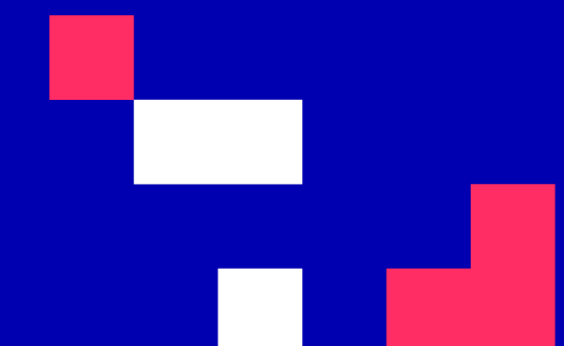
MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

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

CNN Architectures: *Training Neural Networks*

Overview

1. **One time set up**: activation functions, preprocessing, weight initialization, regularization, gradient checking
2. **Training dynamics**: babysitting the learning process, parameter updates, hyperparameter optimization
3. **Evaluation**: model ensembles, test-time augmentation, transfer learning

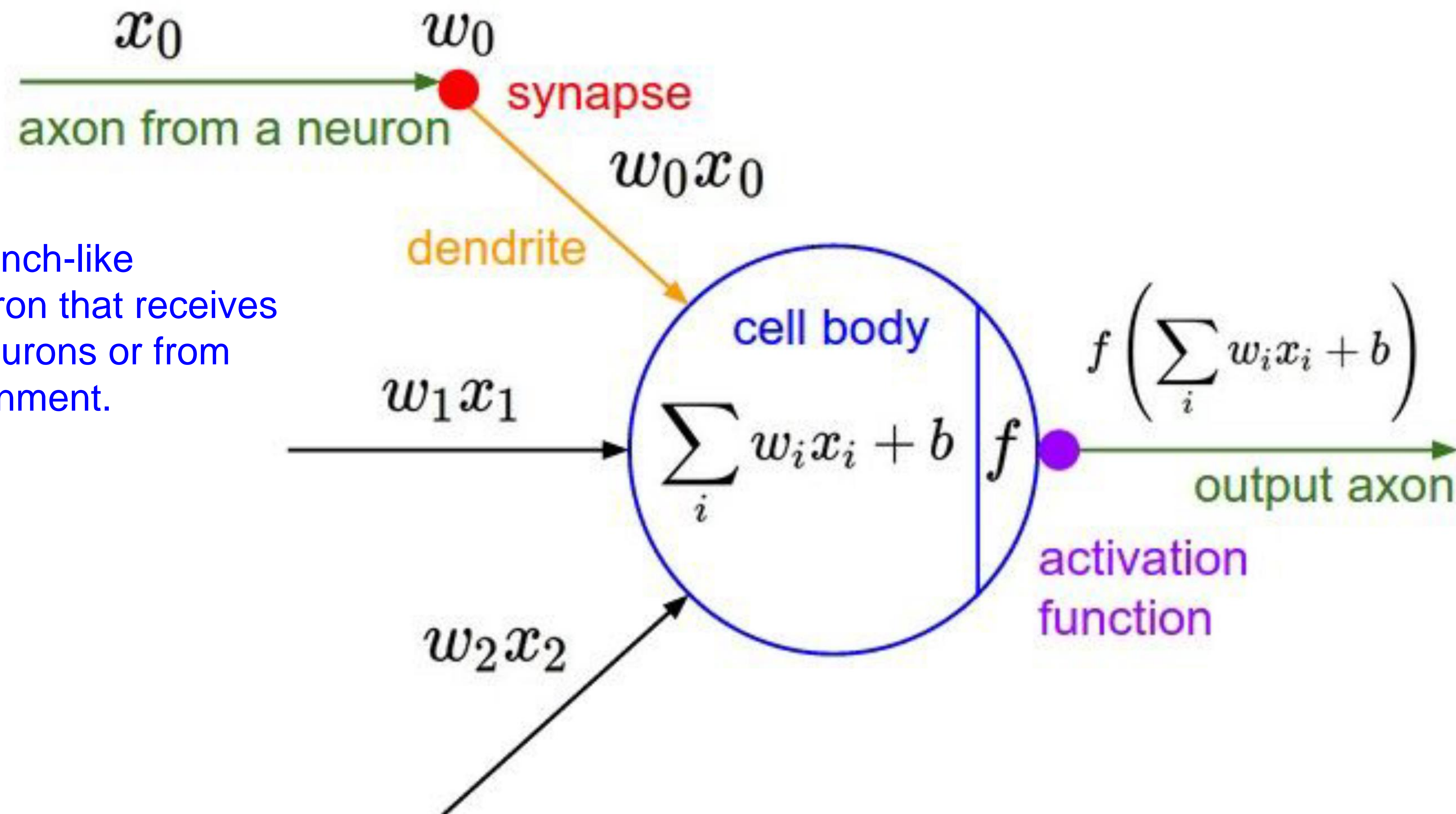
Co-financed by the European Union
Connecting Europe Facility

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

Activation function

A **synapse** is a structure that allows information to flow from one neuron to another in a neural network.

A **dendrite** is a branch-like extension of a neuron that receives input from other neurons or from the external environment.



Co-financed by t
Connecting Europe Facility
r the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

Activation function

The **dendrites** are a critical component of the neural network, as they receive input from other neurons and from the external environment, which is then transformed by the activation function to produce the neuron's output. The behavior of the dendrites, along with the activation function and the synapses, determines the overall behavior and output of the neural network.

The **synapse** in the activation function is a crucial component of the neural network, as it determines how much influence each input has on the output of the neuron, and ultimately, the output of the network as a whole.

Activation functions are used to introduce nonlinearity into neural networks, allowing them to model complex relationships between inputs and outputs. When a neuron receives input from other neurons, it computes a weighted sum of those inputs, which is then passed through the activation function. The output of the activation function is the neuron's output, which is then passed on to other neurons via synapses.

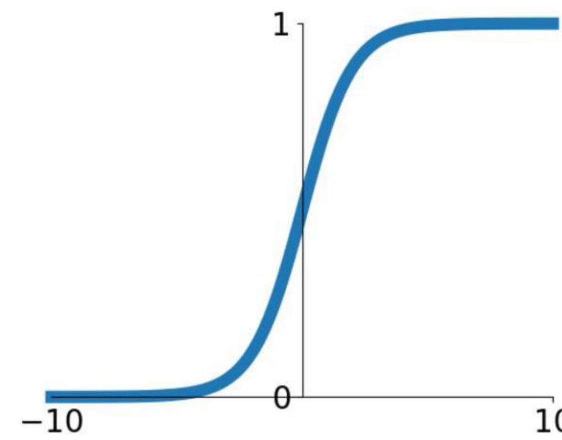
Co-financed by the European Union
Connecting Europe Facility

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

Activation function

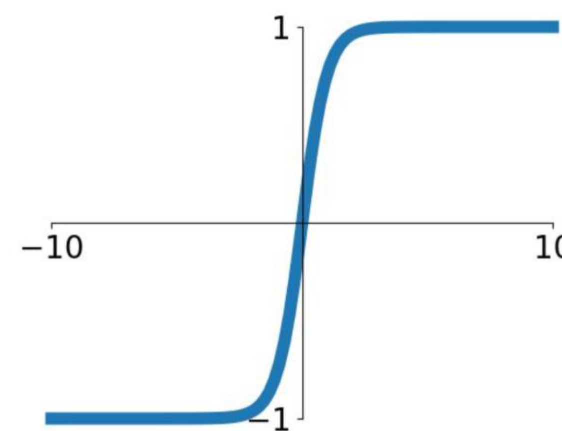
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



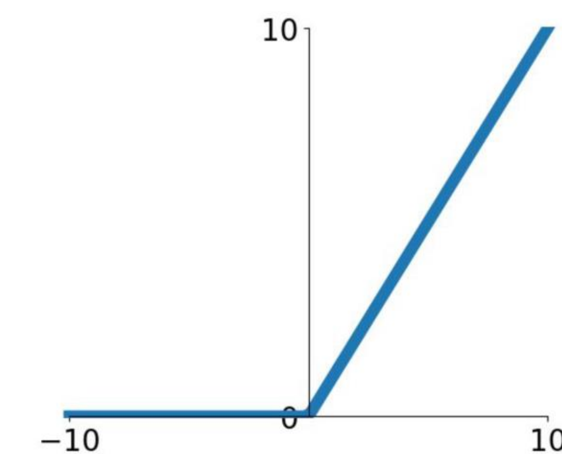
tanh

$\tanh(x)$



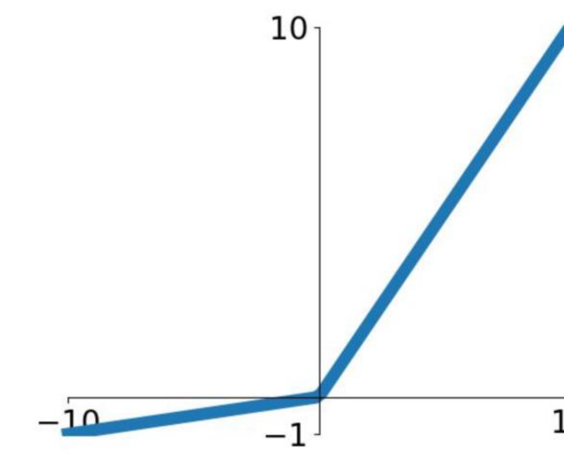
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

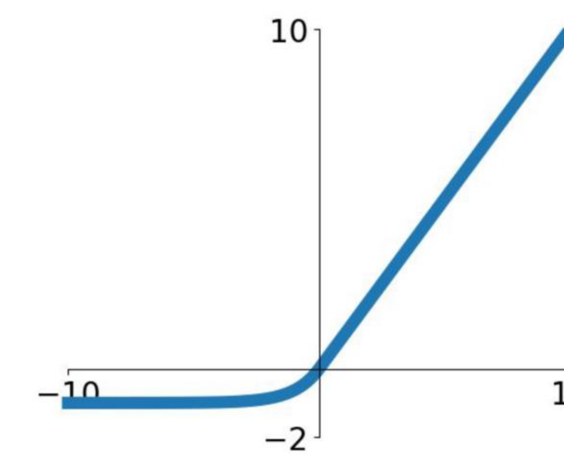


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



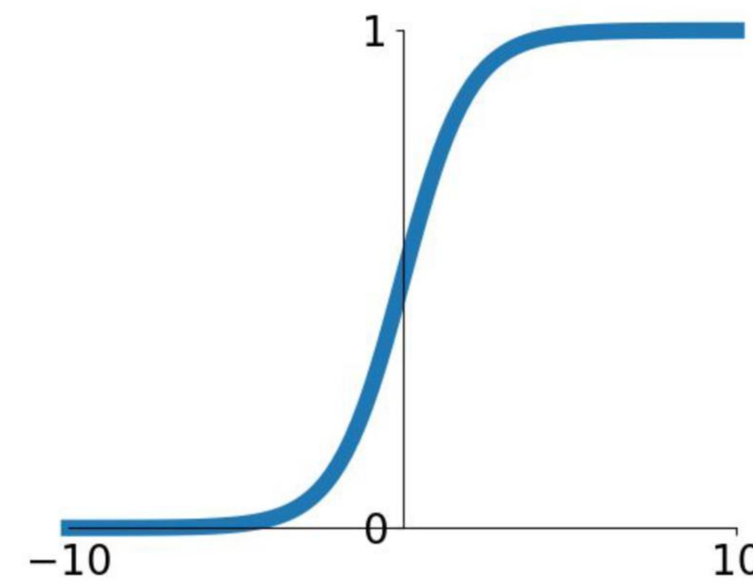
Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it.

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

Activation function: *Sigmoid*

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



The sigmoid non-linearity takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).

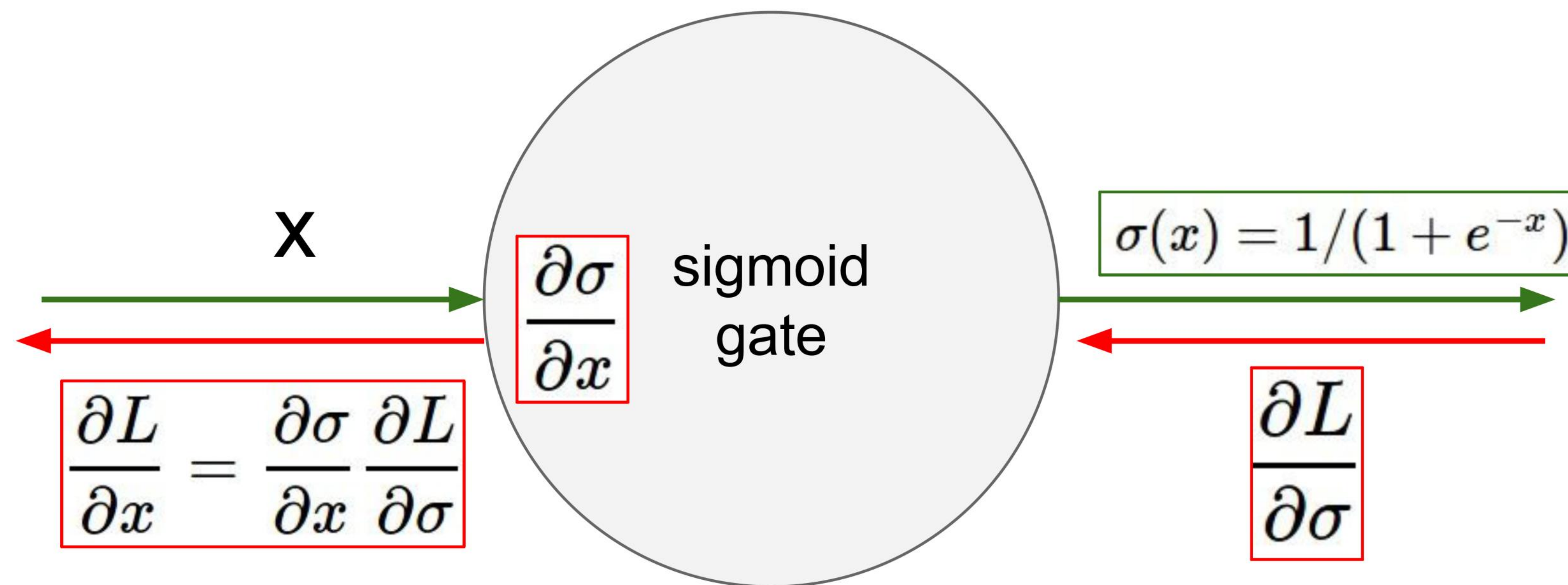
Main Drawbacks

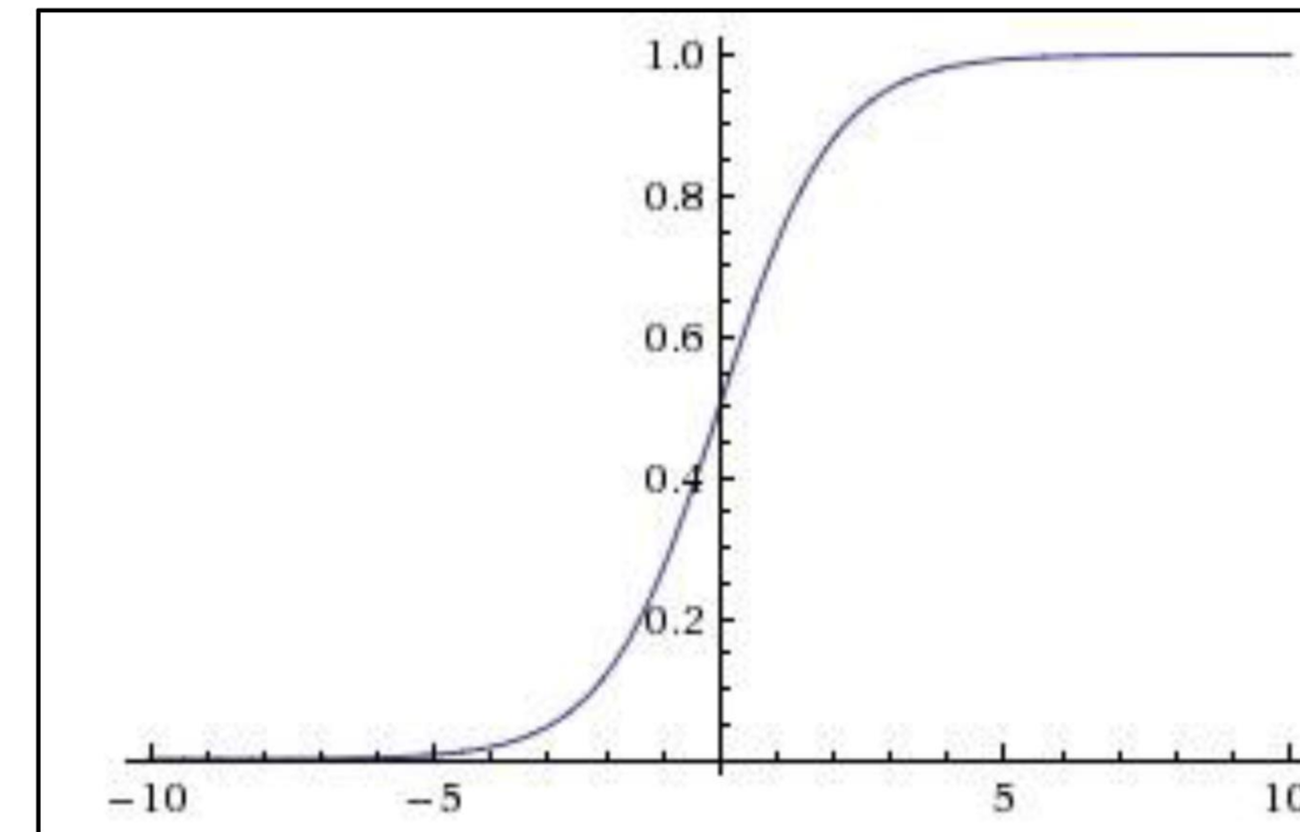
1. ***Sigmoids saturate and kill gradients.*** A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate's output for the whole objective. Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. For example, if the initial weights are too large then most neurons would become saturated and the network will barely 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

Activation function: *Sigmoid*





What happens when x = -10?

