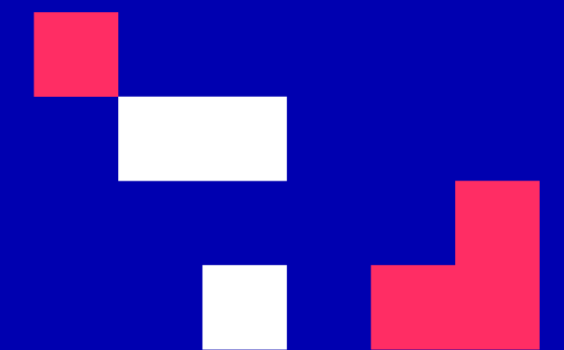
MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI645 - Machine Learning for Graphics and Computer Vision

Andreas Aristidou, PhD

Spring Semester 2025



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

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

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.

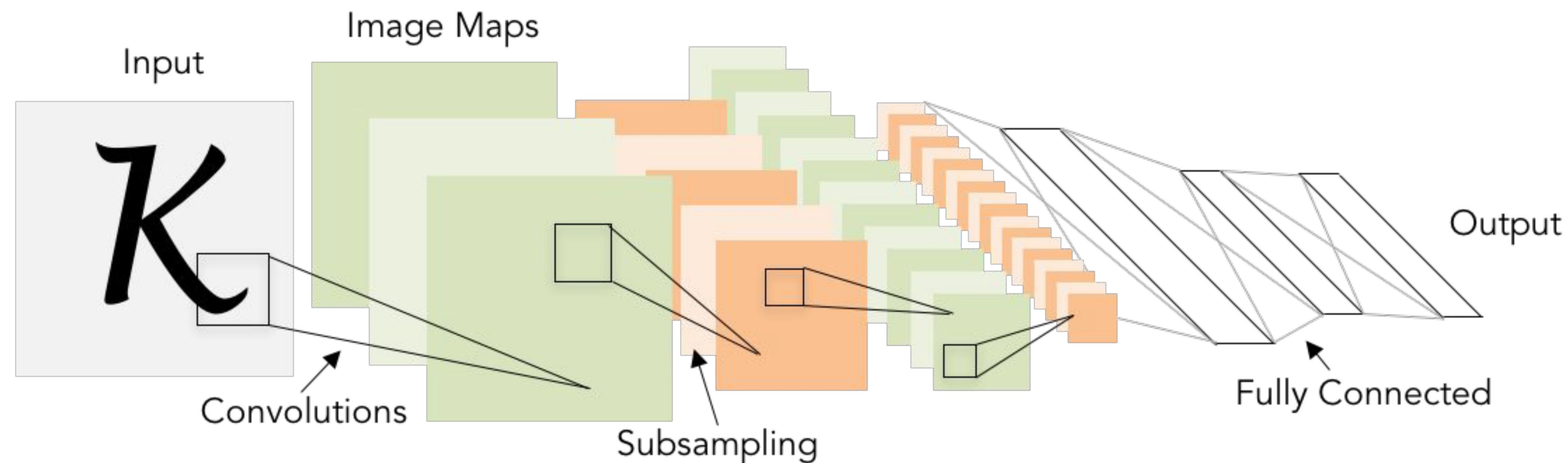


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

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.

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

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.

Co-financed by the European Union
Connecting Europe Facility

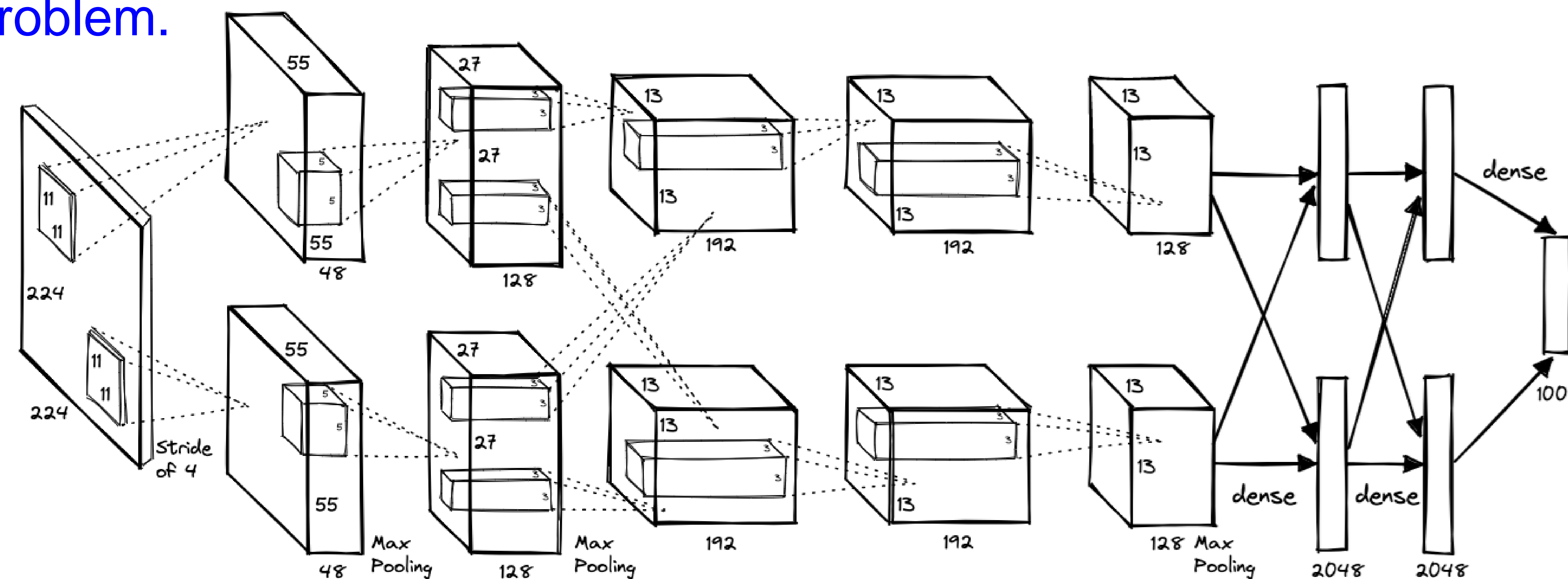
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *A bit of history*

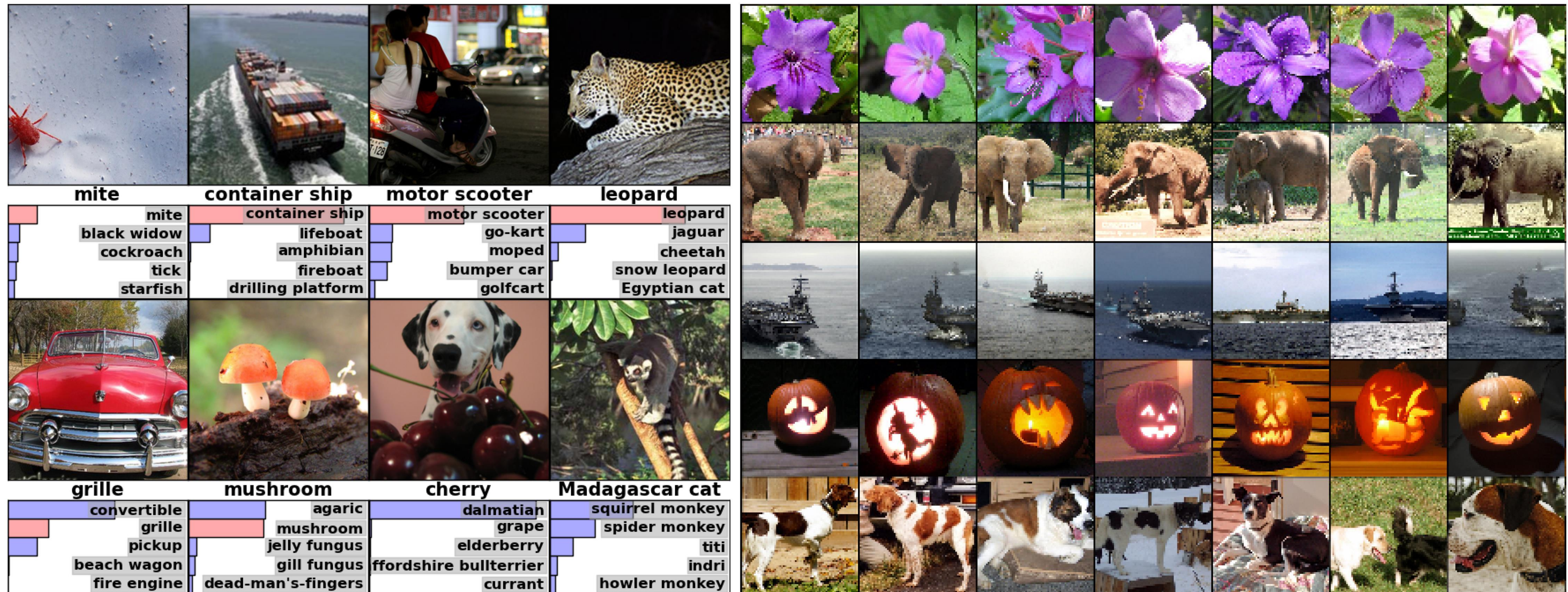
AlexNet is a deep convolutional neural network architecture that was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. At the time it was introduced, AlexNet was a breakthrough in the field of computer vision and image processing, achieving state-of-the-art performance on the 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*

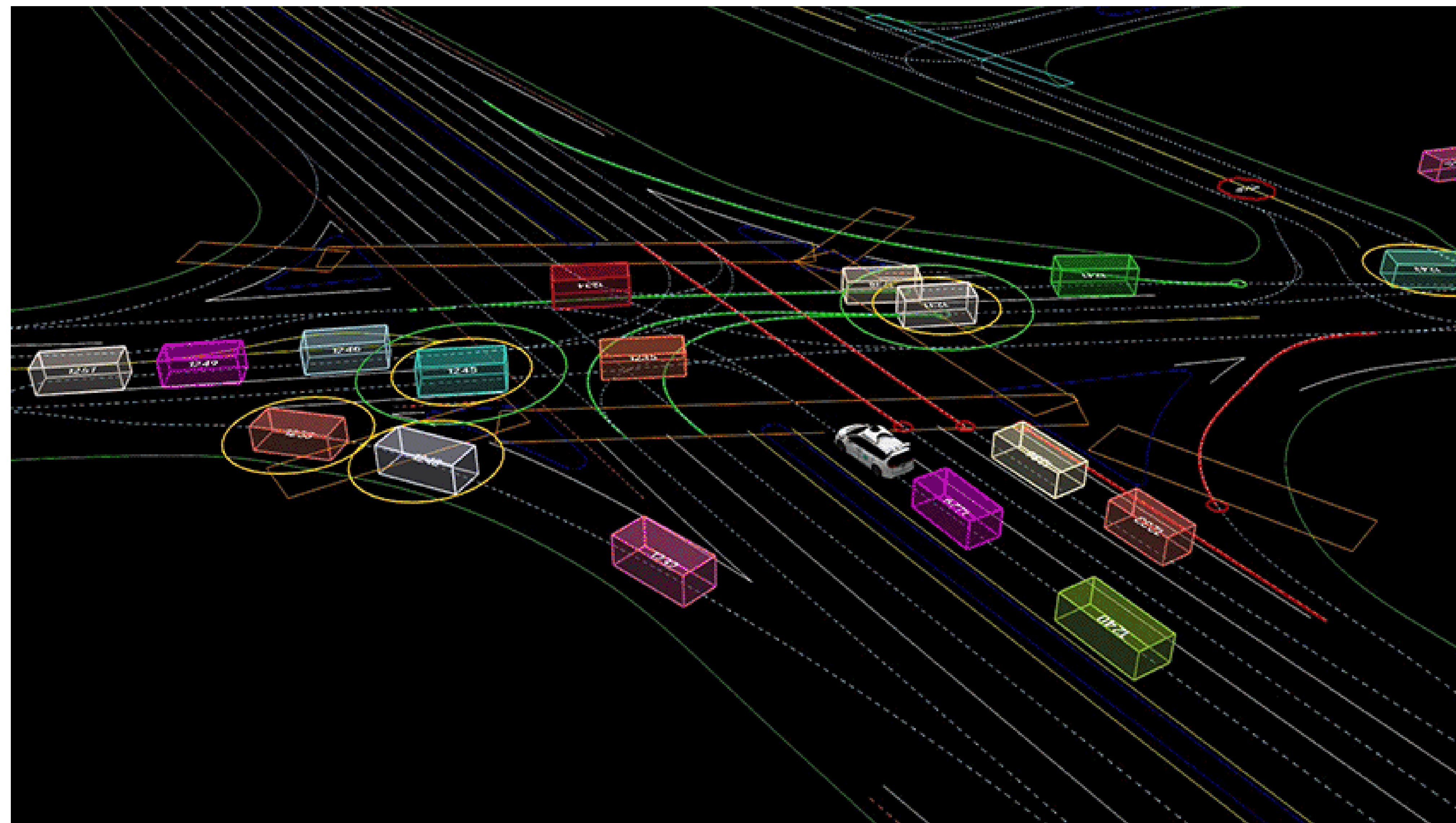


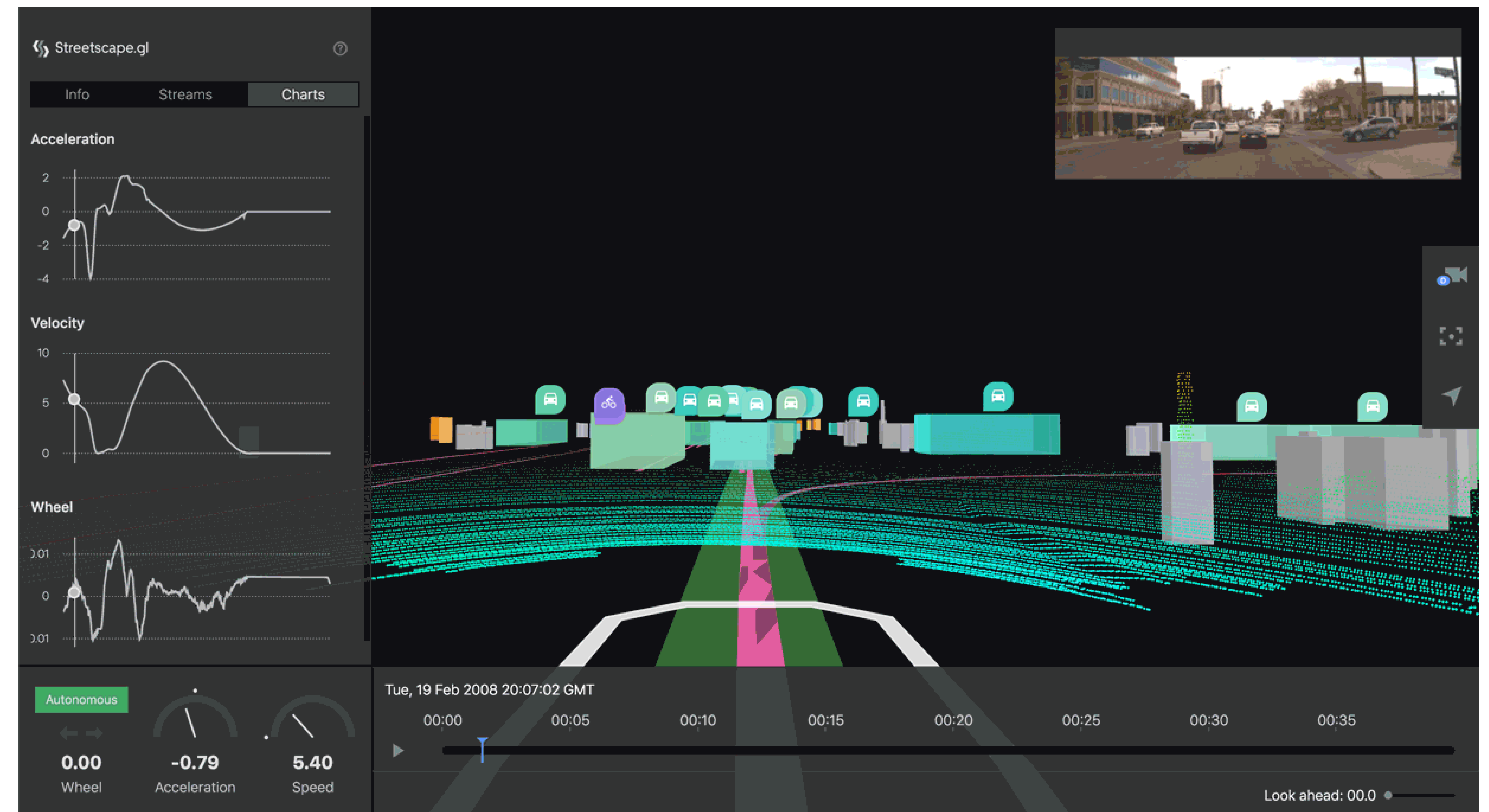
Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012

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

Object detection

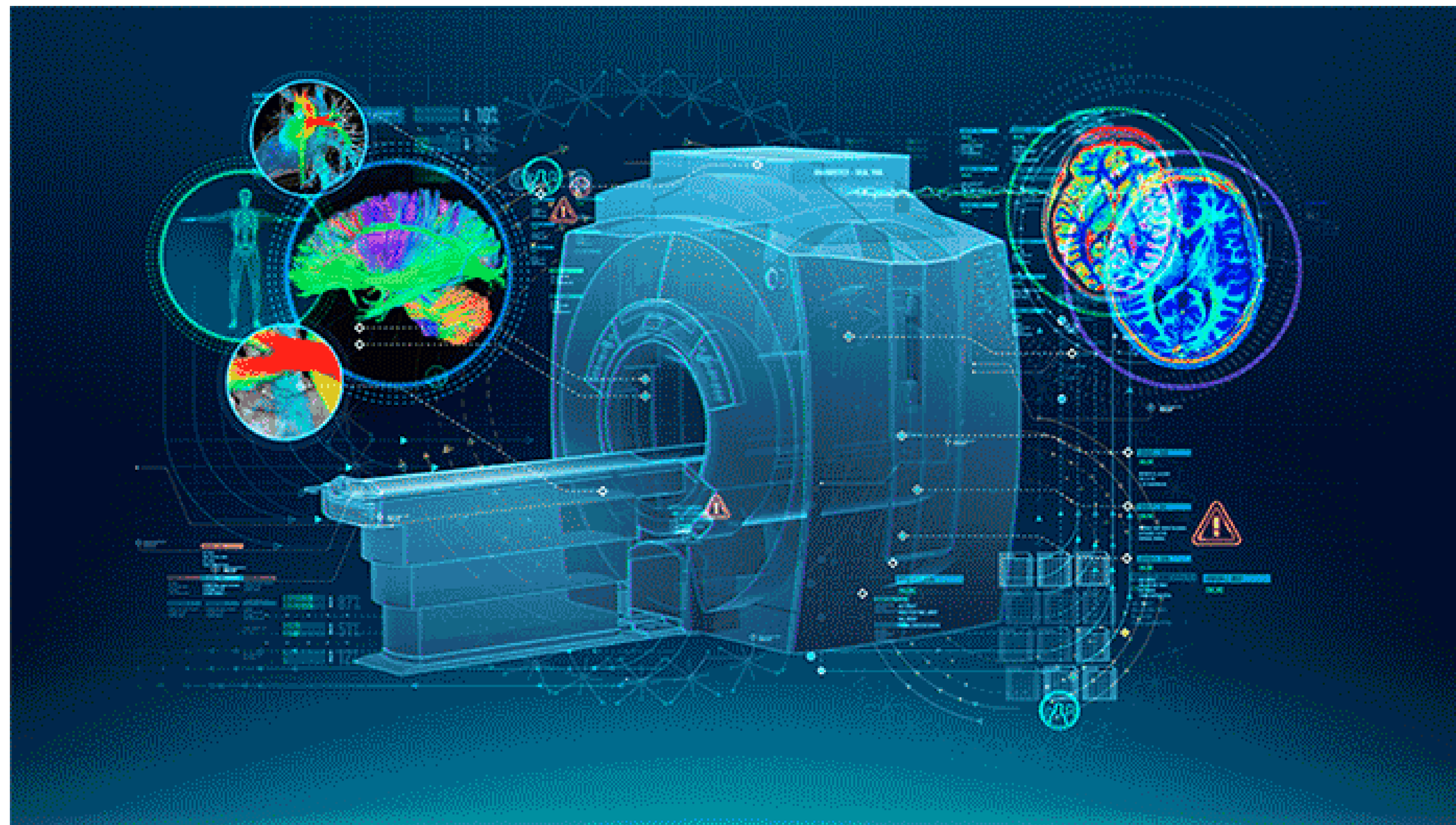
Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*

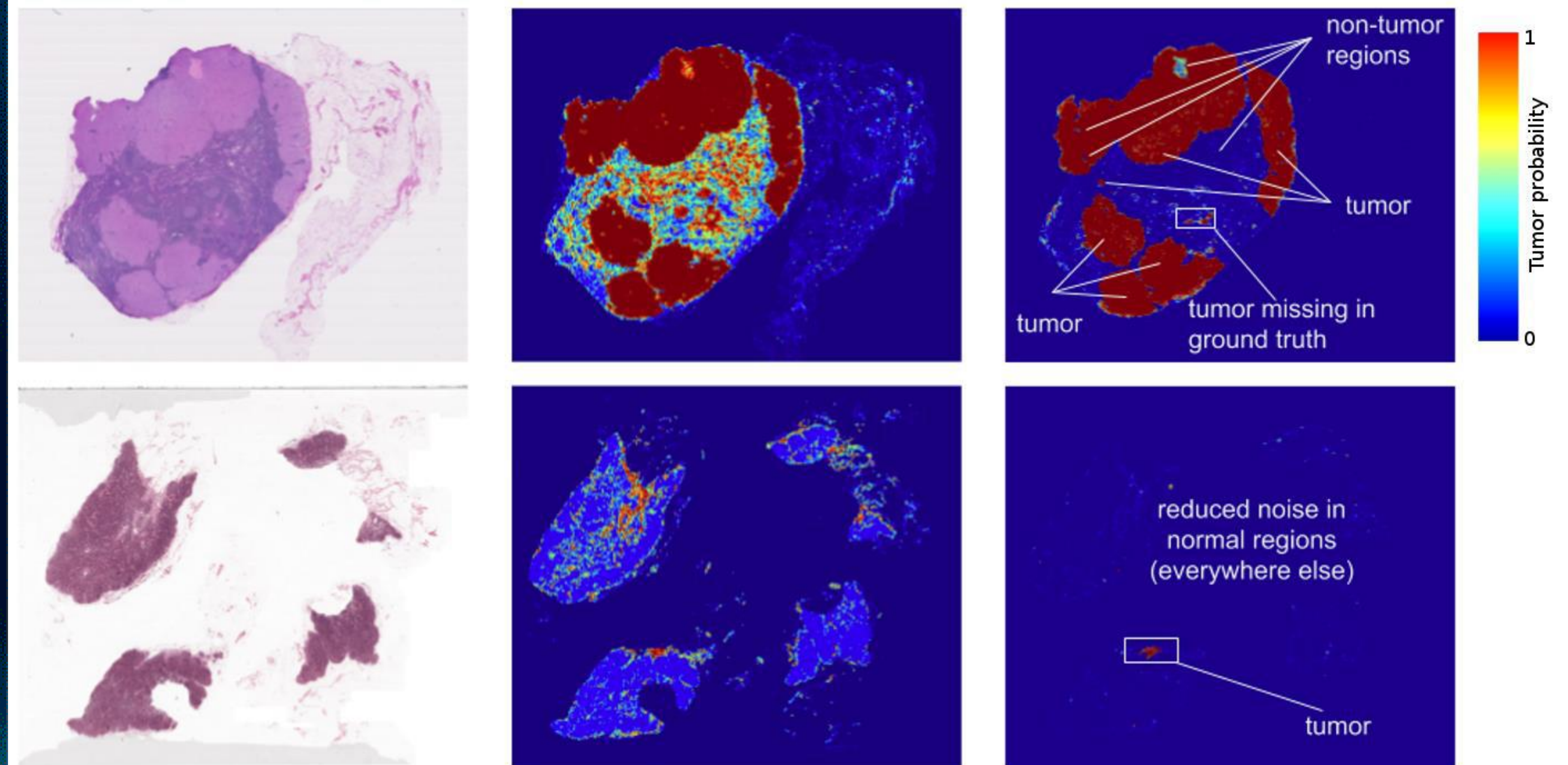




Self-driving cars

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





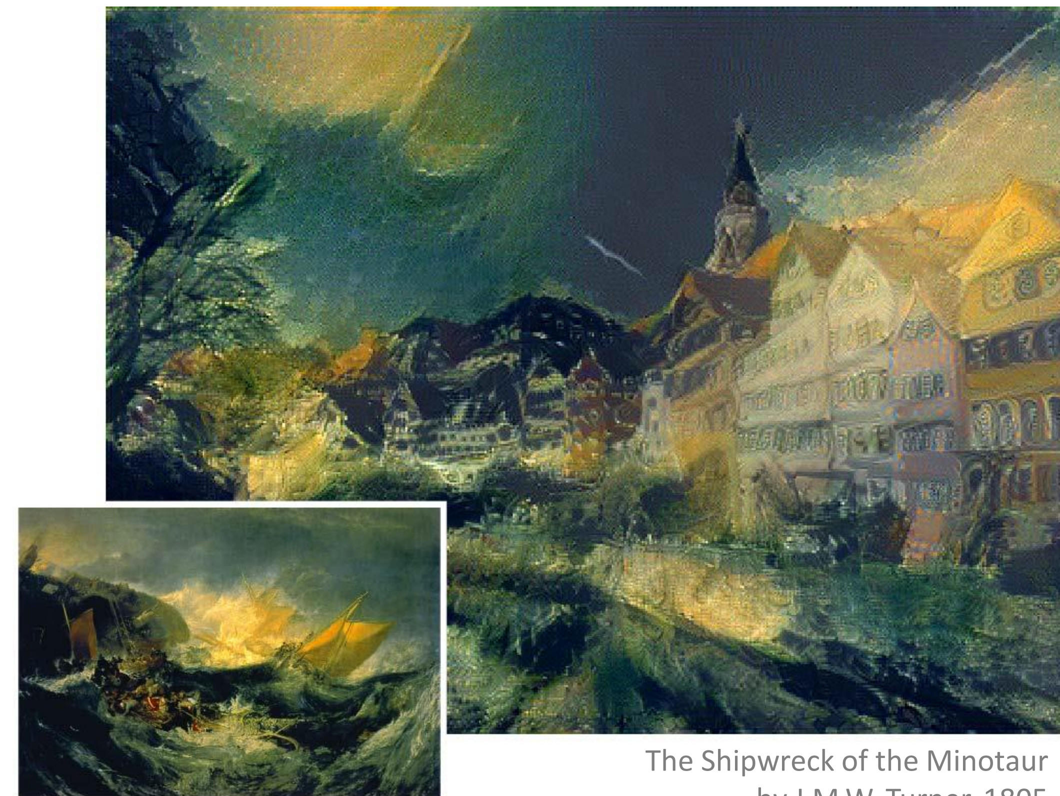
Healthcare, cancer detection

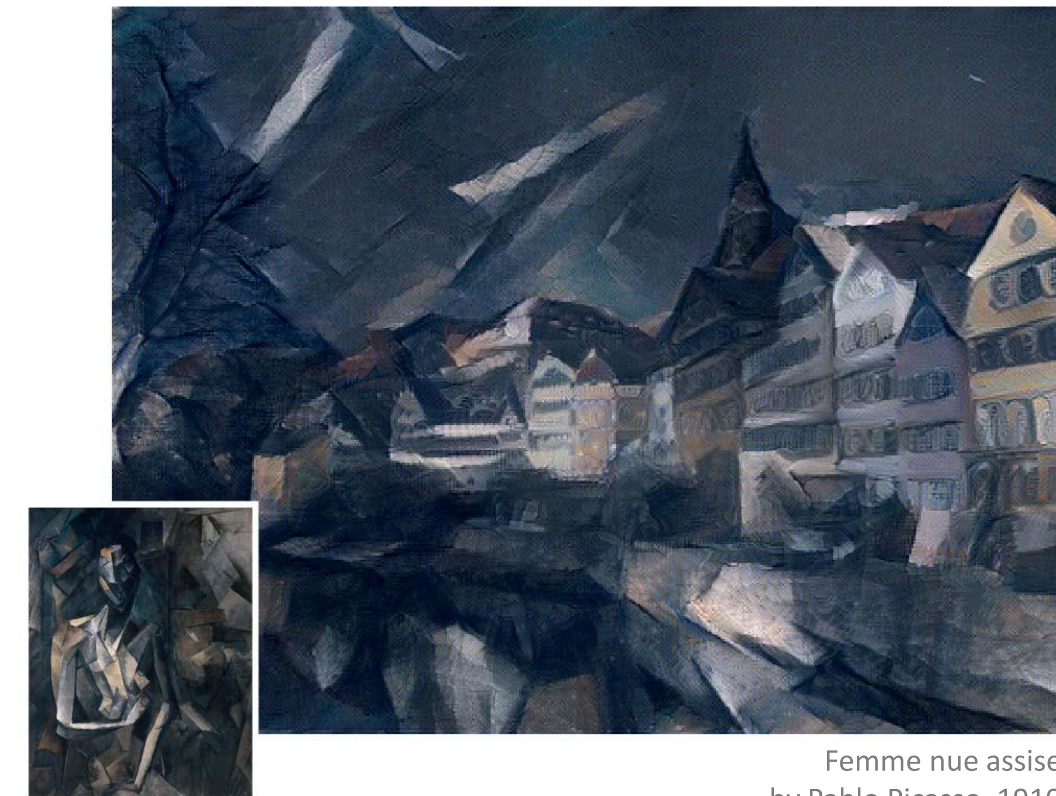
Co-financed by the European Union
Connecting Europe Facility

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

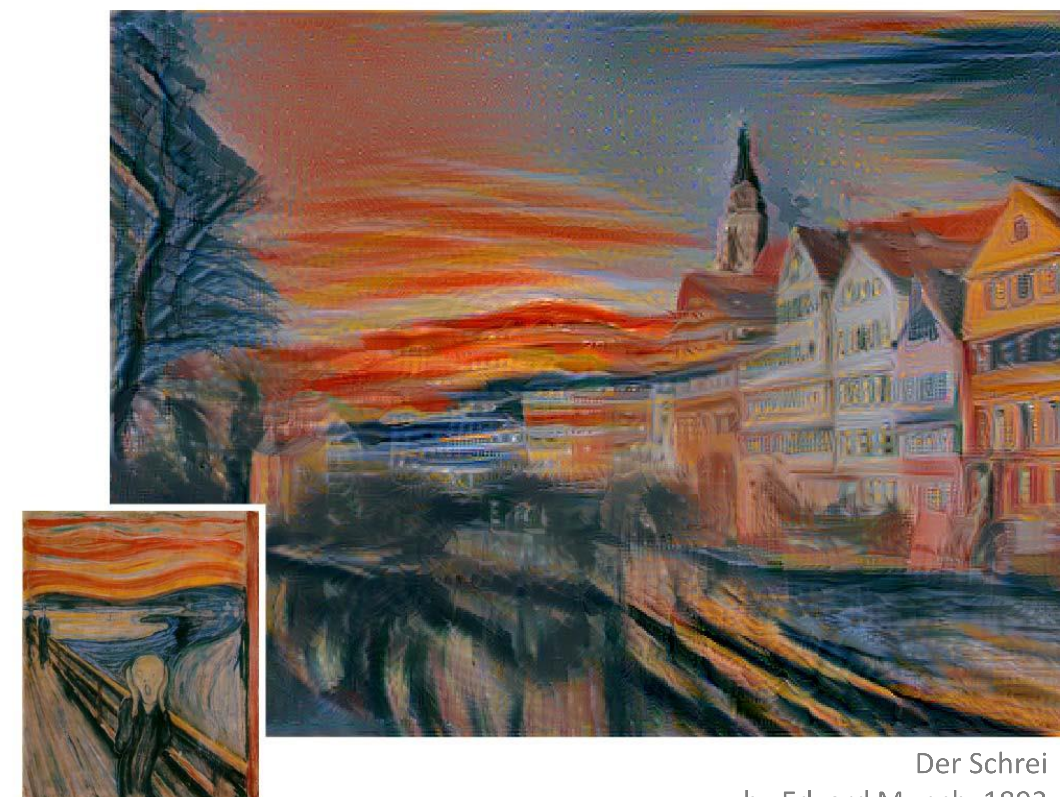
Convolutional Neural Networks: *Fast-forward to today - ConvNets are everywhere*

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

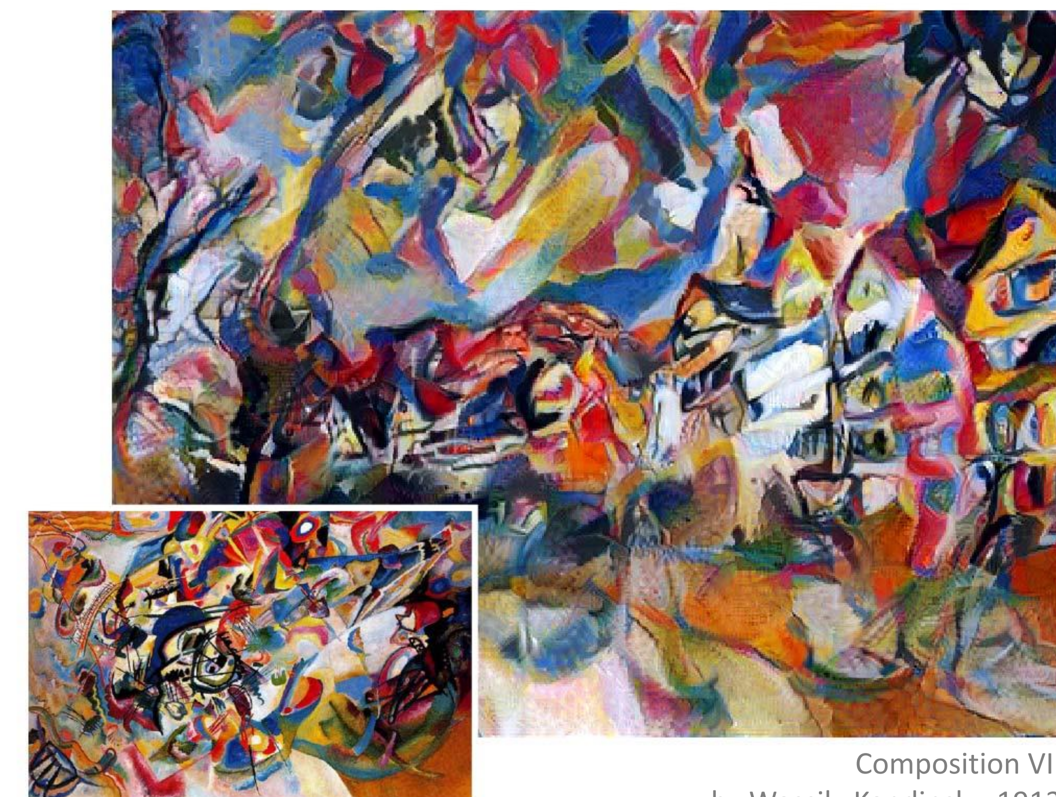

Composition VII by Wassily Kandinsky, 1913

Image style transfer

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

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*

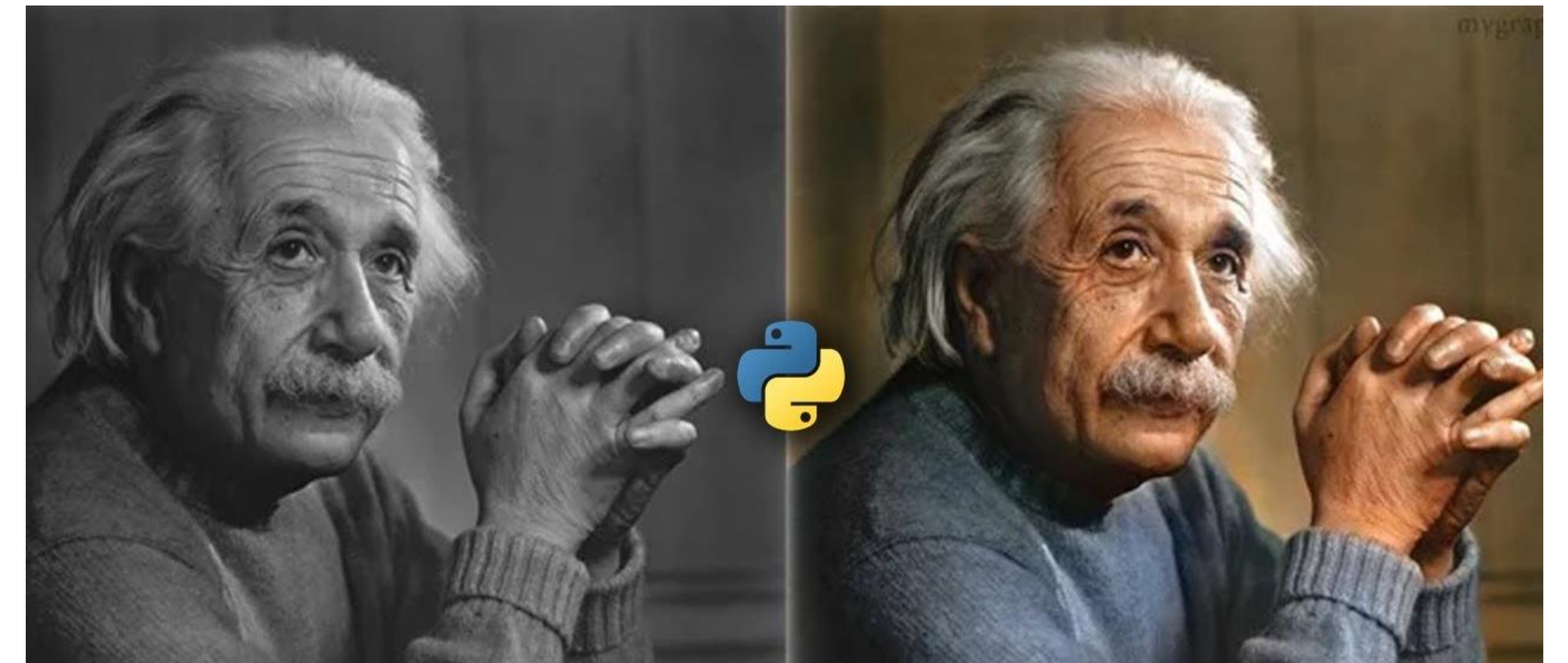
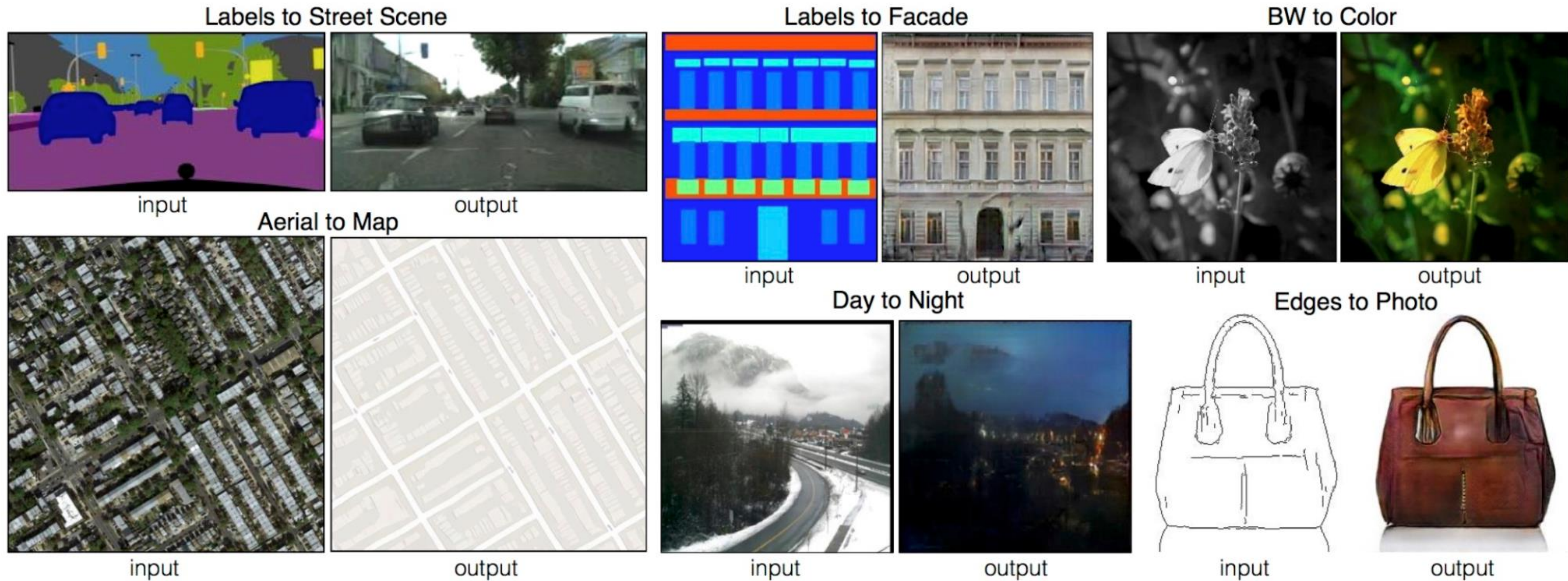


Image coloring

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*



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*

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.

