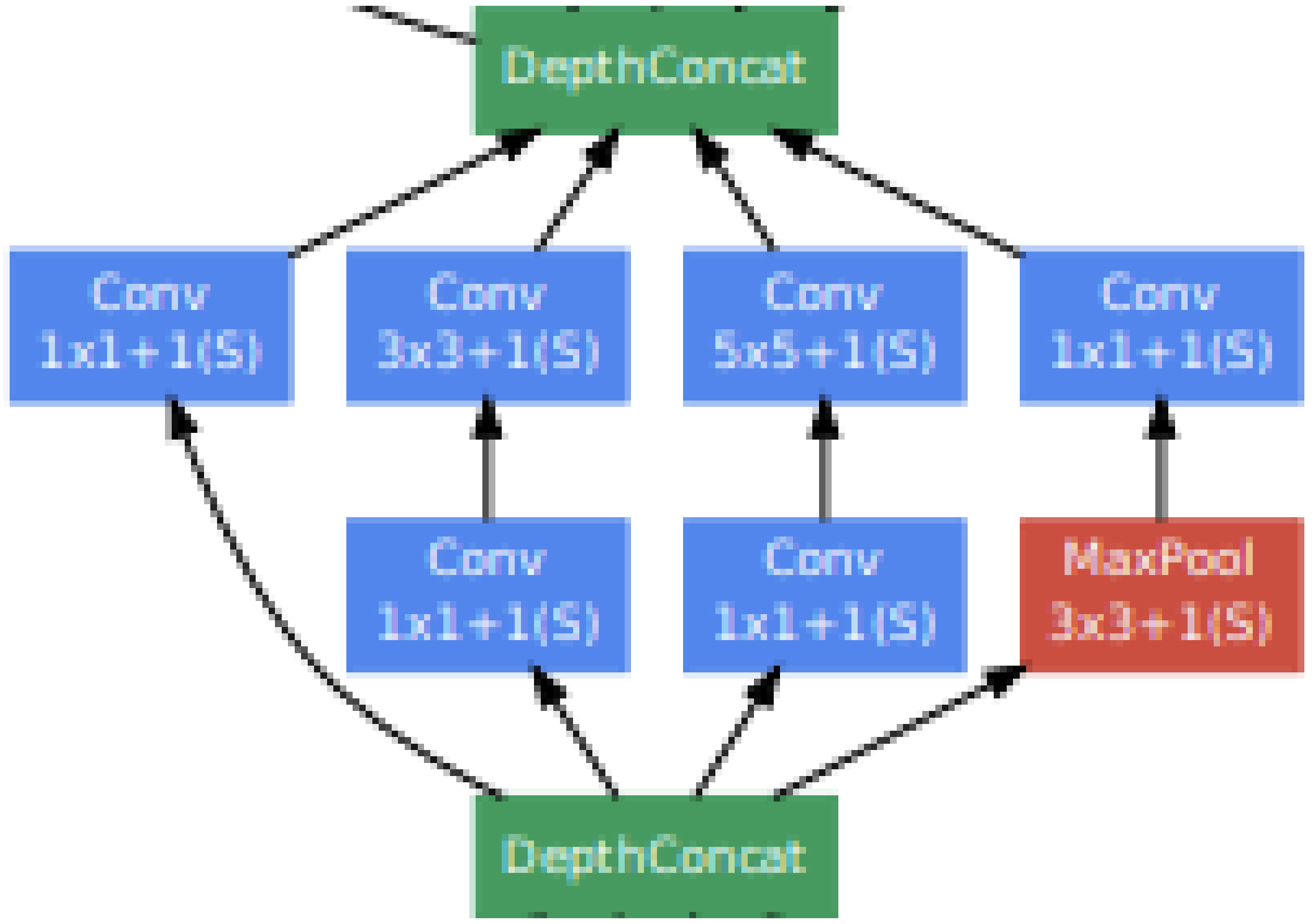
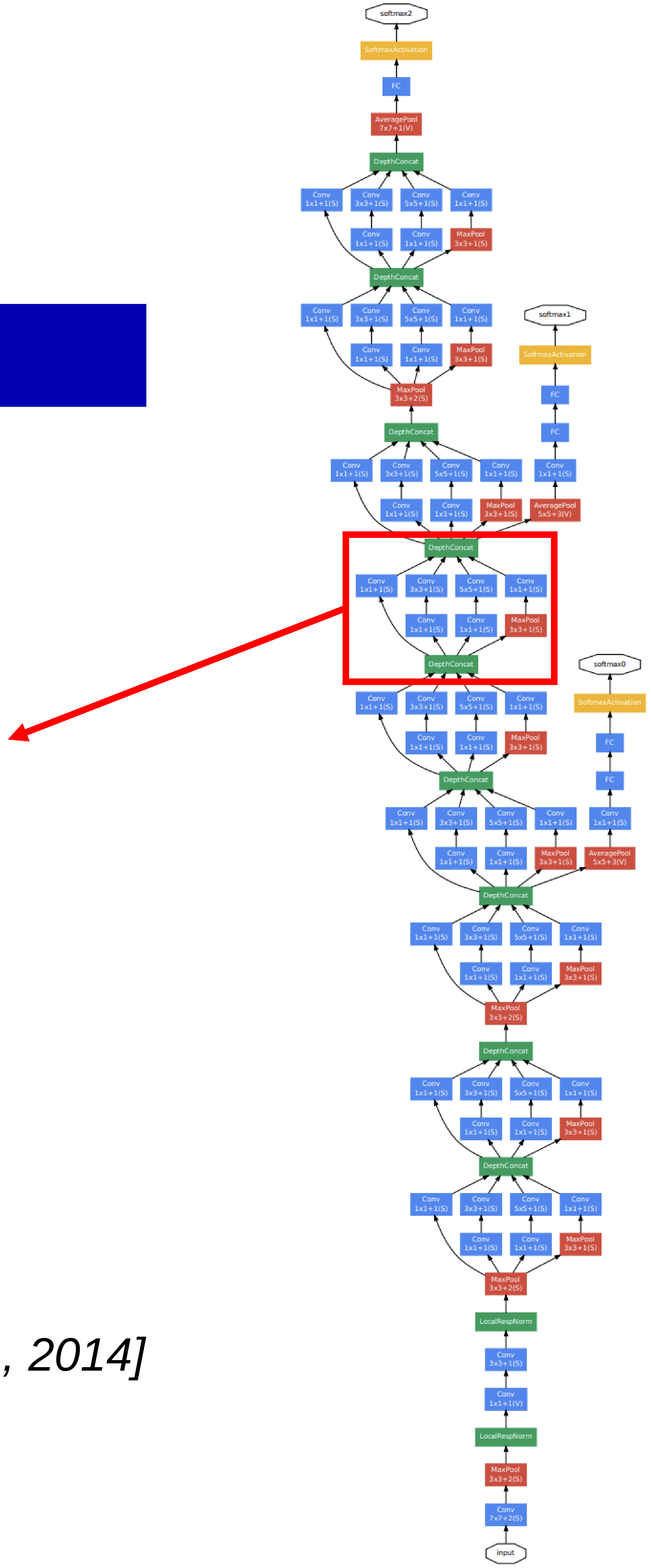


Figure 3: GoogLeNet network with all the bells and whistles

CNN Architectures: *GoogLeNet*

“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other





[Szegedy et al., 2014]

Co-financed by the European Union
Connecting Europe Facility

CNN Architectures: *GoogLeNet*

GoogleNet, also known as Inception v1, is a deep convolutional neural network designed for image classification, proposed by researchers at Google in 2014. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014 with a significant margin and marked a major breakthrough in the field of computer vision.

The architecture of GoogleNet is composed of 22 layers, including convolutional layers, pooling layers, and fully connected layers. The network takes an input image of size 224 x 224 pixels and produces a probability distribution over 1000 different classes.

The architecture of GoogleNet is much deeper and wider than previous convolutional neural networks, with more parameters and a larger number of neurons in each layer. It was trained on a large-scale dataset of 1.2 million images, which made it possible to learn a rich set of hierarchical features and achieve state-of-the-art accuracy on the ILSVRC challenge. The success of GoogleNet inspired a series of follow-up models, such as Inception v2, v3, v4, and Inception-ResNet.

CNN Architectures: *GoogLeNet*

The main contributions of GoogleNet are as follows:

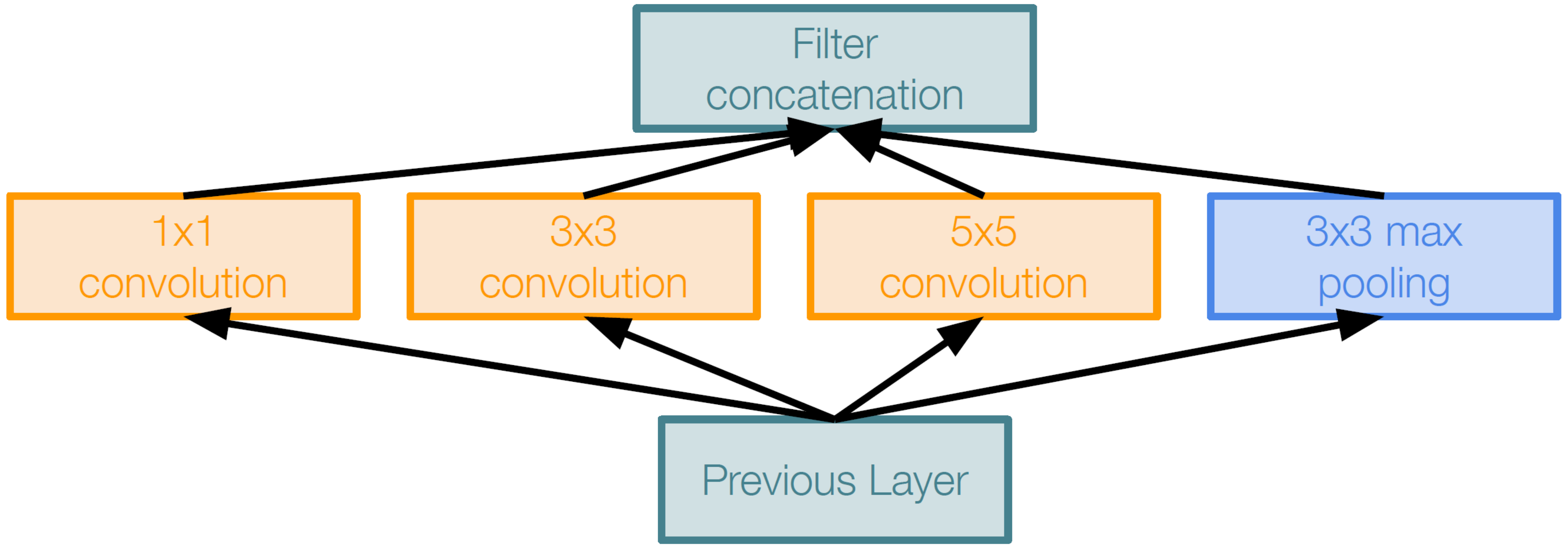
- **Use of Inception modules:** The Inception module is a novel building block that consists of a combination of different convolutions with different kernel sizes (1x1, 3x3, 5x5), as well as a max pooling layer. The Inception module allows the network to capture features at multiple scales and reduces the number of parameters.
- **Use of global average pooling:** Global average pooling is used to replace the fully connected layer, which reduces the number of parameters and improves the generalization ability of the model.
- **Use of auxiliary classifiers:** Auxiliary classifiers were added to the network to provide intermediate supervision and prevent overfitting. The loss from the auxiliary classifiers is added to the main loss, which helps to improve the training process.
- **Use of batch normalization:** Batch normalization was applied to the input of each non-linear activation function, which helps to reduce the internal covariate shift and improves the training process.

CNN Architectures: *GoogLeNet*

The main contributions of GoogleNet are as follows:

- **Use of Inception modules:** The Inception module is a novel building block that consists of a combination of different convolutions with different kernel sizes (1x1, 3x3, 5x5), as well as a max pooling layer. The Inception module allows the network to capture features at multiple scales and reduces the number of parameters.
- **Use of global average pooling:** Global average pooling is used to replace the fully connected layer, which reduces the number of parameters and improves the generalization ability of the model.
- **Use of auxiliary classifiers:** Auxiliary classifiers were added to the network to provide intermediate supervision and prevent overfitting. The loss from the auxiliary classifiers is added to the main loss, which helps to improve the training process.
- **Use of batch normalization:** Batch normalization was applied to the input of each non-linear activation function, which helps to reduce the internal covariate shift and improves the training process.

CNN Architectures: *GoogLeNet*



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

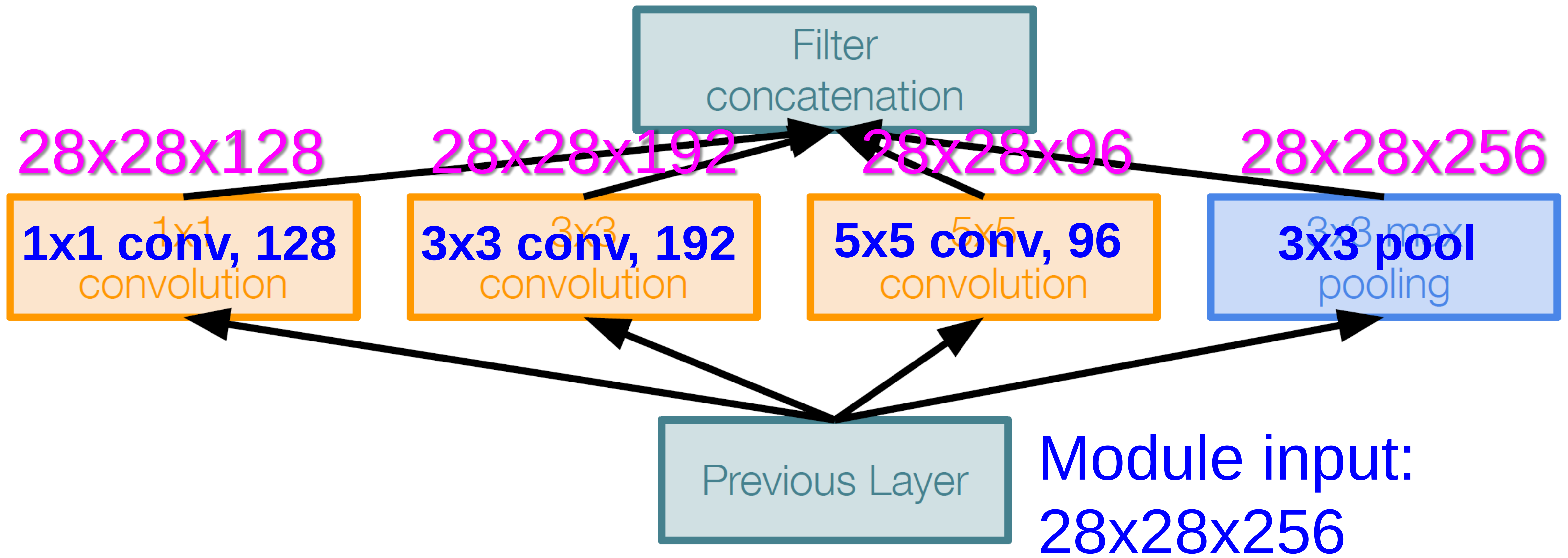
Q: What is the problem with this?

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *GoogLeNet*

28x28x(128+192+96+256) = 28x28x672



Filter concatenation

28x28x128
28x28x192
28x28x96
28x28x256

1x1 convolution
1x1 conv, 128

3x3 convolution
3x3 conv, 192

5x5 convolution
5x5 conv, 96

3x3 max pooling
3x3 pool

Previous Layer

Module input:
28x28x256

Naive Inception module

Q: What is the problem with this?

Computational complexity

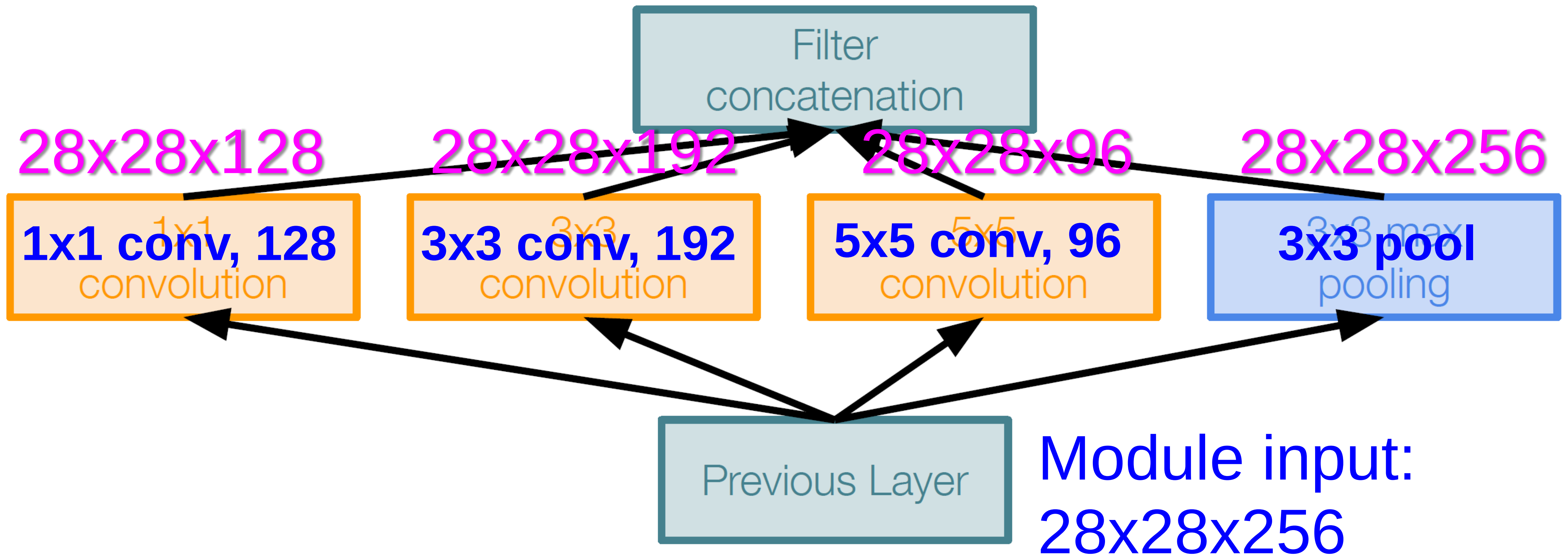
Q1: What are the output sizes of all different filter operations?

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x**192x3x3x256**
[5x5 conv, 96] 28x28x**96x5x5x256**
Total: 854M ops

Very expensive to compute. Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

CNN Architectures: *GoogLeNet*

28x28x(128+192+96+256) = 529k



28x28x128

28x28x192

28x28x96

28x28x256

1x1 conv, 128

3x3 conv, 192

5x5 conv, 96

3x3 pool

Module input:
28x28x256

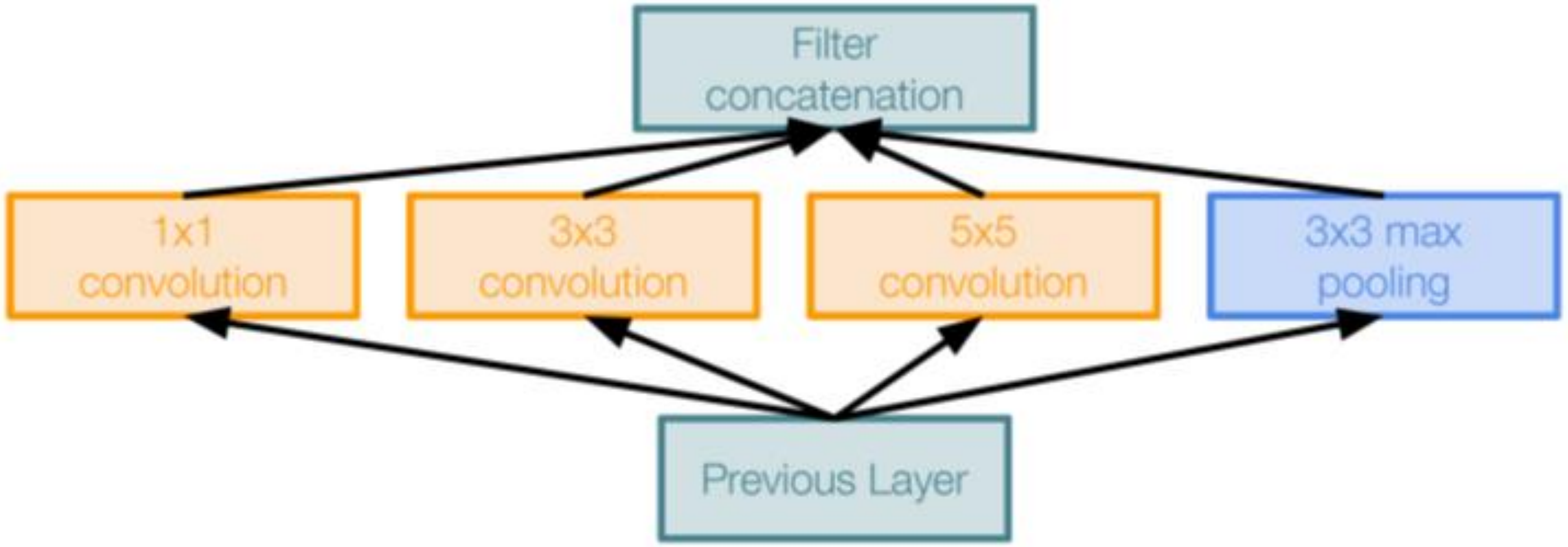
Naive Inception module

Q: What is the problem with this?

Computational complexity

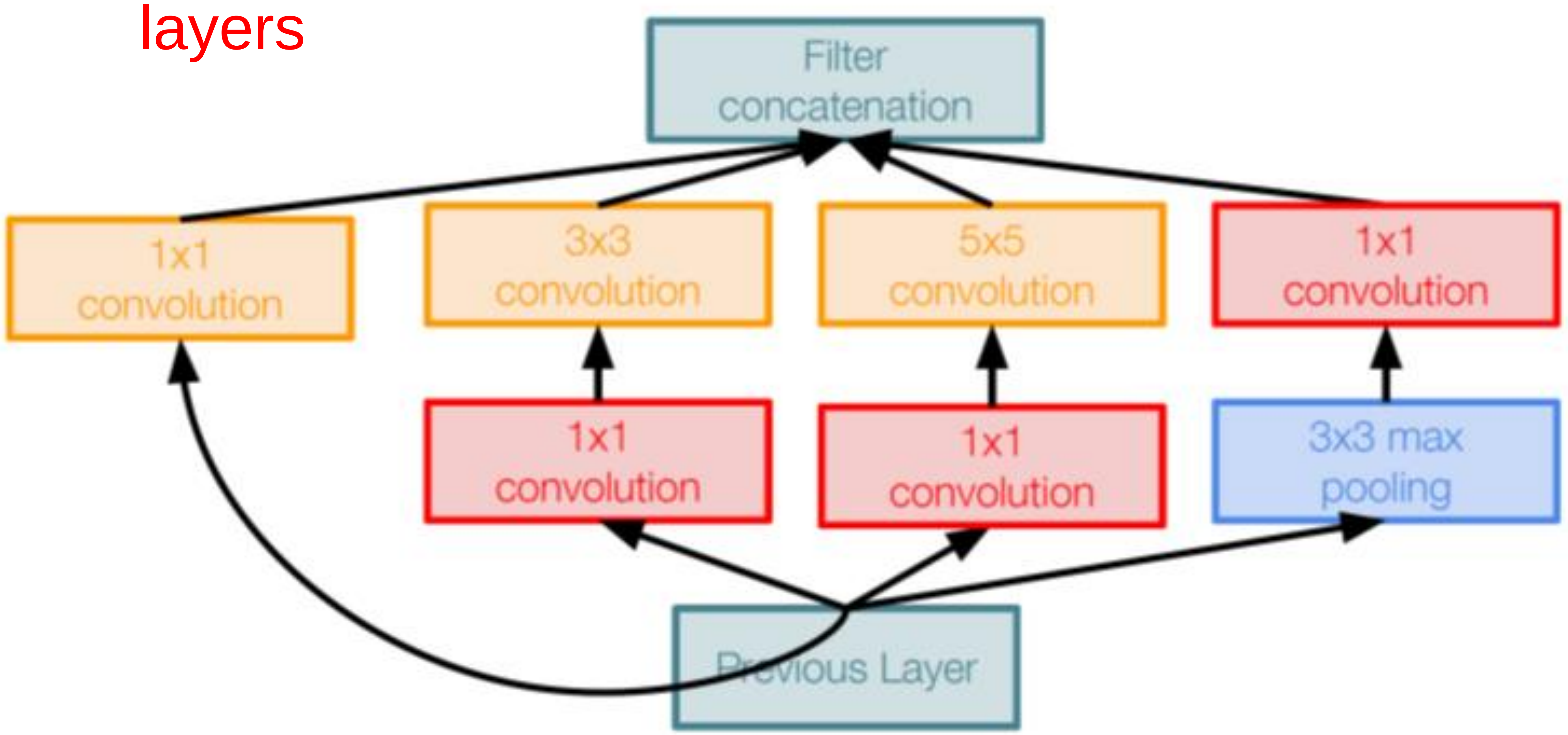
Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature channel size

CNN Architectures: *GoogLeNet*



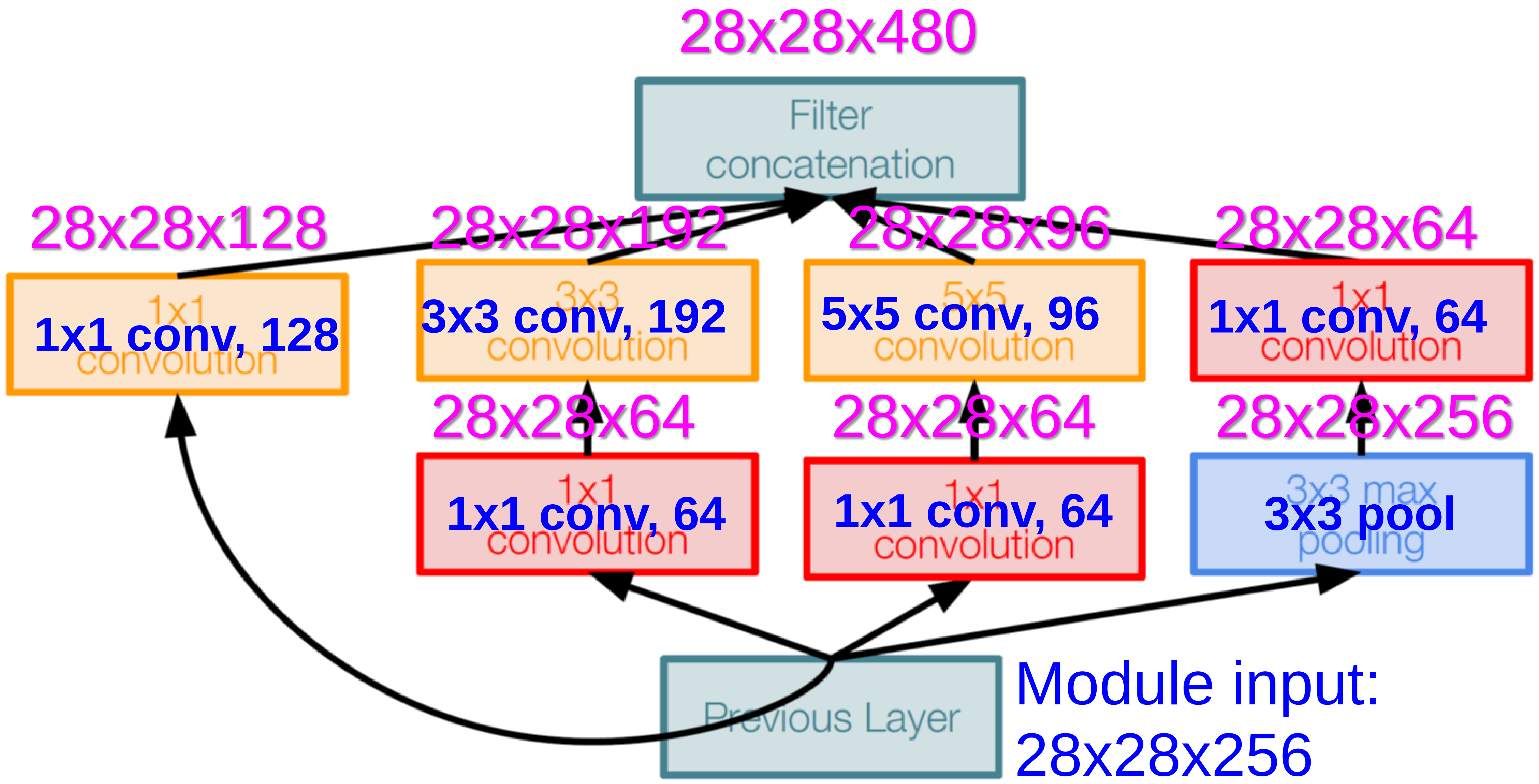
Naive Inception module

1x1 conv “bottleneck”
layers



Inception module with dimension reduction

CNN Architectures: *GoogLeNet*



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

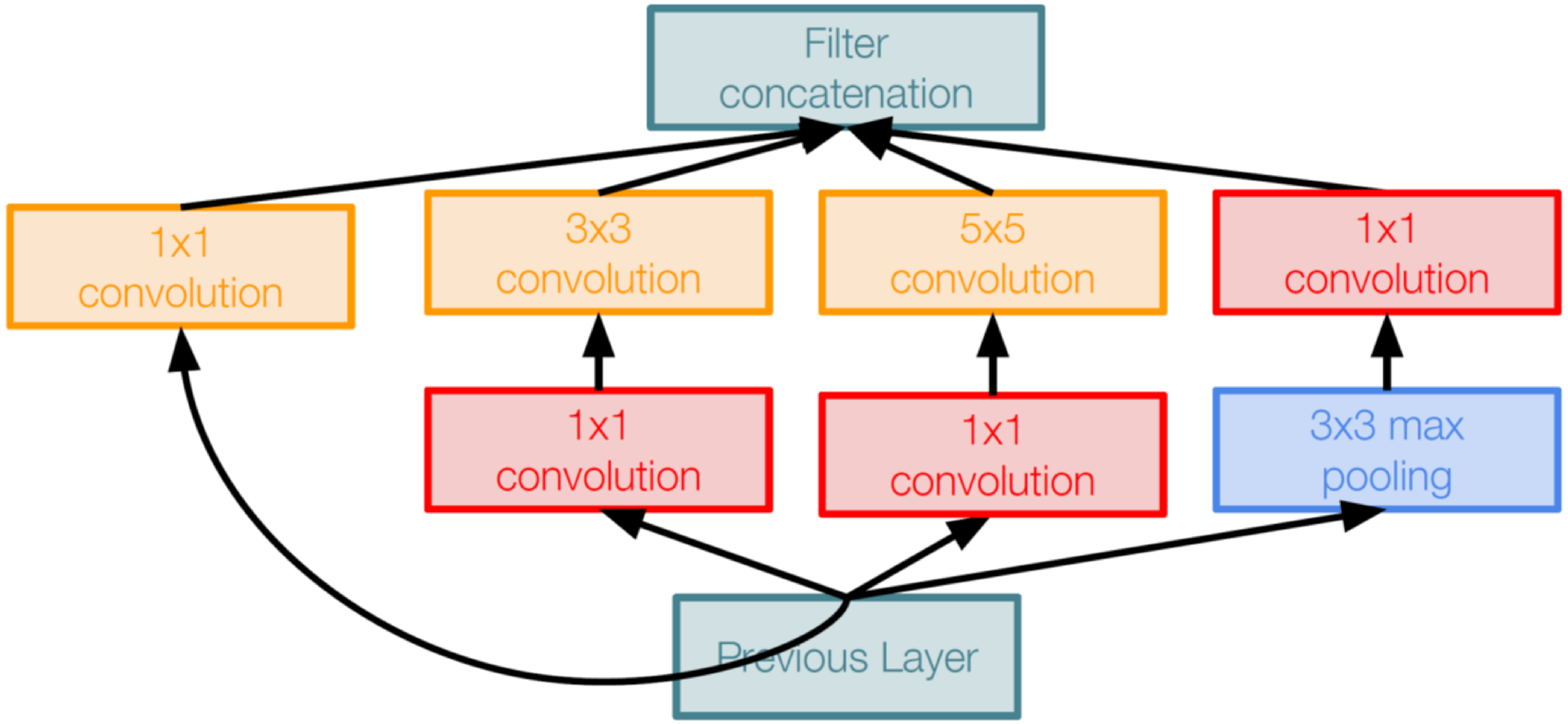
Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Compared to 854M ops for naive version Bottleneck can also reduce depth after pooling layer

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

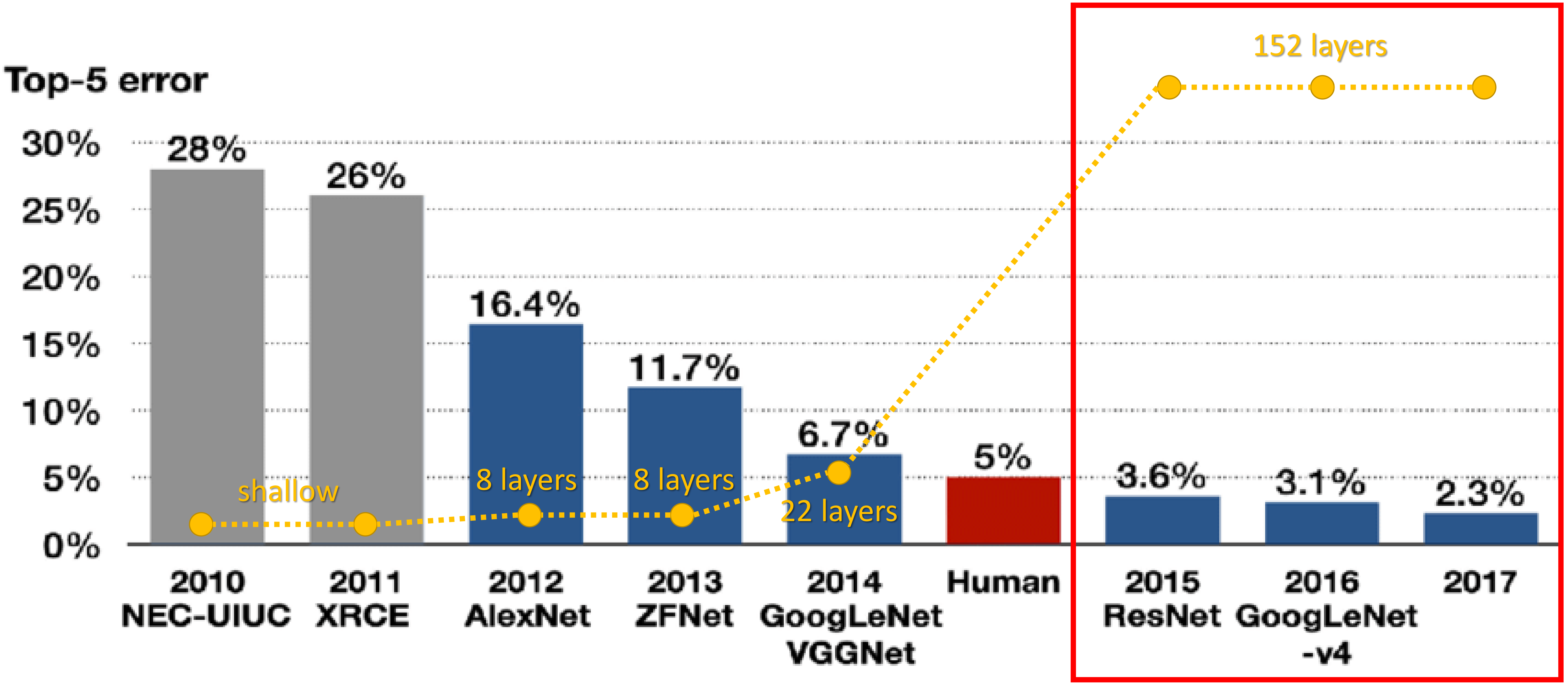
CNN Architectures: *GoogLeNet*



Inception module with dimension reduction

Stack Inception modules with dimension reduction on top of each other

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

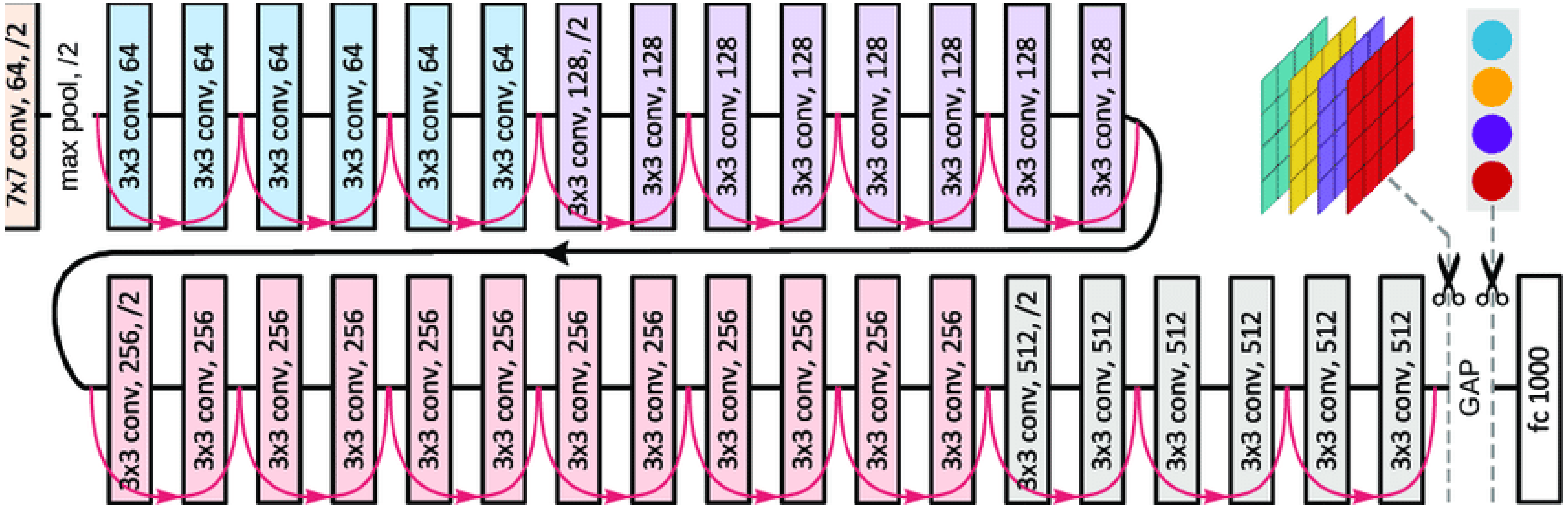


Revolution of depth

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *ResNet*



- 152-layer model for ImageNet
- ILSVRC’15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC’15 and COCO’15!

Very deep networks using residual connections

[He et al., 2015]

CNN Architectures: *ResNet*

ResNet (short for "Residual Network") is a type of deep neural network architecture that was introduced in a 2015 paper by researchers at Microsoft Research. It is one of the most widely used deep learning architectures for image classification tasks.

The main idea behind ResNet is to use "skip connections" to allow information to bypass one or more layers in a deep neural network. In a traditional neural network, each layer processes the output of the previous layer to produce a new set of features. However, as the network gets deeper, it can become increasingly difficult for the network to learn meaningful representations of the input data, and the gradients used for backpropagation can vanish, making it harder to optimize the network.

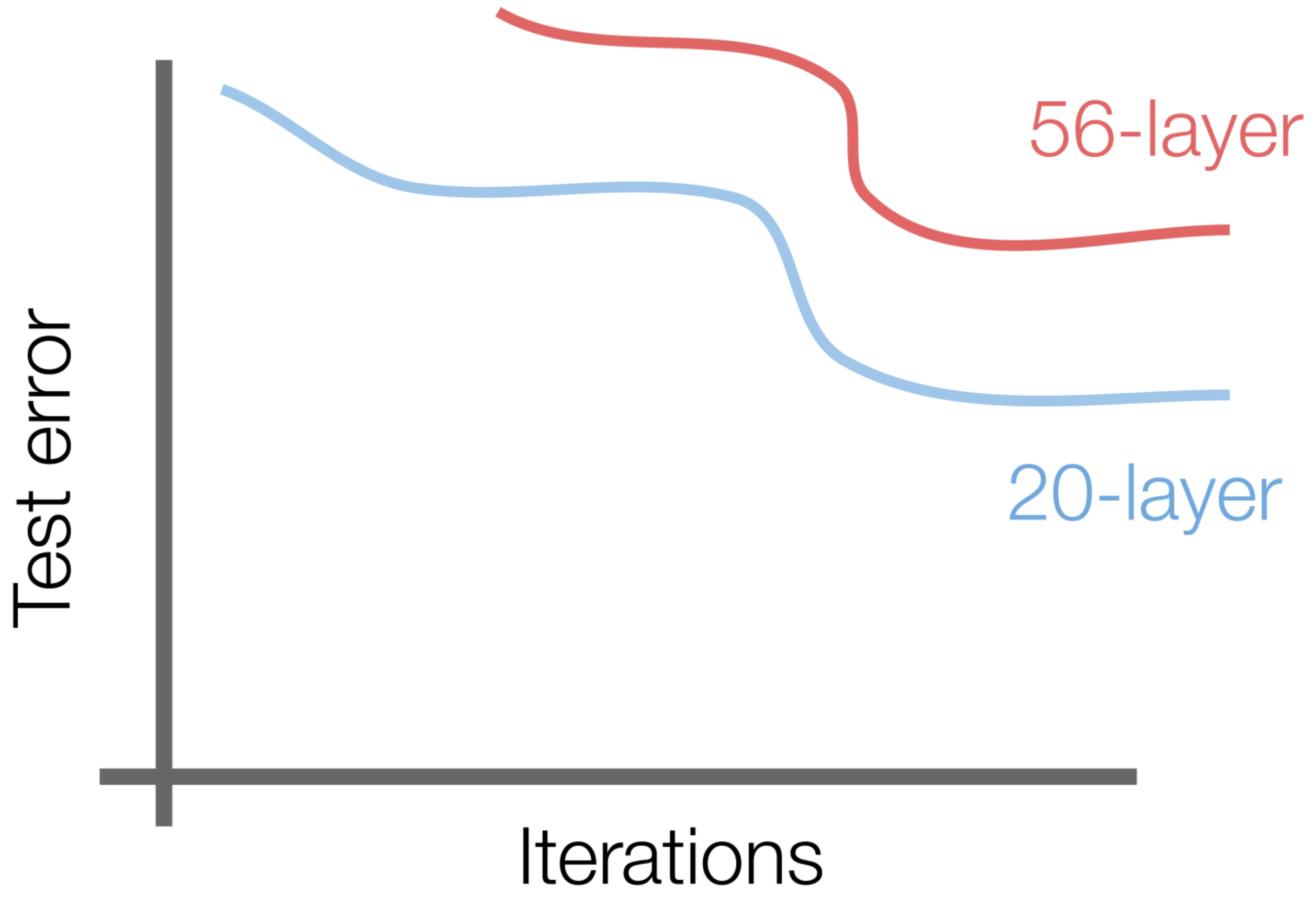
CNN Architectures: *ResNet*

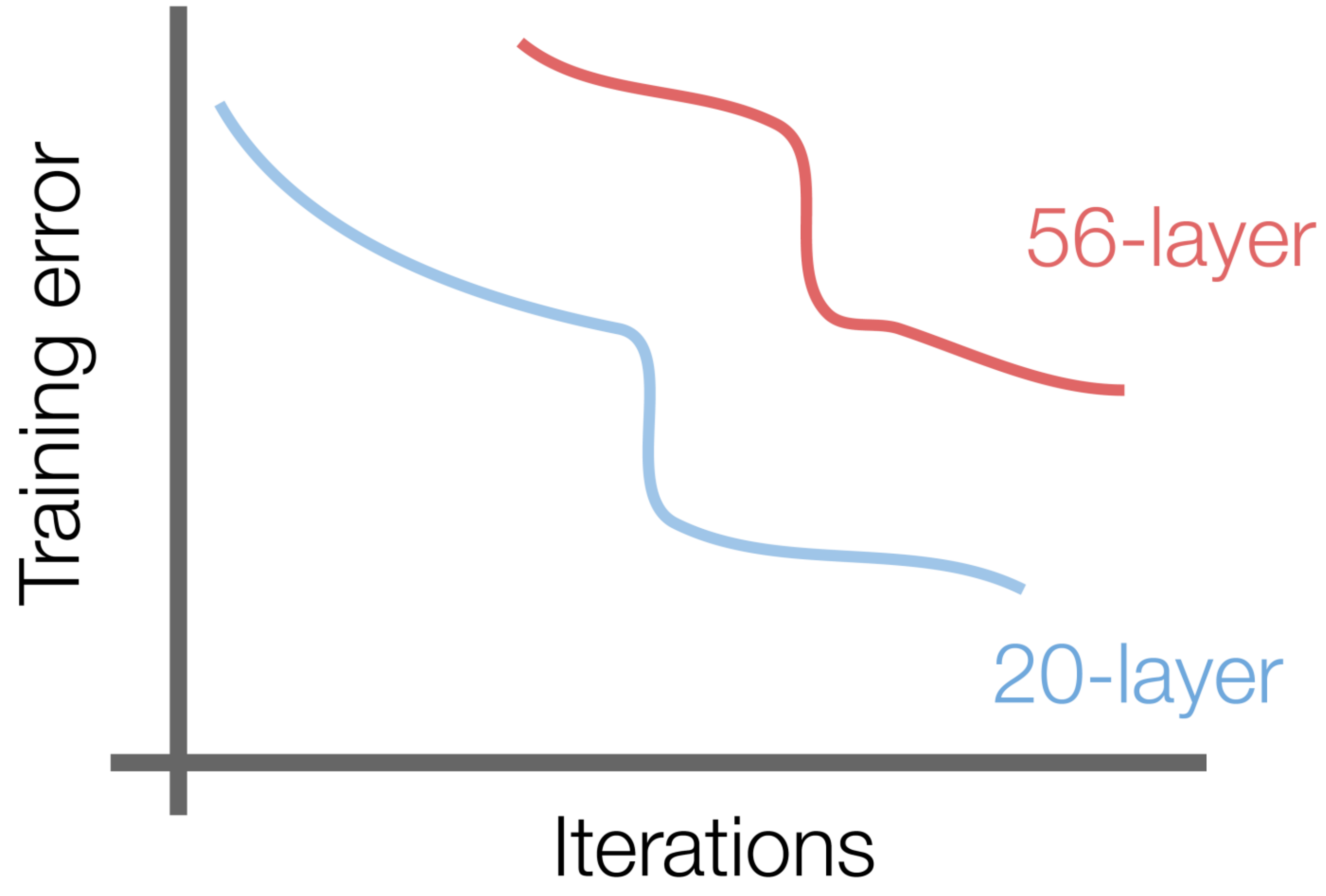
In ResNet, the skip connections allow the output of a layer to be added to the output of one or more layers further along in the network. This creates a "residual" that represents the difference between the input to the layer and its output. By adding this residual to the output of the layer, the network can learn to make small, incremental changes to the input features, rather than trying to learn the entire transformation from scratch. This allows the network to be much deeper without sacrificing performance, and has been shown to improve the accuracy of the network on a variety of image classification tasks.