$$\sigma(x) = \sim 0$$

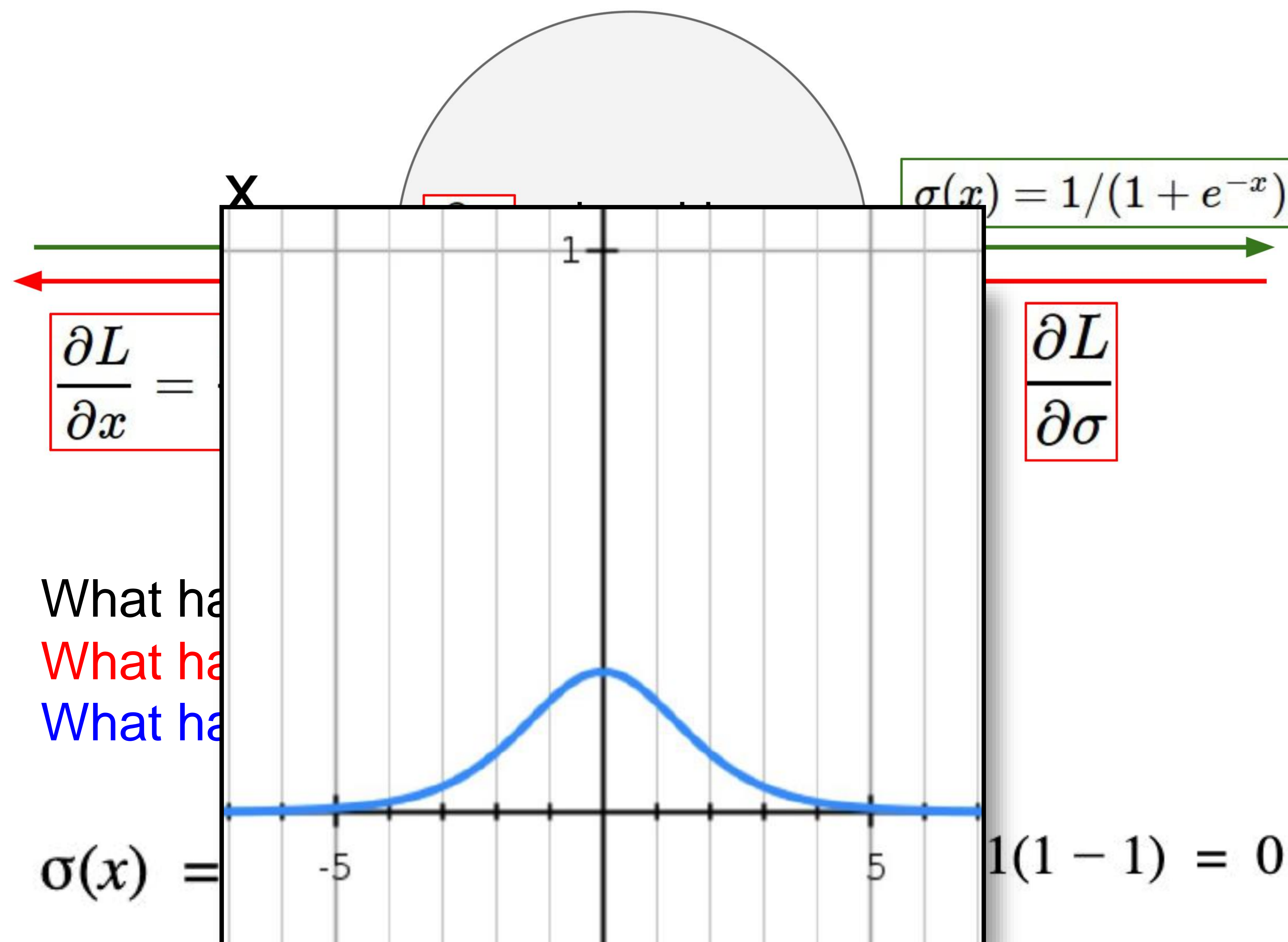
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x)\,) = 0(1-0) = 0$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x)\,)$$

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

Activation function: *Sigmoid*

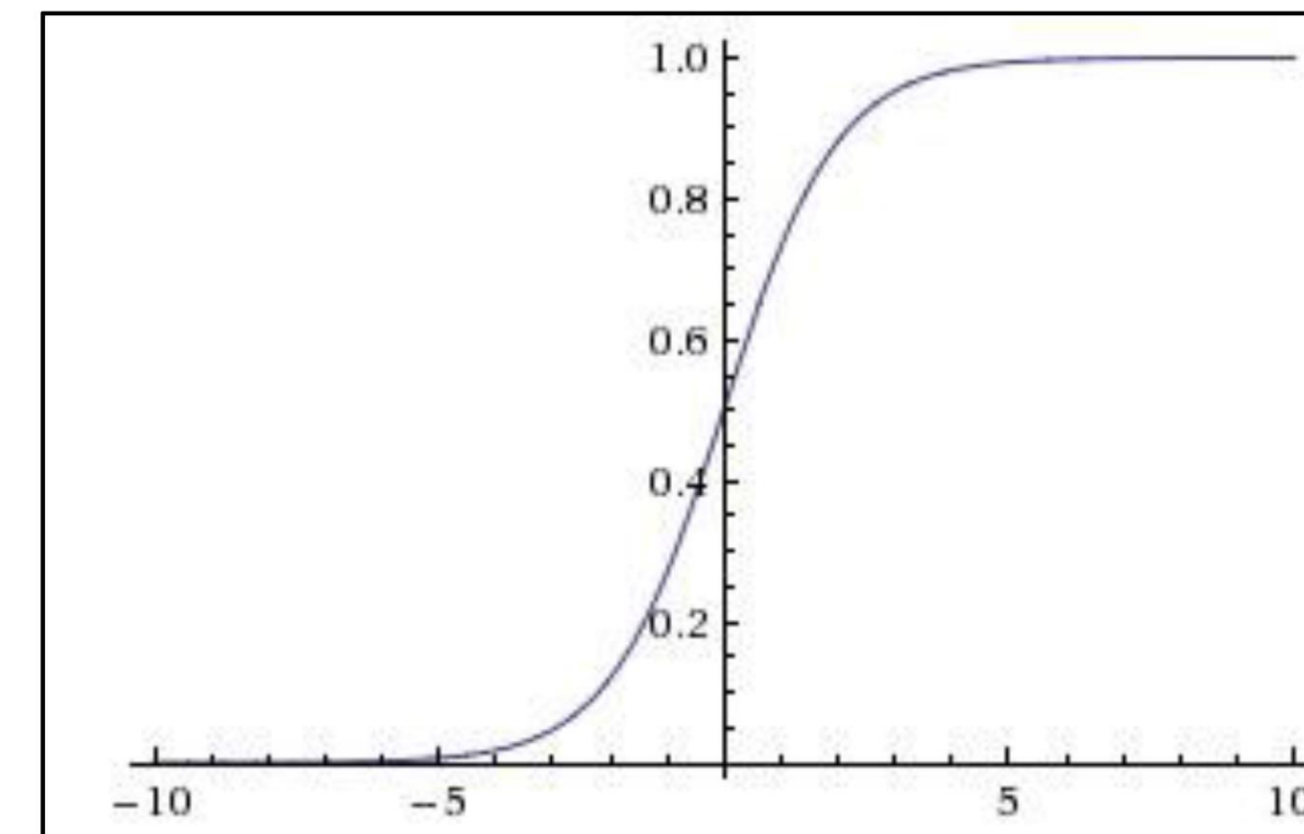


$$\frac{\partial L}{\partial x} =$$

$$\sigma(x) = 1/(1+e^{-x})$$

$$\frac{\partial L}{\partial \sigma}$$

$$\sigma(x) = \quad 1(1-1) = 0$$

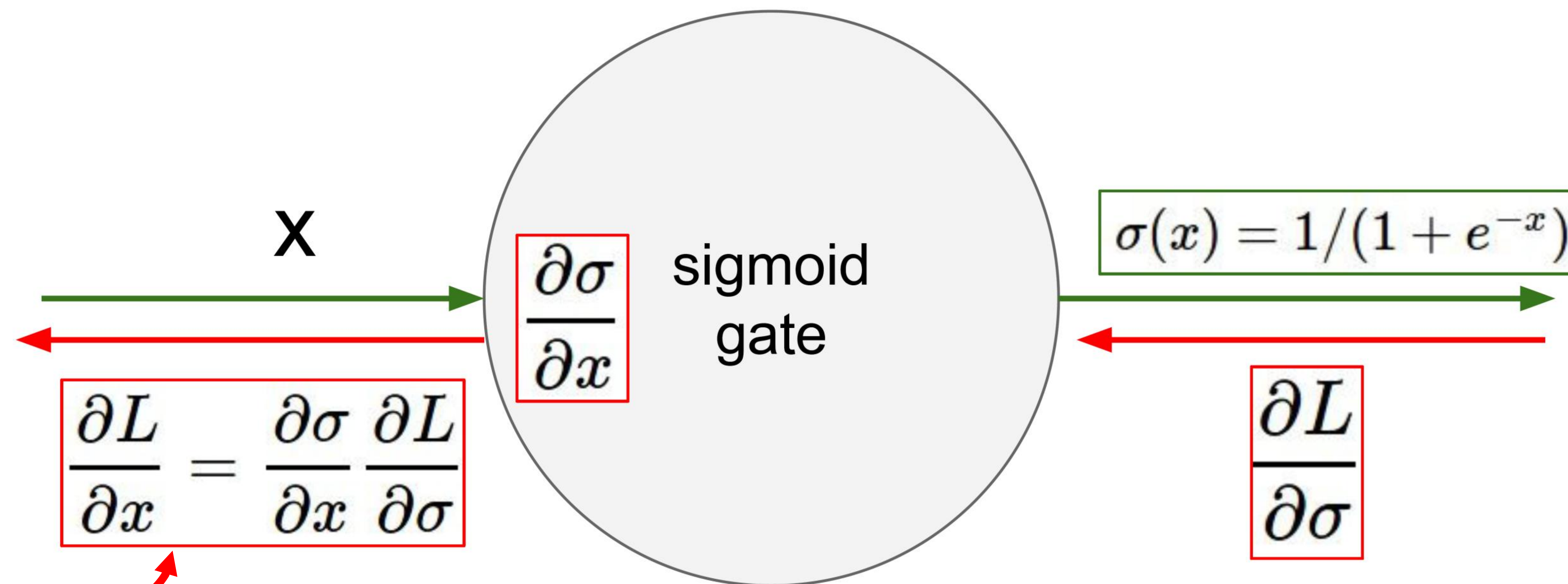


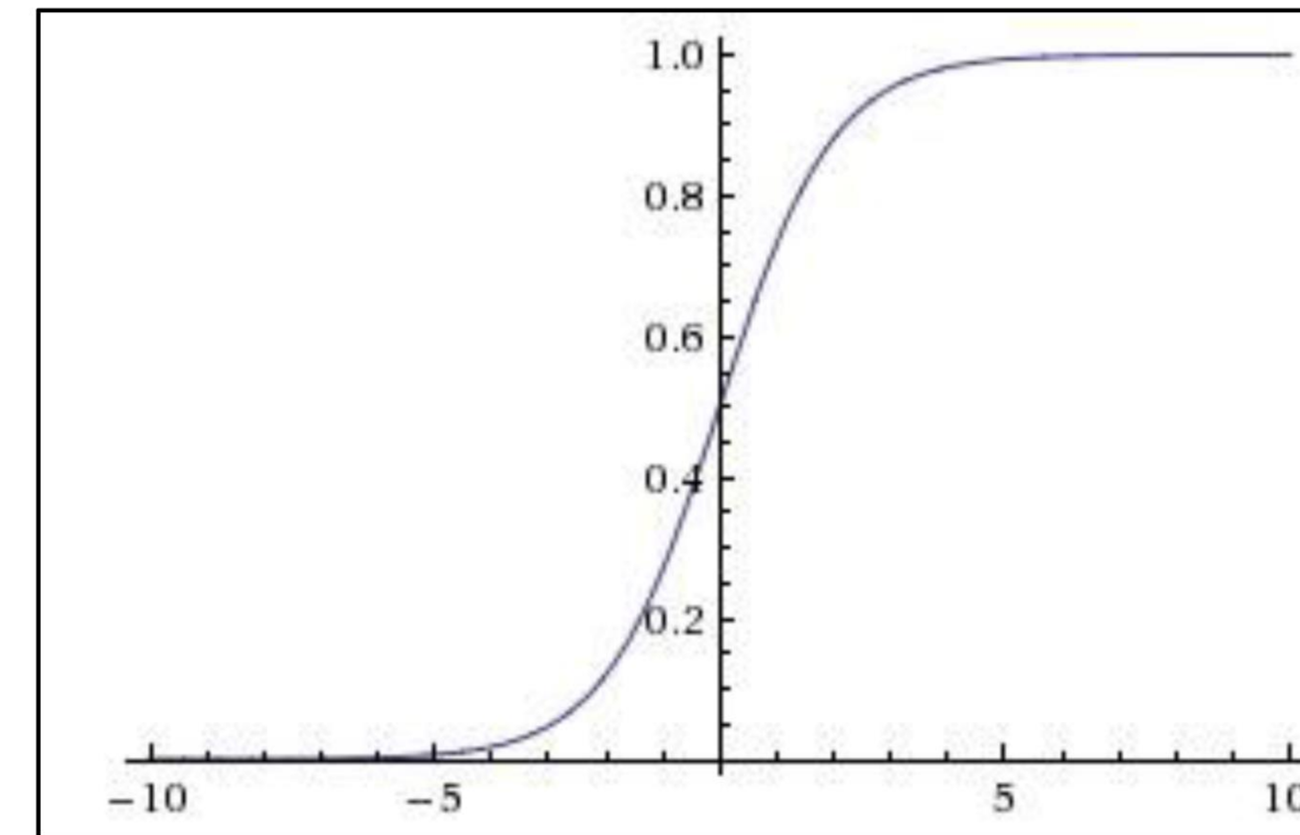
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x)\,)$$

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

Activation function: *Sigmoid*





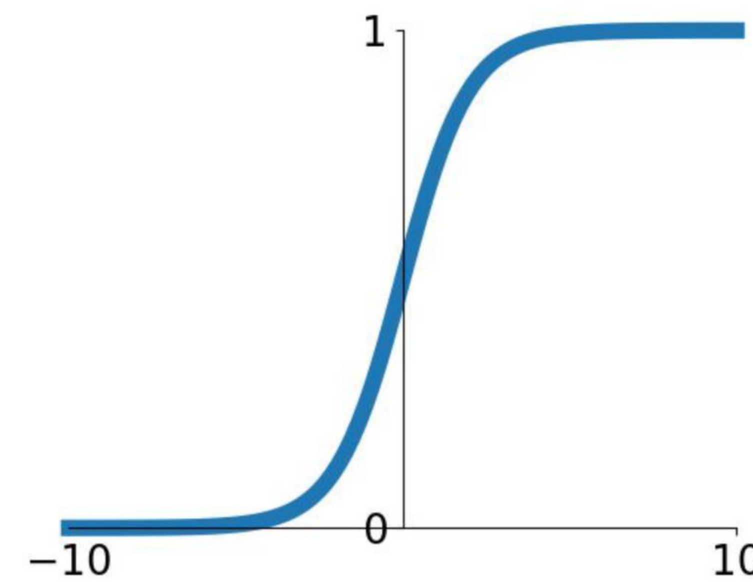
Why is this a problem?
If all the gradients flowing back will be zero and weights will never change

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$

Activation function: *Sigmoid*

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



The sigmoid non-linearity takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).

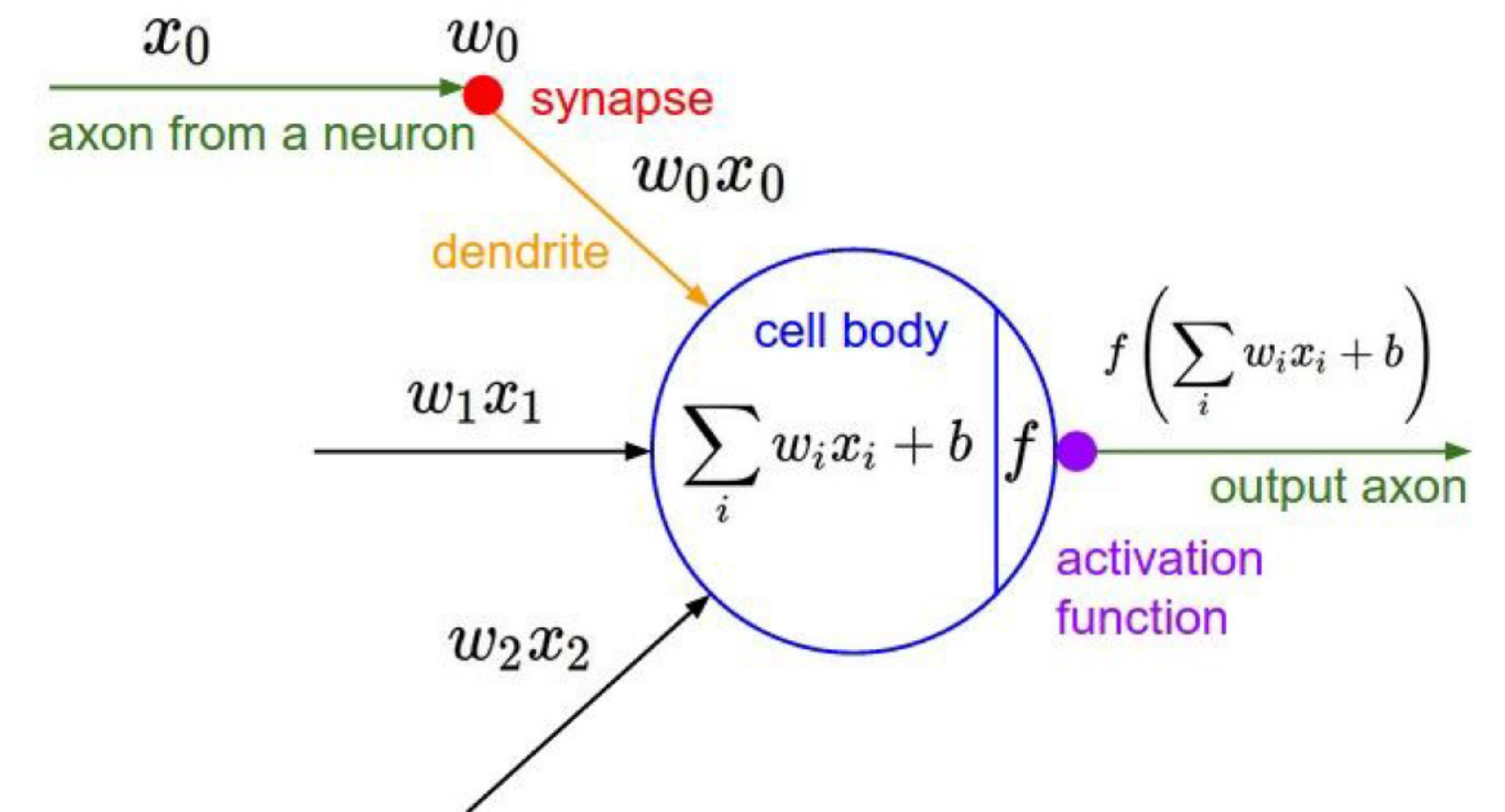
Main Drawbacks

2. ***Sigmoid outputs are not zero-centered.*** This is undesirable since neurons in later layers of processing in a Neural Network would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive, then the gradient on the weights *w* will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression *f*). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

Activation function: *Sigmoid*

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on ***w***?