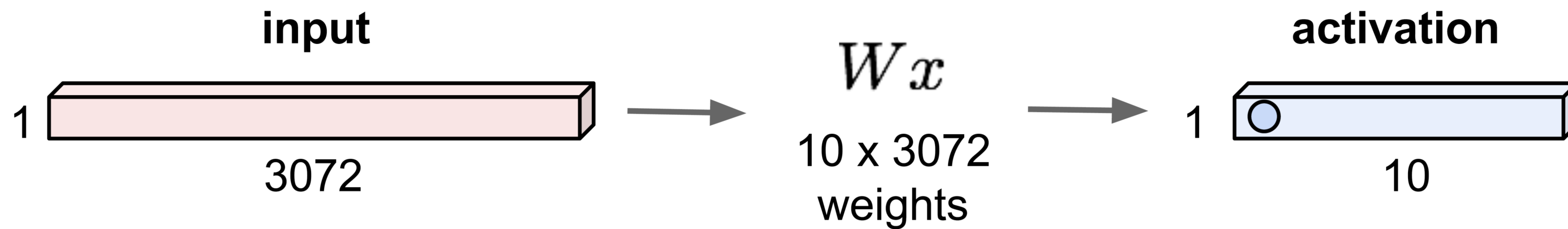
In contrast, a **convolutional layer** processes input data using a set of learnable filters (also known as kernels or weights). Each filter is applied to a small region of the input data, and the output from each filter is then combined to create a feature map. This process is repeated for each region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

Co-financed by the European Union
Connecting Europe Facility

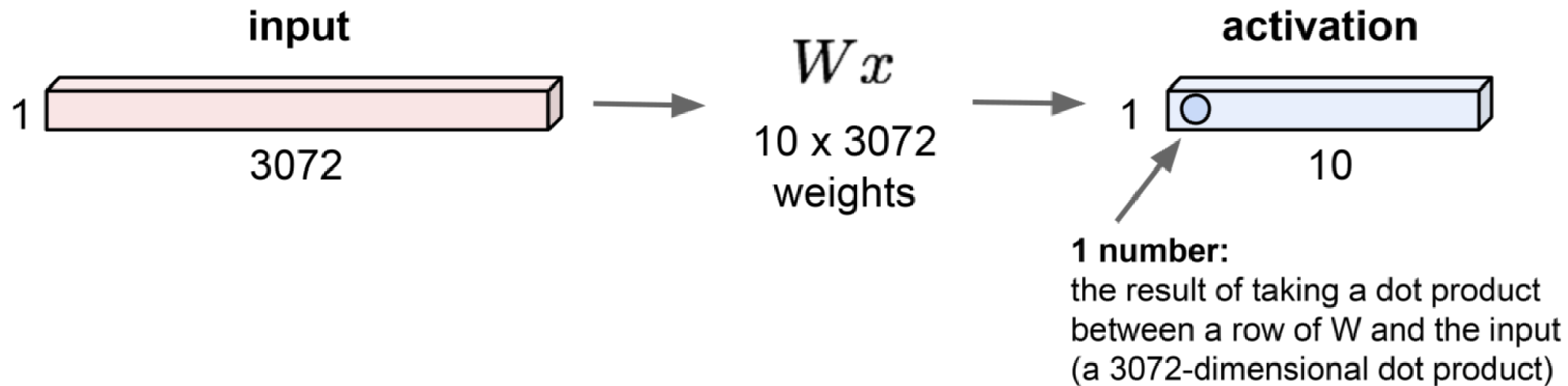
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Fully Connected Layer*

32x32x3 image -> stretch to 3072 x 1



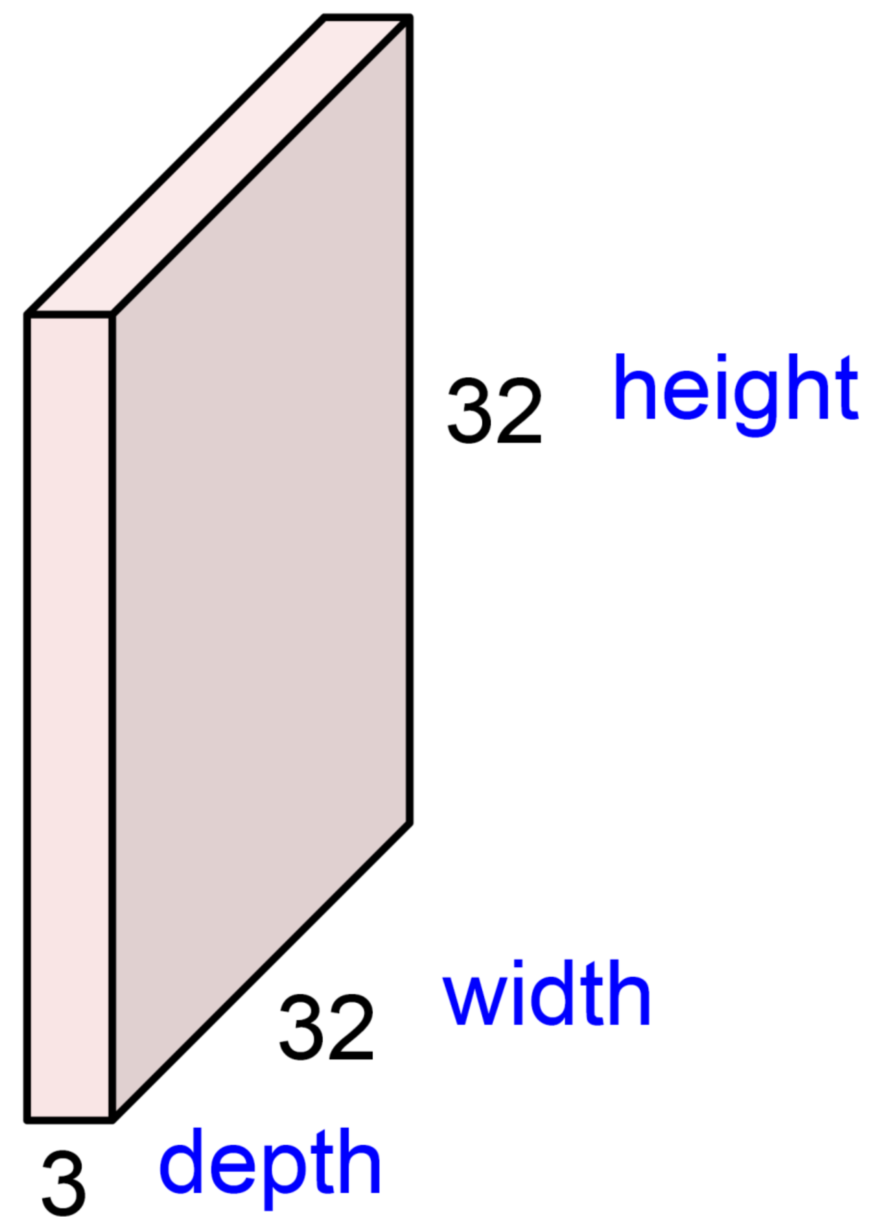
Convolutional Neural Networks: *Fully Connected Layer*



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

Convolutional Neural Networks: *Convolution Layer*

32x32x3 image -> preserve spatial structure

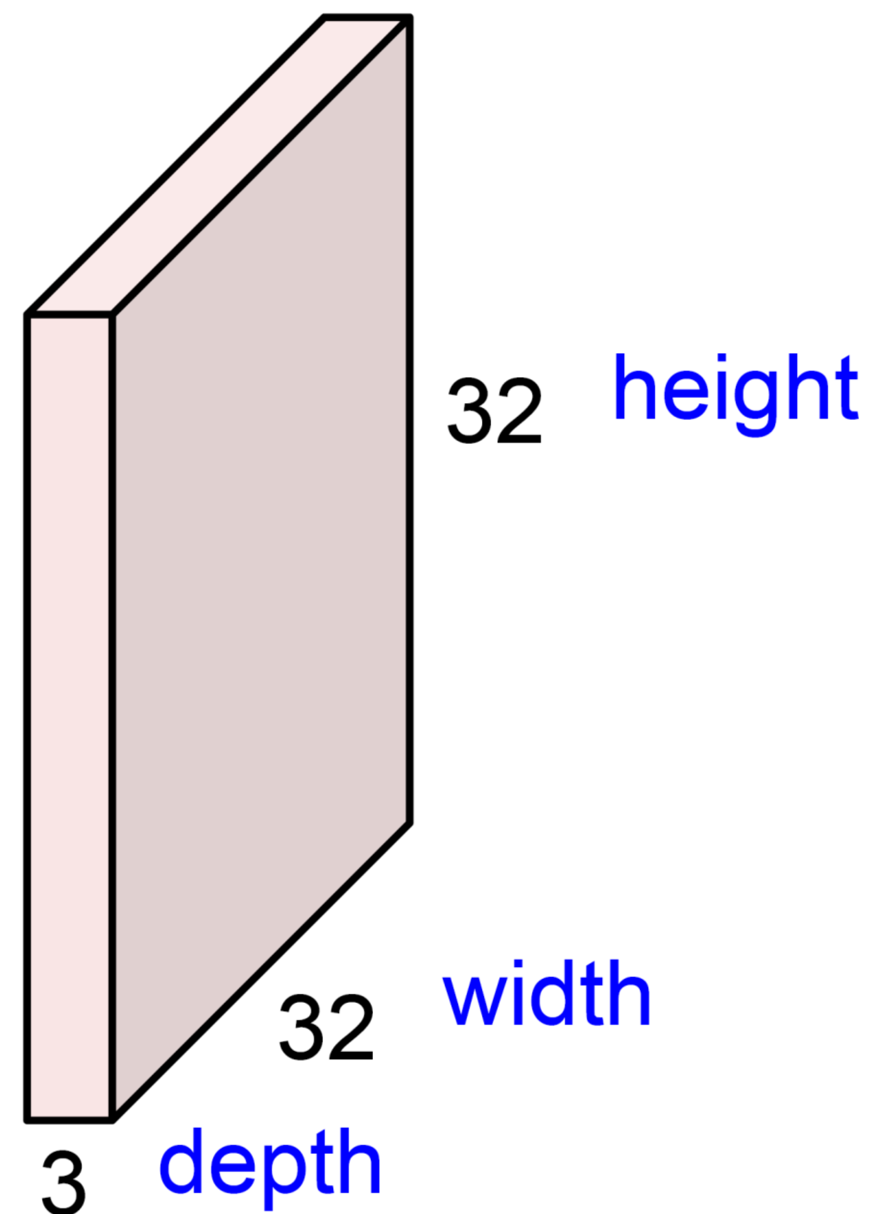


Co-financed by the European Union
Connecting Europe Facility

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*

32x32x3 image

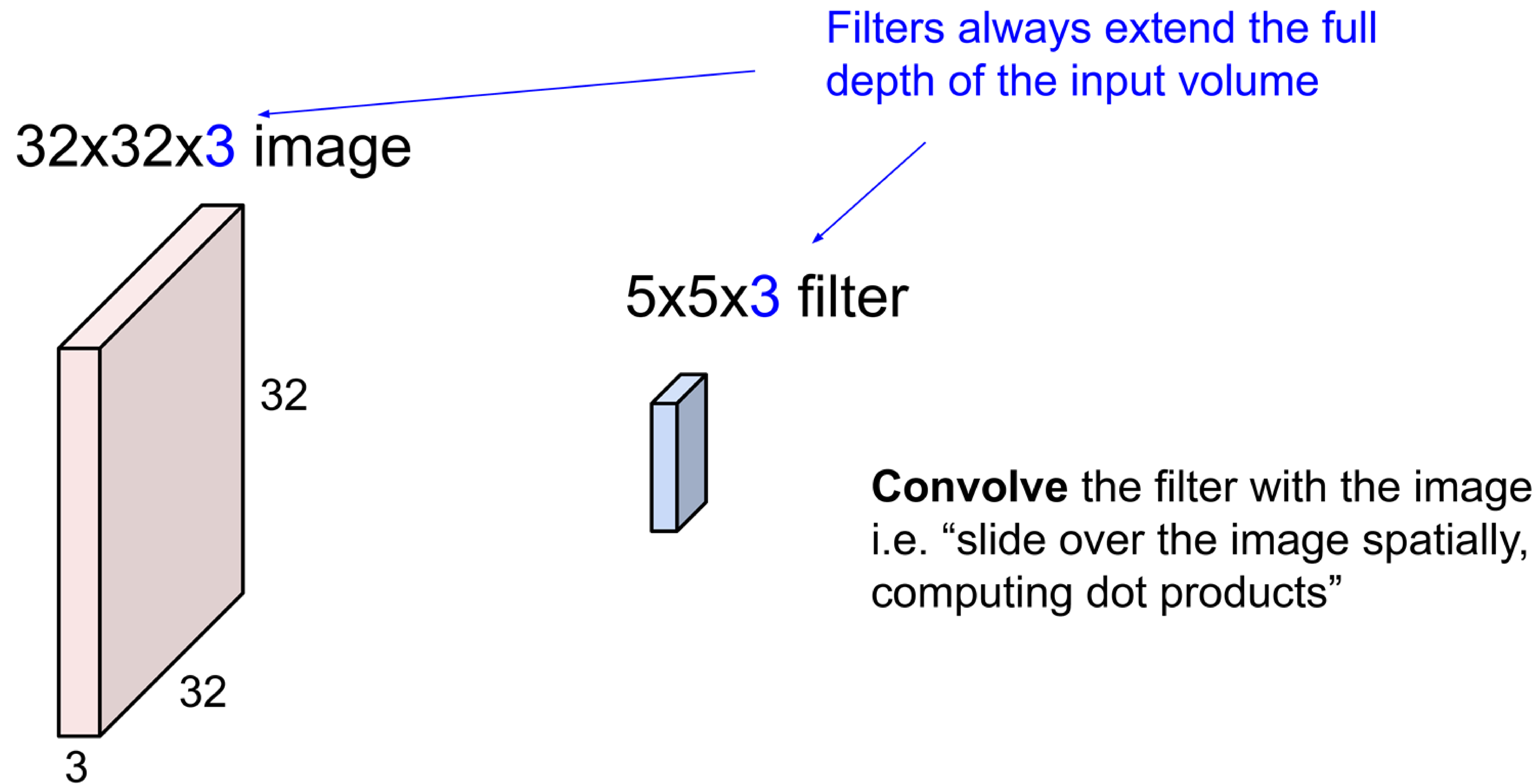


5x5x3 filter

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

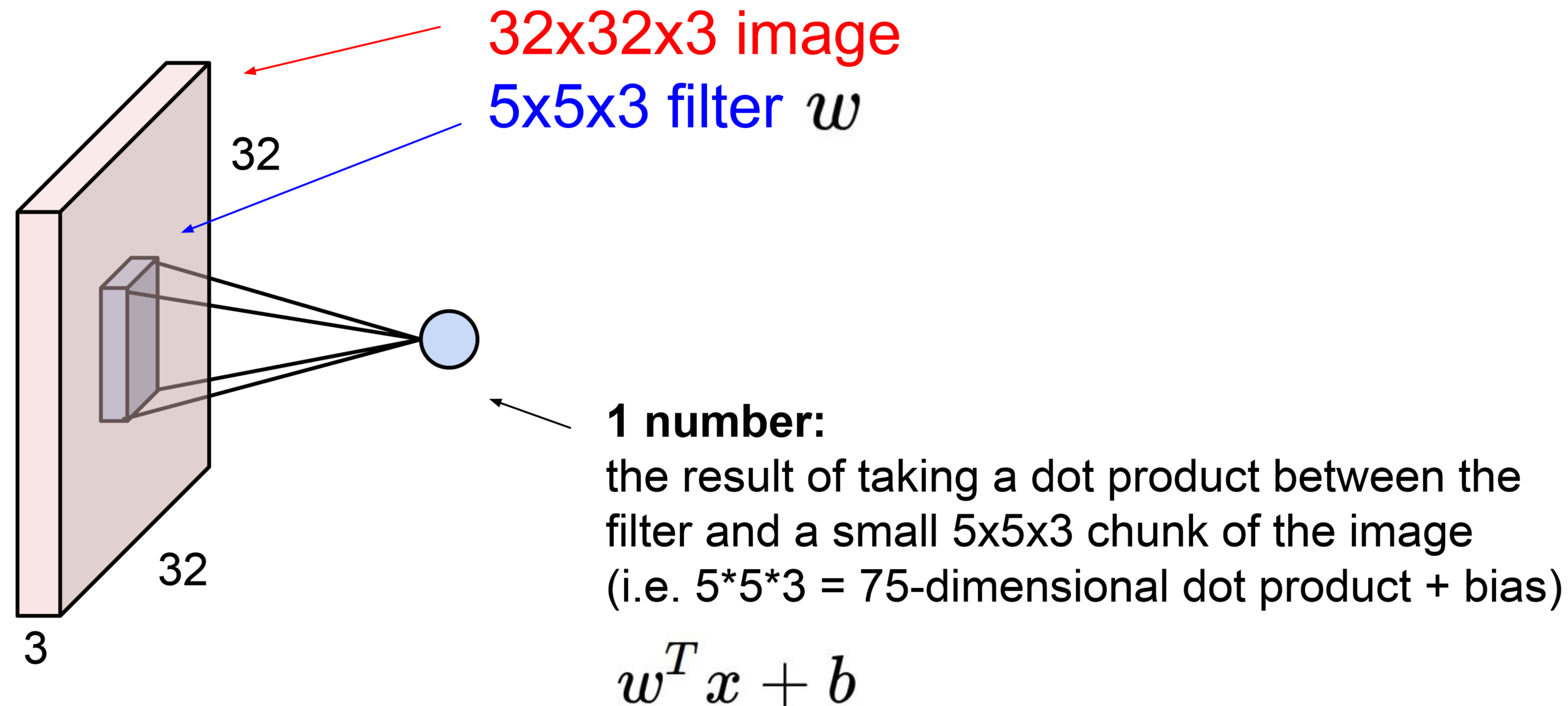
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



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

Convolutional Neural Networks: *Convolution Layer*



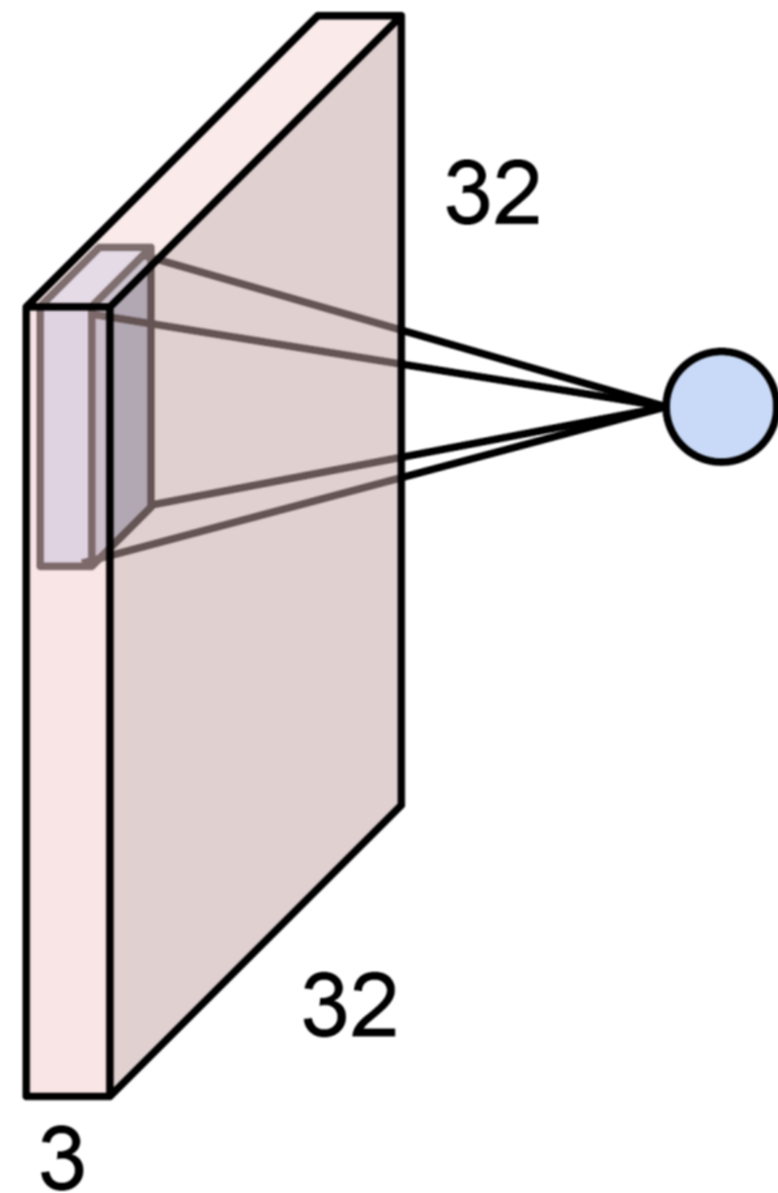
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Co-financed by the European Union
Connecting Europe Facility

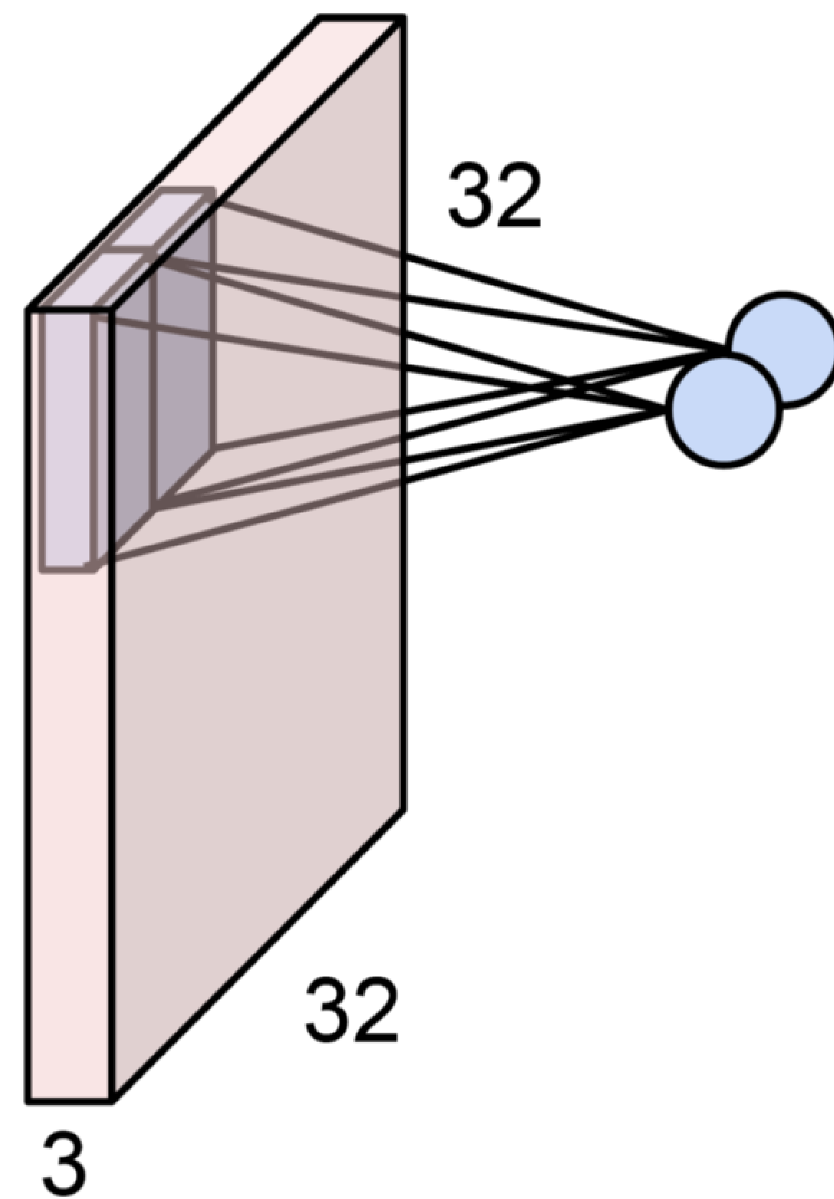
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



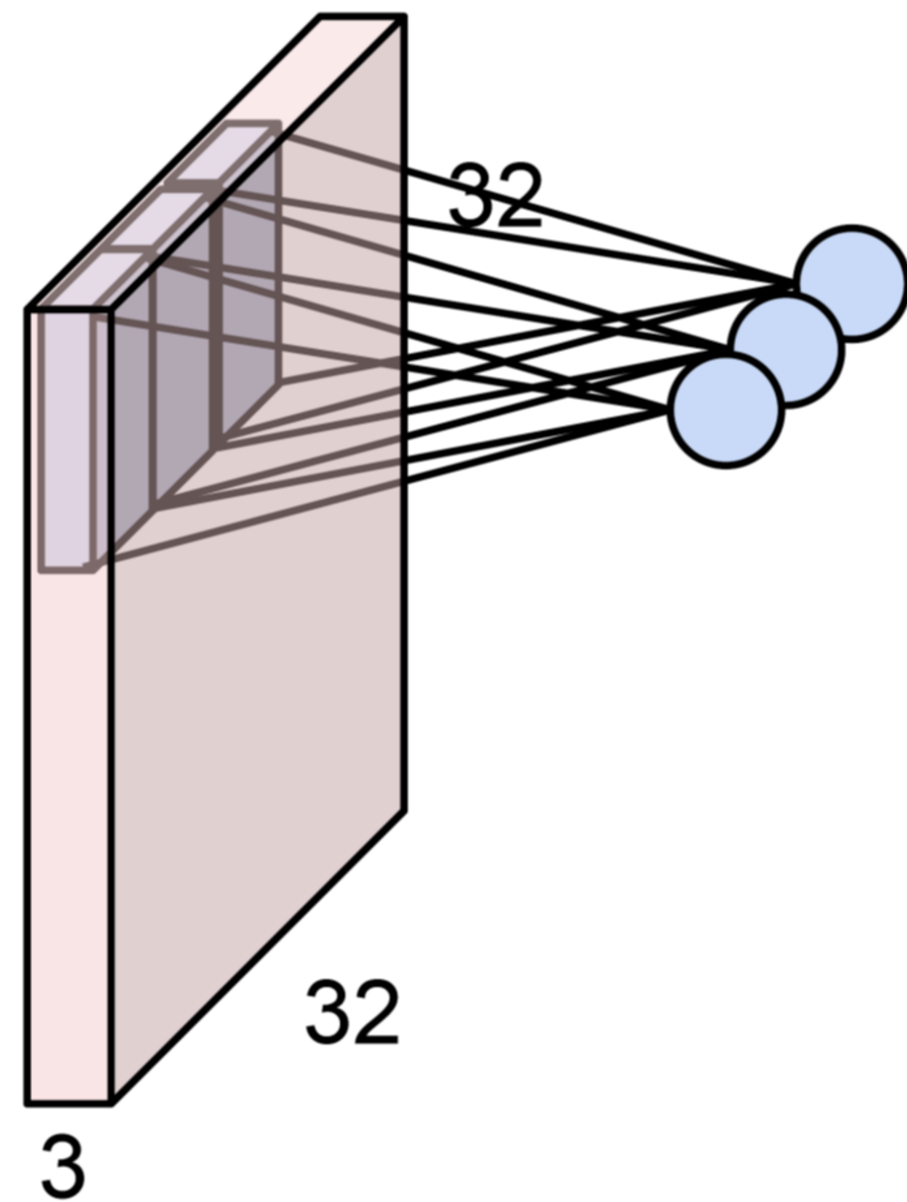
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



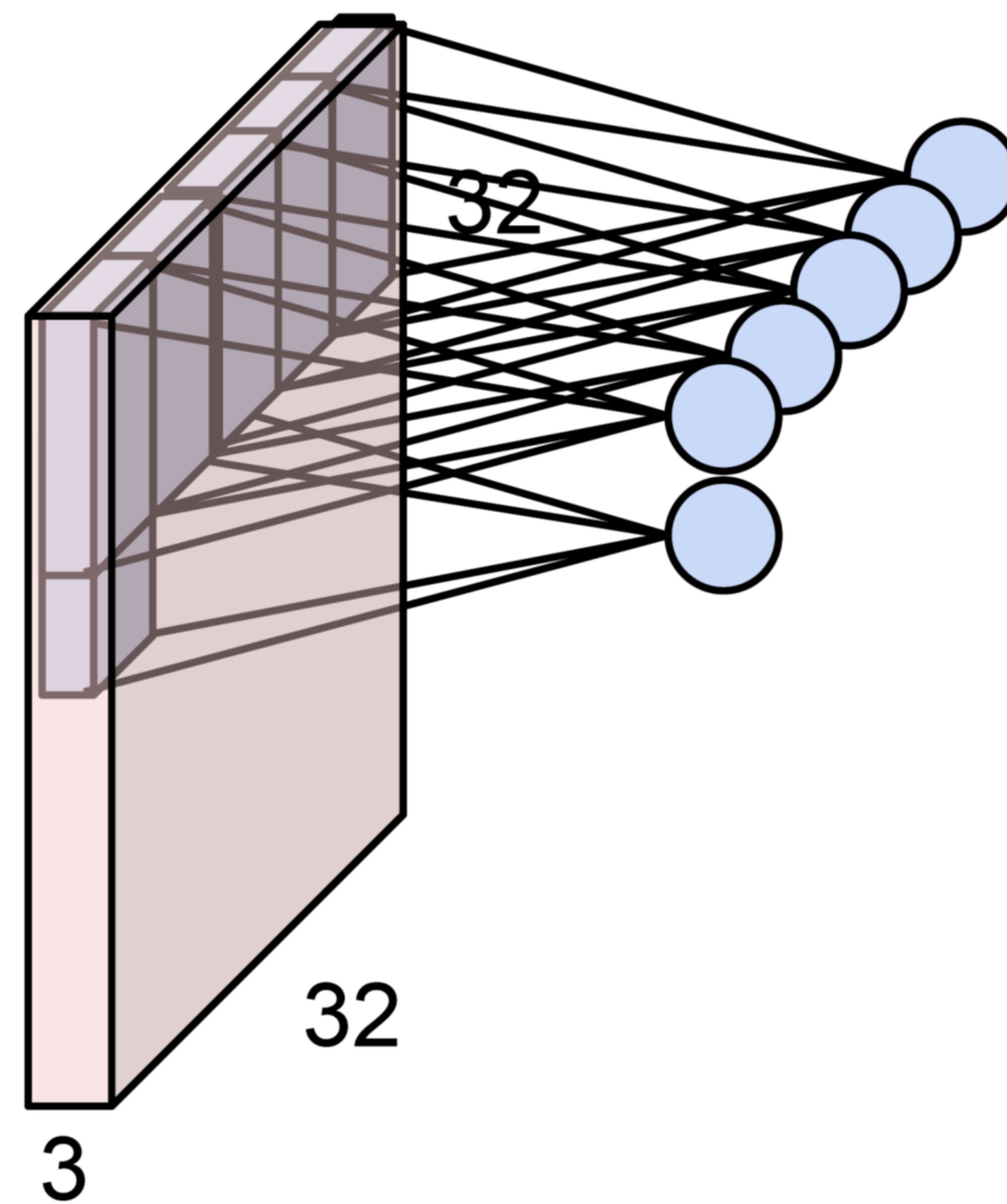
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*



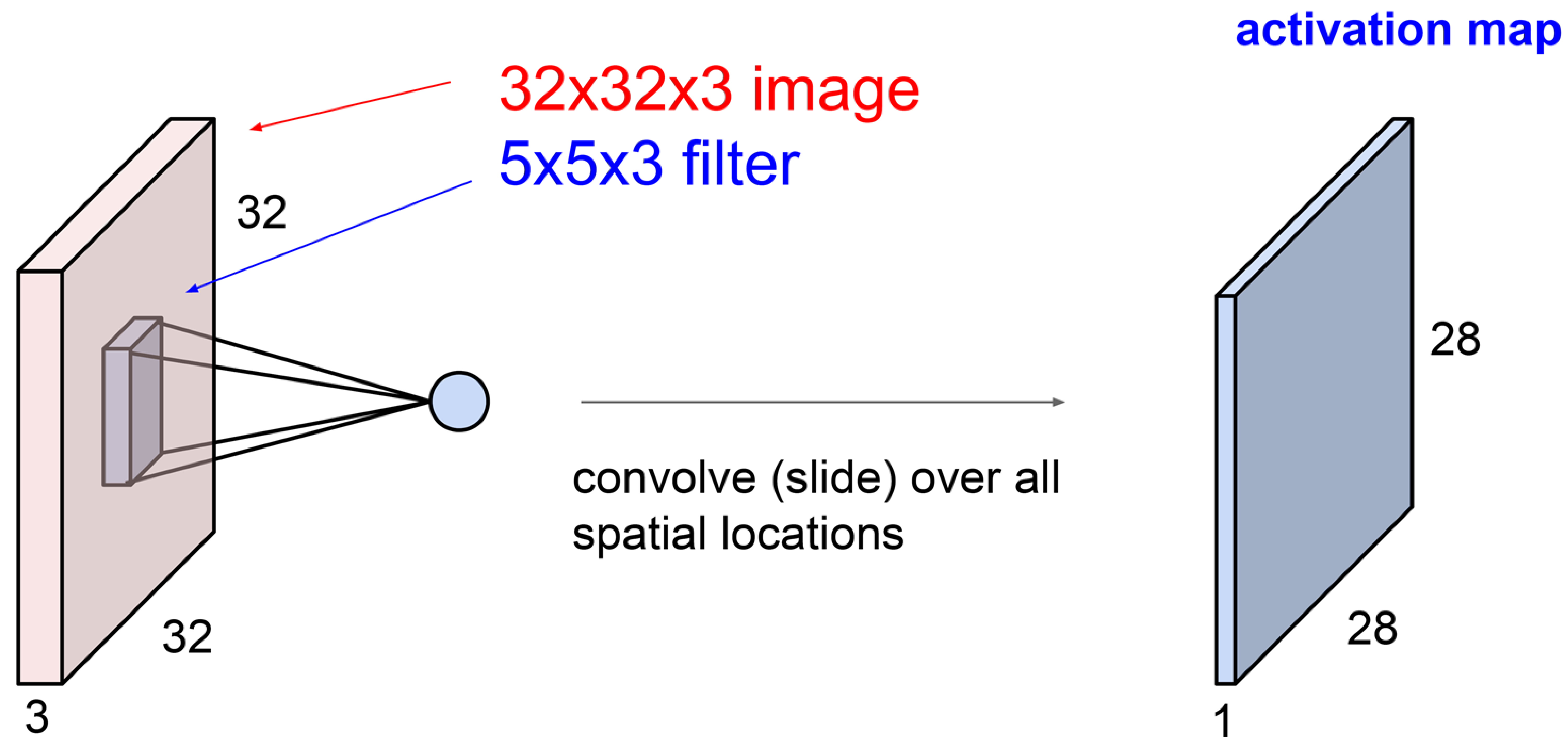
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Convolutional Neural Networks: *Convolution Layer*

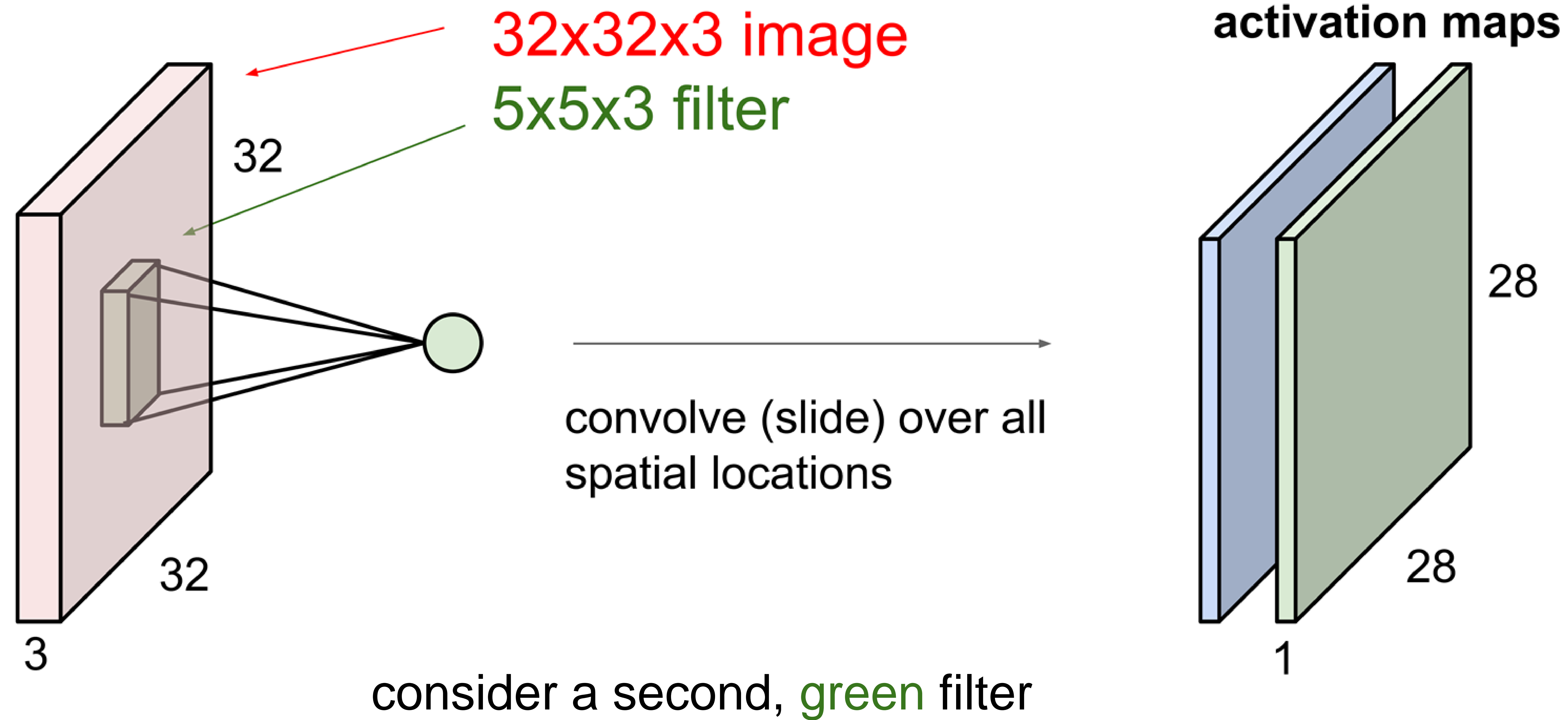


This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

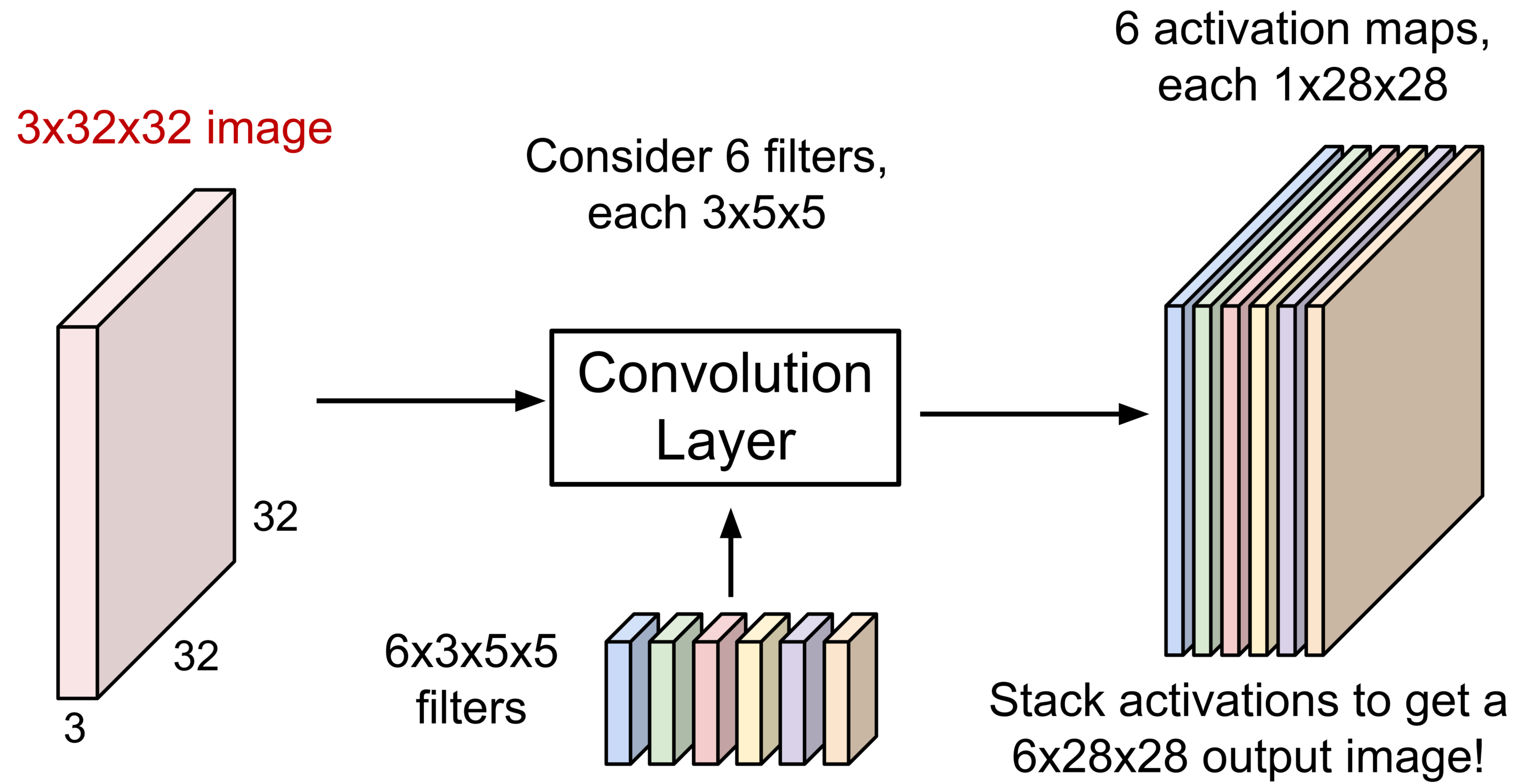
Convolutional Neural Networks: *Convolution Layer*



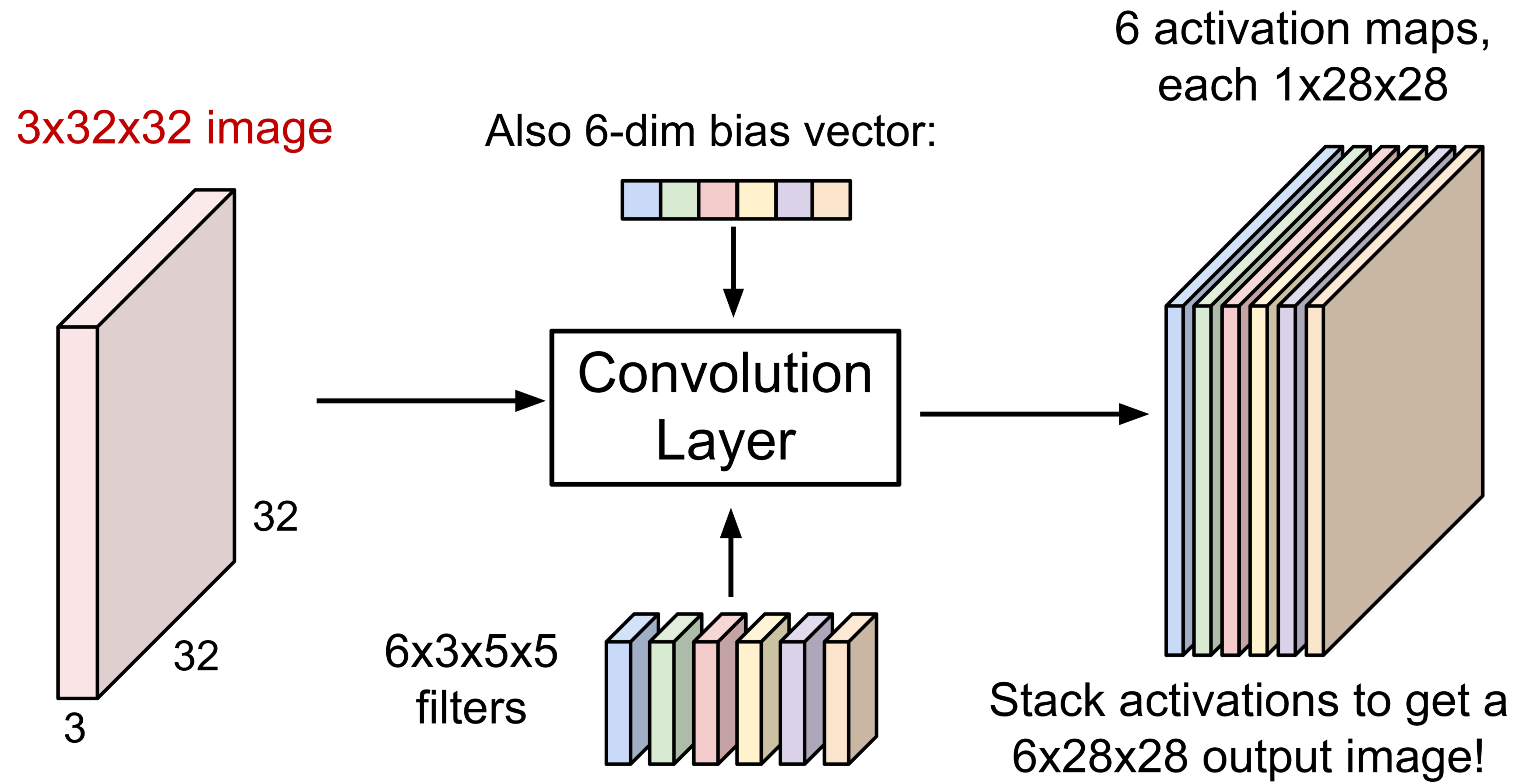
Convolutional Neural Networks: *Convolution Layer*



Convolutional Neural Networks: *Convolution Layer*



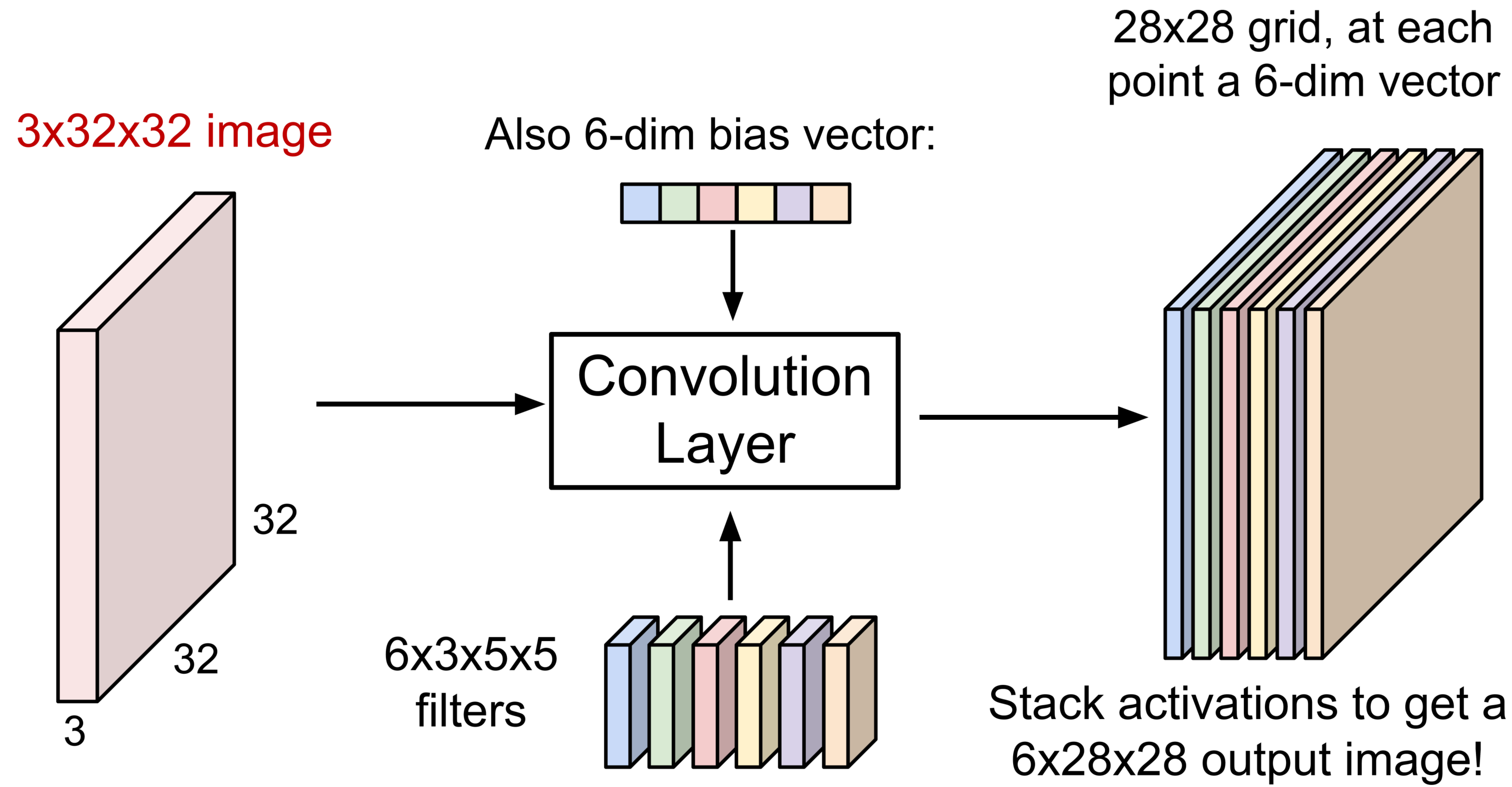
Convolutional Neural Networks: *Convolution Layer*



6 activation maps, each 1x28x28

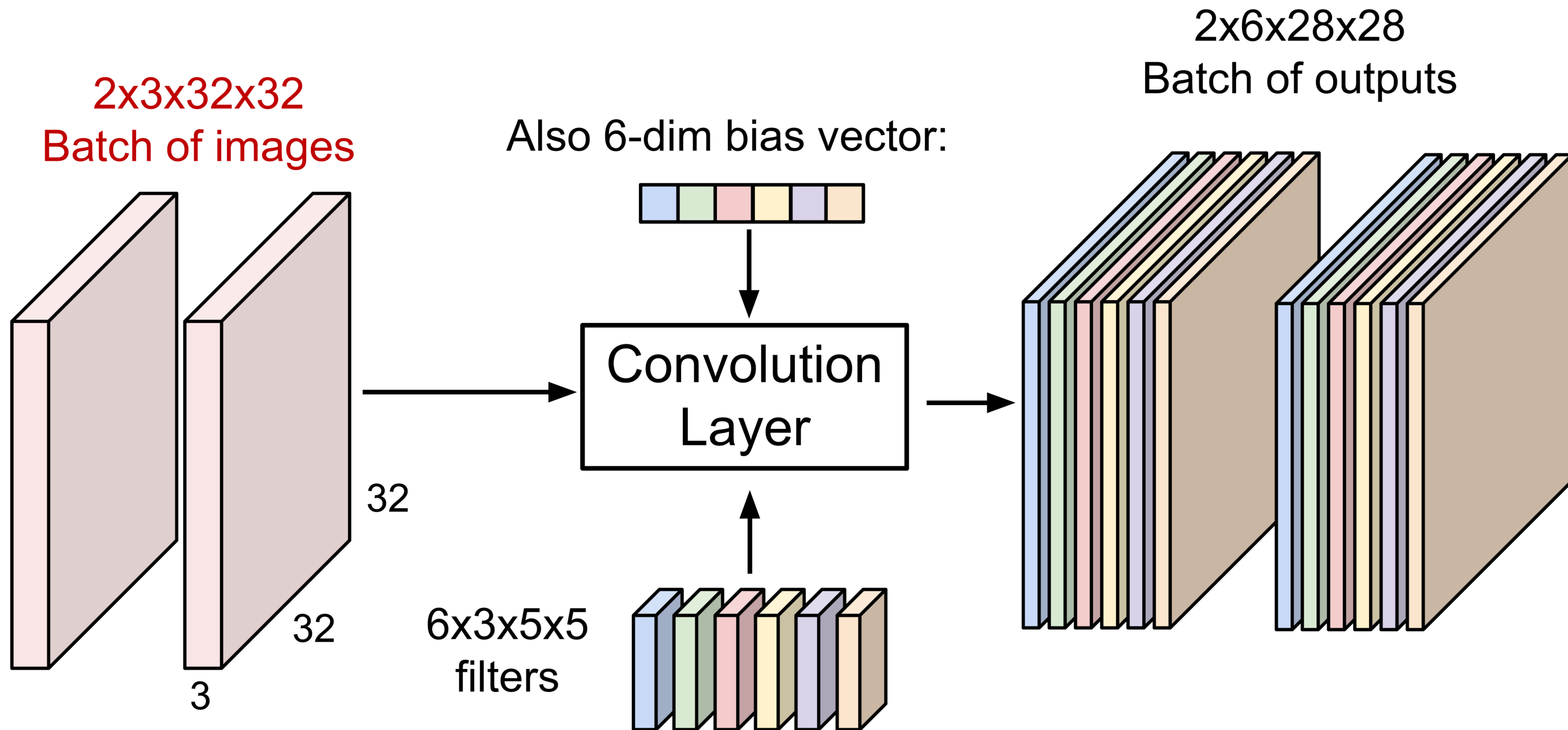
Stack activations to get a 6x28x28 output image!

Convolutional Neural Networks: *Convolution 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

Convolutional Neural Networks: *Convolution Layer*



Convolutional Neural Networks: *Convolution Layer*

$N \times C_{in} \times H \times W$
Batch of images

Also C_{out}-dim bias vector:

$N \times C_{out} \times H' \times W'$
Batch of outputs

Convolution Layer

H

W

C_{in}

C_{out}

$C_{out} \times Ci_n \times K_w \times K_h$
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.