ResNet comes in several variants, including ResNet-18, ResNet-34, ResNet-50, and so on, each with a different number of layers. The deeper variants (such as ResNet-50 and ResNet-101) are typically used for more complex image classification tasks, while the shallower variants (such as ResNet-18 and ResNet-34) are used when computational resources are limited.

CNN Architectures: *ResNet*

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

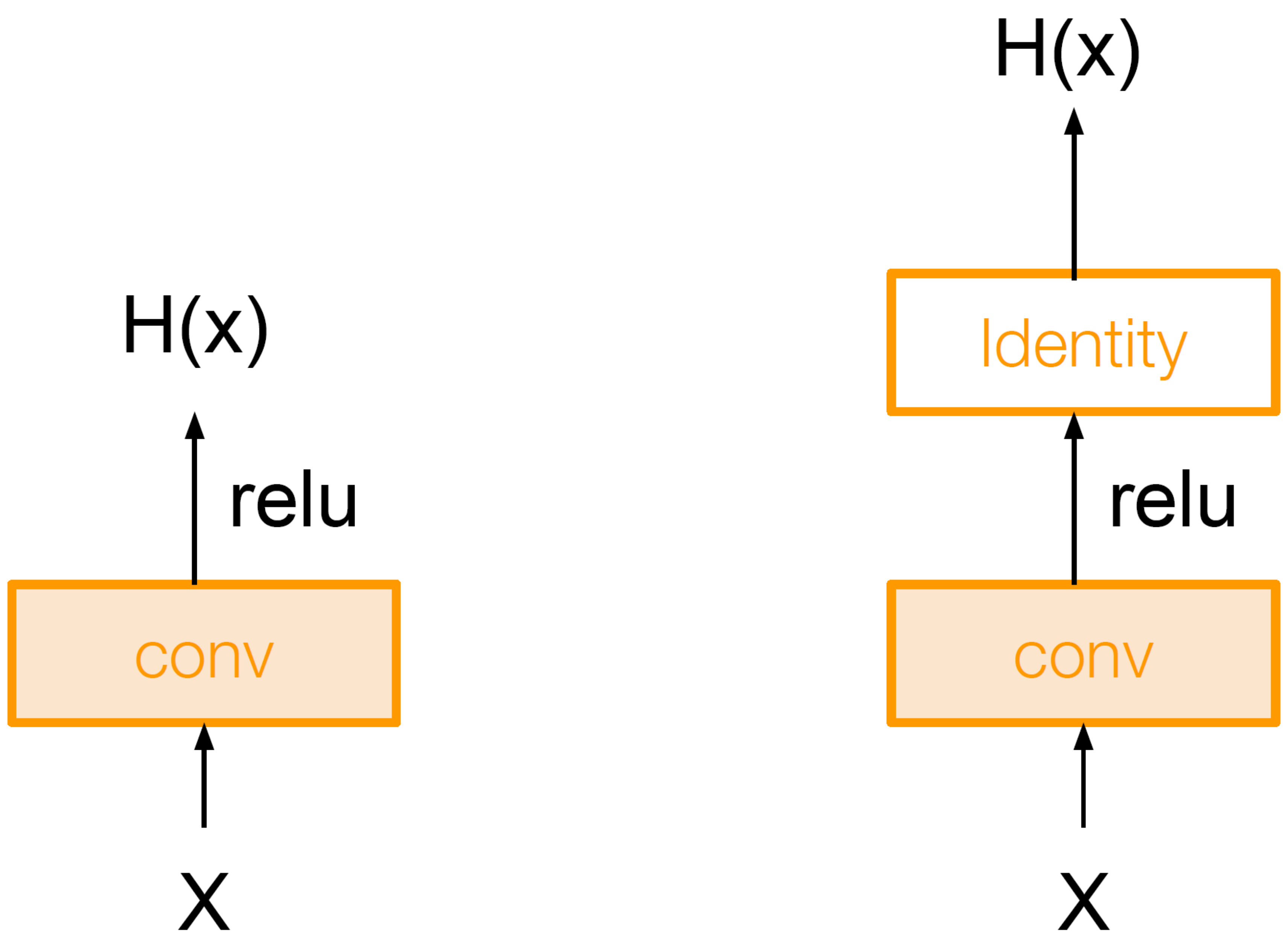
CNN Architectures: *ResNet*

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, **deeper models are harder to optimize**

What should the deeper model learn to be at least as good as the shallower model?

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

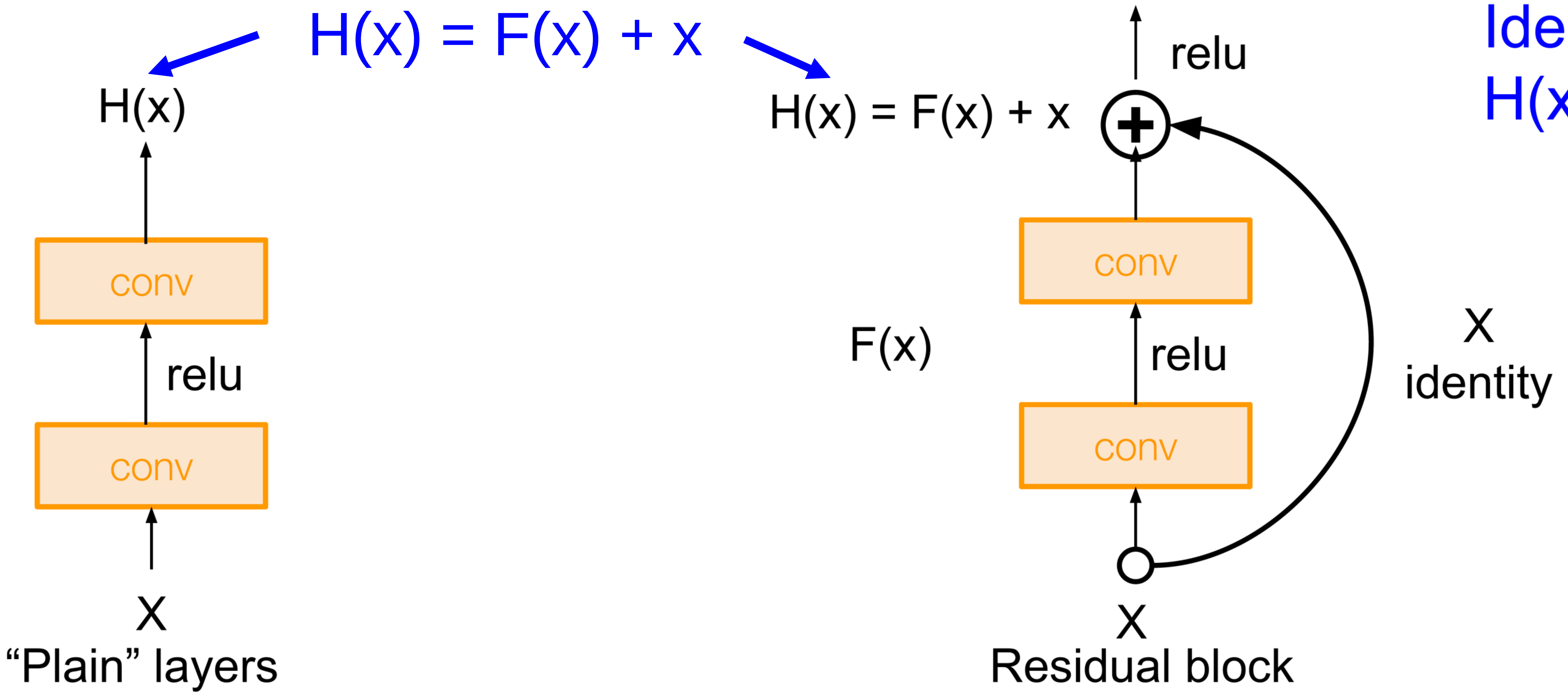


Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *ResNet*

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

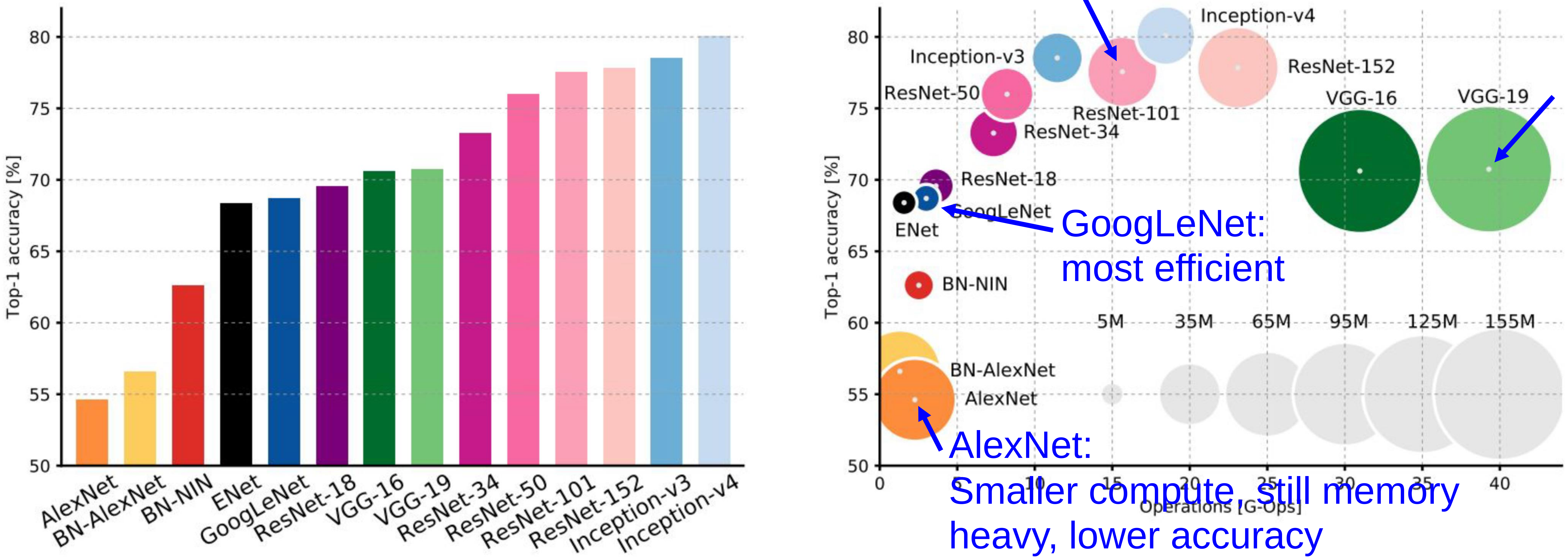
Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

ResNet:
Moderate efficiency depending on model, highest accuracy

CNN Architectures: *Comparing complexity...*



VGG: most parameters, most operations

GoogLeNet: most efficient

AlexNet:
Smaller compute, still memory heavy, lower accuracy

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017.

CNN Architectures: *Improving ResNets...*

[Shao et al. 2016] - Good Practices for Deep Feature Fusion
- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models

[Shao et al. 2016] - Squeeze-and-Excitation Networks (SENet)
- Add a “feature recalibration” module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights

[He et al. 2016] - Identity Mappings in Deep Residual Networks
- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network
- Gives better performance

ResNeXT, DenseNet, MobileNets, NASNet etc.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

CNN Architectures: *Main takeaways*

AlexNet showed that you can use CNNs to train Computer Vision models.

ZFNet, VGG shows that bigger networks work better

GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers

ResNet showed us how to train extremely deep networks

- Limited only by GPU & memory!

- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to

Efficient networks:

- Lots of tiny networks aimed at mobile devices: **MobileNet**, **ShuffleNet, Neural Architecture Search** can now automate architecture design

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Deep neural networks for image classification

Deep learning has proven to take computer vision tasks to an even higher level of accuracy and efficiency, all thanks to convolutional neural networks (CNNs). The aim of it is to emulate the neural networks of the human mind in order to complete specific computer processes with minimal human interference. The variety of layers, starting with the input layer, to the hidden inner layers, and output layer are what make the network considered "deep." In brief, this is how image classification is done via CNNs:

- The input image is fed into the network.
- Various filters are applied to the image in order to generate a feature map.
- A pooling layer is applied to each of those maps.
- The pooled layers are flattened into a vector, then that vector is connected to the neural network.
- The final fully-connected output layer with the classified features is received.

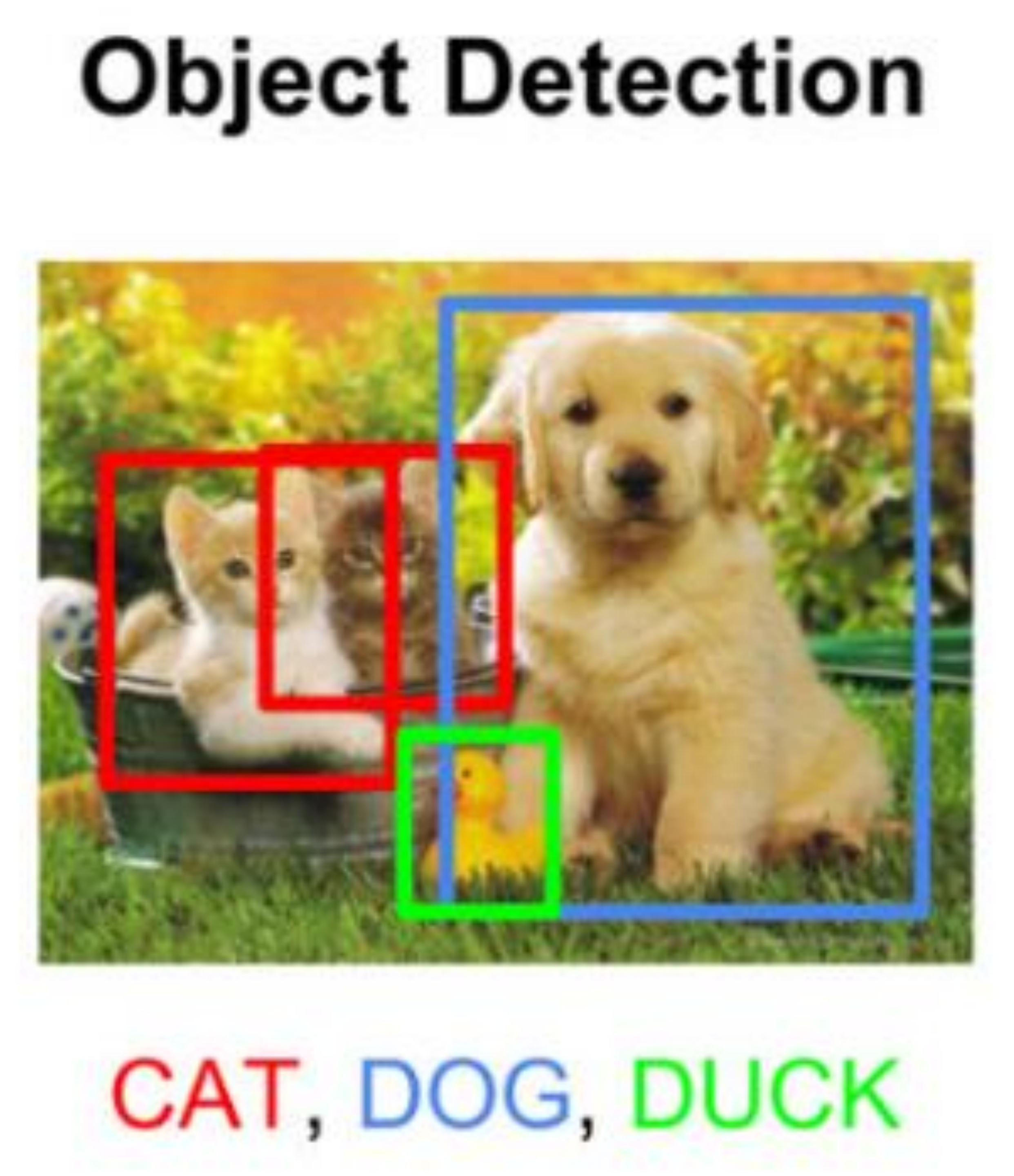
Fully grasping the use of CNNs for image classification requires a much deeper dive into the technical aspect of the model. That deserves a separate crash course of its own if you aim to learn beyond the basics of image classification.

Object detection

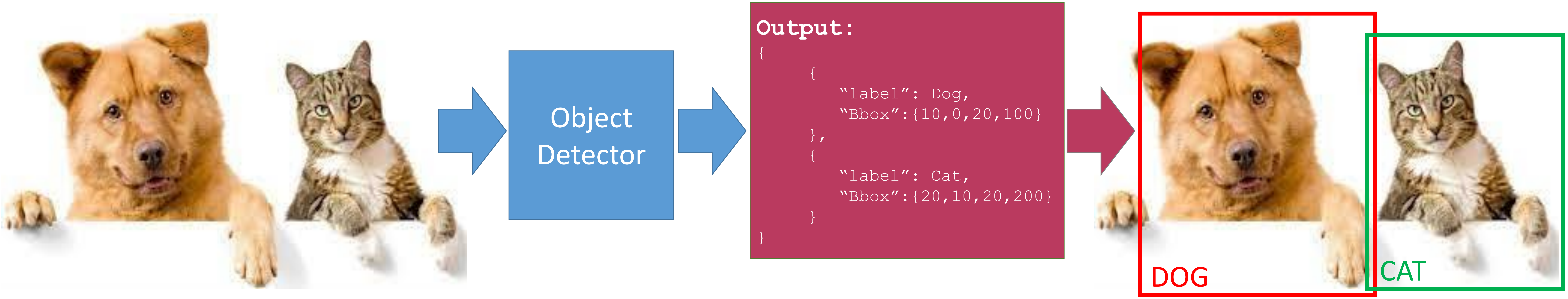
Object detection is a fundamental task in computer vision where the goal is to locate and classify objects within an image or video.

It is a more advanced form of image classification, where instead of just identifying the class of an entire image, they identify multiple instances of multiple classes within an image and locate them with a bounding box; in other words, it deals with more realistic cases in which multiple objects may exist in an image

Object detection algorithms can be used in a variety of applications, such as self-driving cars, security systems, and augmented reality.



Object detection



Object Detector

```
Output:
{
      {
         “label”: Dog,
         “Bbox”:{10,0,20,100}
      },
      {
         “label”: Cat,
         “Bbox”:{20,10,20,200}
      }
}
```

Differences

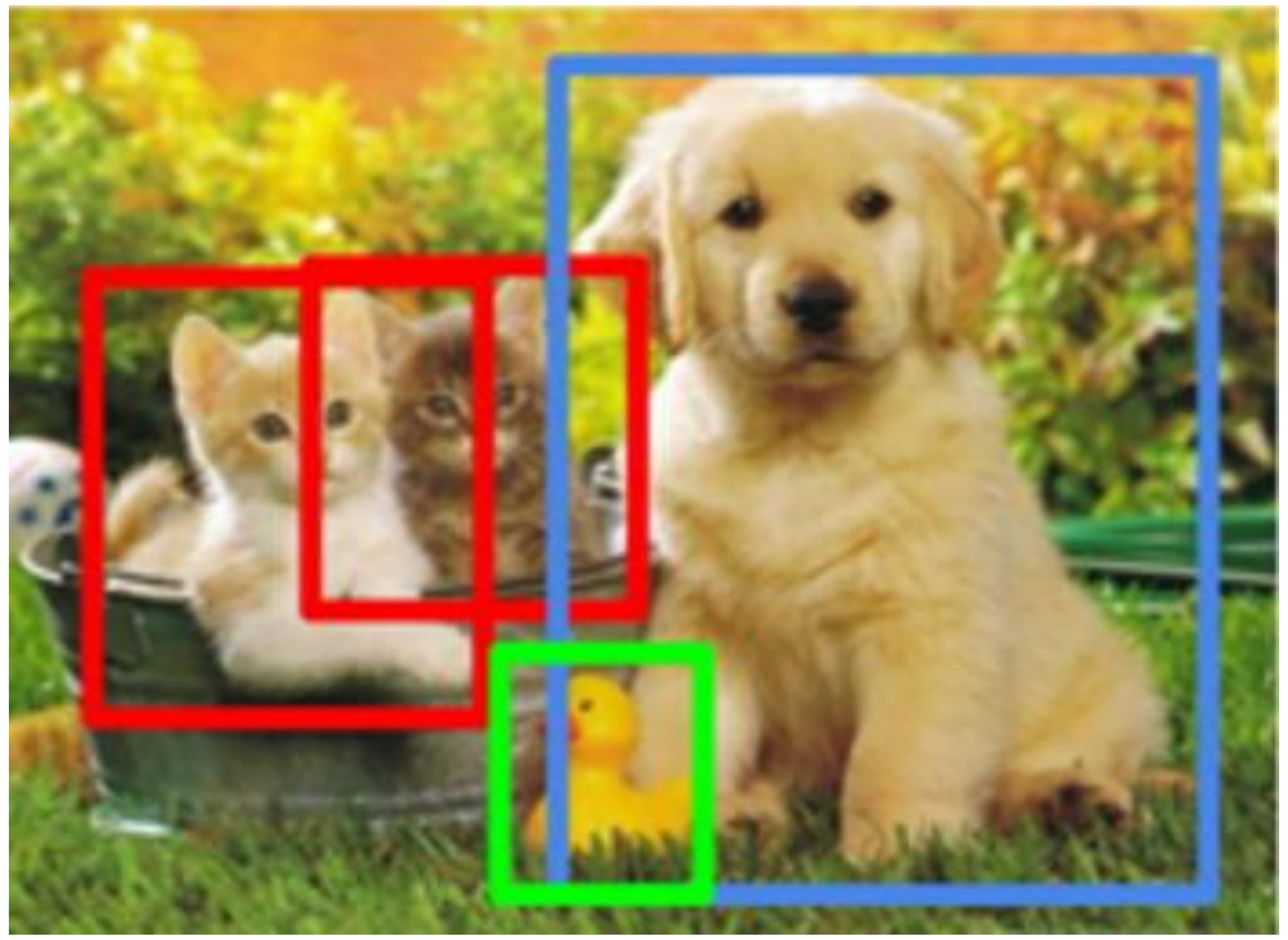
Image Classification

CAT

Localization

CAT

Object Detection

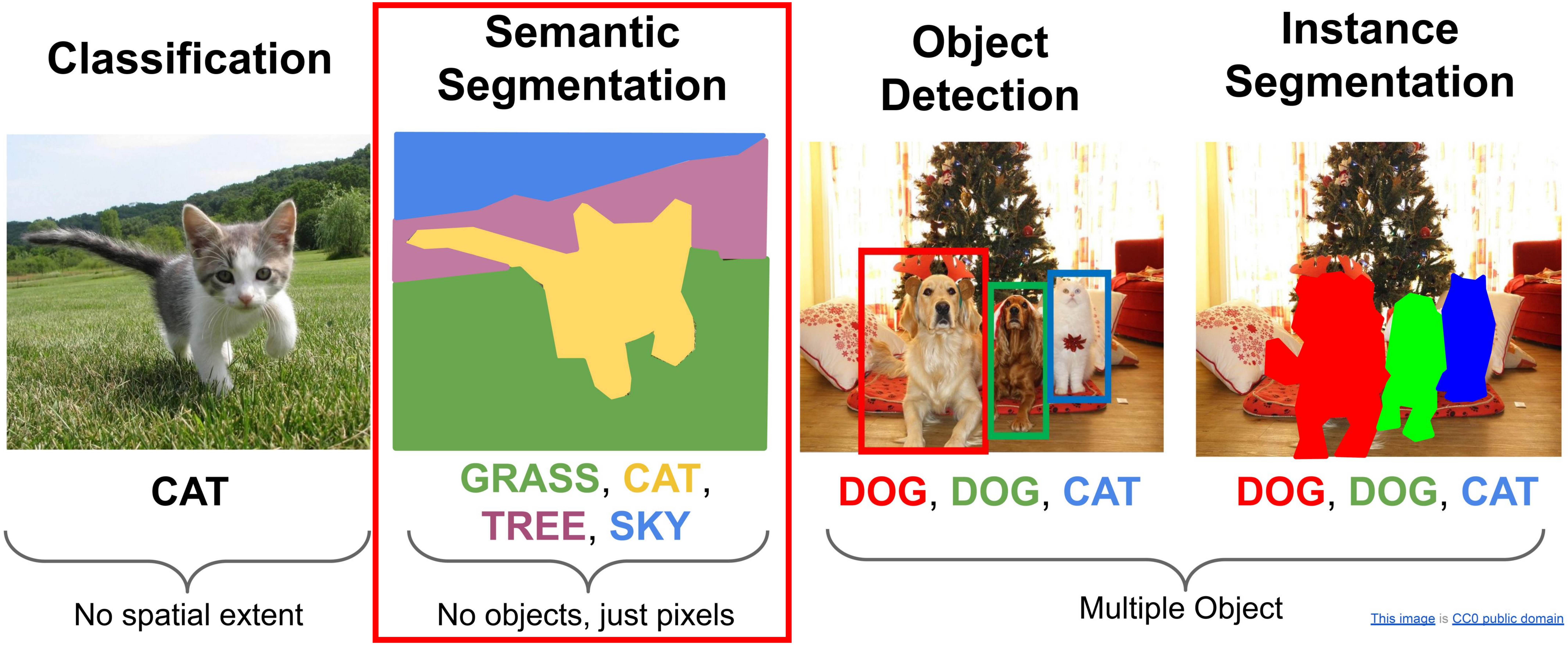


CAT, DUCK, DOG

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Computer Vision Tasks



Classification

Semantic Segmentation

Object Detection

Instance Segmentation

CAT

GRASS, CAT, TREE, SKY

DOG, DOG, CAT

DOG, DOG, CAT

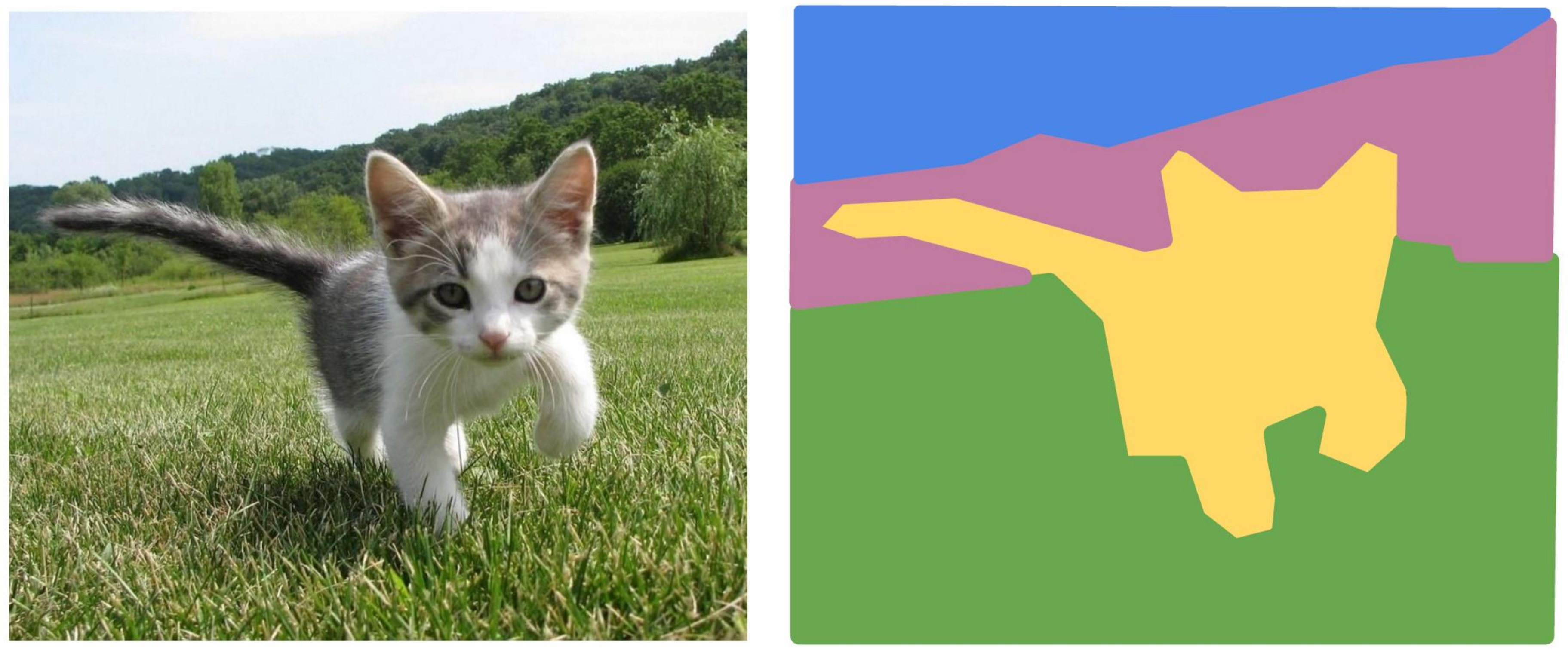
No spatial extent

No objects, just pixels

Multiple Object

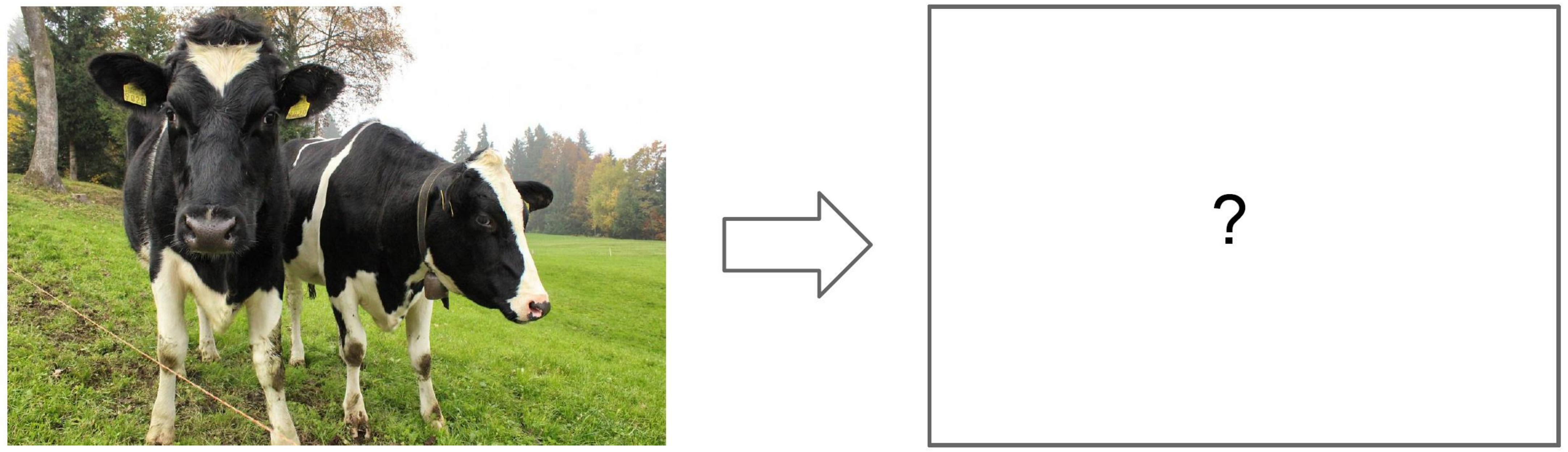
This image is CC0 public domain

Semantic Segmentation: *The Problem*



GRASS, **CAT**,
TREE, **SKY**, **...**

Paired training data: for each training image, each pixel is labeled with a semantic category.



At test time, classify each pixel of a new image.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Semantic Segmentation: *The Problem*

Semantic segmentation is the process of dividing an image into multiple segments and assigning each segment a label or a class. The sliding window is an idea for semantic segmentation that involves breaking an image into smaller sections, or windows, and processing each window separately to identify the objects and their boundaries.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Semantic Segmentation: *Sliding window*

Full image

Impossible to classify without context

Q: how do we include context?

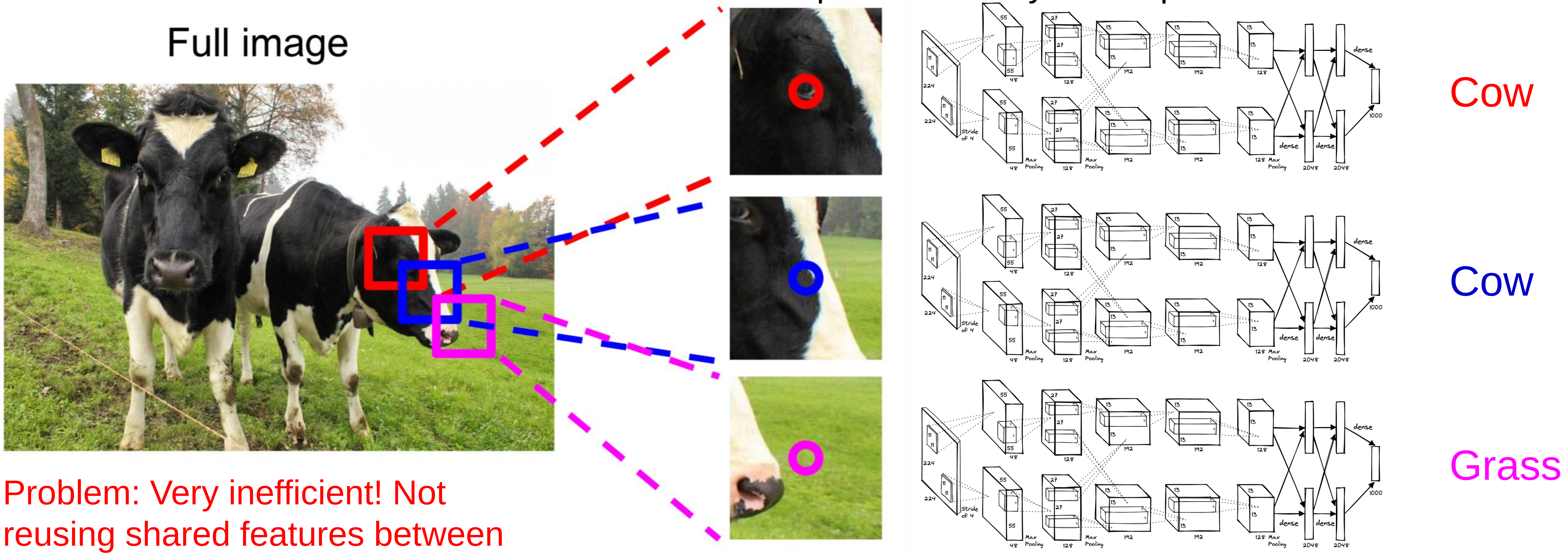
Semantic Segmentation: *Sliding window*

In the sliding window approach, a window of a fixed size moves across the image, classifying each portion of the image that it covers. The classification is performed using a machine learning model that has been trained on a dataset of annotated images, where each pixel in the image is labeled with the corresponding object class.

As the window slides across the image, it classifies each portion of the image that it covers, producing a segmentation map that identifies the object classes and their boundaries within the image. The resulting segmentation map can be used for a variety of tasks, such as object recognition, image segmentation, and scene understanding.

While the sliding window approach can be effective for semantic segmentation, it can also be computationally expensive, especially for larger images. To address this issue, researchers have developed more efficient algorithms, such as convolutional neural networks (CNNs), that can perform semantic segmentation with high accuracy and lower computational cost.

Semantic Segmentation: *Sliding window*



Extract patch

Classify center pixel with CNN

Full image

Cow

Cow

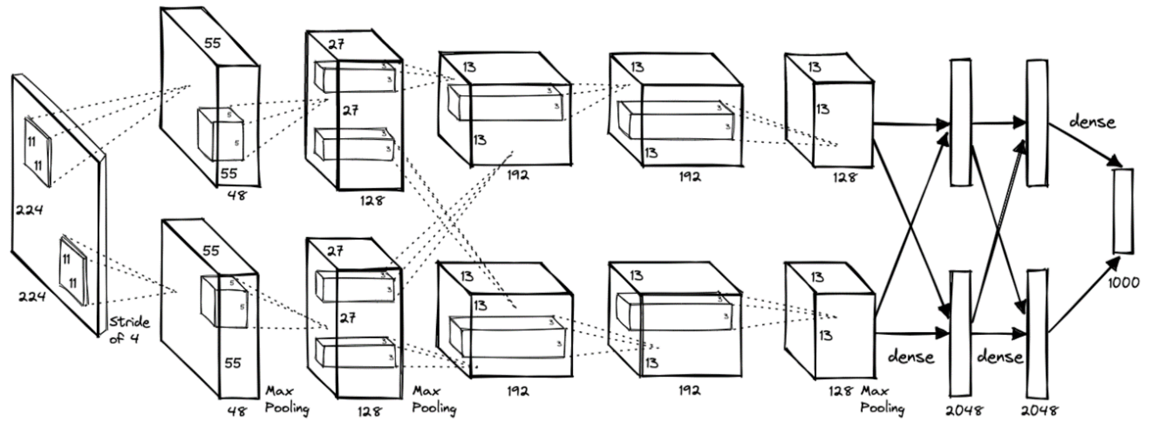
Grass

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Problem: Very inefficient! Not reusing shared features between overlapping patches

Q: how do we model this?

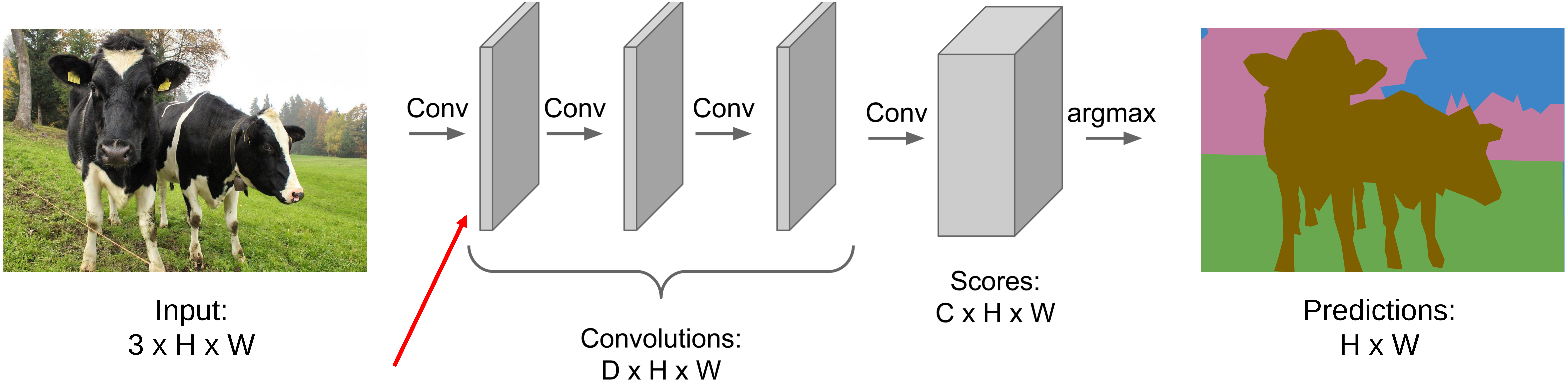
Semantic Segmentation Idea: *Convolution*



An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

Semantic Segmentation Idea: *Convolution*



Input:
3 x H x W

Convolutions:
D x H x W

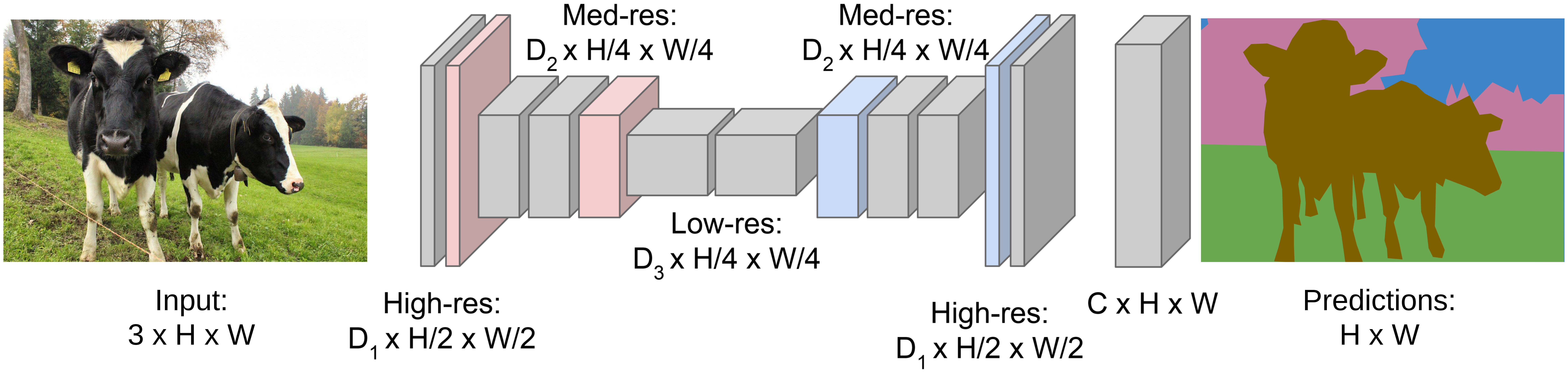
Scores:
C x H x W

Predictions:
H x W

Problem: convolutions at original image resolution will be very expensive ...

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!

Semantic Segmentation Idea: *Fully Convolutional*



Med-res:
D_2 x H/4 x W/4

Med-res:
D_2 x H/4 x W/4

Low-res:
D_3 x H/4 x W/4

Input:
3 x H x W

High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

C x H x W

Predictions:
H x W

Downsampling:
Pooling, strided convolution

Upsampling:
???

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

In-Network upsampling: *“Unpooling”*

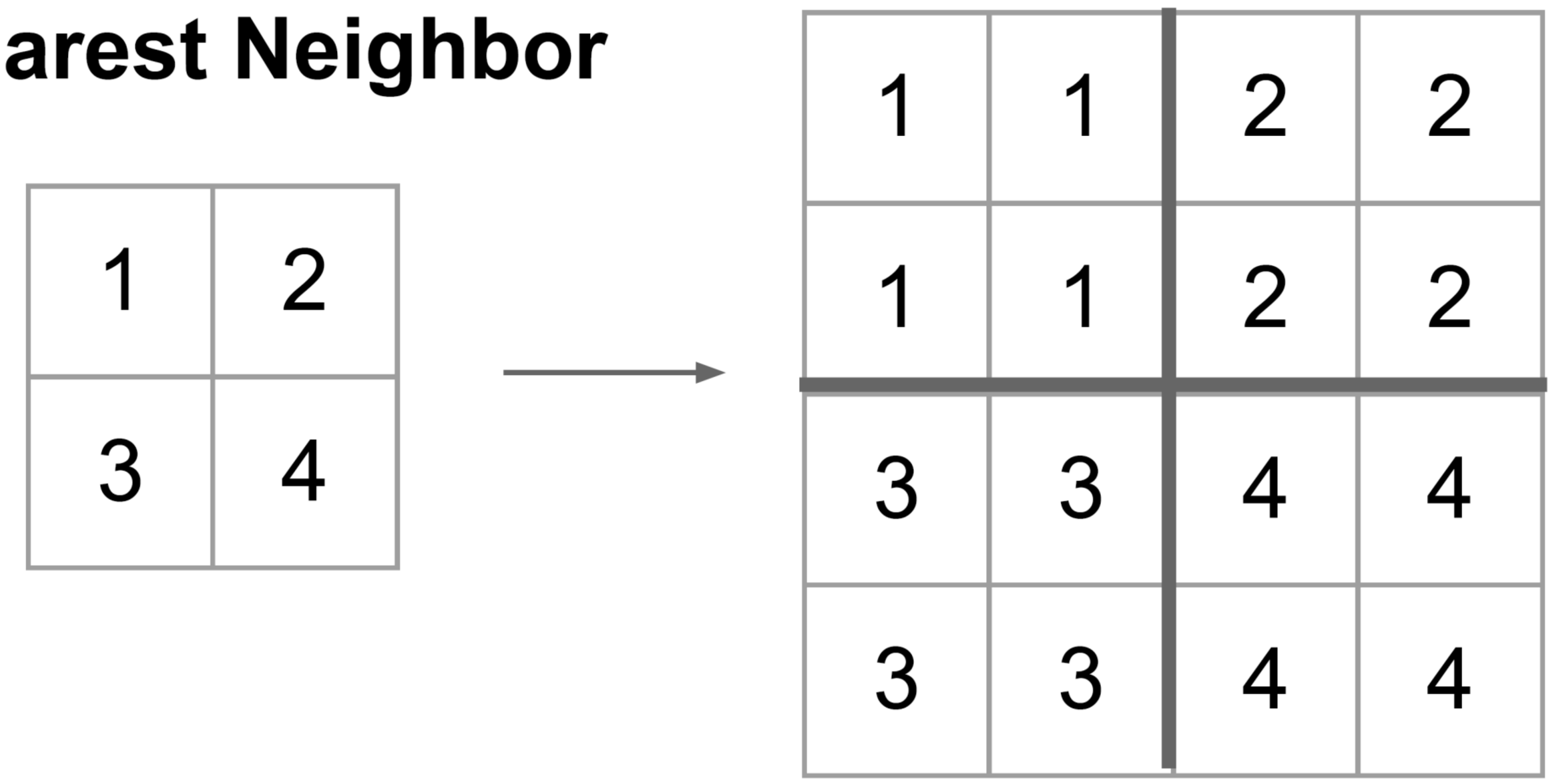
Unpooling is a technique used in convolutional neural networks (CNNs) to increase the resolution of the feature map while preserving the spatial information and avoiding the introduction of artifacts.

Pooling operations in CNNs reduce the spatial dimension of the feature map by down-sampling it, which makes it difficult to reconstruct the original image. Unpooling, on the other hand, performs the inverse operation of pooling, which means it restores the original spatial dimensions of the feature map by upsampling it. The unpooling operation is usually performed in conjunction with a convolutional layer, where each pixel in the output feature map is associated with a receptive field in the input feature map. During the unpooling operation, the output feature map is first resized to match the size of the input feature map, and then the value of each pixel in the output feature map is distributed over its associated receptive field in the input feature map.

There are several approaches to unpooling, such as **nearest-neighbor interpolation**, **bilinear interpolation**, and **max unpooling**. In-network upsampling with unpooling is a useful technique for semantic segmentation tasks because it can improve the resolution of the feature map, which allows for better localization of objects and more precise segmentation boundaries.