We know that local gradient of sigmoid is always positive

We are assuming x is always positive

So!! Sign of gradient **for all w_i** is the same as the sign of upstream scalar gradient!

$$\frac{\partial L}{\partial w} = \sigma(\textstyle\sum_i w_i x_i + b)(1 - \sigma(\textstyle\sum_i w_i x_i + b))x \times upstream_gradient$$

Activation function: *Sigmoid*

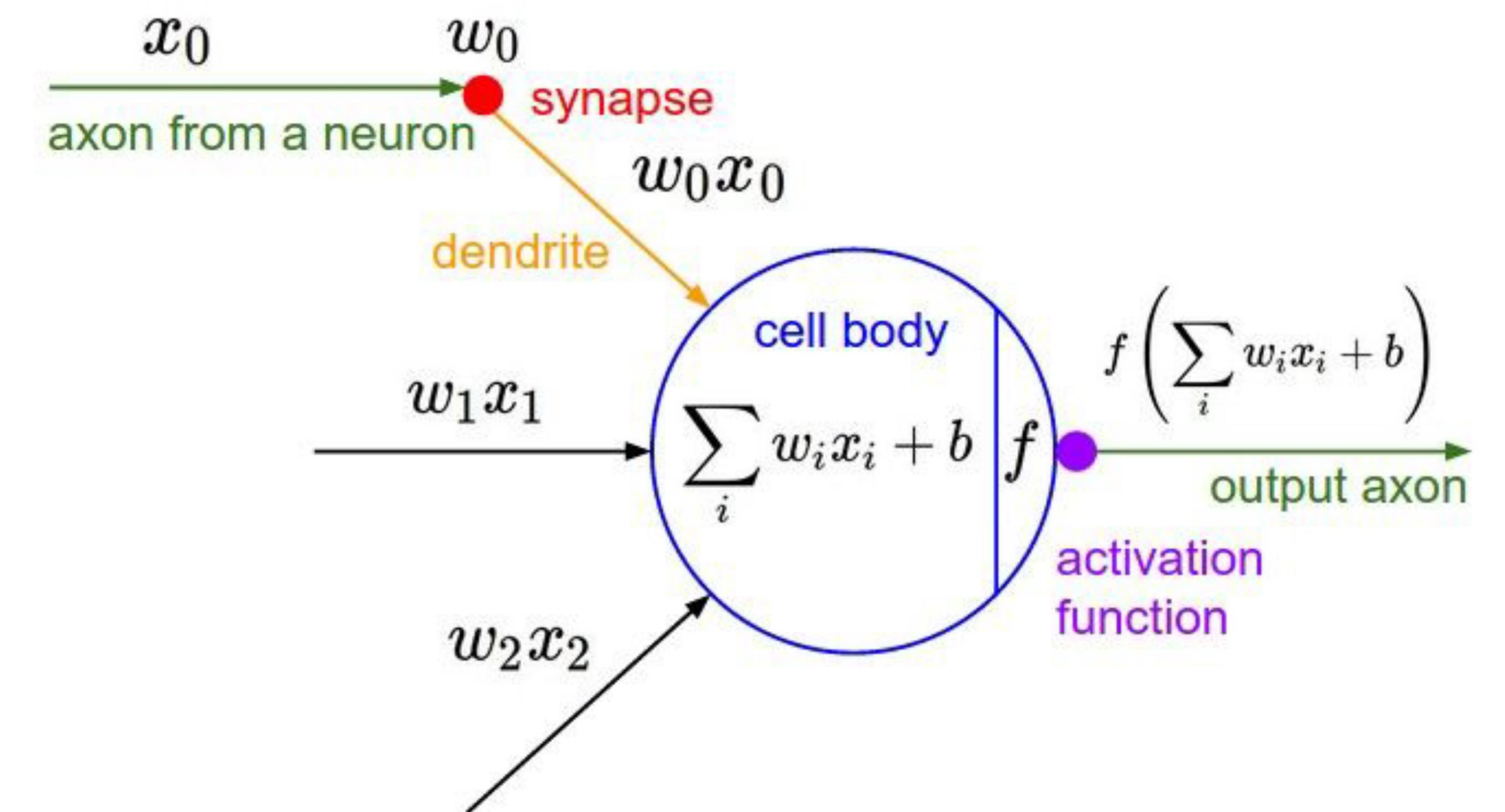
Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$

What can we say about the gradients on ***w***?

Always all positive or all negative :(

(For a single element! Minibatches help)



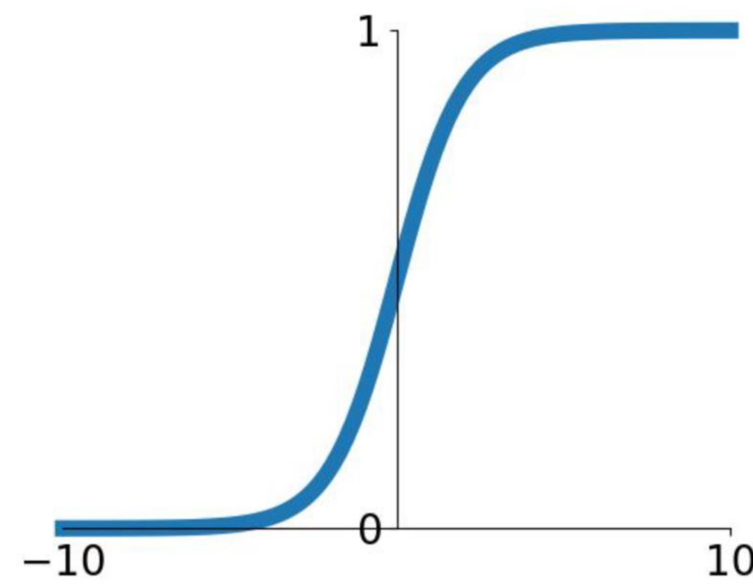
Co-financed by the European Union
Connecting Europe Facility

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

Activation function: *Sigmoid*

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



The sigmoid non-linearity takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).

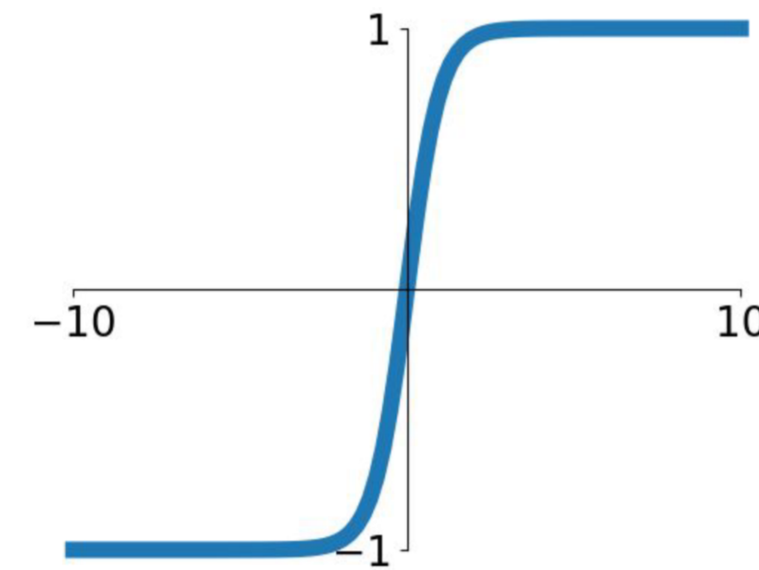
Main Drawbacks

3. Exp() is a bit compute expensive…

Activation function: *tanh*

tanh

$\tanh(x)$



The tanh non-linearity squashes a real-valued number to the range [-1, 1]. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the *tanh non-linearity is always preferred to the sigmoid nonlinearity.* Also note that the tanh neuron is simply a scaled sigmoid neuron.

Main Characteristics

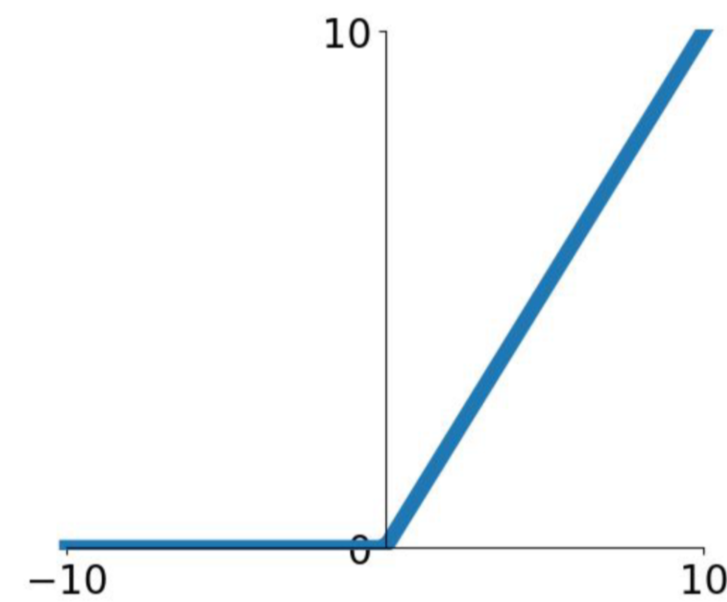
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation function: *tanh*

ReLU

$$\max(0, x)$$



The Rectified Linear Unit has become very popular in the last few years. In other words, the activation is simply thresholded at zero (see image above on the left).

Main Advantages

1. **It does not saturate**
2. **Converges much faster than sigmoid/tanh in practice:** It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al.) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
3. **Very computationally efficient:** Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

[Krizhevsky et al., 2012]

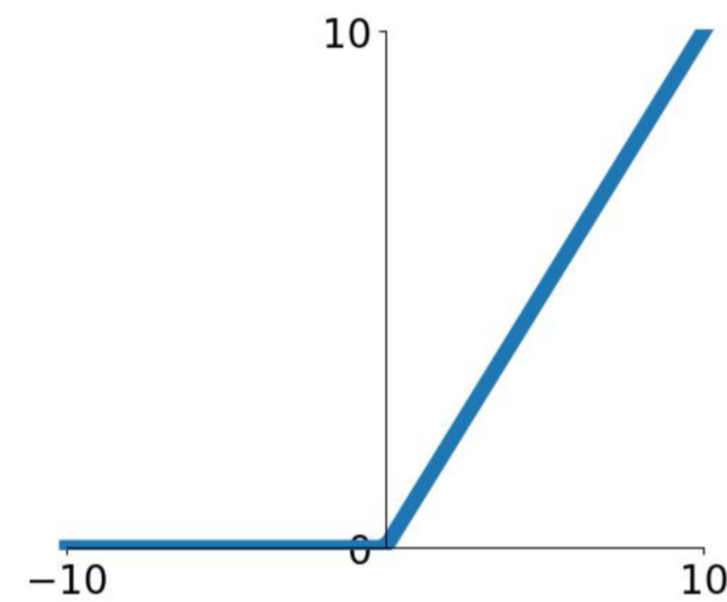
Co-financed by the European Union
Connecting Europe Facility

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

Activation function: *ReLU*

ReLU

$\max(0, x)$



The Rectified Linear Unit has become very popular in the last few years. In other words, the activation is simply thresholded at zero (see image above on the left).

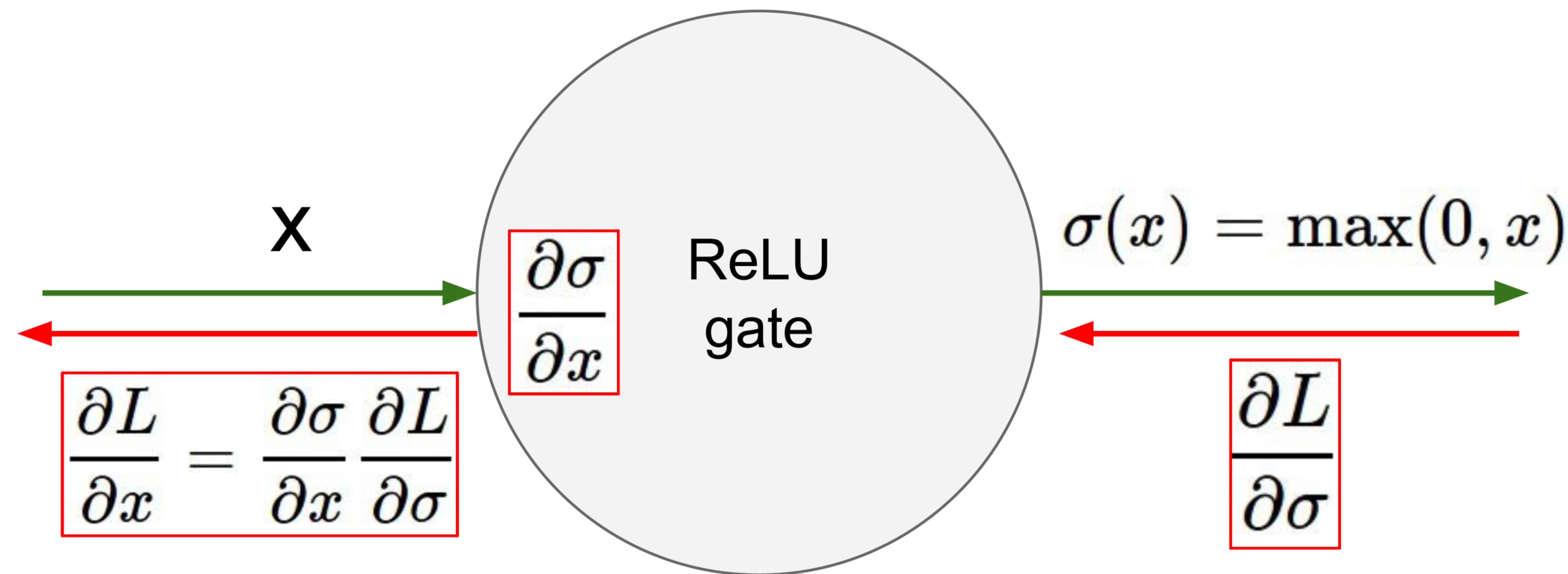
Main Drawbacks

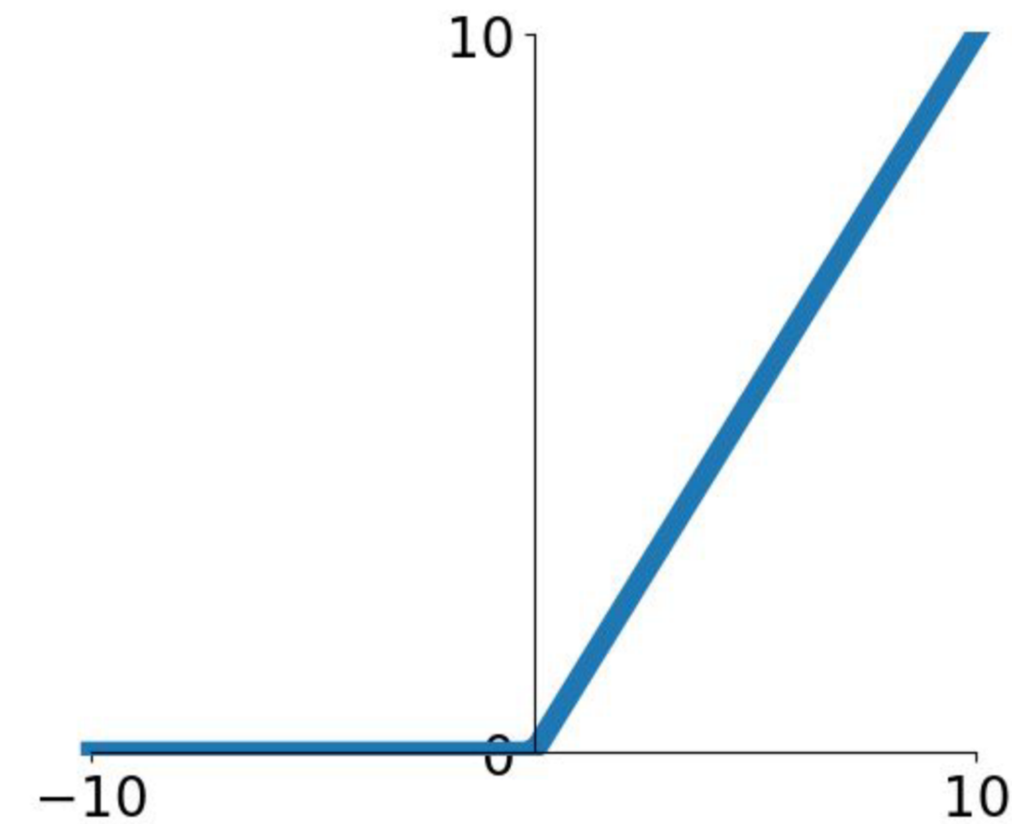
1. **Not zero-centered output**
2. Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

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

Activation function: *ReLU*





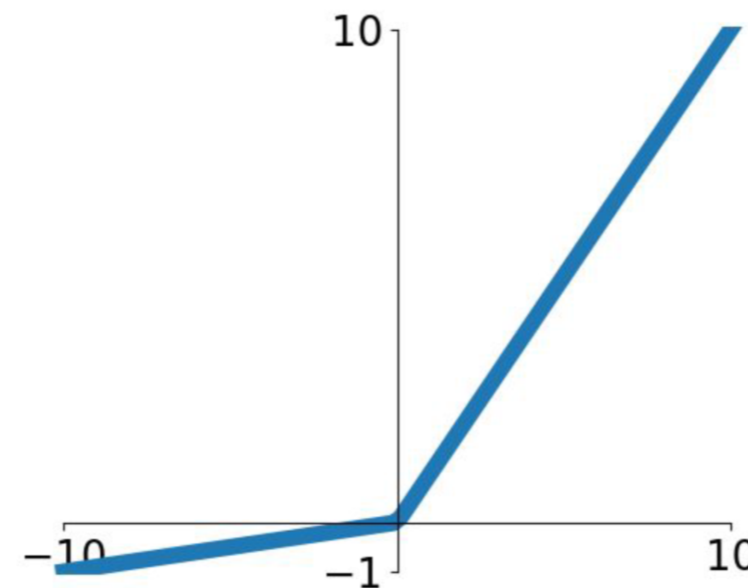
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

=> people like to initialize ReLU neurons with slightly positive biases (e.g. 0.01)

Activation function: *Leaky ReLU*

Leaky ReLU

$$\max(0.1x, x)$$



Leaky ReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small positive slope (of 0.01, or so).