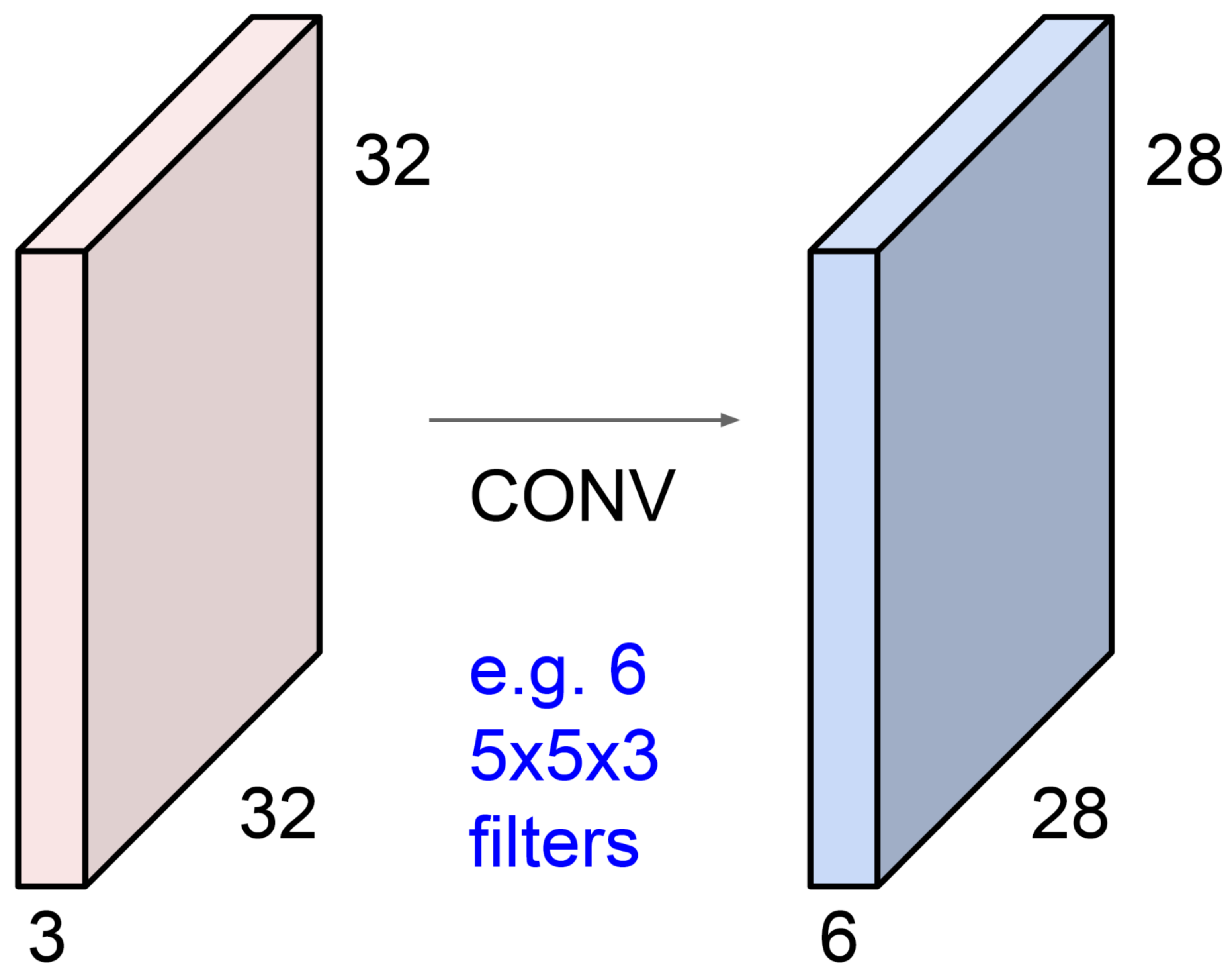
Another important difference between fully connected and convolutional layers is the way they handle input data. Fully connected layers require input data to be flattened into a one-dimensional vector, while convolutional layers can accept input data with multiple dimensions (e.g., height, width, and 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.

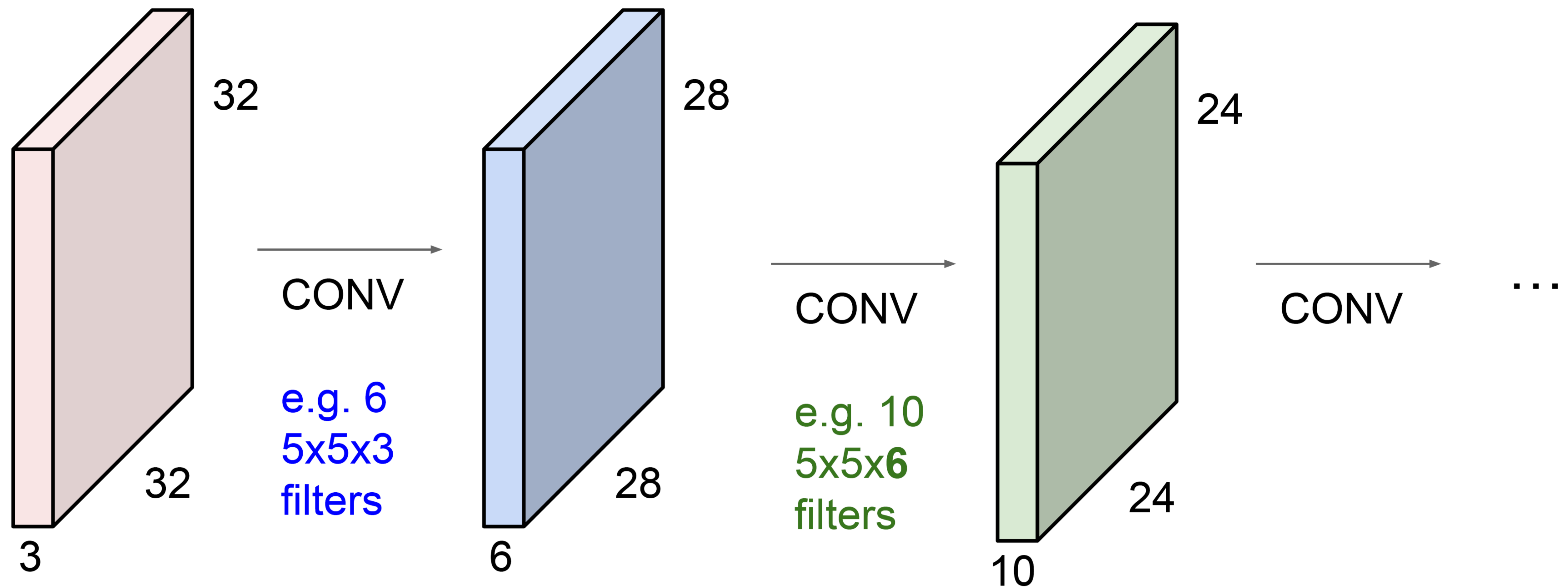
Co-financed by the European Union
Connecting Europe Facility

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

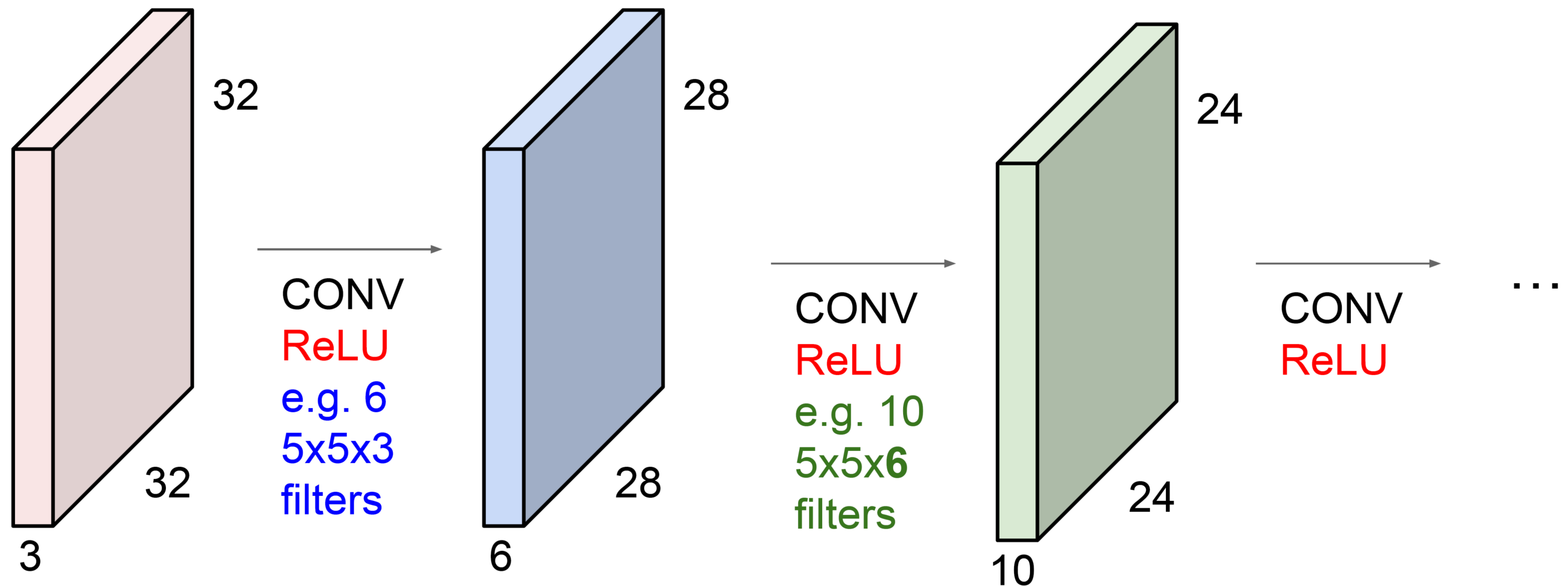
ConvNet is a sequence of Convolution Layers



ConvNet is a sequence of Convolution Layers



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



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.

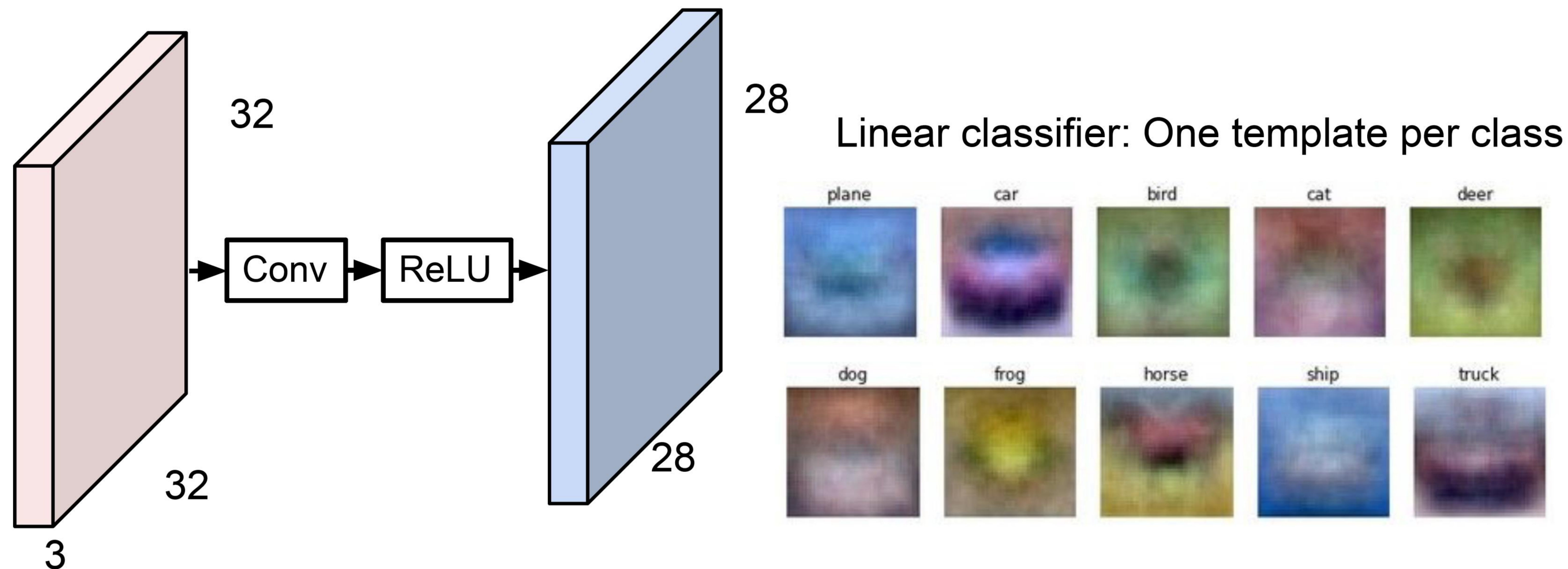
It is important to note that filters in a convolutional layer are designed to be translation-invariant, meaning that they can recognize the same pattern regardless of where it appears in the input data. This is achieved by sharing the same set of weights across different regions of the input.

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?

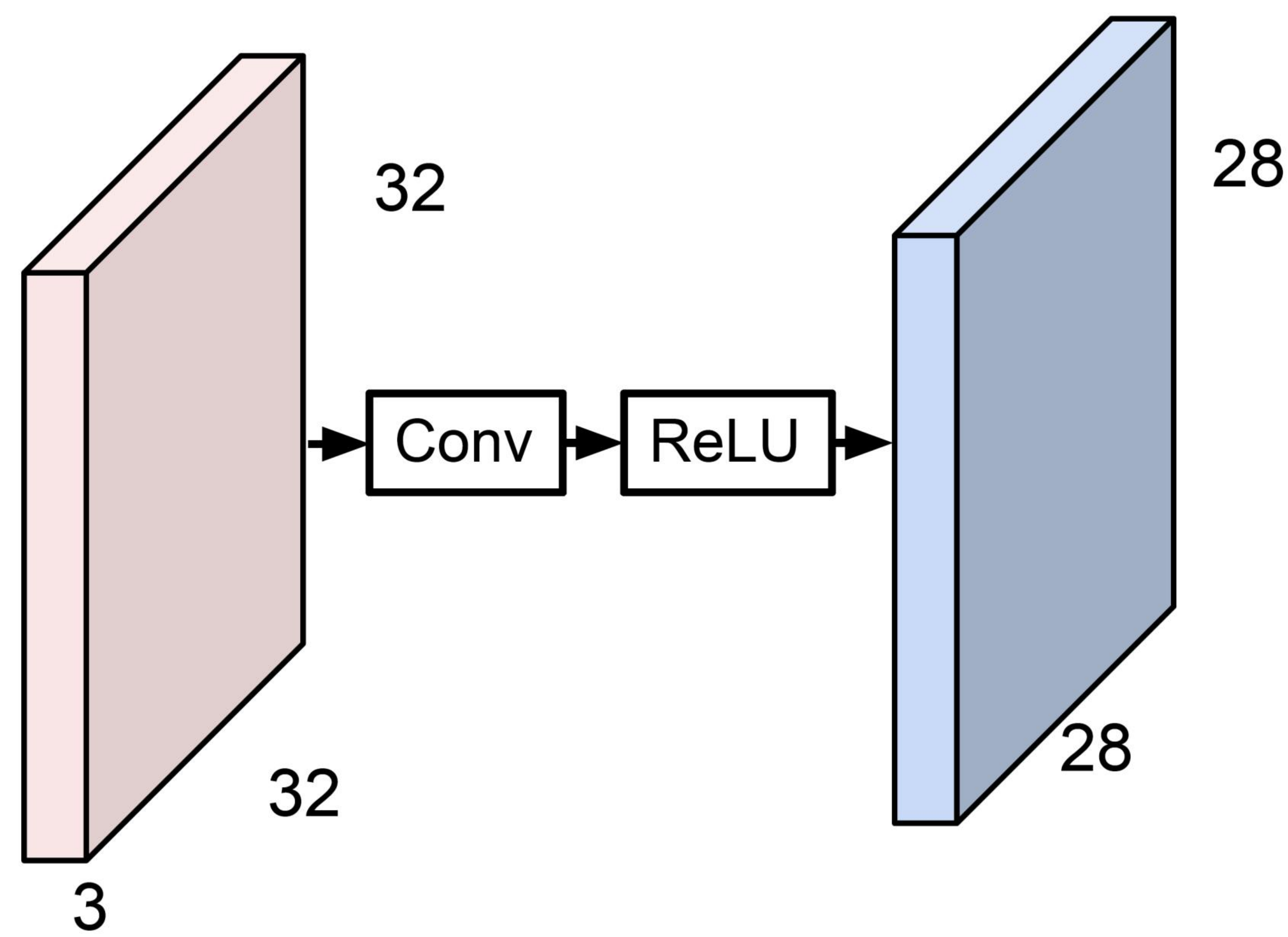
Overall, convolutional filters learn to extract relevant features from the input data, which can be used to make accurate predictions or classifications. By stacking multiple convolutional layers, a deep learning model can learn increasingly complex representations of the input data, which enables it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



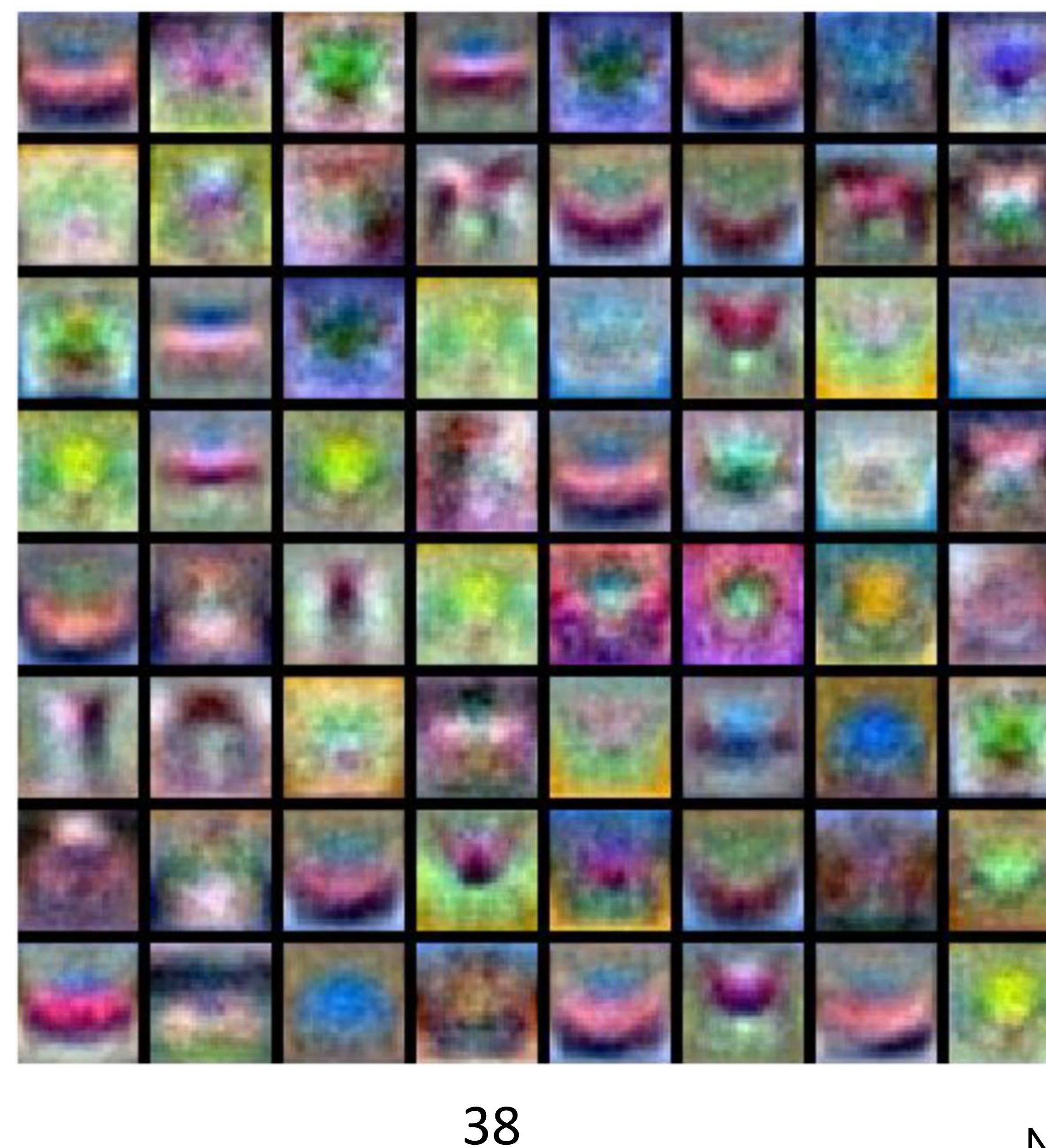
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

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.



MLP: Bank of whole-image templates

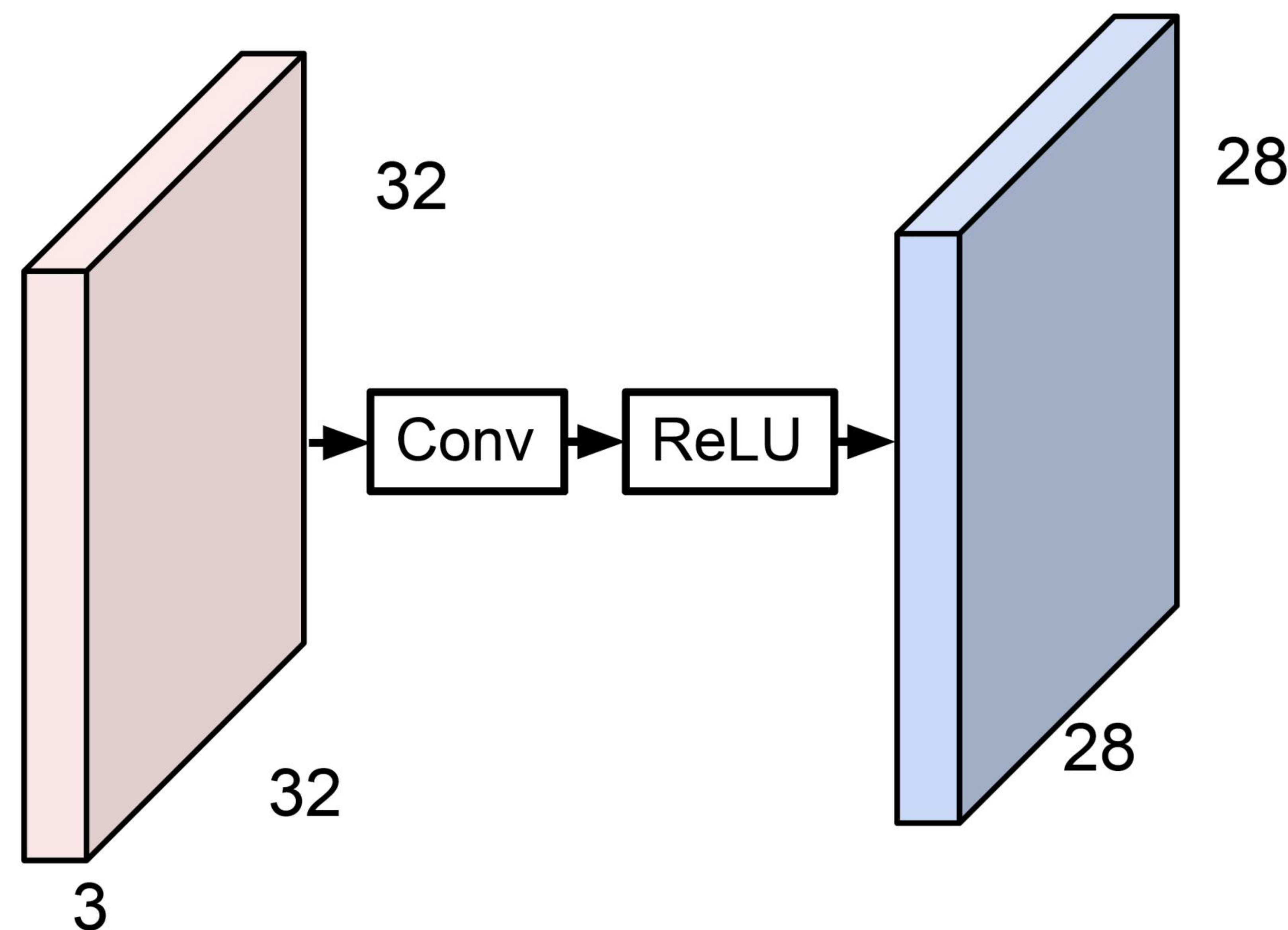


Co-financed by the European Union
Connecting Europe Facility

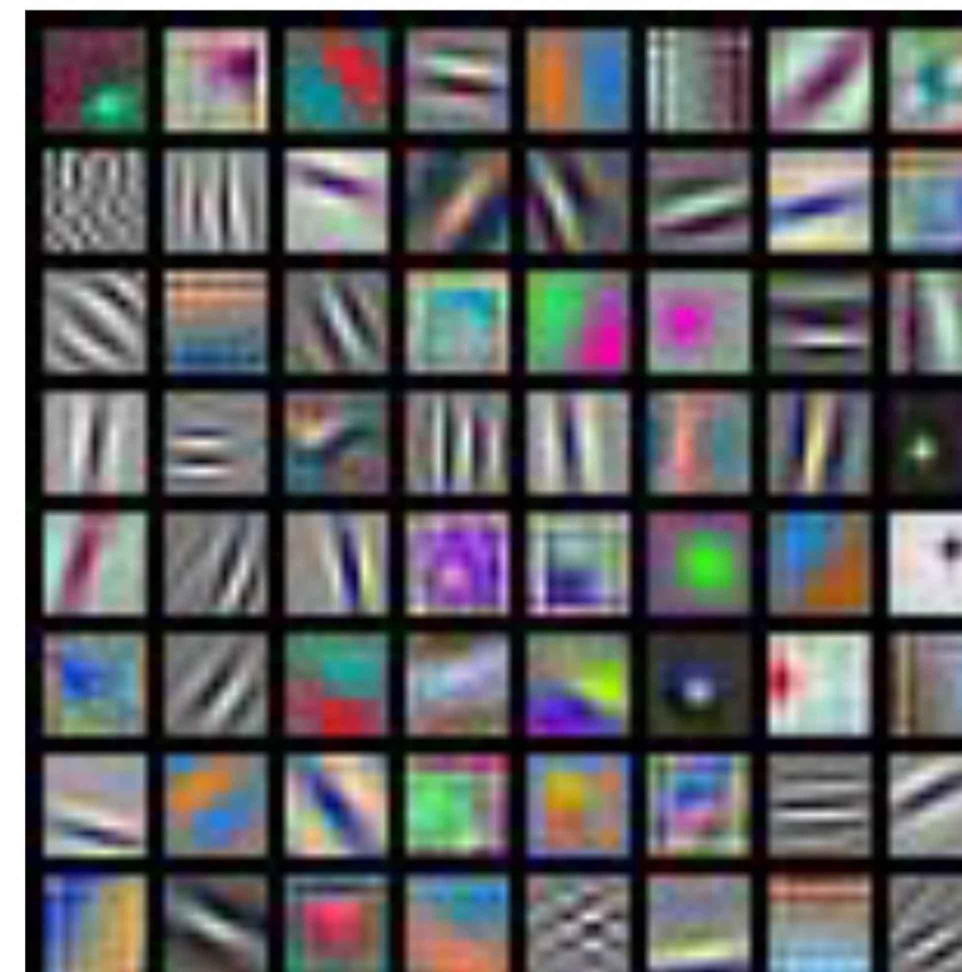
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

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.



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.

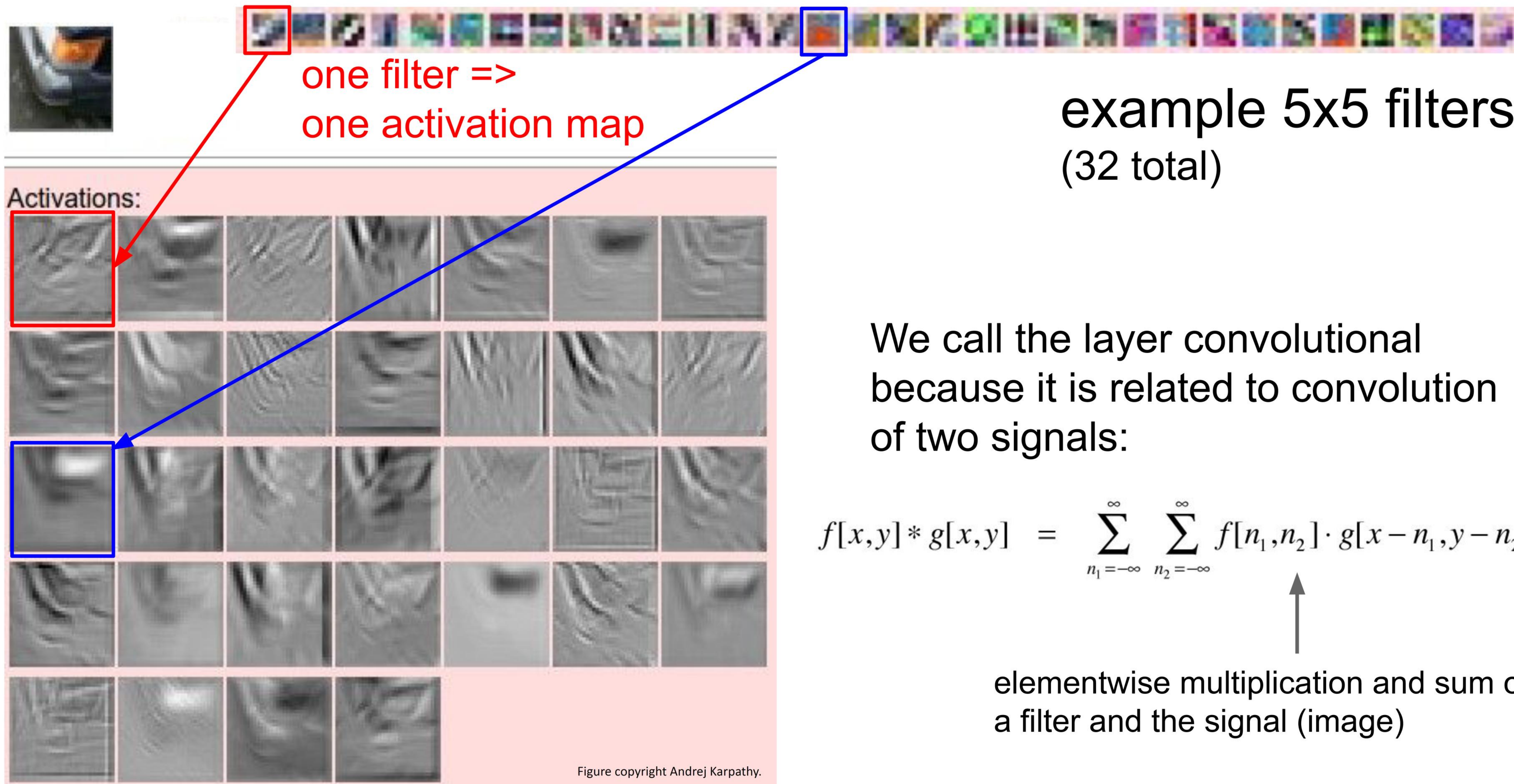
The number of filters in the convolutional layer determines the number of activation maps in the output. Each filter produces a separate activation map, which represents a different aspect or feature of the input. For example, in an image recognition task, one filter might detect horizontal edges, while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

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?



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.

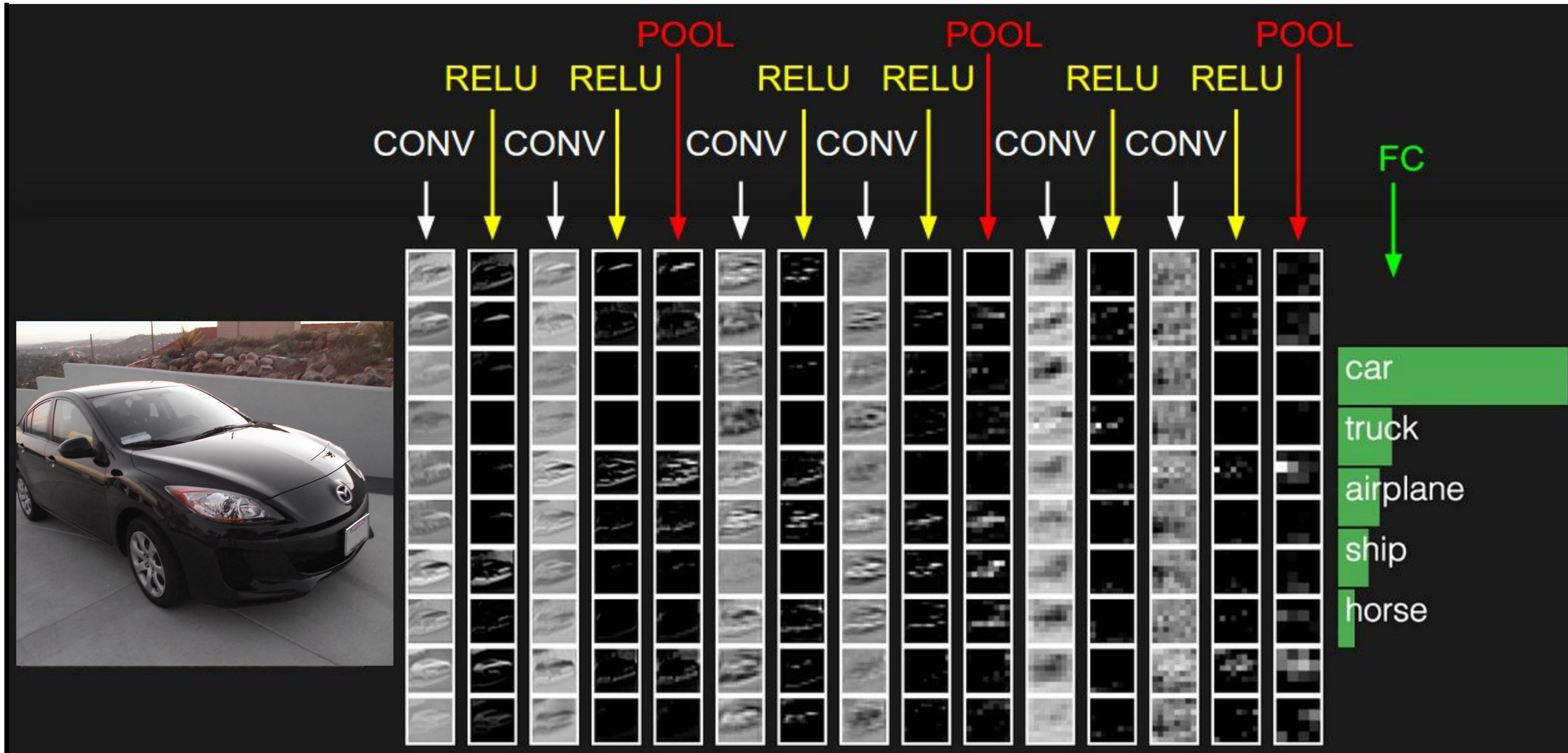
A convolutional layer can have multiple filters, each producing its own feature map.

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.

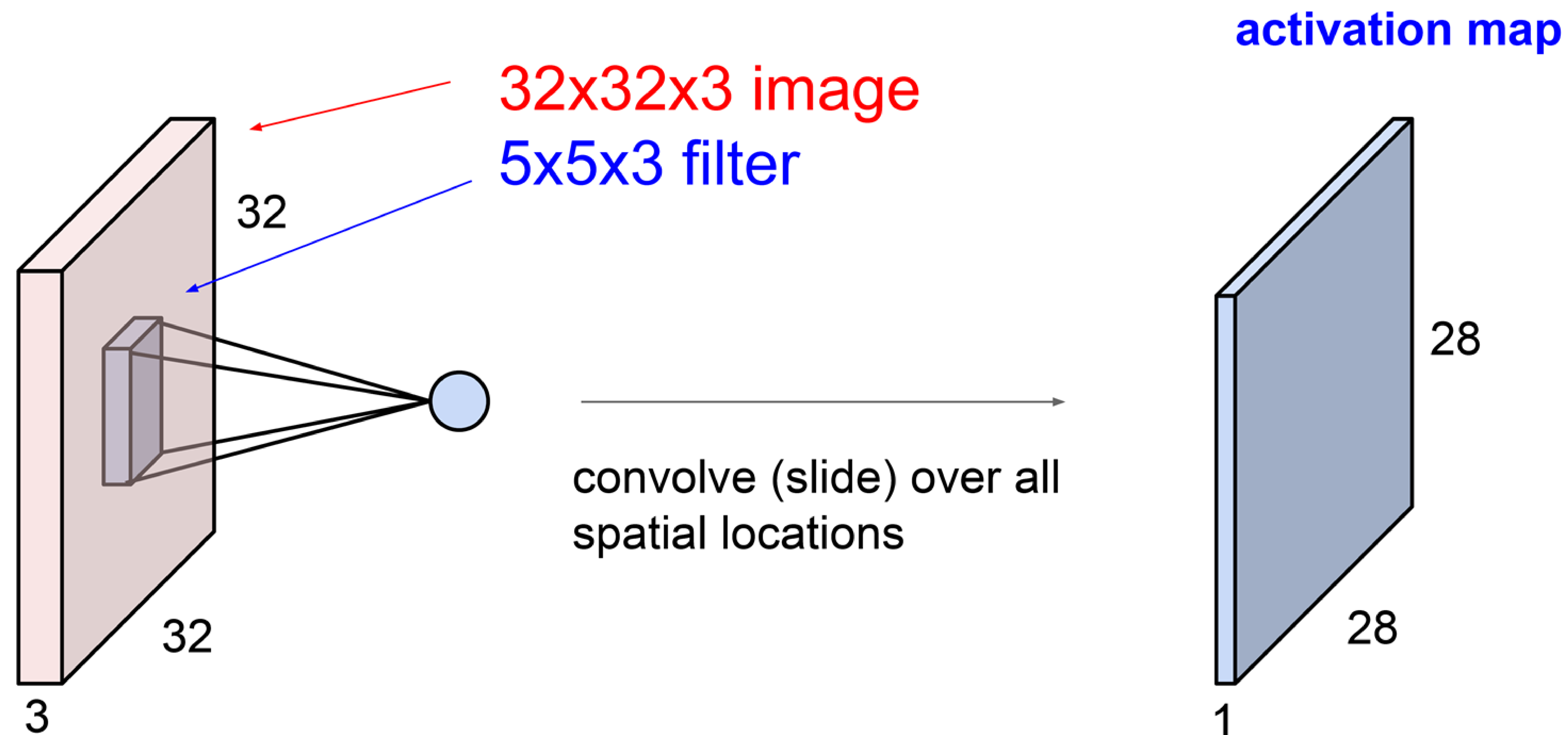
Co-financed by the European Union
Connecting Europe Facility

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 Neural Networks: *A closer look at spatial dimensions*



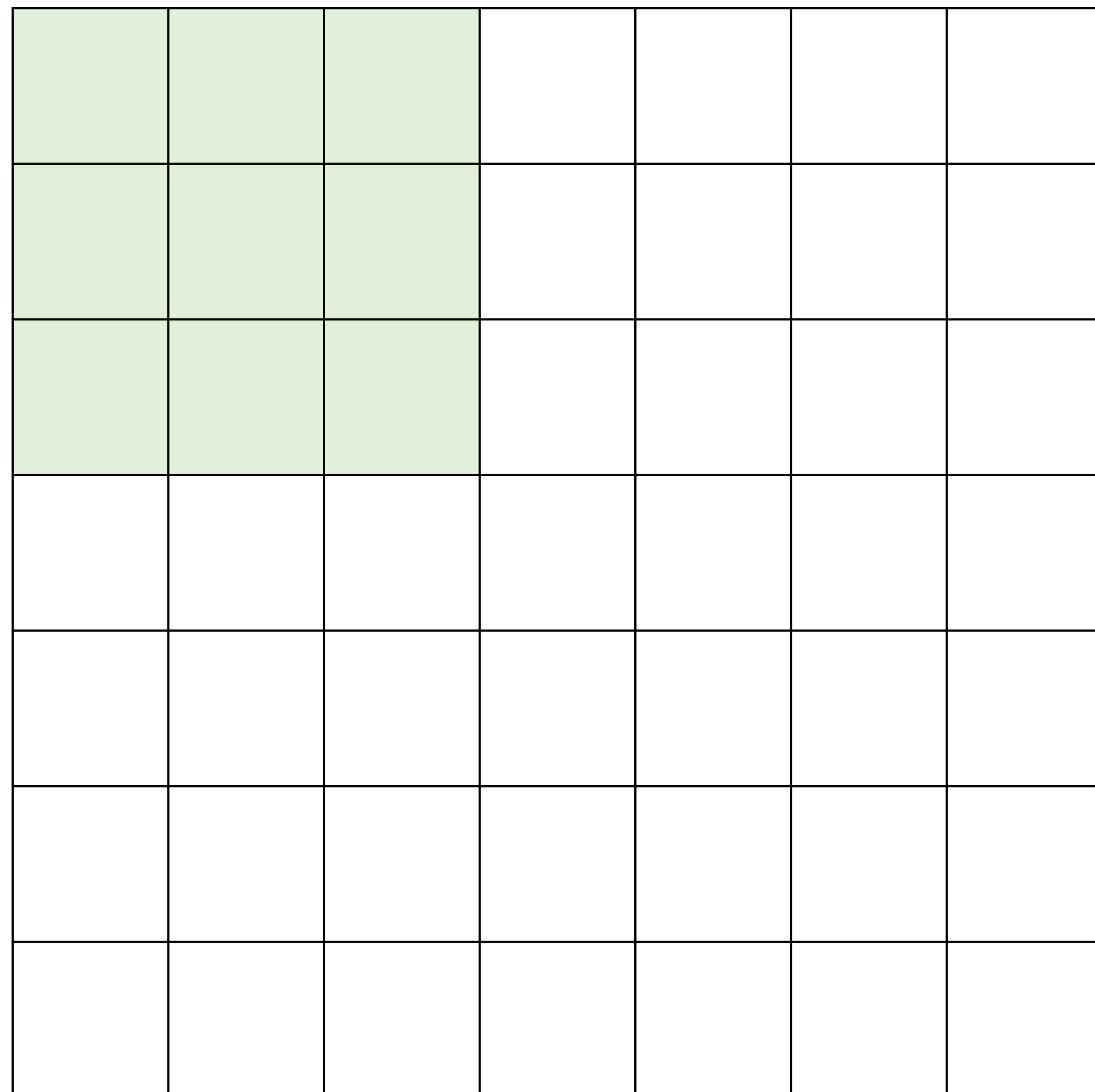
activation map

28

28

1

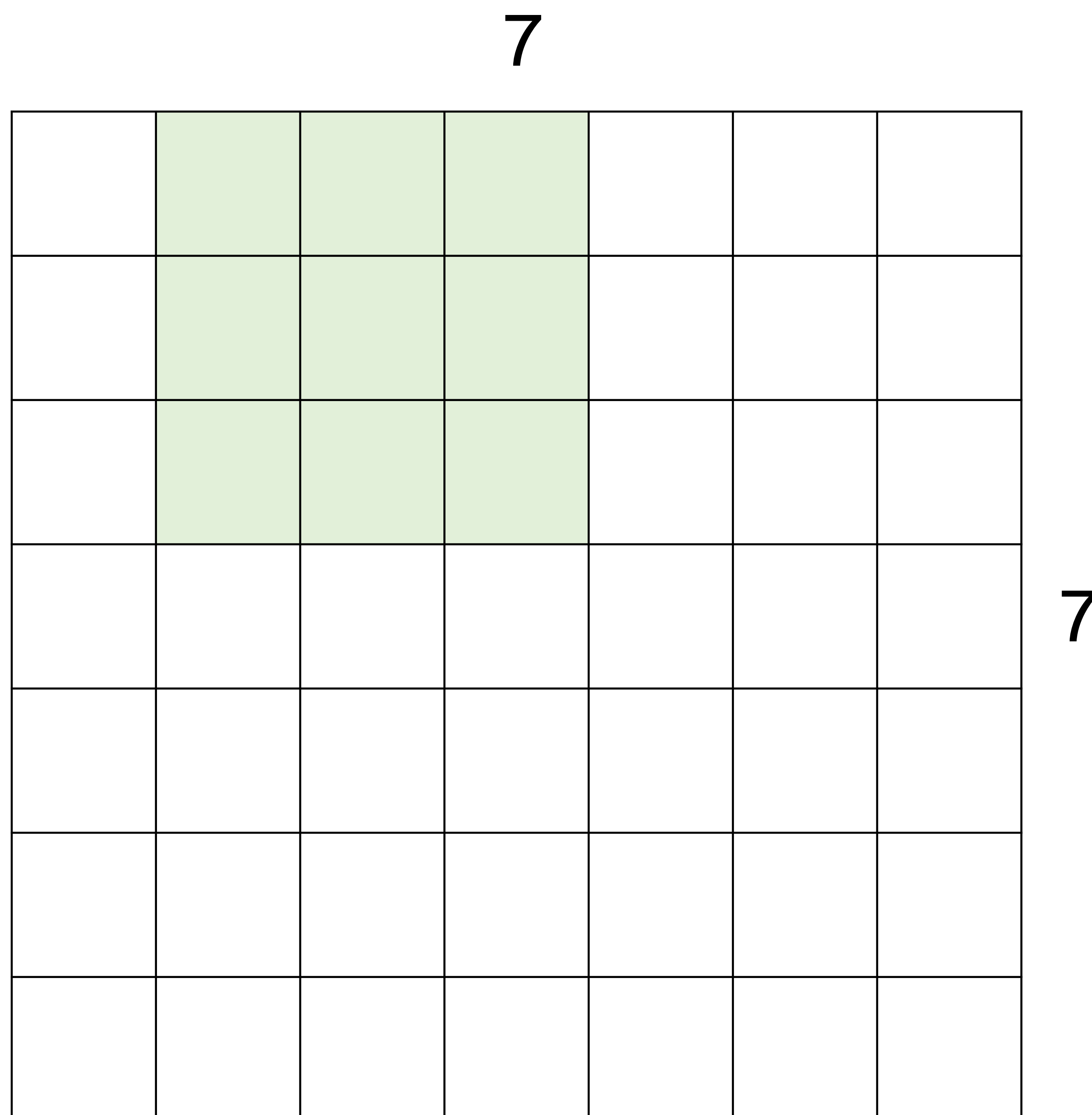
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