In-Network upsampling: *“Unpooling”*

Nearest Neighbor



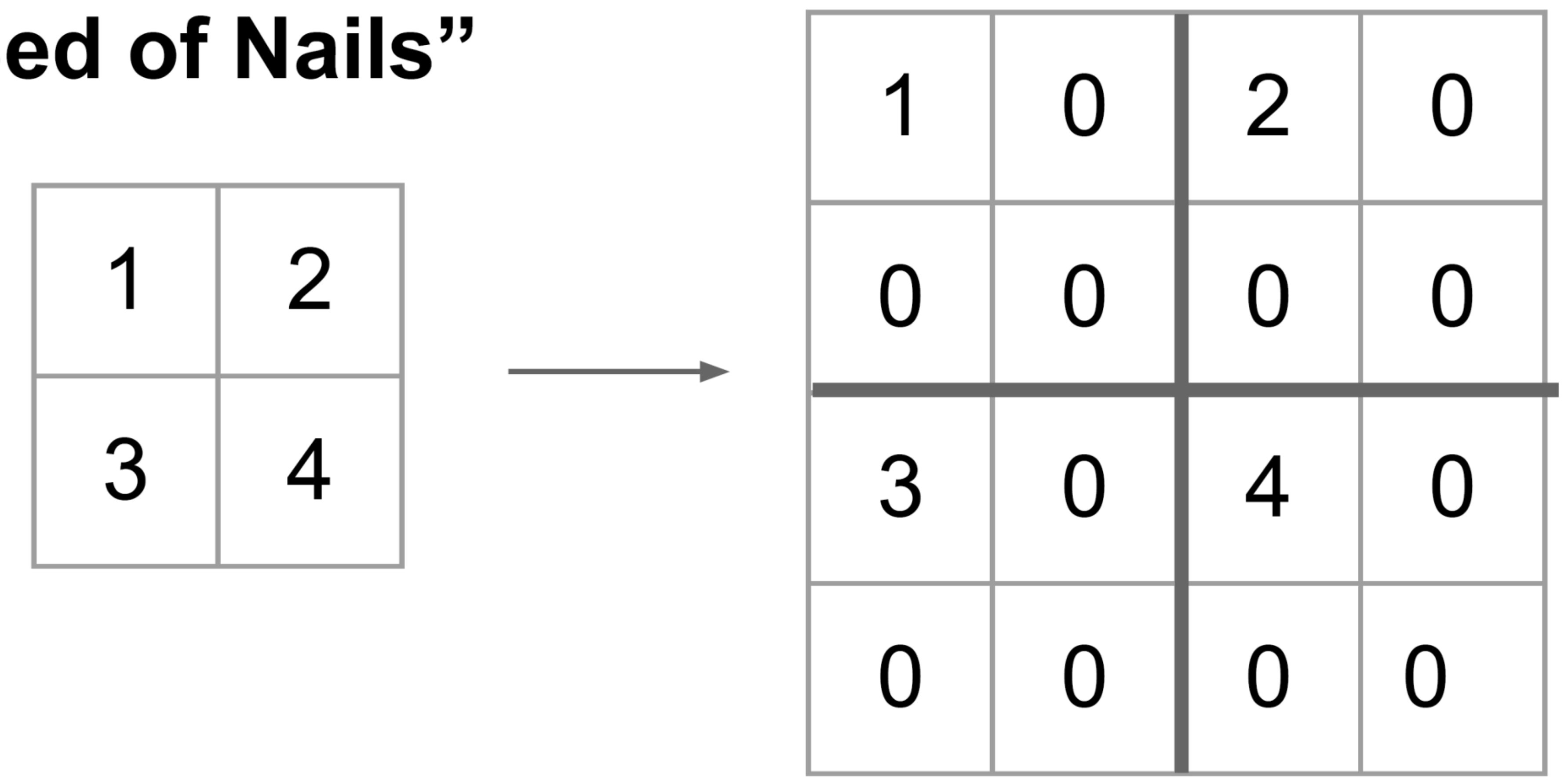
1	2
3	4

Input: 2 x 2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

“Bed of Nails”



1	2
3	4

Input: 2 x 2

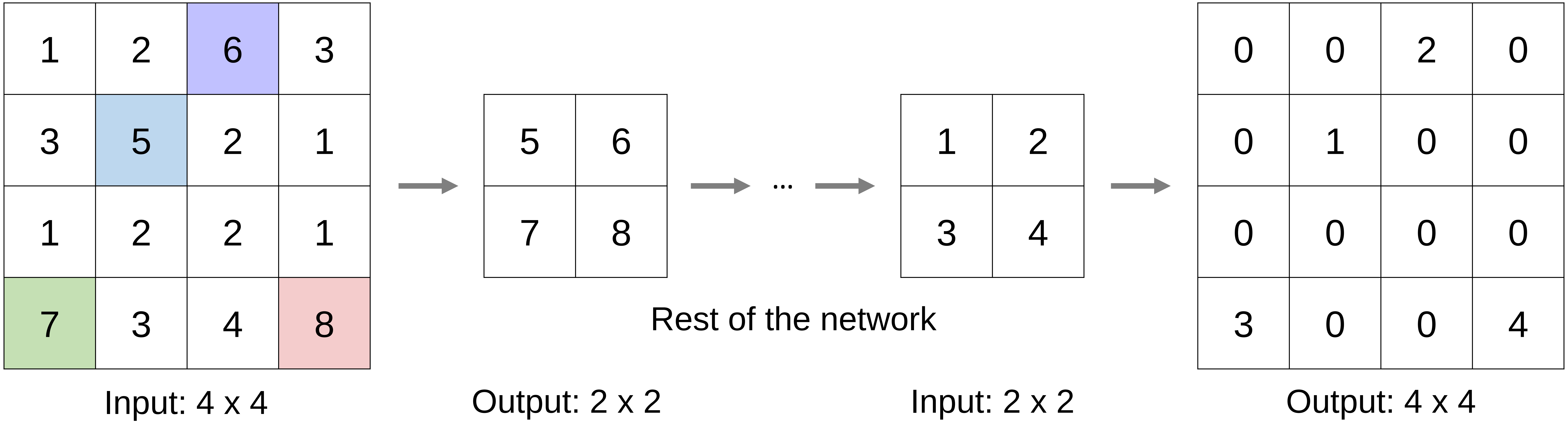
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

In-Network upsampling: *“Max Unpooling”*

Max Pooling
Remember which element was max!

Max Unpooling
Use positions from pooling layer



Input: 4 x 4

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

Output: 2 x 2

5	6
7	8

Input: 2 x 2

1	2
3	4

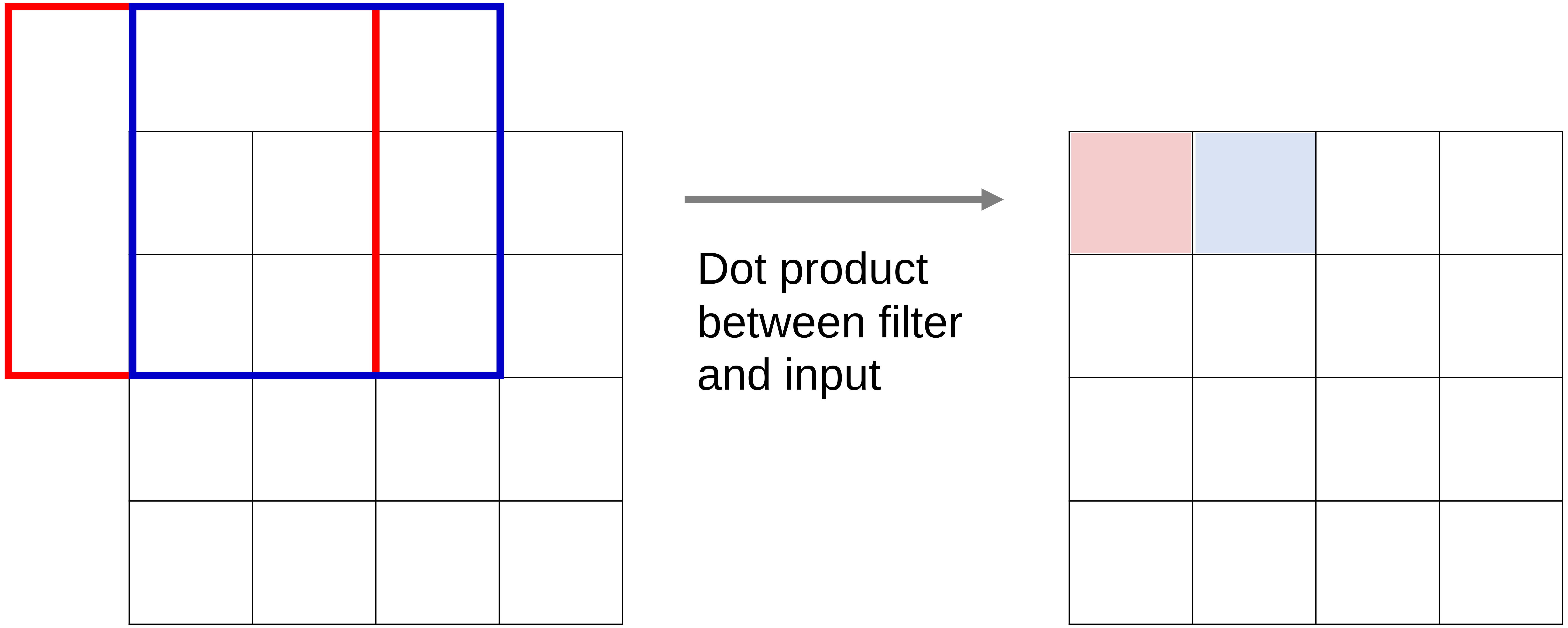
Output: 4 x 4

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

In max unpooling, the position of the maximum value in each pooling operation is recorded and used during the unpooling operation to place the value back into its original location in the feature map.

Learnable Upsampling

Recall: Normal 3 x 3 convolution, stride 1 pad 1



Dot product between filter and input

Input: 4 x 4

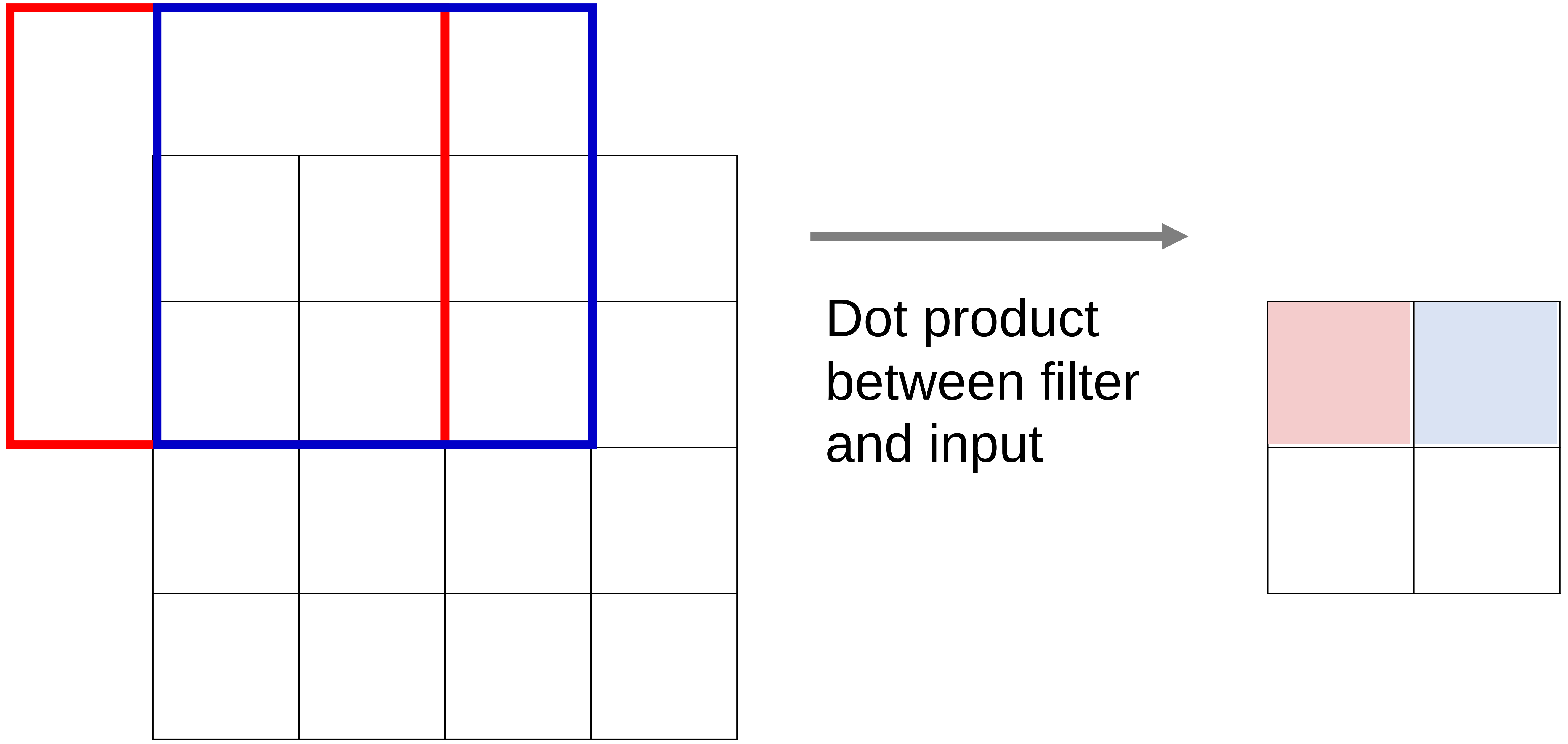
Output: 4 x 4

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Learnable Upsampling

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Dot product between filter and input

Input: 4 x 4

Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

We can interpret strided convolution as “learnable downsampling”.

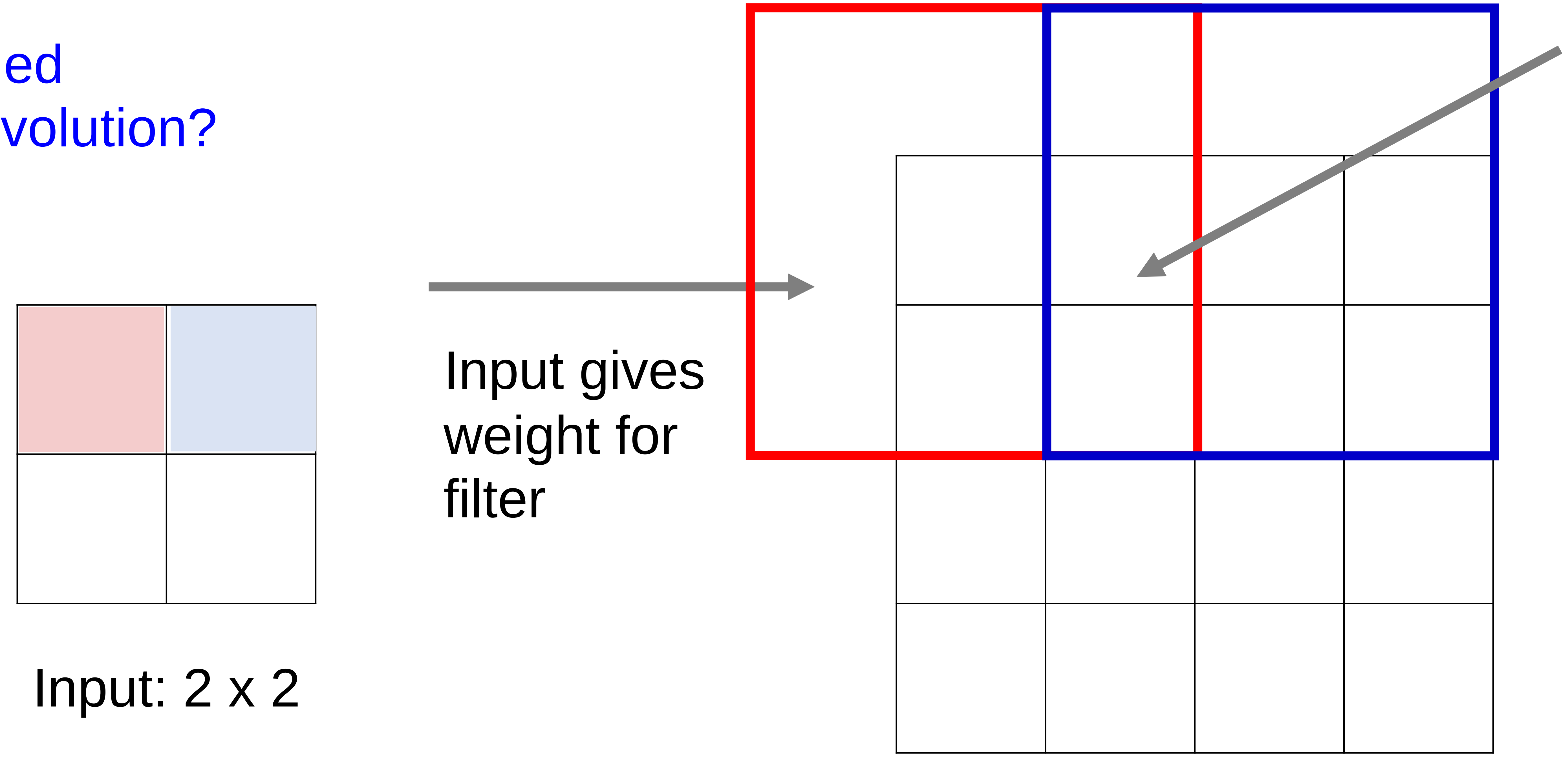
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Learnable Upsampling: *Transposed Convolution*

3 x 3 **transposed convolution**, stride 2 pad 1

Q: Why is it called transposed convolution?



Input gives weight for filter

Sum where output overlaps

Input: 2 x 2

Output: 4 x 4

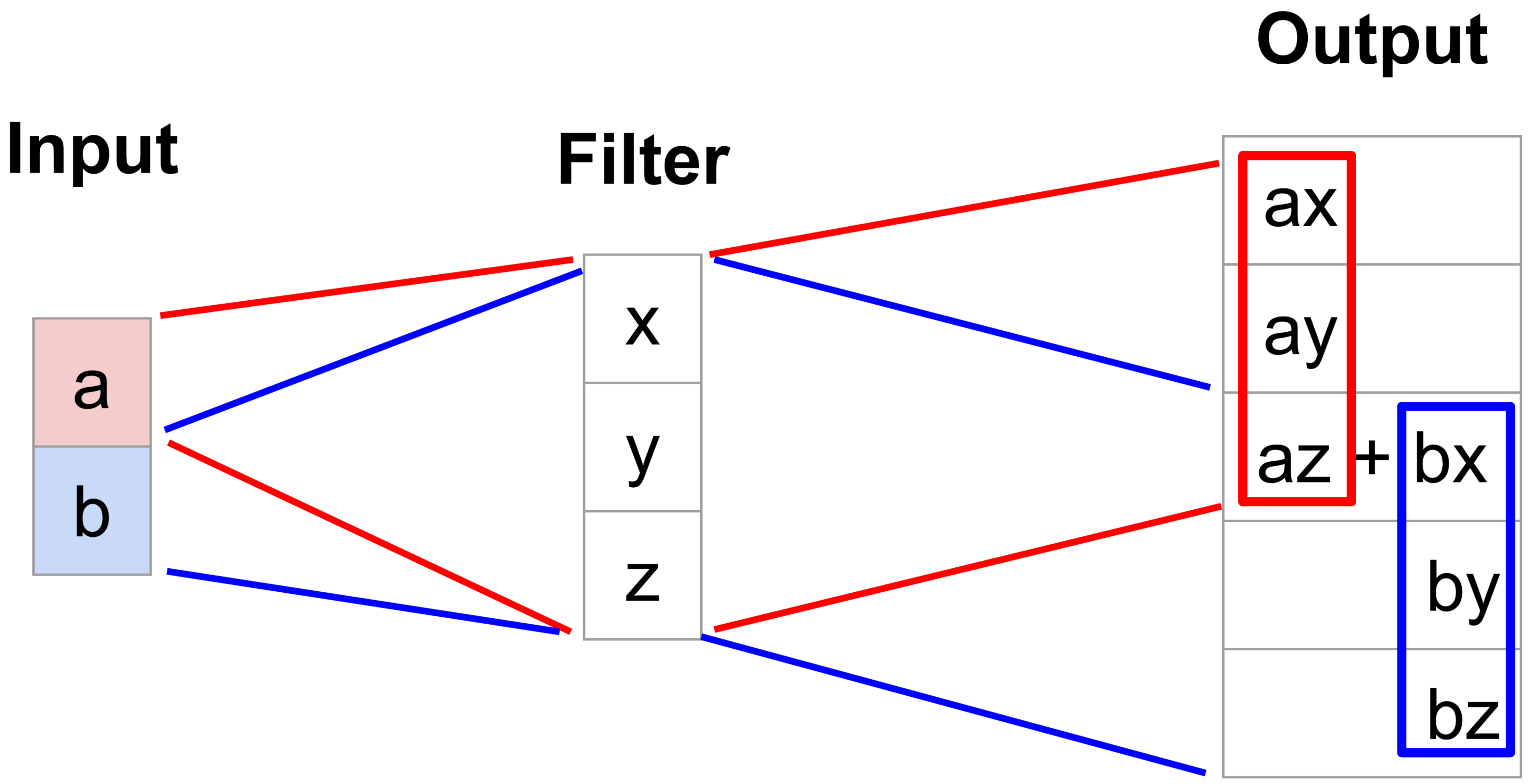
Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Learnable Upsampling: *1D Example*



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

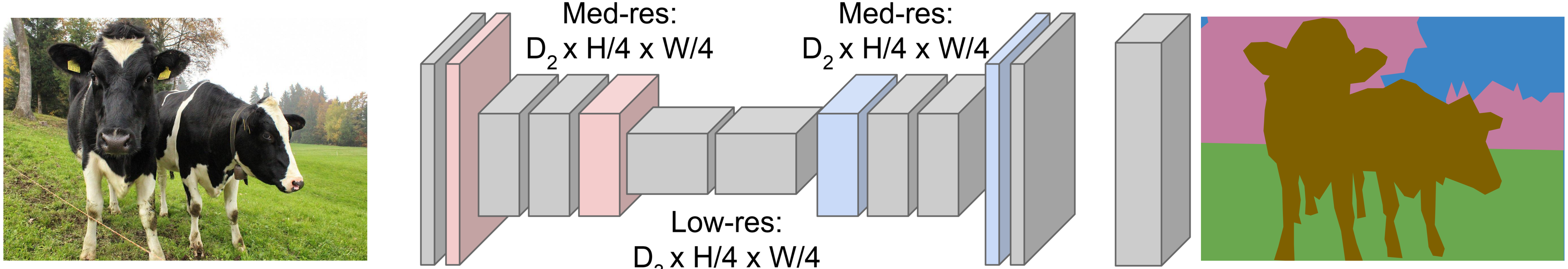
Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Example: 1D transposed conv, kernel size=3, <u>stride=2</u>, padding=0

Semantic Segmentation Idea: *Fully Convolutional*



Med-res: D_2 x H/4 x W/4

Med-res: D_2 x H/4 x W/4

Low-res: D_3 x H/4 x W/4

Input: 3 x H x W

High-res: D_1 x H/2 x W/2

High-res: D_1 x H/2 x W/2

C x H x W

Predictions: H x W

Downsampling:
Pooling, strided convolution

Unpooling or strided transposed convolution

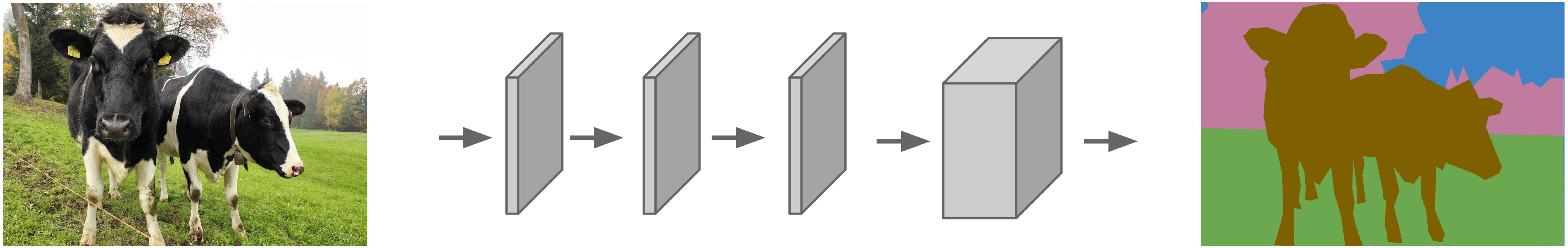
Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

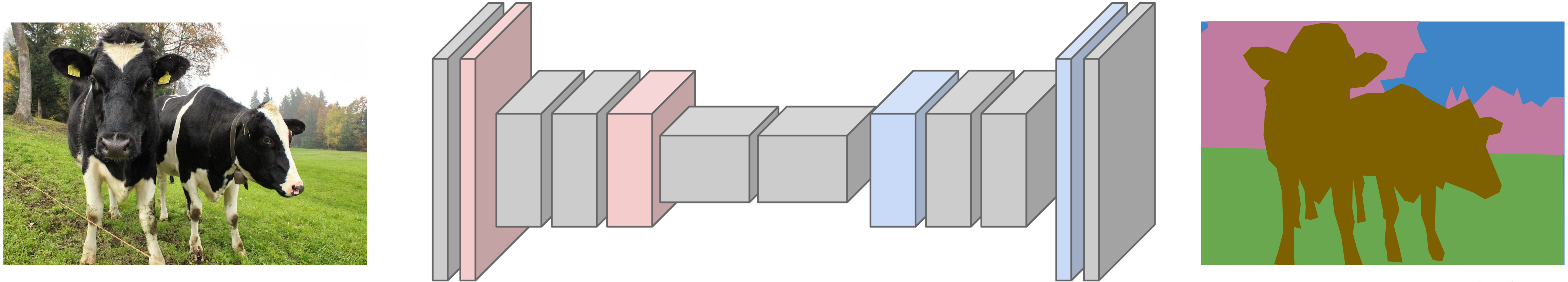
Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Semantic Segmentation: *Summary*

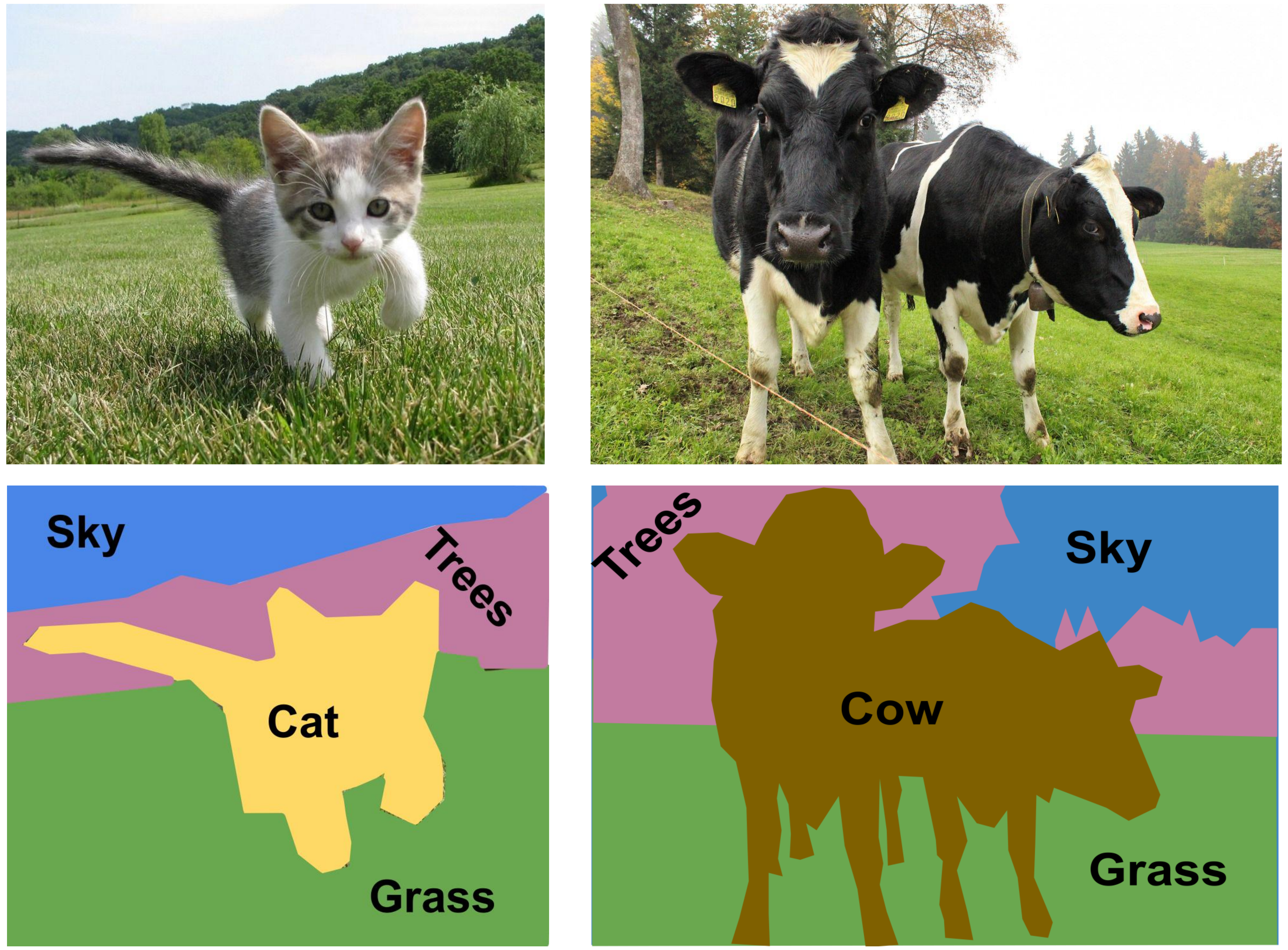




Semantic Segmentation

Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels



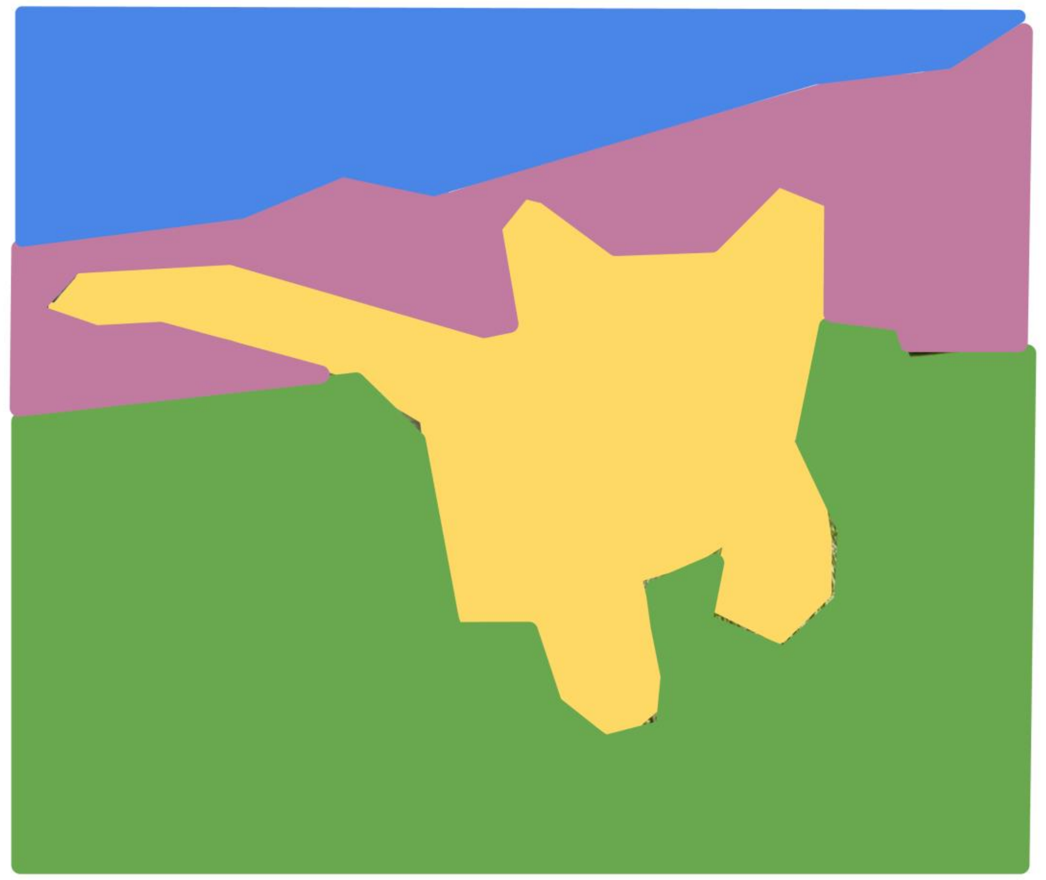
Computer Vision Tasks

Classification

CAT

No spatial extent

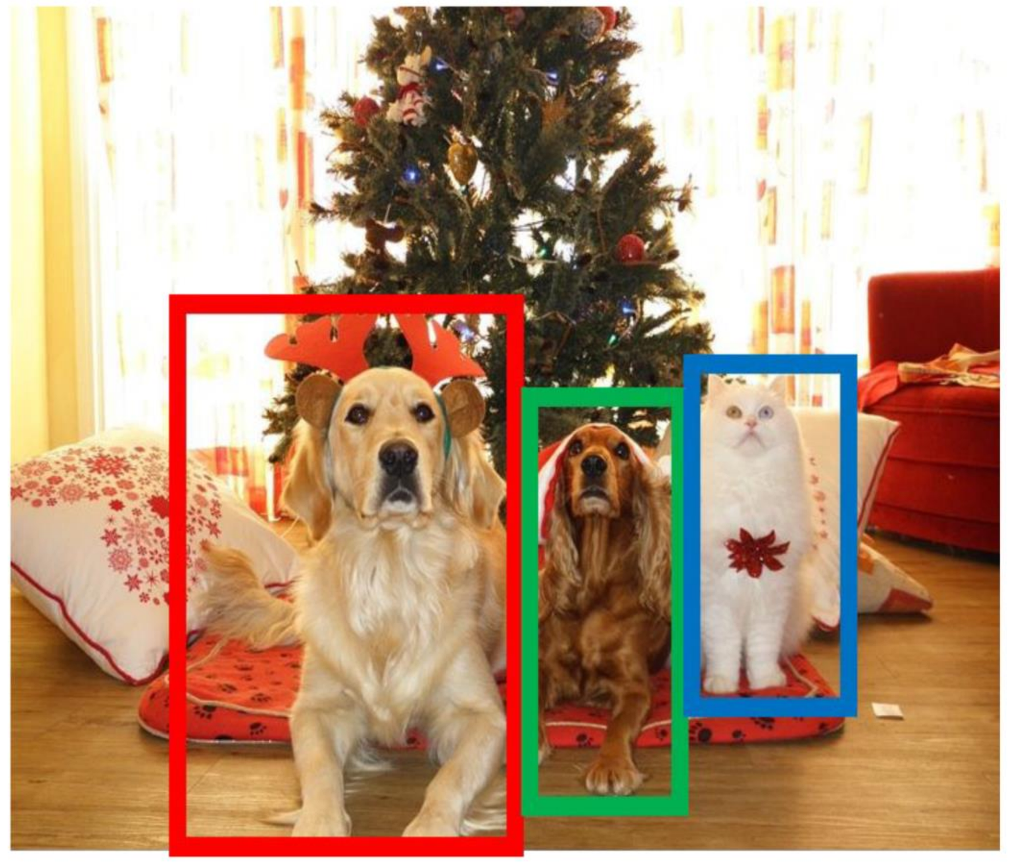
Semantic Segmentation



GRASS, CAT, TREE, SKY

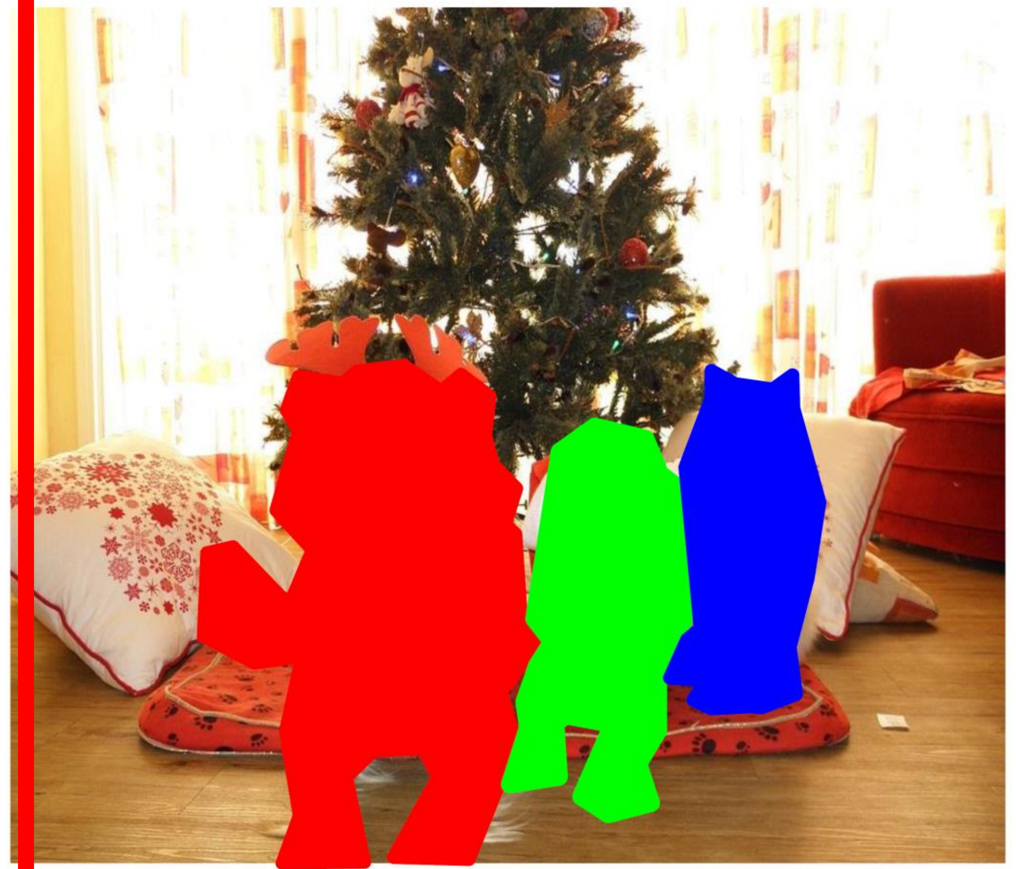
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

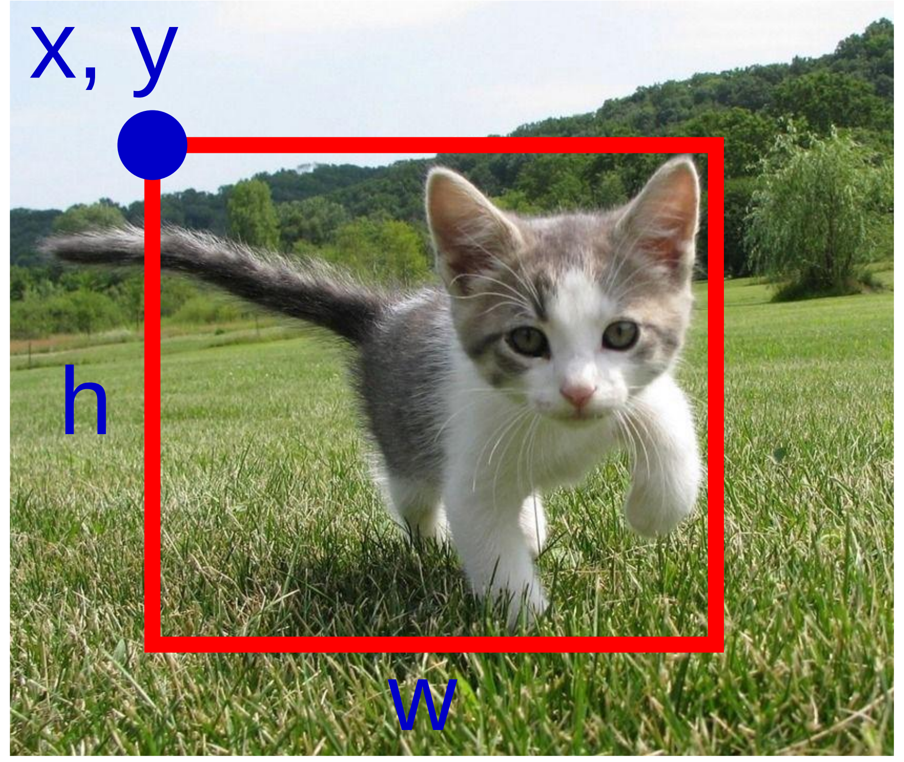


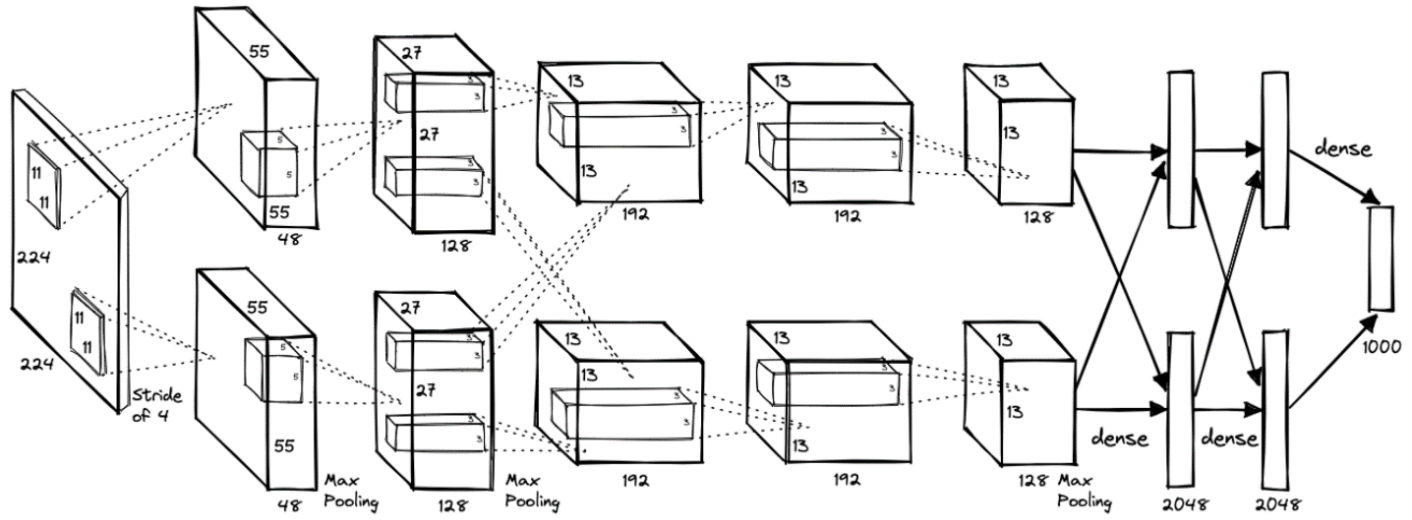
DOG, DOG, CAT

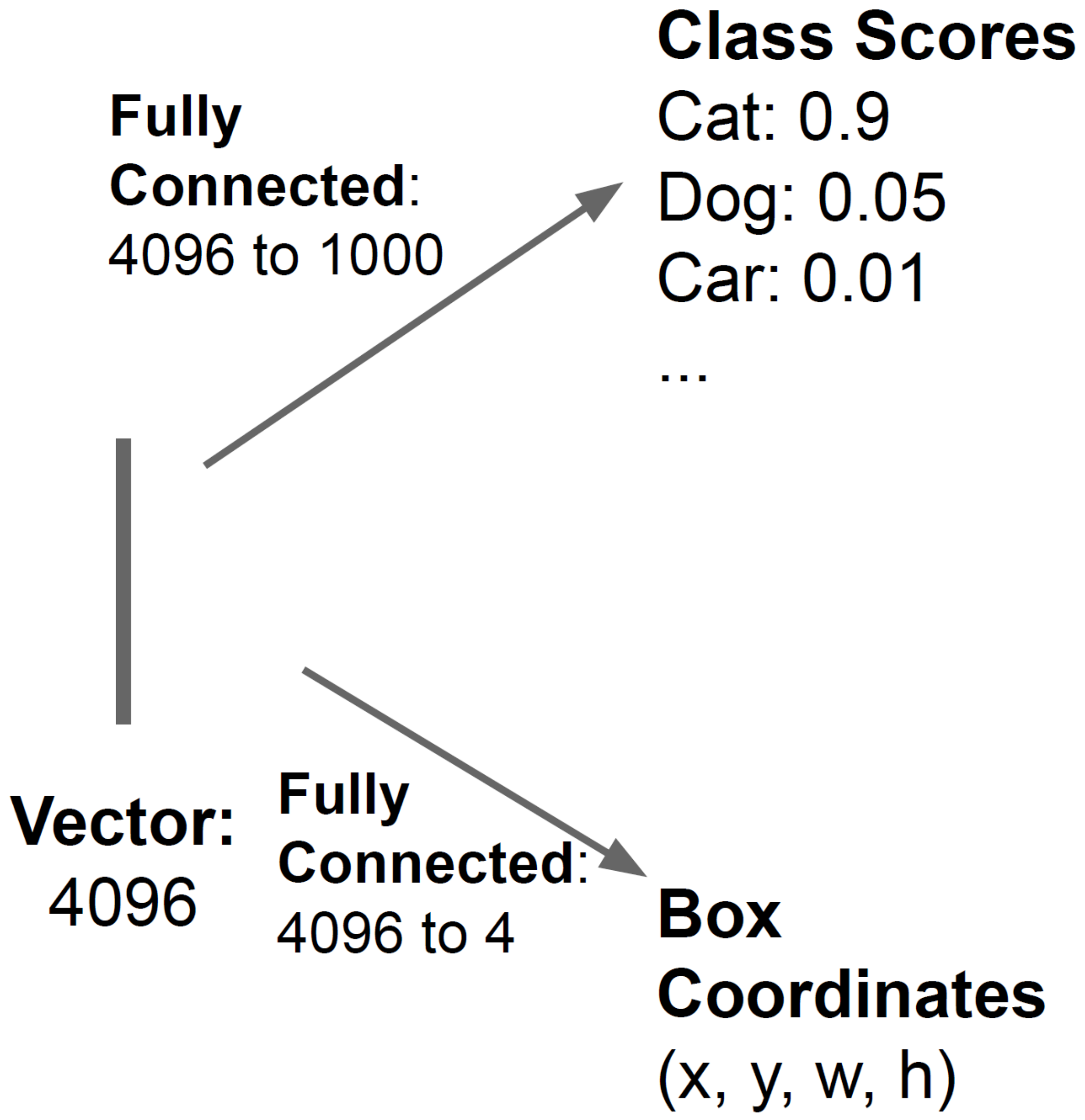
Multiple Object

This image is CC0 public domain

Object Detection: *Single Object (Classification + Localization)*







Vector: 4096

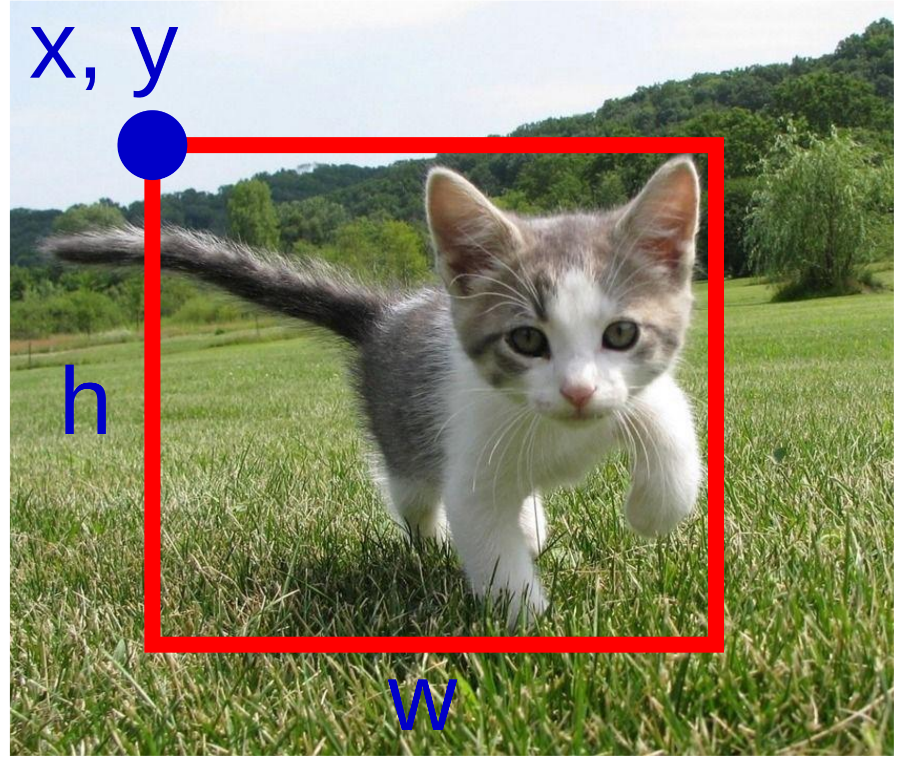
Fully Connected: 4096 to 1000

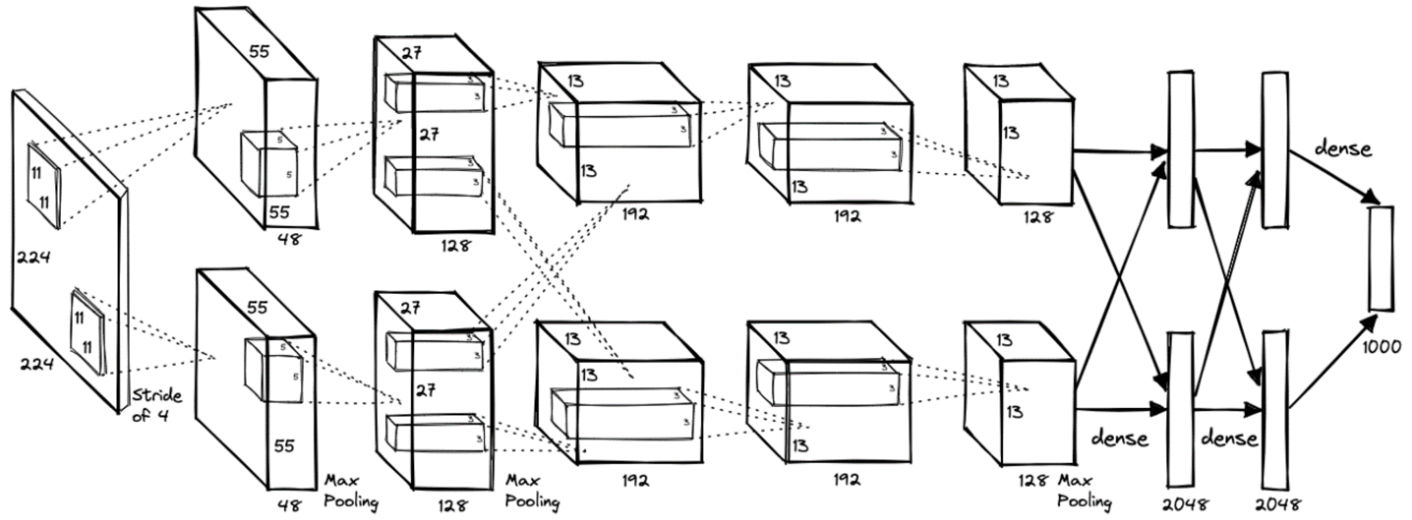
Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Fully Connected: 4096 to 4

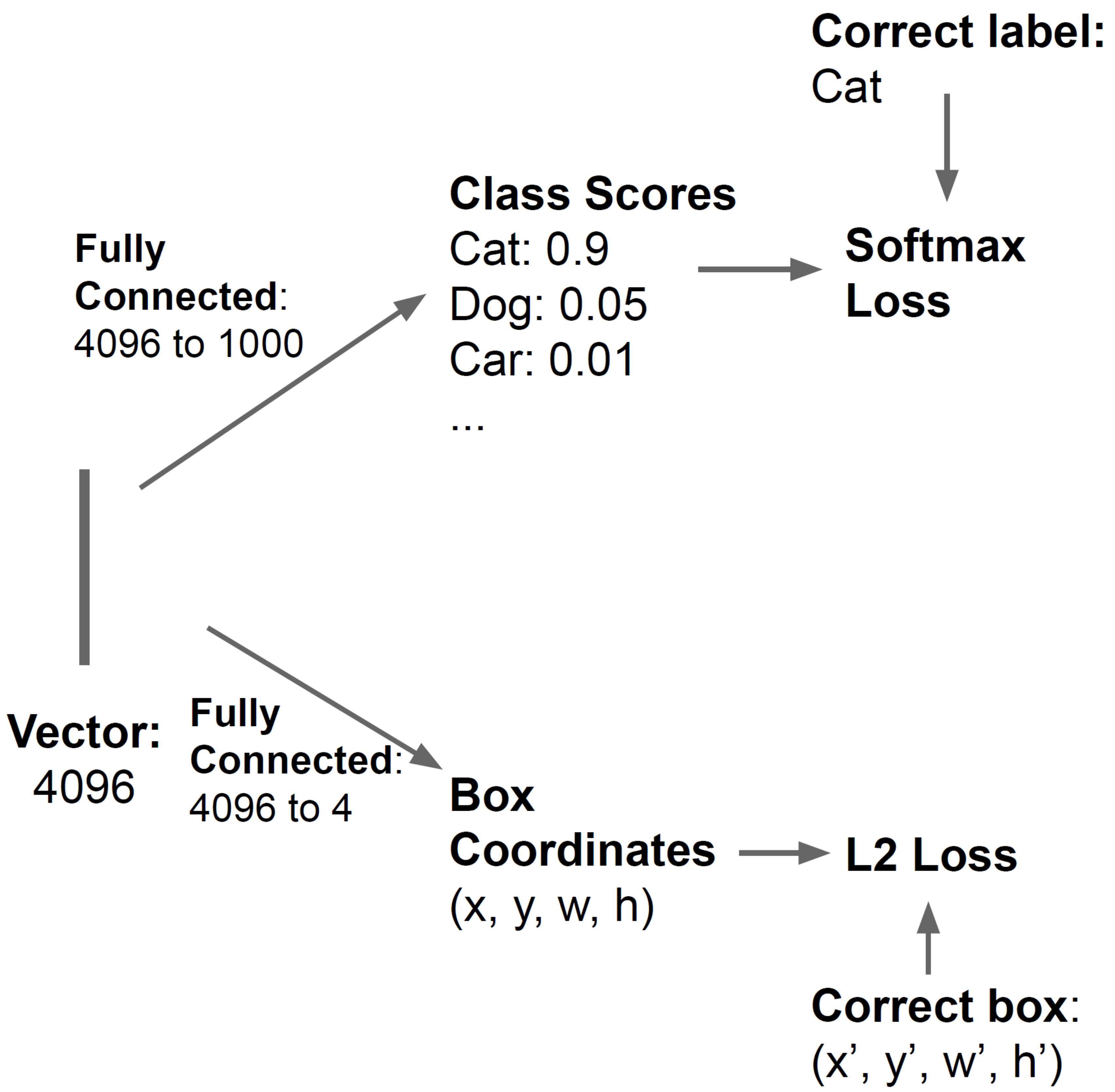
Box Coordinates
(x, y, w, h)

Object Detection: *Single Object (Classification + Localization)*





Treat localization as a regression problem!