Main Advantages

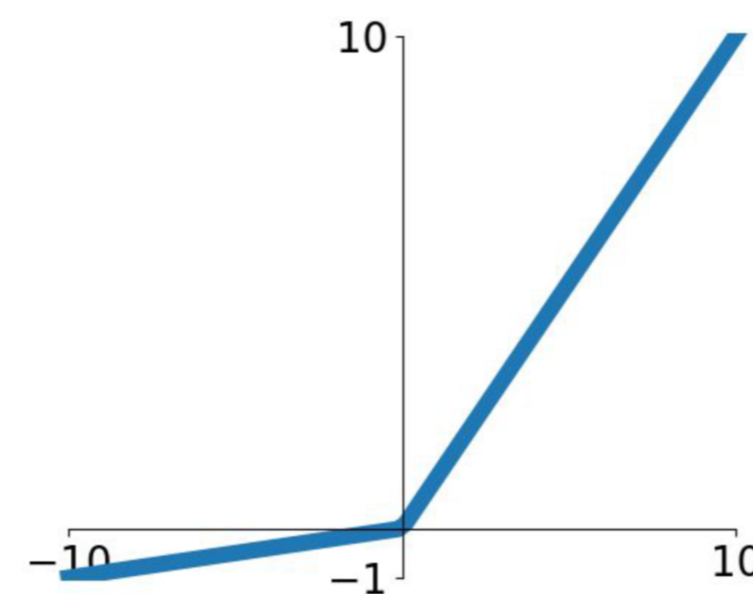
1. **It does not saturate**
2. **Converges much faster than sigmoid/tanh in practice:** It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al.) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
3. **Very computationally efficient:** Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
4. **It will not “die”.**

[Mass et al., 2013] [He et al., 2015]

Activation function: *Leaky ReLU*

Leaky ReLU

$$\max(0.1x, x)$$



Leaky ReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small positive slope (of 0.01, or so).

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into α (parameter)

[Mass et al., 2013] [He et al., 2015]

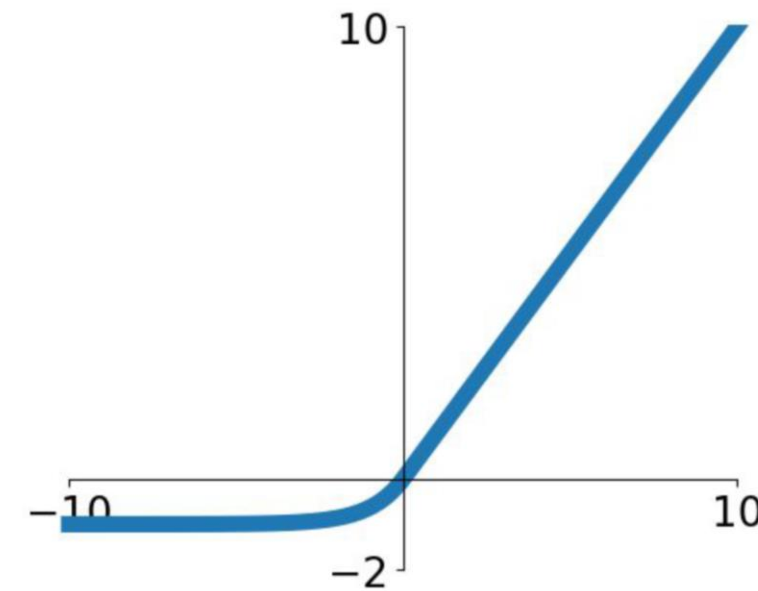
Co-financed by the European Union
Connecting Europe Facility

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

Activation function: *Exponential Linear Units (ELU)*

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



alpha is a hyperparameter that determines the value that the function approaches for large negative inputs. The parameter alpha is usually set to 1.0, but it can be tuned during training.

ELUs are similar to other activation functions such as ReLU. However, ELUs have some advantages over ReLU. One advantage is that ELUs can produce negative values, which can be important for certain types of data. Another advantage is that ELUs can help avoid the "dying ReLU" problem, which occurs when the output of a ReLU neuron becomes stuck at zero and cannot produce meaningful gradients during backpropagation.

Main Advantages

- **All benefits of ReLU**
- **Closer to zero mean outputs**
- **Negative saturation** regime compared with Leaky ReLU, while it adds some robustness to noise

Main Drawbacks

1. **Computation** more expensive since it requires to compute the exp()

[Clevert et al., 2015]

Co-financed by the European Union
Connecting Europe Facility

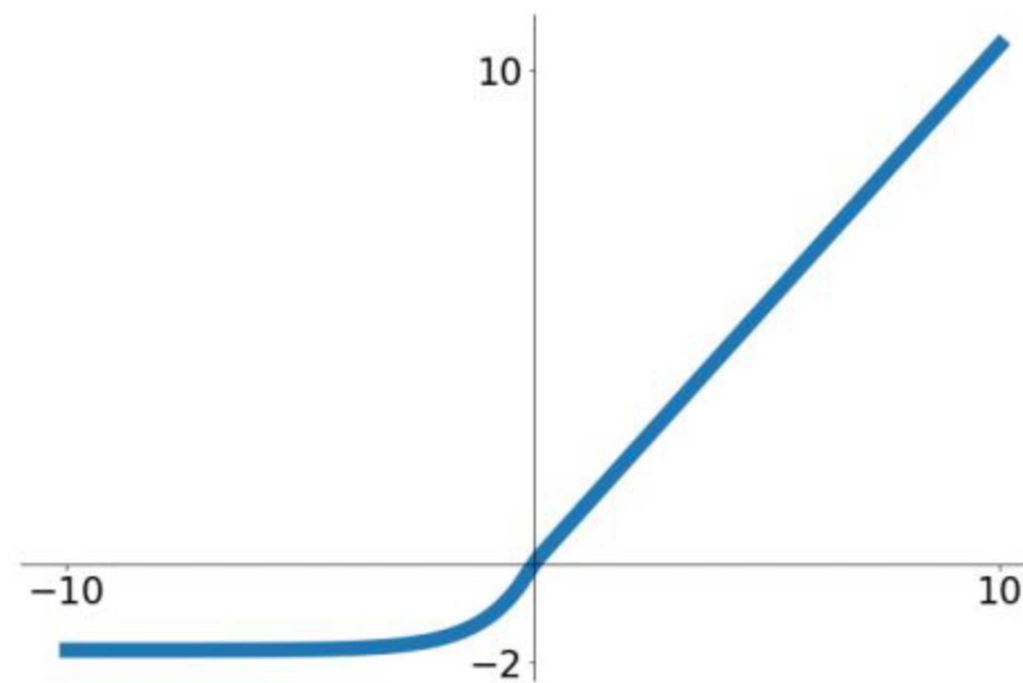
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Activation function: *Scaled Exponential Linear Units (SELU)*

SELU

$$f(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda\alpha(e^x - 1) & \text{otherwise} \end{cases}$$

$$\alpha = 1.6732632423543772848170429916717$$
$$\lambda = 1.0507009873554804934193349852946$$



Main Advantages

1. **Scaled version of ELU that works better for deep networks**
2. **“Self-normalizing” property**
3. **Can train deep SELU networks without BatchNorm**

[Klambauer et al. ICLR 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

Activation function: *Maxout "Neuron"*

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Other types of units have been proposed that do not have the functional form f(wTx+b) where a non-linearity is applied on the dot product between the weights and the data. It generalizes the ReLU and its leaky version. Notice that both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have w1,b1=0).

Main Advantages

1. **Generalizes ReLU and Leaky ReLU**
2. **Linear Regime!** The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU).

Main Drawbacks

1. It **doubles** the number of parameters for every single neuron, leading to a high total number of parameters.

[Goodfellow et al., 2013]

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

Activation function: *Summary*

"What neuron type should I use?"

Use the **ReLU** non-linearity, be careful with your learning rates and possibly monitor the fraction of "dead" units in a network. If this concerns you, give **Leaky ReLU** or **Maxout** a try. Never use sigmoid, and avoid using tanh (you can try tanh, but expect it to work worse than ReLU/Maxout).

Co-financed by the European Union
Connecting Europe Facility

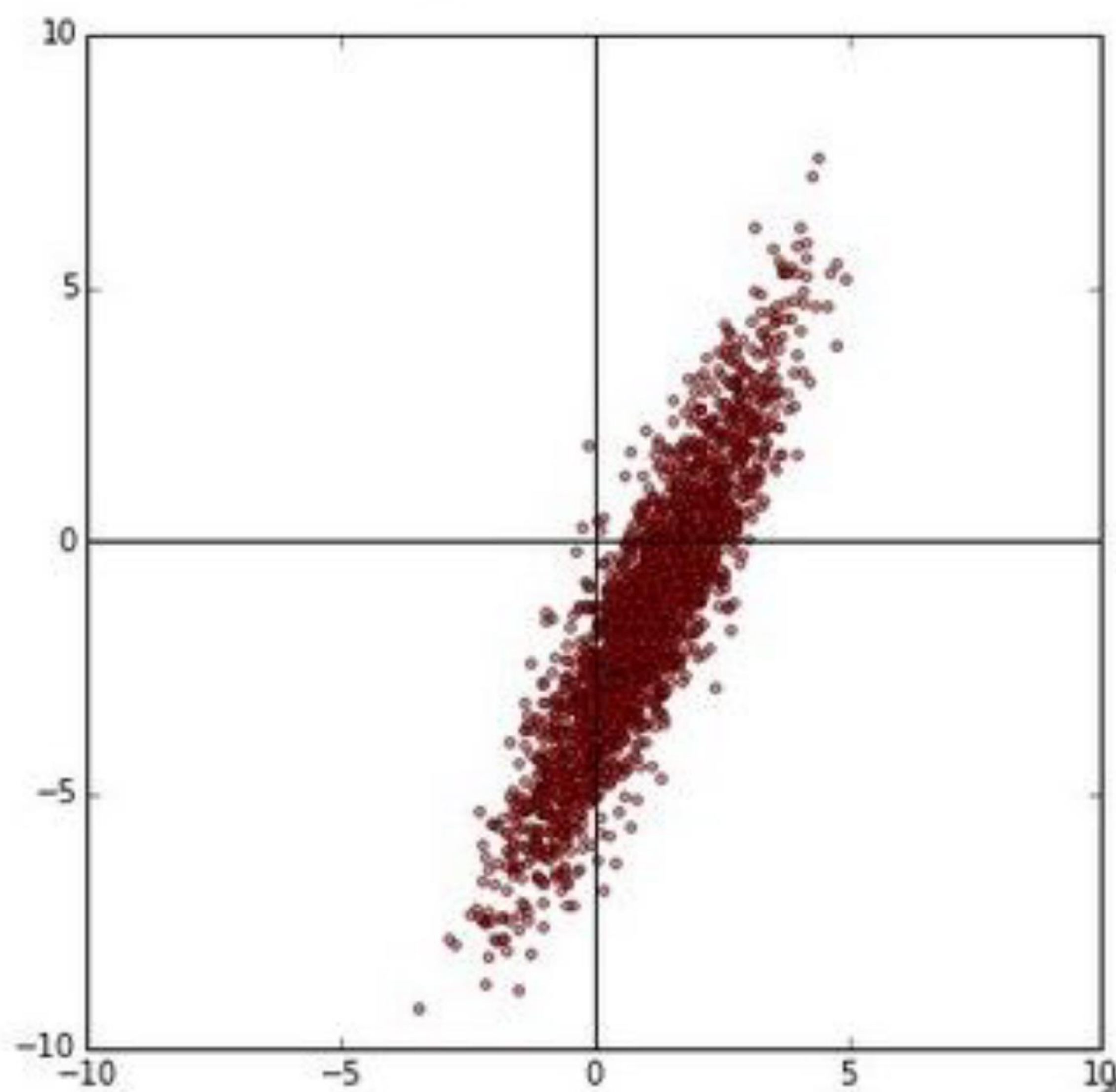
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

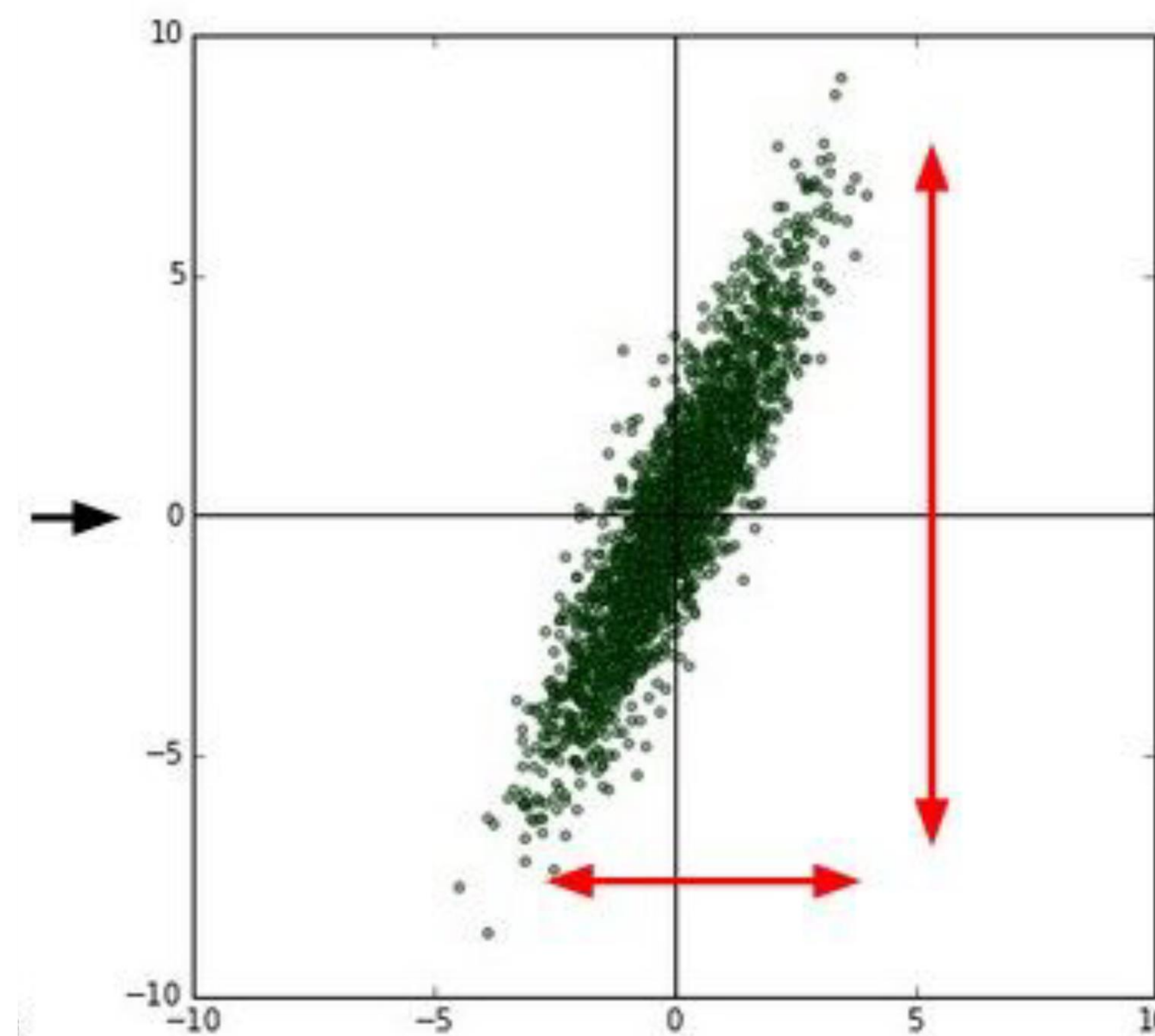
Data Preprocessing

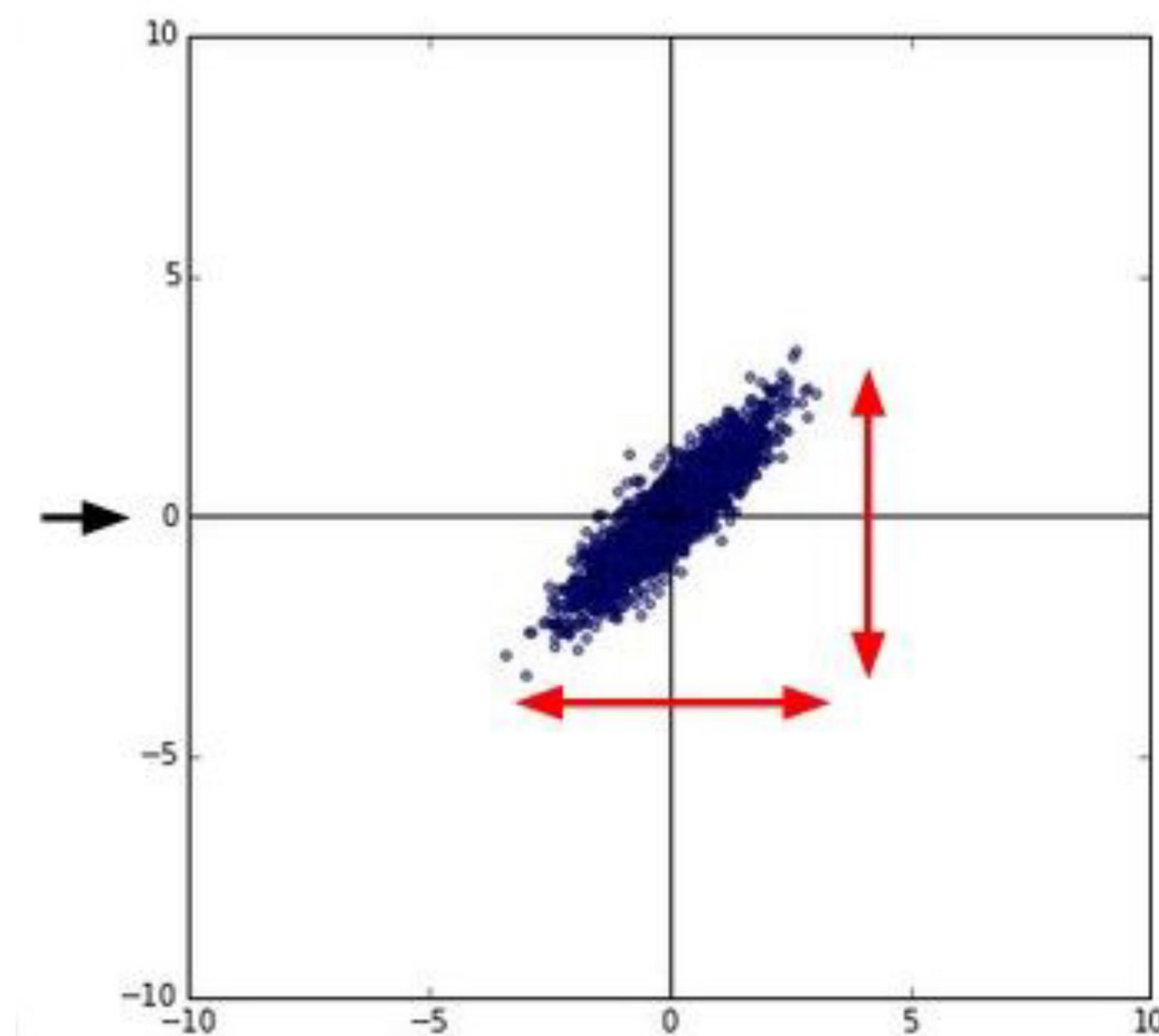
Co-financed by the European Union
Connecting Europe Facility