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

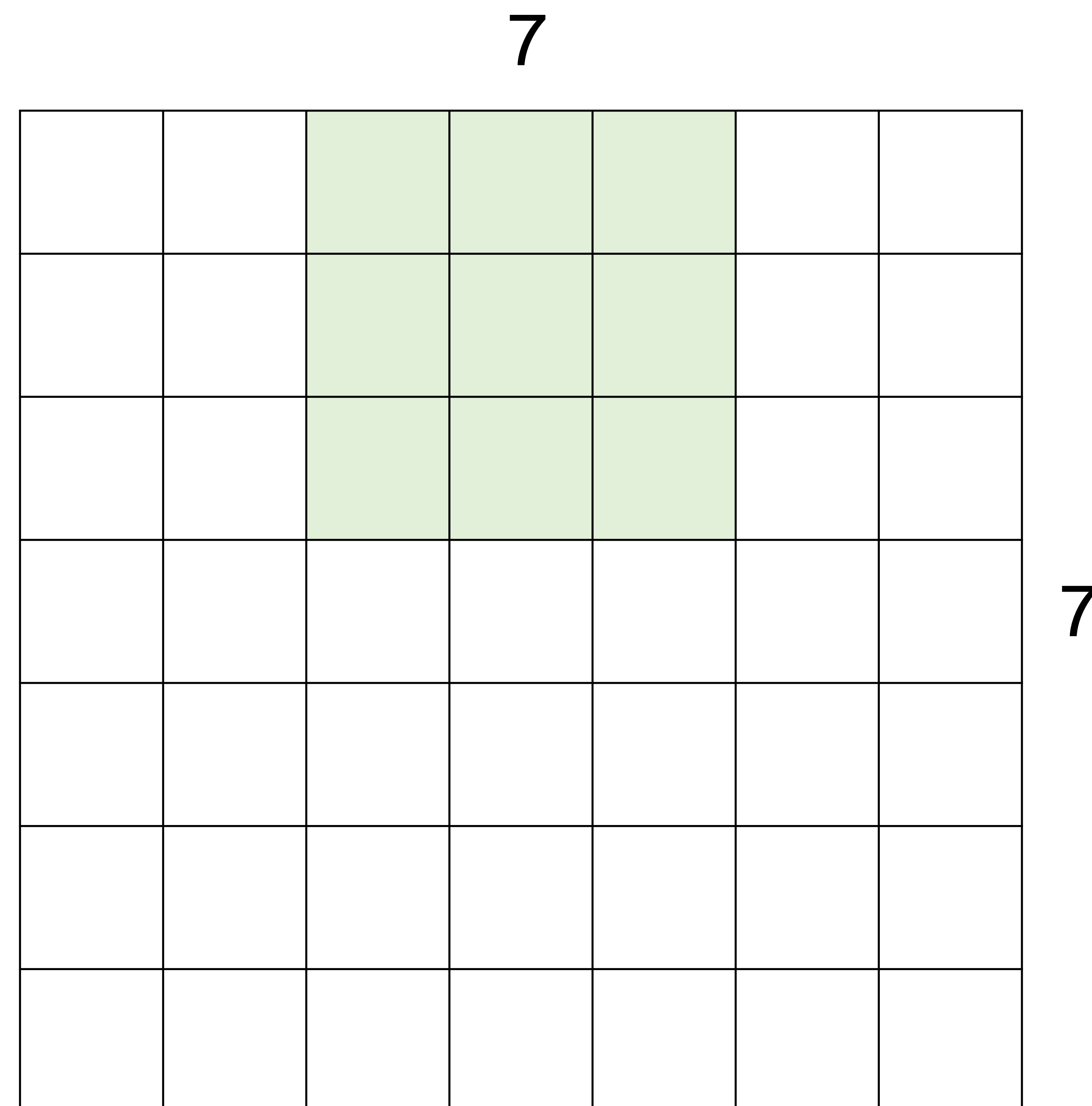
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

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

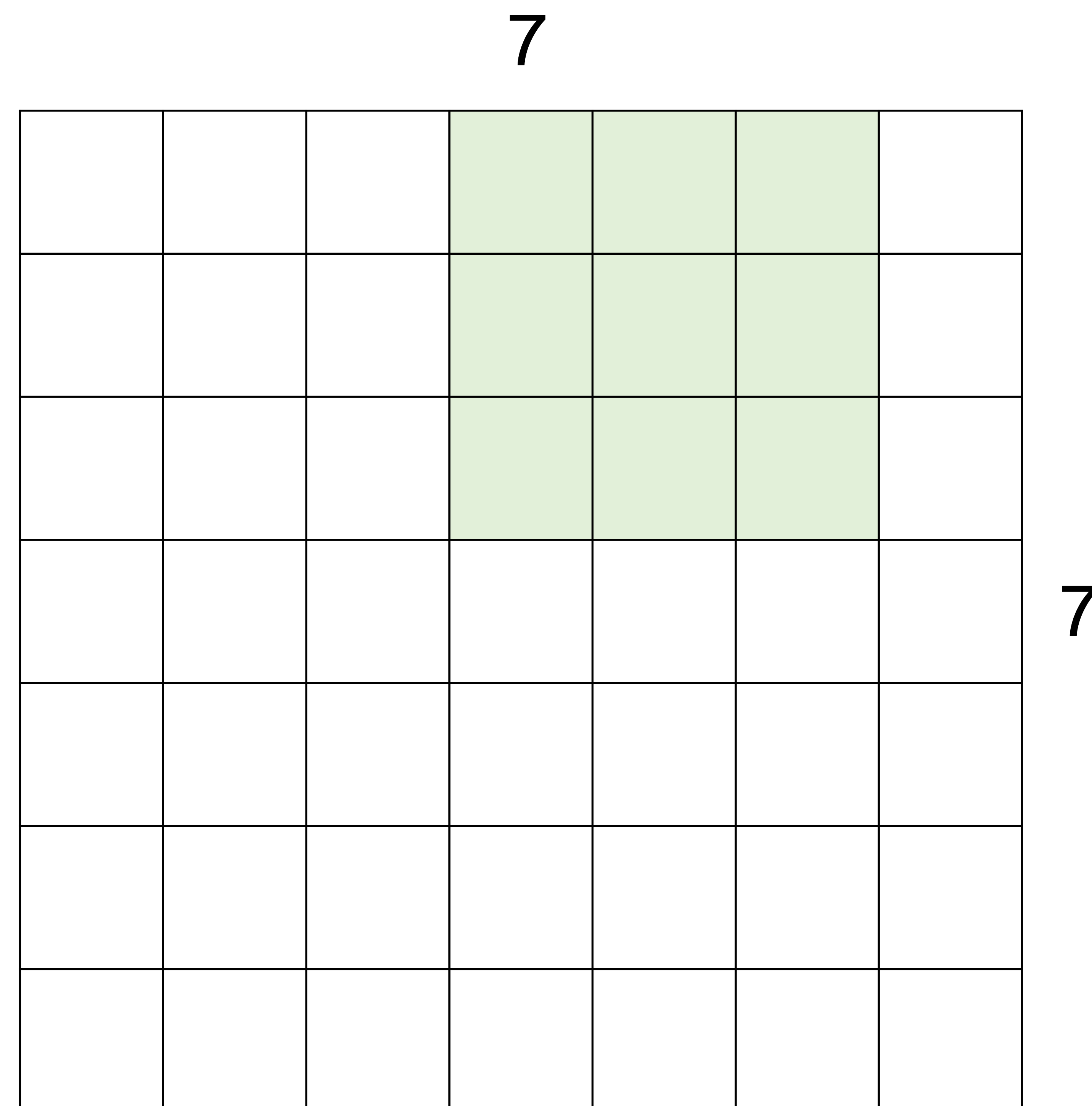
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

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

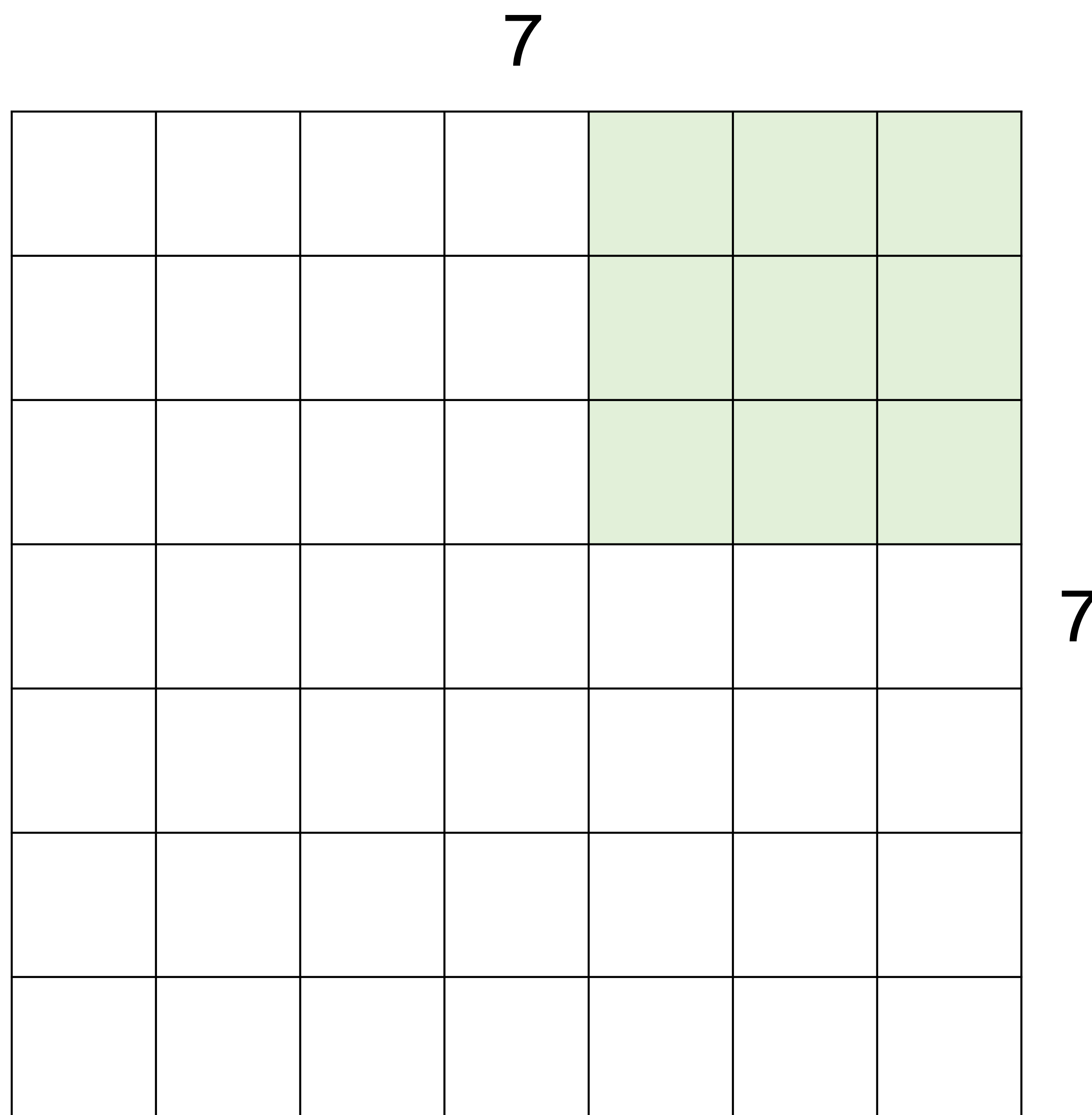
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

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

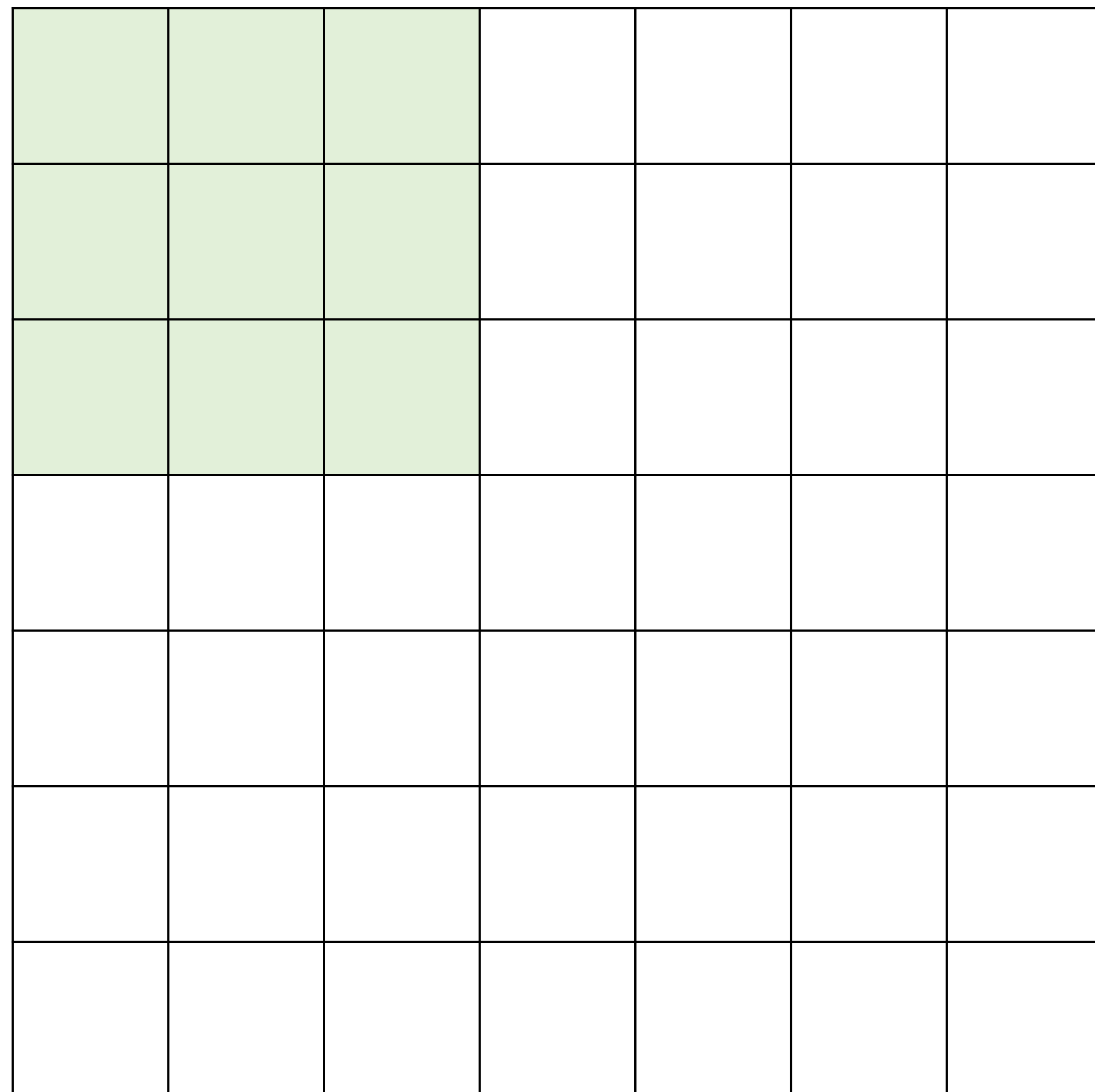
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

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

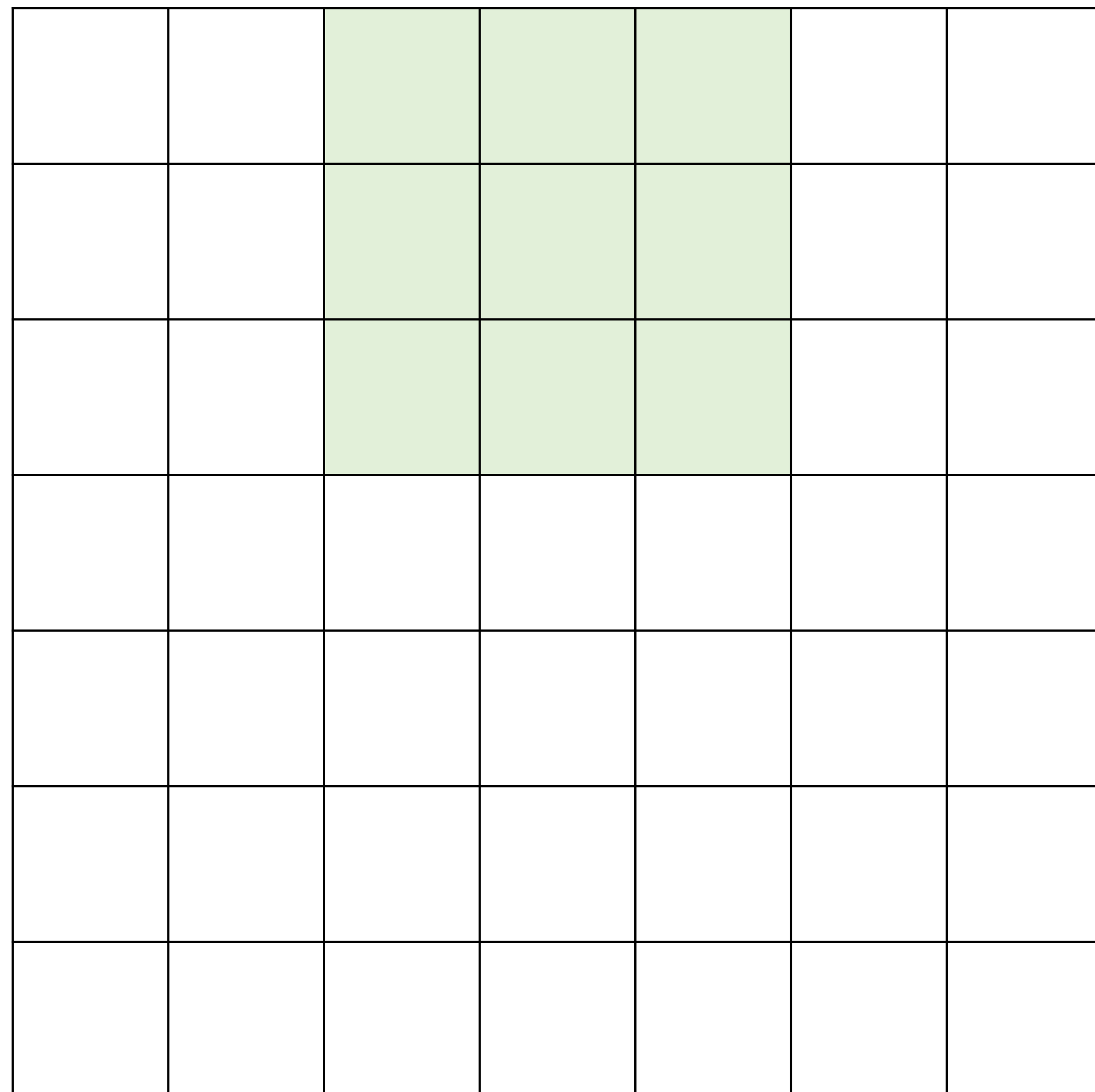


7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

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

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

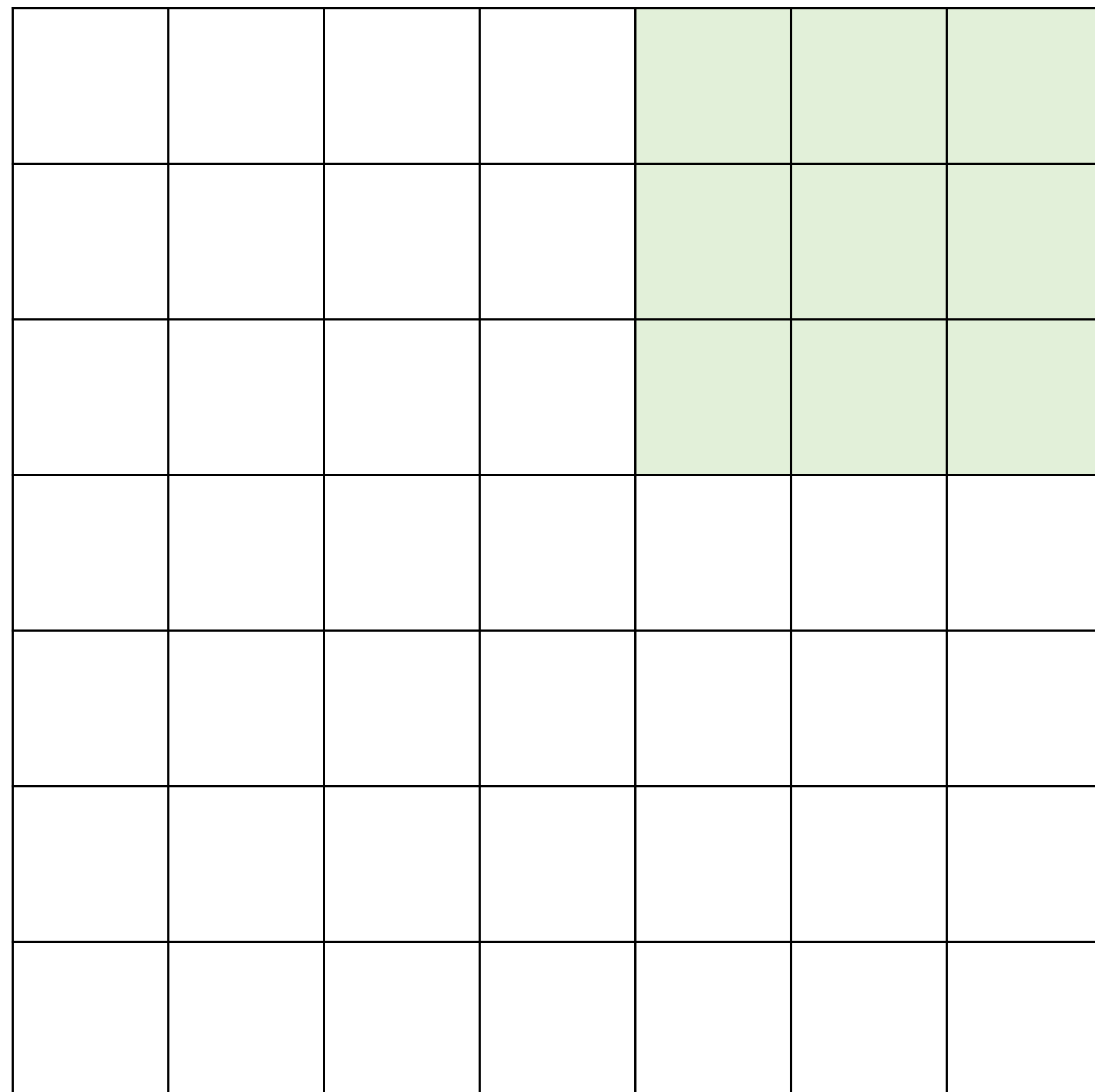


7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

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

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

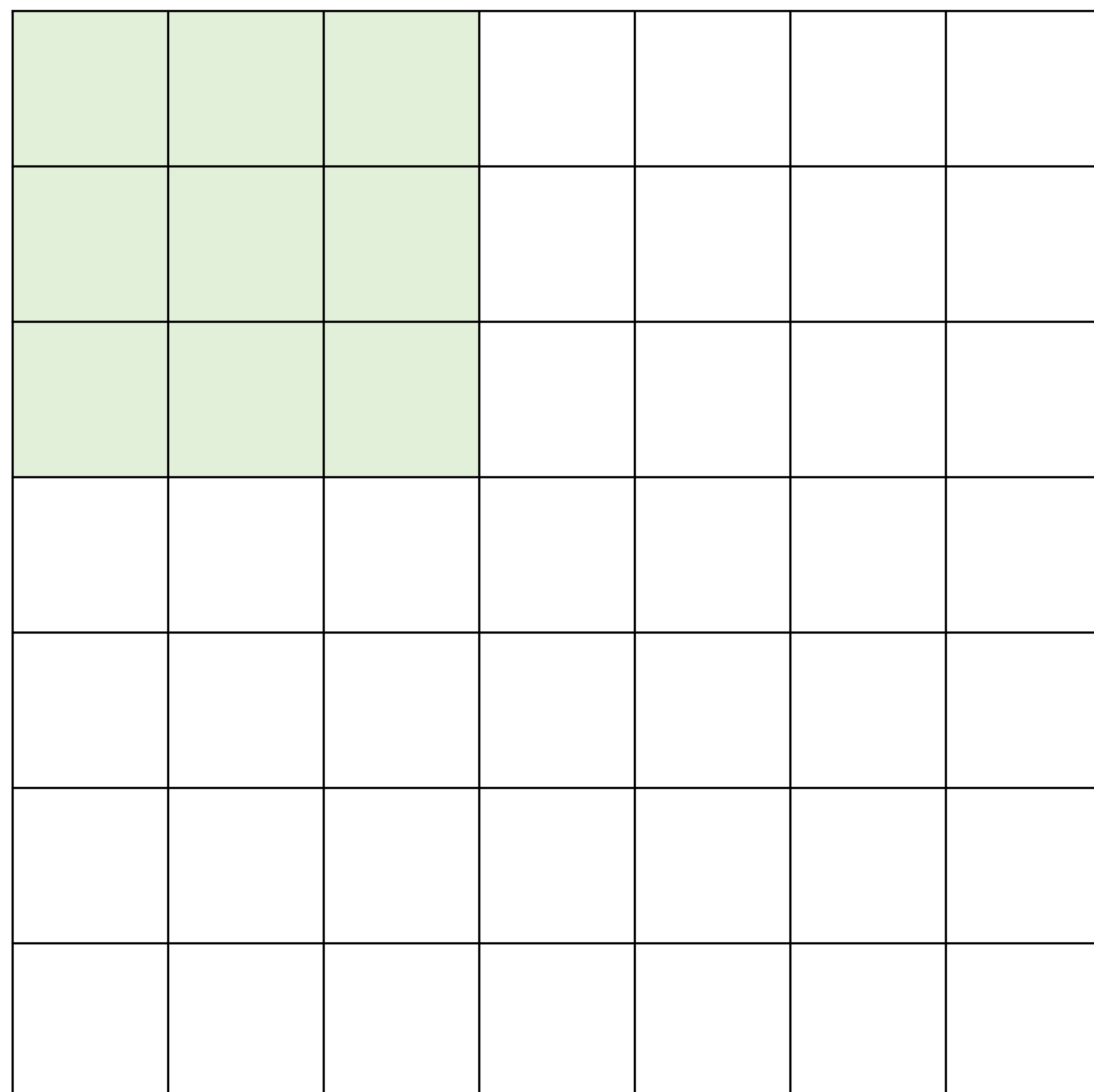


7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

=> 3x3 output!

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

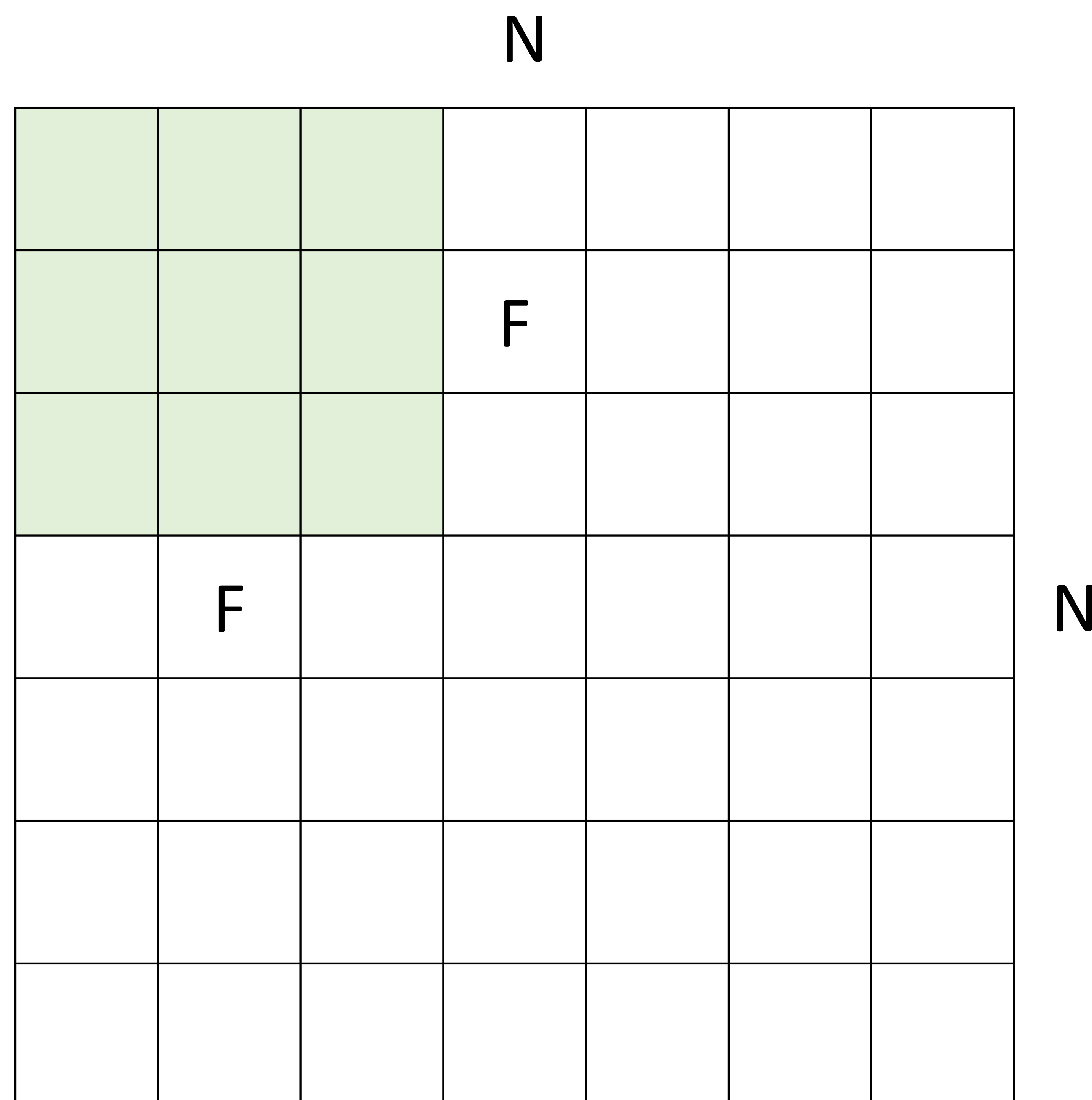
Convolutional Neural Networks: *A closer look at spatial dimensions*



7x7 input (spatially)
assume 3x3 filter
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

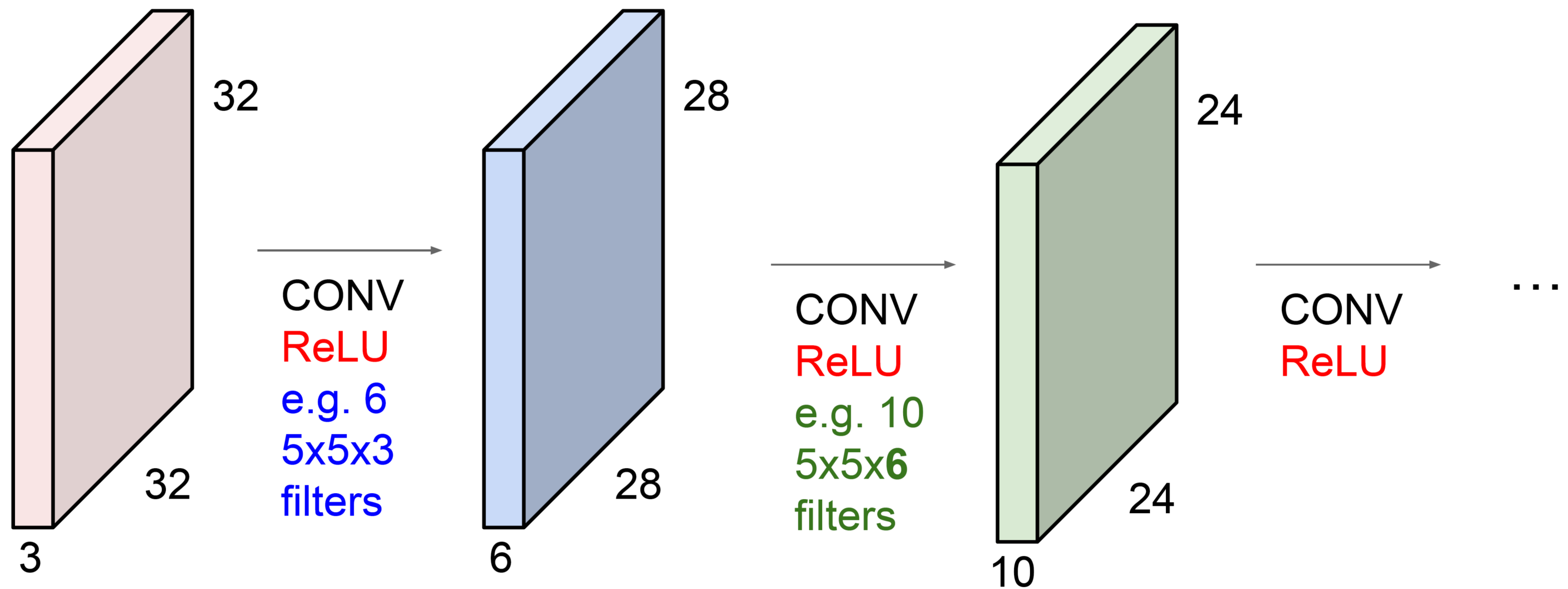
(recall:)
(N - F) / stride + 1

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

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.



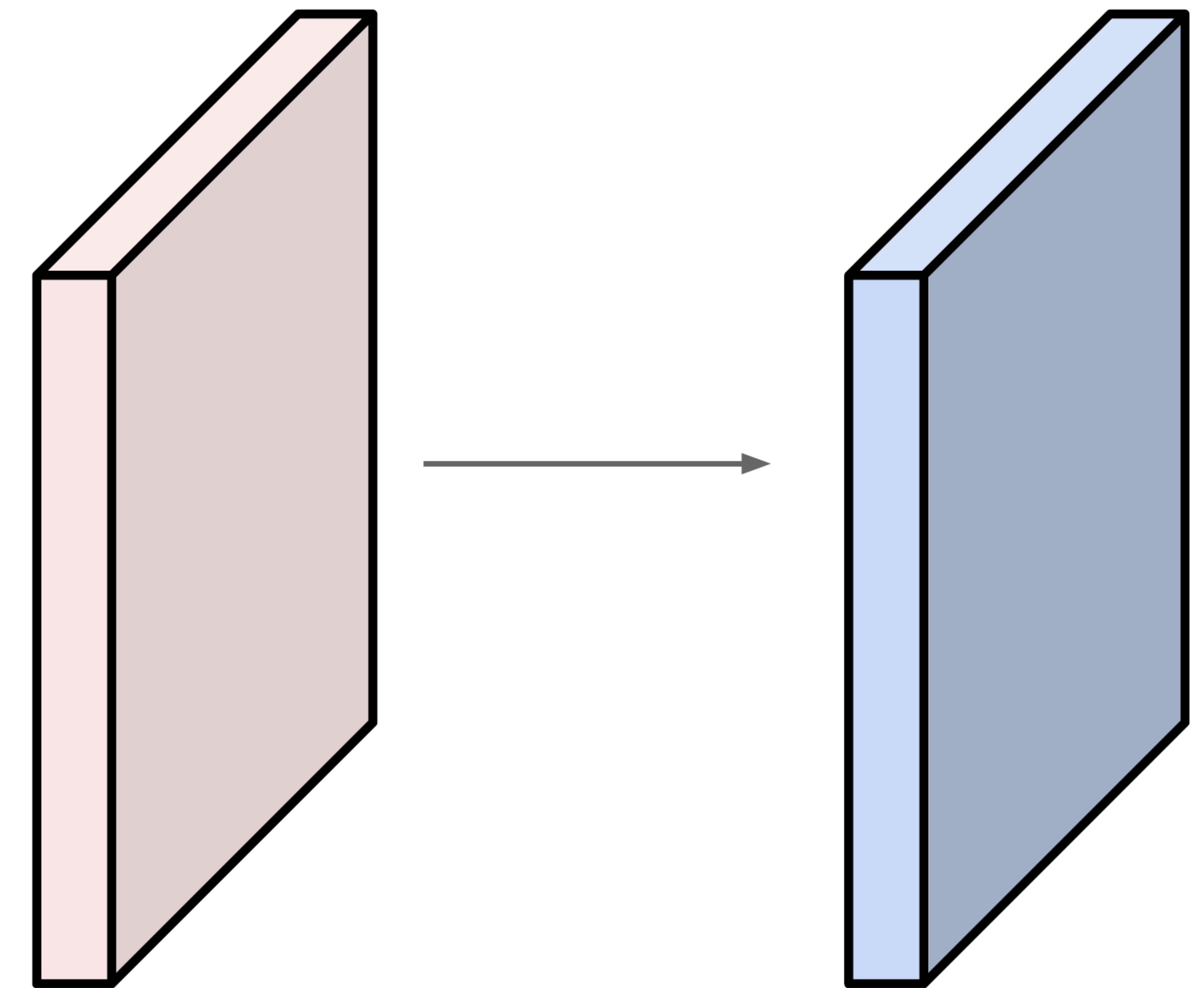
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

ConvNet: *Example*

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

What is the output volume size: ?



ConvNet: *Example*

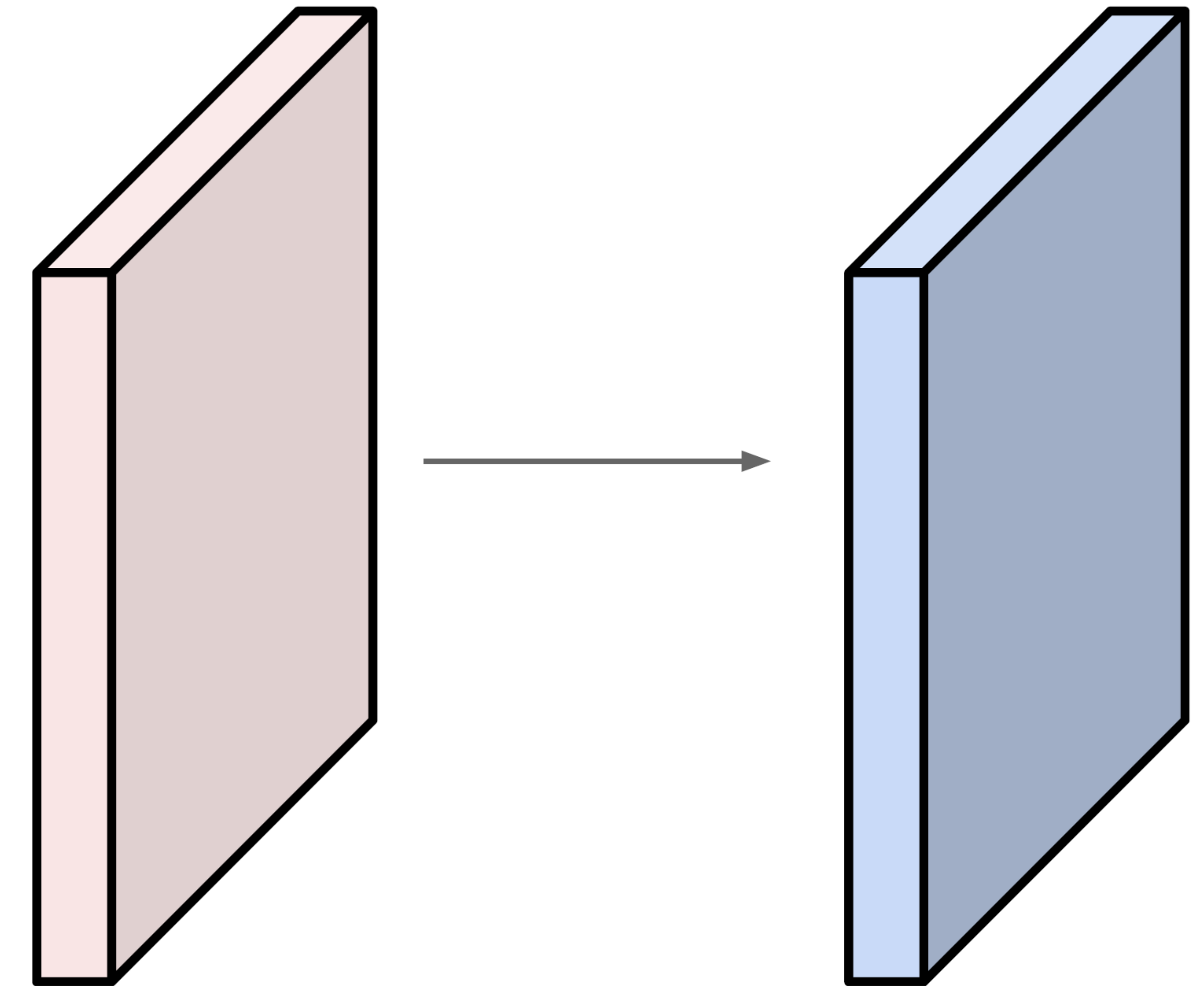
Input volume: **32x32**x3

10 5x5 filters with stride 1, pad 2

Output volume size:

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

32x32x10



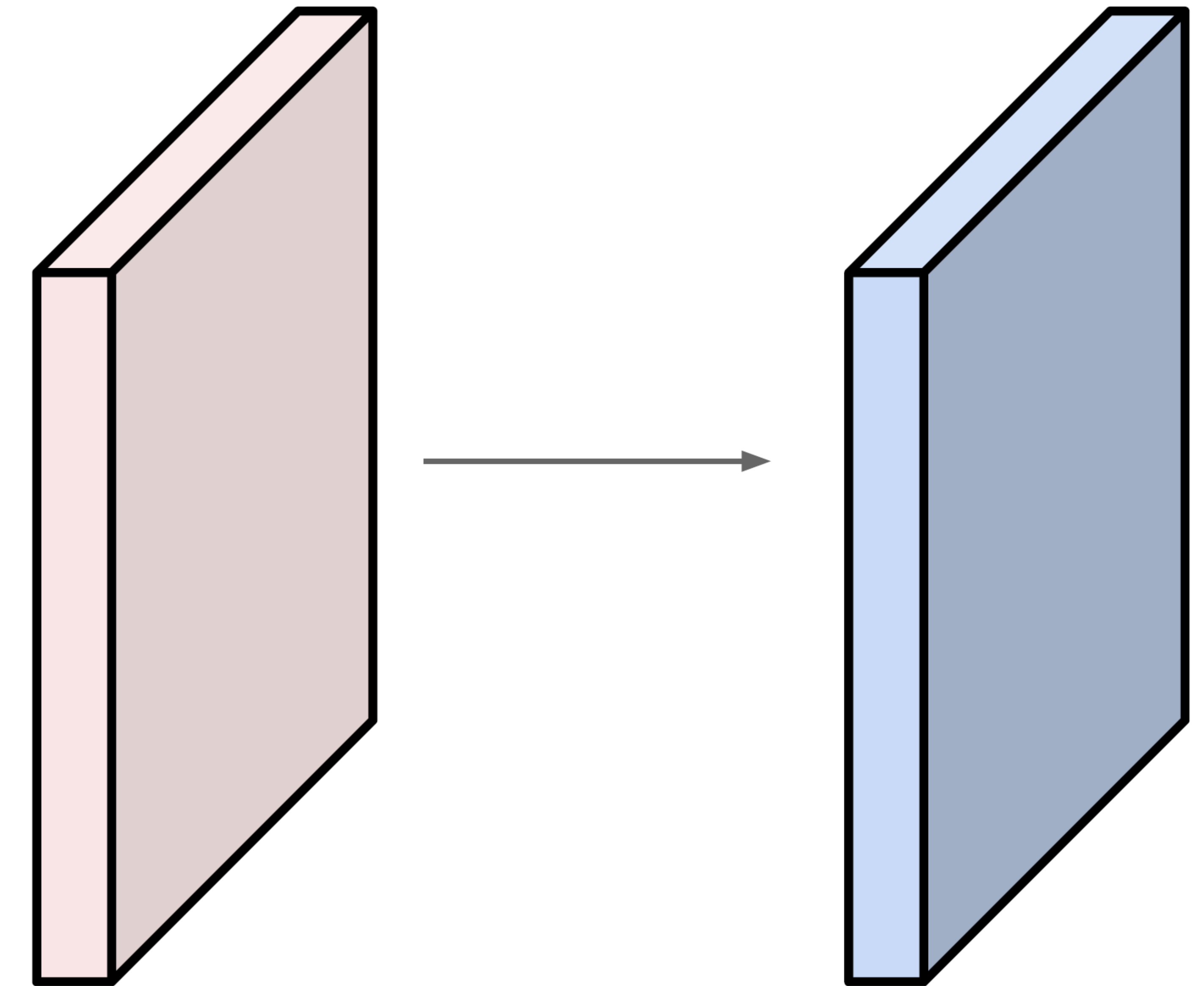
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

ConvNet: *Example*

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

What is the number of parameters in this 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

ConvNet: *Example*

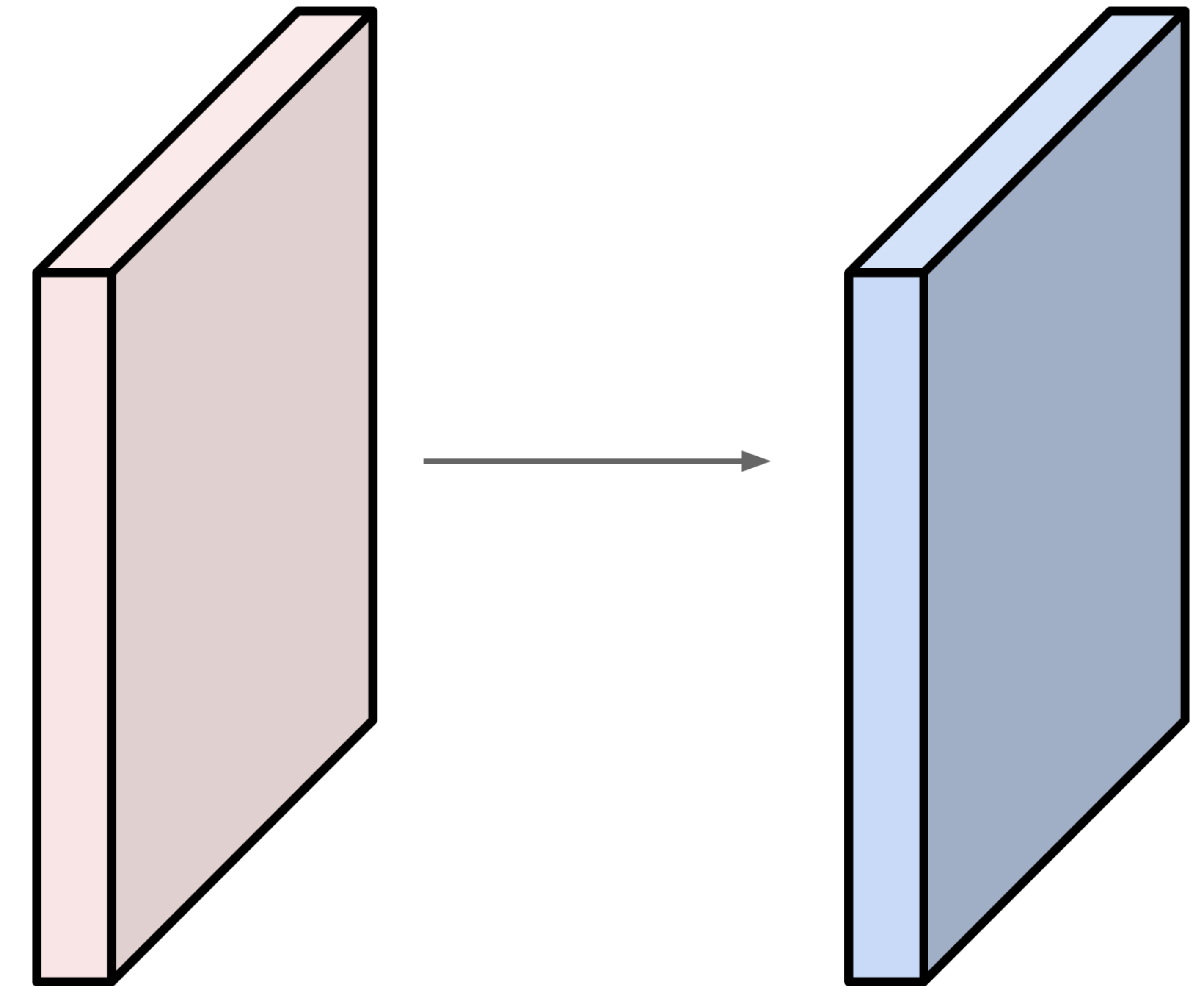
Input volume: **32x32**x3

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



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

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.