Correct label:
Cat

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Fully Connected: 4096 to 1000

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

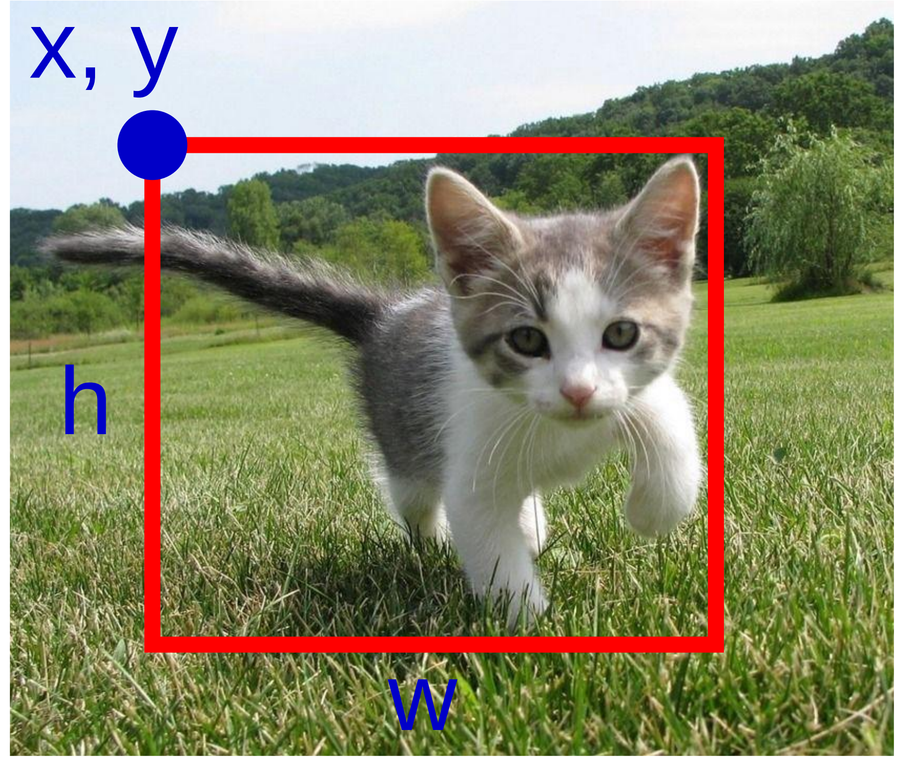
L2 Loss

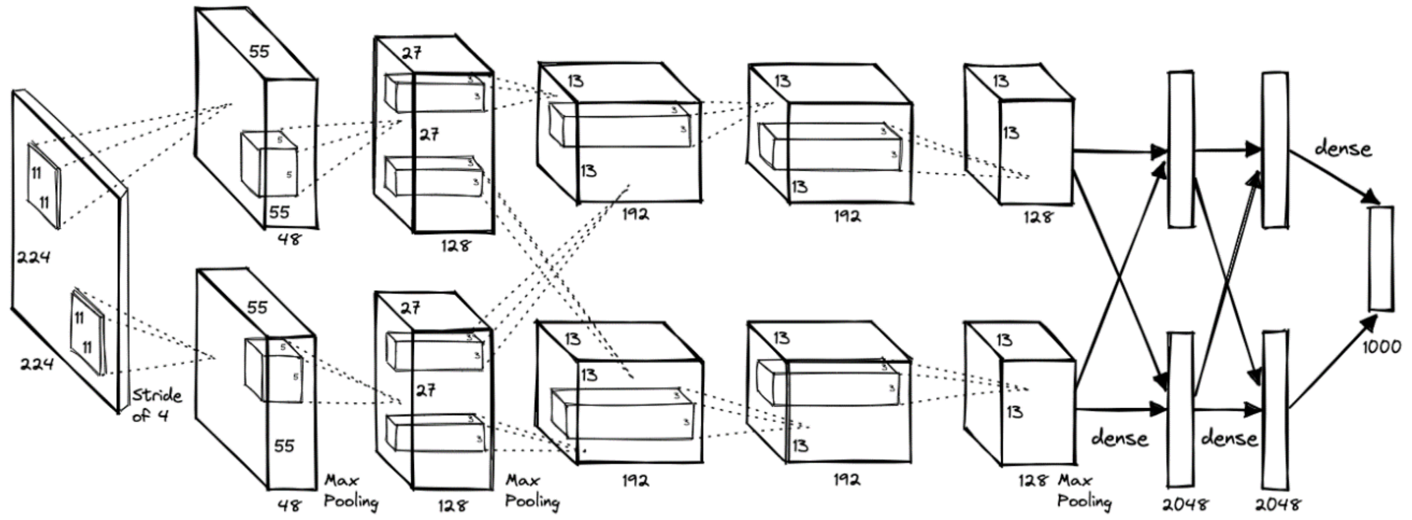
Correct box:
(x', y', w', h')

Co-financed by the European Union
Connecting Europe Facility

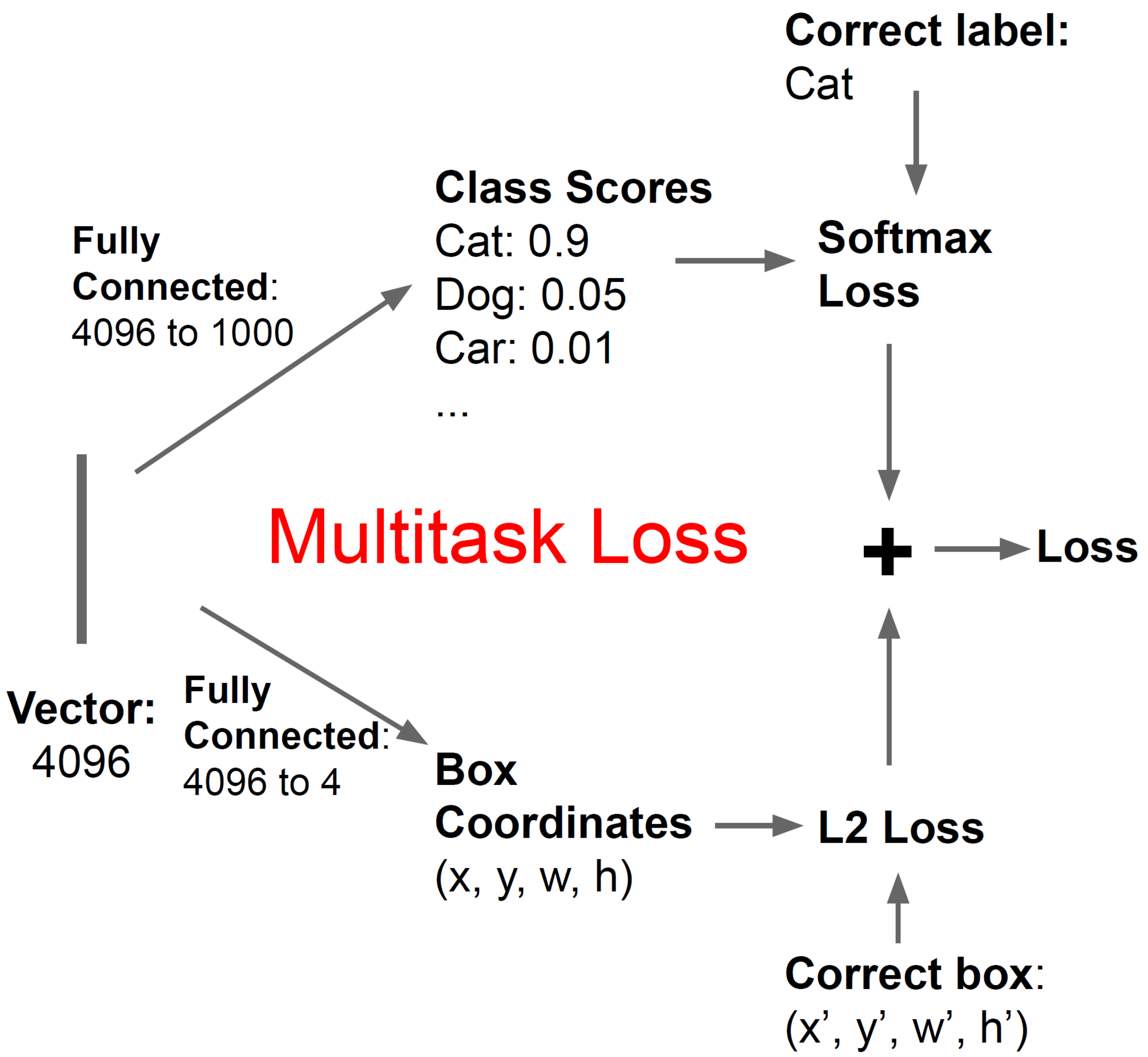
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Object Detection: *Single Object (Classification + Localization)*





Treat localization as a regression problem!



Correct label:
Cat

Fully Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Vector:
4096

Multitask Loss

+

Loss

Fully Connected:
4096 to 4

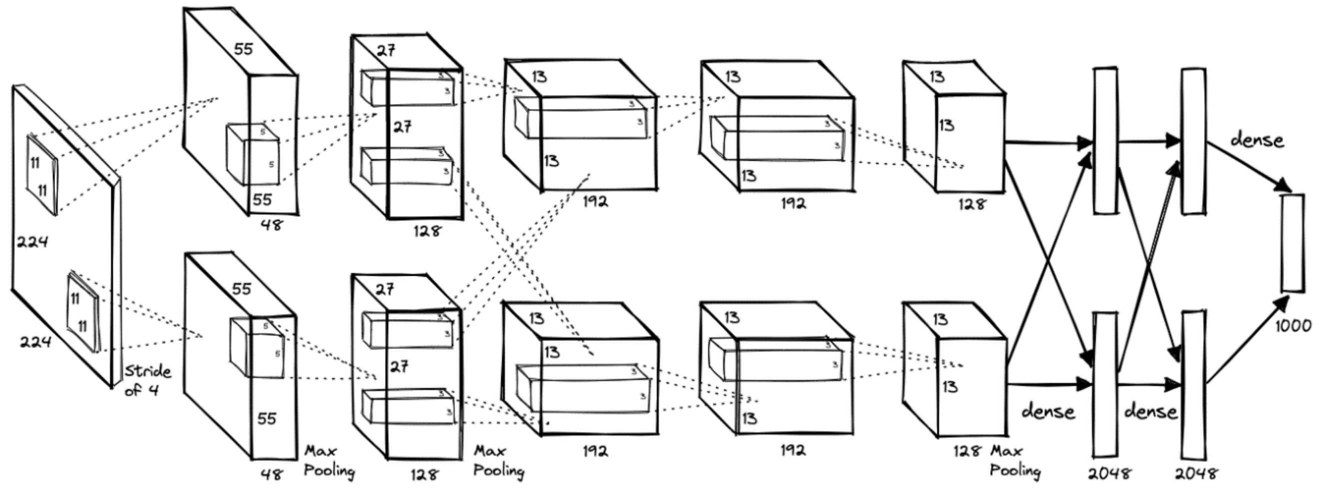
Box Coordinates
(x, y, w, h)

L2 Loss

Correct box:
(x’, y’, w’, h’)

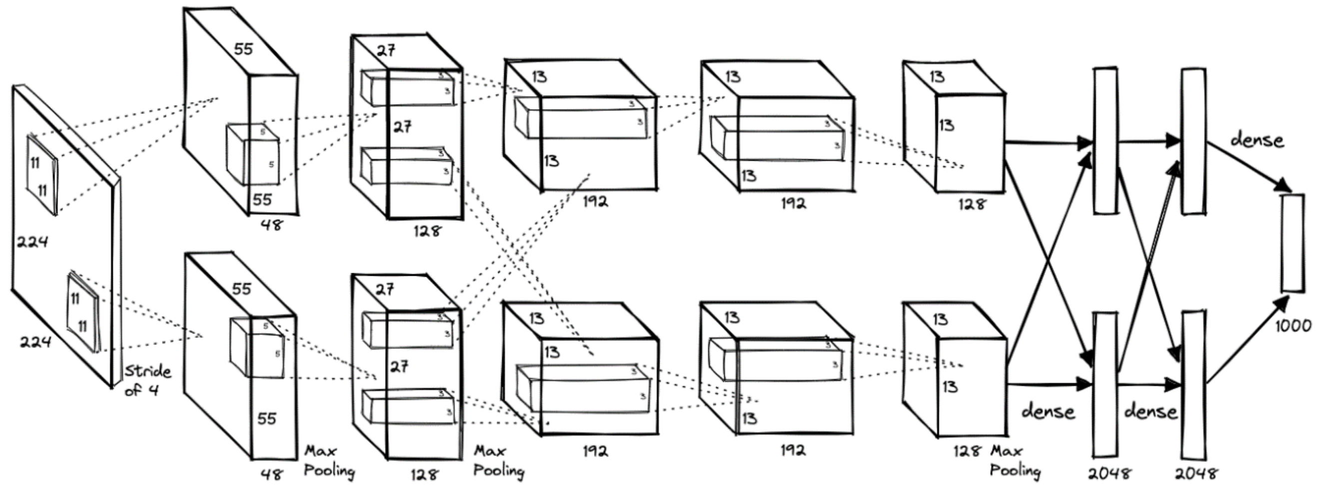
Object Detection: *Multiple Objects*

Each image needs a different number of outputs!



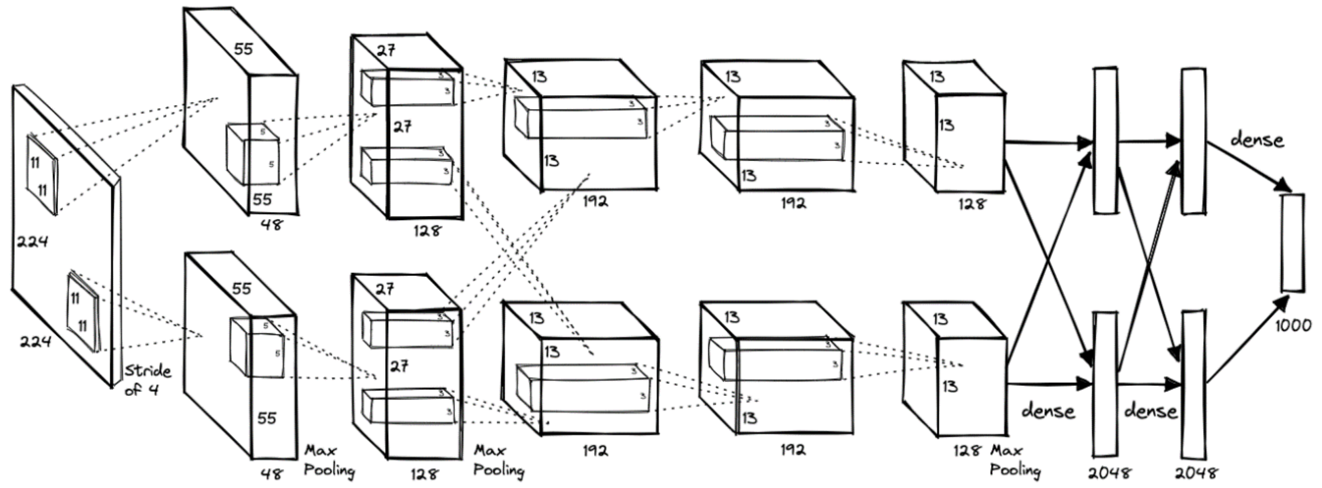
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

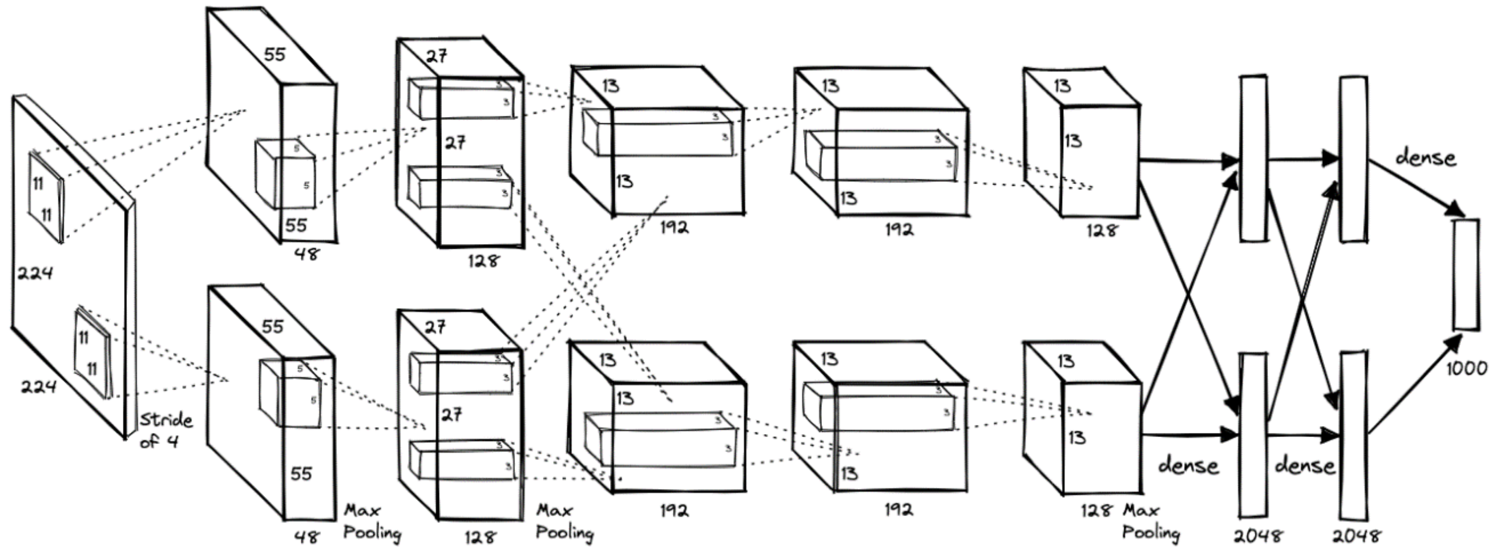
12 numbers



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

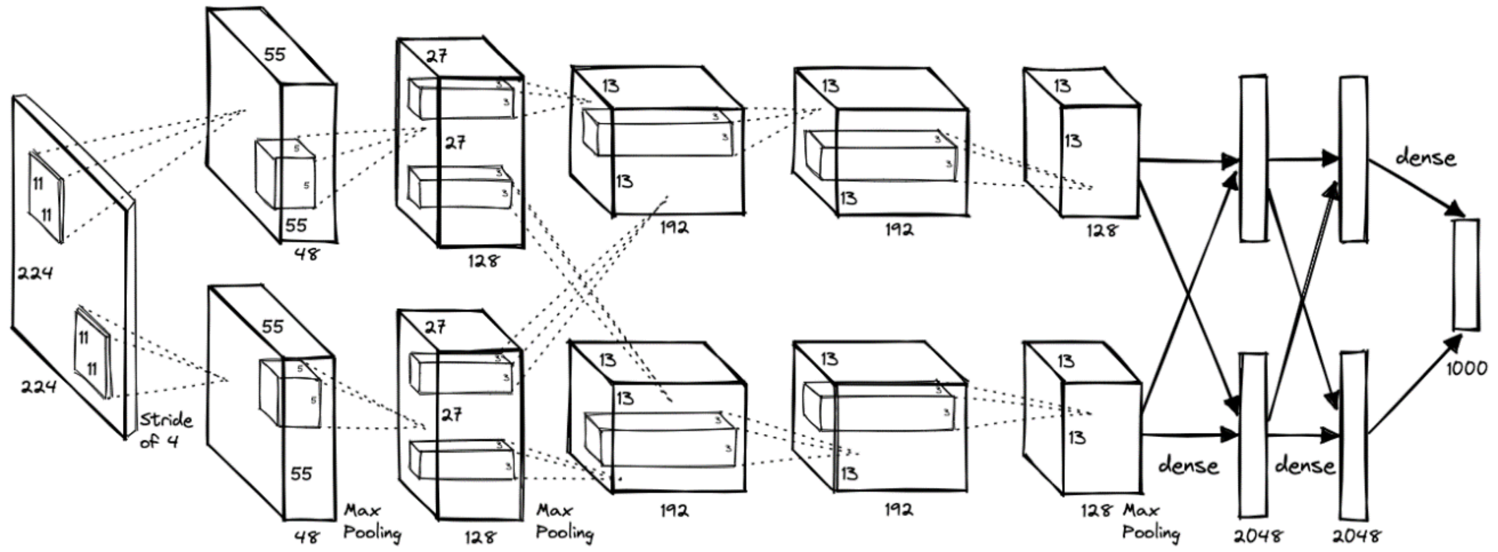
Object Detection: *Multiple Objects*



Dog? NO
Cat? NO
Background? YES

Apply a CNN to many different crops of the image,
CNN classifies each crop as object or background

Object Detection: *Multiple Objects*



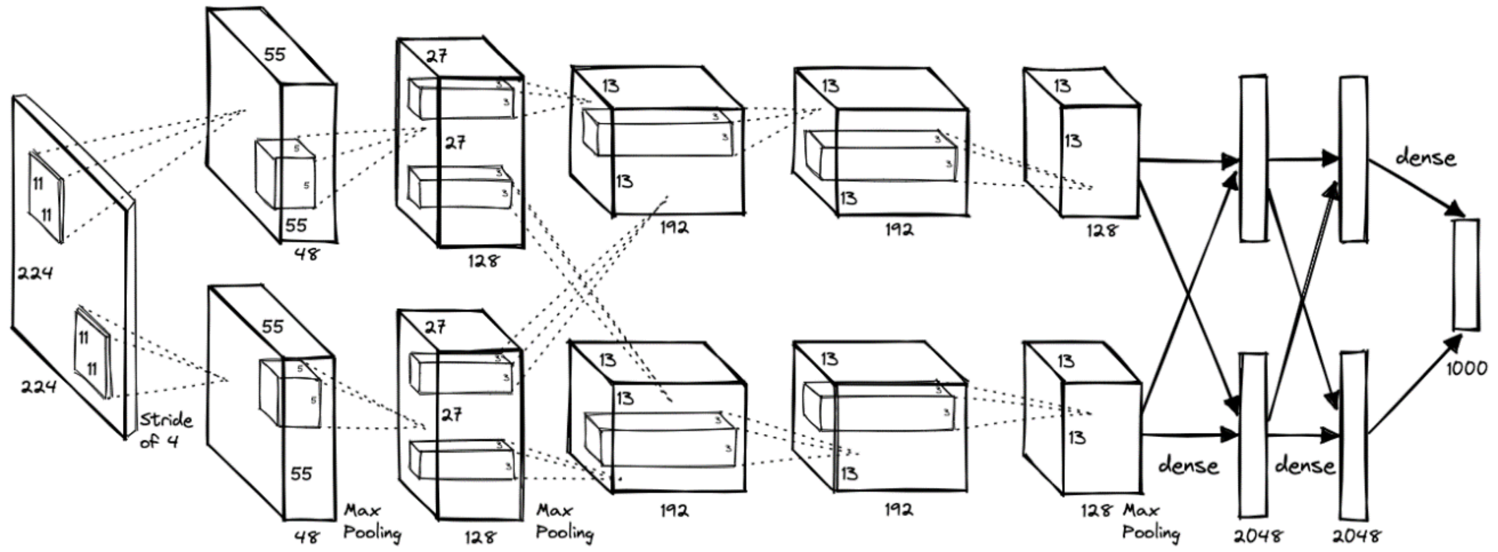
Dog? YES
Cat? NO
Background? NO

Apply a CNN to many different crops of the image,
CNN classifies each crop as object or background

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Object Detection: *Multiple Objects*



Dog? YES
Cat? NO
Background? NO

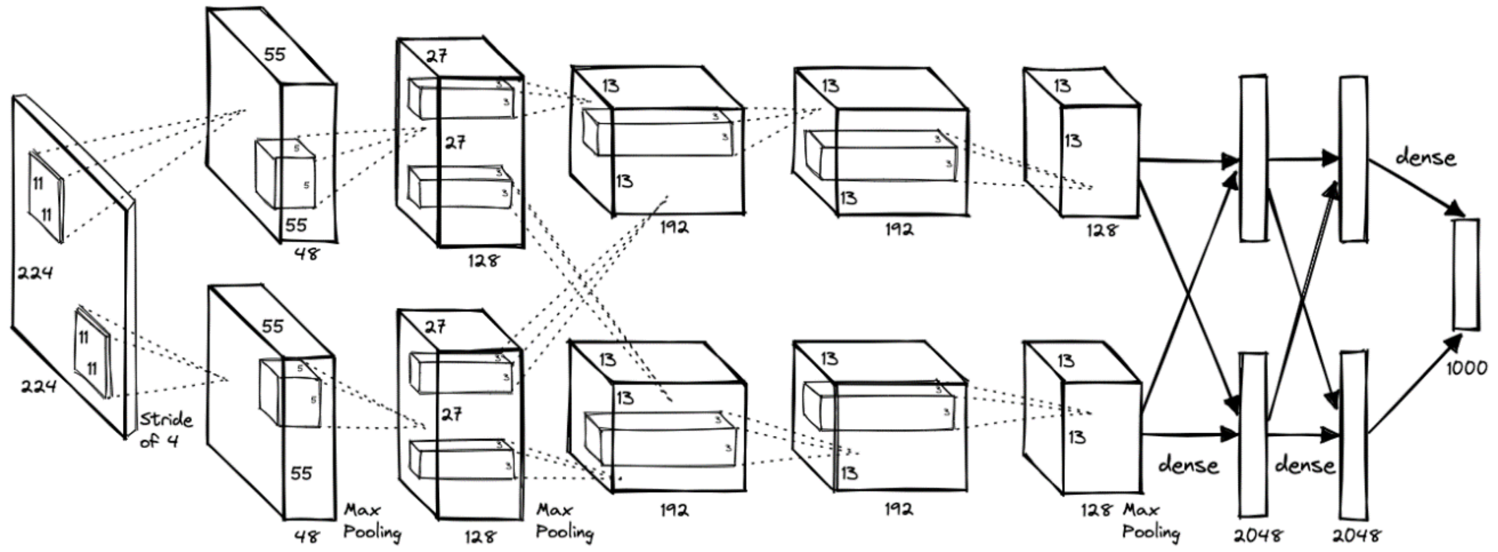
Apply a CNN to many different crops of the image,
CNN classifies each crop as object or background

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Object Detection: *Multiple Objects*

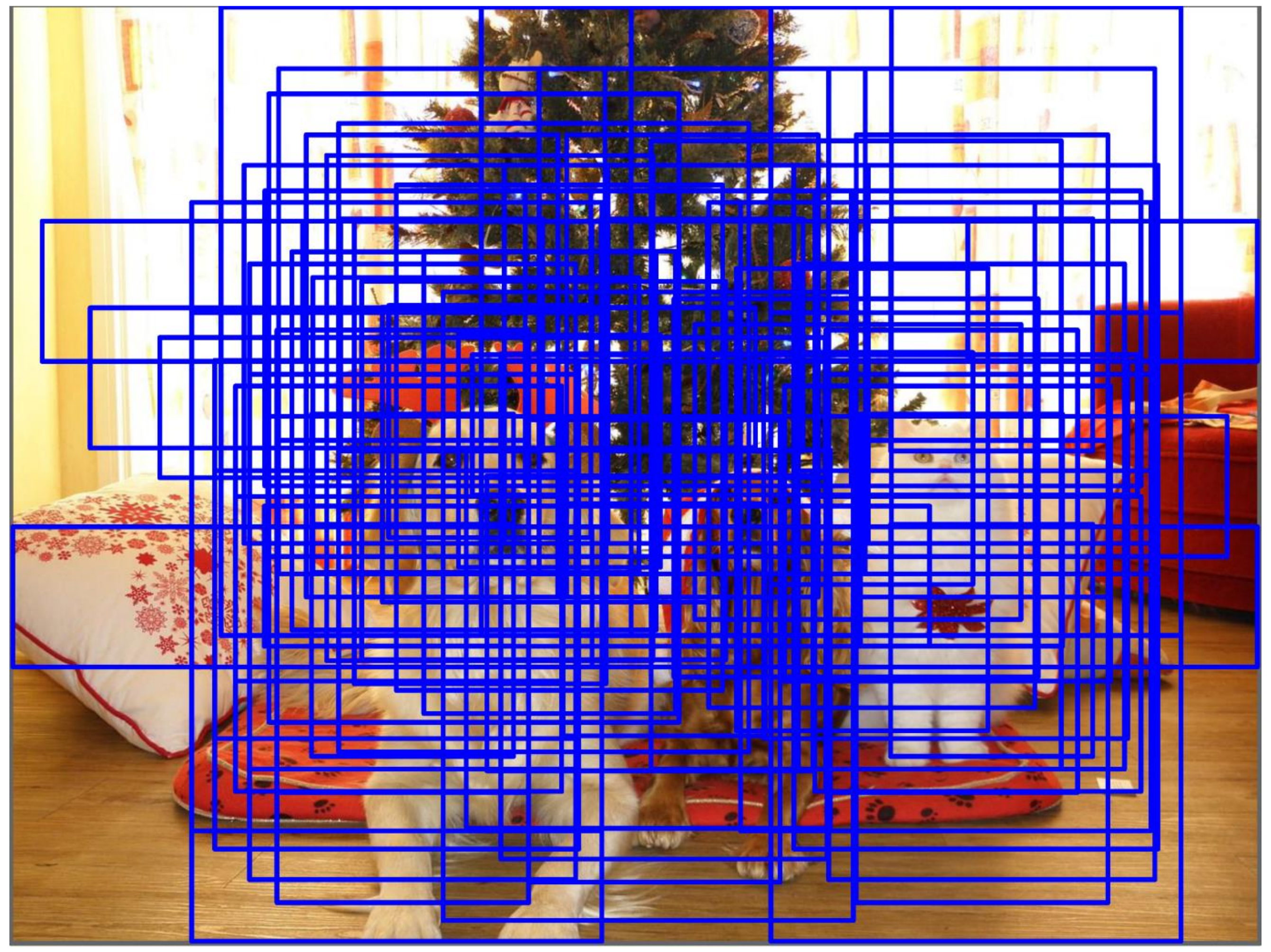
Q: What’s the problem with this approach?



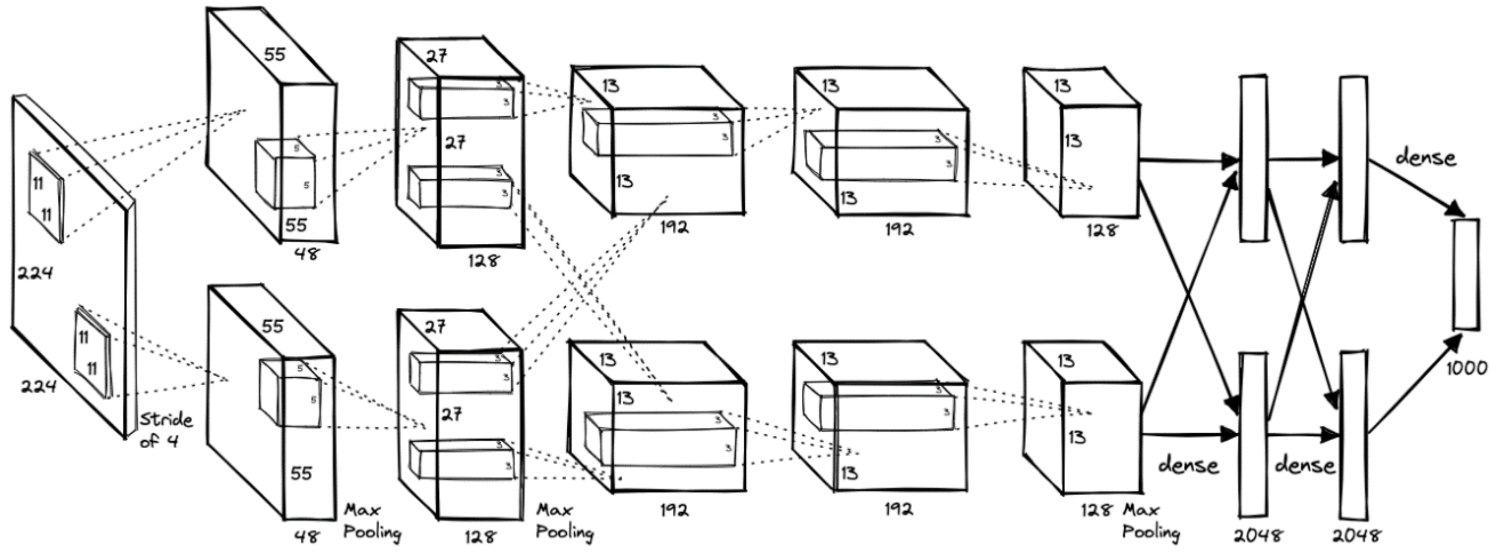
Dog? NO
Cat? YES
Background? NO

Apply a CNN to many different crops of the image,
CNN classifies each crop as object or background

Object Detection: *Multiple Objects*



Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!



Dog? NO
Cat? YES
Background? NO

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Region Proposals: *Selective Search*

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU

Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

R-CNN (Region-based Convolutional Neural Network)

R-CNN (Region-based Convolutional Neural Network) is a popular object detection framework that consists of three main components: region proposal, feature extraction, and object classification.

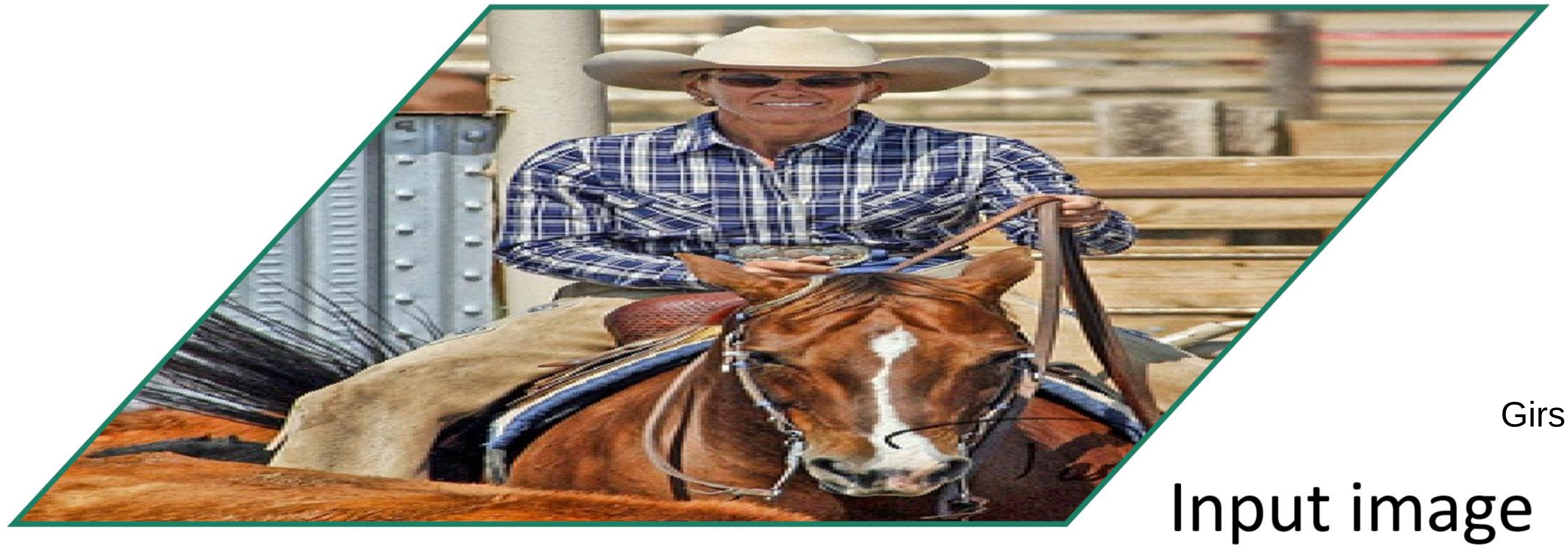
Here's how R-CNN works:

1. **Region Proposal:** First, the image is divided into regions using a selective search algorithm. The algorithm analyzes the image at multiple scales and identifies regions that are likely to contain objects.
2. **Feature Extraction:** Next, a convolutional neural network (CNN) is used to extract a fixed-length feature vector from each region proposal. The CNN is typically pre-trained on a large dataset such as ImageNet to learn a generic set of features that can be used for a wide range of object detection tasks.
3. **Object Classification:** Finally, a set of support vector machines (SVMs) are trained to classify the extracted features into different object categories. A separate SVM is trained for each object category of interest.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

R-CNN



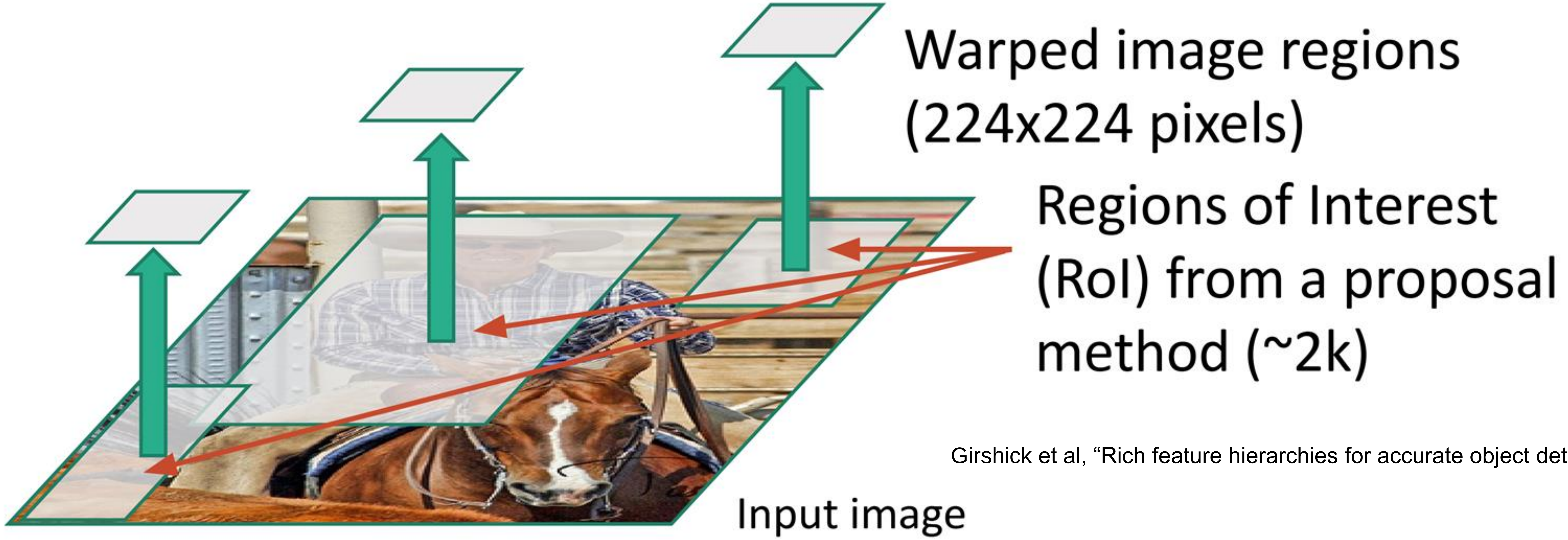
Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN