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

Data Pre-processing








```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

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

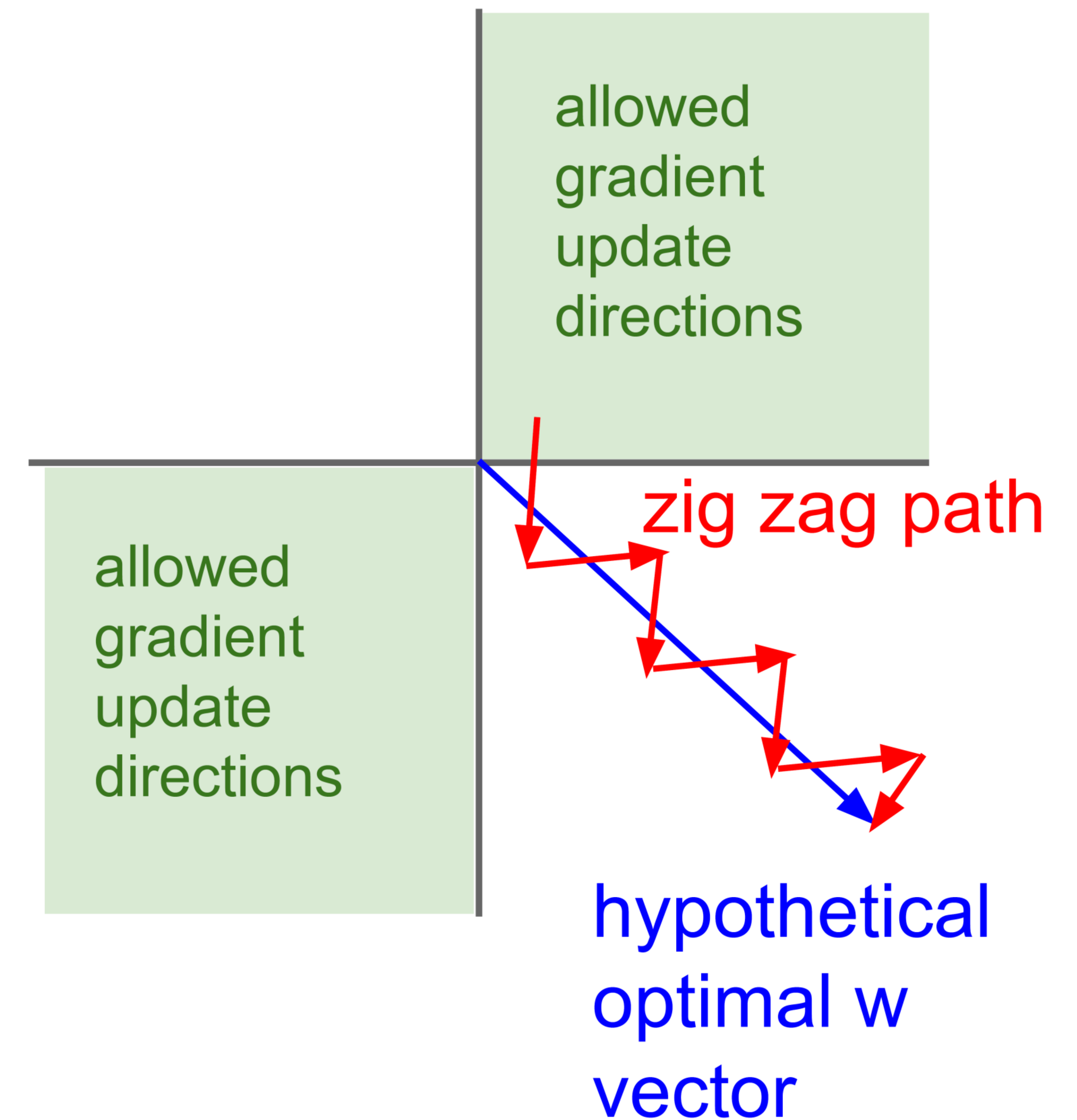
Data Pre-processing

Remember: Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$

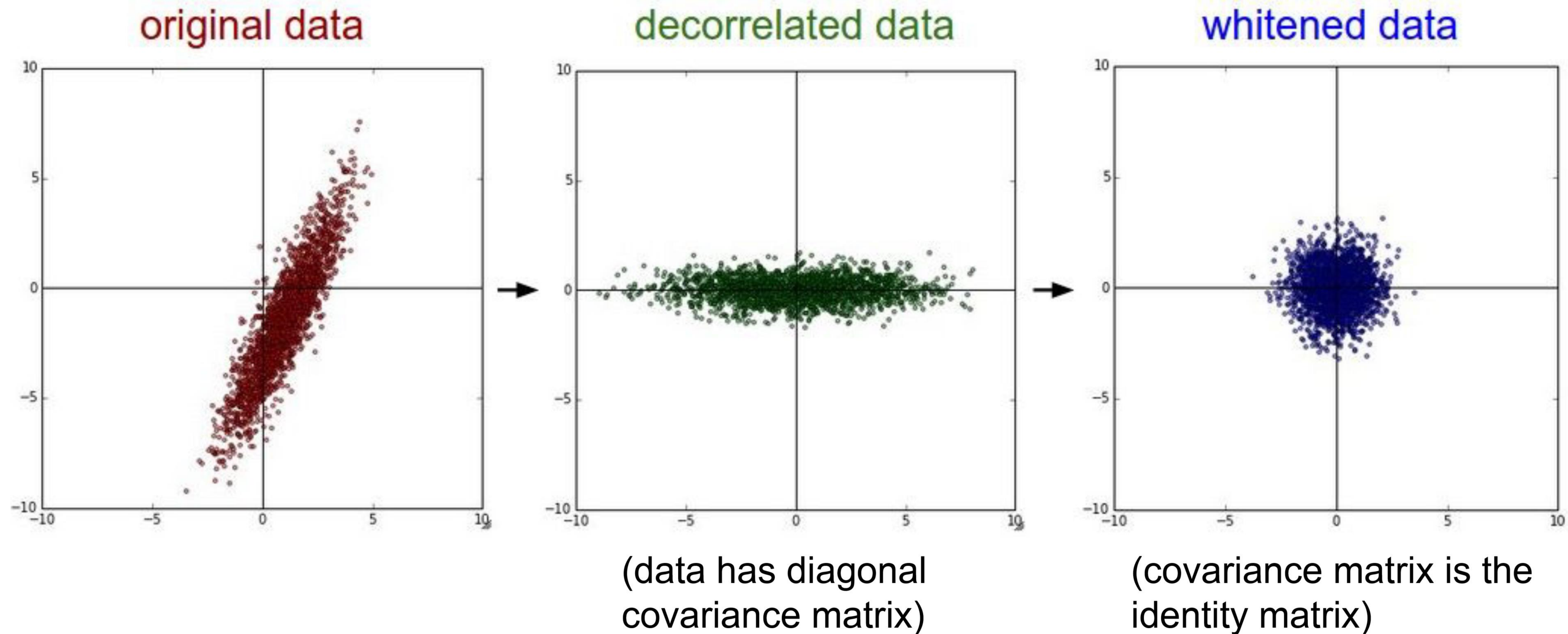
We know that local gradient of sigmoid is always positive

We are assuming x is always positive



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

Data Pre-processing



(data has diagonal covariance matrix)

(covariance matrix is the identity matrix)

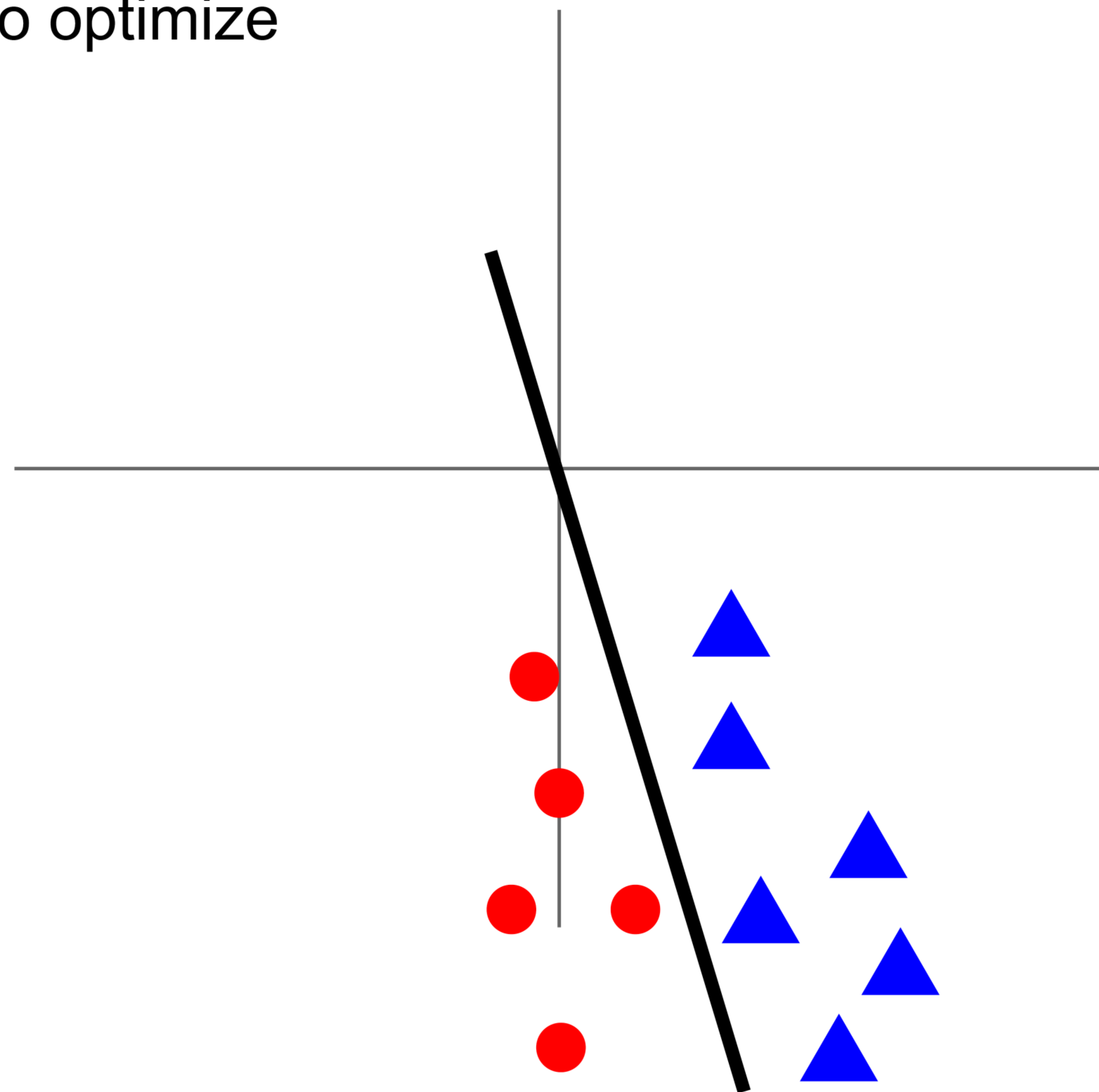
In practice, you may also see PCA and Whitening of the data

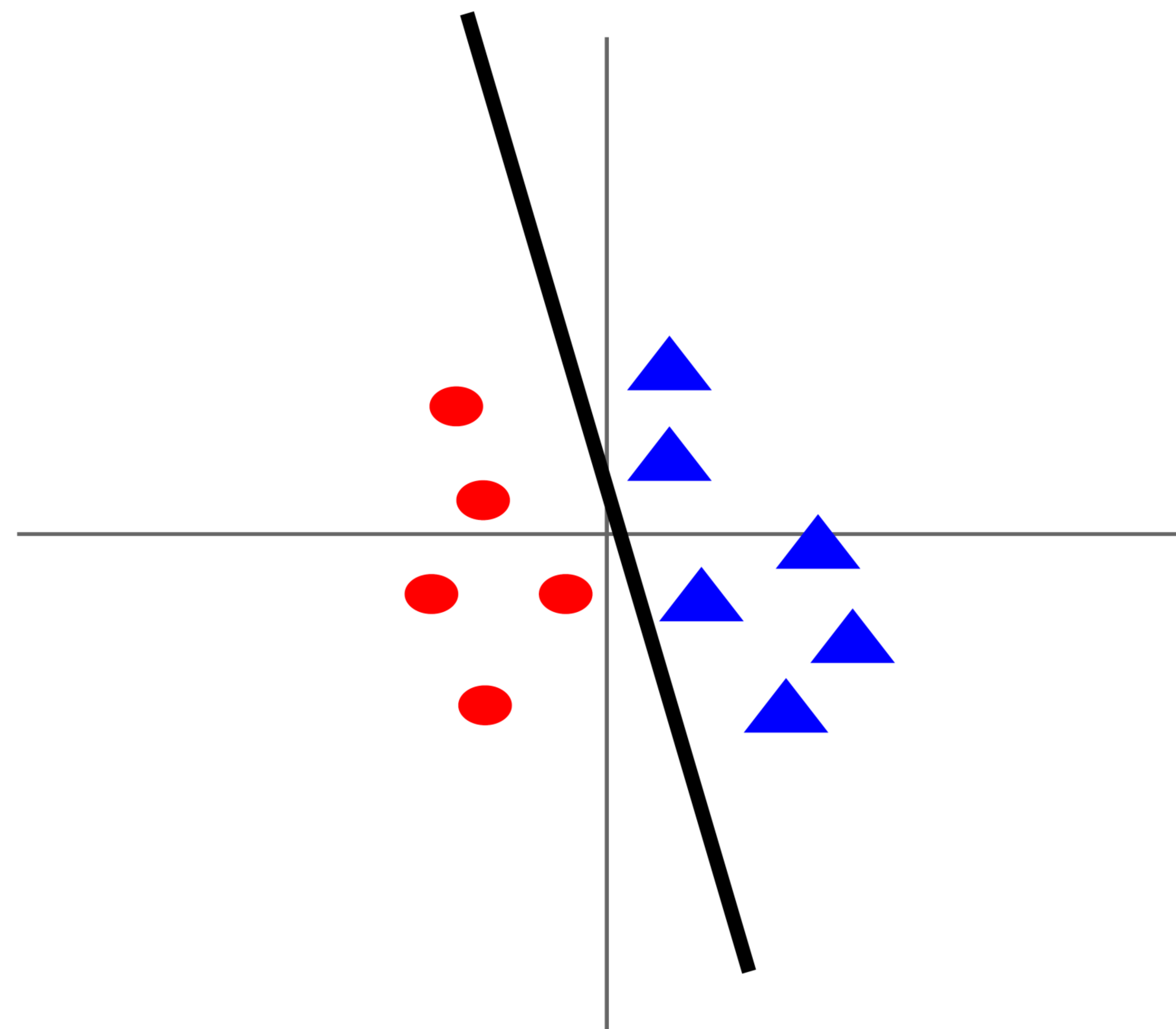
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Data Pre-processing

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize

After normalization: less sensitive to small changes in weights; easier to optimize





Data Pre-processing: *In practice for Images*

For instance, consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)

(mean image = [32,32,3] array)

- Subtract per-channel mean (e.g. VGGNet)

(mean along each channel = 3 numbers)

- Subtract per-channel mean and Divide by per-channel std (e.g. ResNet)

(mean along each channel = 3 numbers)

Not common to do PCA or whitening

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

Weight Initialization

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

Weight initialization

Weight initialization is an important step in training Convolutional Neural Networks (CNNs) because it sets the initial values of the network's weights, which can significantly affect the learning process and the performance of the network.

When initializing the weights of a CNN, there are different methods that can be used. One commonly used method is to initialize the weights randomly, using a Gaussian distribution with zero mean and a small standard deviation. However, this can lead to the problem of vanishing or exploding gradients, which can make it difficult for the network to learn.