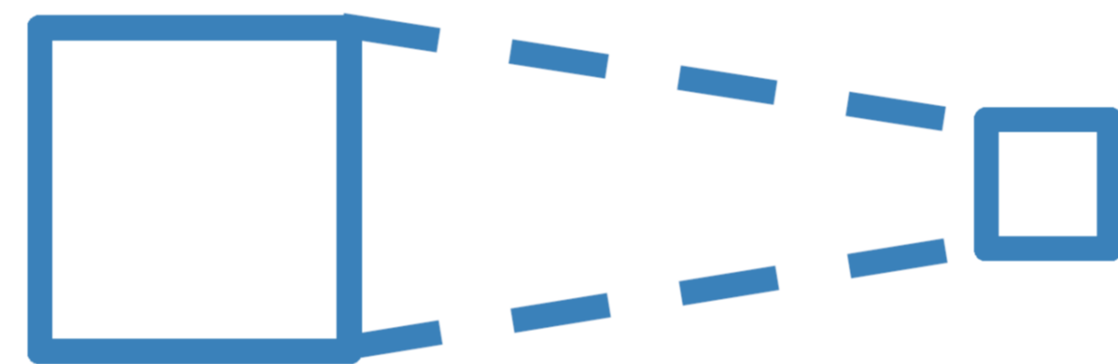
Overall, the choice of receptive field size depends on the specific requirements of the task at hand and the resources available for training the network. Balancing the trade-off between model complexity, overfitting, and computational cost is an important consideration when designing ConvNets with appropriate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

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

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

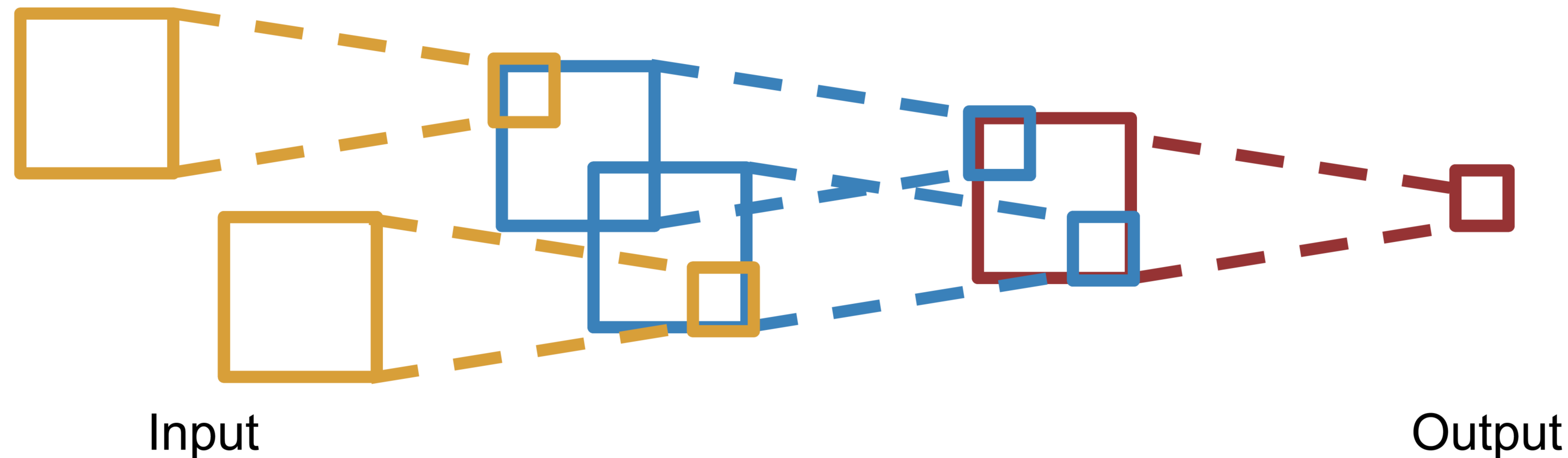


Input Output

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)

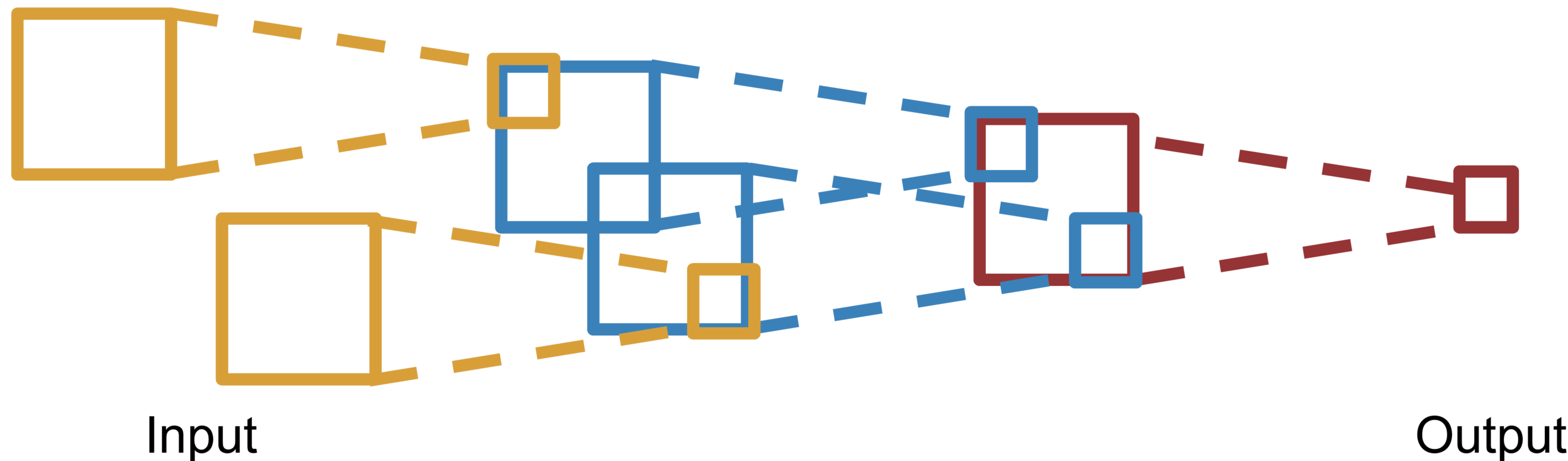


Be careful – ”receptive field in the input” vs. “receptive field in the previous layer”

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

Receptive Fields

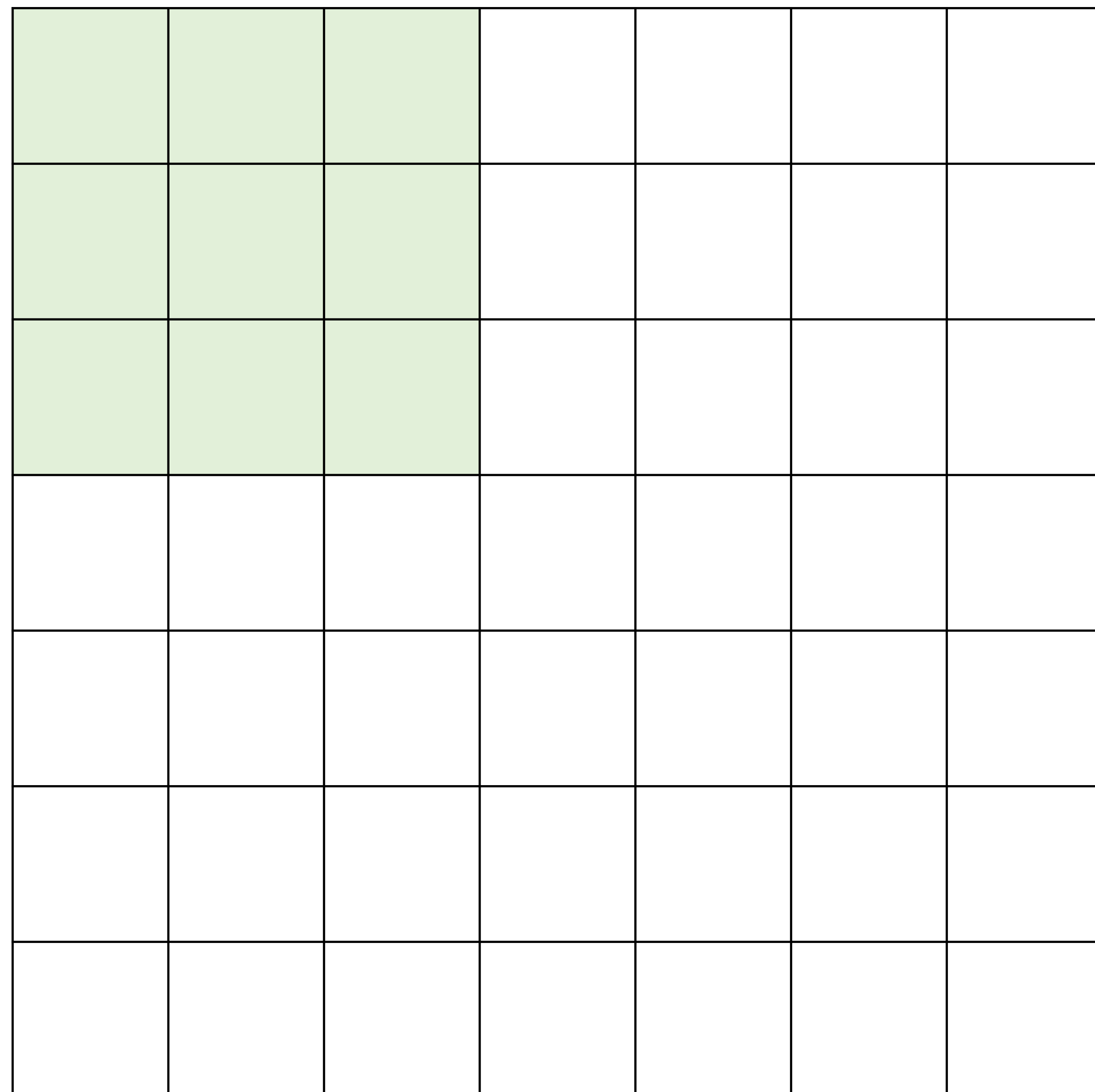
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Solution: Downsample inside the network

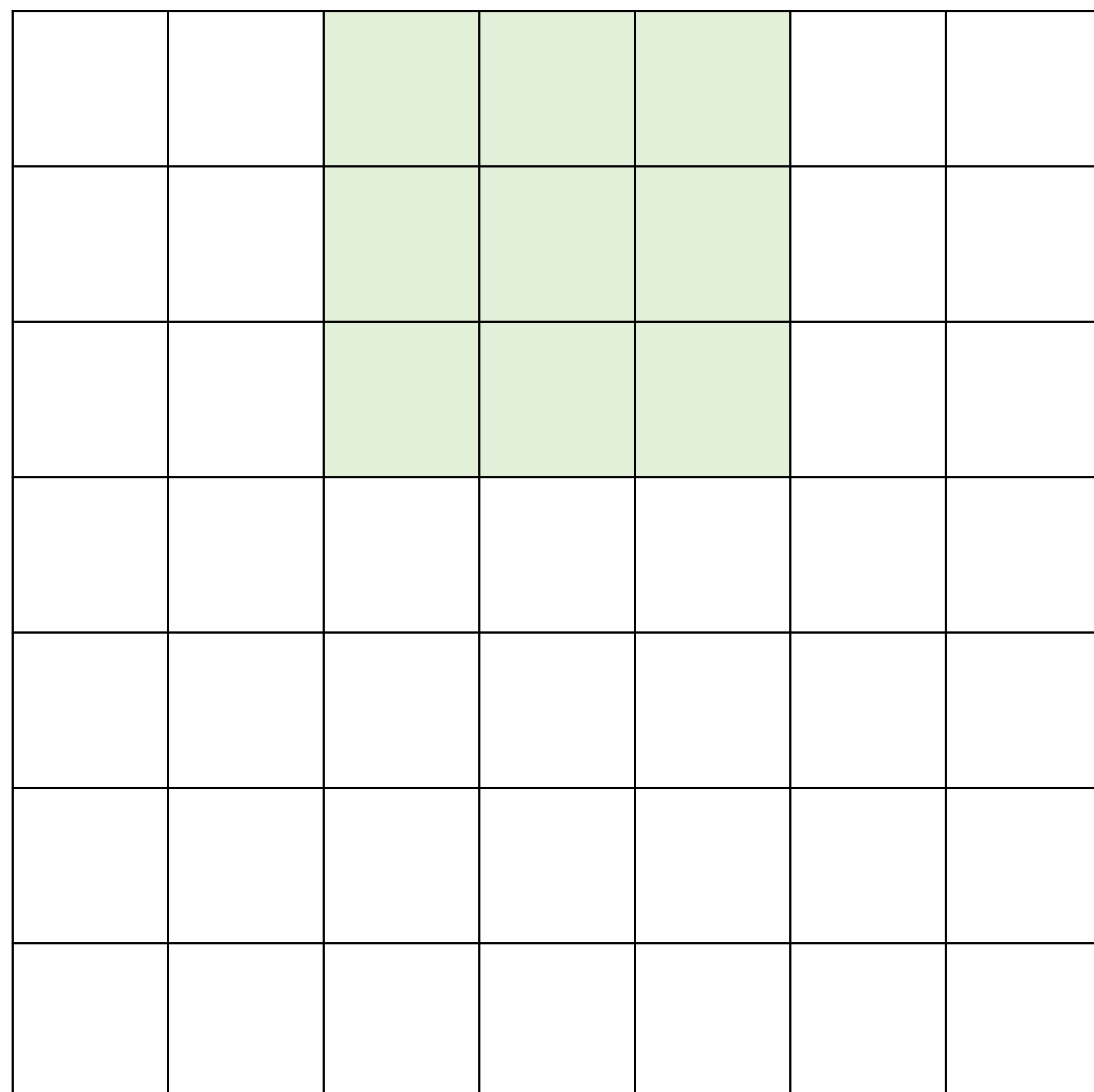
Problem: For large images we need many layers for each output to “see” the whole image

Receptive Fields: *Solution - Strided Convolution*



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

Receptive Fields: *Solution - Strided Convolution*



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

=> 3x3 output!

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:

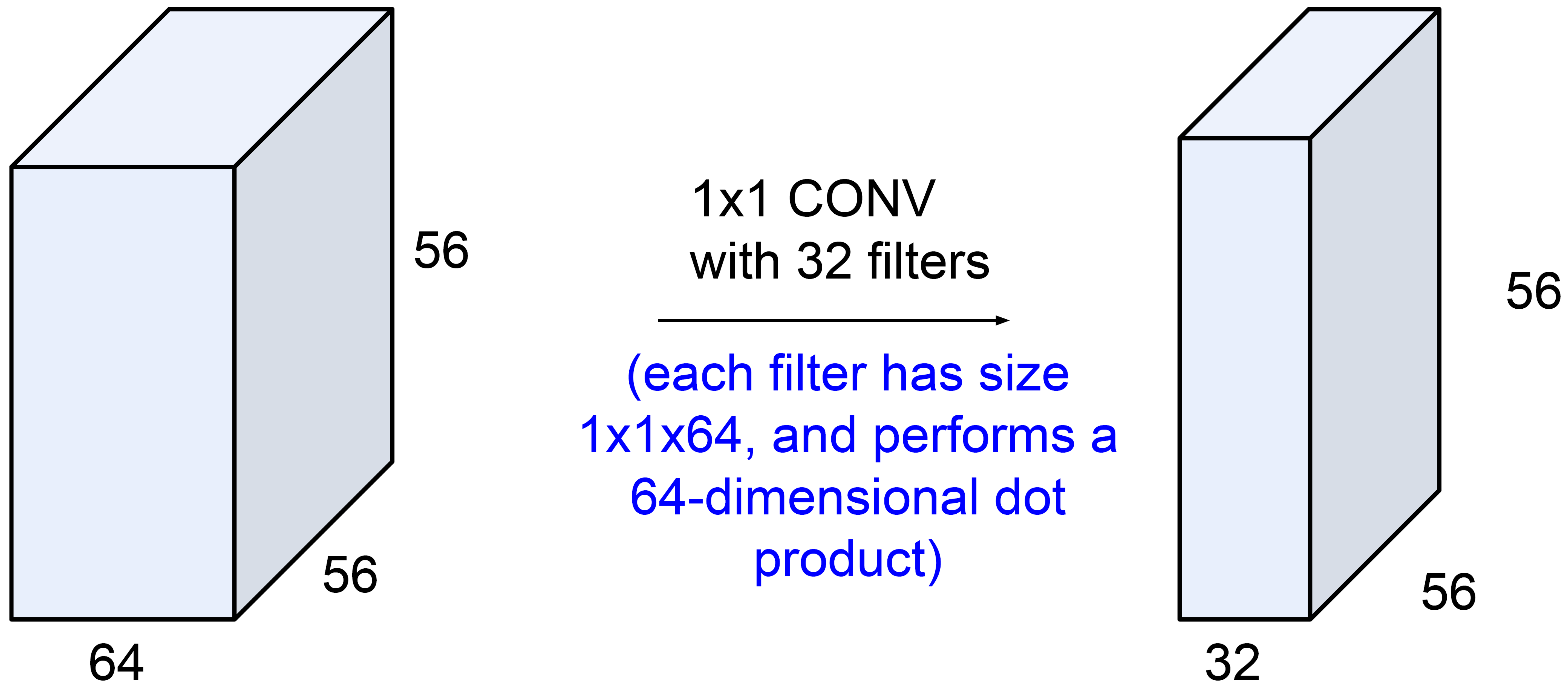
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

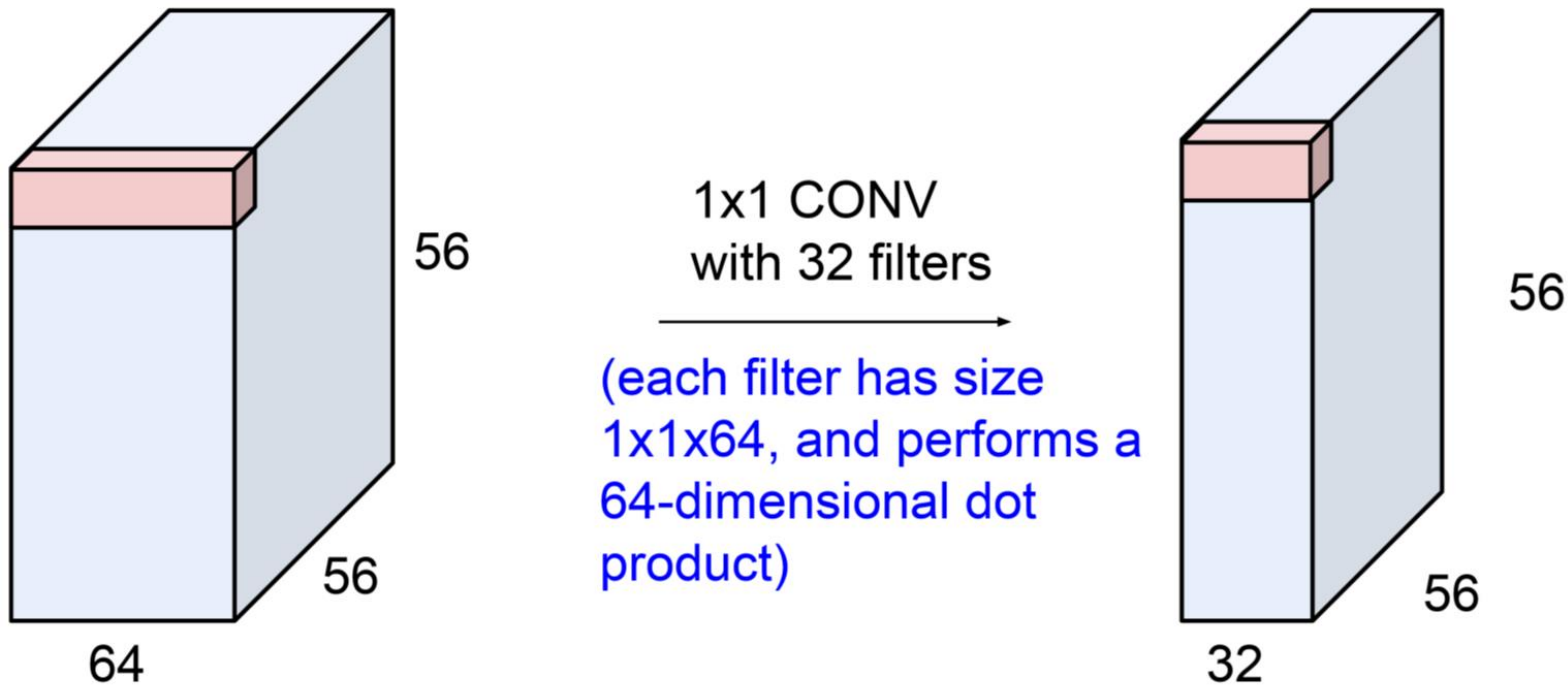
Co-financed by the European Union
Connecting Europe Facility

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

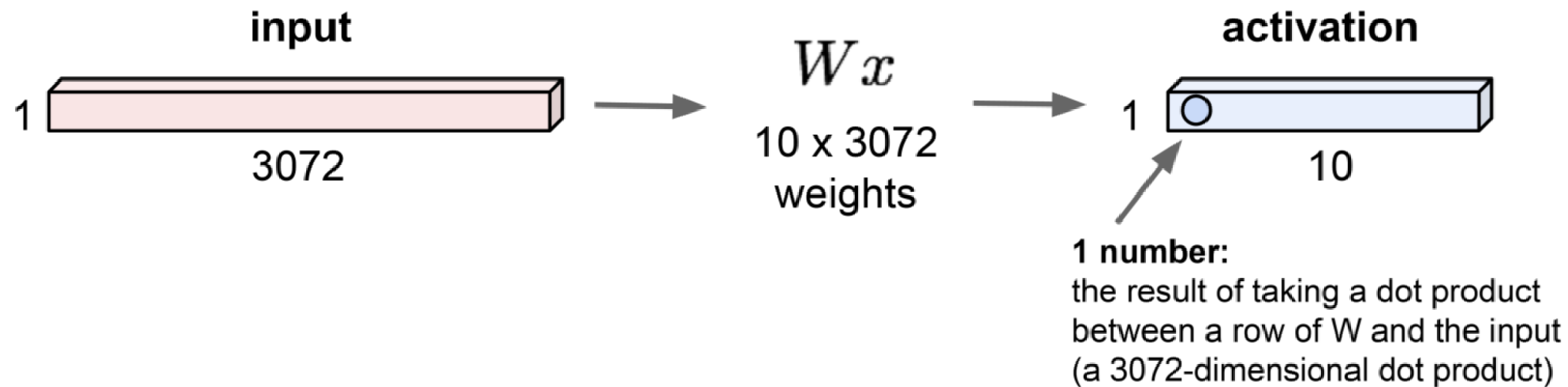


1x1 convolution layers make perfect sense



Reminder: *Fully Connected Layer*

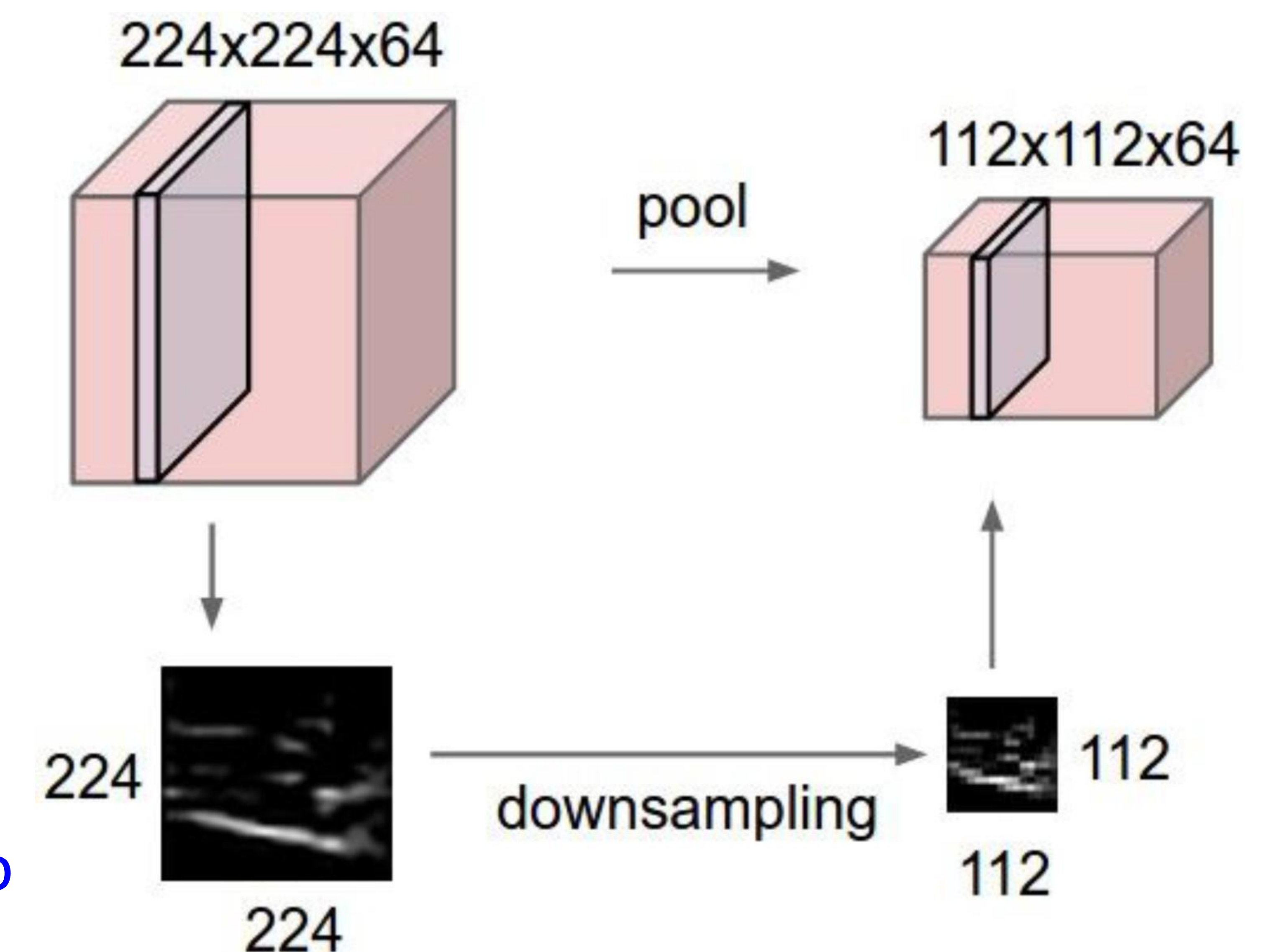
Each neuron looks at the full input volume



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.



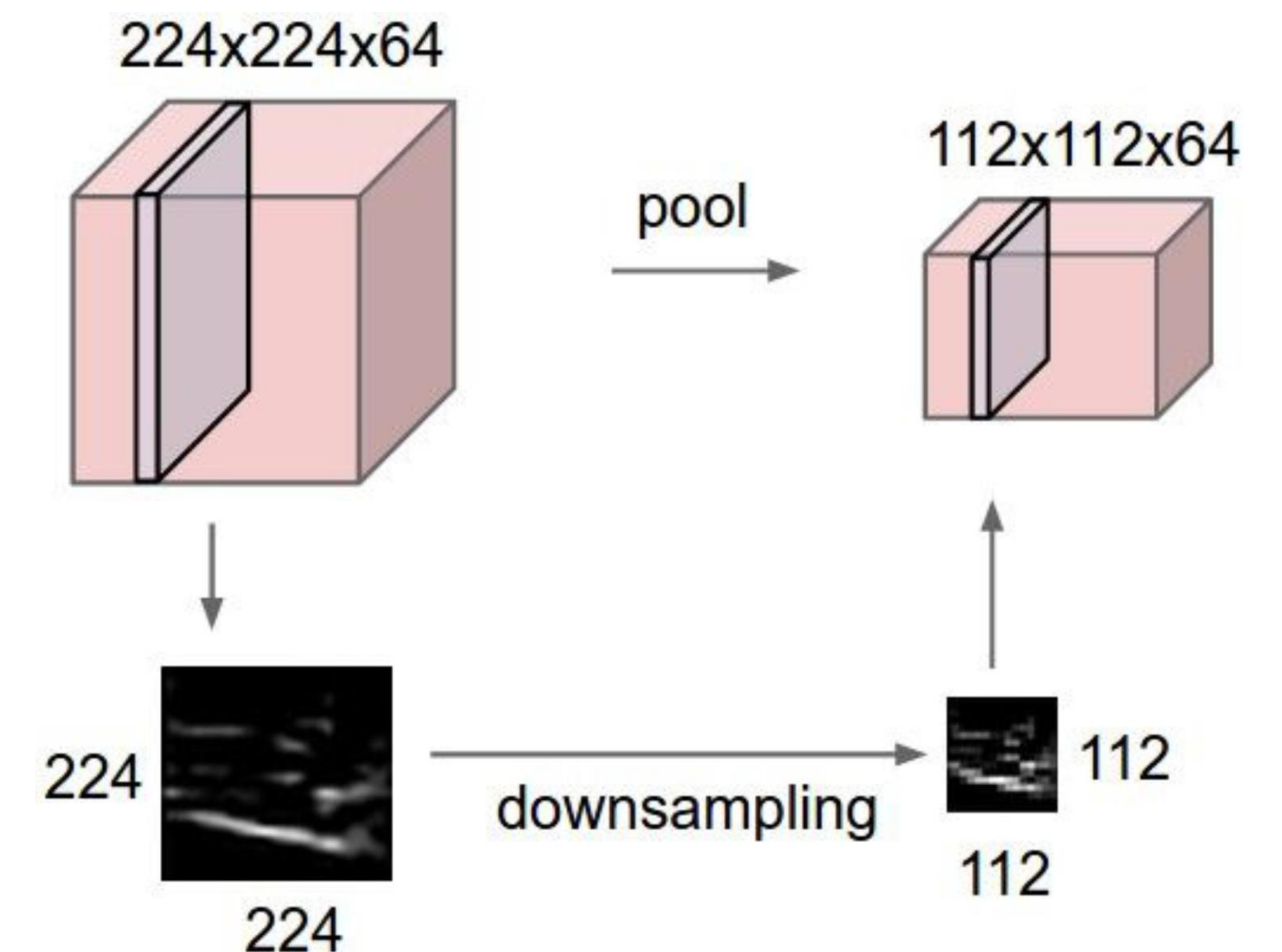
The primary purpose of pooling layers is to help the network to become more robust to small variations in the input. This is achieved by reducing the spatial resolution of the feature maps and summarizing them in a way that retains the most important information. Pooling can also help to reduce the computational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

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: *Pooling Layer*

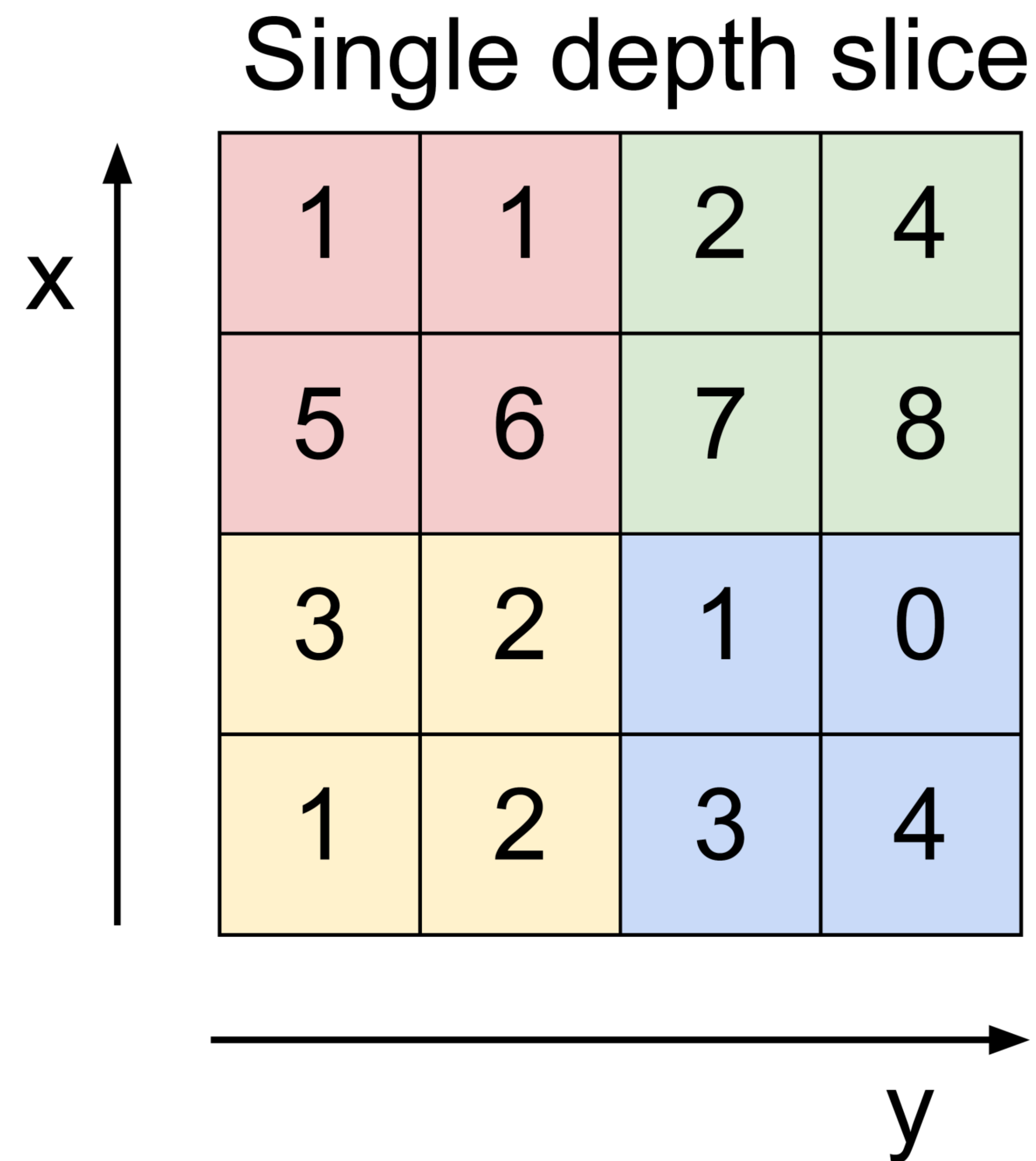
Pooling layers are a type of layer in ConvNets that are used to reduce the spatial dimensions of the input volume.

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.

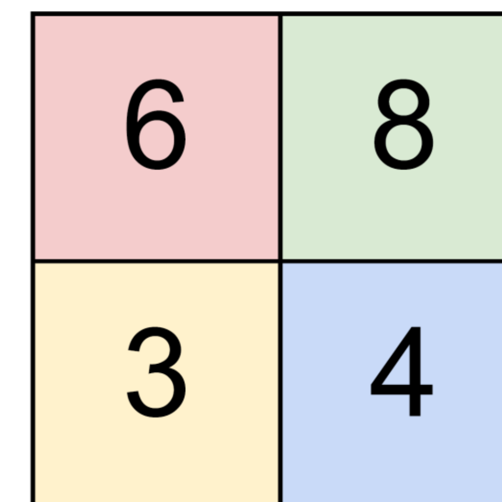


This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Pooling Layer: *Max pooling*



max pool with 2x2 filters and stride 2



- No learnable parameters
- Introduces spatial invariance

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*

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

Conv layer needs 2 hyperparameters:

- The spatial extent F
- The stride S

This will produce an output of $W_2 \times H_2 \times C$ 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

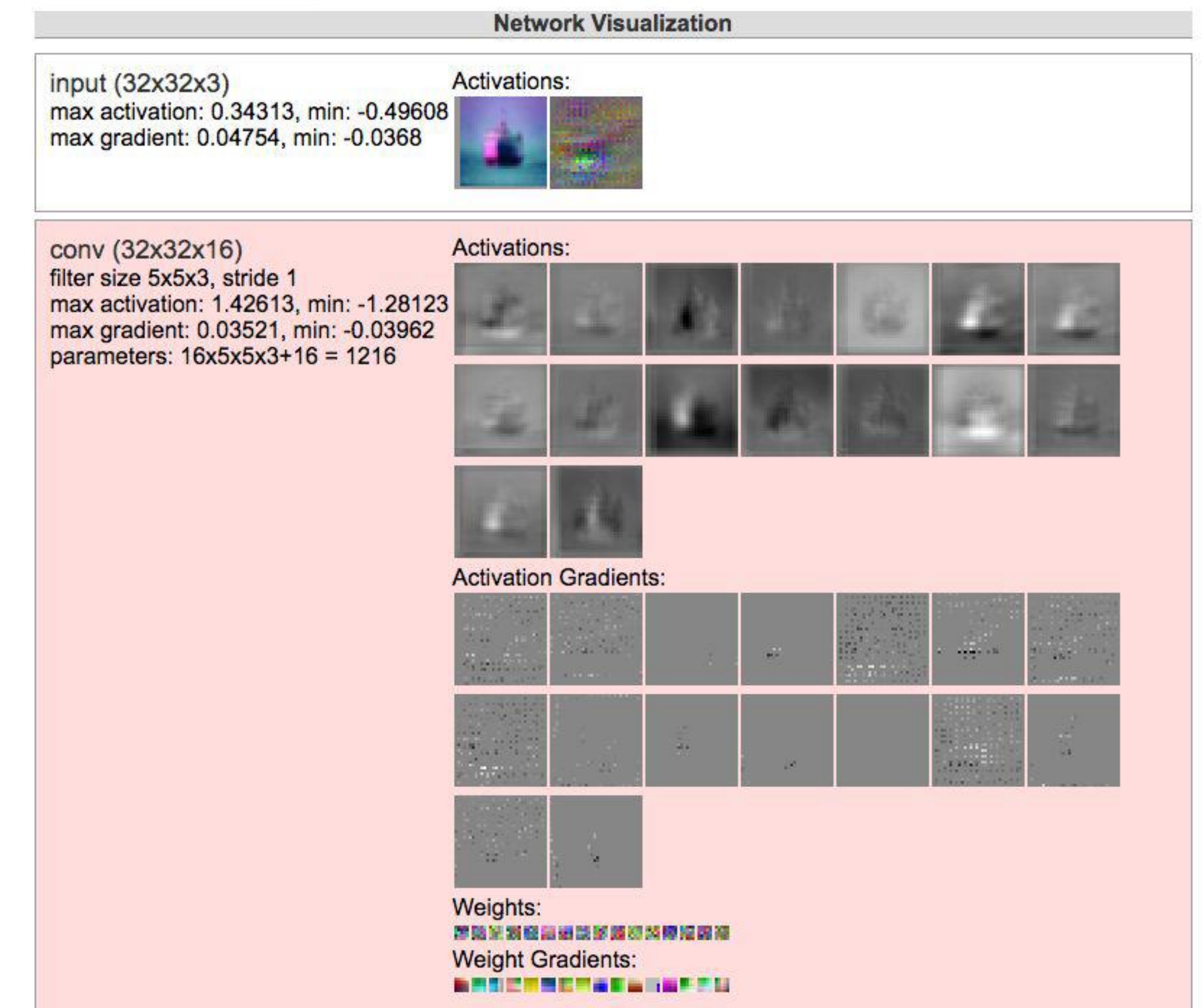
Description

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.



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

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

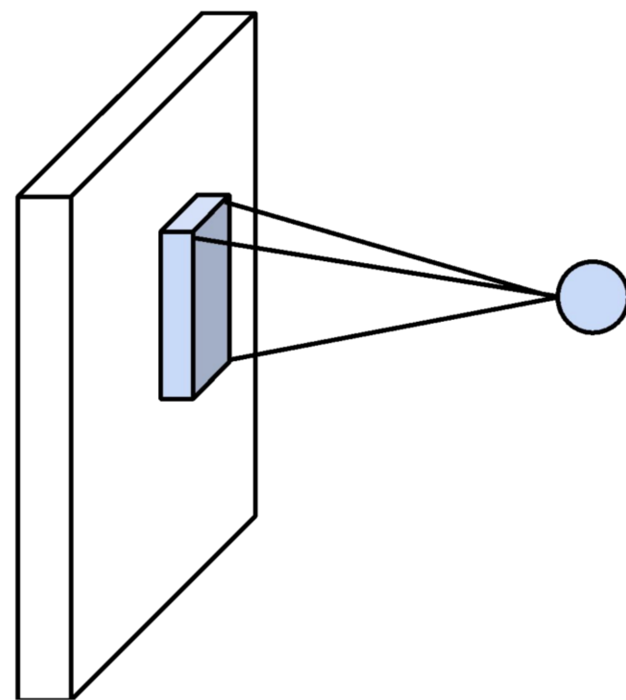
CNN Architectures

Co-financed by the European Union
Connecting Europe Facility

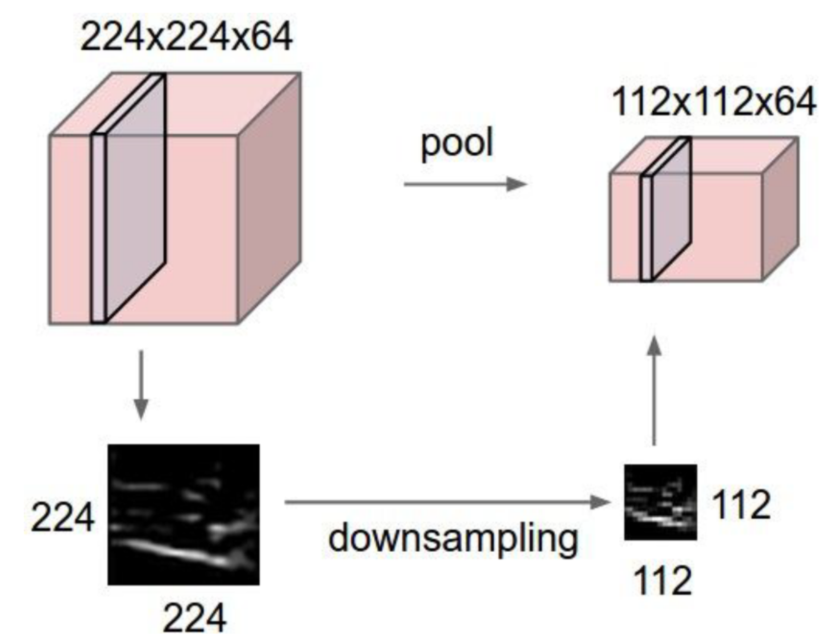
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Components of CNNs

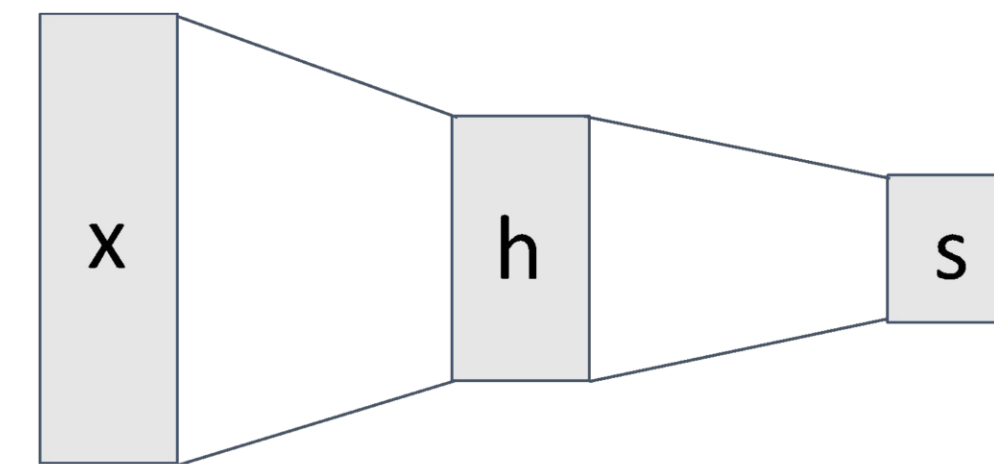
Convolution Layers



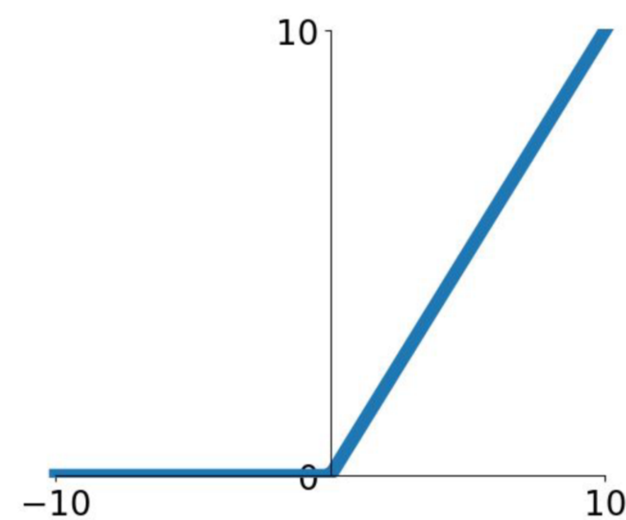
Pooling Layers



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.

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 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.