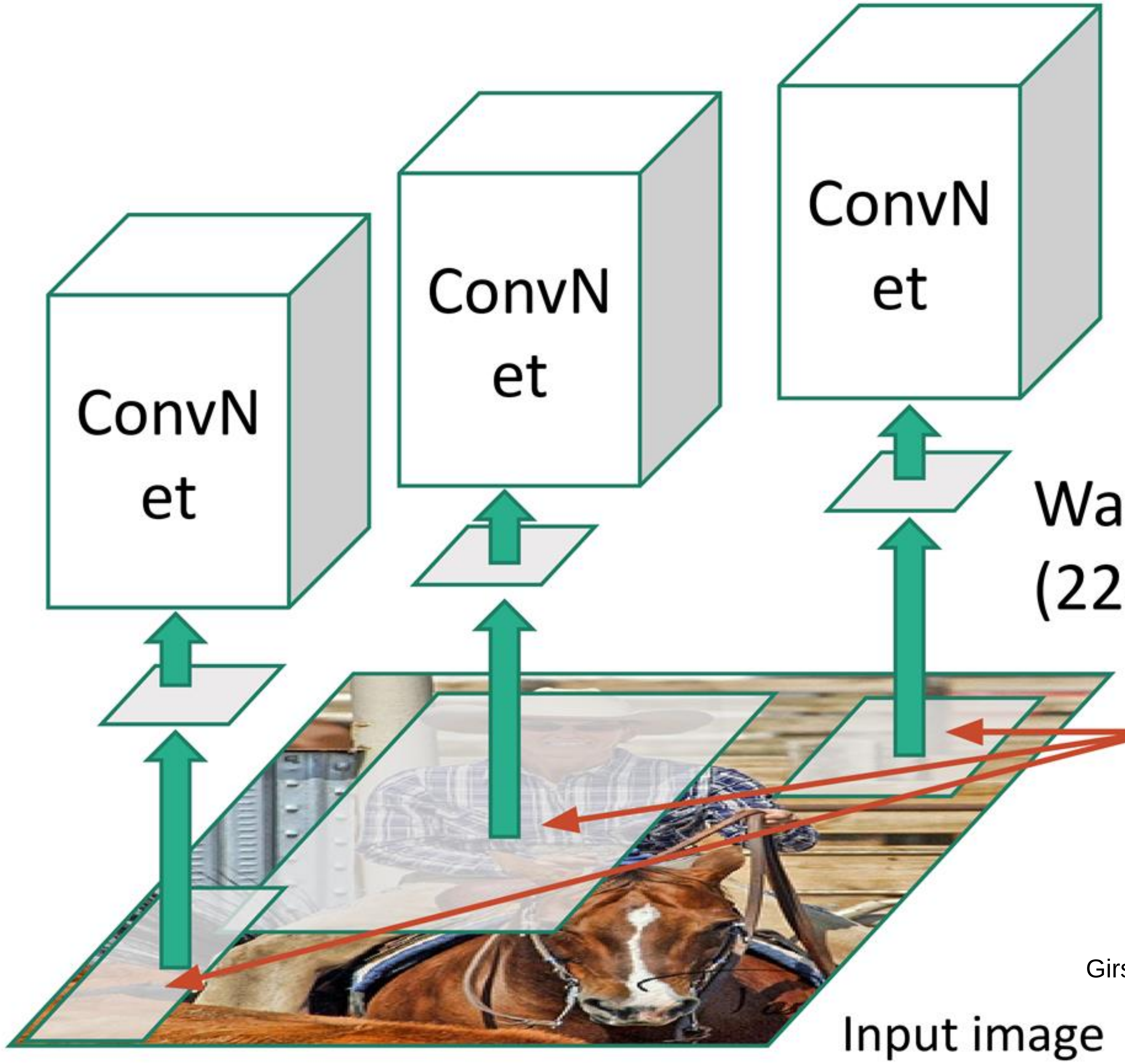
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



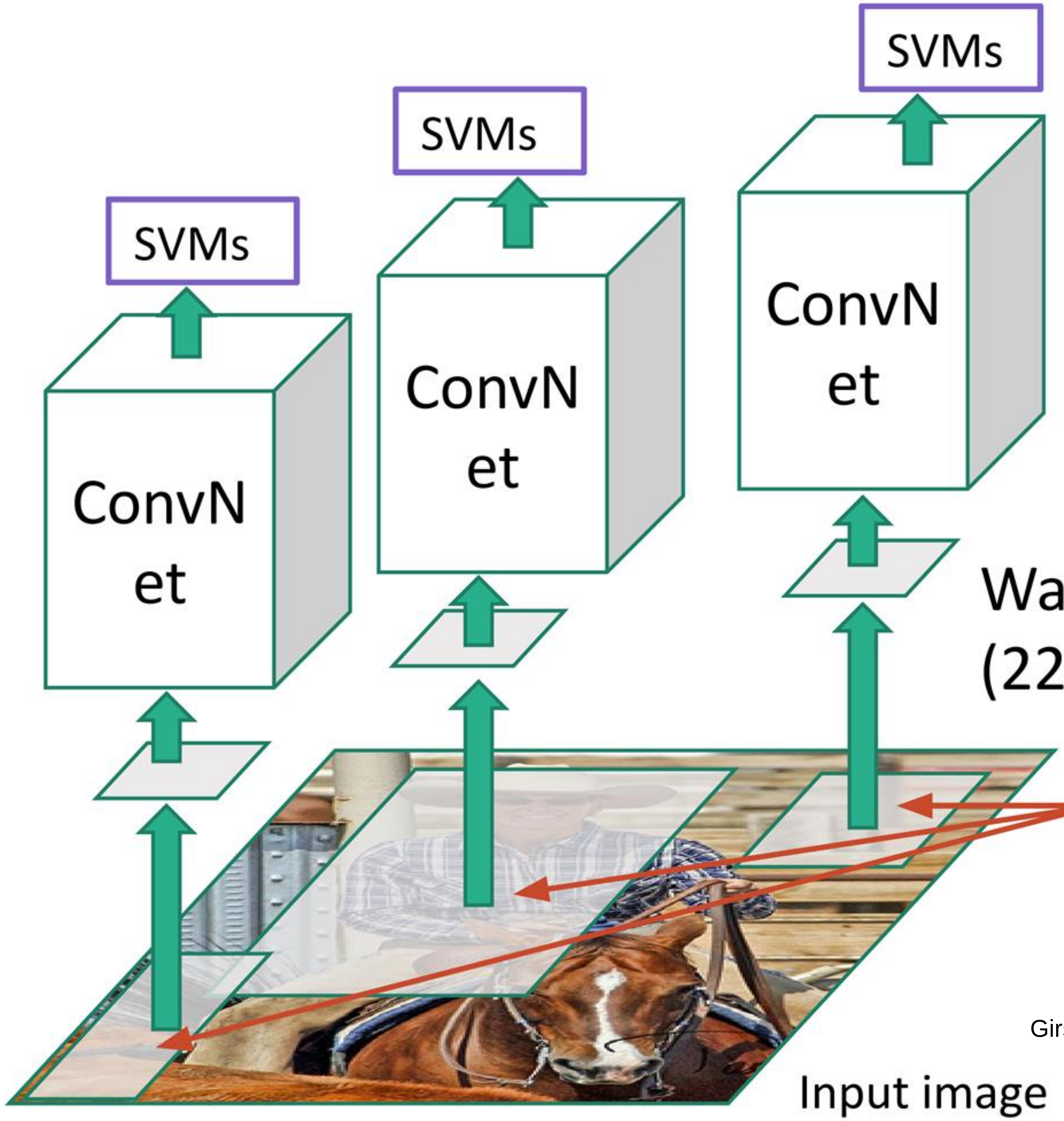
Forward each region through ConvNet (ImageNet-pretranied)

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



Classify regions with SVMs

Forward each region through ConvNet (ImageNet-pretranied)

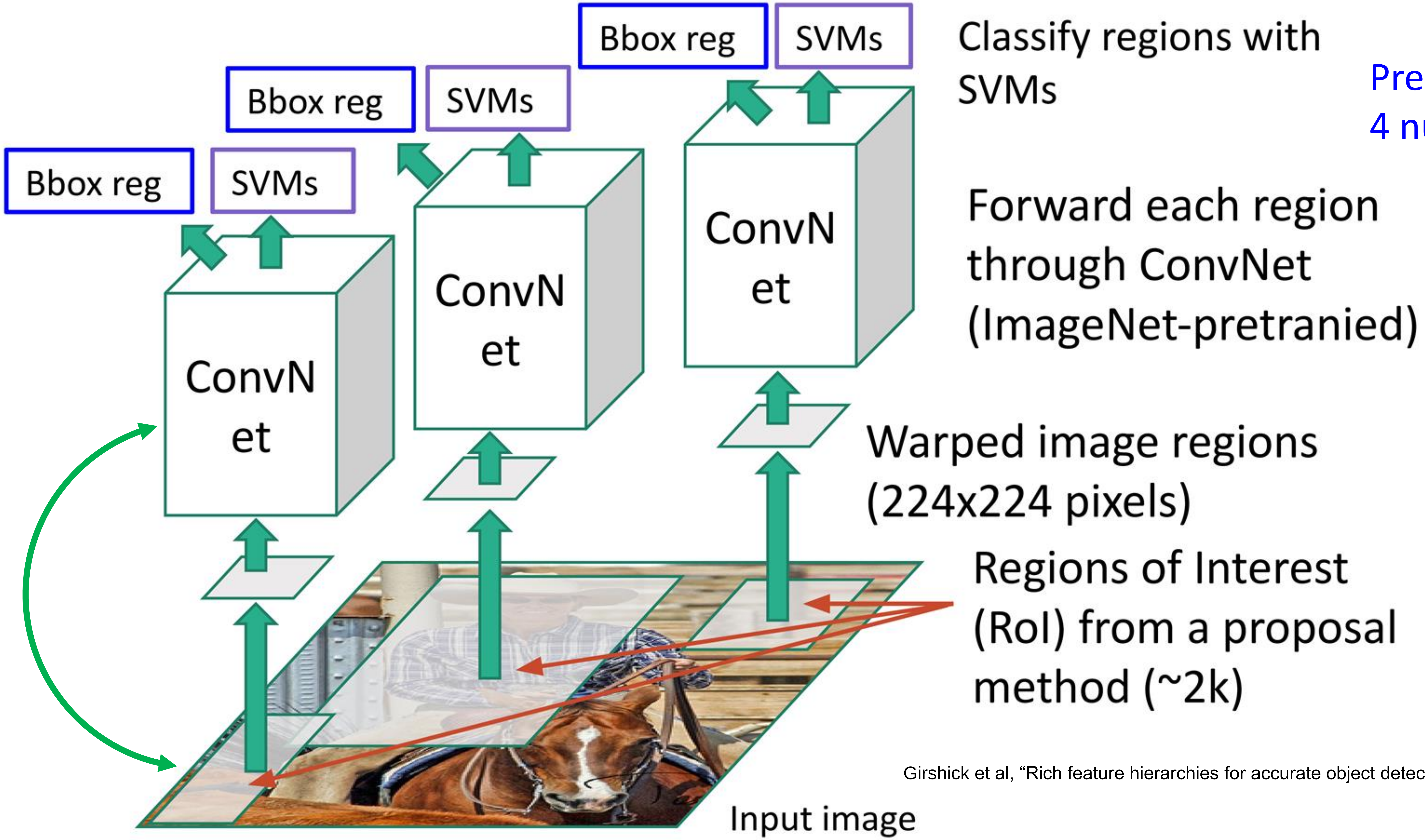
Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)

Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN: *Limitations*

1. **Slow Training and Inference:** R-CNN is a slow method since it requires multiple stages to detect objects. The selective search algorithm used for region proposal is computationally expensive, and the CNN feature extraction also takes a lot of time. This makes R-CNN unsuitable for real-time object detection applications.
2. **High Memory Usage:** R-CNN requires a lot of memory to store the intermediate results from the selective search algorithm and the CNN feature extraction. This can be a problem when processing high-resolution images or when running on devices with limited memory.
3. **Non-End-to-End Training:** The training of R-CNN is not end-to-end, which means that the different components are trained independently. This can lead to suboptimal performance since the different components may not be optimized for the overall task.
4. **Difficulty Handling Overlapping Objects:** R-CNN can struggle with detecting overlapping objects since the selective search algorithm tends to generate a lot of proposals, and the CNN feature extraction may not be able to distinguish between different objects that are close together.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Fast R-CNN

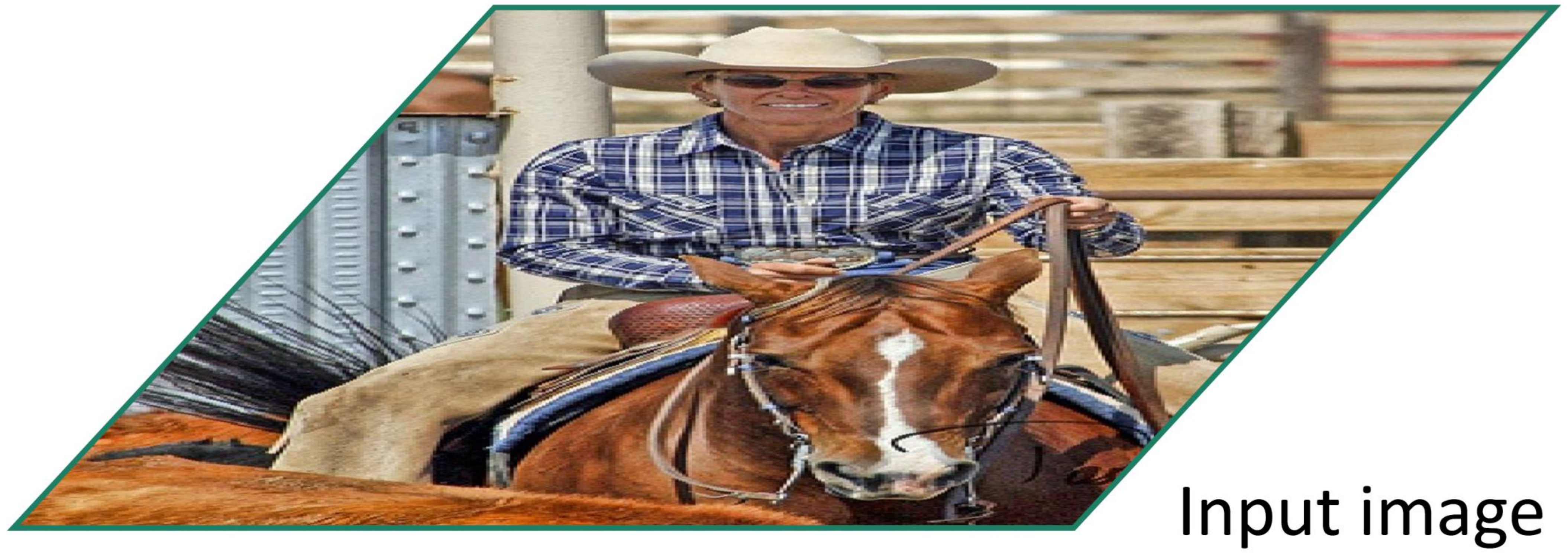
Fast R-CNN is an improvement over the original R-CNN framework that addresses some of its limitations, particularly its slow training and inference times. Here's how Fast R-CNN works:

1. **Region Proposal:** Similar to R-CNN, the image is divided into regions using a selective search algorithm.
2. **Feature Extraction:** Instead of using a separate CNN for each region proposal, Fast R-CNN uses a single CNN to extract features from the entire image. The selective search regions are then warped to a fixed size and fed into the CNN as input.
3. **RoI Pooling:** Fast R-CNN uses a region of interest (RoI) pooling layer to extract a fixed-length feature vector from each region proposal. The RoI pooling layer divides each proposal into a fixed number of sub-windows and applies max-pooling to each sub-window, resulting in a fixed-size feature map that can be fed into a fully connected layer.
4. **Object Classification and Localization:** Fast R-CNN uses a single network to perform both object classification and localization. The output of the RoI pooling layer is fed into a series of fully connected layers that produce class probabilities and bounding box coordinates for each region proposal.

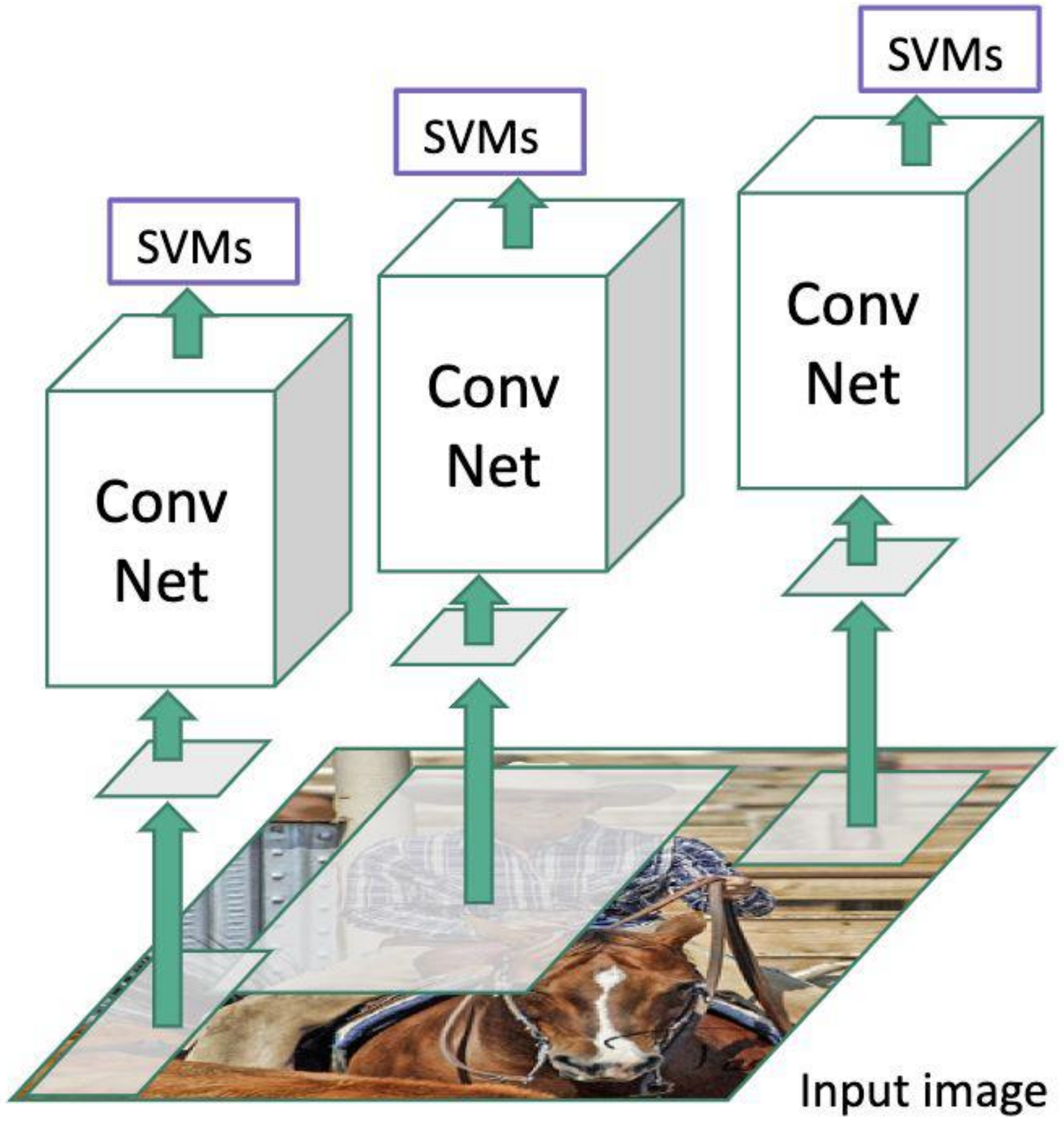
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Fast R-CNN



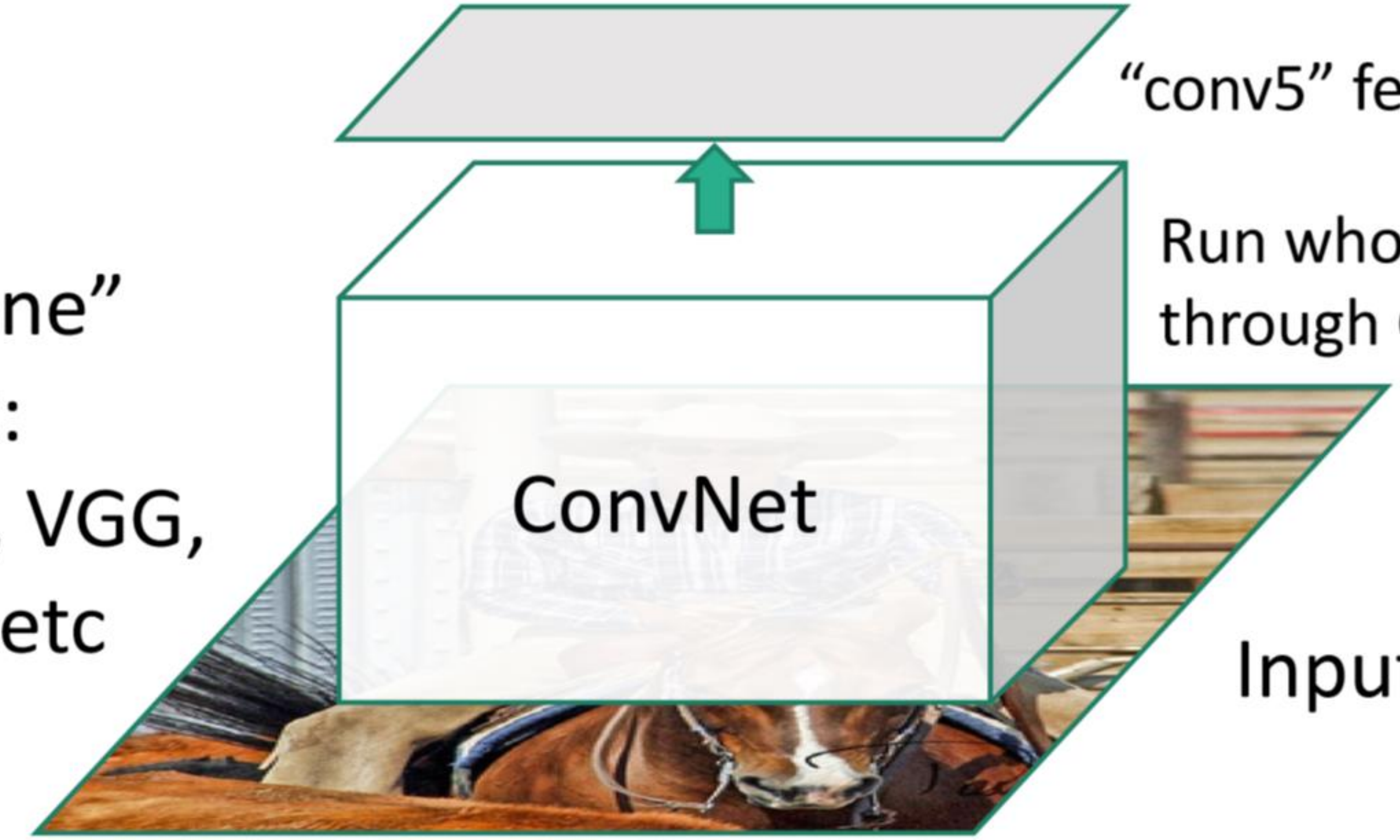
“Slow” R-CNN



Girshick, “Fast R-CNN”, ICCV 2015.

Fast R-CNN

“Backbone” network: AlexNet, VGG, ResNet, etc

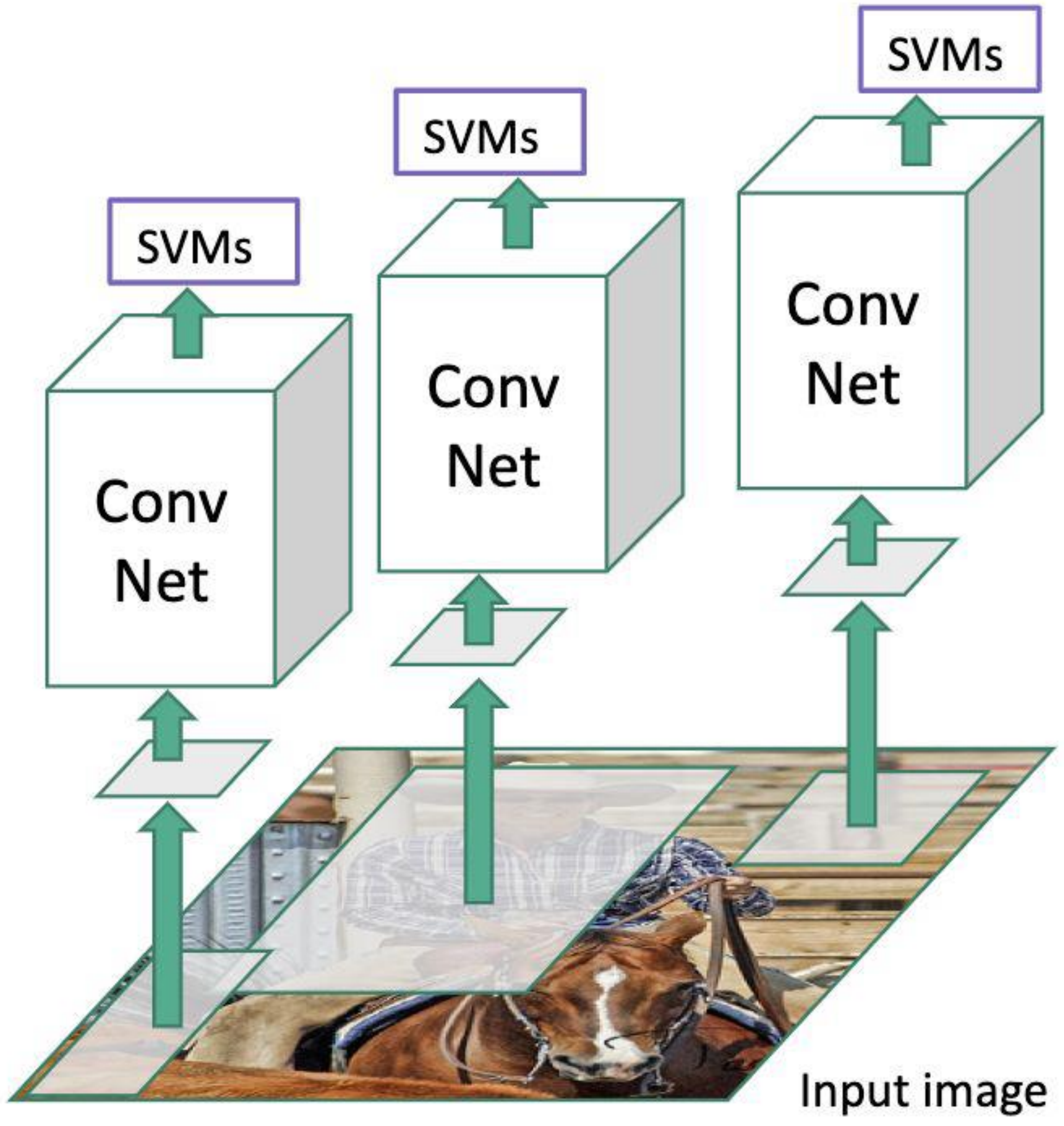


“conv5” features

Run whole image through ConvNet

Input image

“Slow” R-CNN

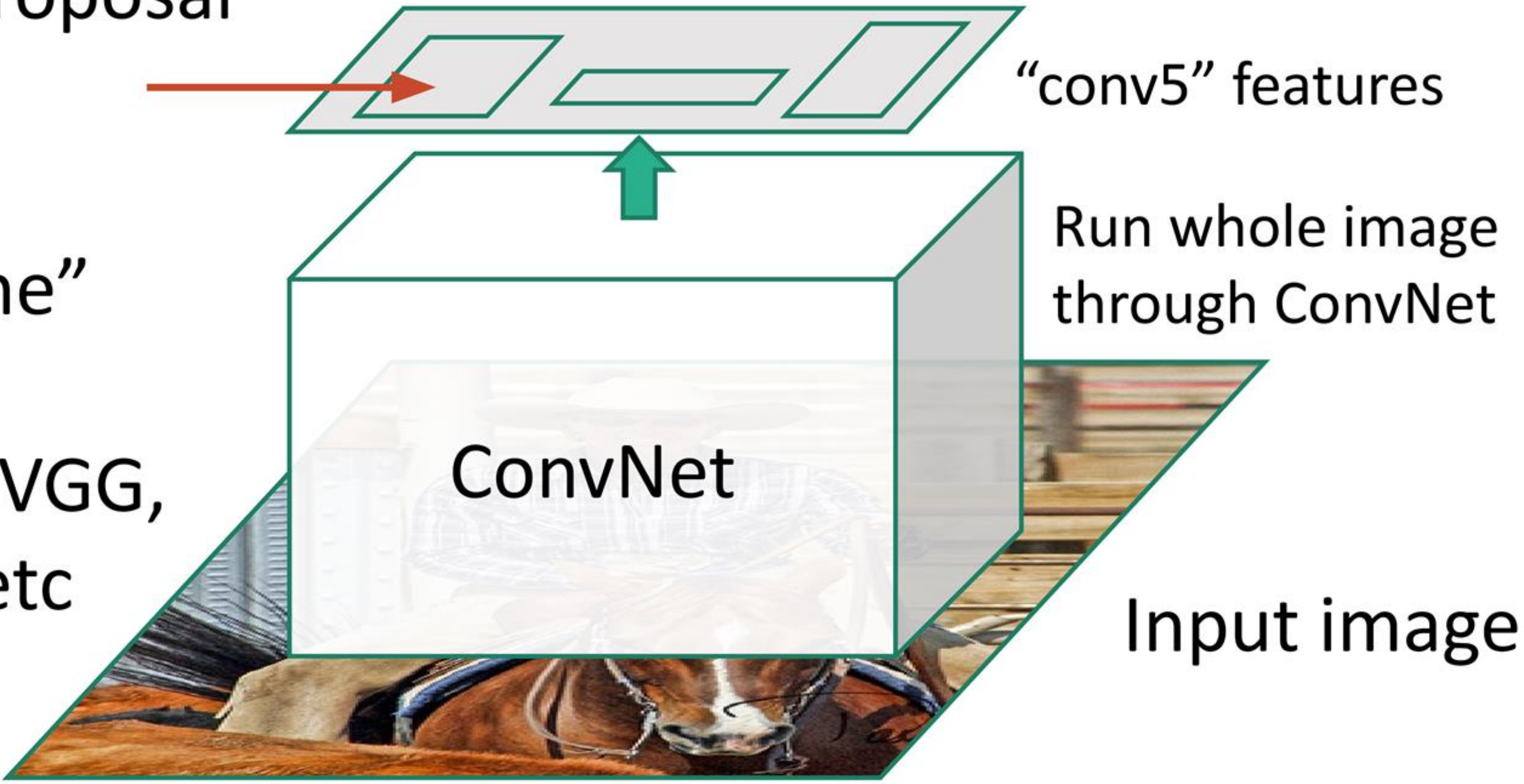


Girshick, “Fast R-CNN”, ICCV 2015.

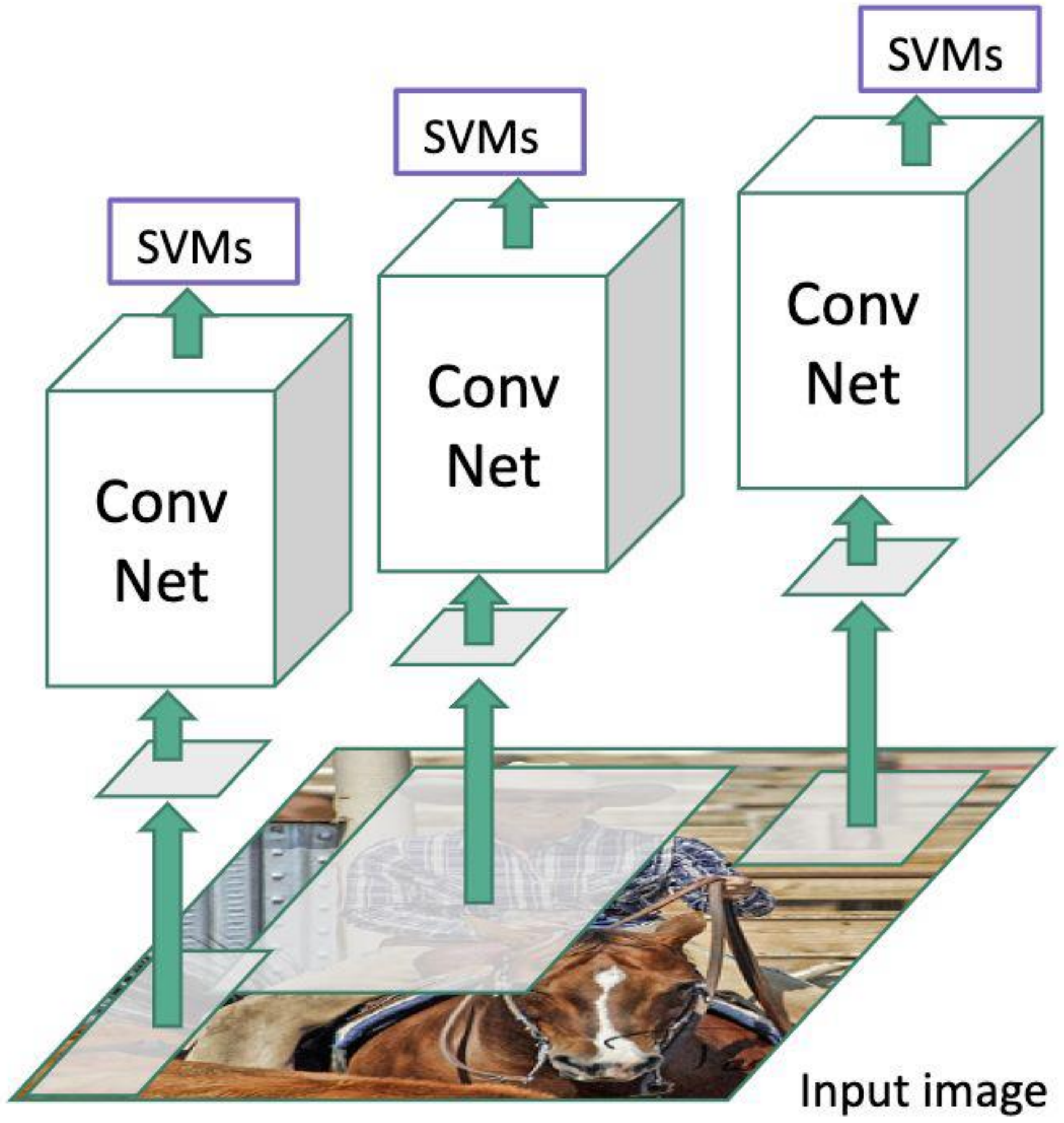
Fast R-CNN

Regions of Interest (RoIs) from a proposal method

“Backbone” network: AlexNet, VGG, ResNet, etc



“Slow” R-CNN

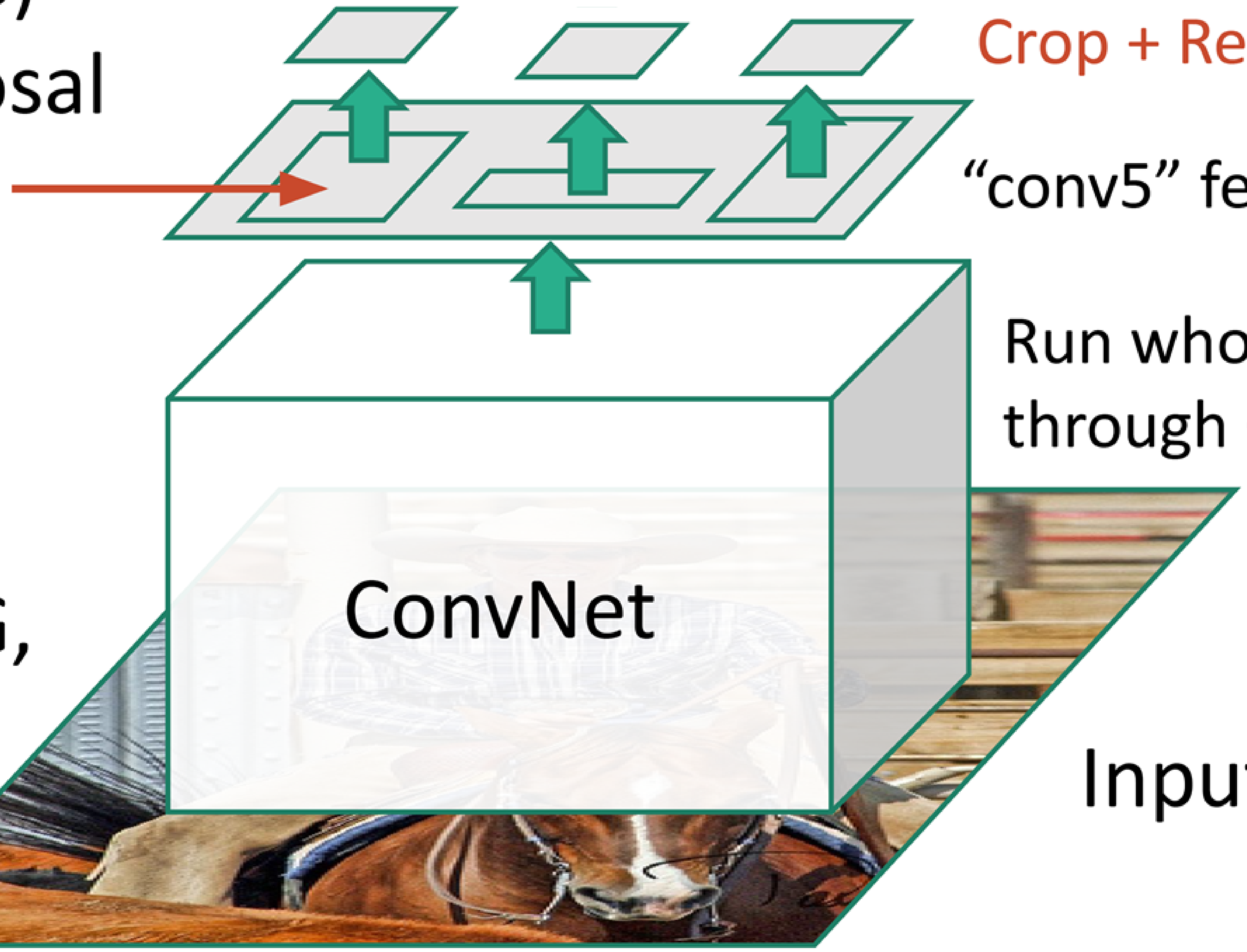


Girshick, “Fast R-CNN”, ICCV 2015.

Fast R-CNN

Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc



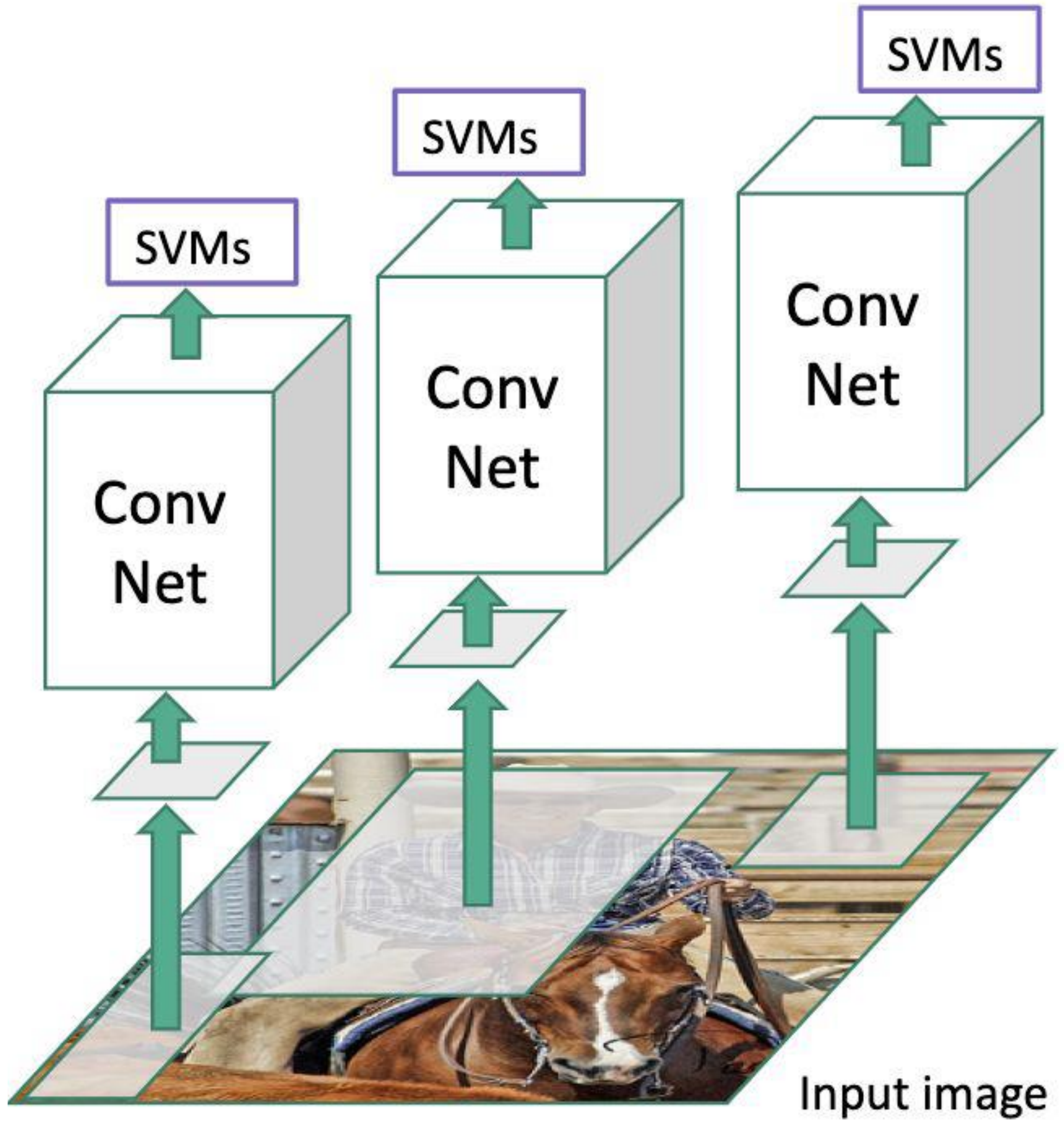
Crop + Resize features

"conv5" features

Run whole image through ConvNet

Input image

"Slow" R-CNN

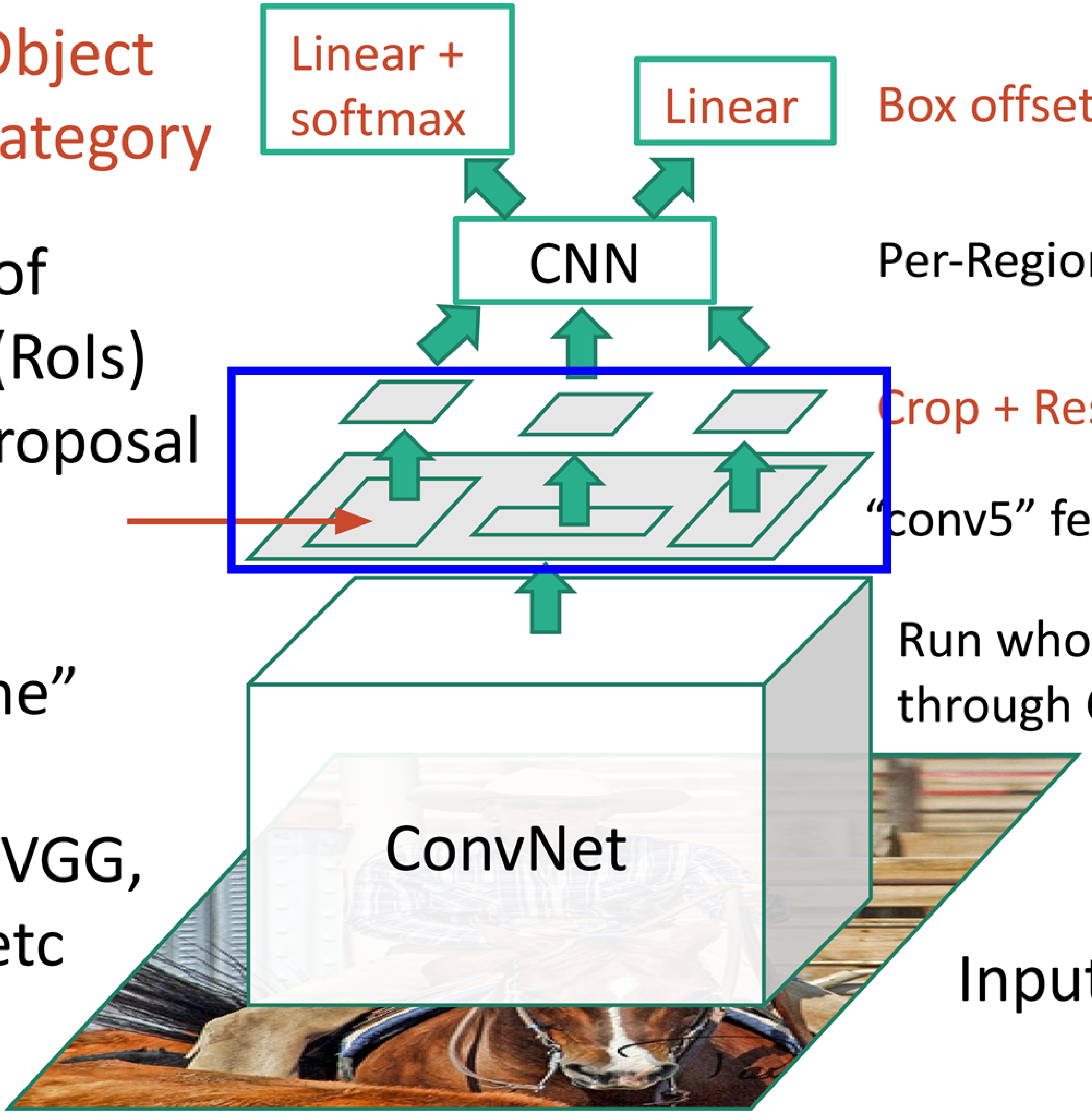


Girshick, "Fast R-CNN", ICCV 2015.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Fast R-CNN



Object category

Box offset

Per-Region Network

Crop + Resize features

"conv5" features

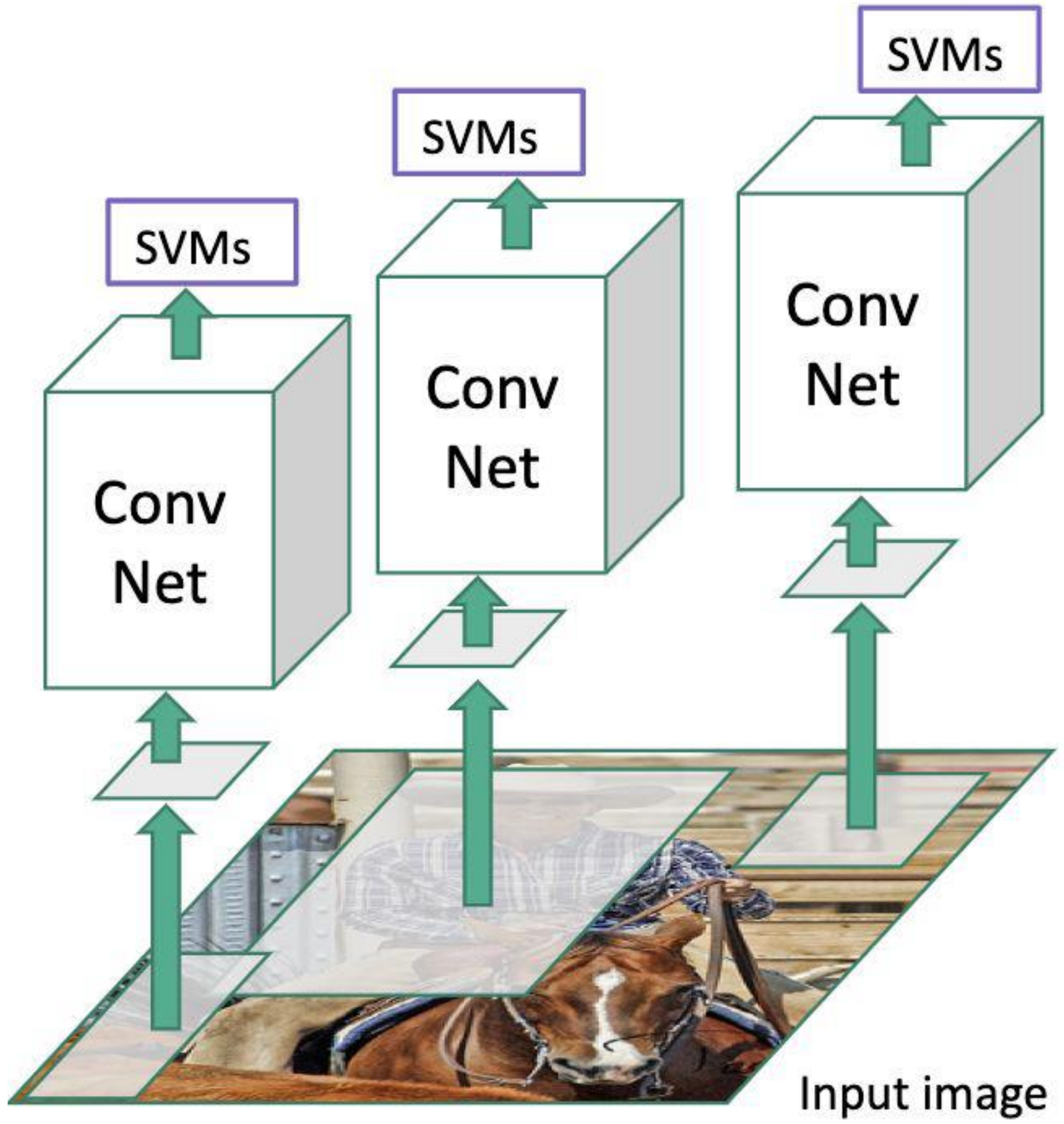
Run whole image through ConvNet

Input image

Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc

"Slow" R-CNN



Girshick, "Fast R-CNN", ICCV 2015.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