```
W = 0.01 * np.random.randn(Din, Dout)
```

Works ~okay for small networks, but problems with 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

Weight initialization: *Activation statistics*

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

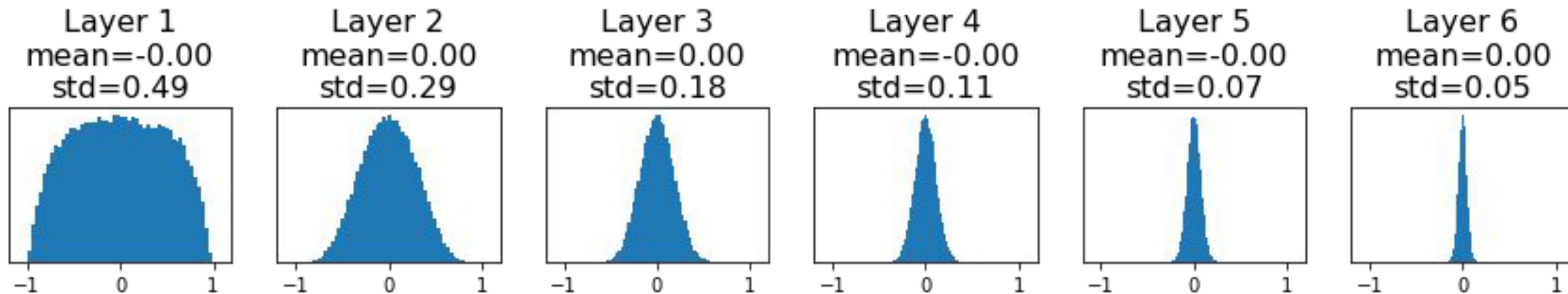
Forward pass for a 6-layer net with hidden size 4096

What will happen to the activations for the last layer?

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?

A: All zero, no learning



Weight initialization: *Activation statistics*

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

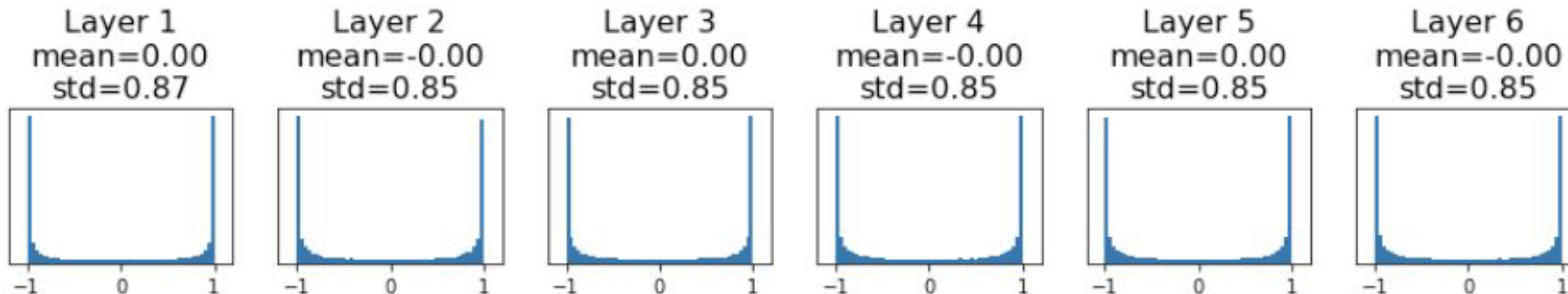
Increase std of initial weights from 0.01 to 0.05

What will happen to the activations for the last layer?

All activations saturate

Q: What do the gradients dL/dW look like?

A: Local gradients all zero, no learning



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

Weight initialization: *Xavier Initialization*

Xavier Initialization: This method initializes the weights using a Gaussian distribution with zero mean and a standard deviation that is calculated based on the number of input and output neurons for each layer. This method ensures that the variance of the activations and gradients remain roughly the same across layers, which can prevent the vanishing and exploding gradient problems.

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

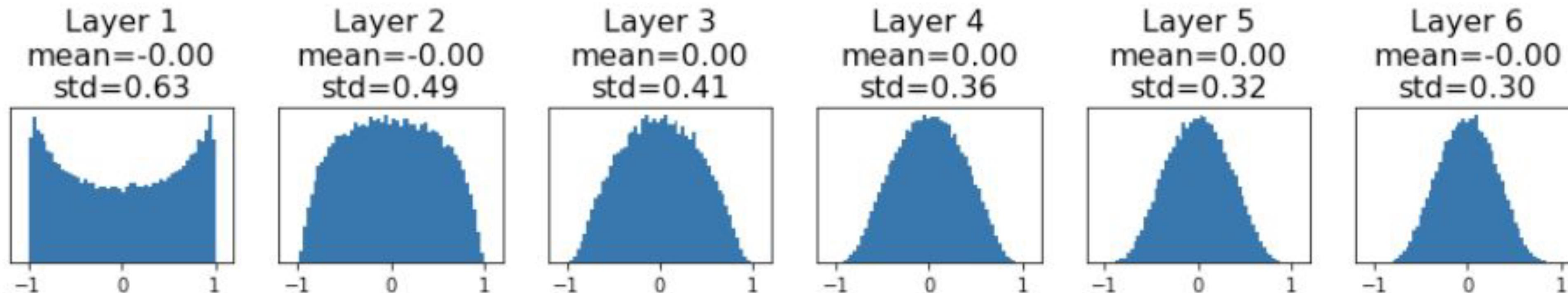
Weight initialization: *Xavier Initialization*

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010



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

Weight initialization: *Xavier Initialization*

- One potential drawback of Xavier initialization is that it assumes that the activations of the input and output layers are independent and identically distributed (IID), which may not always be the case in practice. If the activation distribution of the input or output layer is significantly different from what is assumed by Xavier initialization, it may lead to suboptimal performance.
- Another potential issue with Xavier initialization is that it may not work as well for deep neural networks with many layers, especially when using certain activation functions such as ReLU. In deep networks, the signal from the input may become very small by the time it reaches the output, leading to vanishing gradients. While Xavier initialization can help mitigate this problem, it may not be sufficient in very deep networks, and more advanced techniques like layer normalization or residual connections may be needed.

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

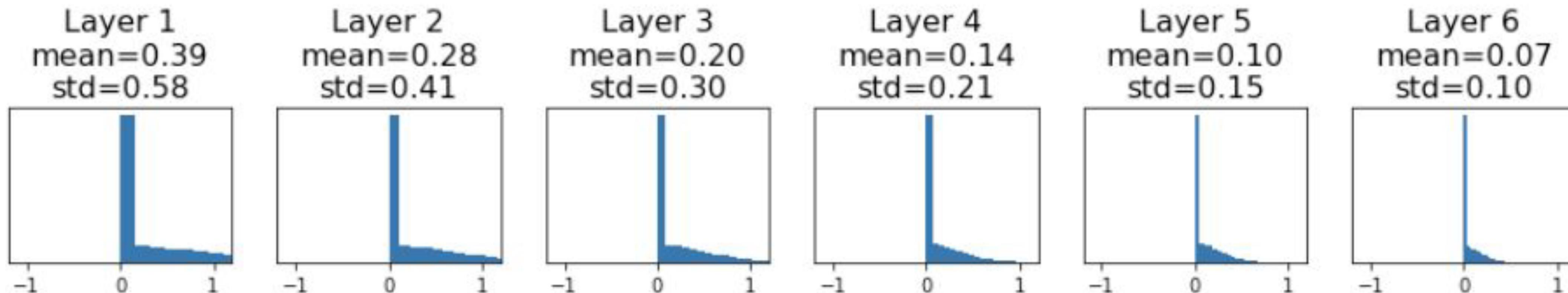
Weight initialization: *Xavier Initialization - What about ReLU?*

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning



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

Weight initialization: *Kaiming / MSRA Initialization*

Kaiming / MSRA Initialization: This method is similar to Xavier initialization but uses a different formula for calculating the standard deviation. It is specifically designed for networks that use Rectified Linear Unit (ReLU) activation functions, which can lead to sparsity in the network's activations. He initialization helps to counteract this sparsity and improve the performance of 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

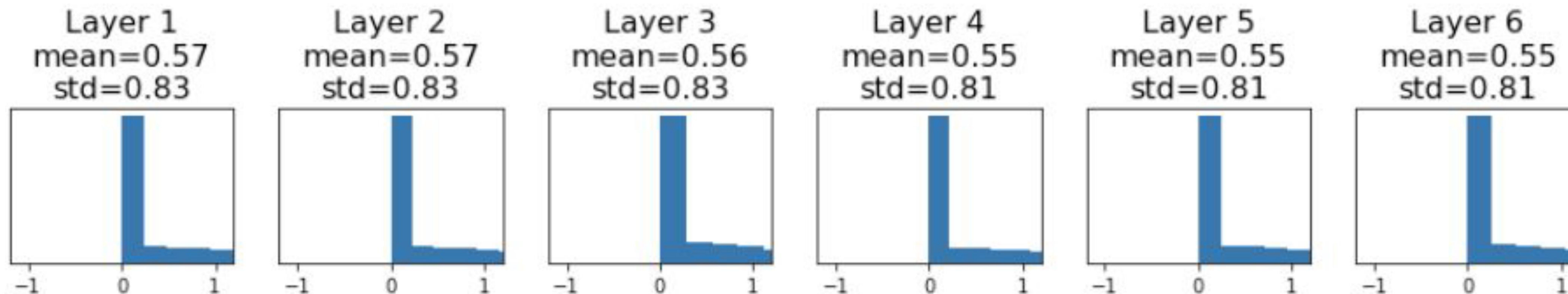