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

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

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

"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:

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

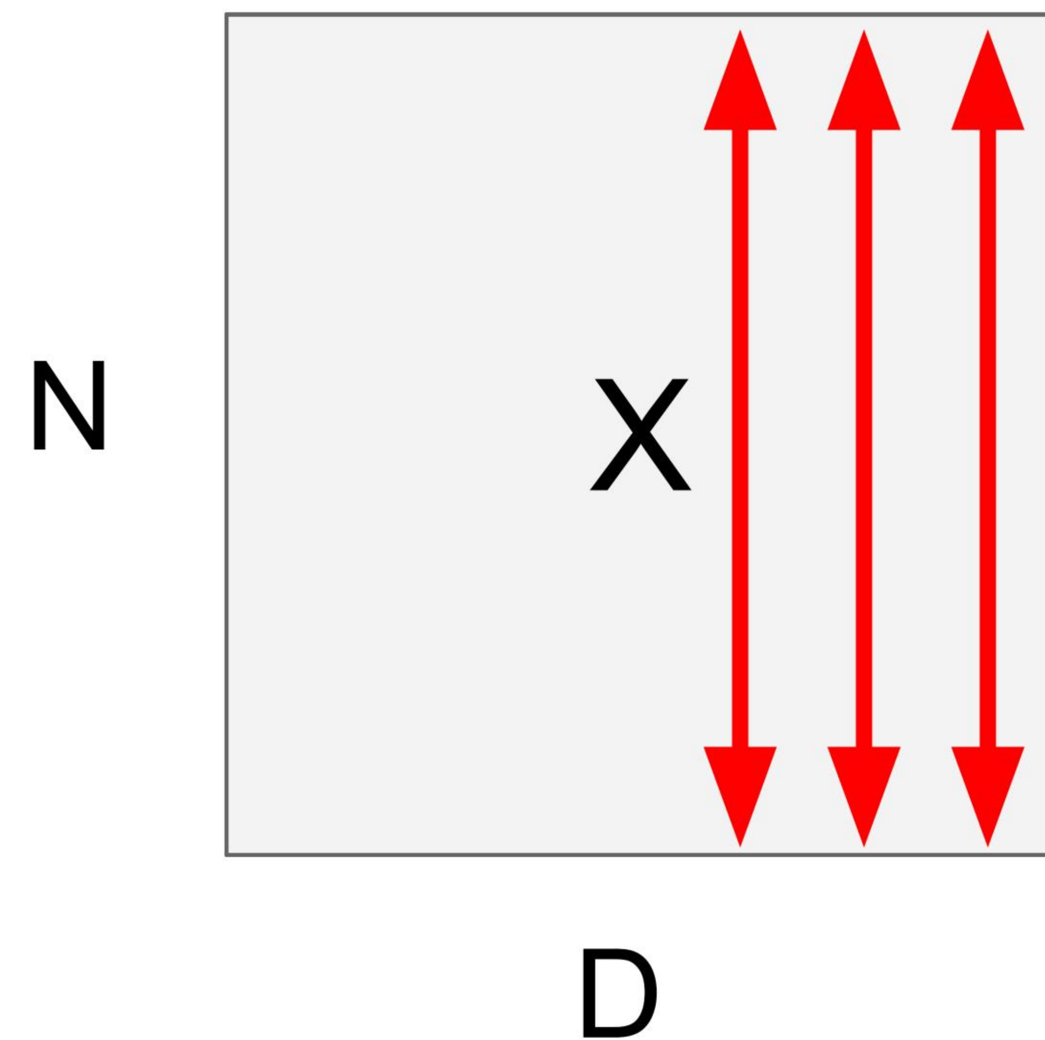
[Ioffe and Szegedy, 2015]

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

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

[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: $x : N \times D$

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

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

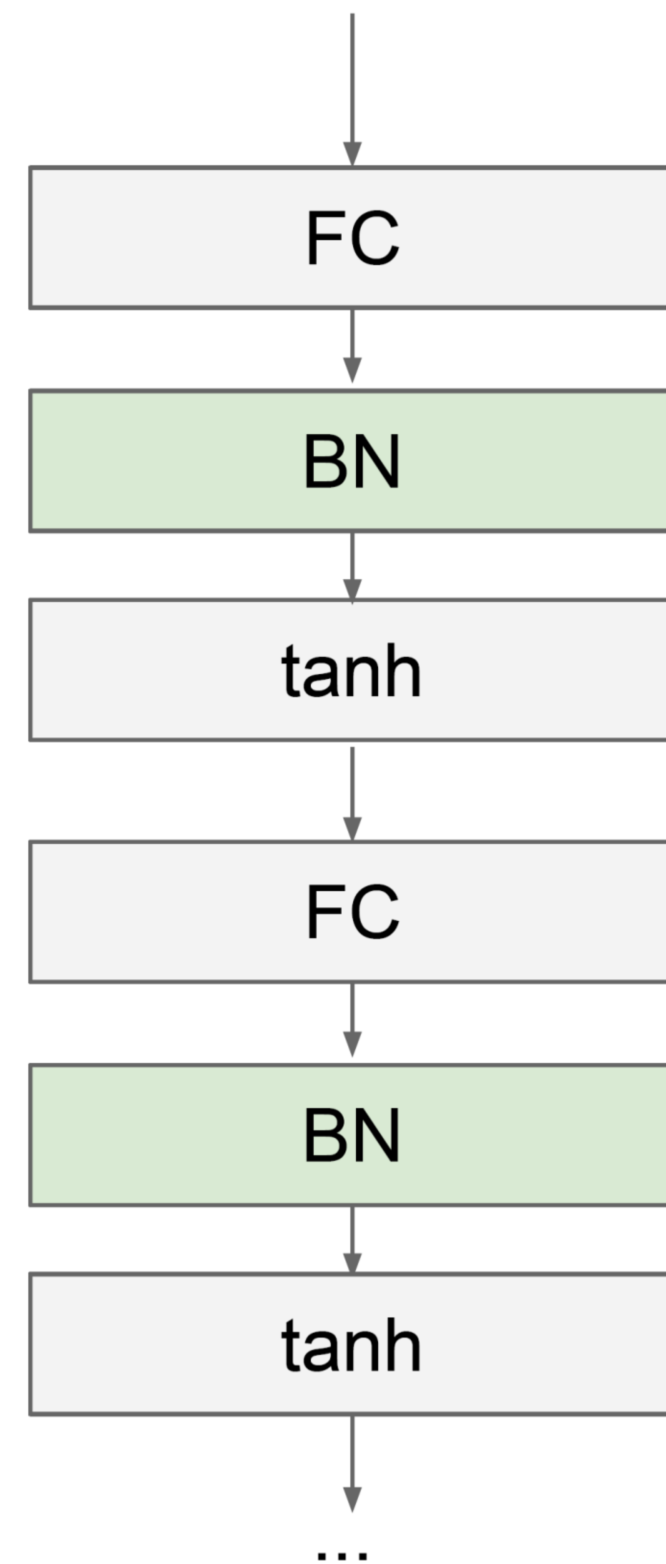
Normalized x, Shape is N x D

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

Output, Shape is N x D

[Ioffe and Szegedy, 2015]

Batch Normalization



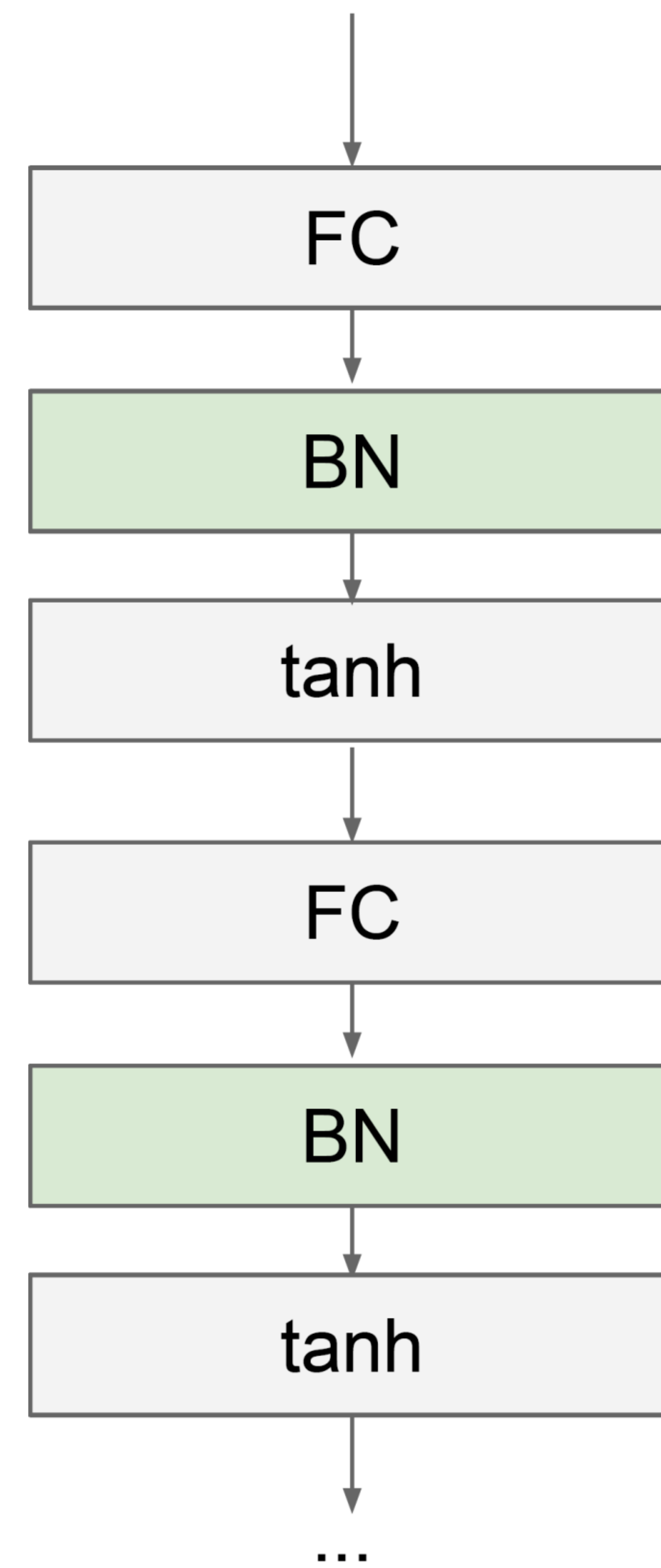
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

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

[Ioffe and Szegedy, 2015]

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



- 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]

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.

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

$x: N \times D$

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)

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

Normalize ↓ ↓ ↓

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

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

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

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

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×C×H×W

Normalize ↓ (N) ↓ (H) ↓ (W)

μ,σ: 1×C×1×1

γ,β: 1×C×1×1

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

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

x: N×C×H×W

Normalize ↓ (H) ↓ (W)

μ,σ: N×C×1×1

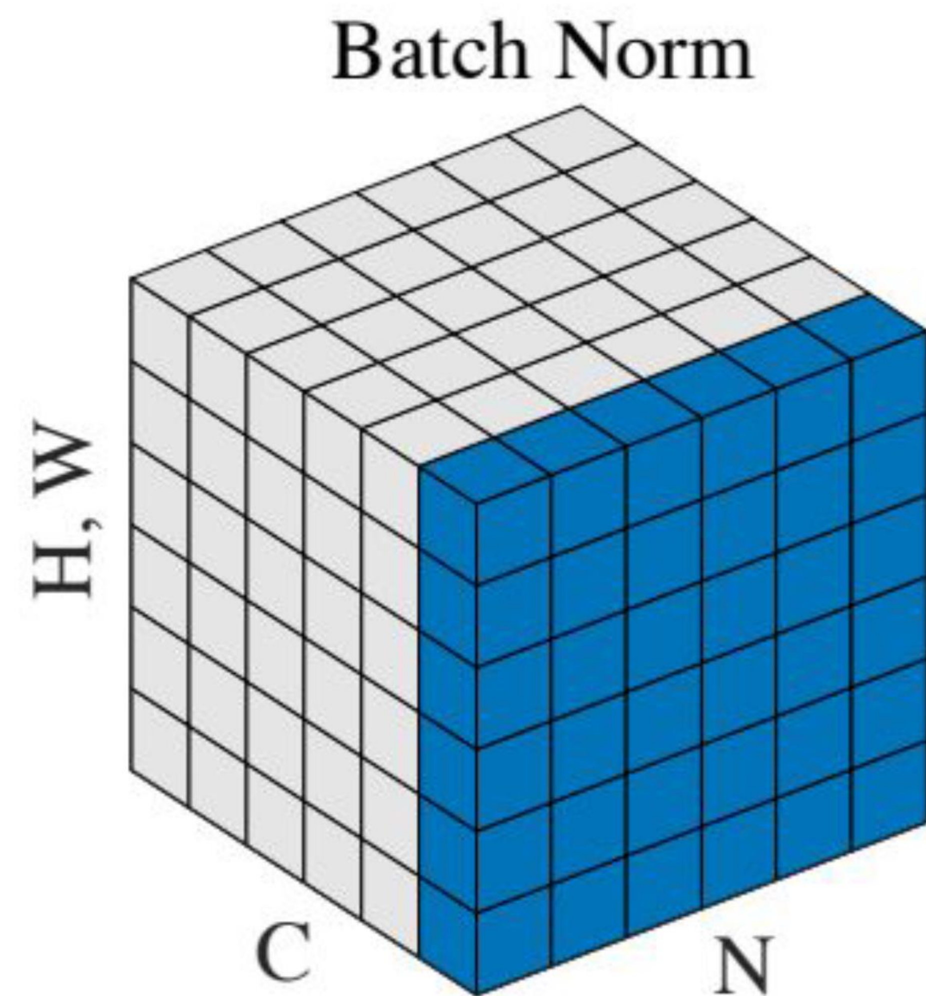
γ,β: 1×C×1×1

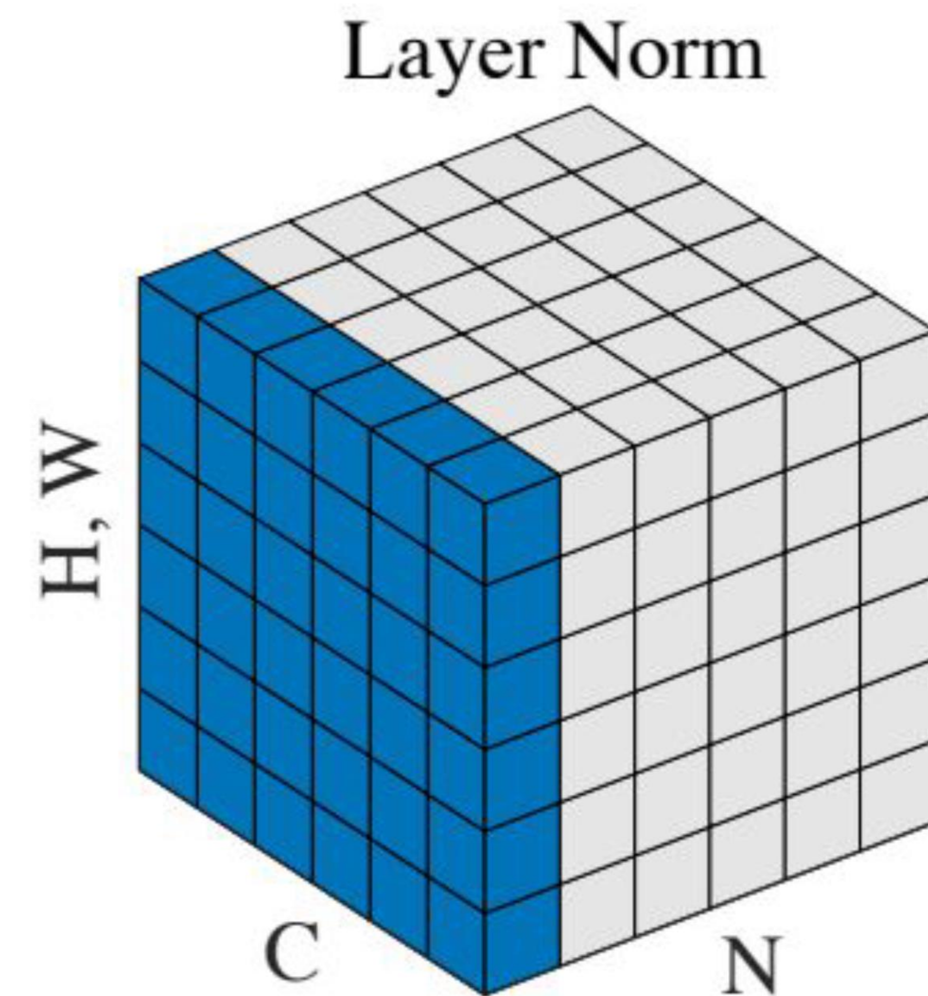
$y = \gamma(x-\mu)/\sigma+\beta$

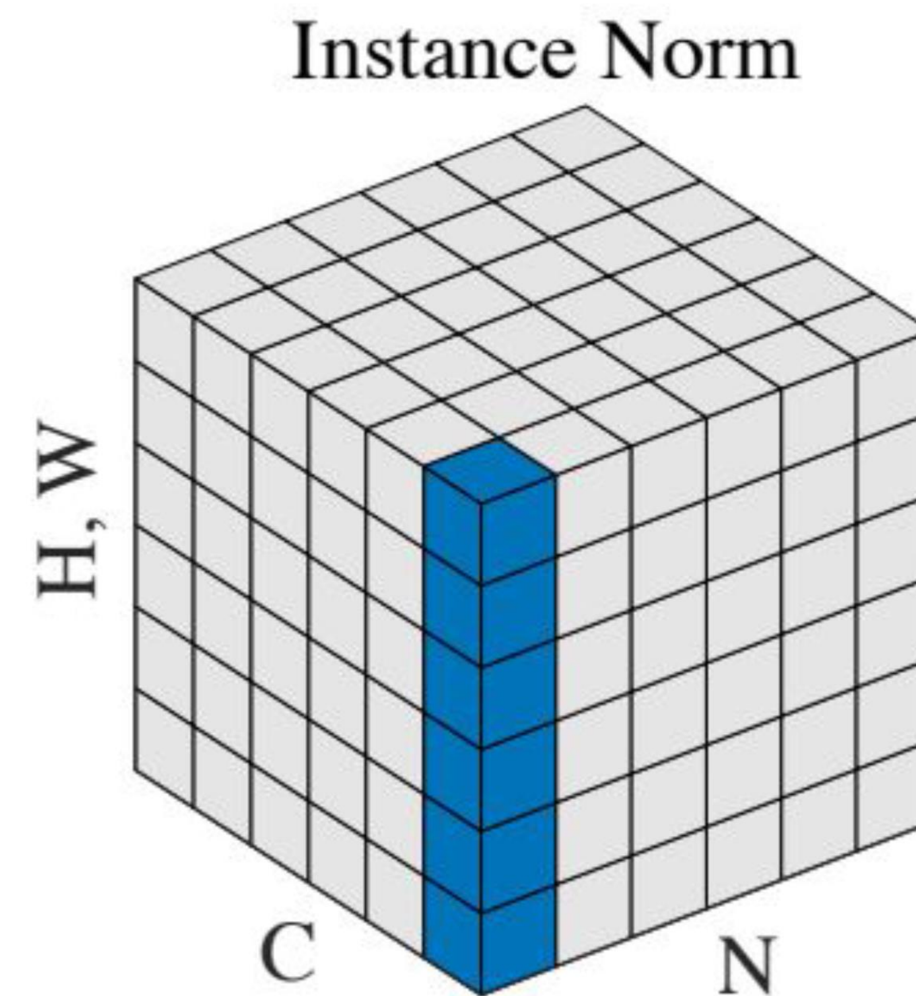
Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

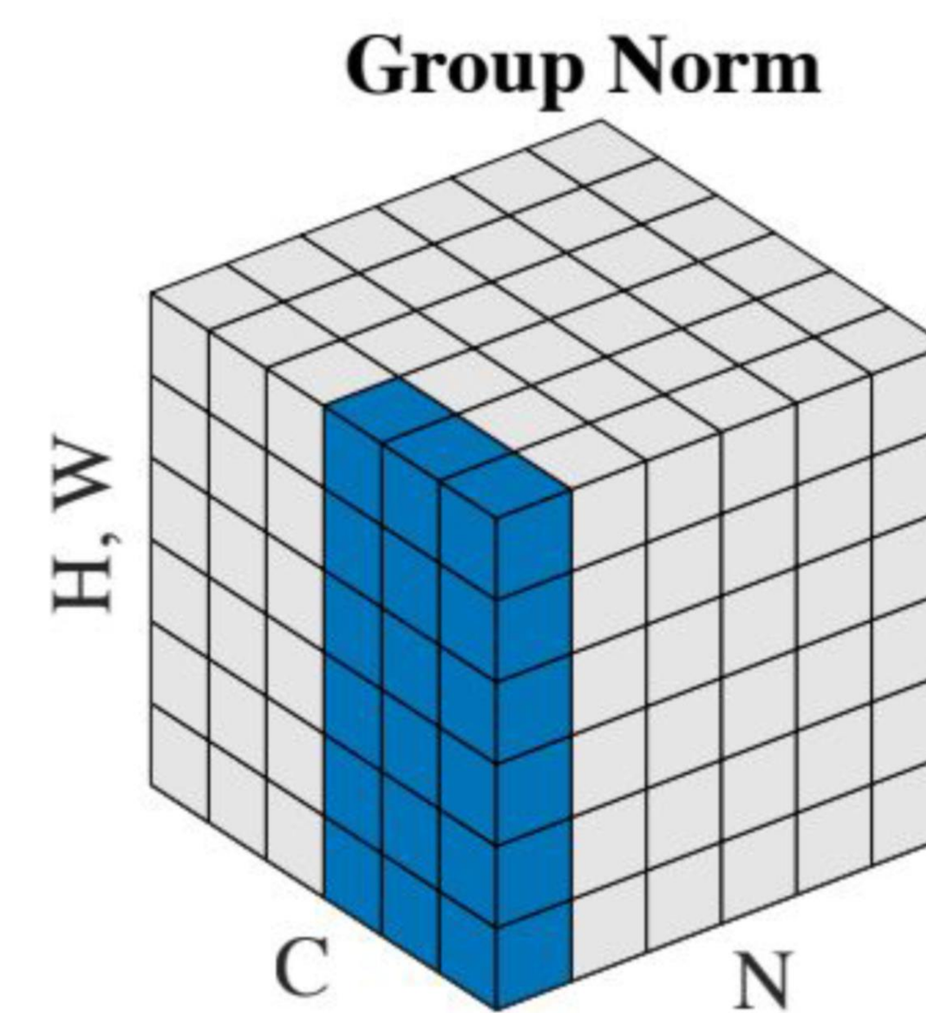
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Comparison of Normalization Layers





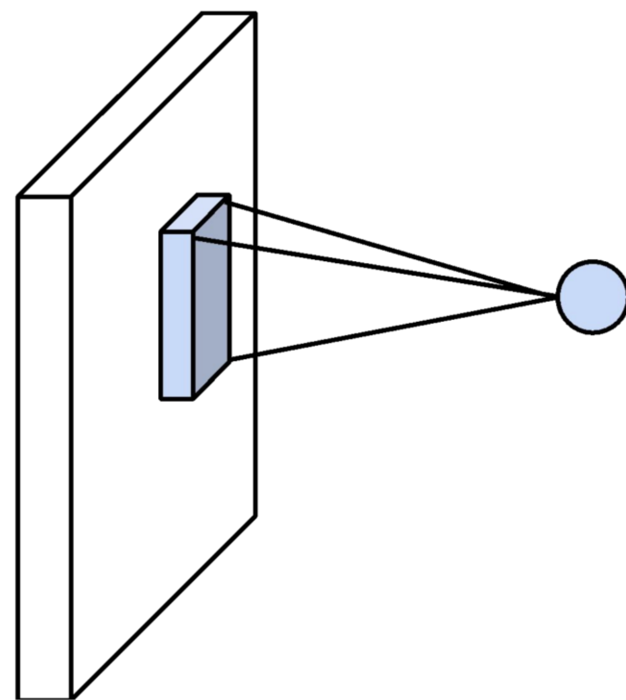




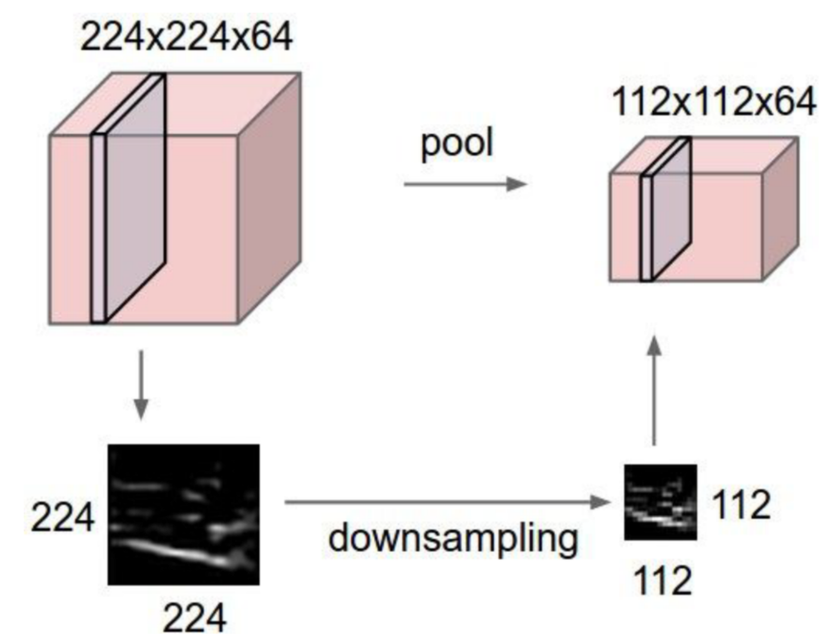
Wu and He, “Group Normalization”, ECCV 2018

Components of CNNs

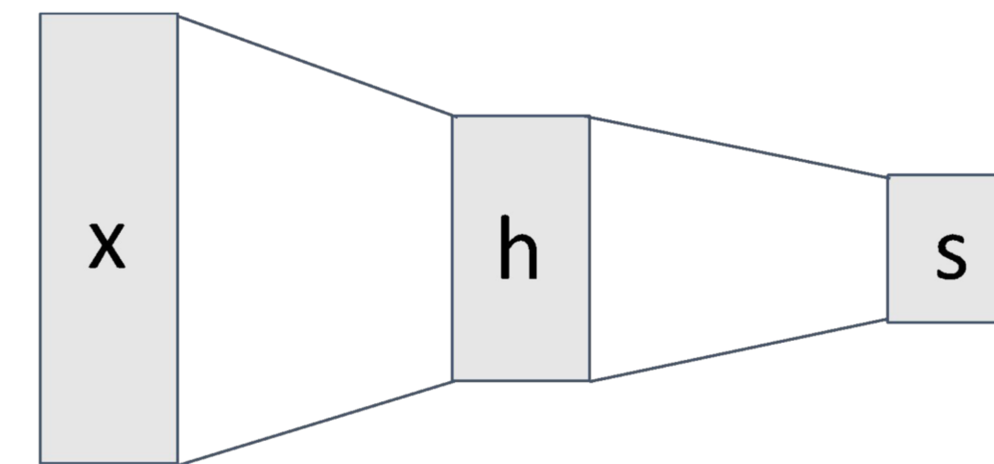
Convolution Layers



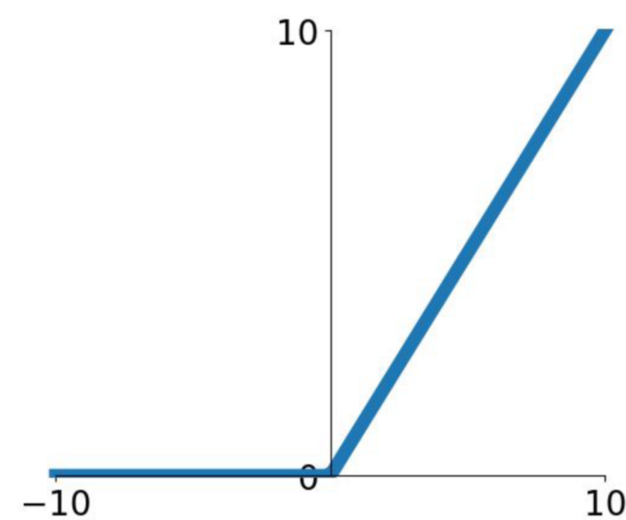
Pooling Layers



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*

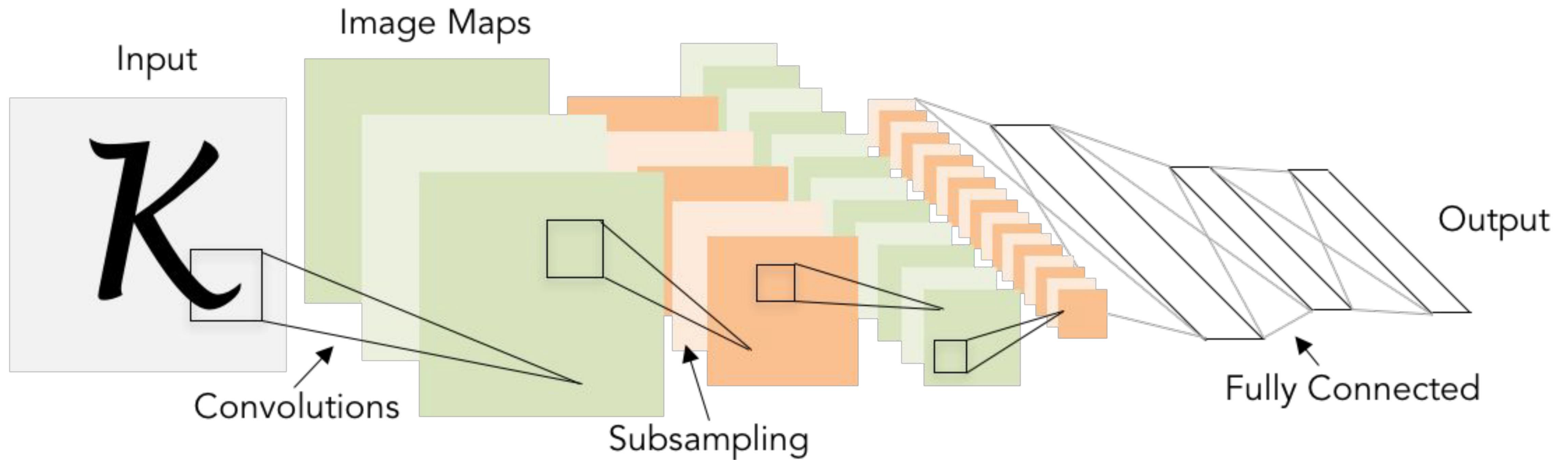
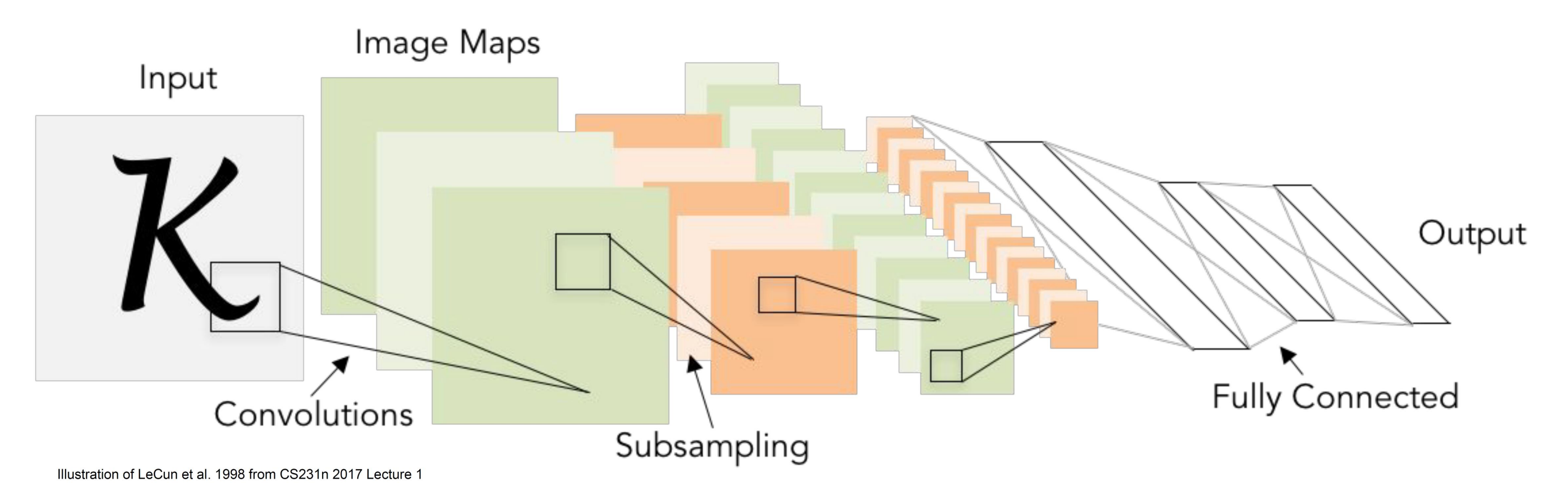


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

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: *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:

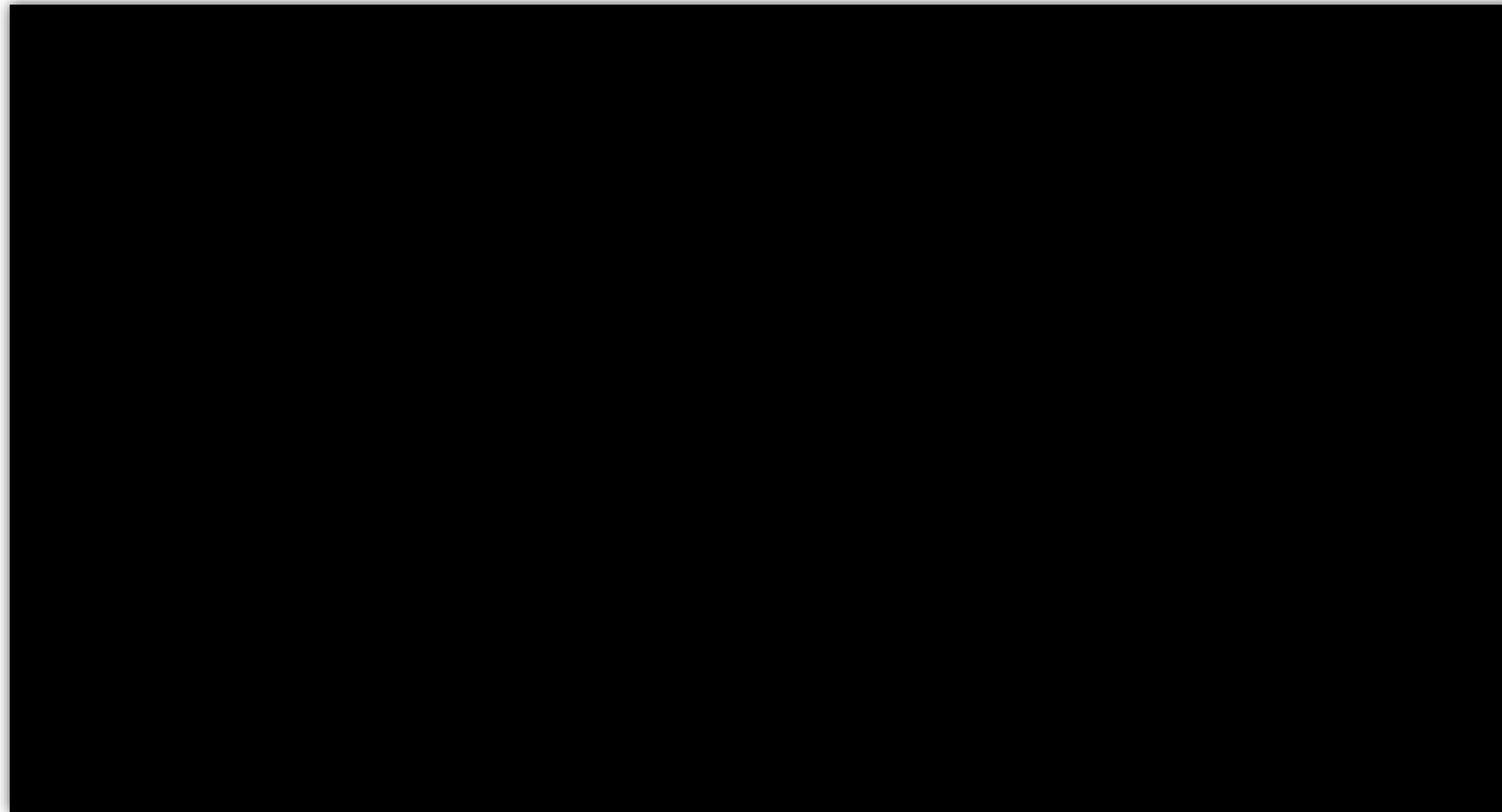
- Convolutional Layer: 6 filters of size 5x5 with a stride of 1 and a sigmoid activation function
- Average Pooling Layer: non-overlapping 2x2 window with a stride of 2
- Convolutional Layer: 16 filters of size 5x5 with a stride of 1 and a sigmoid activation function
- Average Pooling Layer: non-overlapping 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.

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

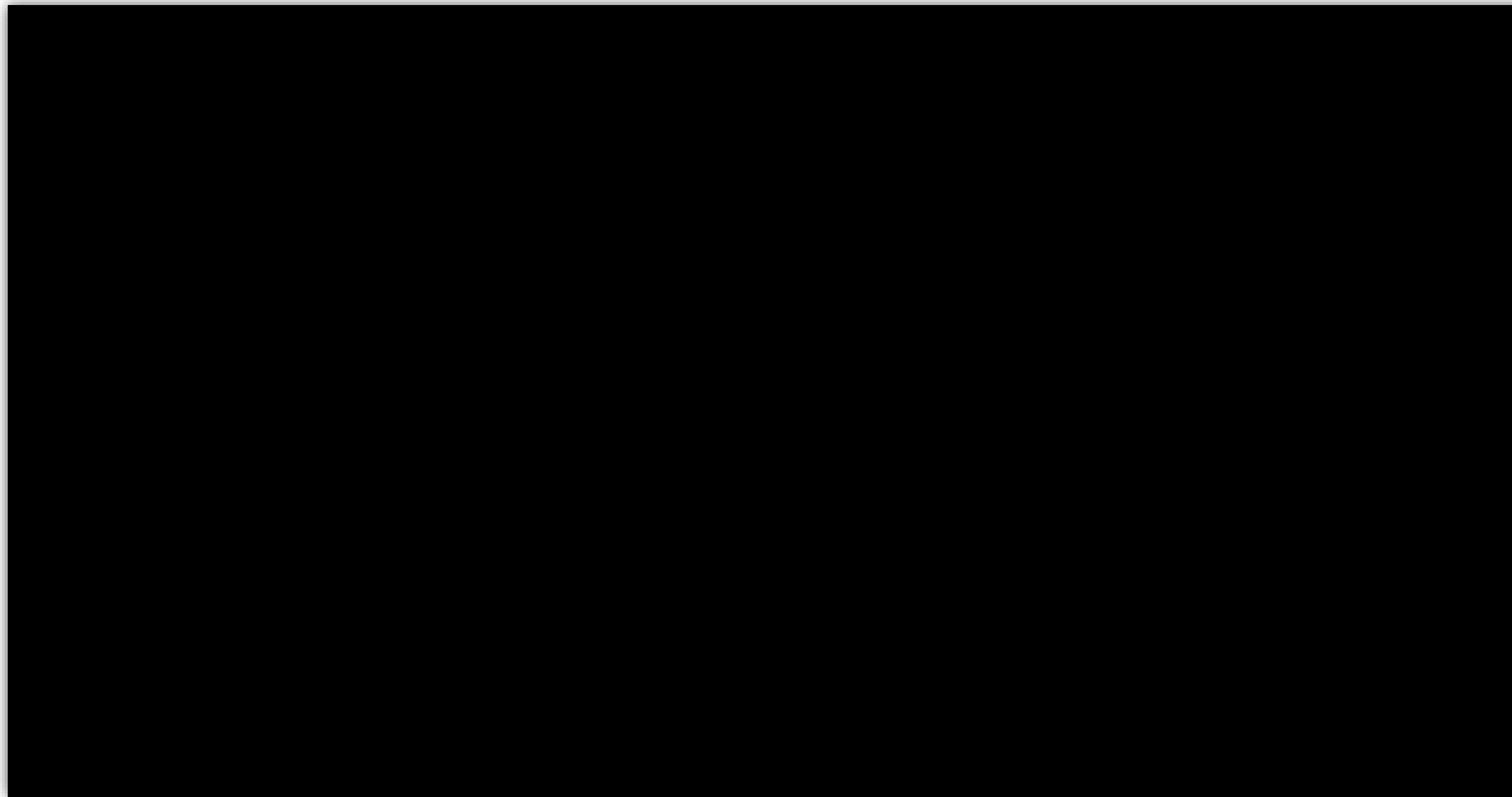


https://youtu.be/Tsvxx-GGlTg

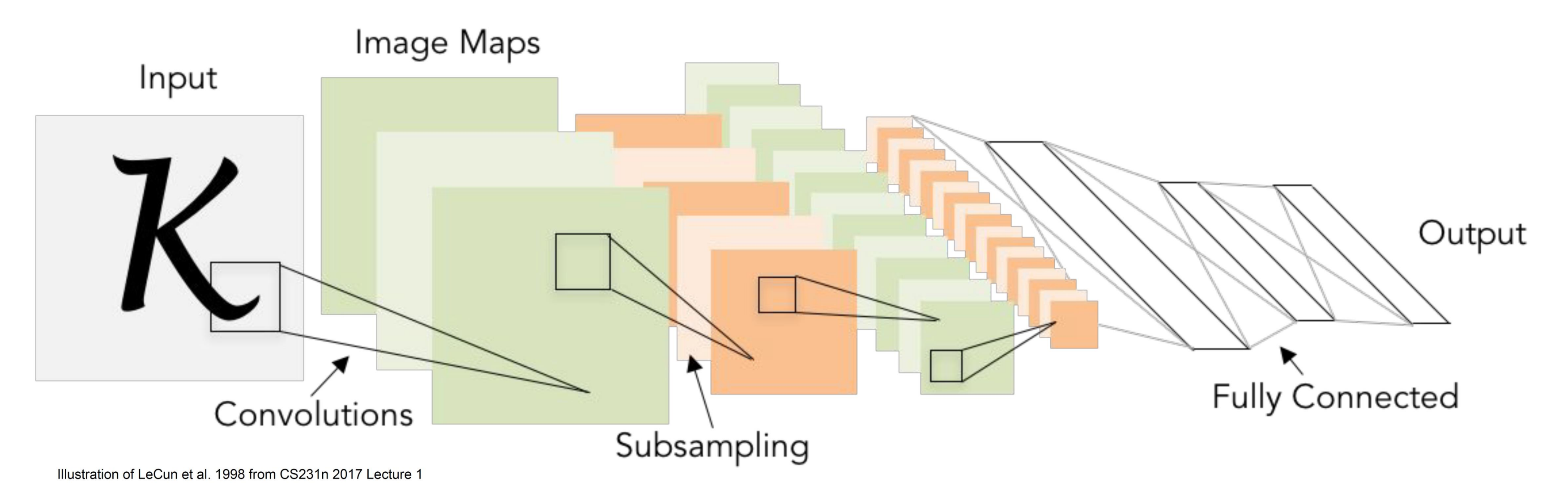
Co-financed by the European Union
Connecting Europe Facility

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



https://youtu.be/JboZfxUjLSk



CNN Architectures: *Case studies*

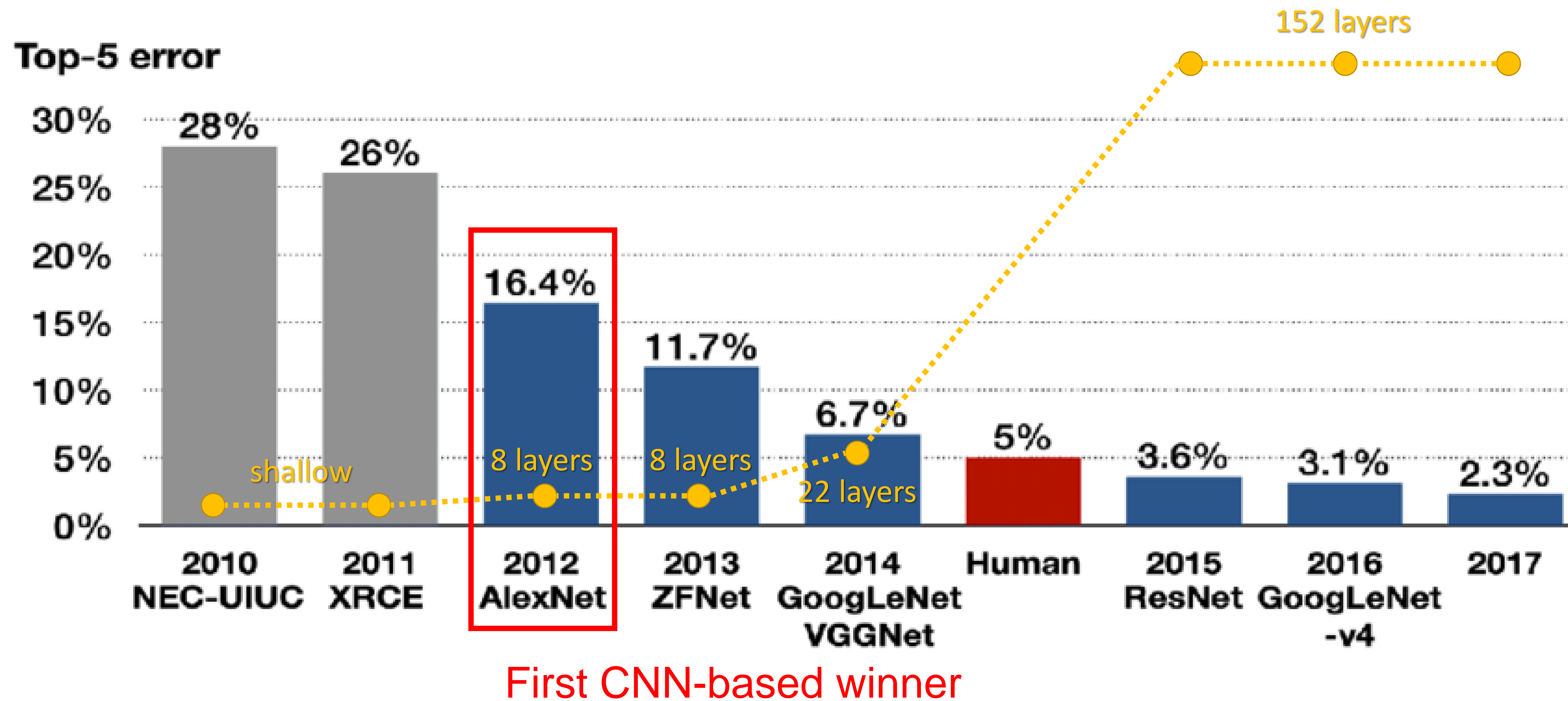
- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

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

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

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.