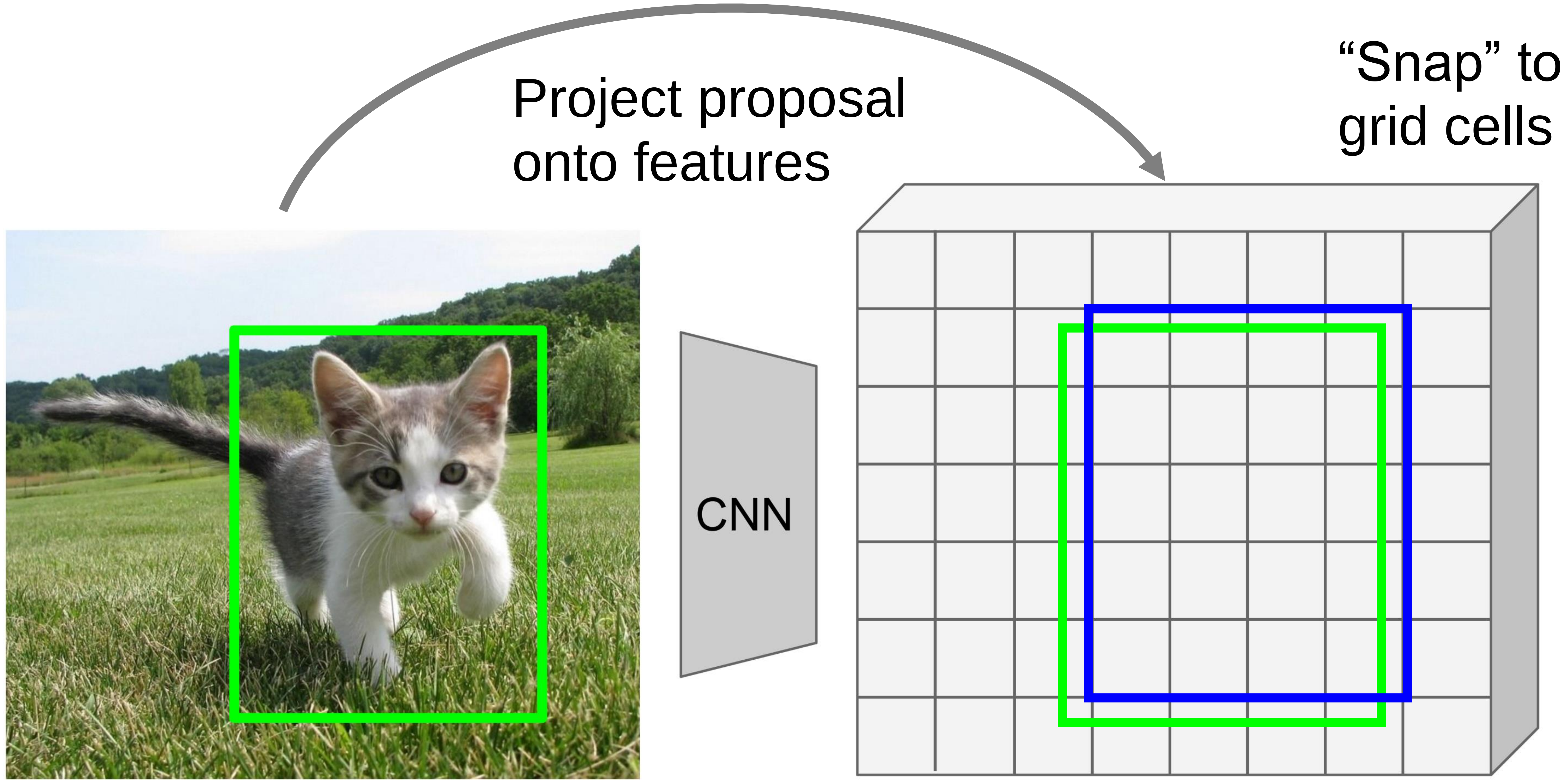
Fast R-CNN: *Advantages over R-CNN*

1. **Faster Training and Inference:** Since Fast R-CNN uses a single CNN to extract features from the entire image, it can process multiple region proposals in parallel, resulting in faster training and inference times compared to R-CNN.
2. **End-to-End Training:** Fast R-CNN enables end-to-end training of the entire system, which means that all components are optimized together to improve overall performance.
3. **Better Localization Accuracy:** Fast R-CNN uses the RoI pooling layer to extract features from each region proposal, resulting in more accurate object localization compared to R-CNN.
4. **Reduced Memory Usage:** Fast R-CNN requires less memory compared to R-CNN since it only needs to store the feature map for the entire image, instead of storing intermediate results for each region proposal.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Cropping Features: *RoI Pool*



Project proposal onto features

"Snap" to grid cells

Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

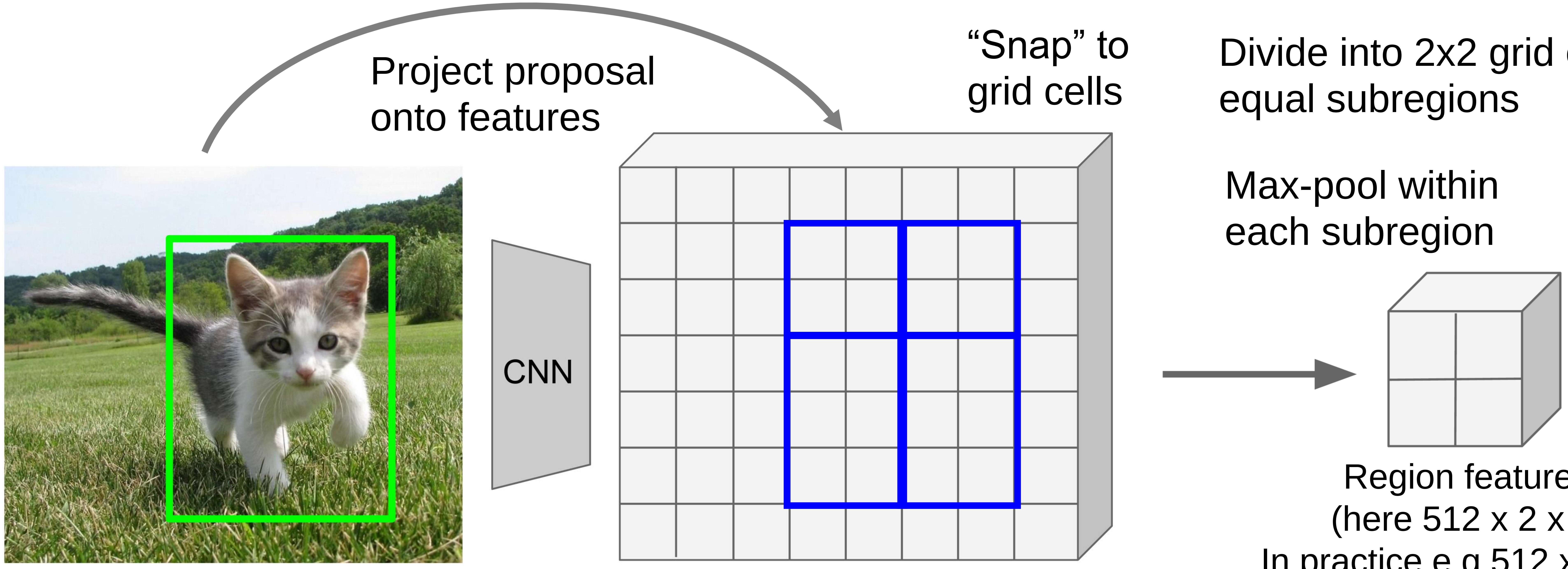
Q: how do we resize the 512 x 5 x 4 region to, e.g., a 512 x 2 x 2 tensor?.

Girshick, "Fast R-CNN", ICCV 2015.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Cropping Features: *RoI Pool*



Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion

Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

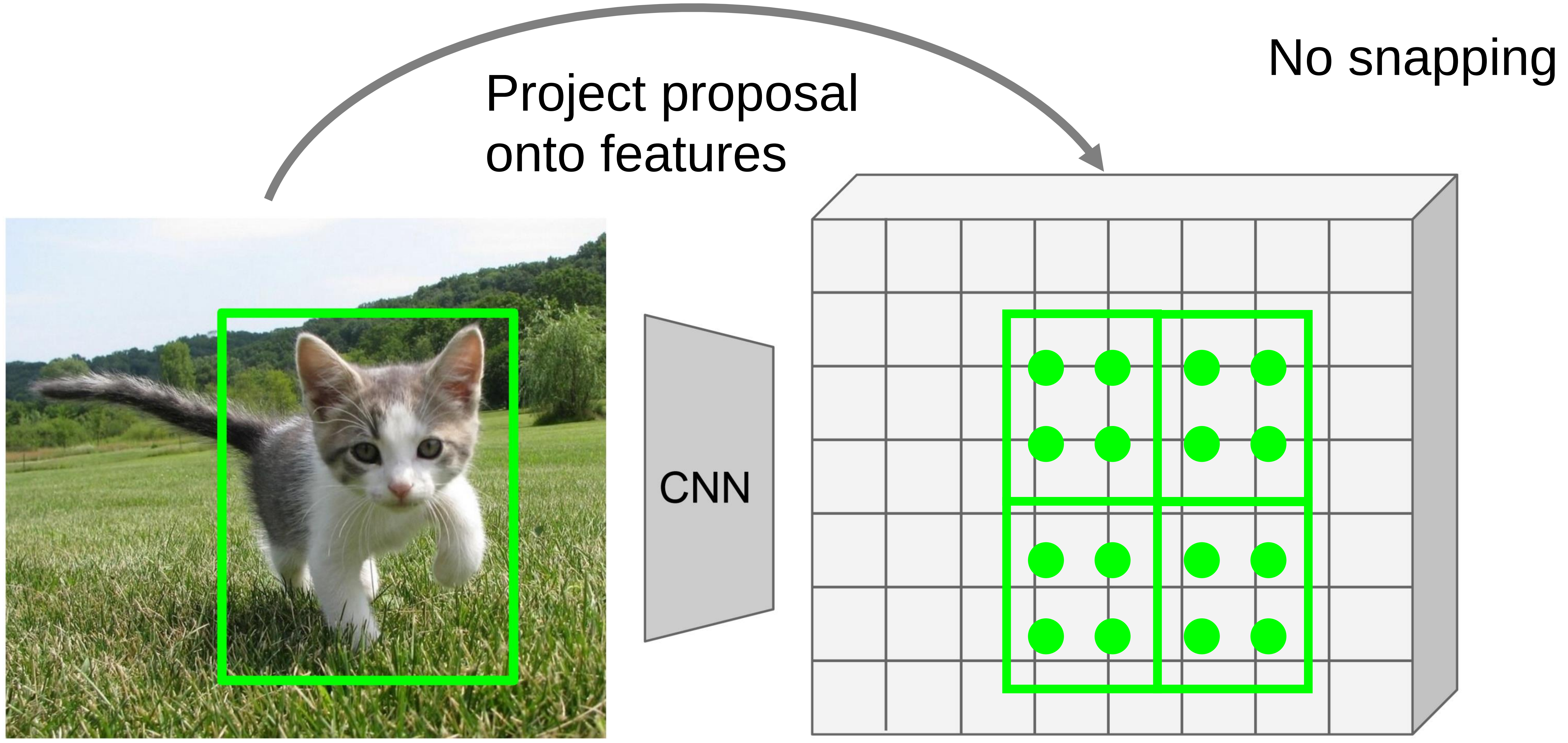
Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Region features always the same size even if input regions have different sizes!

Girshick, "Fast R-CNN", ICCV 2015.

Problem: Region features slightly misaligned

Cropping Features: *RoI Align*



Project proposal
onto features

No snapping

Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

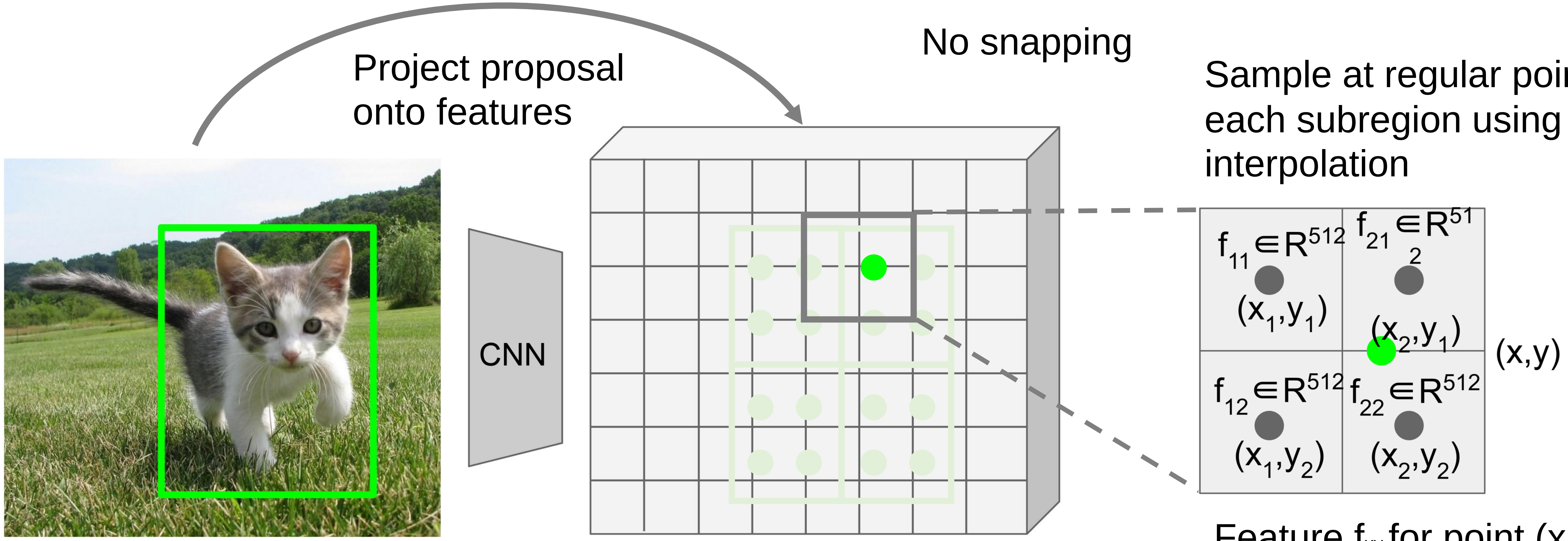
Sample at regular points in each subregion using bilinear interpolation

Girshick, "Fast R-CNN", ICCV 2015.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Cropping Features: *RoI Align*



Project proposal onto features

No snapping

Sample at regular points in each subregion using bilinear interpolation

Input Image
(e.g. 3 x 640 x 480)

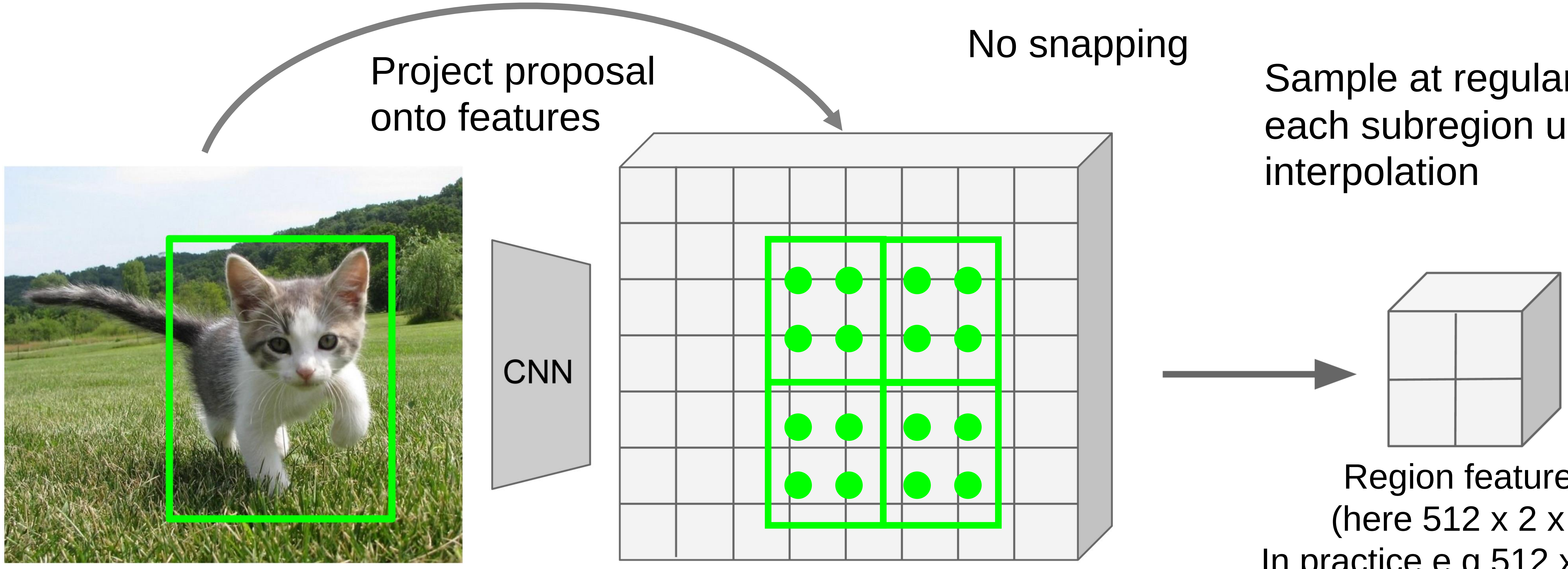
Image features: C x H x W
(e.g. 512 x 20 x 15)

Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

$$f_{xy} = \sum_{i,j=1}^{2} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_j|)$$

Girshick, “Fast R-CNN”, ICCV 2015.

Cropping Features: *RoI Pool*



Project proposal
onto features

No snapping

Sample at regular points in
each subregion using bilinear
interpolation

Max-pool within
each subregion

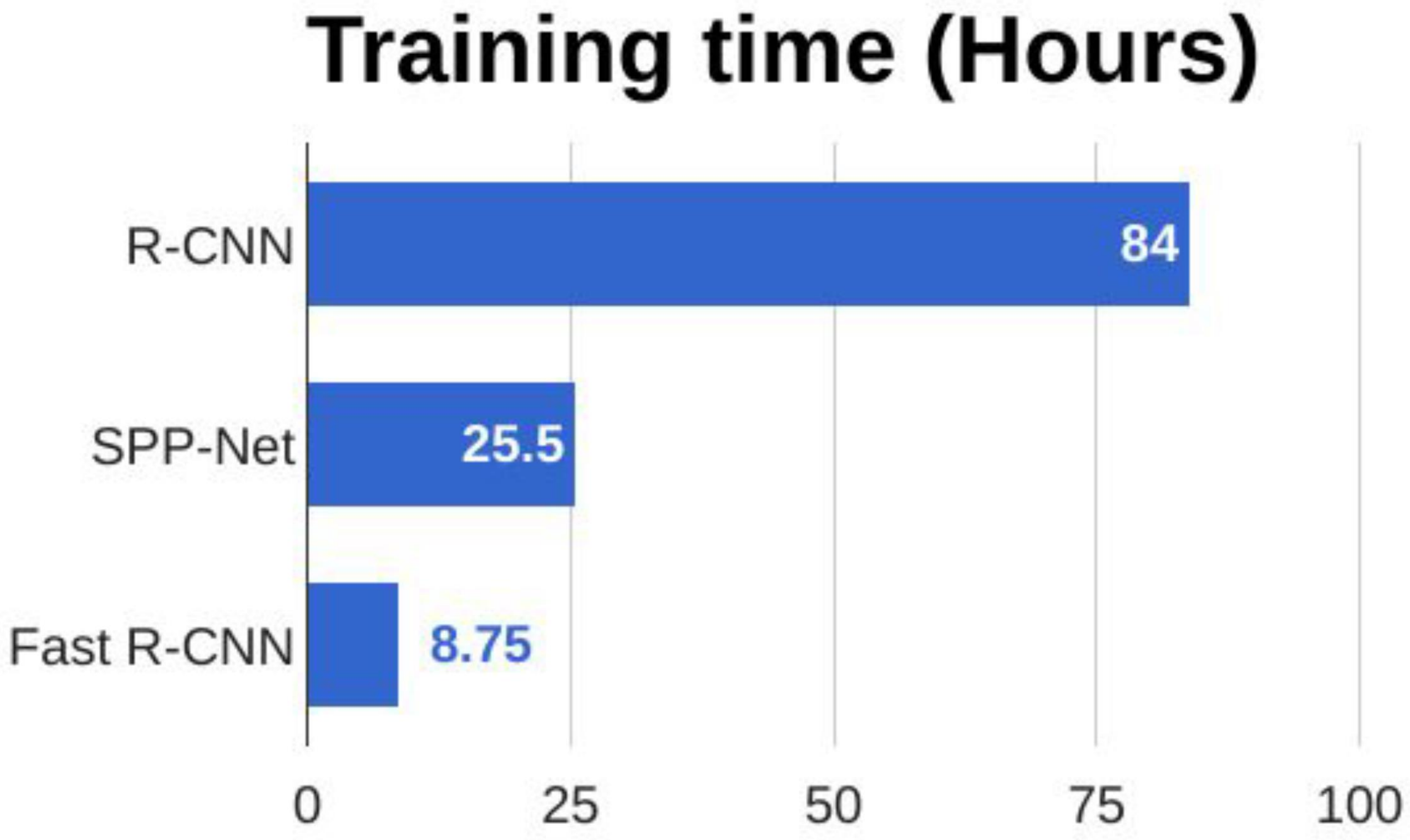
Input Image
(e.g. 3 x 640 x 480)

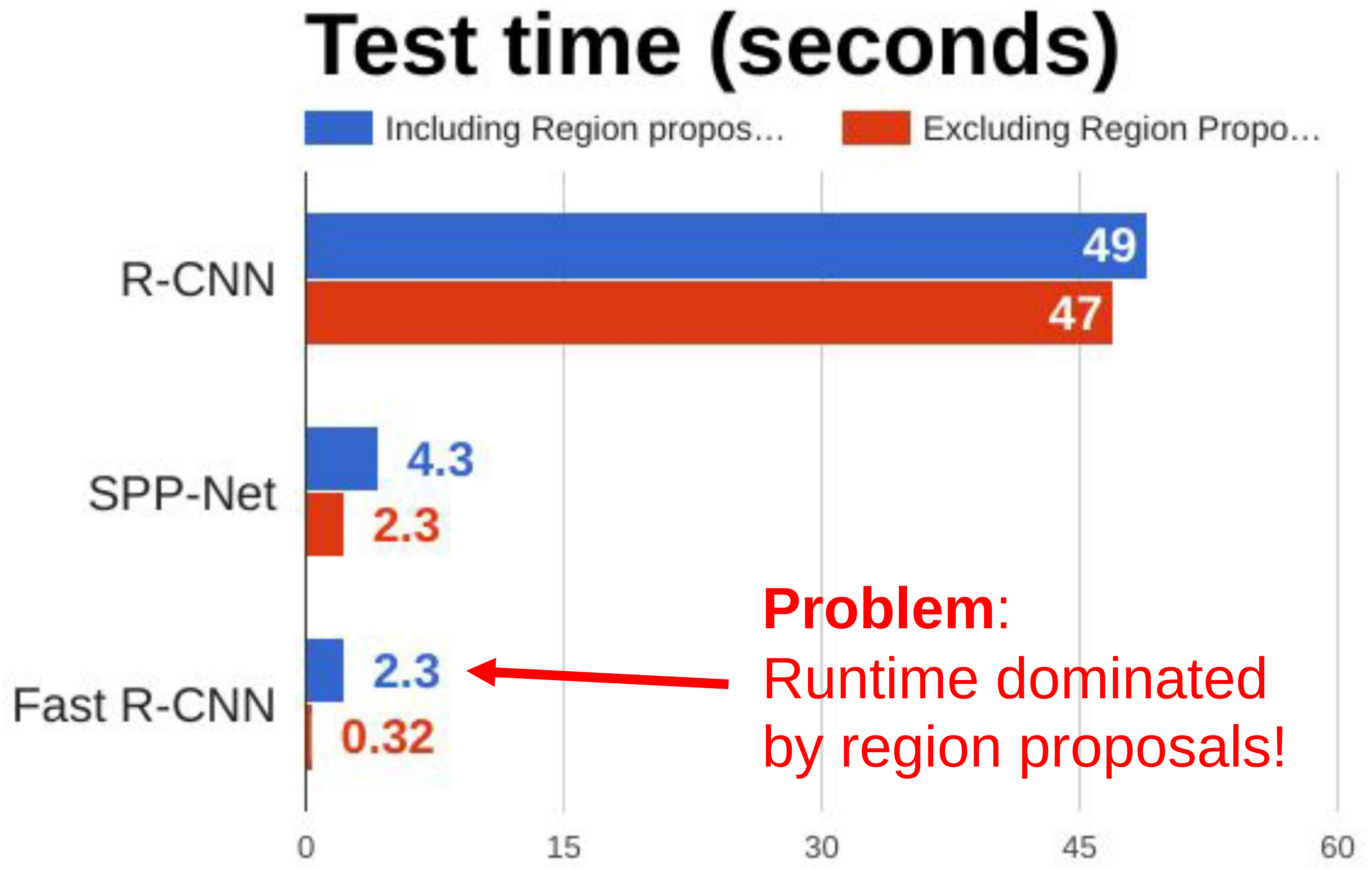
Image features: C x H x W
(e.g. 512 x 20 x 15)

Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Girshick, “Fast R-CNN”, ICCV 2015.

Cropping Features: *RoI Pool*





Problem:
Runtime dominated
by region proposals!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

Faster R-CNN: *Make CNN do proposals!*

Faster R-CNN is another improvement that addresses the limitations of previous networks by introducing a novel region proposal network (RPN). The RPN generates region proposals directly from the feature map, eliminating the need for the computationally expensive selective search algorithm used in R-CNN and Fast R-CNN.

Here's how Faster R-CNN works:

1. **Feature Extraction:** Similar to Fast R-CNN, Faster R-CNN uses a single CNN to extract features from the entire image.
2. **Region Proposal Network:** The RPN is a small CNN that takes the feature map as input and outputs a set of object proposals along with their objectness scores. The proposals are generated by sliding a small network, known as an anchor box, over the feature map at different scales and aspect ratios.
3. **RoI Pooling:** The RoI pooling layer is used to extract a fixed-length feature vector from each region proposal, similar to Fast R-CNN.
4. **Object Classification and Localization:** The output of the RoI pooling layer is fed into a series of fully connected layers that produce class probabilities and bounding box coordinates for each region proposal.

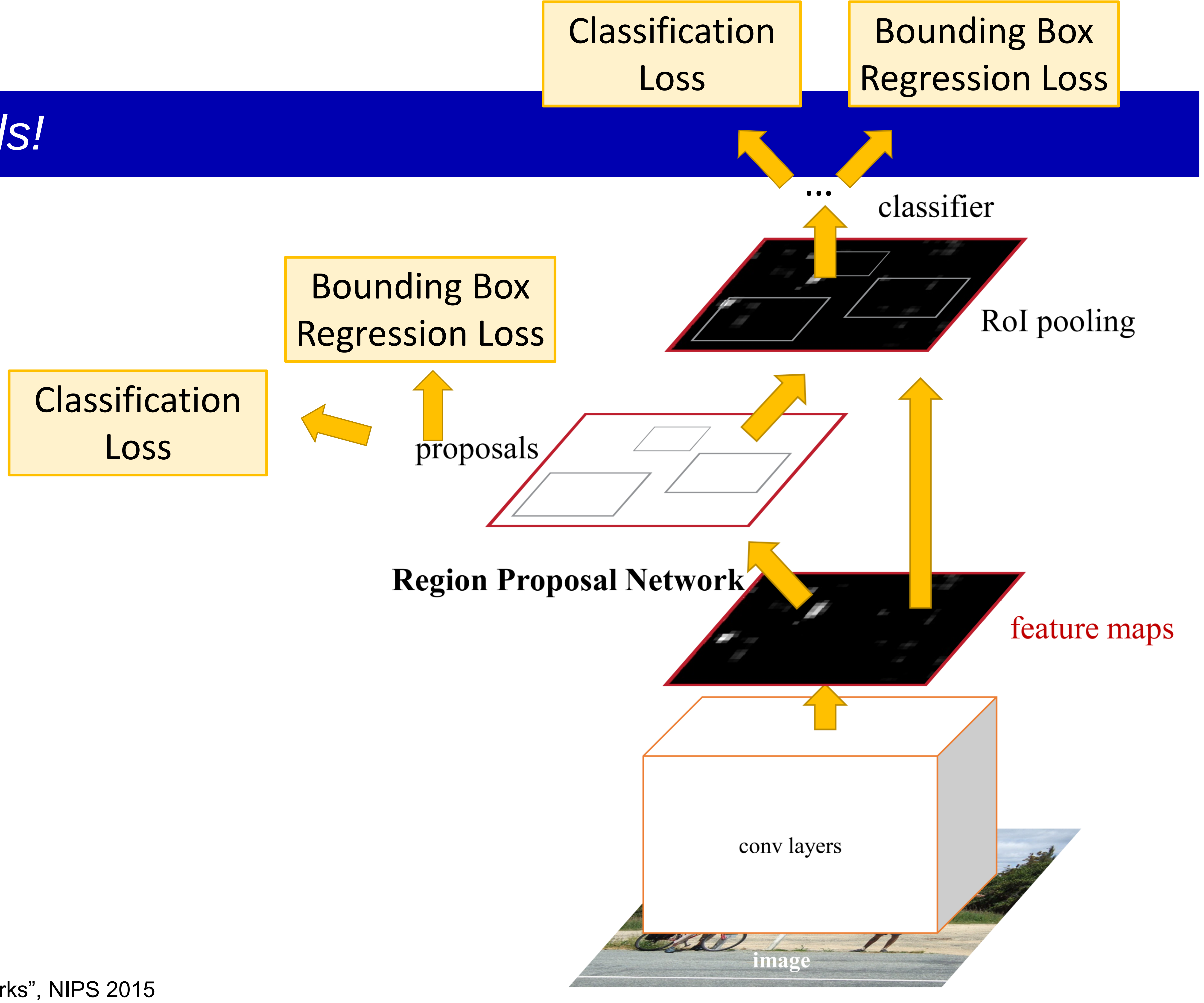
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Faster R-CNN: *Make CNN do proposals!*

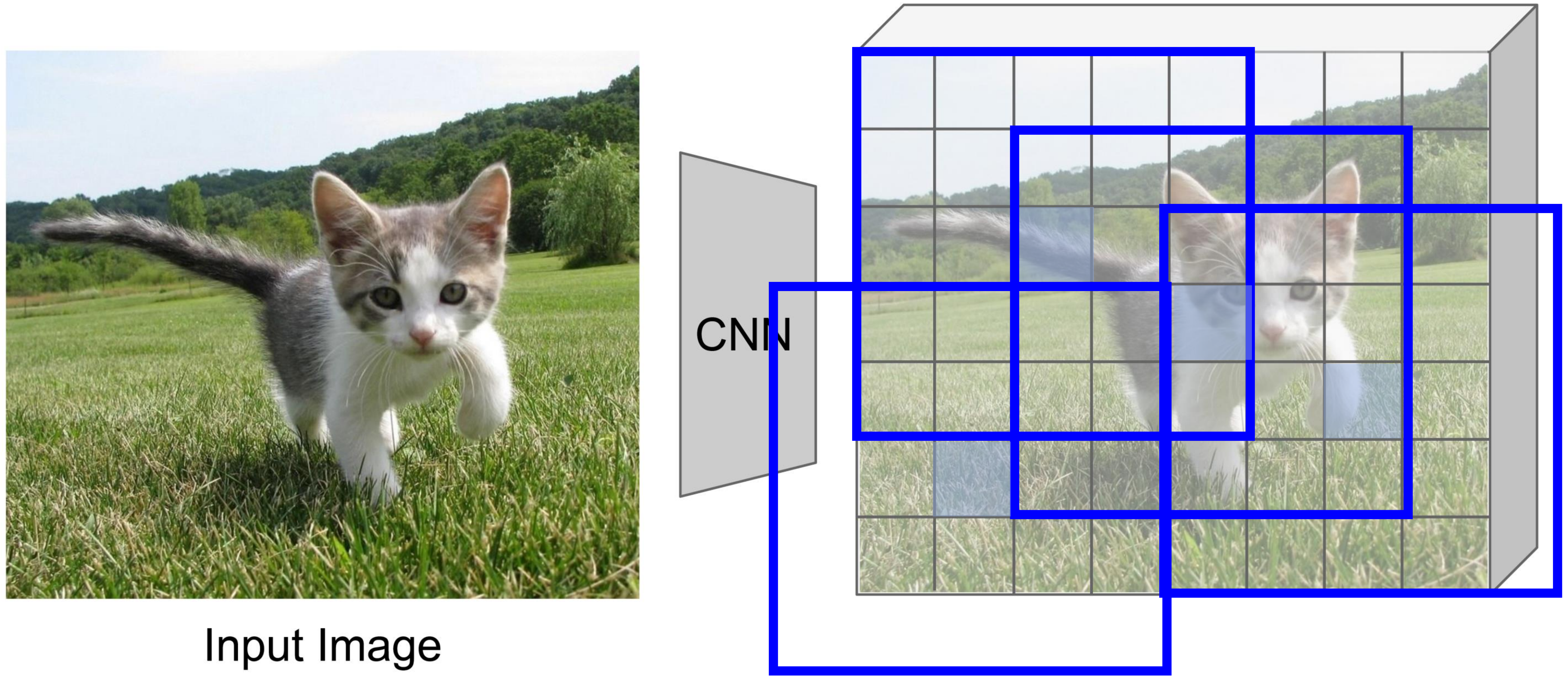
Insert **Region Proposal Network (RPN)** to predict proposals from features

Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network



Input Image
(e.g. 3 x 640 x 480)

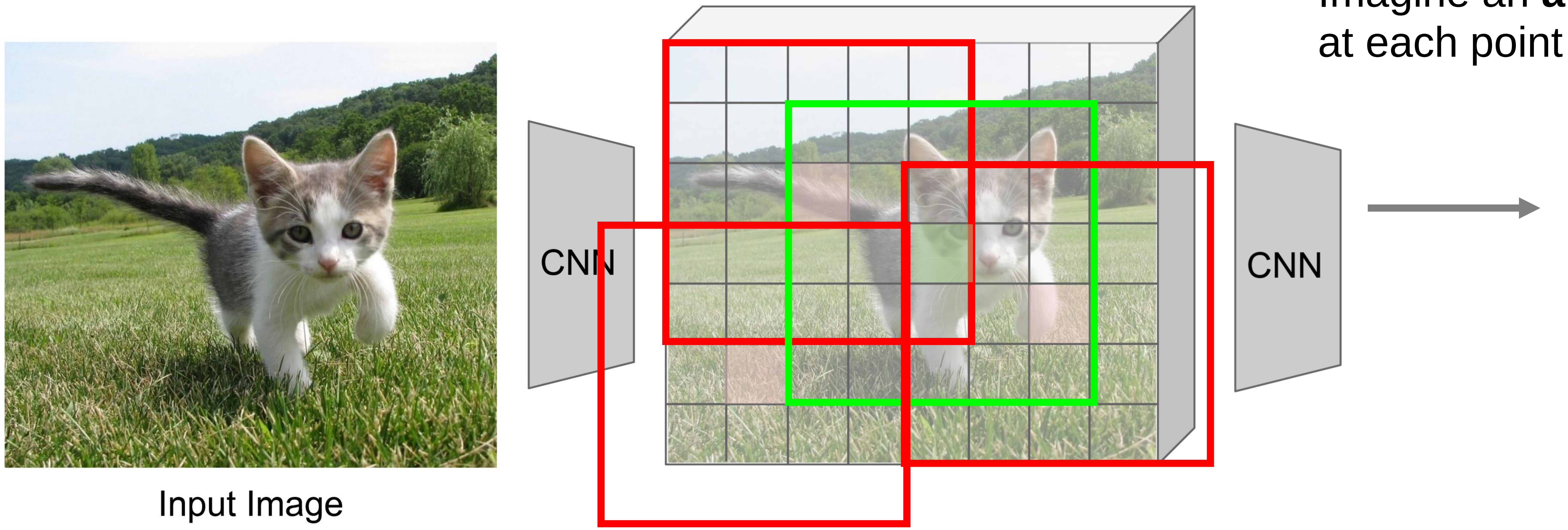
Image features
(e.g. 512 x 20 x 15)

Imagine an **anchor box** of fixed size at each point in the feature map

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Region Proposal Network



Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512 x 20 x 15)

Imagine an **anchor box** of fixed size at each point in the feature map

Anchor is an object?
1 x 20 x 15

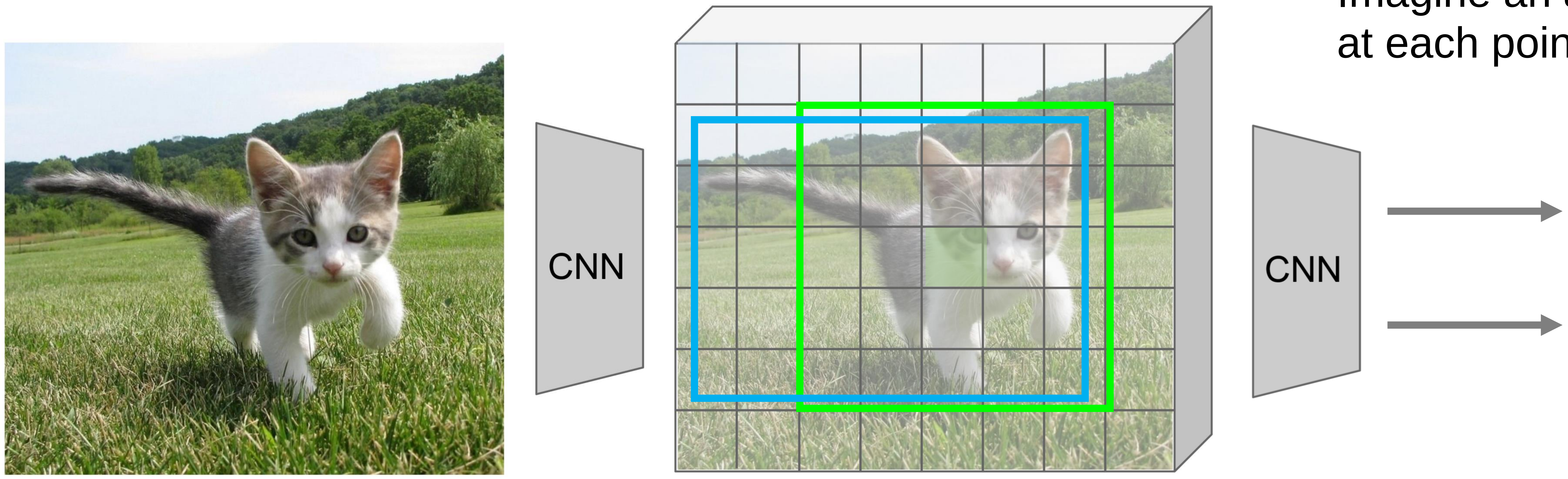
At each point, predict whether the corresponding anchor contains an object (binary classification)

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Region Proposal Network

Imagine an **anchor box** of fixed size at each point in the feature map



Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512 x 20 x 15)

Anchor is an object?
1 x 20 x 15

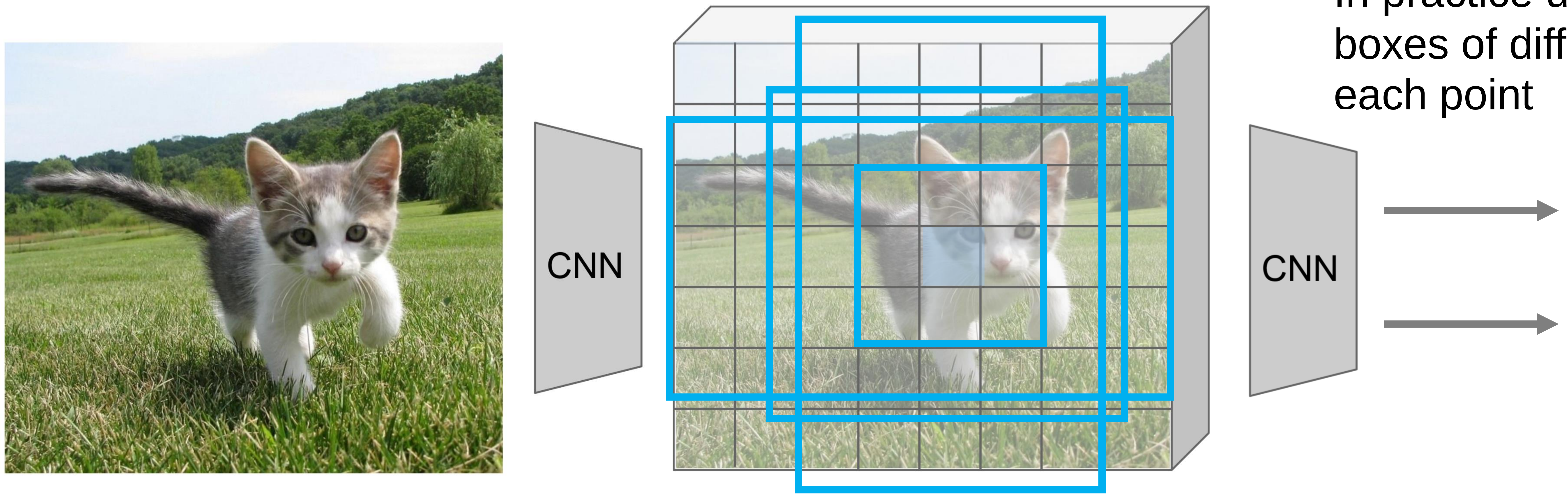
Box corrections
4 x 20 x 15

At each point, predict whether the corresponding anchor contains an object (binary classification)

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Region Proposal Network



Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512 x 20 x 15)

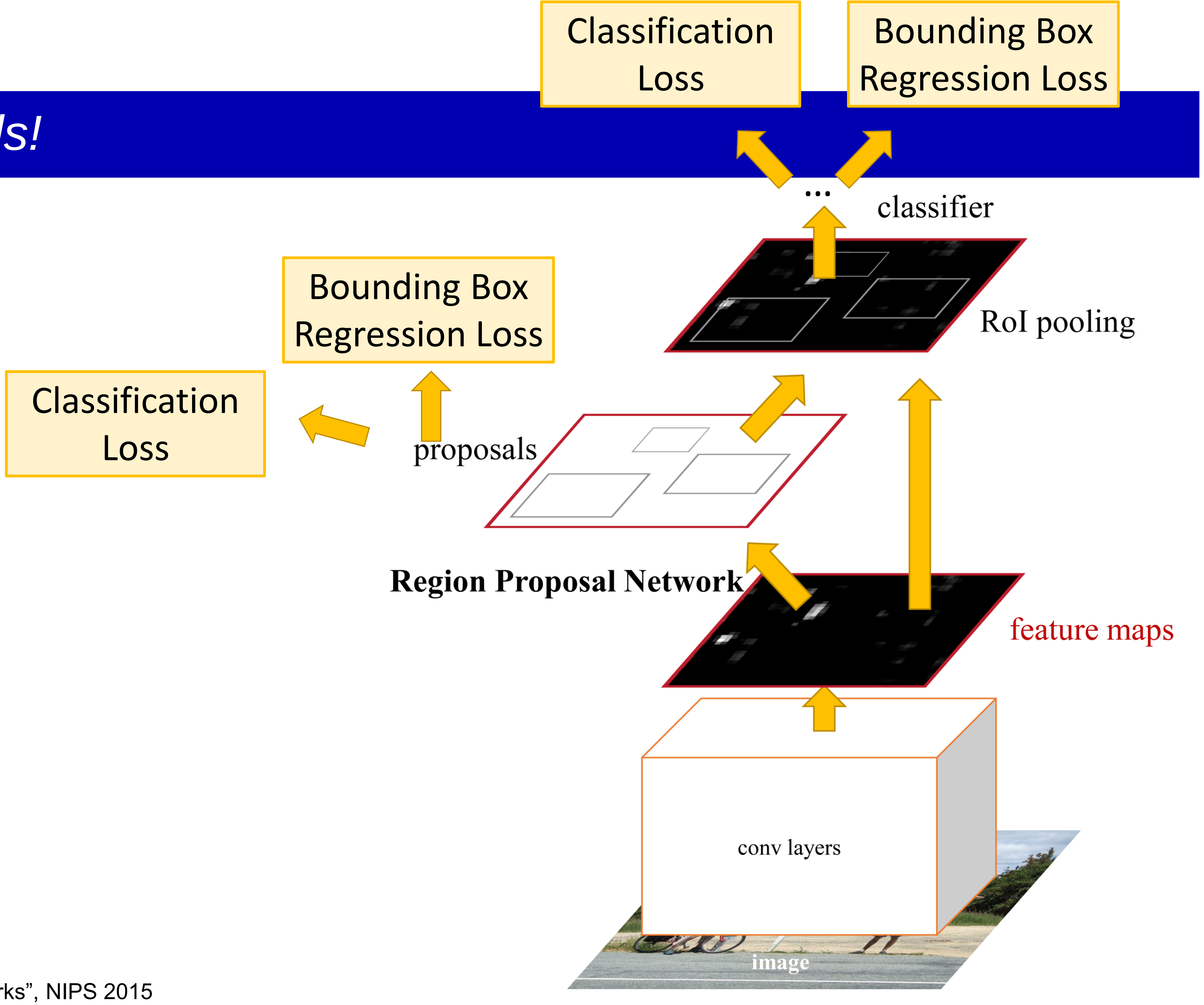
In practice use K different anchor boxes of different size / scale at each point

Anchor is an object?
1 x 20 x 15

Box corrections
4 x 20 x 15

Sort the K*20*15 boxes by their “objectness” score, take top ~300 as our proposals