Weight initialization: *Kaiming / MSRA Initialization*

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)

“Just right”: Activations are nicely scaled for all layers!

He et al, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, 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

Weight initialization: *Summary*

Overall, the choice of weight initialization method can depend on the specific network architecture and problem being solved. The goal is to find a method that helps the network converge faster and achieve better performance, without introducing other problems such as vanishing or exploding gradients.

Proper initialization is an active area of research…

- ***Understanding the difficulty of training deep feedforward neural networks*** by Glorot and Bengio, 2010
- ***Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*** by Saxe et al, 2013
- ***Random walk initialization for training very deep feedforward networks*** by Sussillo and Abbott, 2014
- ***Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*** by He et al., 2015
- ***Data-dependent Initializations of Convolutional Neural Networks*** by Krähenbühl et al., 2015
- ***All you need is a good init***, Mishkin and Matas, 2015
- ***Fixup Initialization: Residual Learning Without Normalization***, Zhang et al, 2019
- ***The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks***, Frankle and Carbin, 2019

Co-financed by the European Union
Connecting Europe Facility

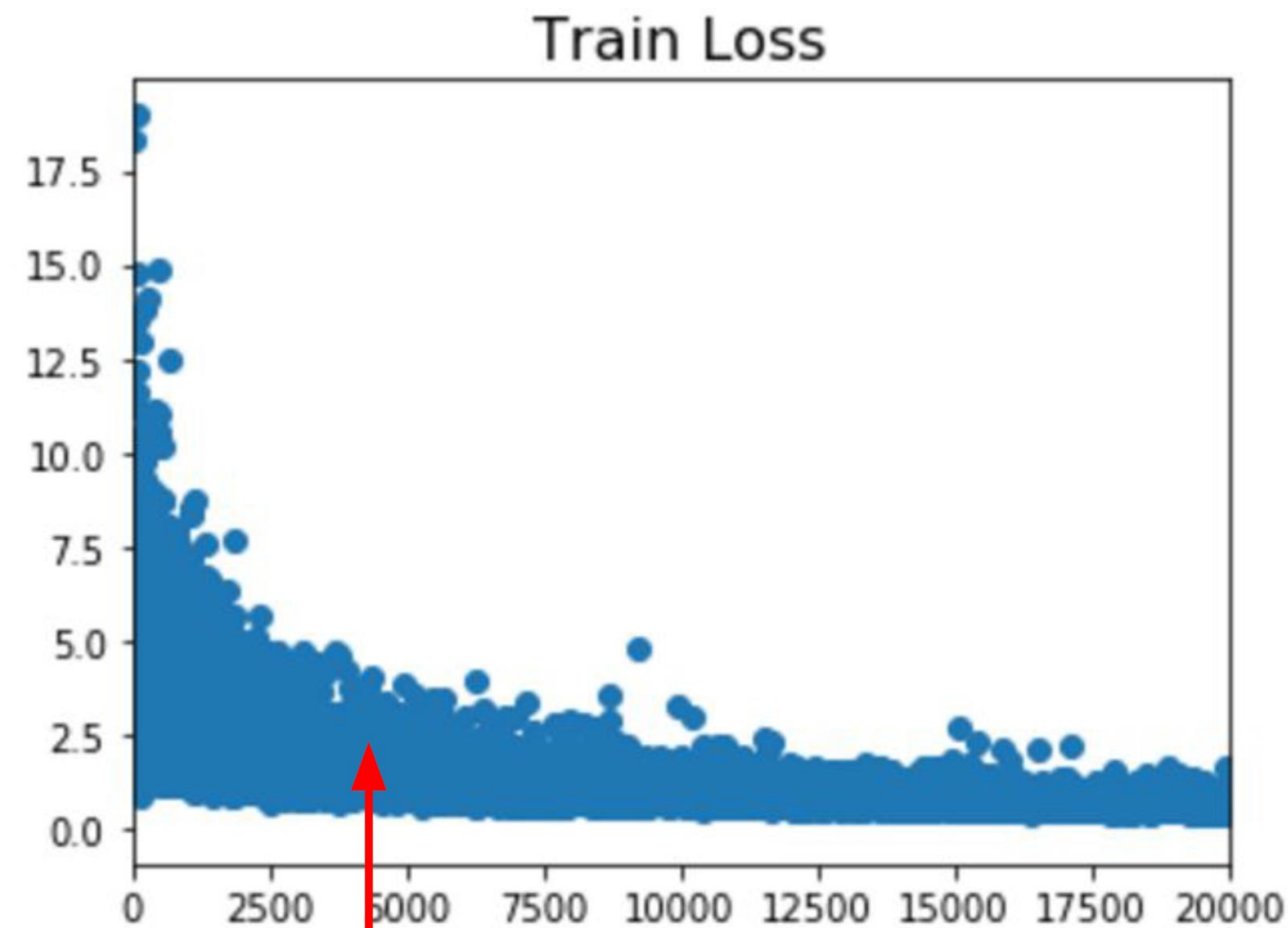
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

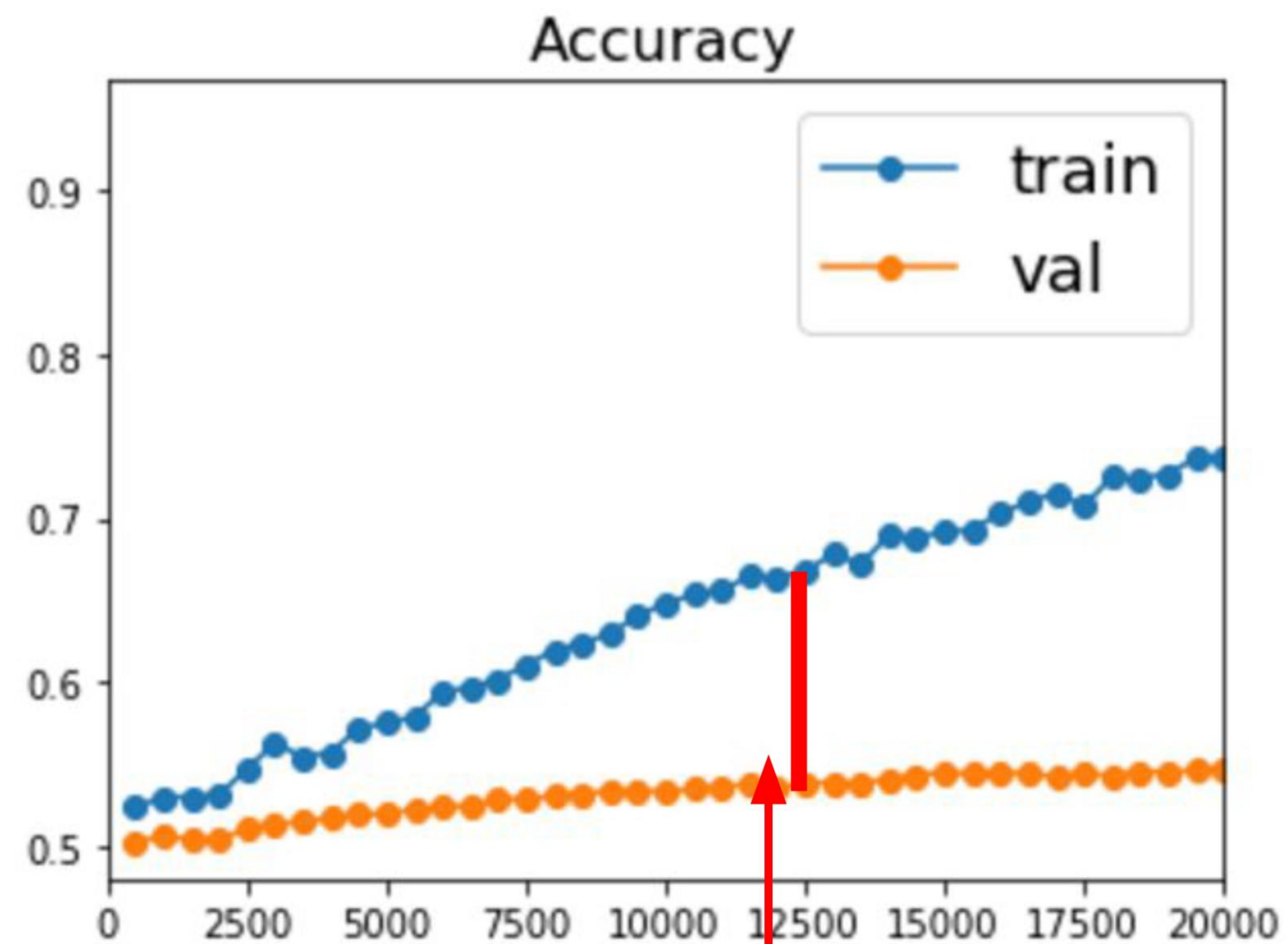
Training vs. Testing Error

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 Training Error





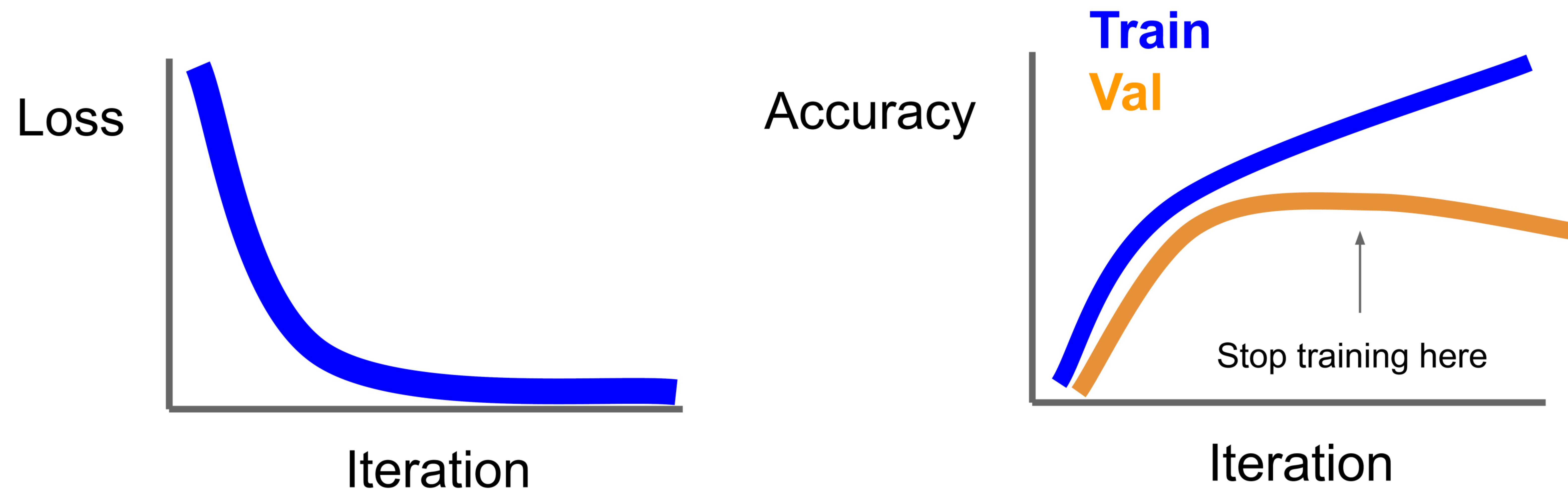
Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

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

Early Stopping: *Always do this*



Stop training the model when accuracy on the **validation** set decreases or train for a long time, but always keep track of the model snapshot that worked best on **validation**

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

Model Ensembles

1. Train multiple independent models
2. At test time average their results

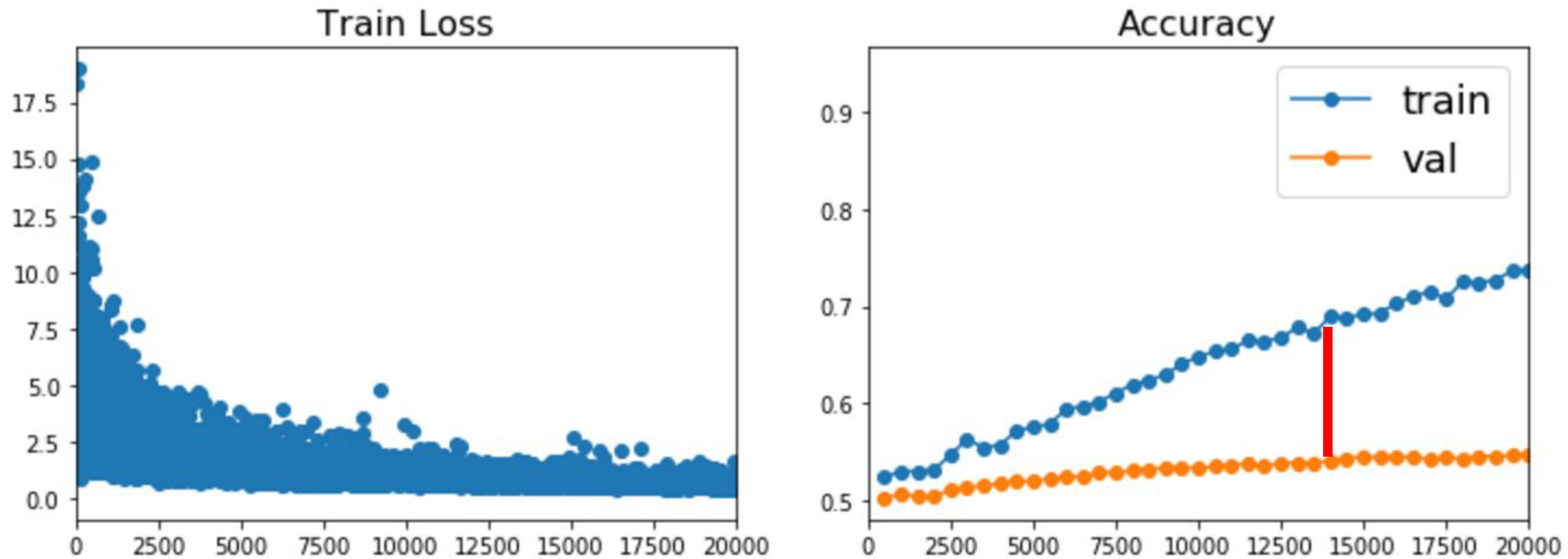
    (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

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

How to improve single-model performance?



Regularization

Regularization: *Add term to loss*

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

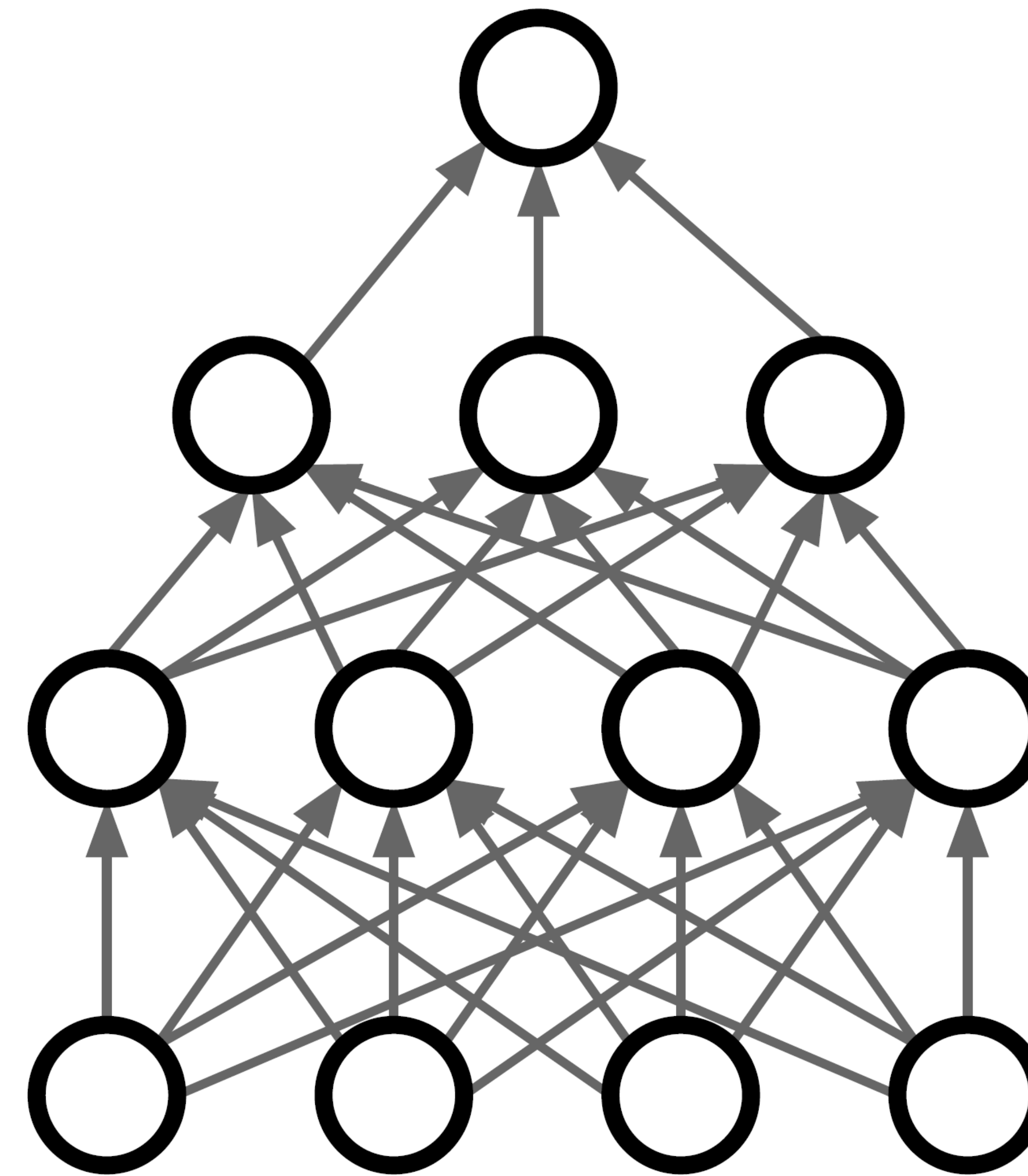
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

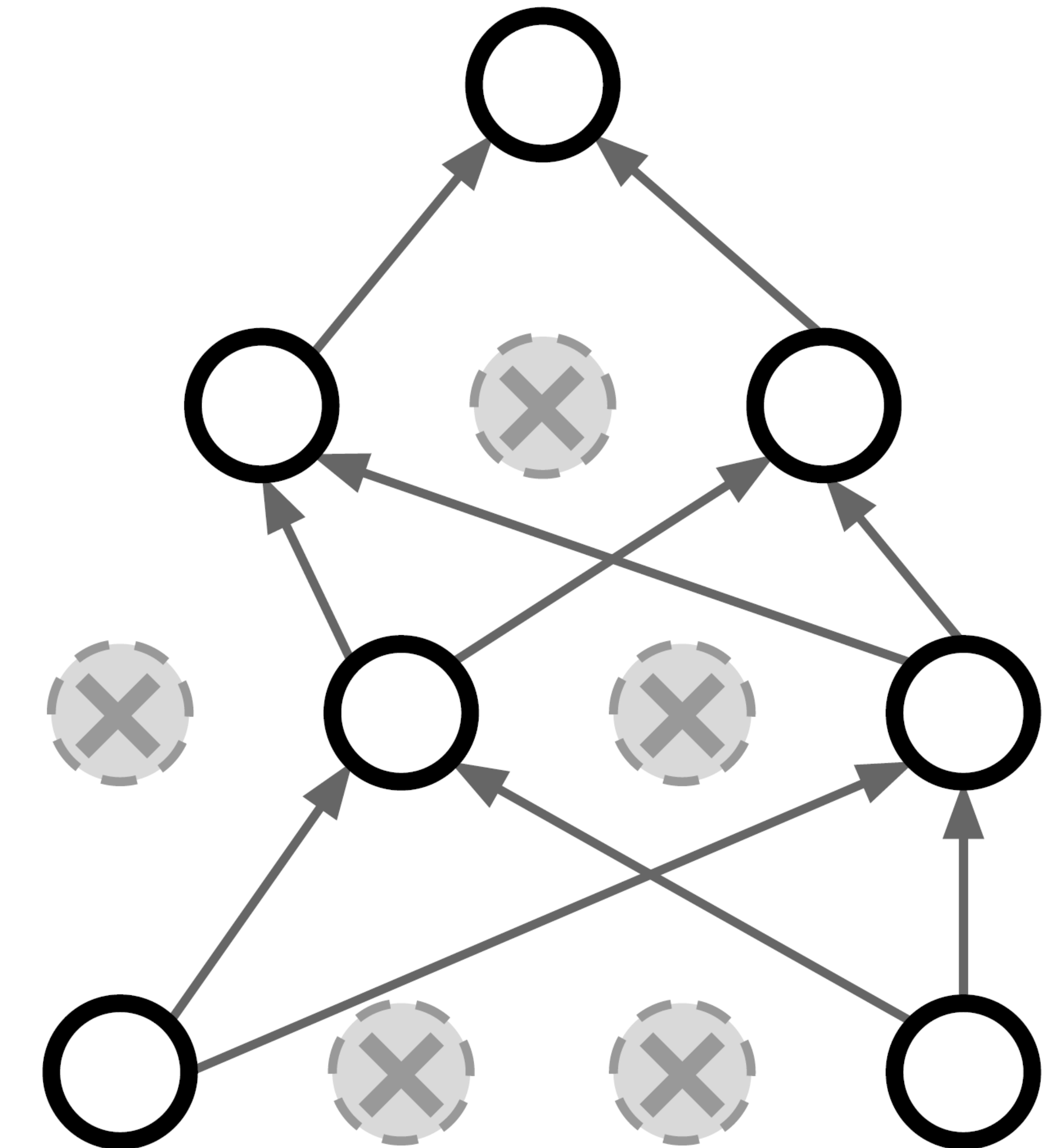
Co-financed by the European Union
Connecting Europe Facility

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

Regularization: *Dropout*

The idea behind dropout is to randomly drop out some of the neurons in a layer during training, with a certain probability. This has the effect of preventing any individual neuron from relying too heavily on any other neuron in the layer, forcing the network to learn more robust and distributed representations of the data.





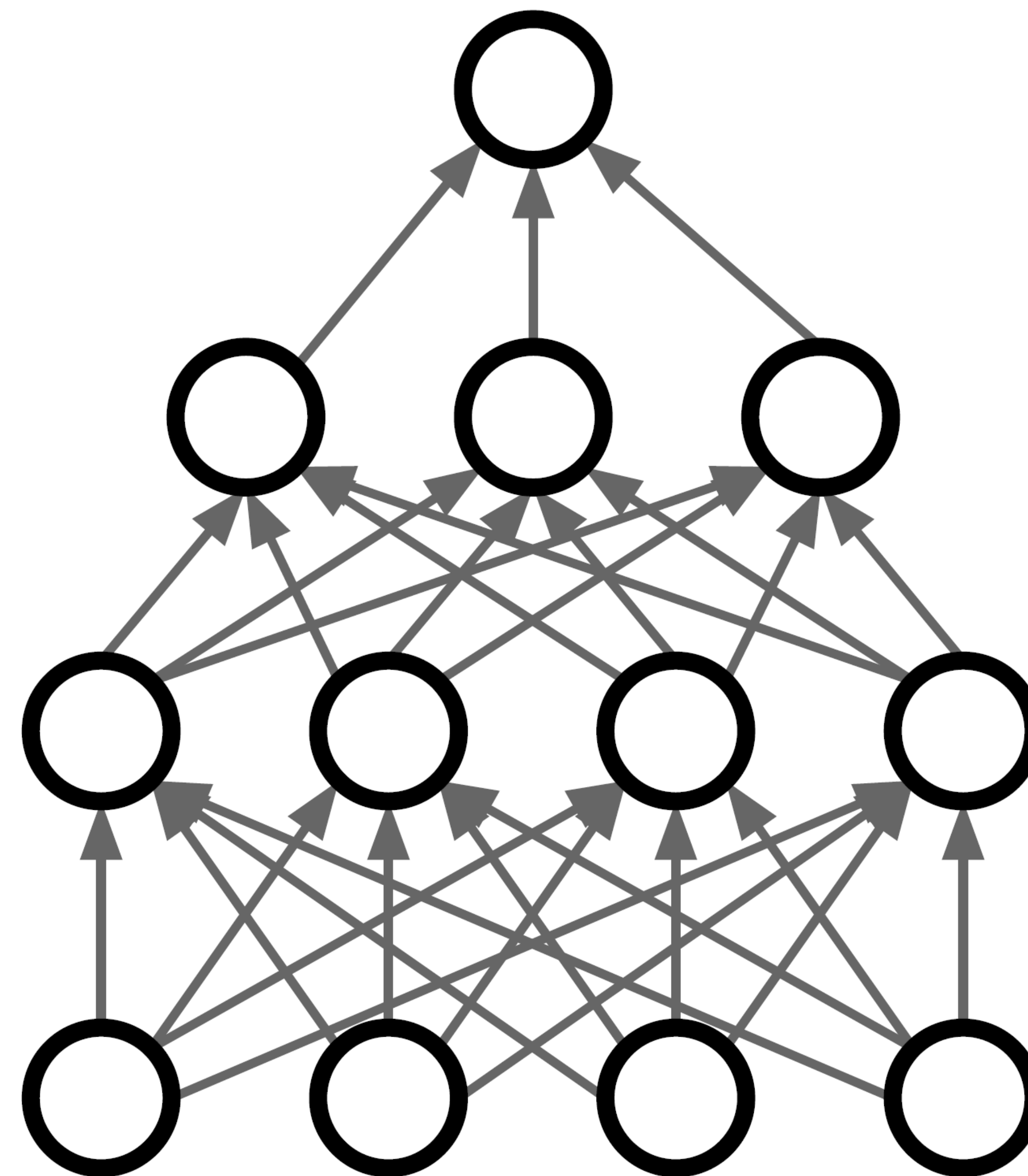
Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

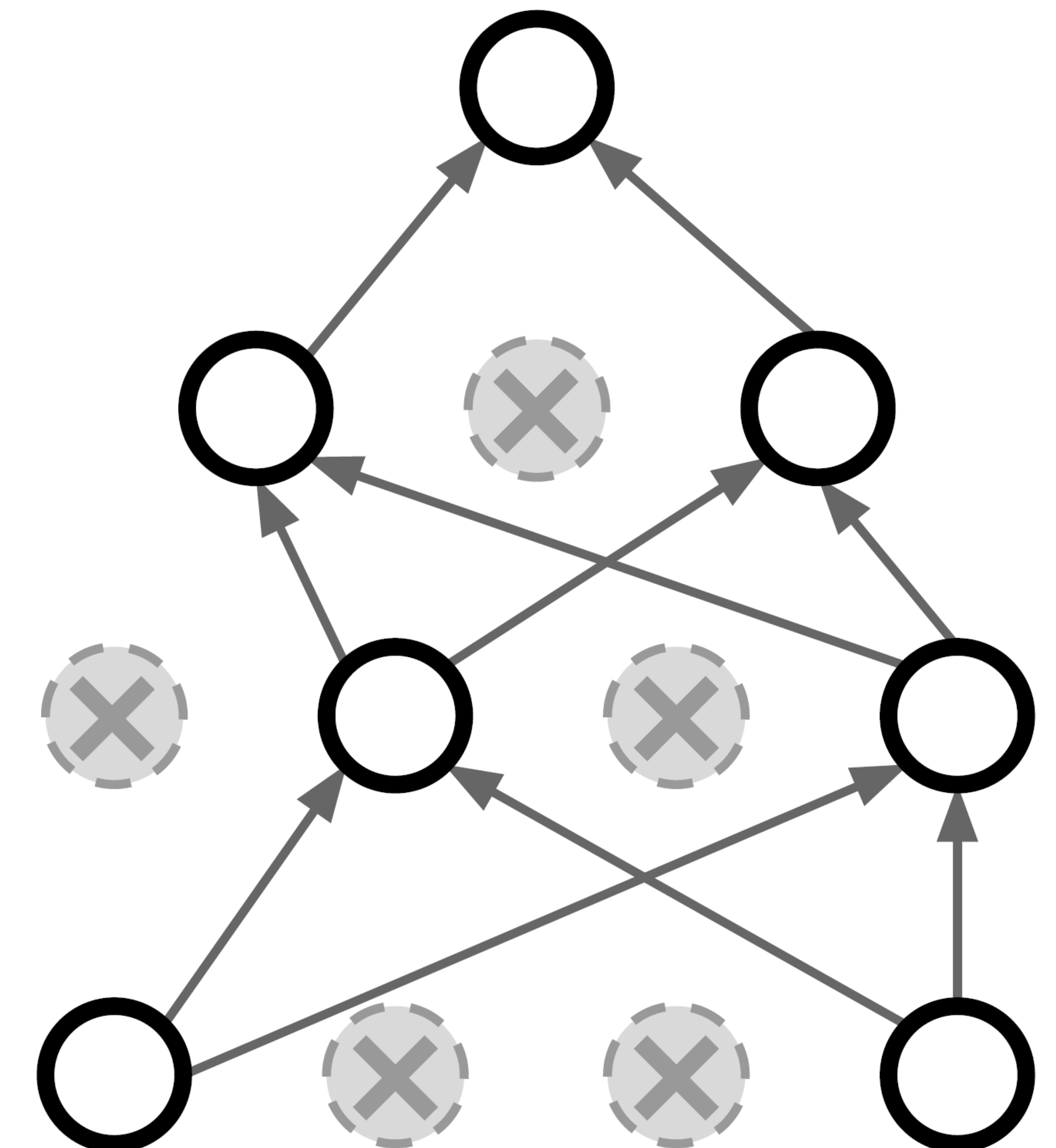
Co-financed by the European Union
Connecting Europe Facility

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

Regularization: *Dropout*