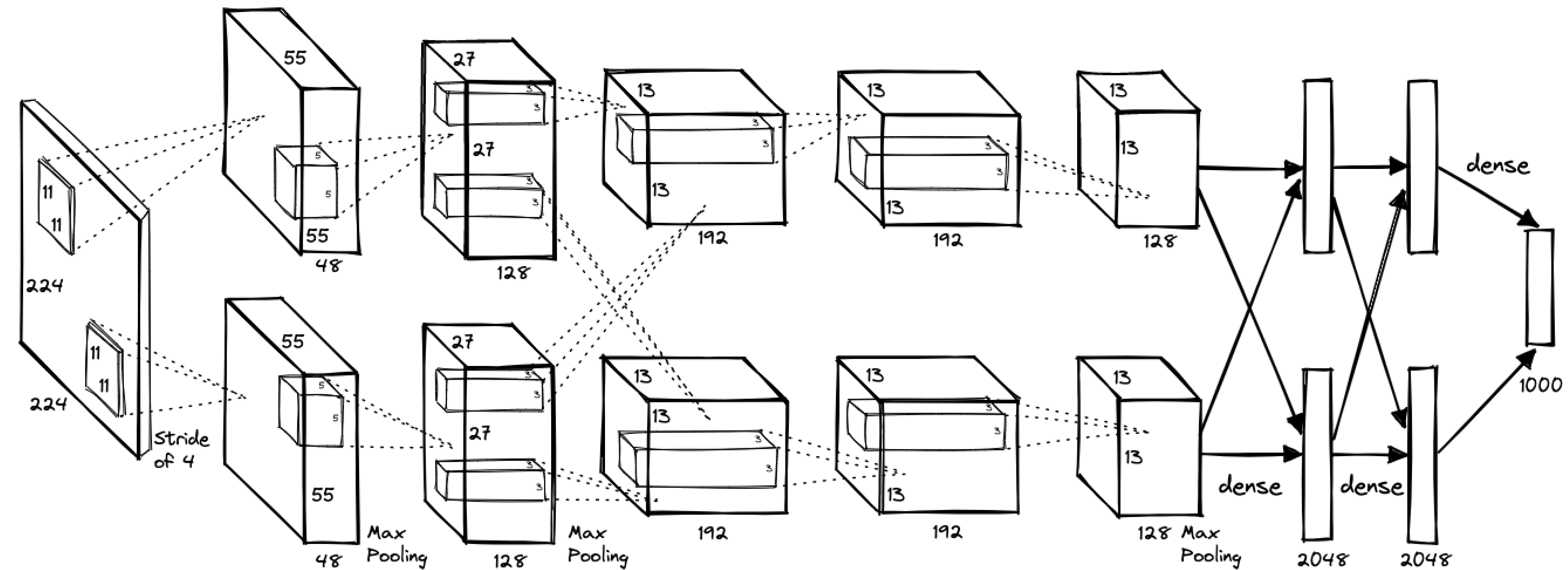
The architecture of AlexNet was much deeper and wider than previous convolutional neural networks, with more parameters and a larger number of neurons in each layer. It was trained on a large-scale dataset of 1.2 million images, which made it possible to learn a rich set of hierarchical features and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

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

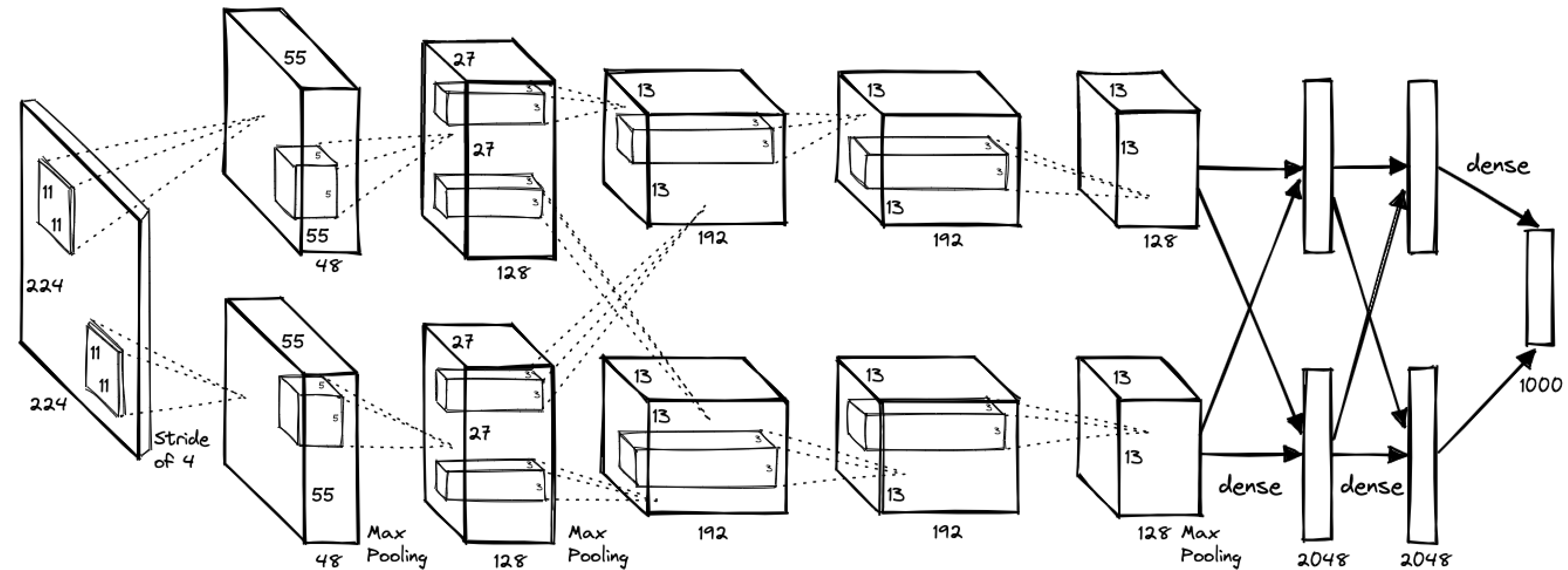
CNN Architectures: *AlexNet*

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



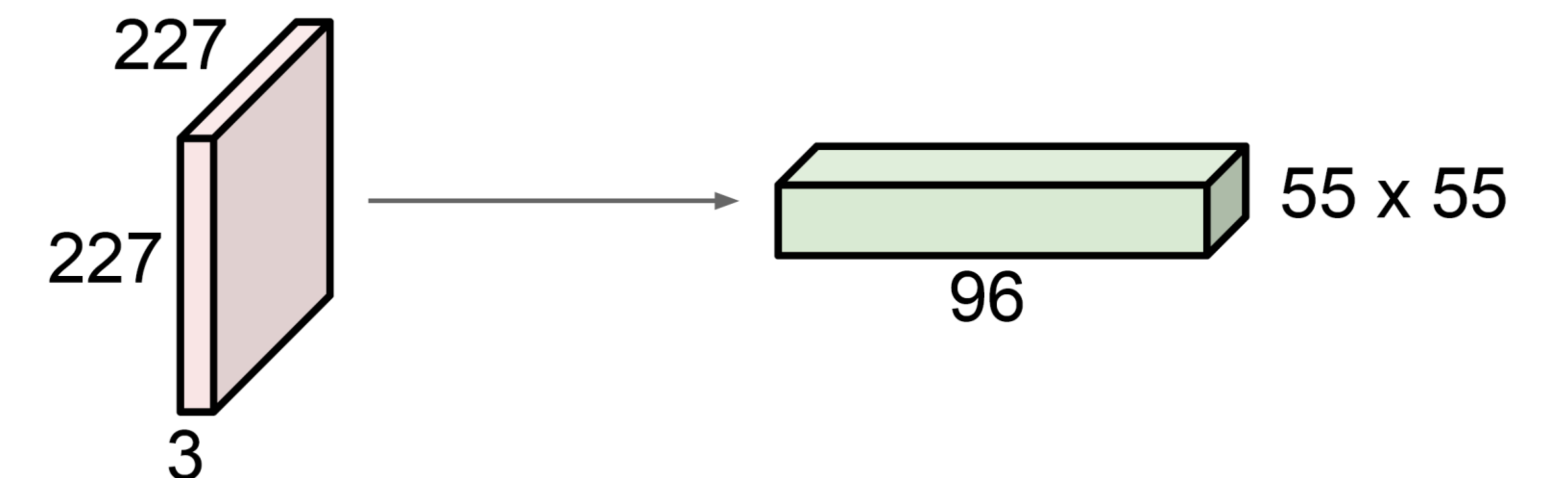
CNN Architectures: *AlexNet*

W’ = (W - F + 2P) / S + 1



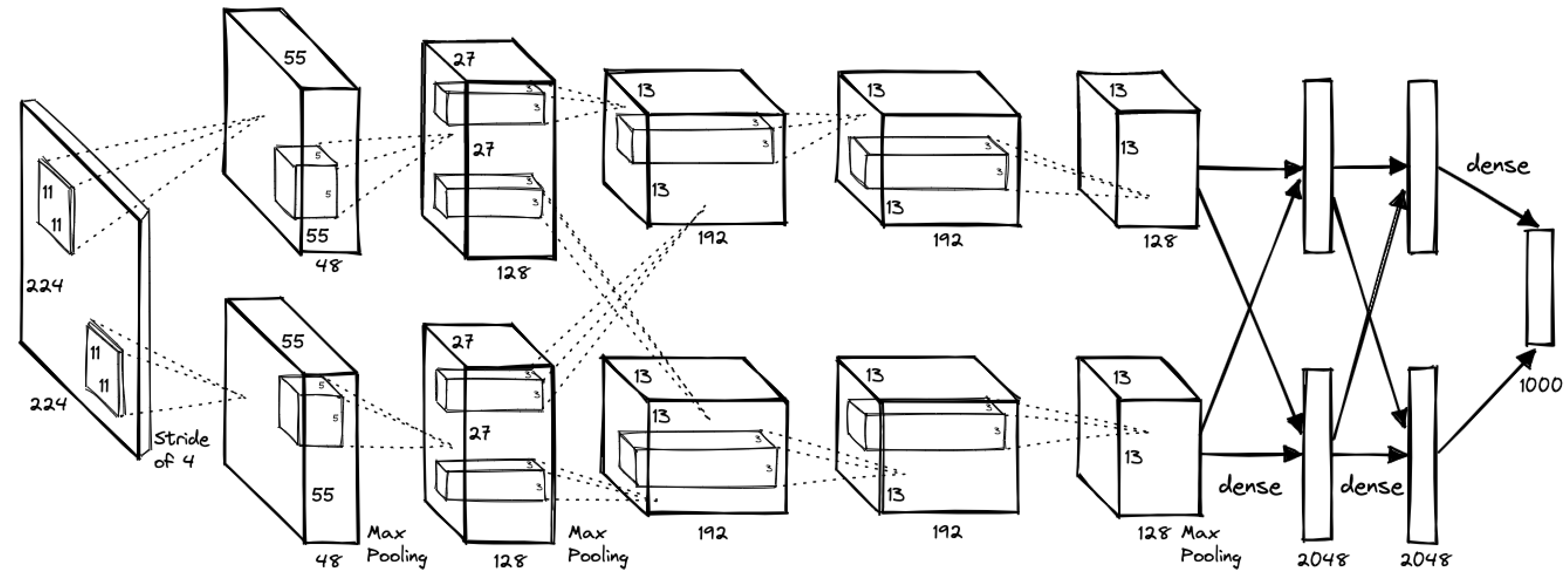
Input: 227x227x3 images
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Output volume **[55x55x96]**

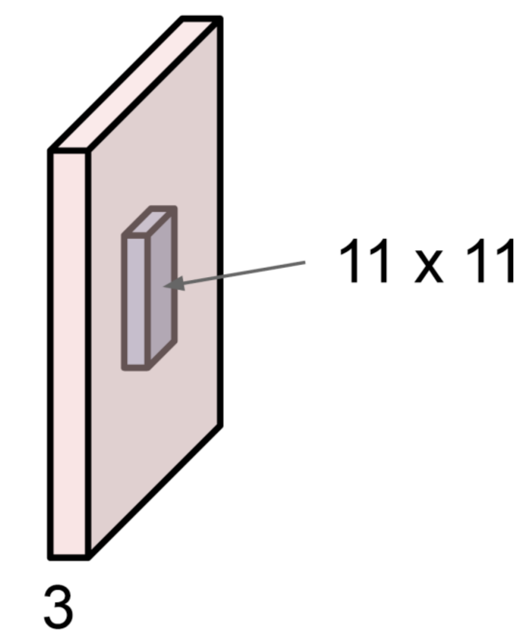


CNN Architectures: *AlexNet*

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



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?



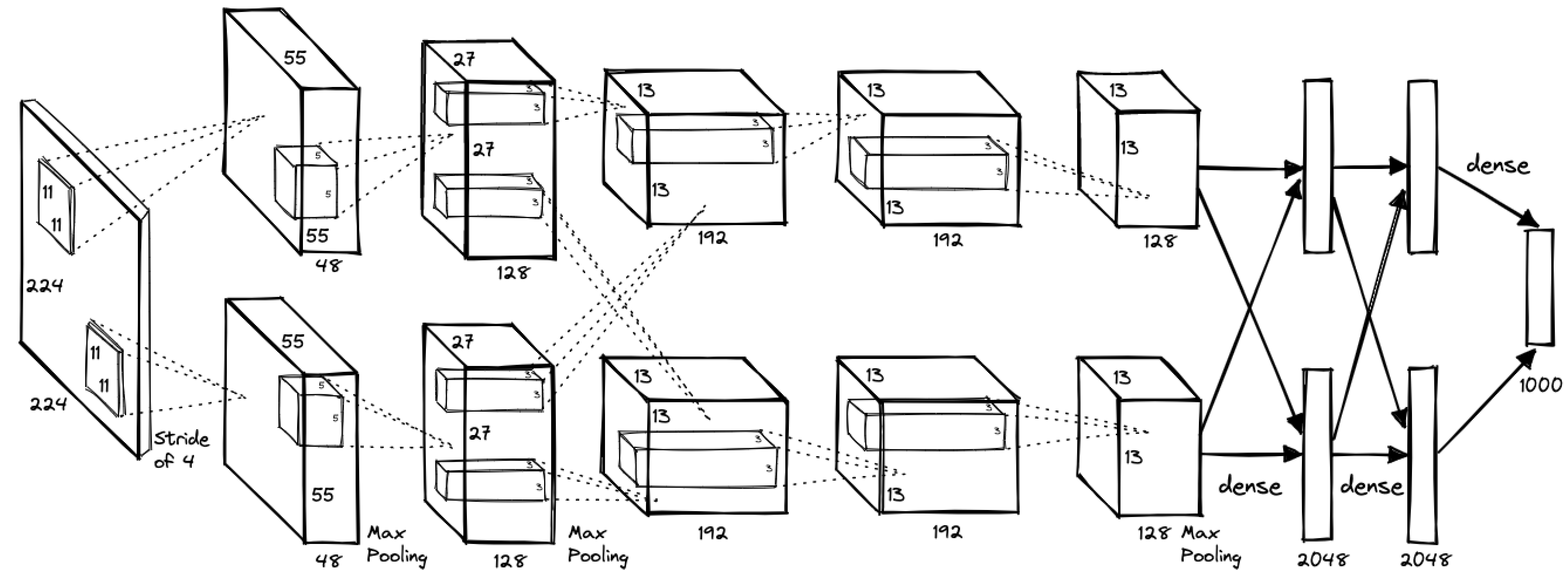
Parameters: (11*11*3 + 1)*96 = **35K**

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*

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

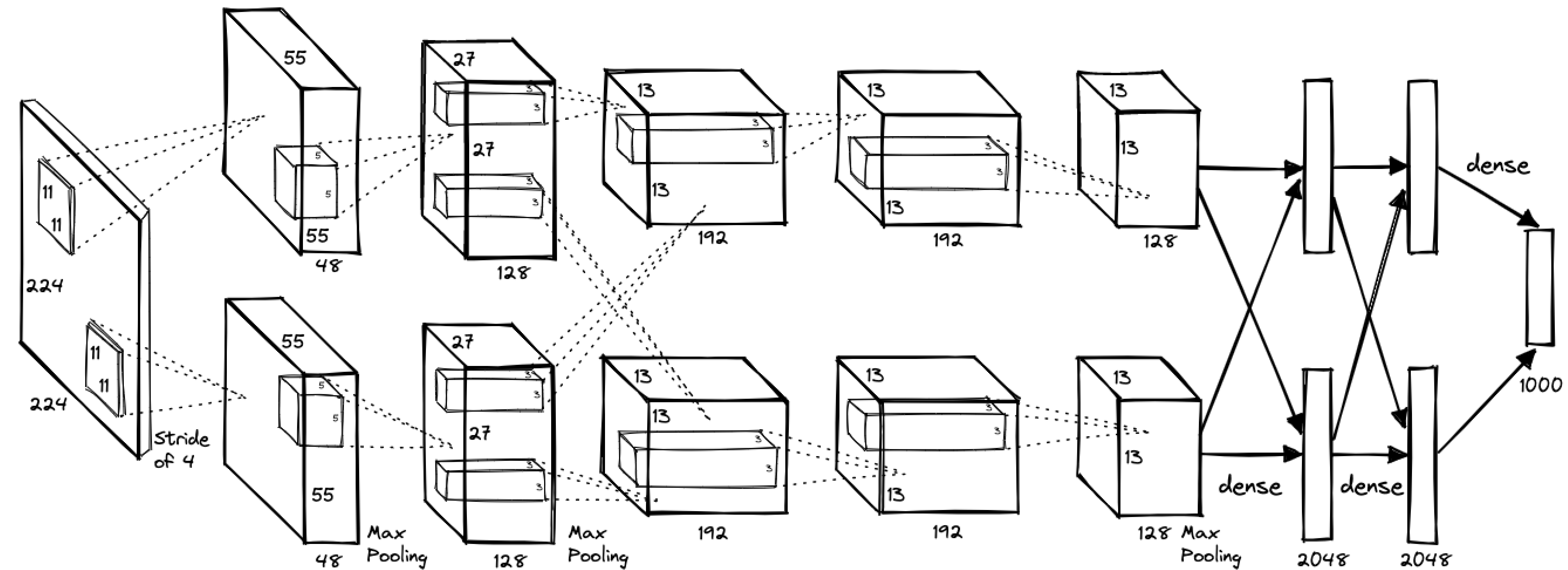


Input: 227x227x3 images
After CONV1: 55x55x96
Second layer (POOL1): 3x3 filters applied at stride 2
Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Output volume: 27x27x96

CNN Architectures: *AlexNet*

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



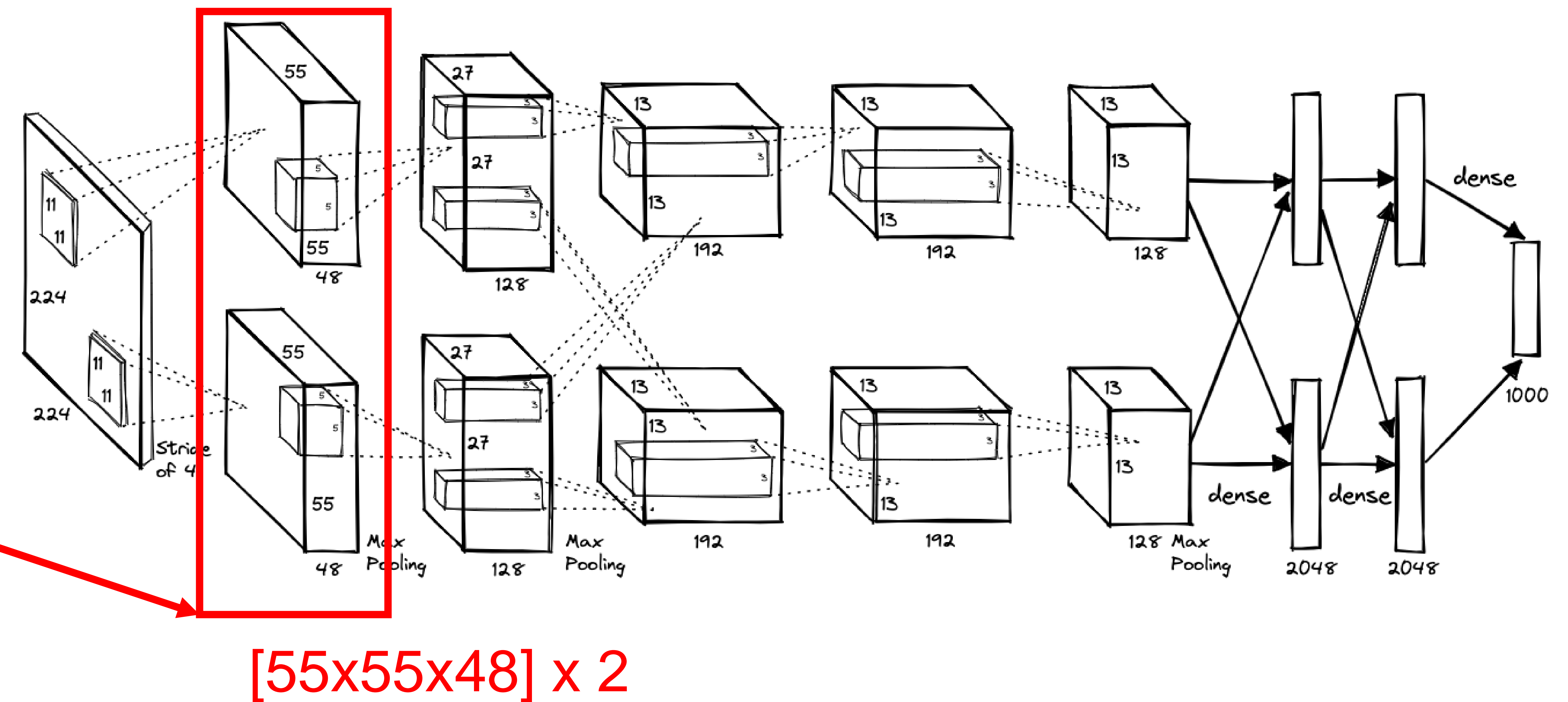
Input: 227x227x3 images
After CONV1: 55x55x96
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Q: what is the number of parameters in this layer?
Parameters: 0!

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

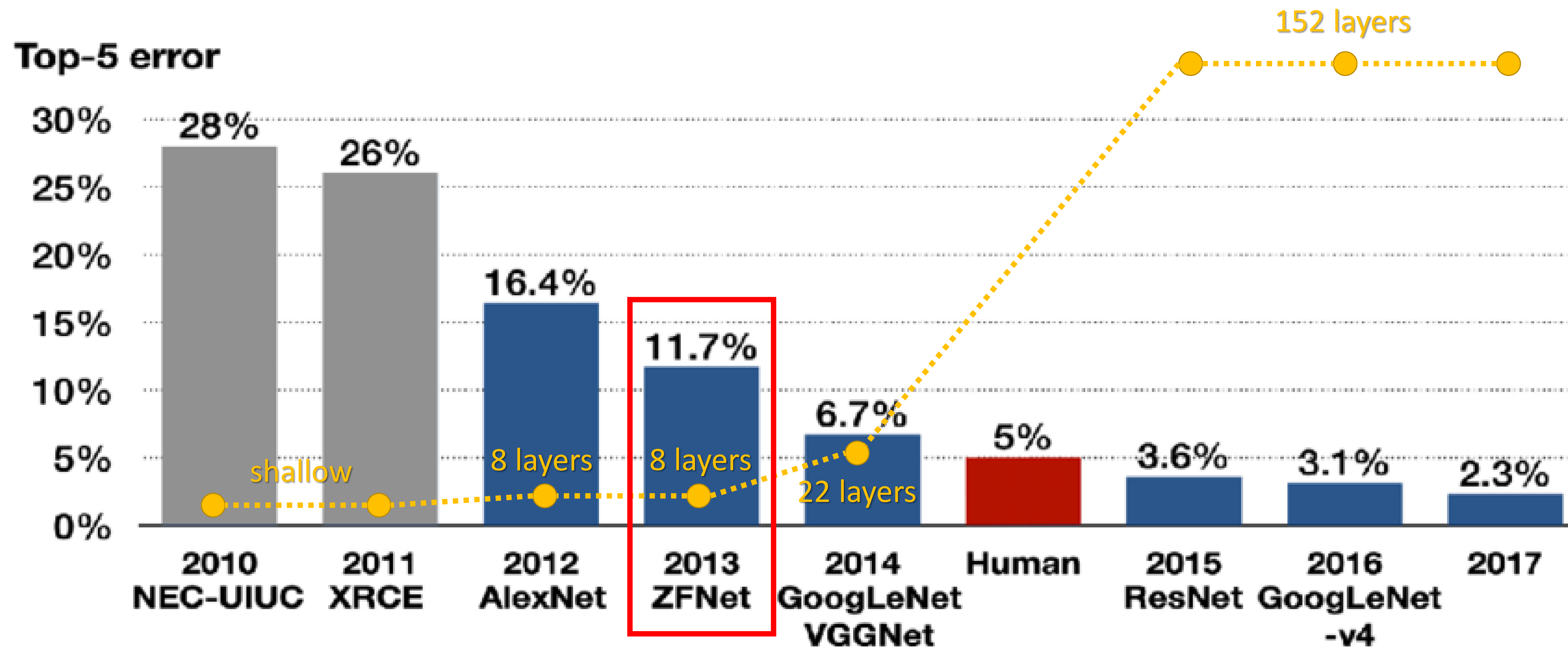
CNN Architectures: *AlexNet*

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)



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

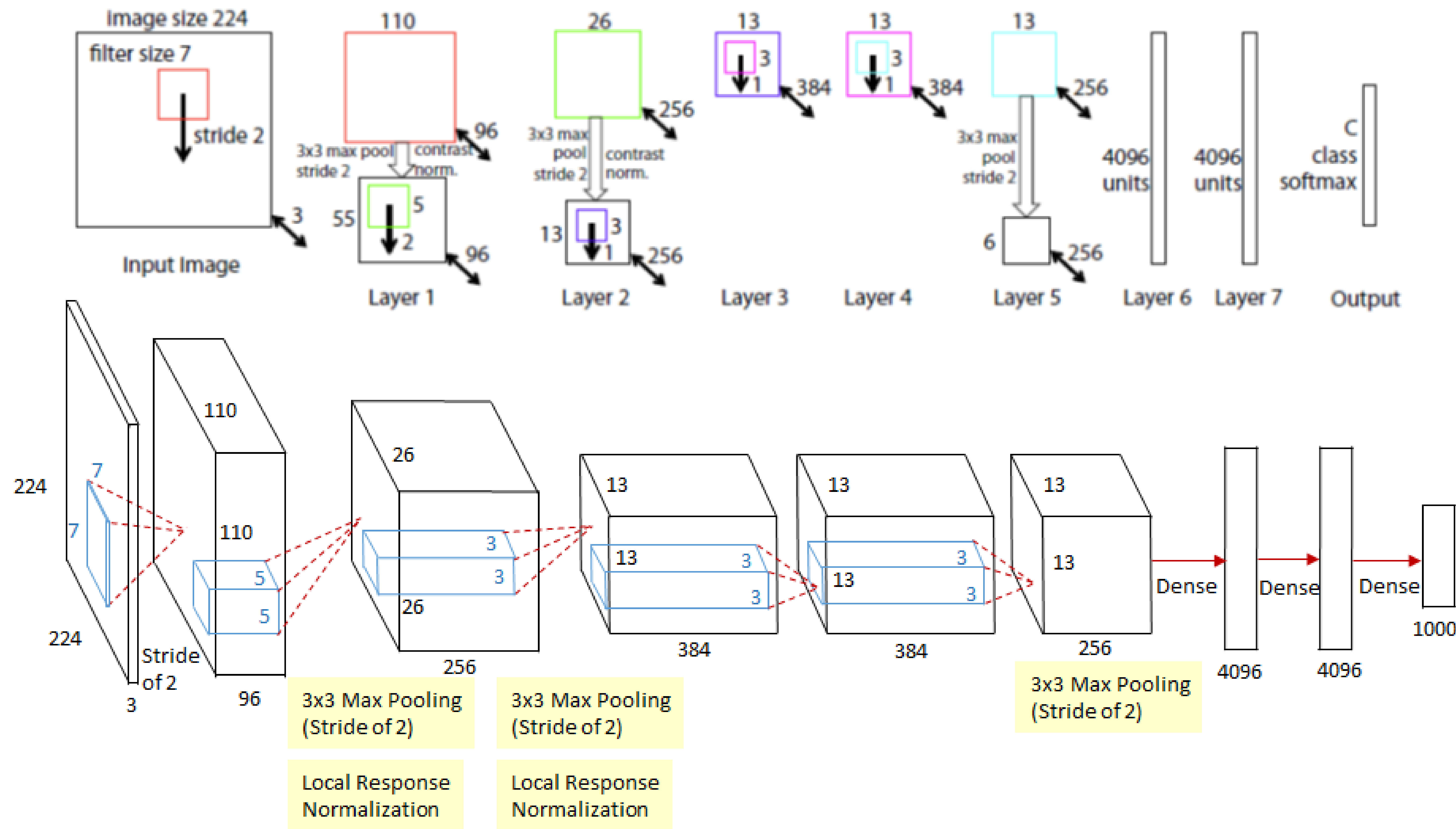
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet: Improved hyperparameters over AlexNet

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: *ZFNet*



AlexNet but:

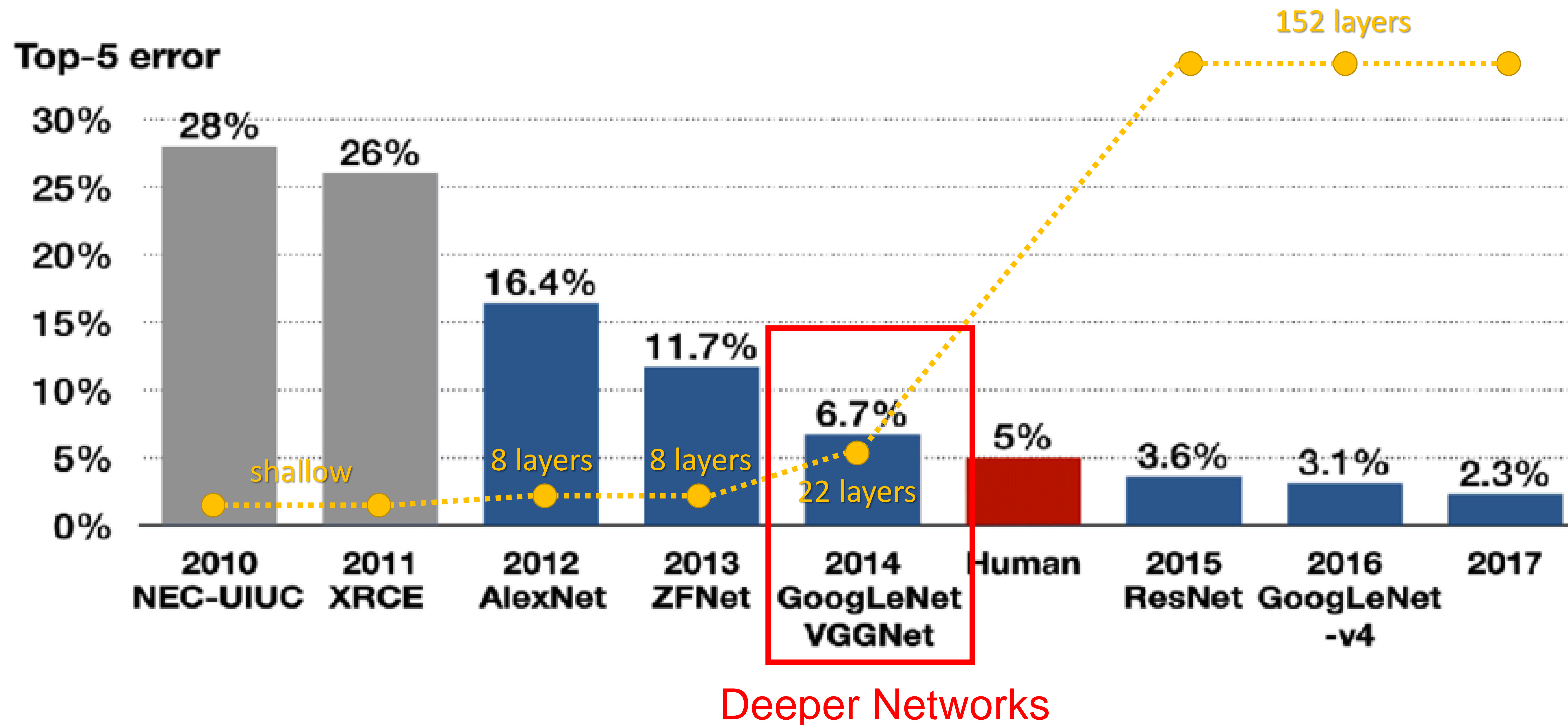
CONV1: change from (11x11 stride 4) to (7x7 stride 2)

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

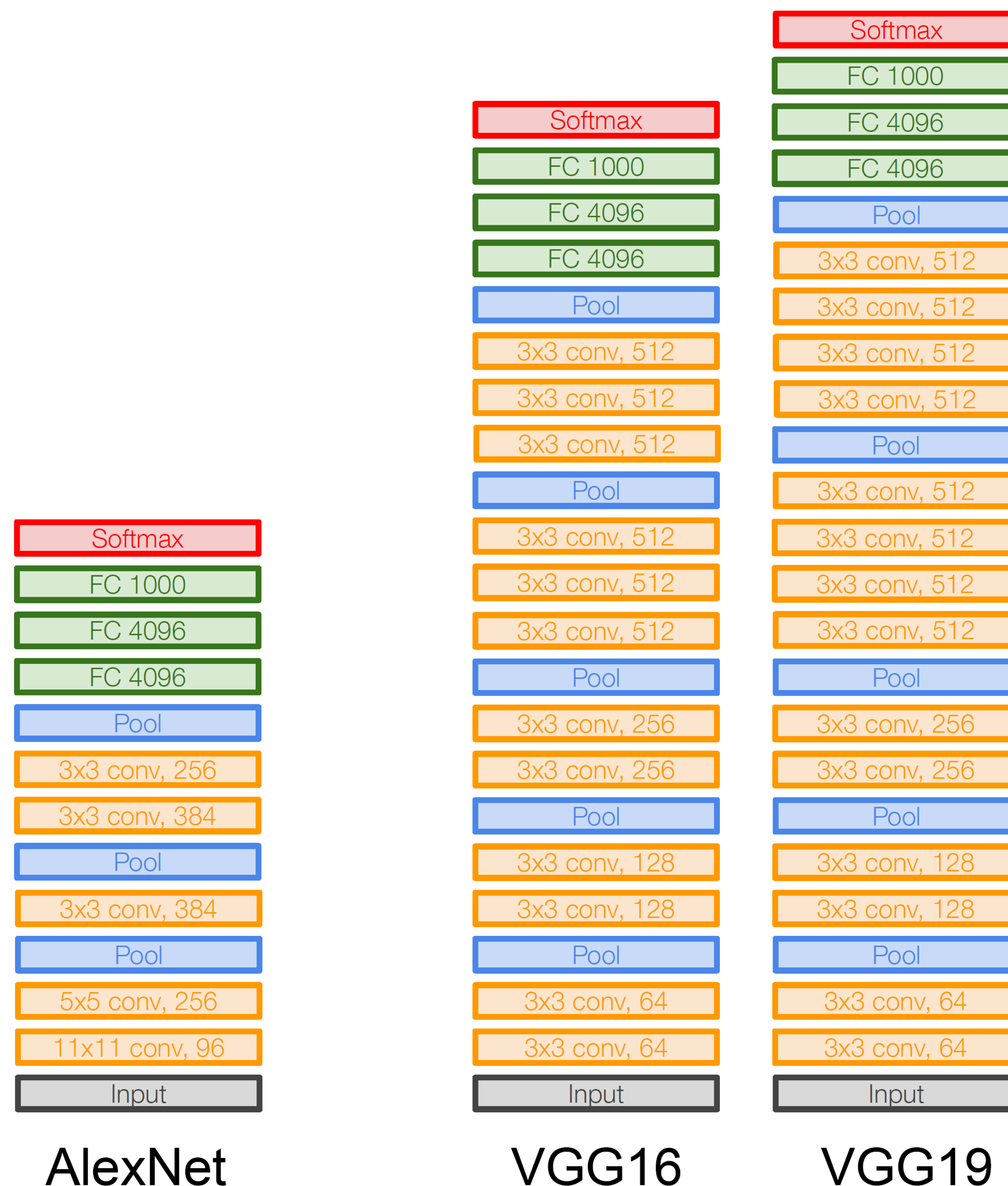


Deeper Networks

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*



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]

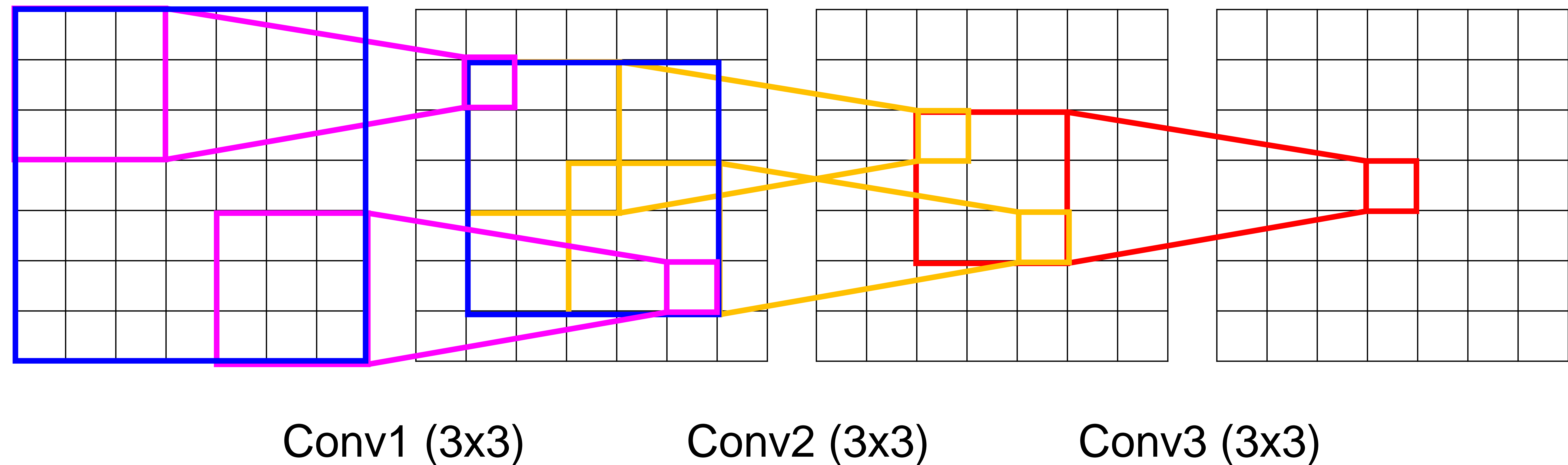
Co-financed by the European Union
Connecting Europe Facility

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*

VGG16 VGG19

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



Conv1 (3x3) Conv2 (3x3) Conv3 (3x3)

[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*

VGG16 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: $3 * (3^2C^2)$ vs. 7^2C^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

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*

VGG16 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]

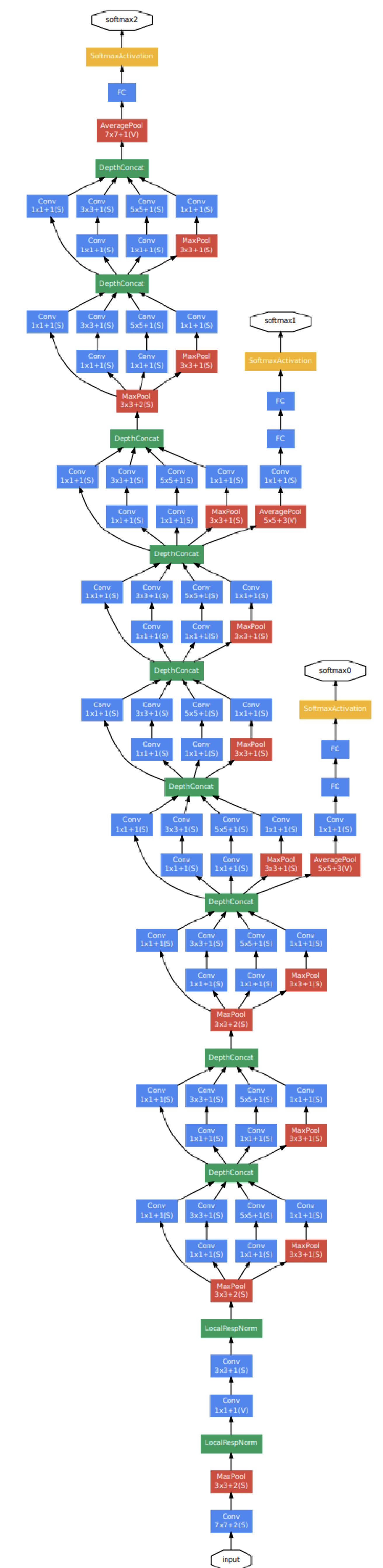
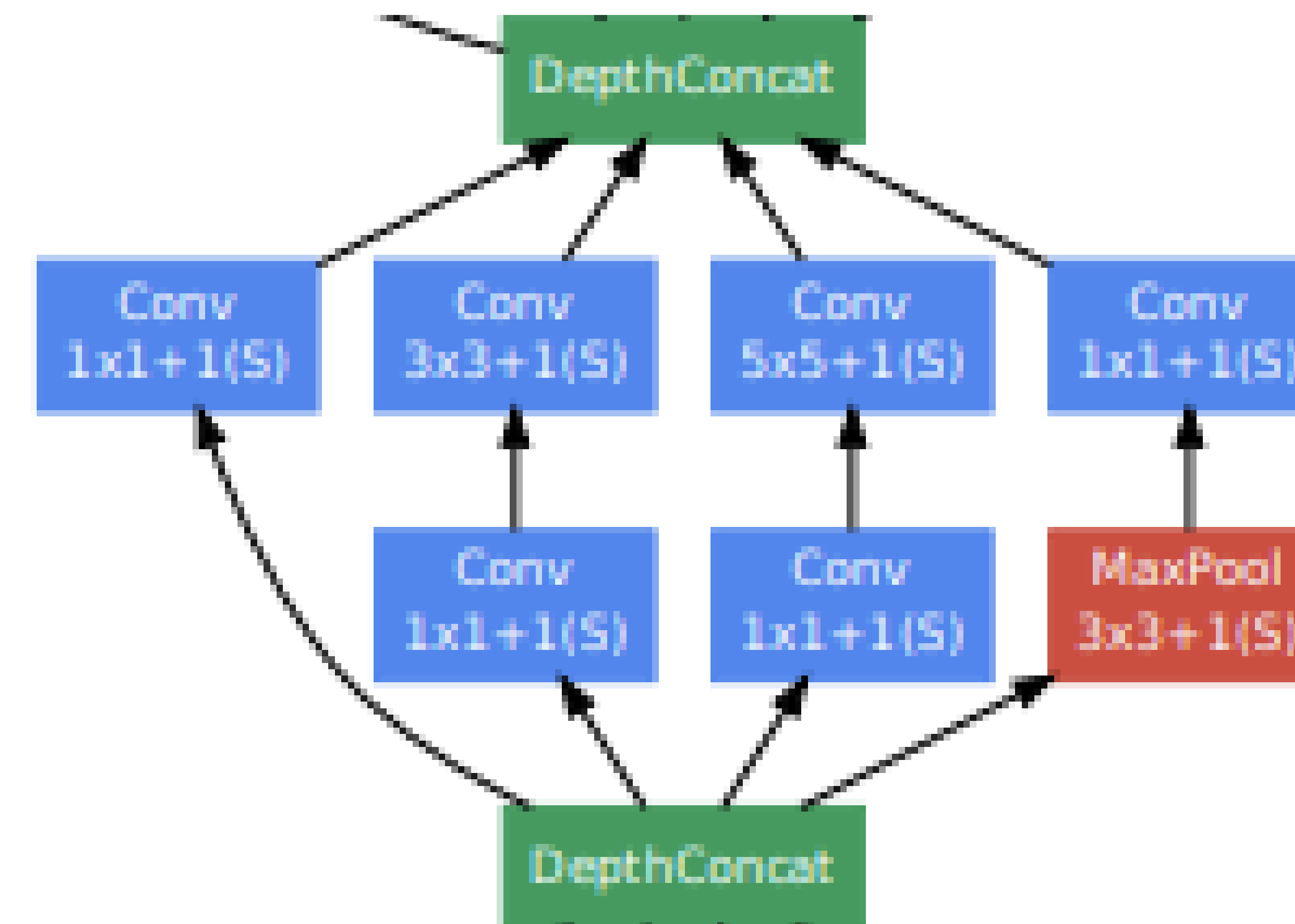
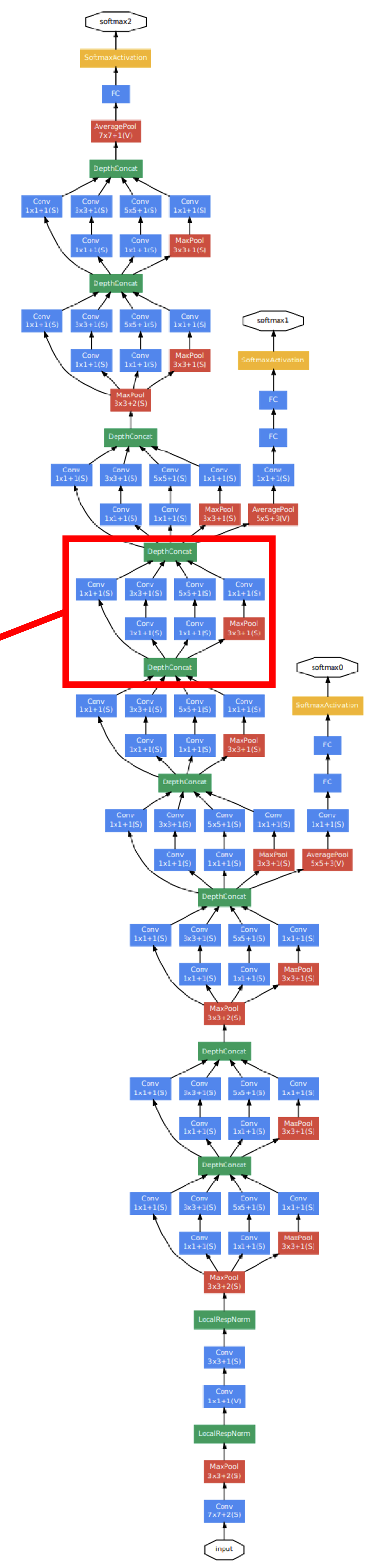


Figure 3: GoogLeNet network with all the bells and whistles

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.

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*

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.

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*

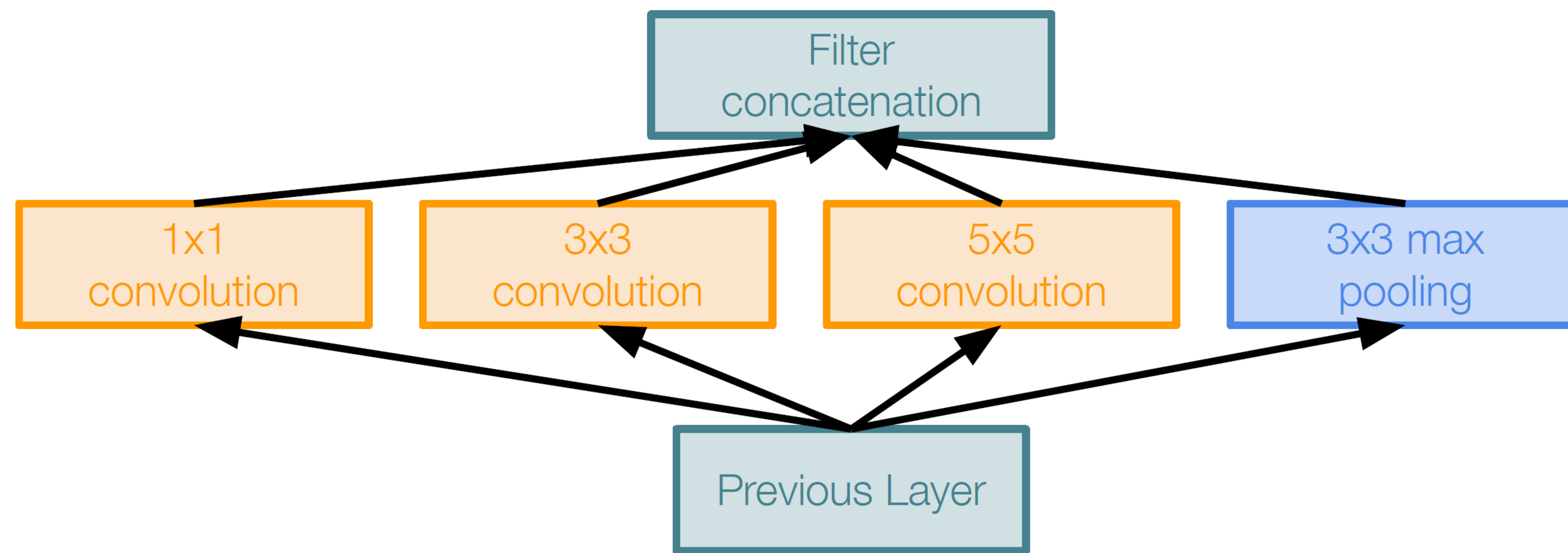
The main contributions of GoogleNet are as follows:

- **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.