Faster R-CNN: *Make CNN do proposals!*



Jointly train with 4 losses:

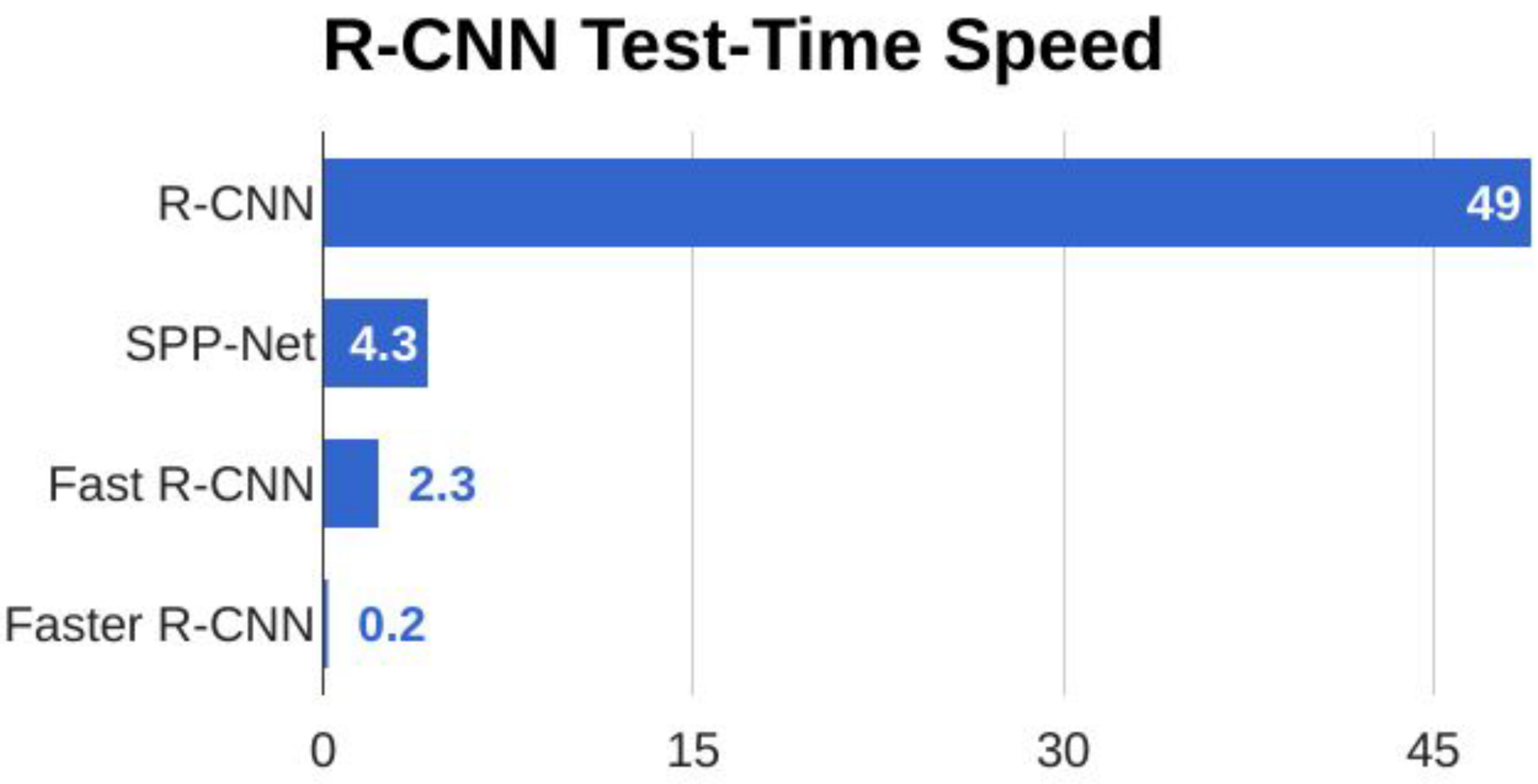
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Faster R-CNN: *Make CNN do proposals!*



Faster R-CNN: *Advantages*

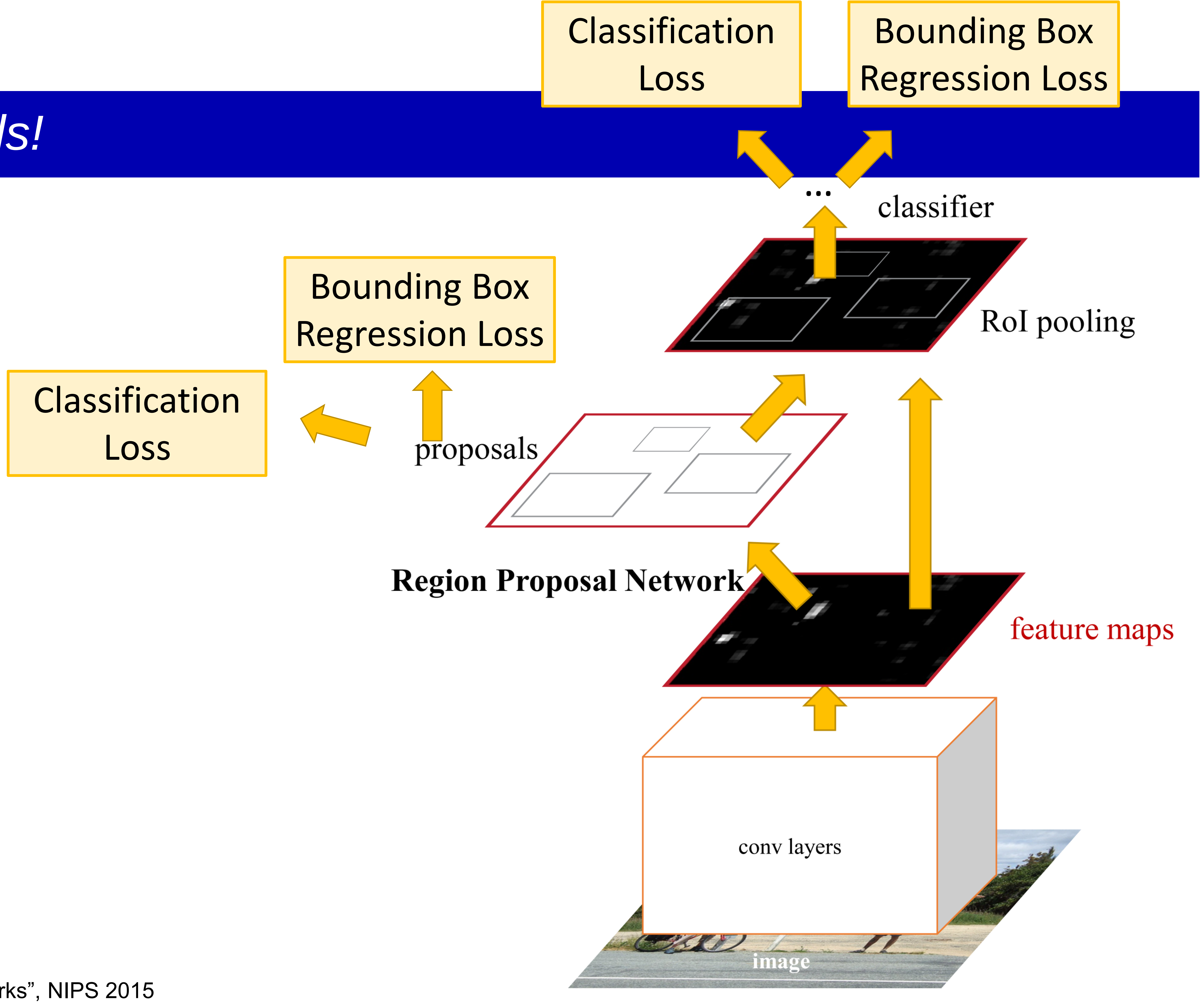
Faster Training and Inference: Faster R-CNN is faster than Fast R-CNN because it eliminates the need for the computationally expensive selective search algorithm used in Fast R-CNN. The RPN generates region proposals directly from the feature map, resulting in faster training and inference times.

Improved Localization Accuracy: Faster R-CNN is more accurate than Fast R-CNN because the RPN generates more accurate region proposals than the selective search algorithm used in Fast R-CNN.

Flexibility: The RPN in Faster R-CNN can be trained end-to-end with the rest of the network, enabling the system to learn more discriminative features for region proposal generation.

Adaptability: The RPN can be modified to handle various input sizes and aspect ratios, making it adaptable to a wide range of object detection tasks.

Faster R-CNN: *Make CNN do proposals!*



Glossing over many details:

- Ignore overlapping proposals with non-max suppression

- How are anchors determined?

- How do we sample positive / negative samples for training the RPN?

- How to parameterize bounding box regression?

Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Faster R-CNN: *Disadvantages*

Complexity: Faster R-CNN is a complex framework that requires a large number of layers and parameters, making it difficult to train and optimize. This can also result in high computational and memory requirements.

Anchors: The RPN in Faster R-CNN relies on anchor boxes to generate region proposals, which can be a limiting factor for certain types of objects and scenes. Choosing the right set of anchor sizes and aspect ratios is also challenging.

Object Scale: Faster R-CNN is designed to handle objects at a wide range of scales, but it may still struggle with objects that are too small or too large. The RPN may generate too many or too few region proposals for such objects, affecting detection accuracy.

Training Data: Like other deep learning models, Faster R-CNN requires large amounts of training data to achieve good performance. Collecting and annotating such data can be time-consuming and expensive.

Performance Tradeoffs: Like all object detection frameworks, Faster R-CNN involves a tradeoff between detection accuracy and inference speed. Higher accuracy usually requires more computation, while faster inference can result in lower accuracy.

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

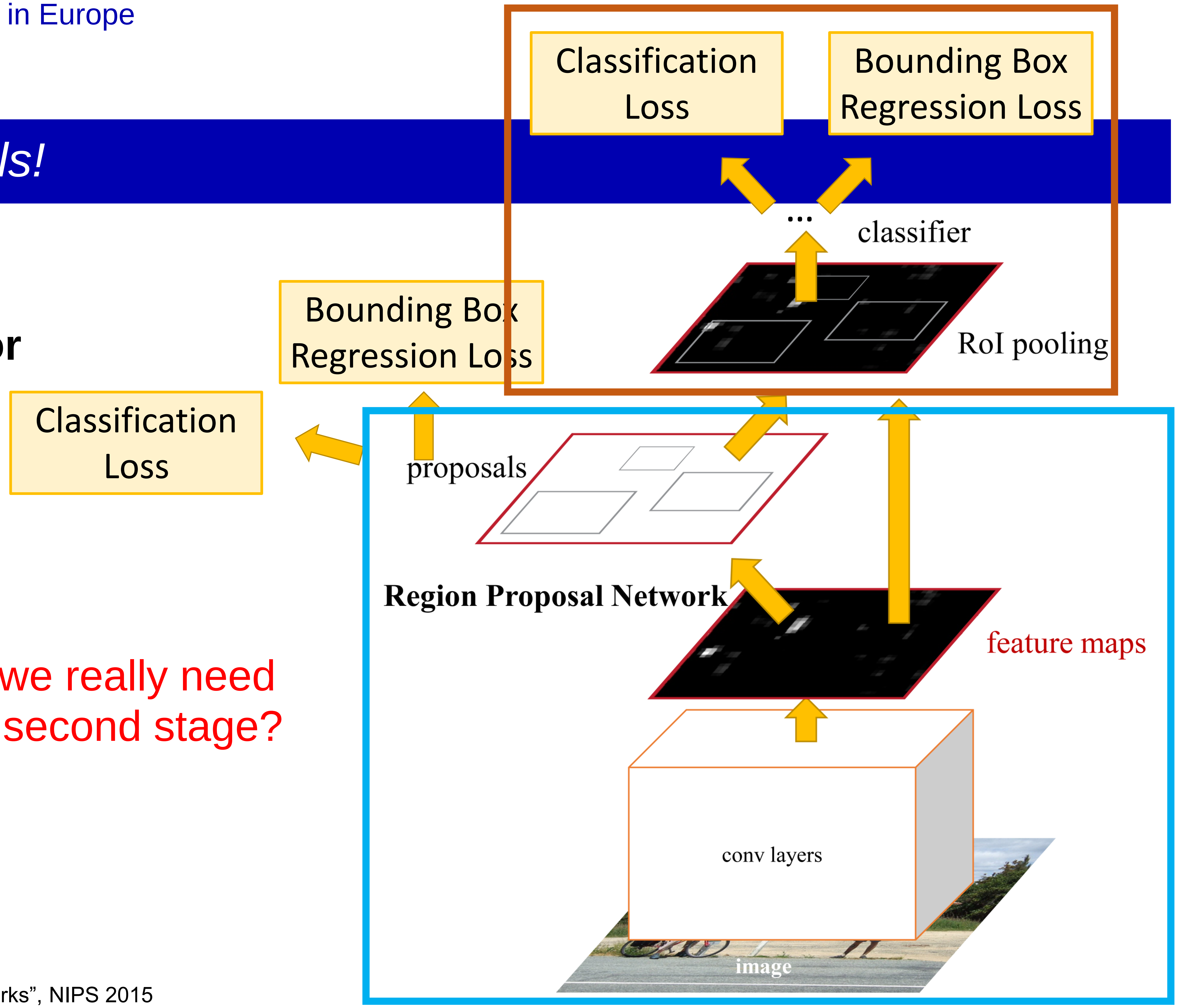
Faster R-CNN: *Make CNN do proposals!*

Faster R-CNN is a **Two-stage object detector**

First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset

Do we really need the second stage?



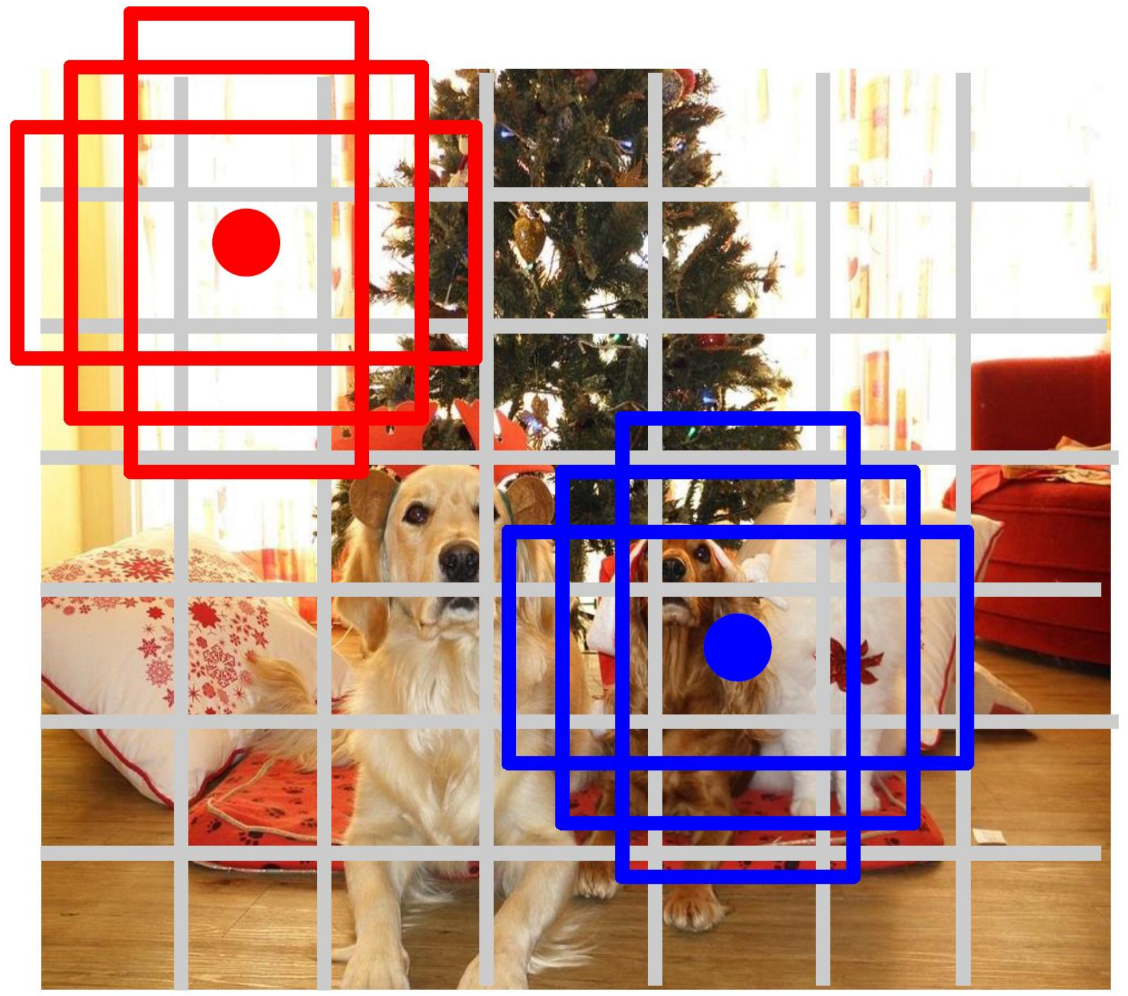
Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Single-Stage Object Detectors: *YOLO / SSD / RetinaNet*

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output: **7 x 7 x (5 * B + C)**

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Single-Stage Object Detectors: *YOLO / SSD / RetinaNet*

YOLO (You Only Look Once) is a real-time object detection system that uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation, rather than using a sliding window approach or region proposal method.

The YOLO algorithm takes an input image and divides it into a grid of cells. For each cell, YOLO predicts the bounding boxes of objects that may be present in that cell, along with their associated class probabilities. The predictions are made using a single neural network that takes the entire image as input and outputs the predicted bounding boxes and class probabilities for all cells in a single pass.

YOLO is known for its speed and real-time performance, making it popular in applications such as self-driving cars, surveillance, and robotics. However, the tradeoff for its speed is lower accuracy compared to some other object detection algorithms.

Redmon et al, “You Only Look Once: Unified, Real-Time Object Detection”, CVPR 2016

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Single-Stage Object Detectors: *YOLO / SSD / RetinaNet*

SSD (Single-Shot MultiBox Detector) is a popular object detection algorithm that builds upon the previous work of the MultiBox object detection framework, but with improvements in terms of accuracy and speed.

Like YOLO, SSD is a real-time object detection algorithm that uses a single neural network to predict the bounding boxes and class probabilities for objects in an image. However, SSD differs from YOLO in that it uses multiple layers with different resolutions to detect objects of various sizes.

SSD generates a set of default bounding boxes of different aspect ratios and scales for each location in the feature maps, and then predicts the offset and class probabilities for these bounding boxes. By doing this, SSD can detect objects of various sizes and aspect ratios with high accuracy.

In general, SSD is known for its high accuracy and efficiency, making it a popular choice for real-time object detection applications such as robotics, self-driving cars, and surveillance systems.

Liu et al, “SSD: Single-Shot MultiBox Detector”, ECCV 2016

Single-Stage Object Detectors: *YOLO / SSD / RetinaNet*

RetinaNet solves the problem of object detection in the presence of a large number of background regions, where most object detectors struggle due to the class imbalance problem.

RetinaNet uses a feature pyramid network (FPN) architecture to generate a set of feature maps with different resolutions. It then applies a novel focal loss function that down-weights the loss assigned to well-classified examples, making the training more robust to the class imbalance problem.

RetinaNet also uses a novel "one-stage" detection approach that combines the benefits of both "one-stage" and "two-stage" detection methods. Specifically, RetinaNet predicts object classification and bounding box regression in a single stage, similar to one-stage detectors like YOLO and SSD. However, it also uses a feature pyramid network and multiple levels of feature maps, similar to two-stage detectors like Faster R-CNN, to improve detection accuracy.

Lin et al, “Focal Loss for Dense Object Detection”, ICCV 2017

Object Detection: *Takeaways*

Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

Bigger / Deeper backbones work better

Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017
Zou et al, “Object Detection in 20 Years: A Survey”, arXiv 2019

R-FCN: Dai et al, “R-FCN: Object Detection via Region-based Fully Convolutional Networks”, NIPS 2016
Inception-V2: Ioffe and Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015
Inception V3: Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”, arXiv 2016
Inception ResNet: Szegedy et al, “Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning”, arXiv 2016
MobileNet: Howard et al, “Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv 2017

Instance Segmentation

Classification | **Semantic Segmentation** | **Object Detection** | **Instance Segmentation**

CAT

GRASS, CAT, TREE, SKY

DOG, DOG, CAT

DOG, DOG, CAT

No spatial extent

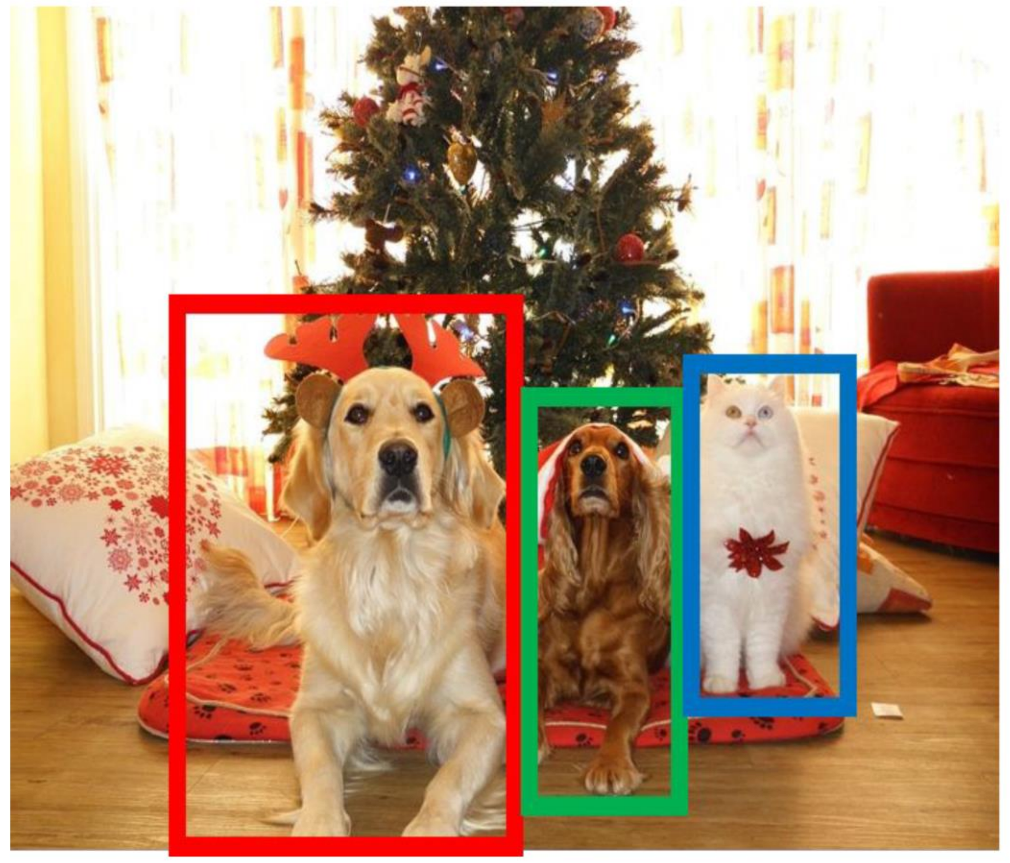
No objects, just pixels

Multiple Object

This image is CC0 public domain

Object Detection: *Faster R-CNN*

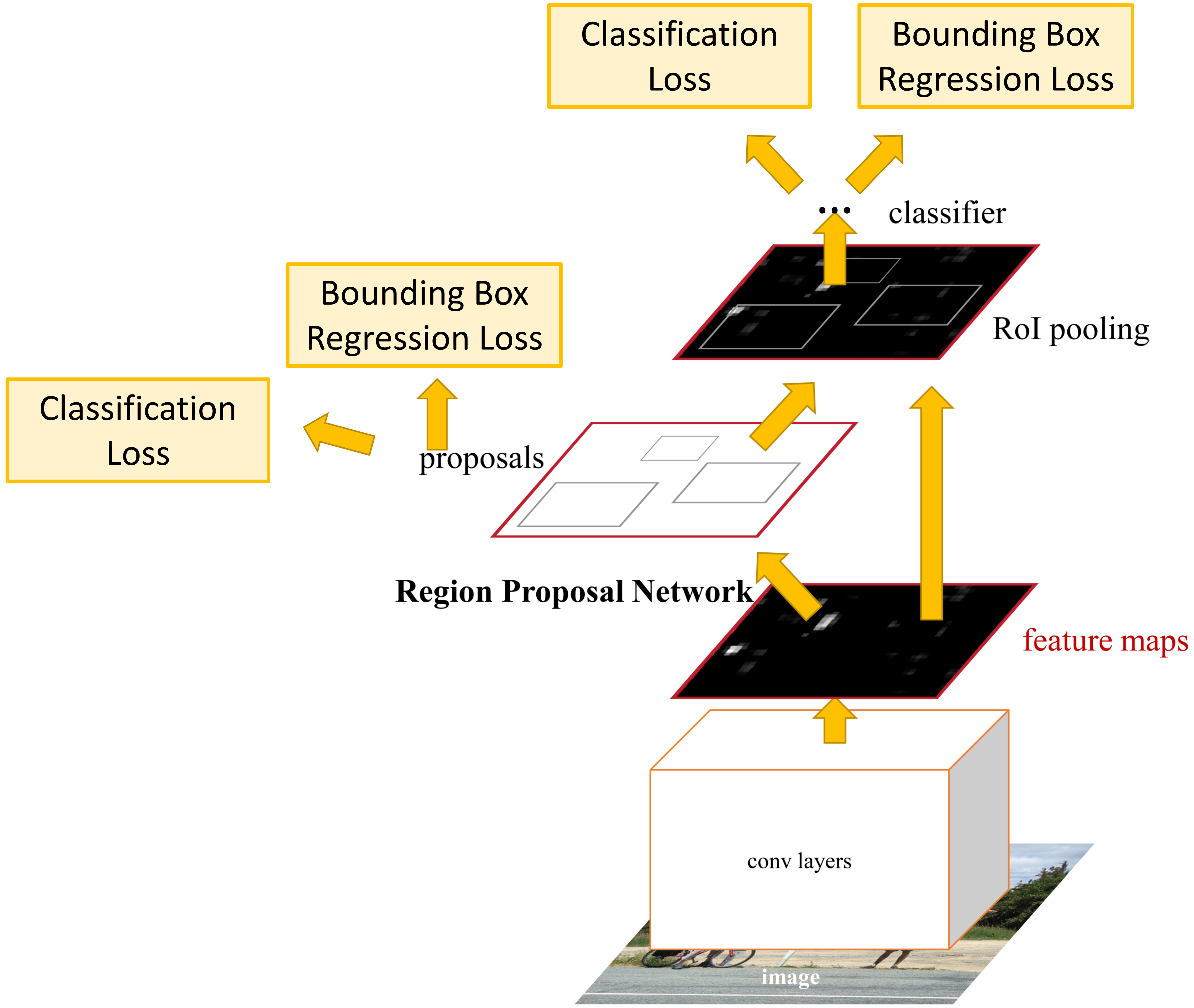
Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT



Classification Loss

Bounding Box Regression Loss

classifier

RoI pooling

Bounding Box Regression Loss

Classification Loss

proposals

Region Proposal Network

feature maps

conv layers

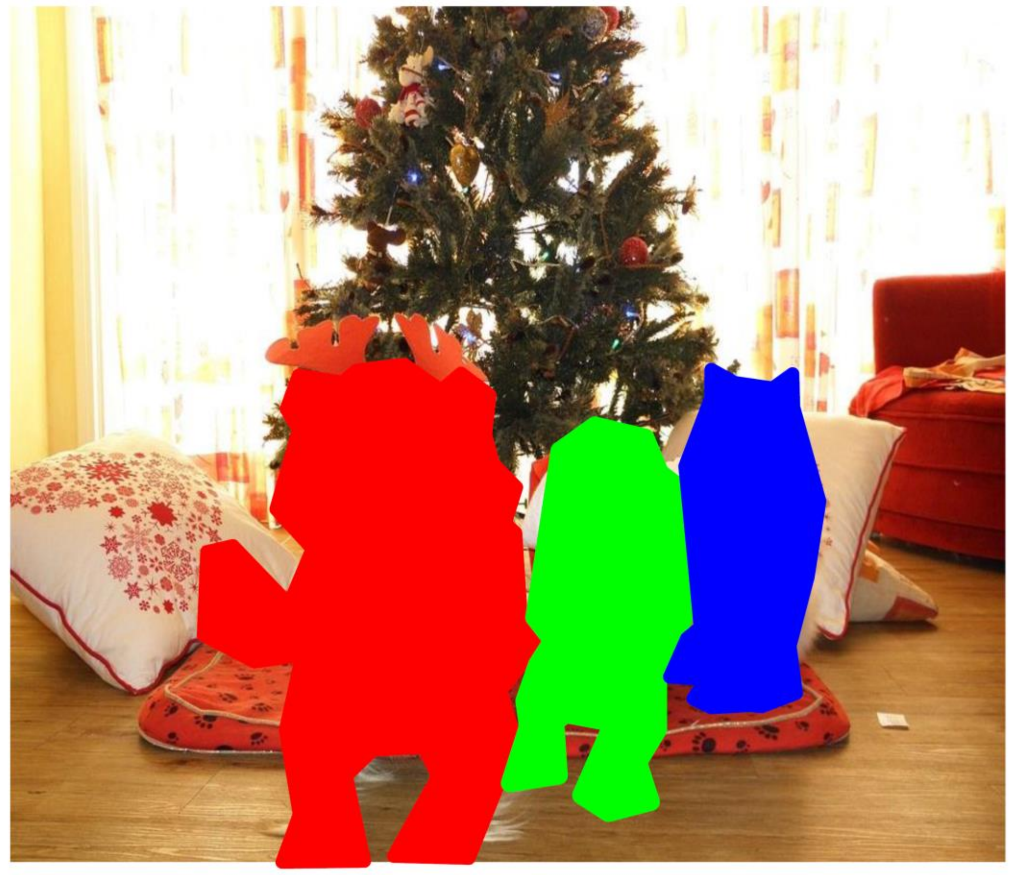
image

Instance Segmentation: *Mask R-CNN*

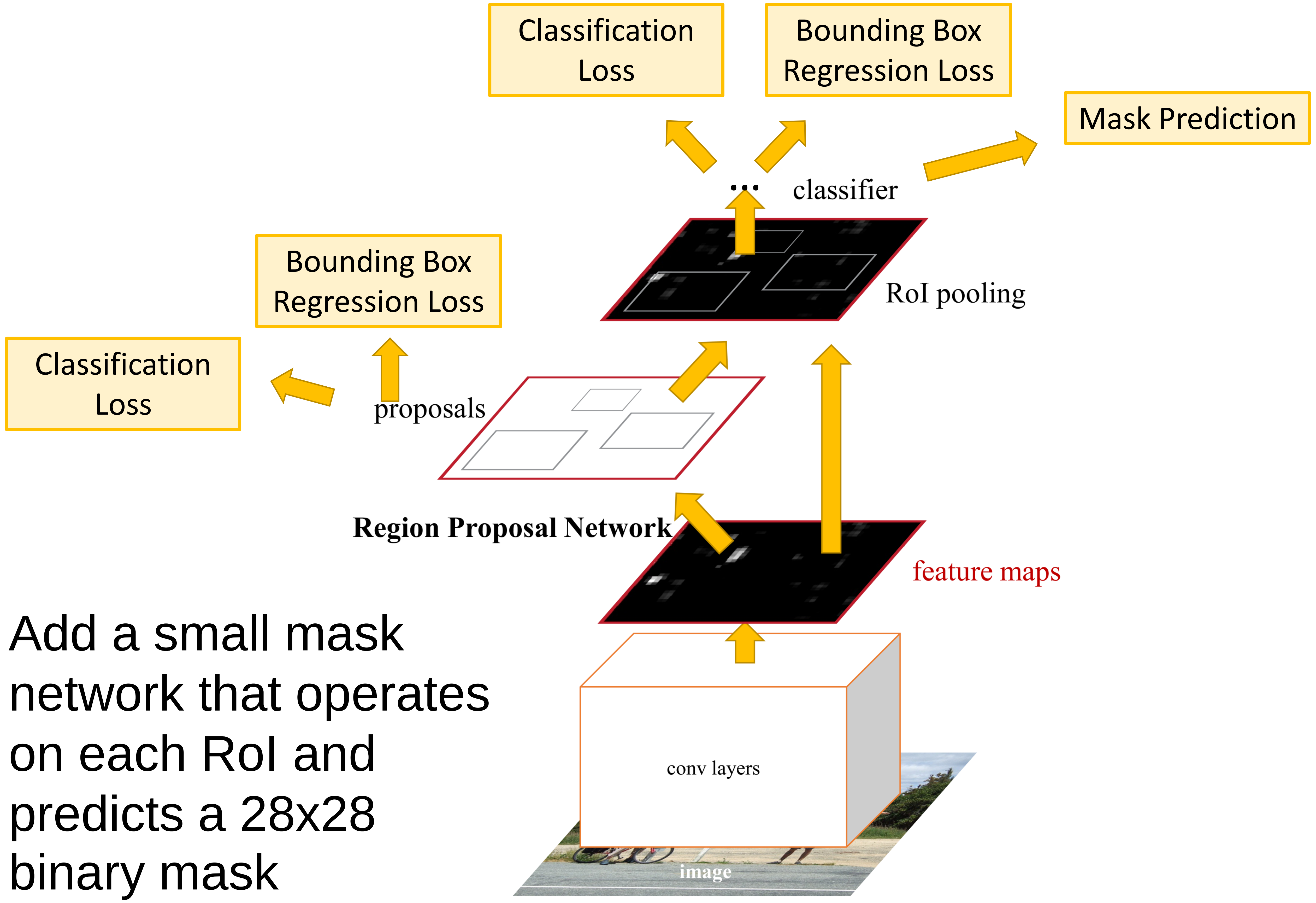
Object Detection

DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT



Add a small mask network that operates on each RoI and predicts a 28x28 binary mask

He et al, “Mask R-CNN”, arXiv 2017

Instance Segmentation: *Mask R-CNN*

Mask R-CNN is a popular object detection and segmentation algorithm that builds upon the earlier Region-based Convolutional Neural Network (R-CNN) and Faster R-CNN object detection algorithms, but with the added capability of instance segmentation.

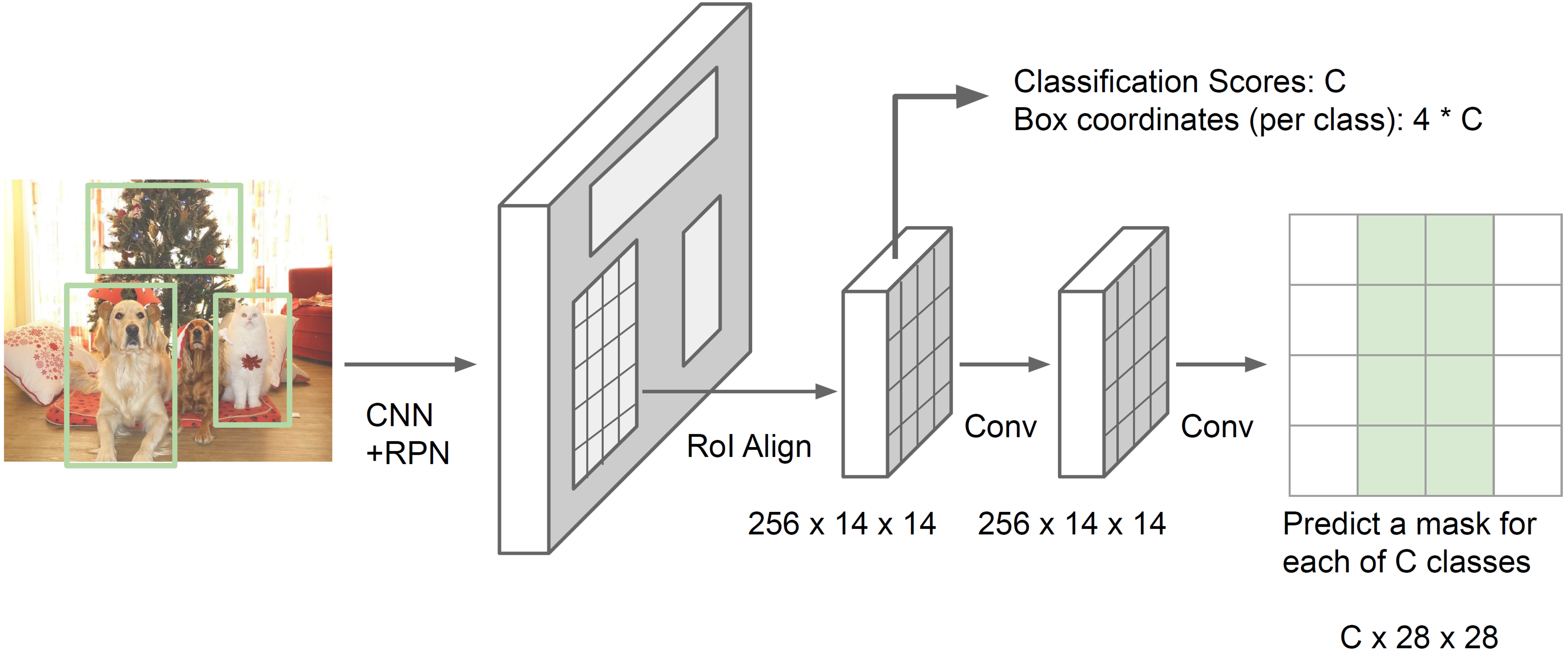
Like its predecessors, Mask R-CNN uses a region proposal network (RPN) to generate candidate object bounding boxes, and a classifier to predict the class and refine the bounding boxes. However, Mask R-CNN also adds a mask branch to the network that predicts binary masks for each object instance, in addition to the bounding box and class labels.

The mask branch takes a cropped feature map of the proposed region and applies a small fully convolutional network to generate a binary mask for the object instance. This allows Mask R-CNN to perform both object detection and instance segmentation in a single pass of the neural network.

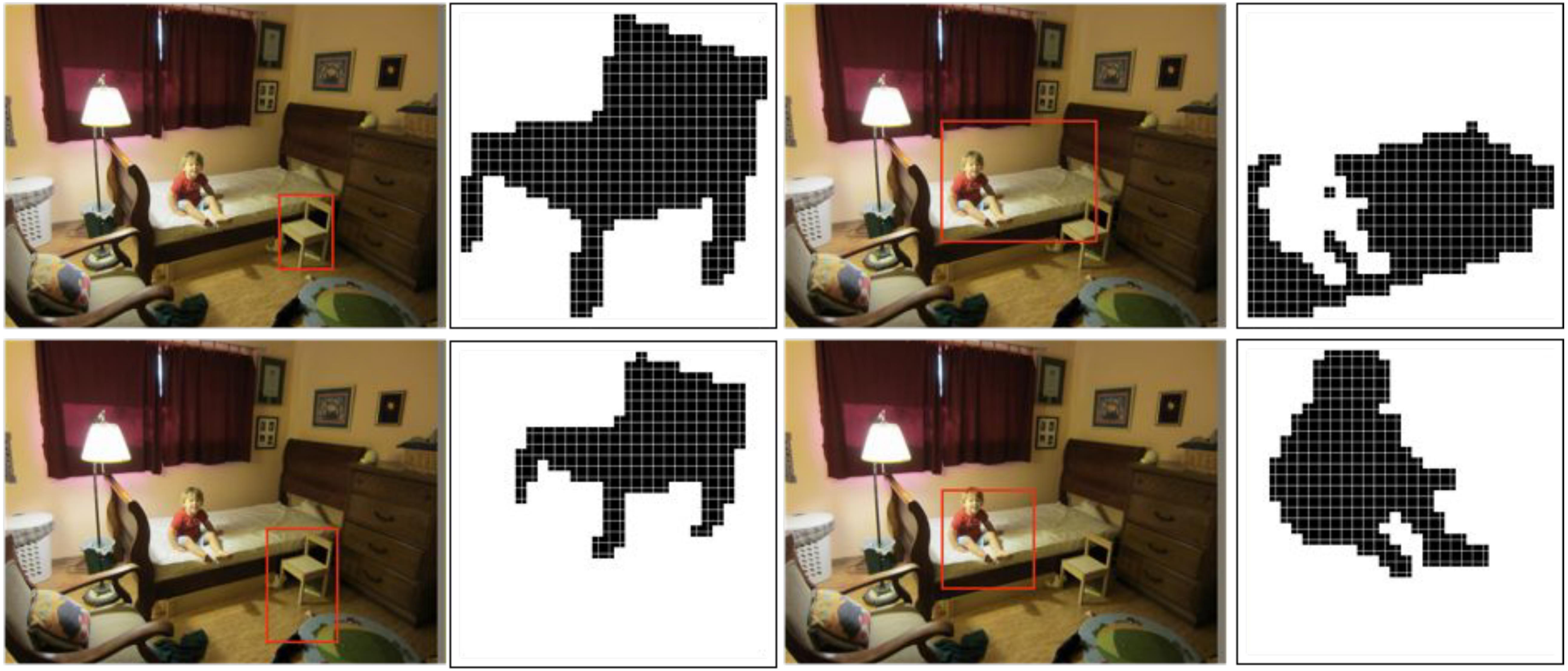
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

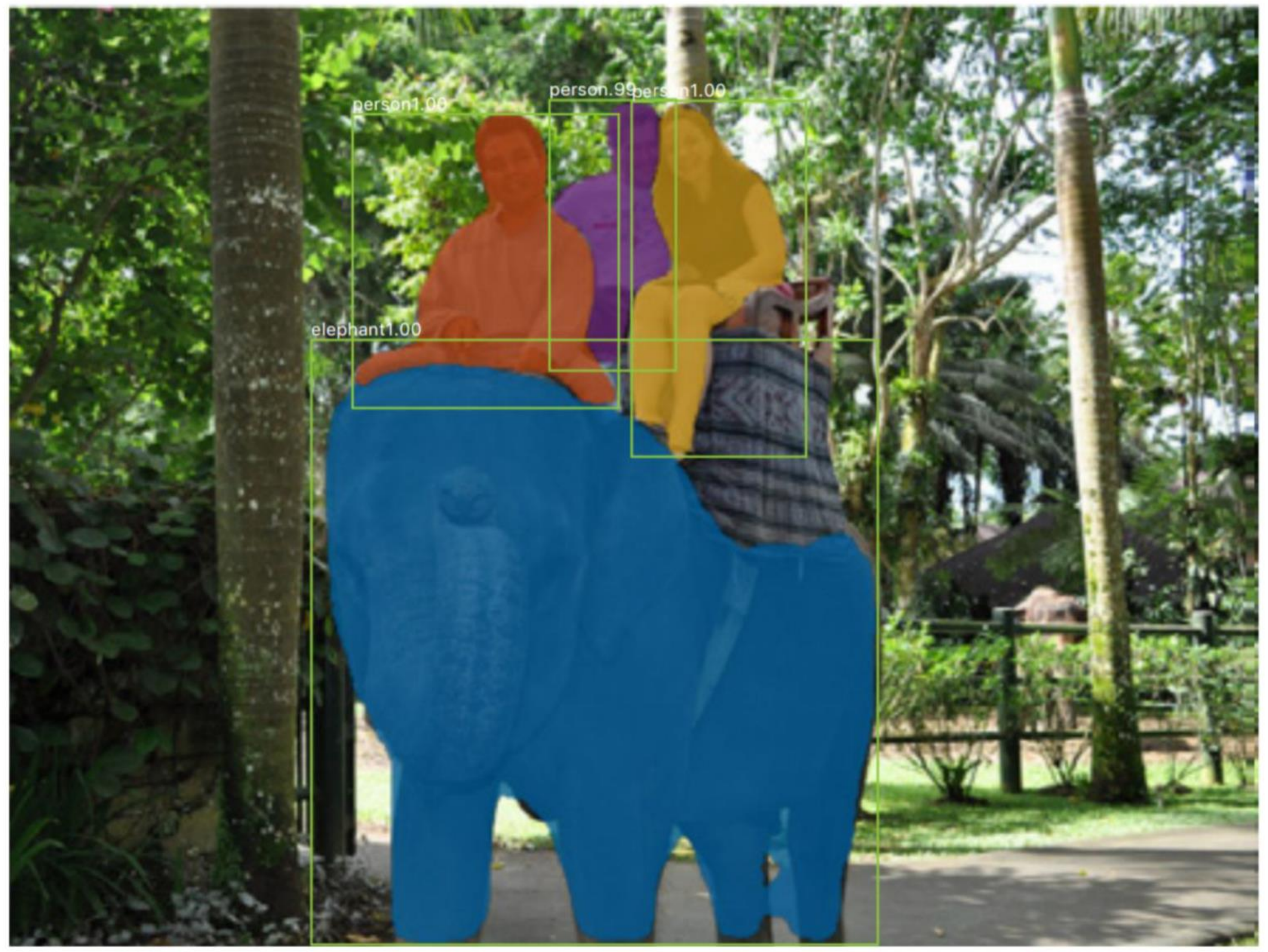
Instance Segmentation: *Mask R-CNN*

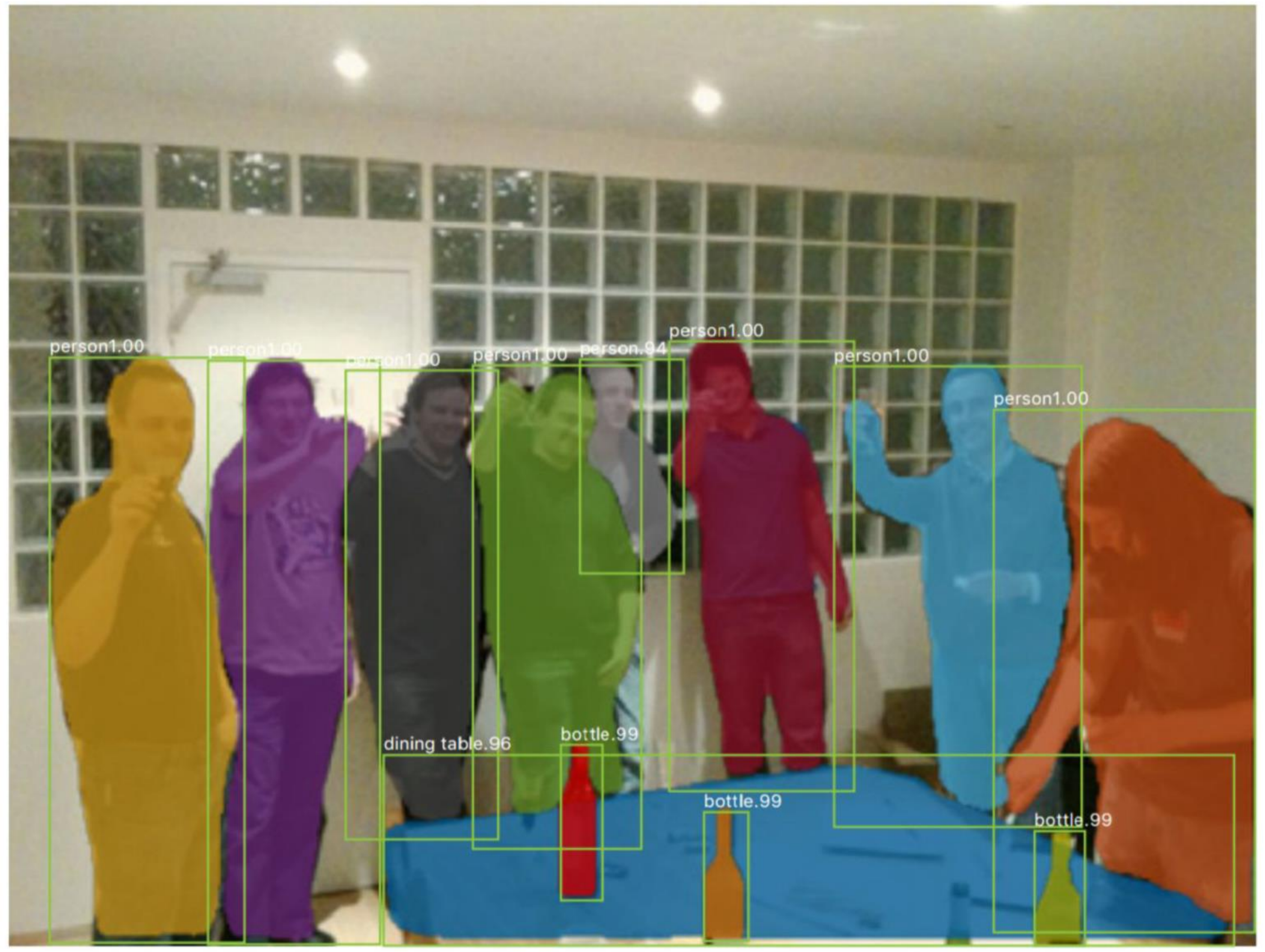


Instance Segmentation: *Mask R-CNN*



Instance Segmentation: *Mask R-CNN - Very Good Results!*





Instance Segmentation: *Mask R-CNN – Also does pose!*

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Open Source Frameworks

Lots of good implementations on GitHub!

TensorFlow Detection API:

https://github.com/tensorflow/models/tree/master/research/object_detection

Faster RCNN, SSD, RFCN, Mask R-CNN, ...

Detectron2 (PyTorch)

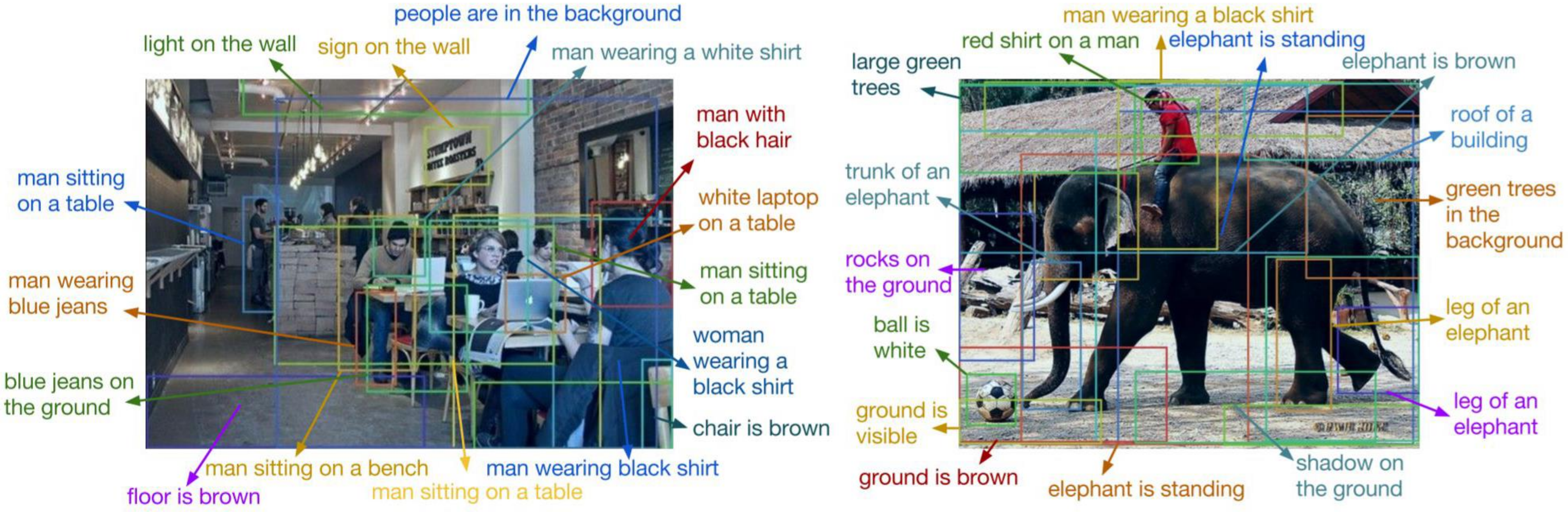
https://github.com/facebookresearch/detectron2

Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN, ...

Finetune on your own dataset with pre-trained models

Beyond 2D Object Detection...

Object Detection + Captioning = Dense Captioning



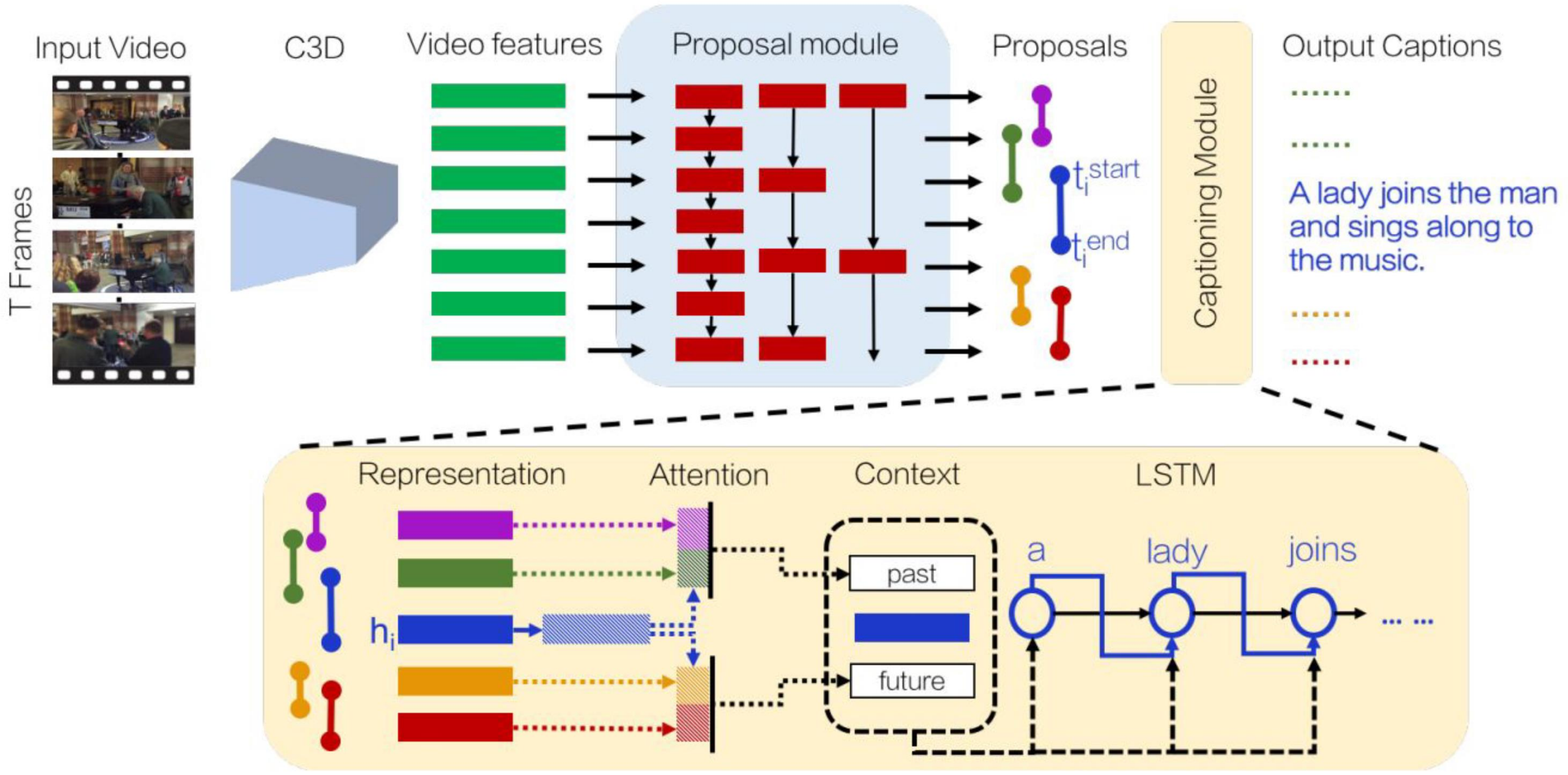
Johnson, Karpathy, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016

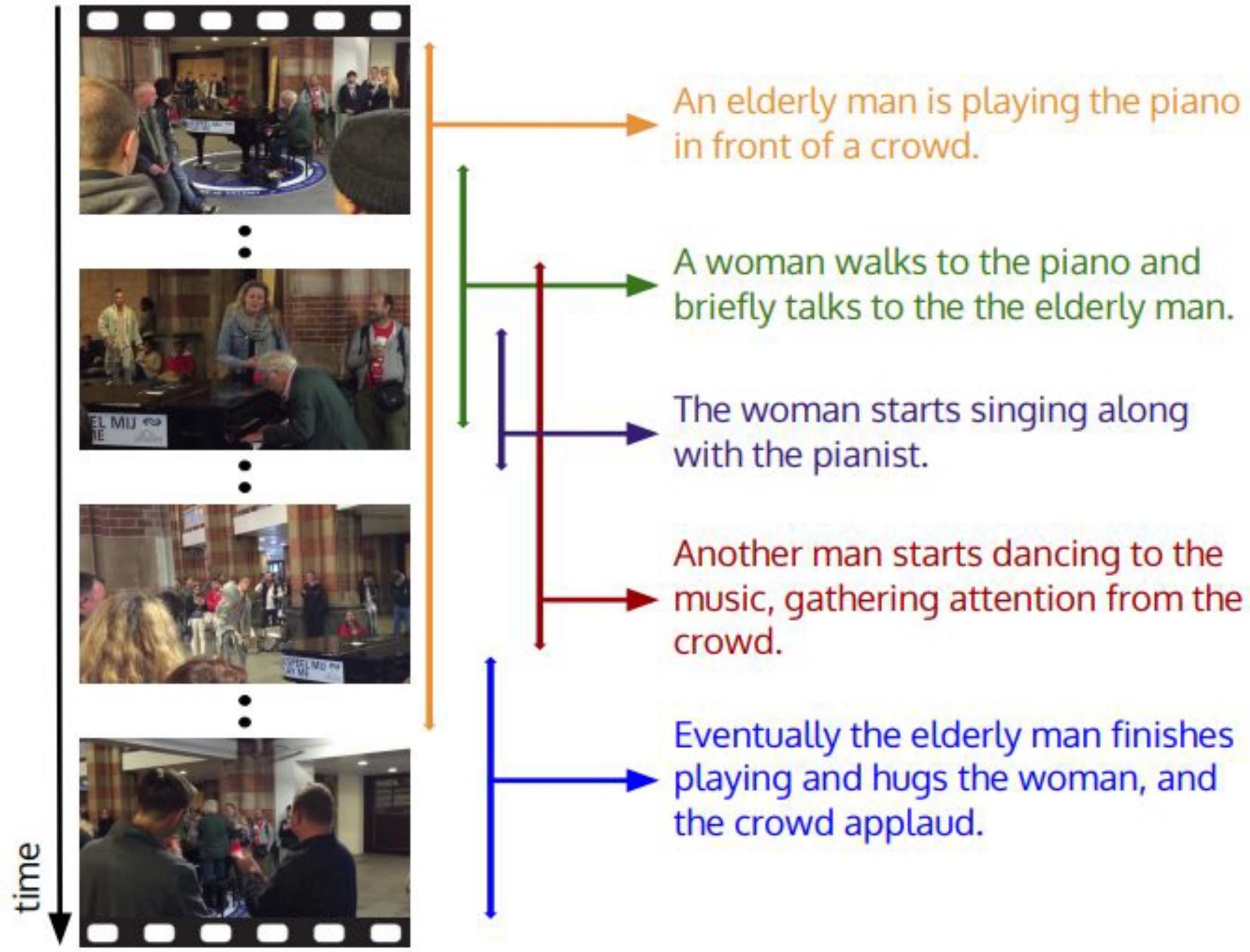
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Beyond 2D Object Detection...

Dense Video Captioning





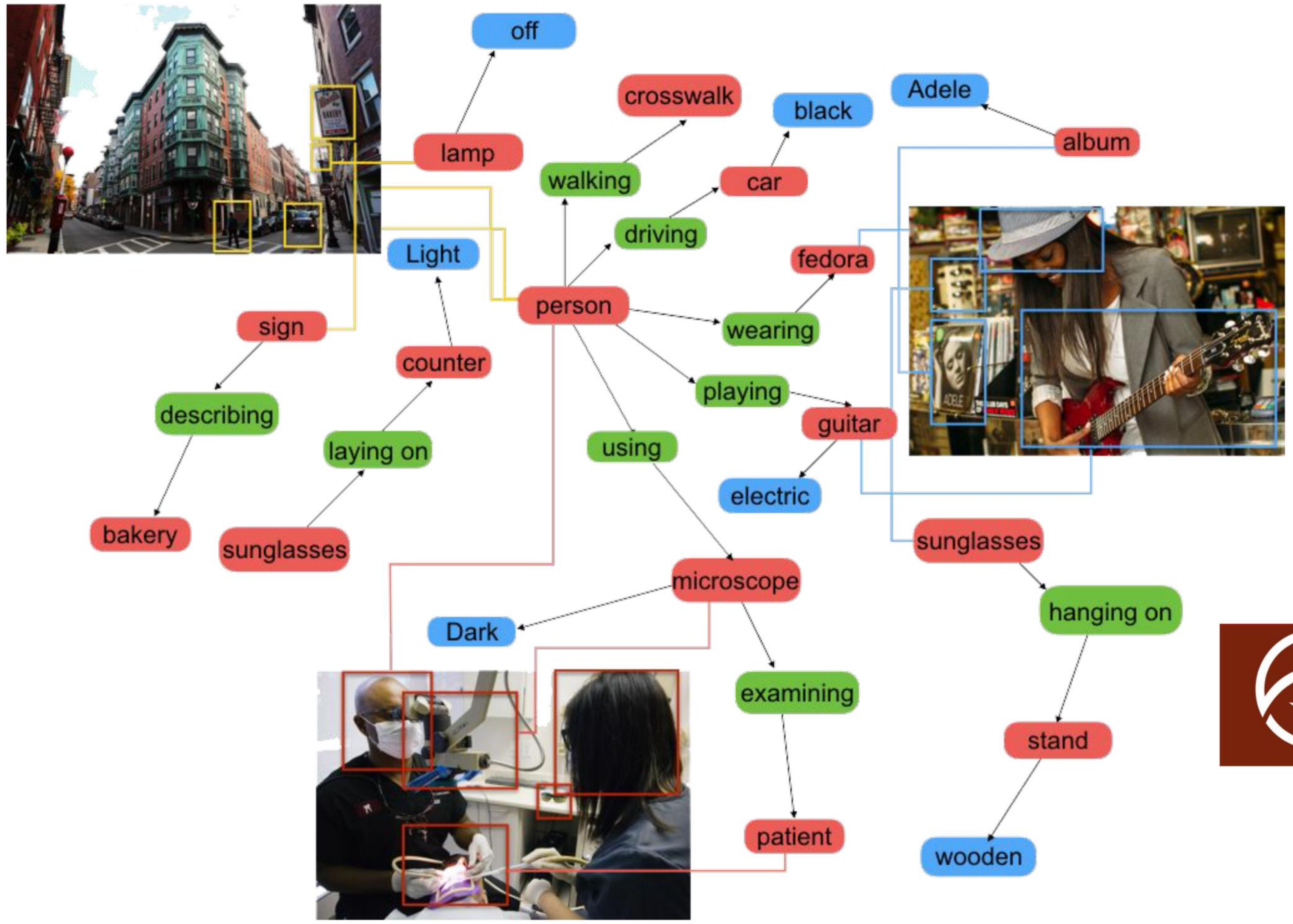
Ranjay Krishna et al., “Dense-Captioning Events in Videos”, ICCV 2017

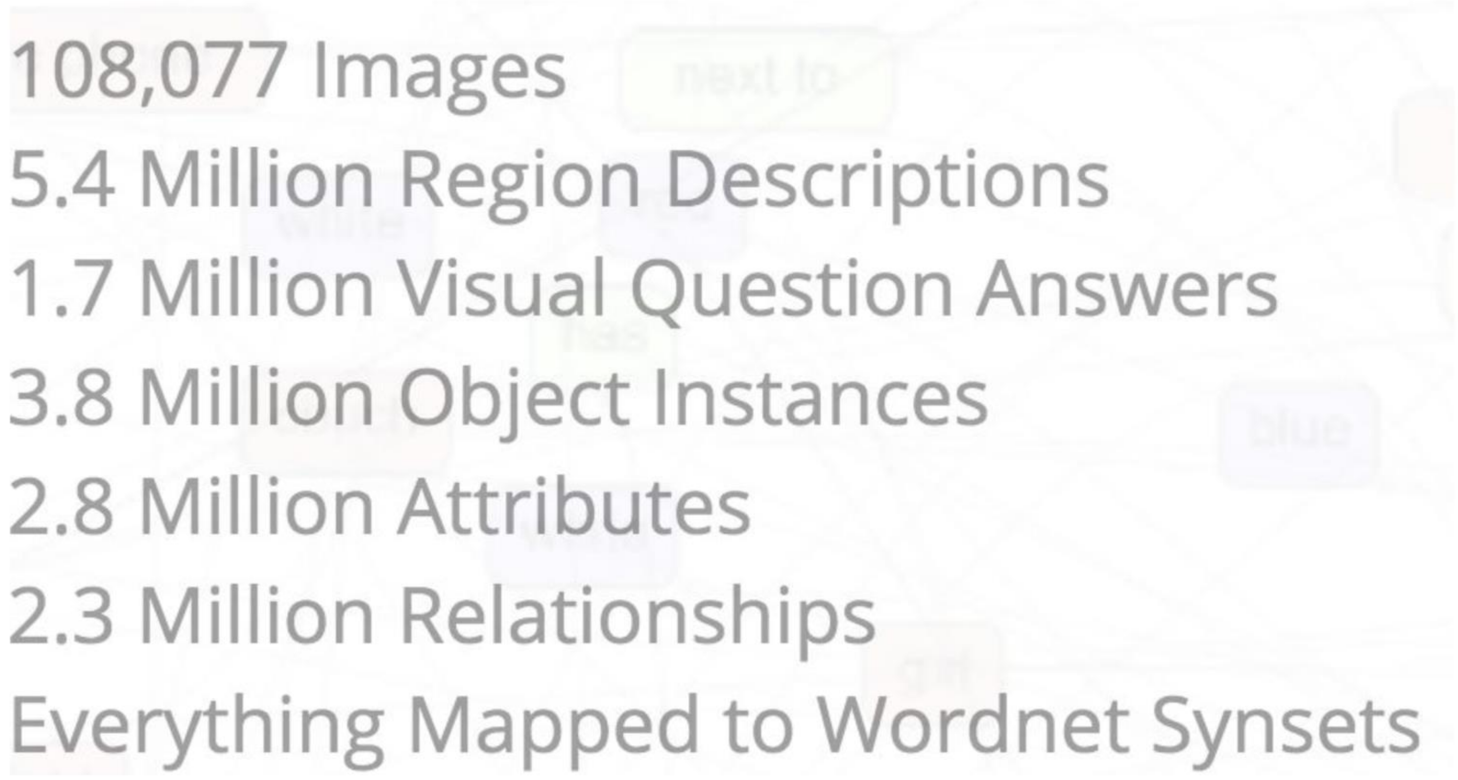
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Beyond 2D Object Detection...

Objects + Relationships = Scene Graphs



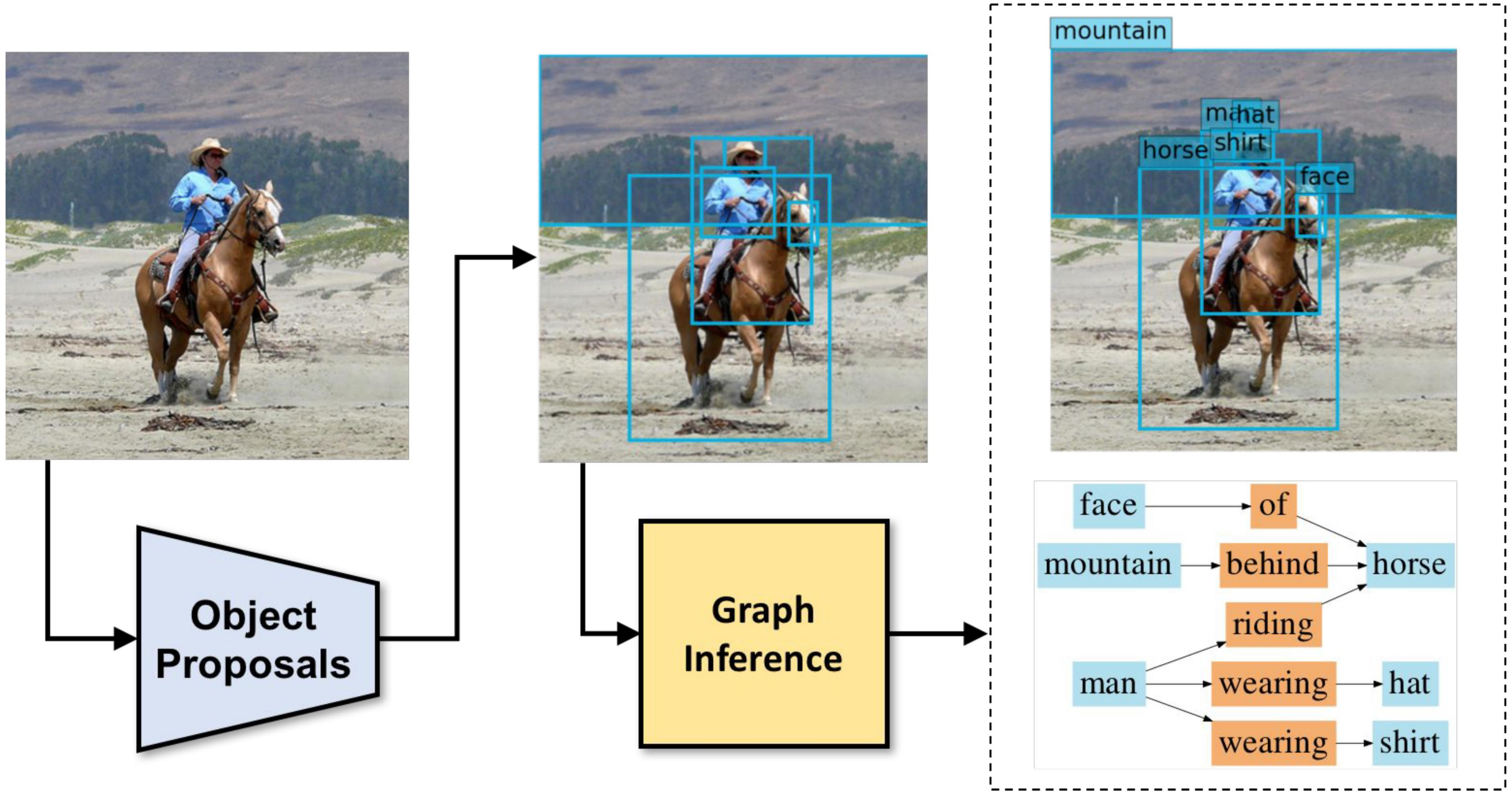

108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

VISUALGENOME

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

Beyond 2D Object Detection...

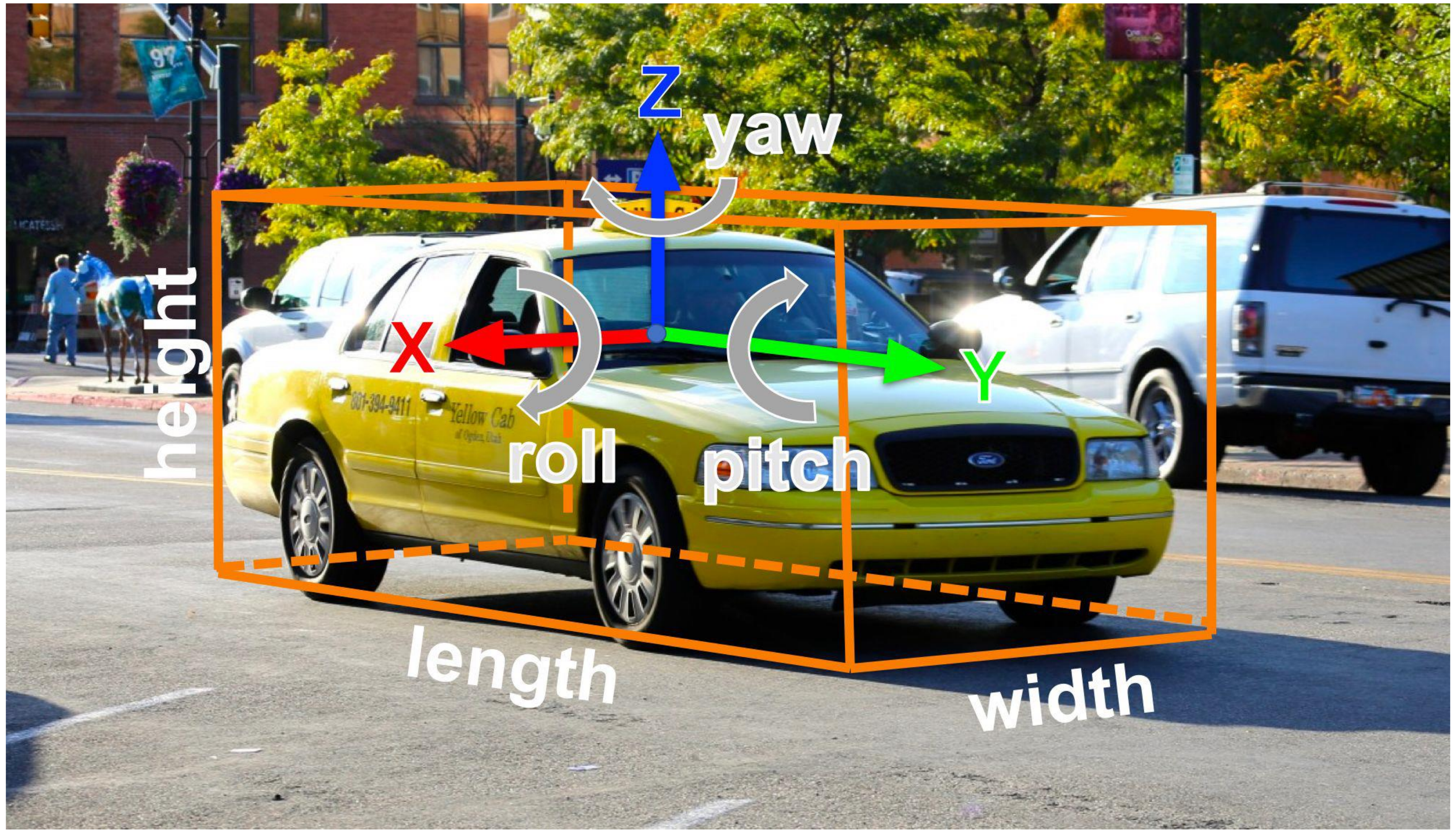
Scene Graph Prediction



Xu, Zhu, Choy, and Fei-Fei, “Scene Graph Generation by Iterative Message Passing”, CVPR 2017

Beyond 2D Object Detection...

3D Object Detection



2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

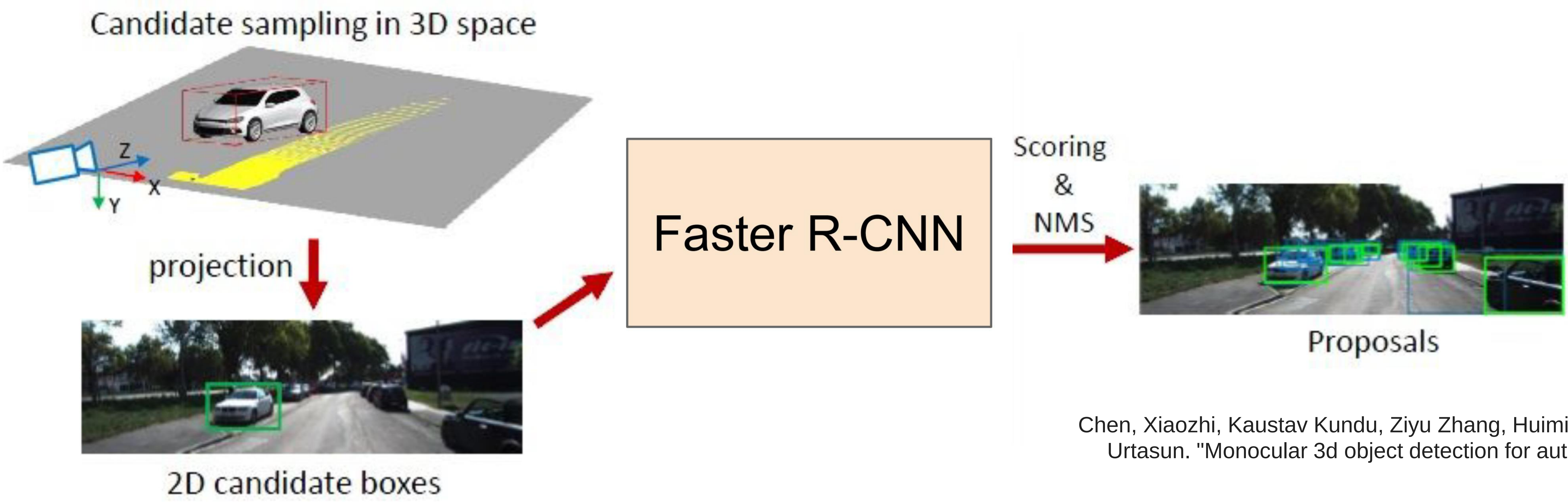
Much harder problem than 2D object detection!

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Beyond 2D Object Detection...

3D Object Detection: Monocular Camera



Chen, Xiaozhi, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. "Monocular 3d object detection for autonomous driving." CVPR 2016.

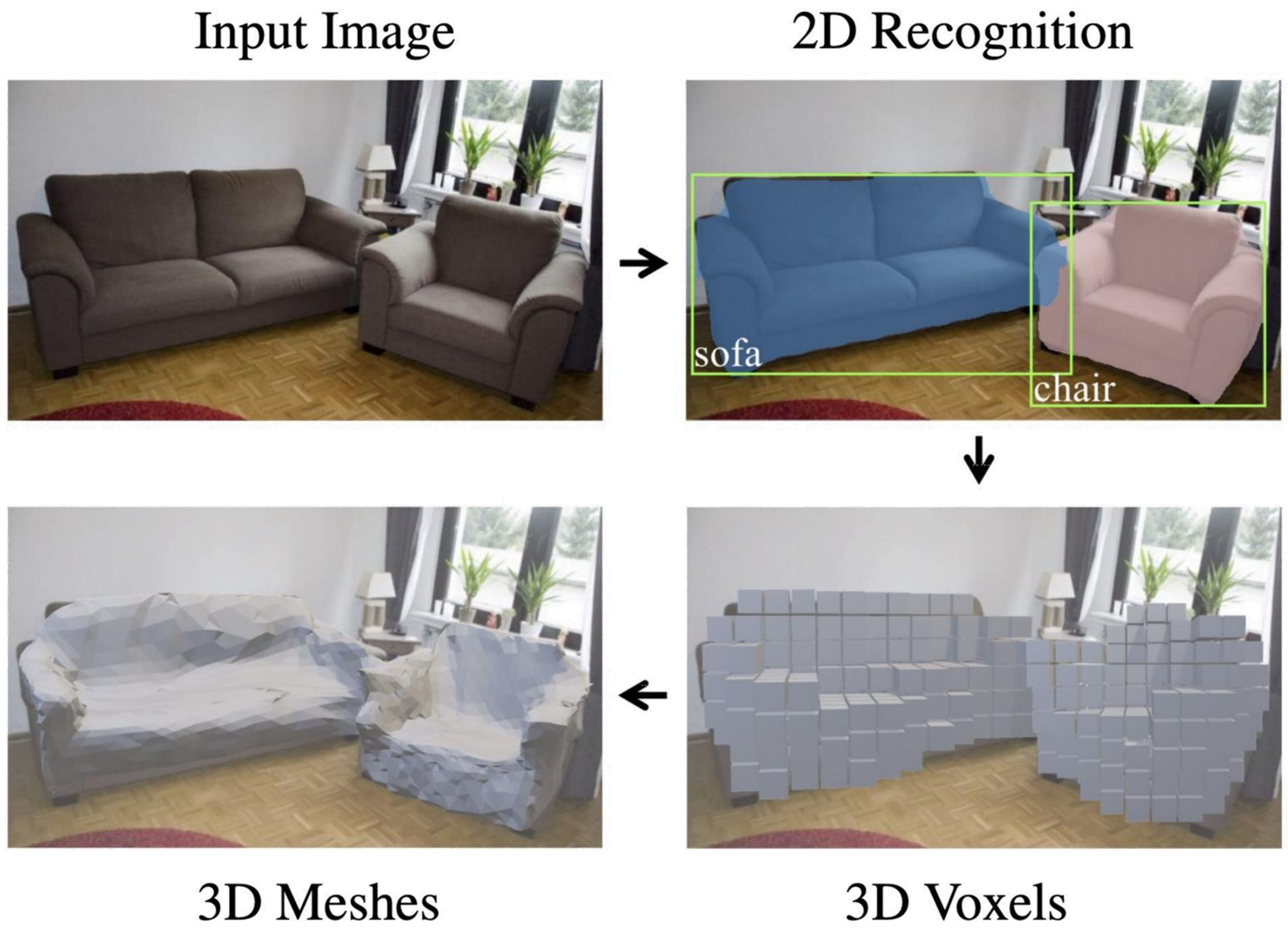
- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Beyond 2D Object Detection...

3D Shape Prediction: Mesh R-CNN



Gkioxari et al., Mesh RCNN, ICCV 2019

Co-financed by the European Union
Connecting Europe Facility

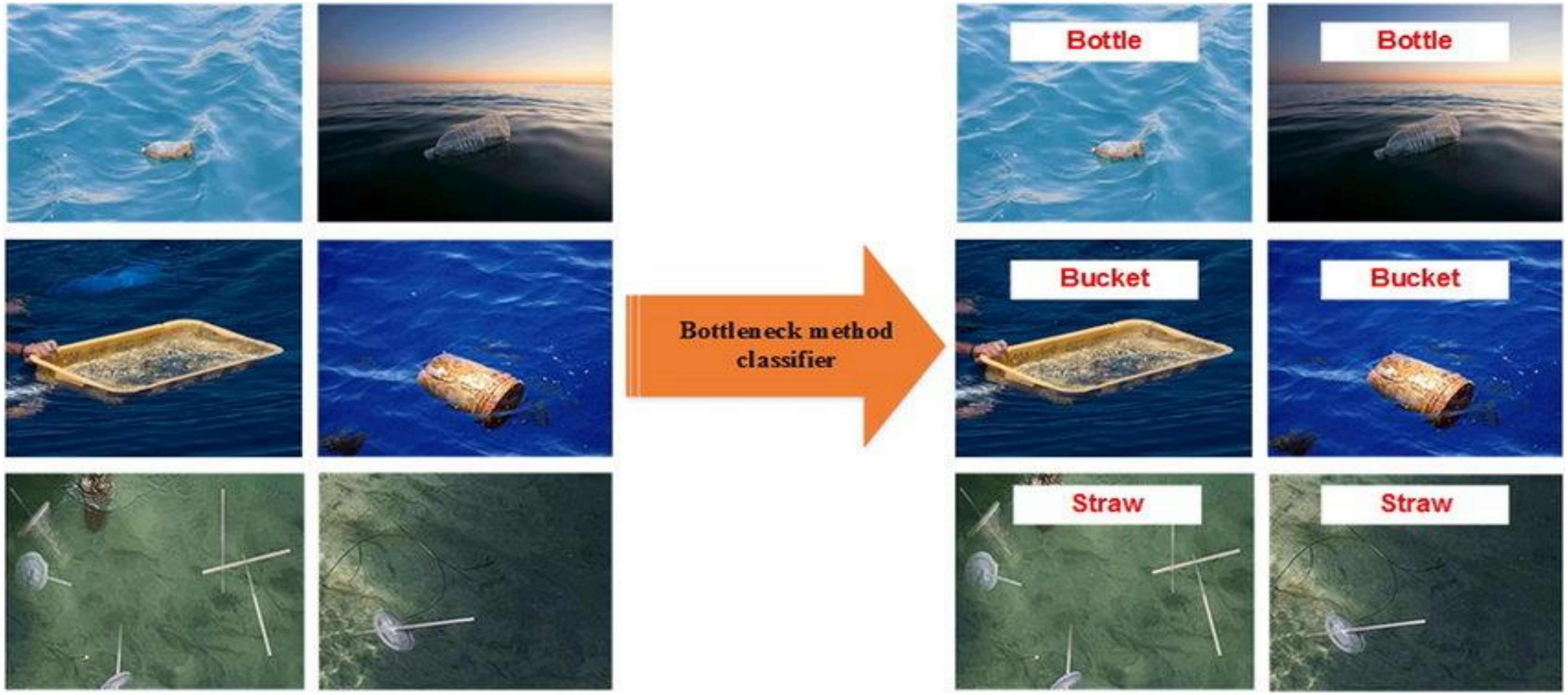
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Research Example from DeepCamera, CYENS Centre of Excellence

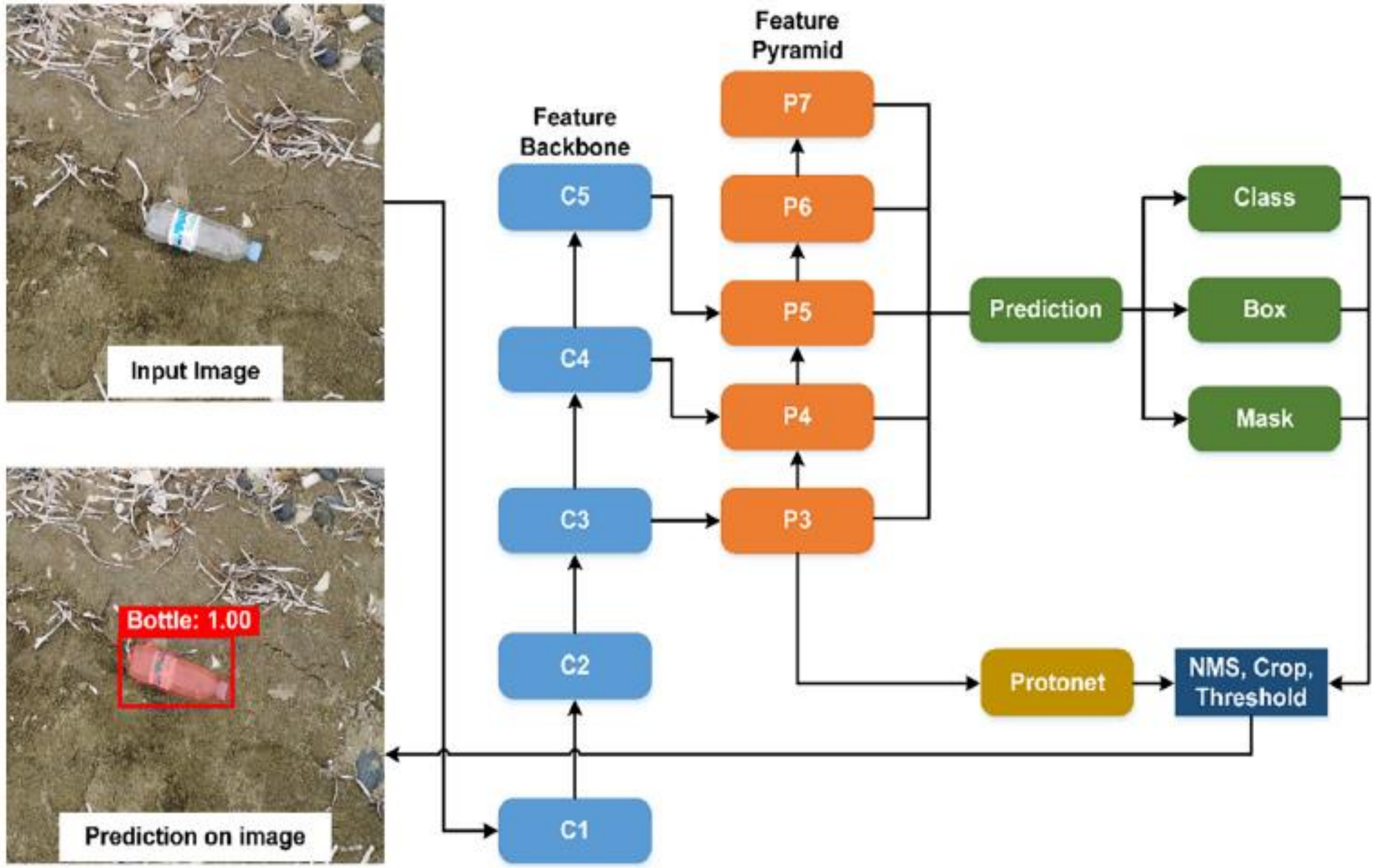
CYENS
CENTRE OF EXCELLENCE

Deep
Camera

Identifying floating plastic marine debris using a deep learning approach



A new paradigm for estimating the prevalence of plastic litter in the marine environment

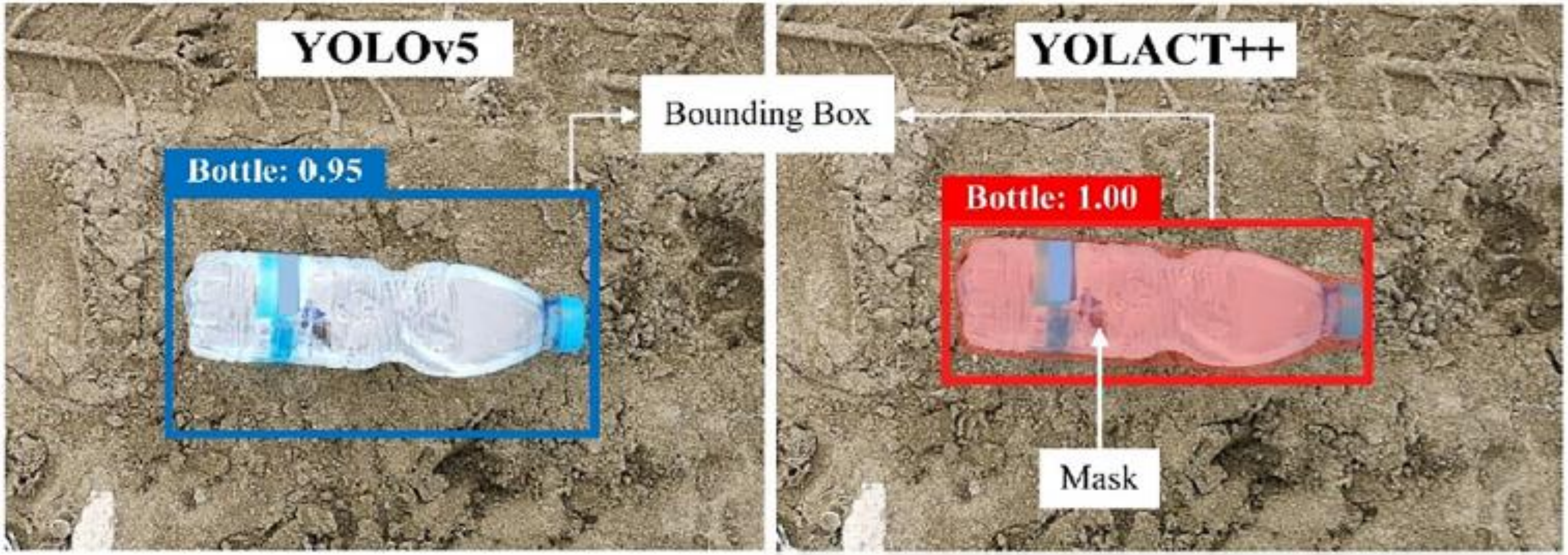


The YOLACT++ model architecture

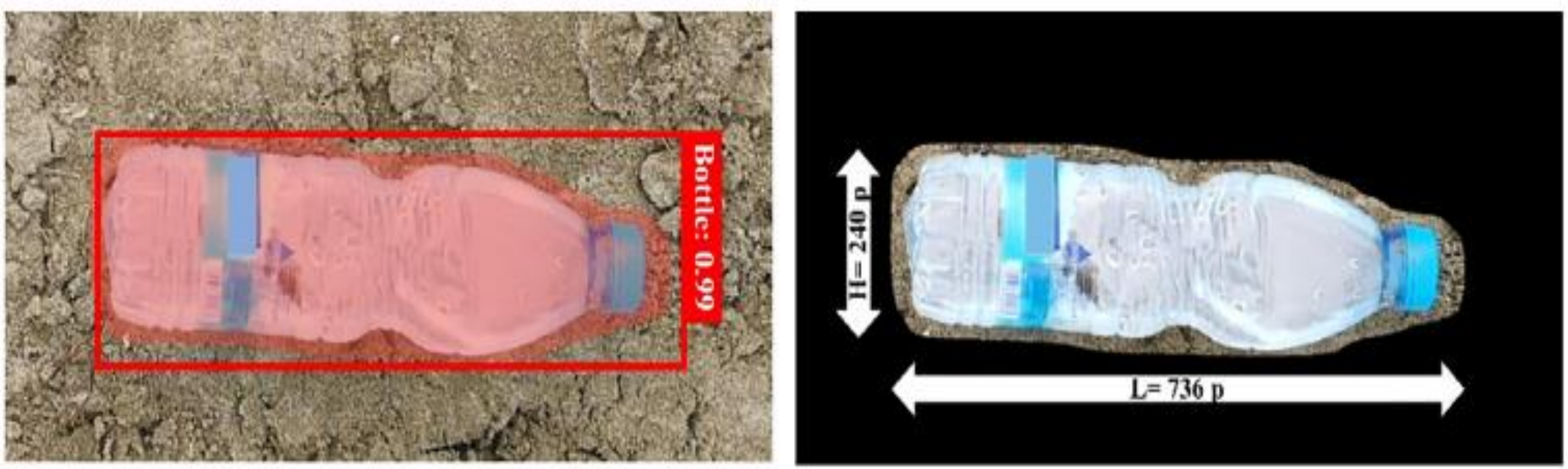
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Comparison of the performance of the YOLOv5 and the YOLACT++ tools



Determining the dimensions of plastic litter



Research in Deep Camera

Deep Camera

CYENS
CENTRE OF EXCELLENCE

Alessandro Artusi

Team Leader
DeepCamera Group

email: a.artusi@cyens.org.cy

Research Interests:

Machine Learning, Deep Learning and its applications in Computer Vision, High Dynamic Range Imaging, Image Processing applied on Computer Graphics and Color Science

https://www.cyens.org.cy/en-gb/research/pillars-groups/visual-sciences/deep-camera/people/alessandro-artusi/

https://deepcamera.cyens.org.cy/about-us/

deepcamera.ai@gmail.com

+357 227 475 81

Dimarchias Square 23 STOA, Nicosia
Nicosia, Nicosia 1016, Cyprus

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Thank you!

See you next week

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423