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*



Naive Inception module

Apply parallel filter operations on the input from previous layer:

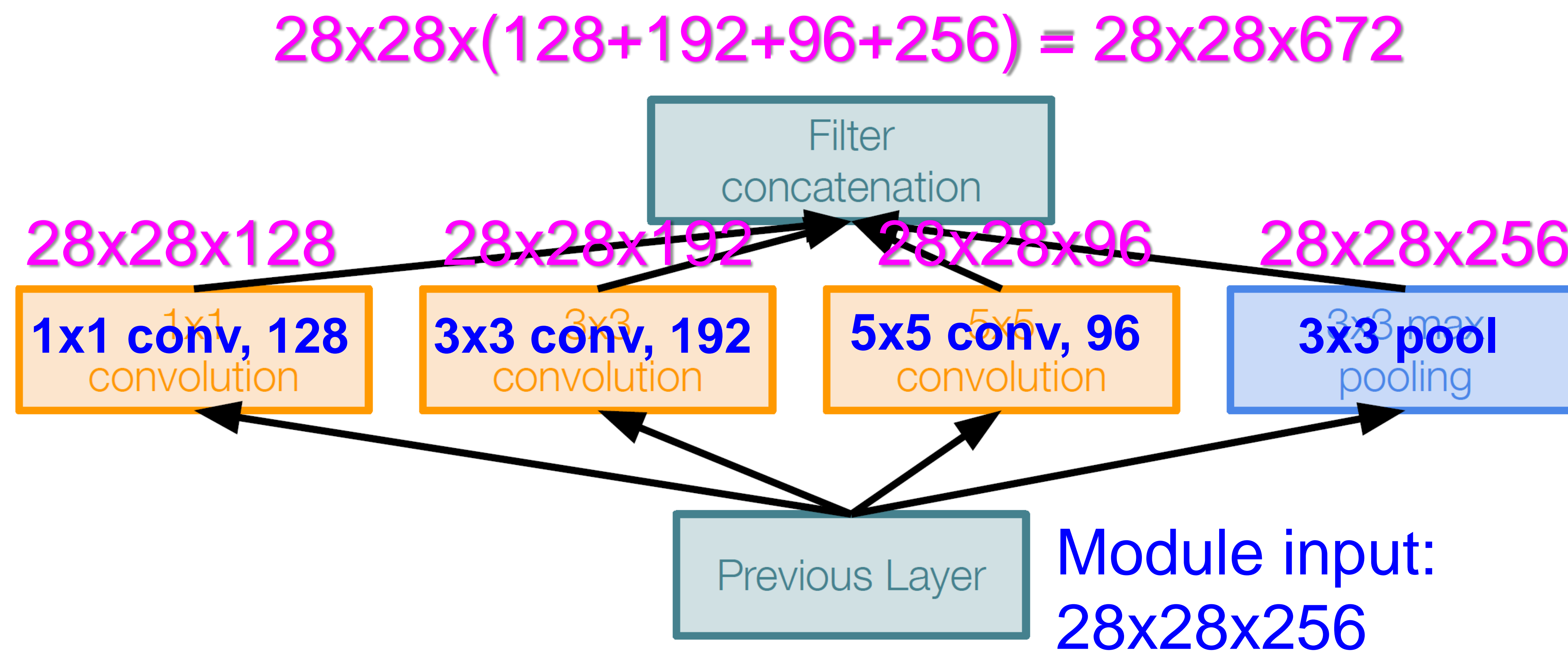
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

Q: What is the problem with this?

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*



Naive Inception module

Q: What is the problem with this?

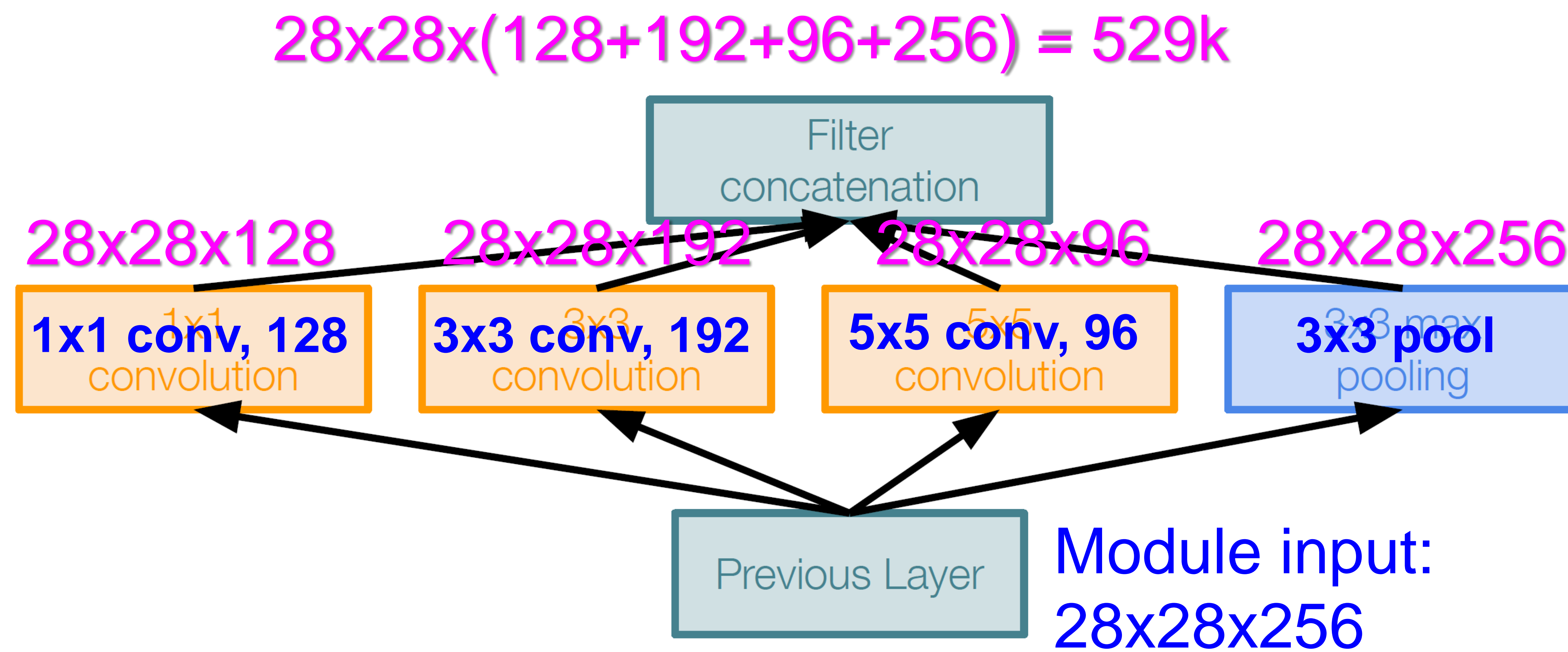
Computational complexity

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*



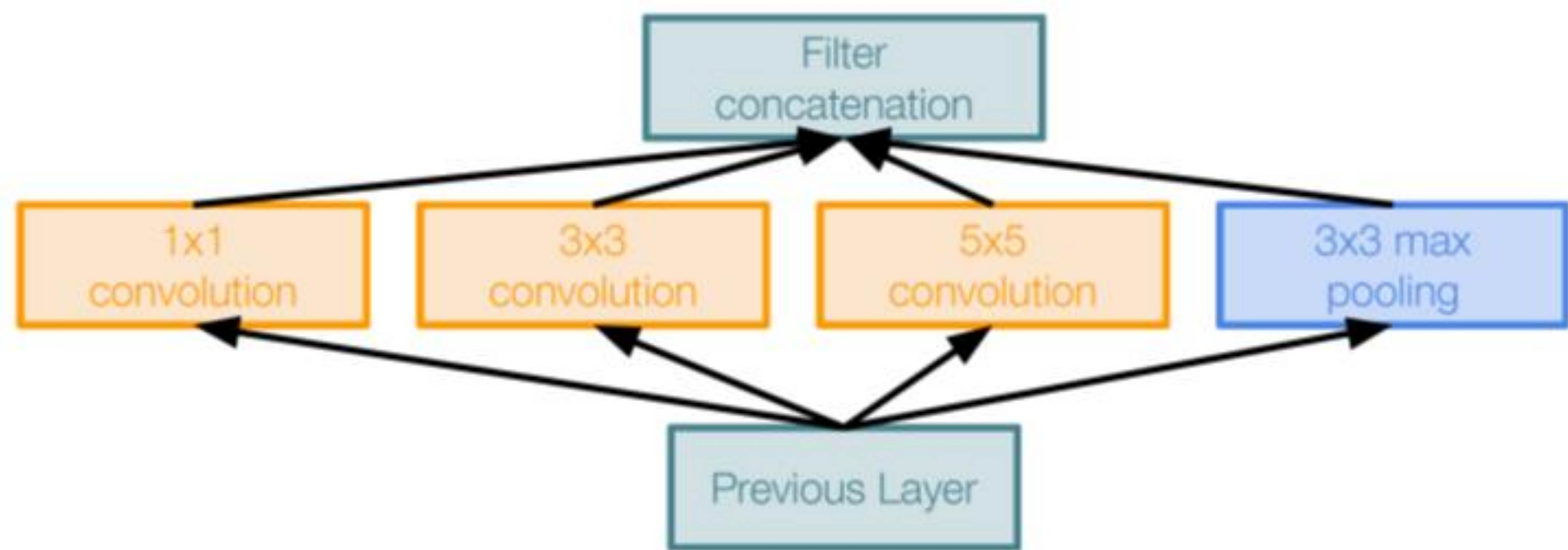
Naive Inception module

Q: What is the problem with this?

Computational complexity

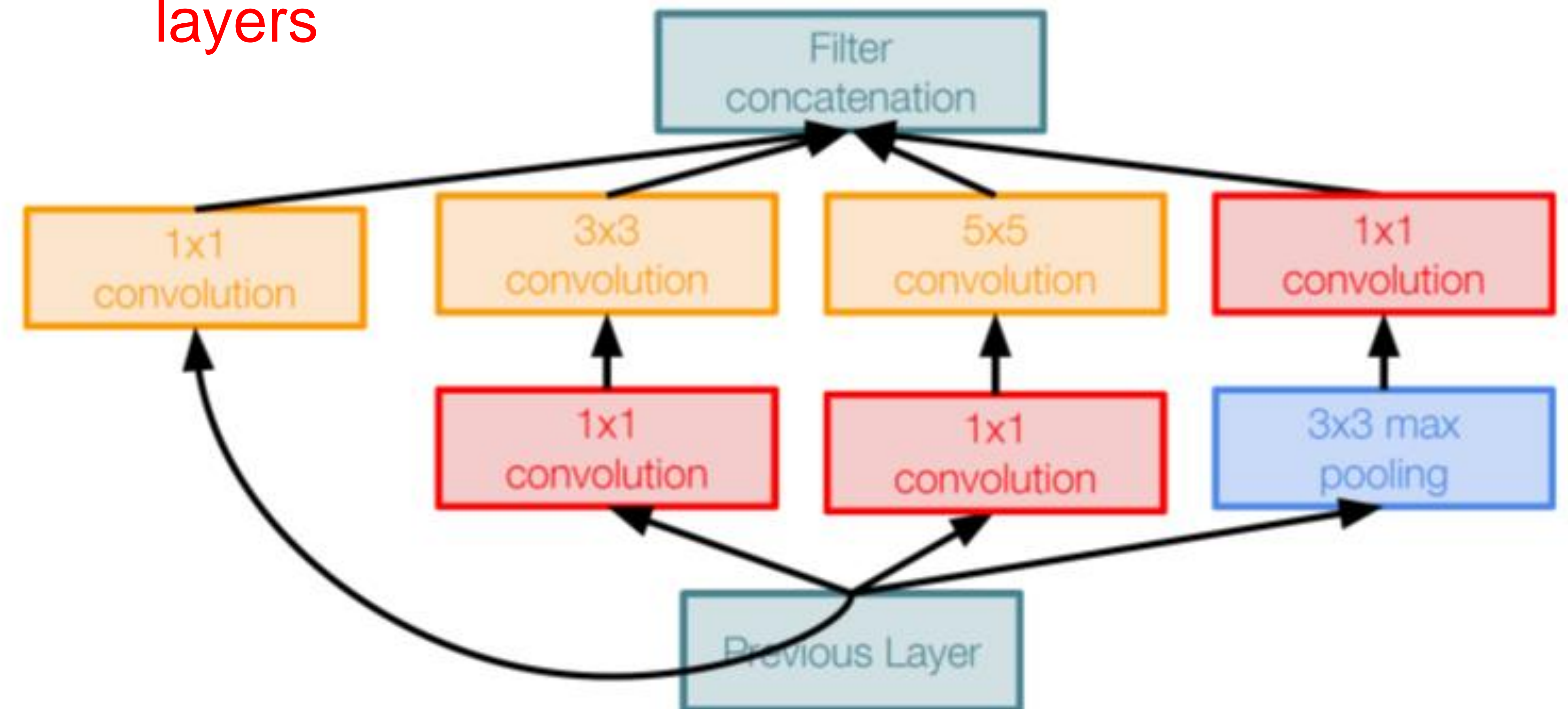
Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature channel size

CNN Architectures: *GoogLeNet*



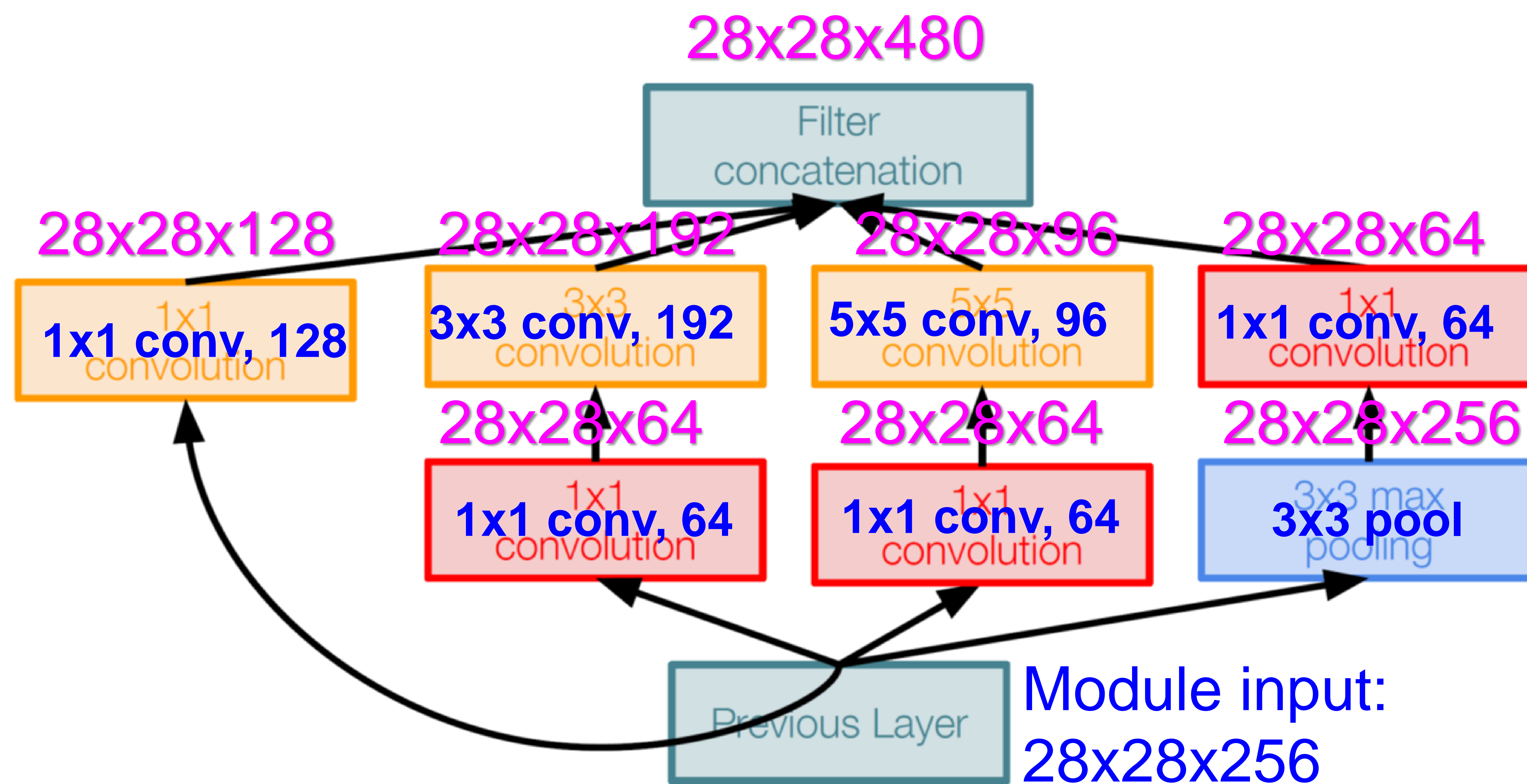
Naive Inception module

1x1 conv "bottleneck" layers



Inception module with dimension reduction

CNN Architectures: *GoogLeNet*



Inception module with dimension reduction

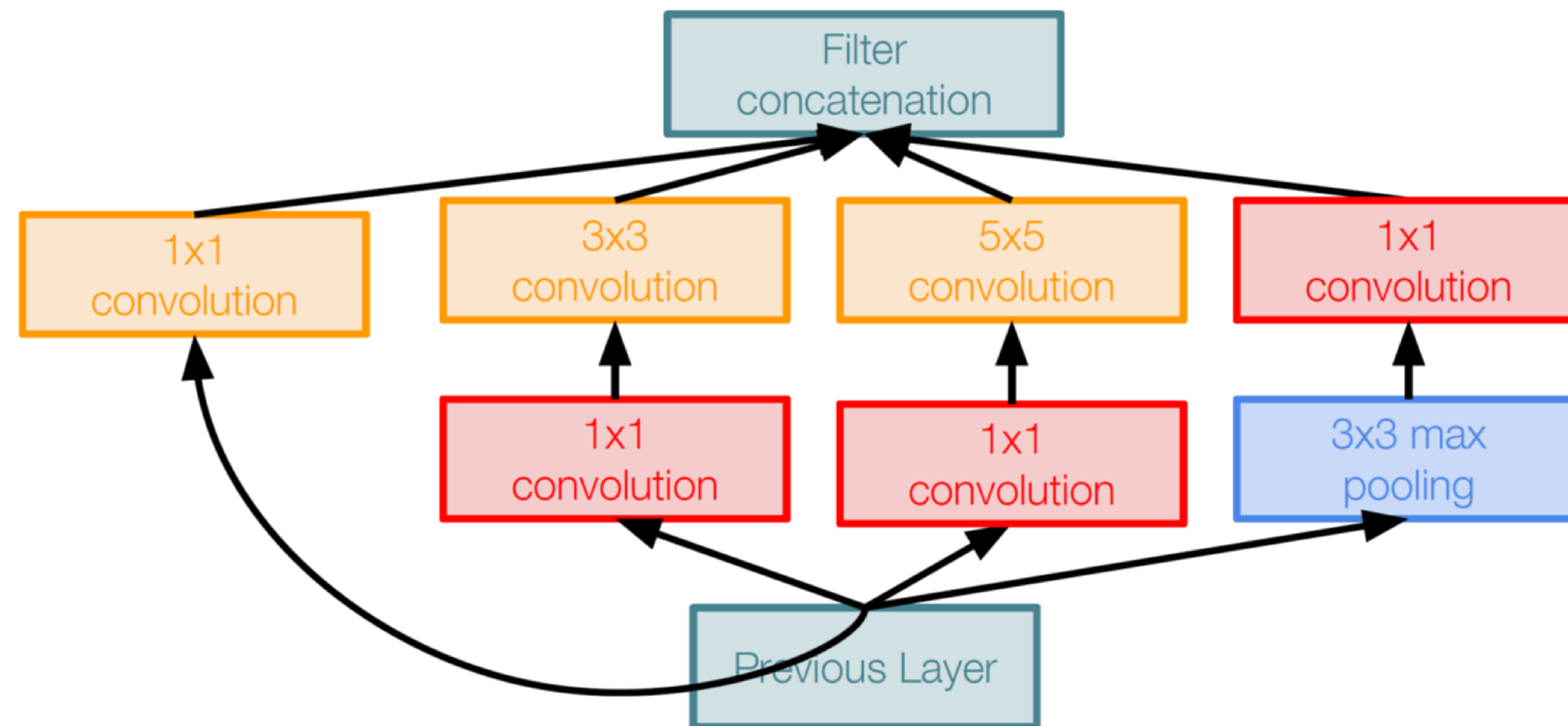
Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

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

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*

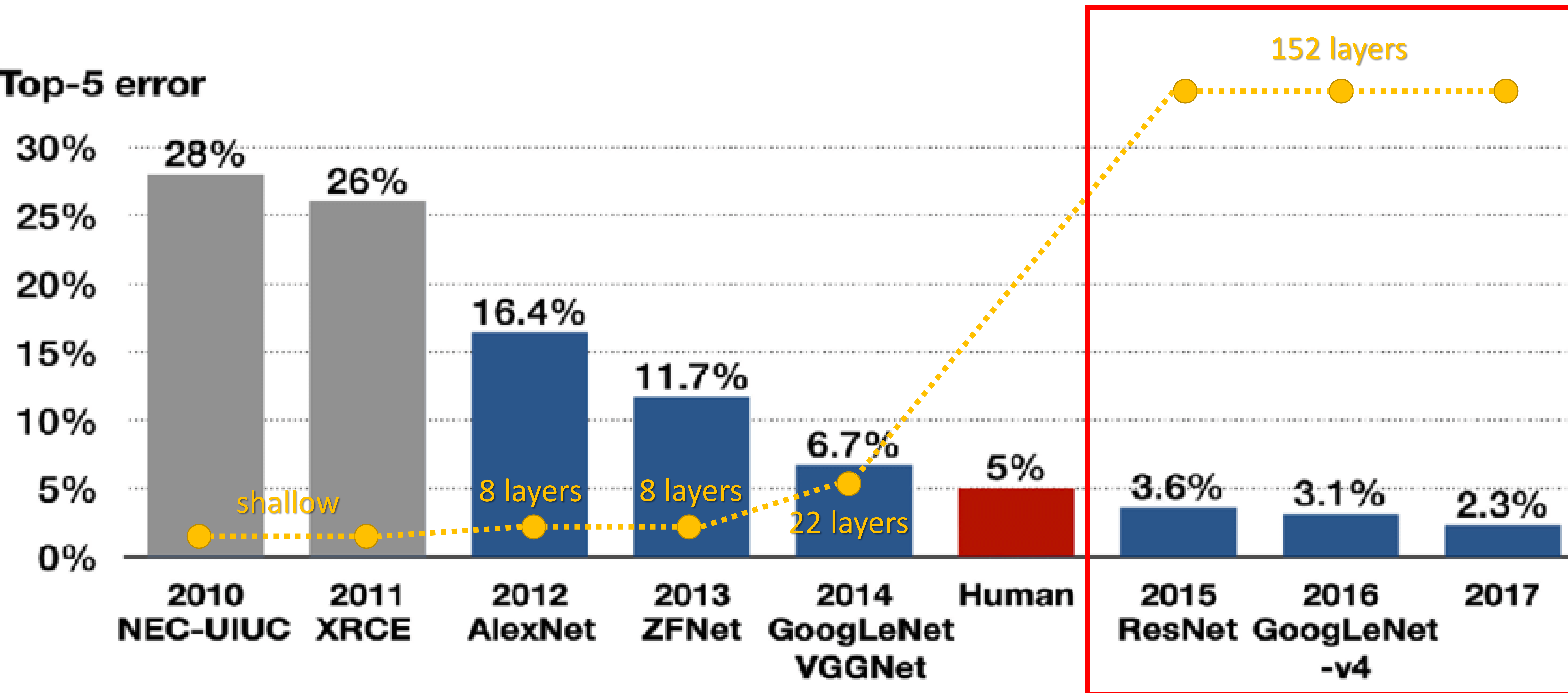


Inception module with dimension reduction

Stack Inception modules with dimension reduction on top of each other

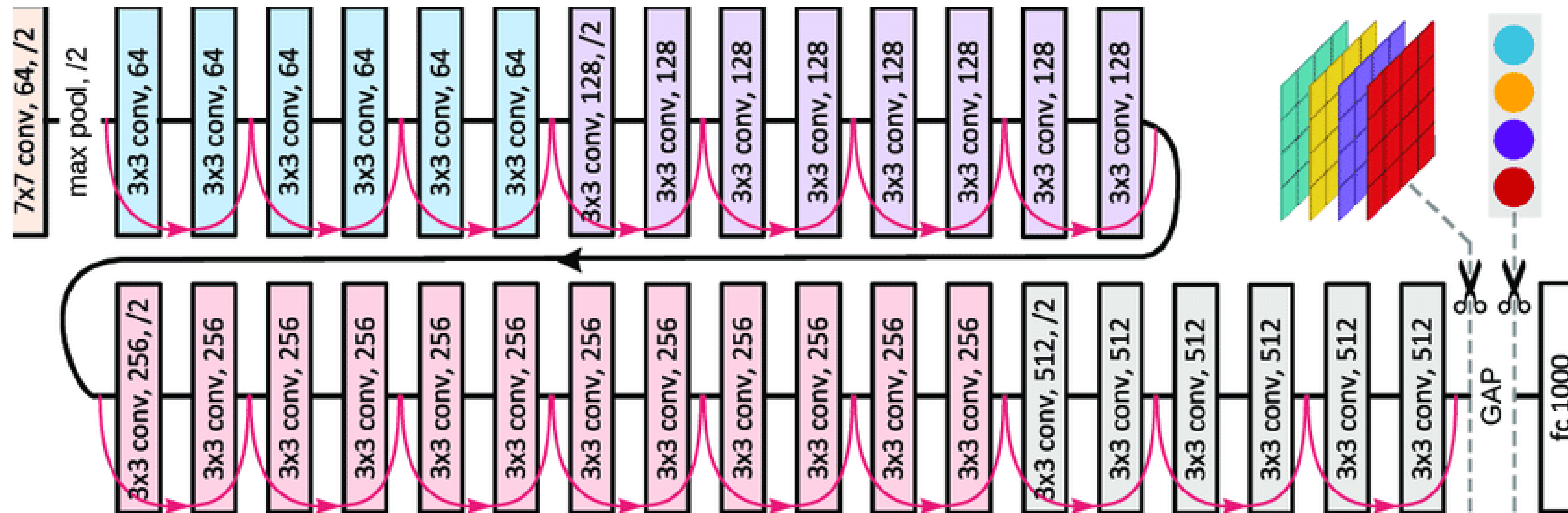
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Revolution of depth

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]

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*

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.

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

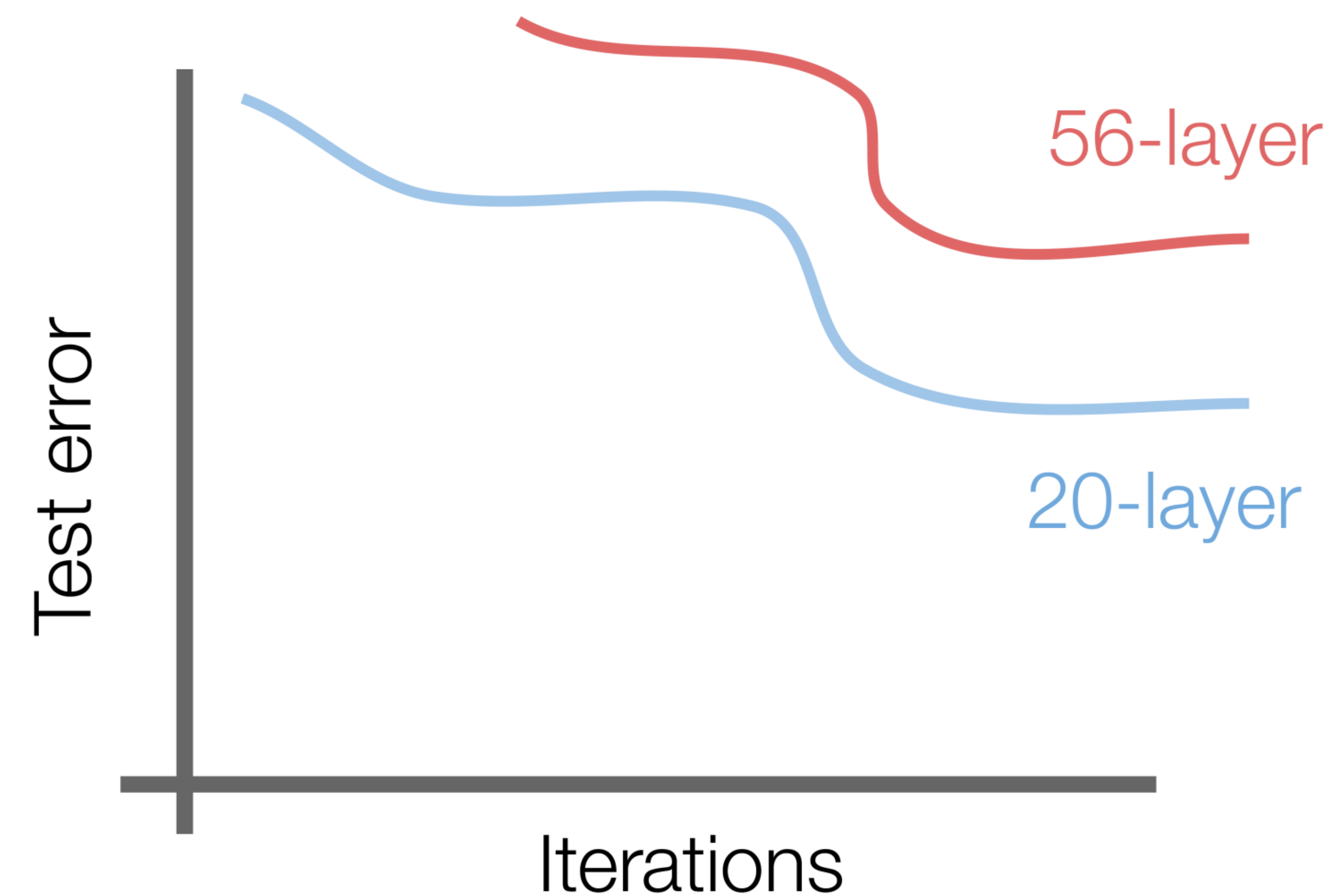
CNN Architectures: *ResNet*

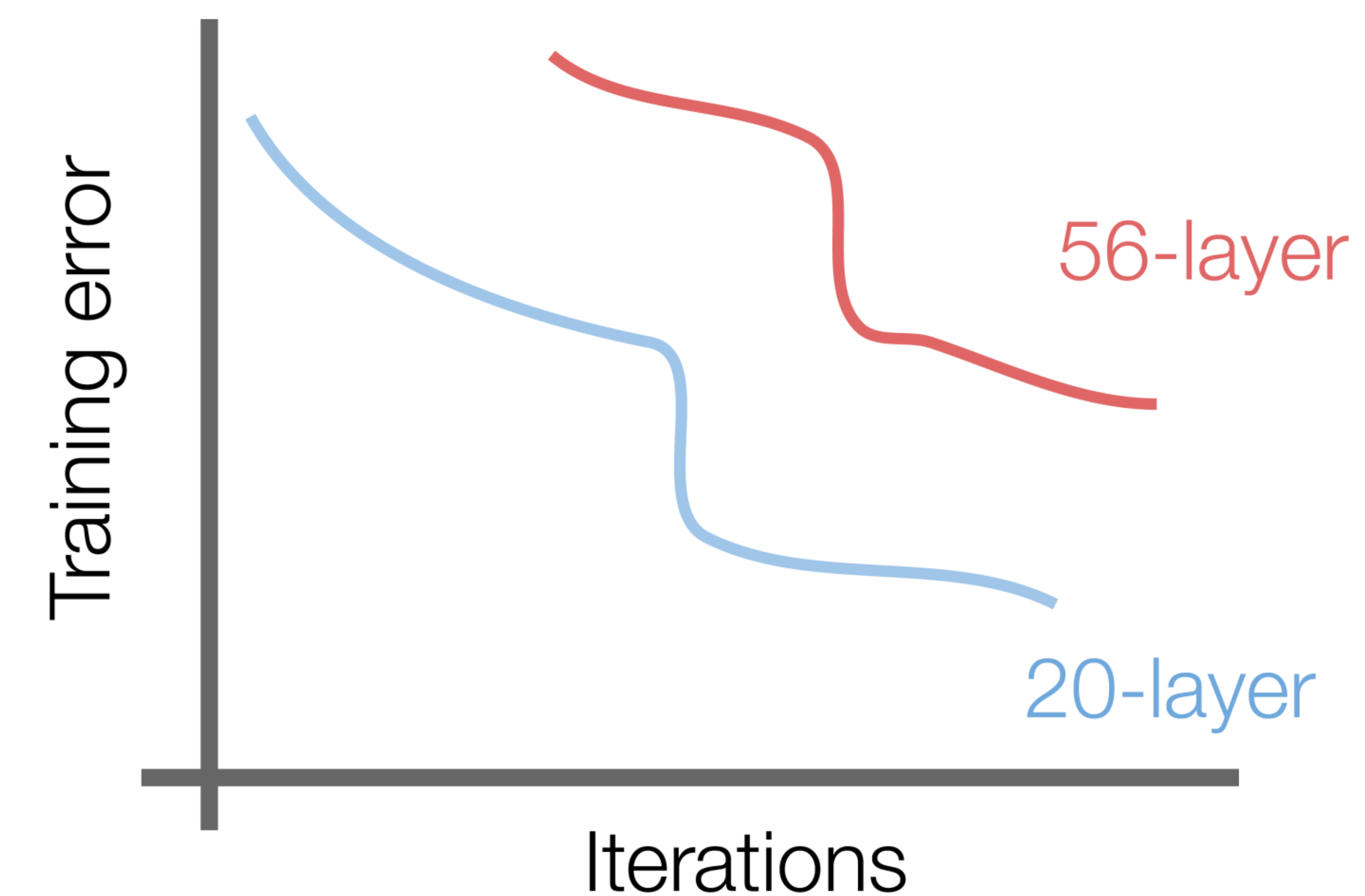
In ResNet, the skip connections allow the output of a layer to be added to the output of one or more layers further along in the network. This creates a "residual" that represents the difference between the input to the layer and its output. By adding this residual to the output of the layer, the network 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!

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

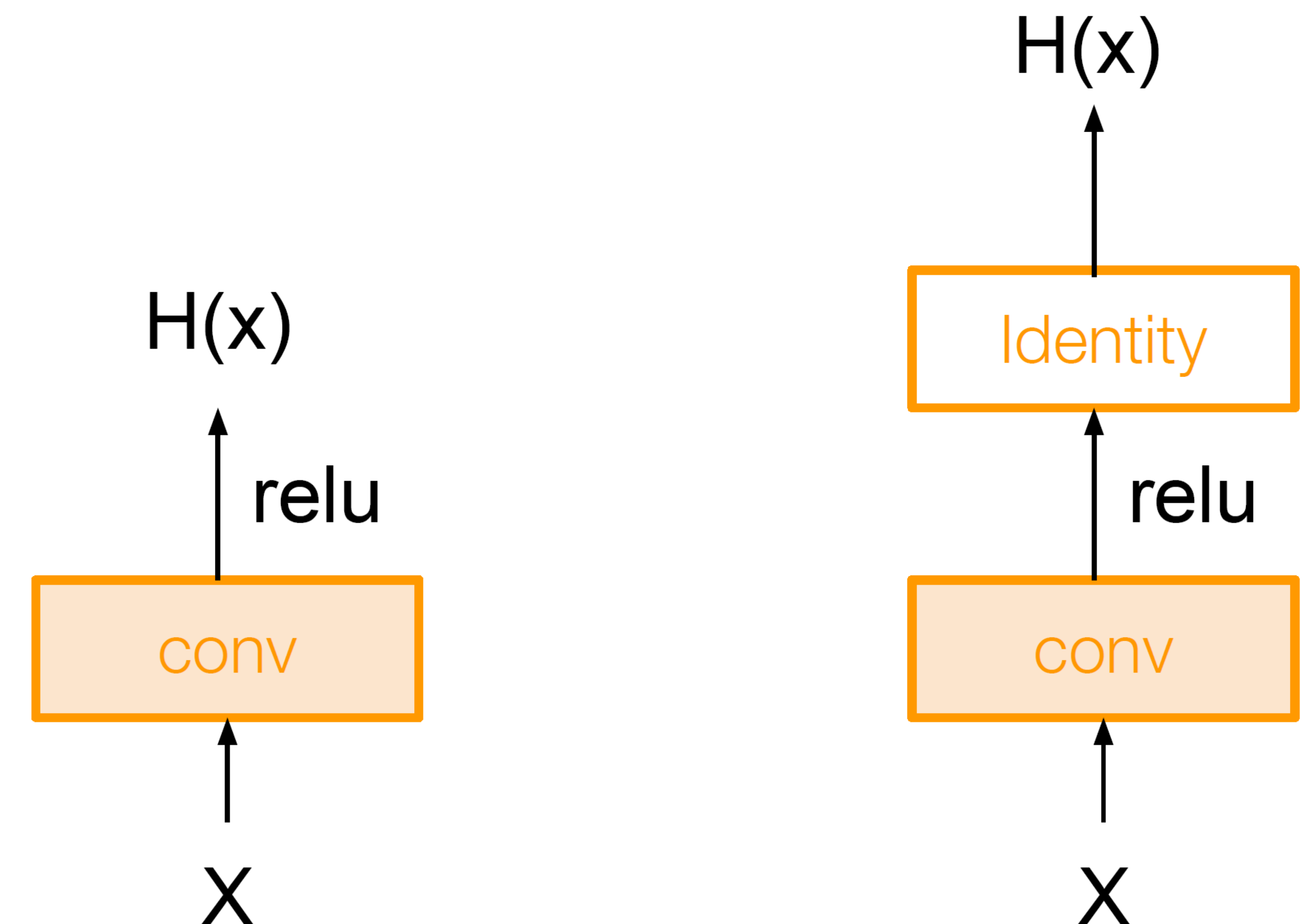
CNN Architectures: *ResNet*

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**

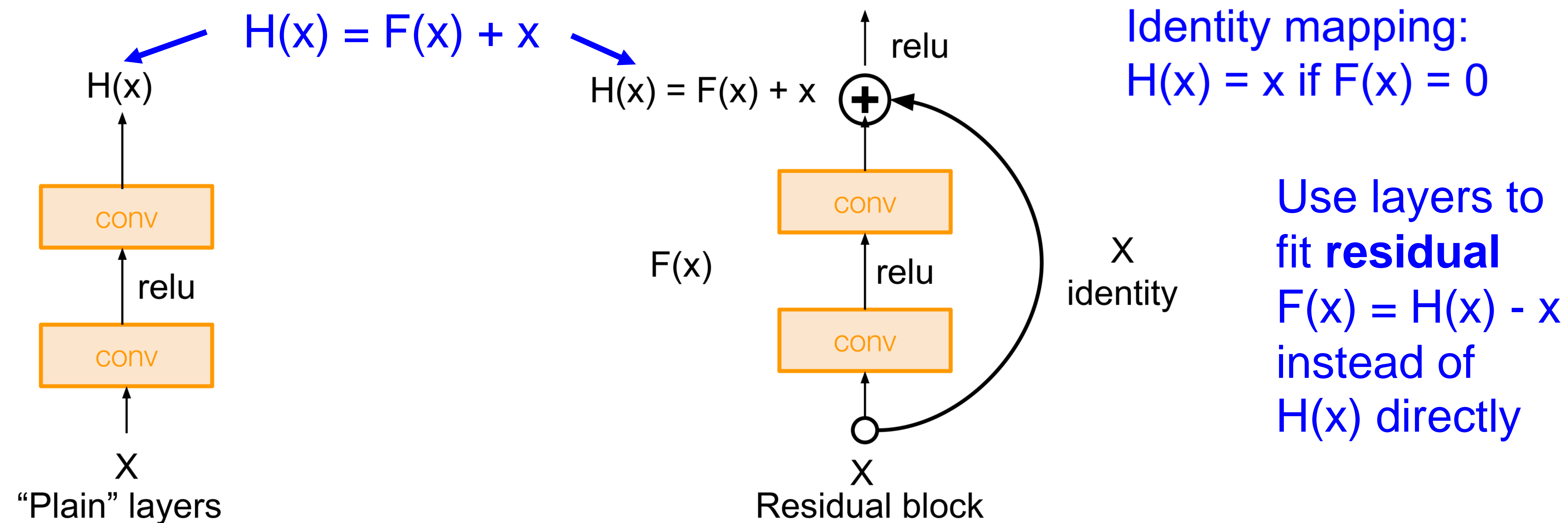
What should the deeper model learn to be at least as good as the shallower model?

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



CNN Architectures: *ResNet*

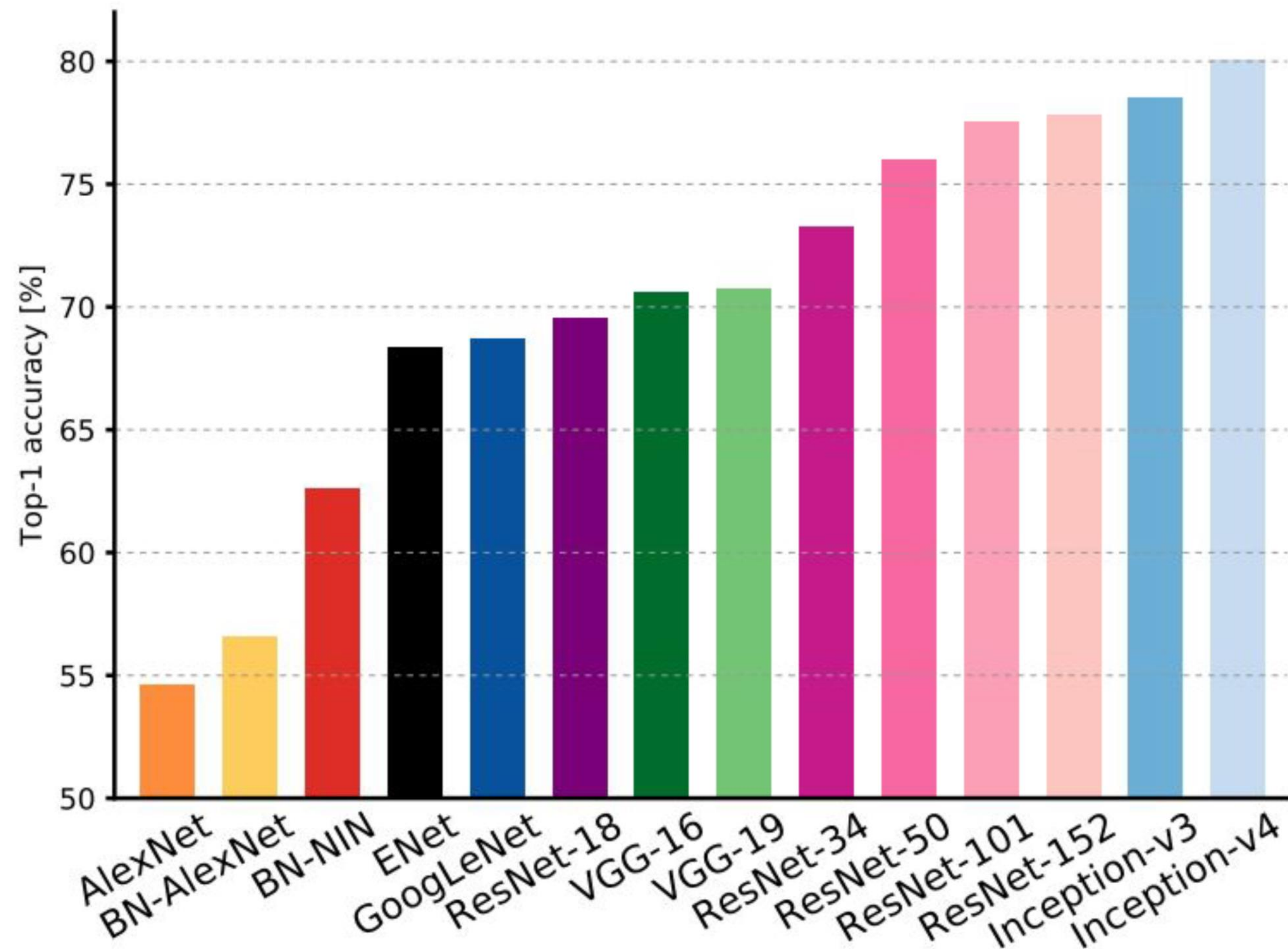
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

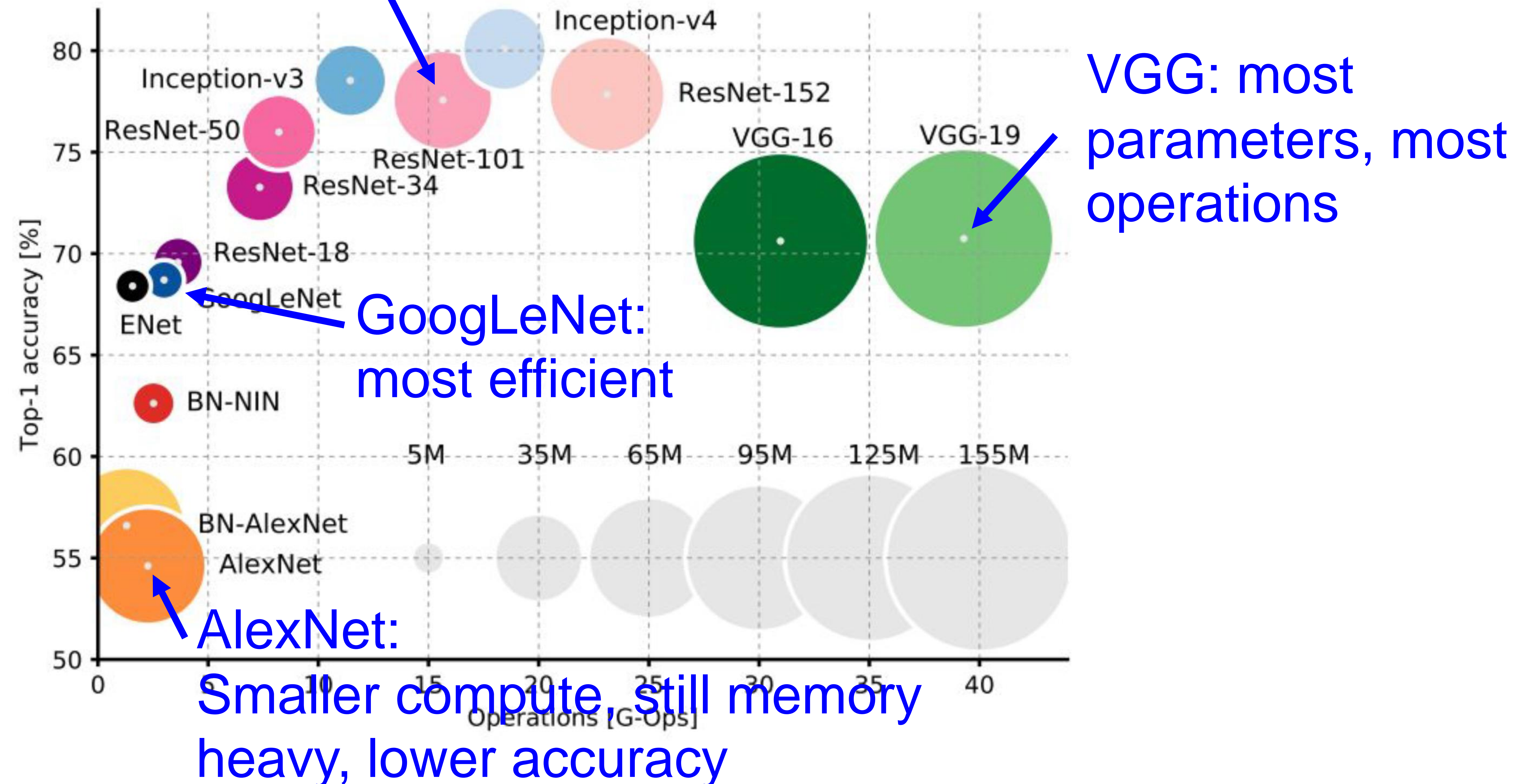


Identity mapping:
H(x) = x if F(x) = 0

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

CNN Architectures: *Comparing complexity...*





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

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