In each forward pass, randomly set some neurons to zero Probability of dropping is a hyperparameter; 0.5 is common





Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Regularization: *Dropout*

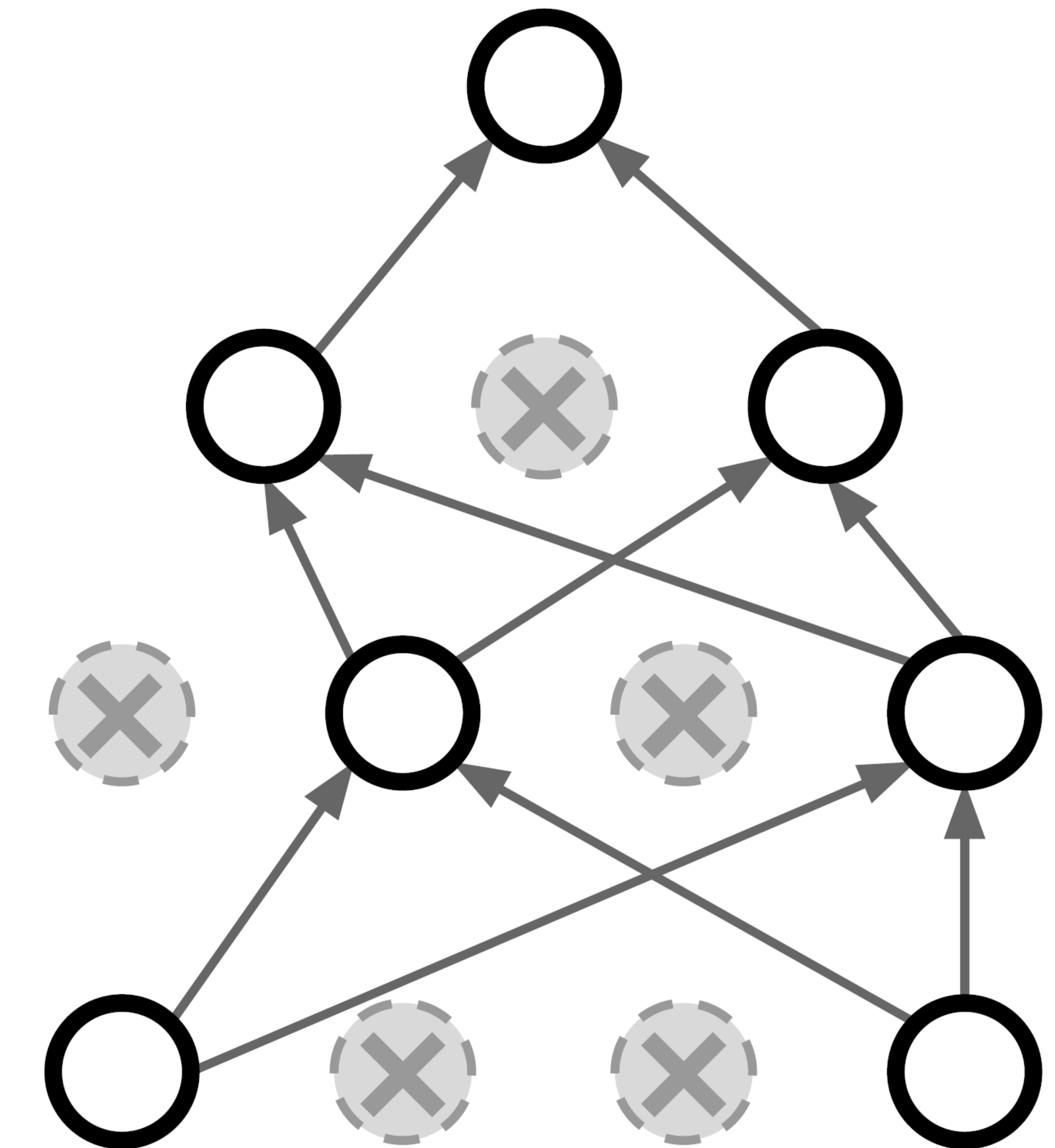
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout



Regularization: *Dropout + Scaling*

Scaling after dropout is a technique used in neural network training that involves rescaling the activations of a layer after applying dropout regularization. The purpose of scaling is to ensure that the expected value of the output of a neuron remains the same, regardless of whether dropout was applied or not.

When dropout is applied to a layer during training, some of the neurons in the layer are randomly dropped out with a certain probability. This has the effect of reducing the capacity of the layer, and can help prevent overfitting by forcing the network to learn more robust and distributed representations of the data. However, when neurons are dropped out, the remaining neurons receive a larger input, which can lead to an increase in the variance of the activations. This can make it difficult to train the network effectively, as the gradient updates may become unstable.

To address this issue, scaling after dropout can be used to rescale the activations of the remaining neurons, so that their expected value remains the same as before dropout was applied. This is typically achieved by dividing the activations by the probability of keeping a neuron during dropout. e.g., if dropout probability is 0.5, then the scaling factor would be 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

Regularization: *A common pattern*

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

Example: Batch Normalization

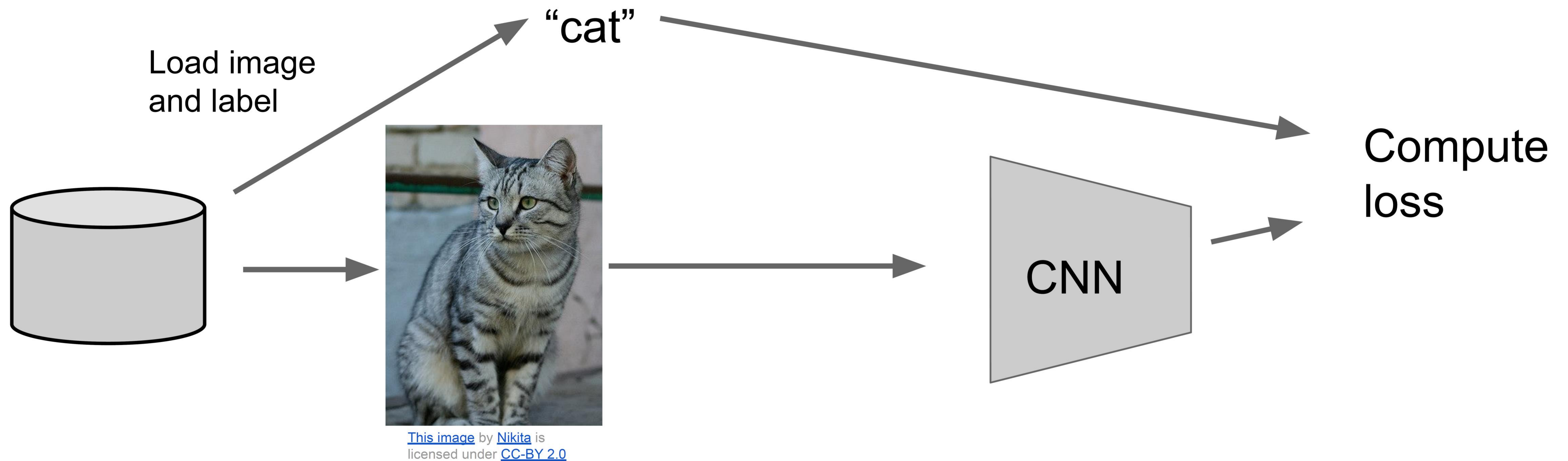
Training:
Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

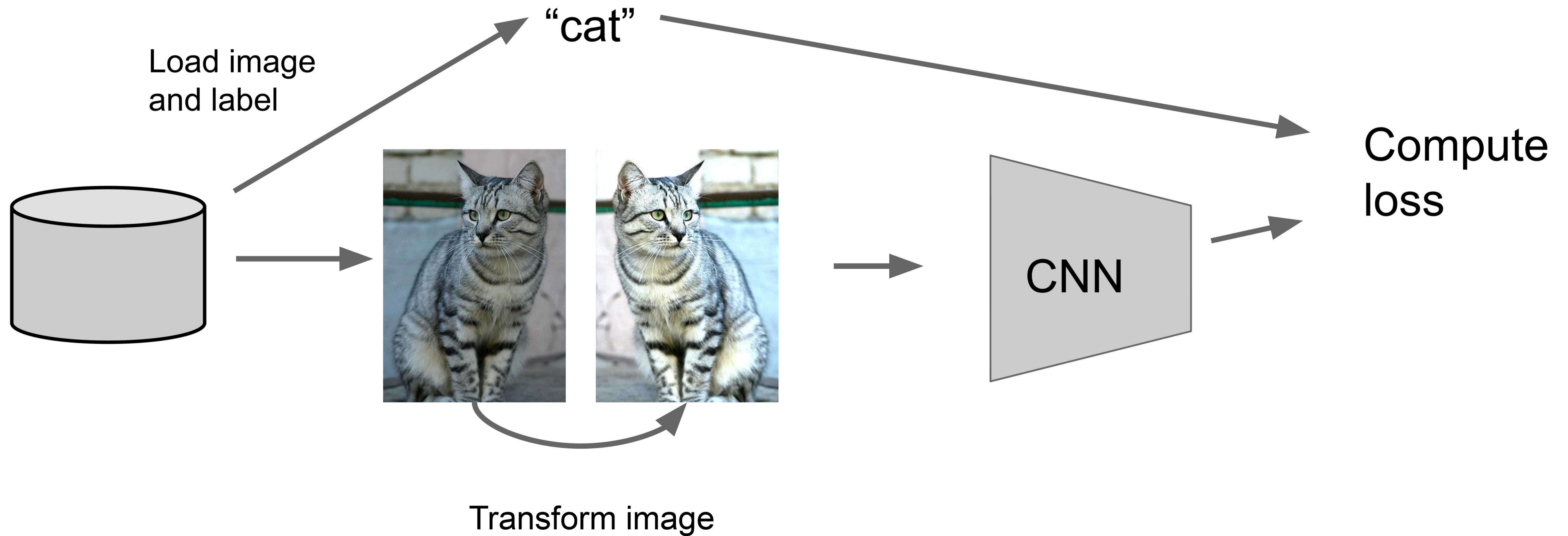
Co-financed by the European Union
Connecting Europe Facility

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

Regularization: *Data Augmentation*



Regularization: *Data Augmentation*



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

Regularization: *Data Augmentation*

Horizontal Flips

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

Regularization: *Data Augmentation*

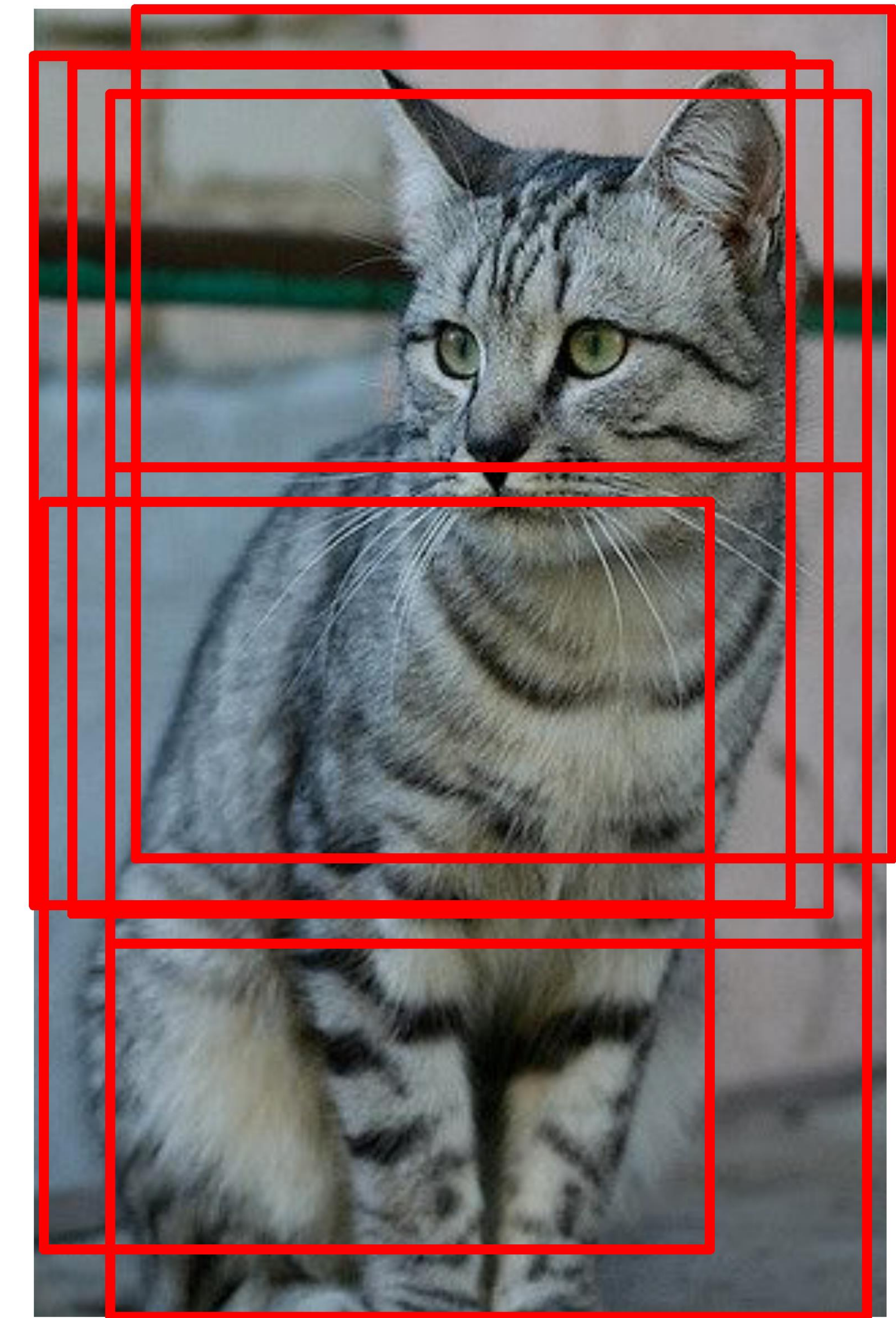
Random crops and scales

Training: sample random crops / scales ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips



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

Regularization: *Data Augmentation*

Color Jitter

Simple: Randomize contrast and brightness

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

Regularization: *Data Augmentation*

Color Jitter

More Complex:
1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

[Krizhevsky et al. 2012], 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

Regularization: *Data Augmentation*

Get creative for your problem!

Examples of data augmentations:

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Cubuk et al., "AutoAugment: Learning Augmentation Strategies from Data", CVPR 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

Regularization: *A common pattern*

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout

Batch Normalization

Data Augmentation

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

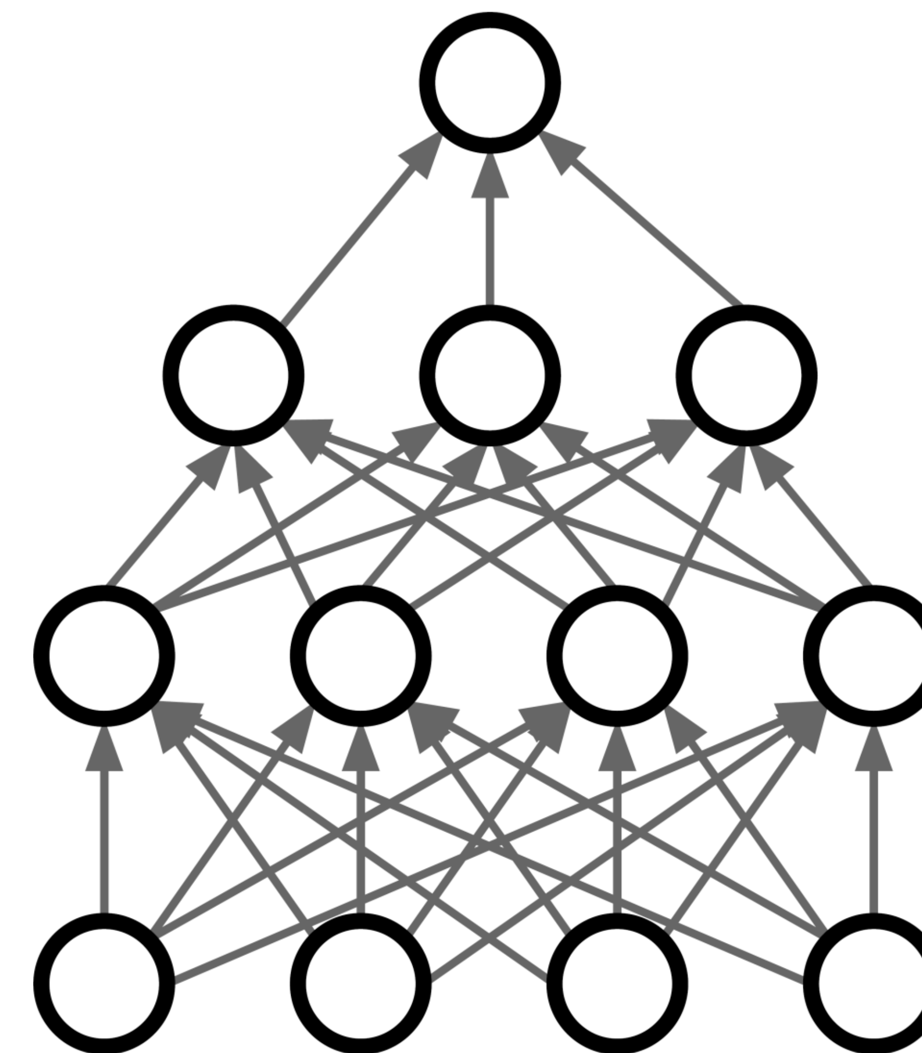
Regularization: *DropConnect*

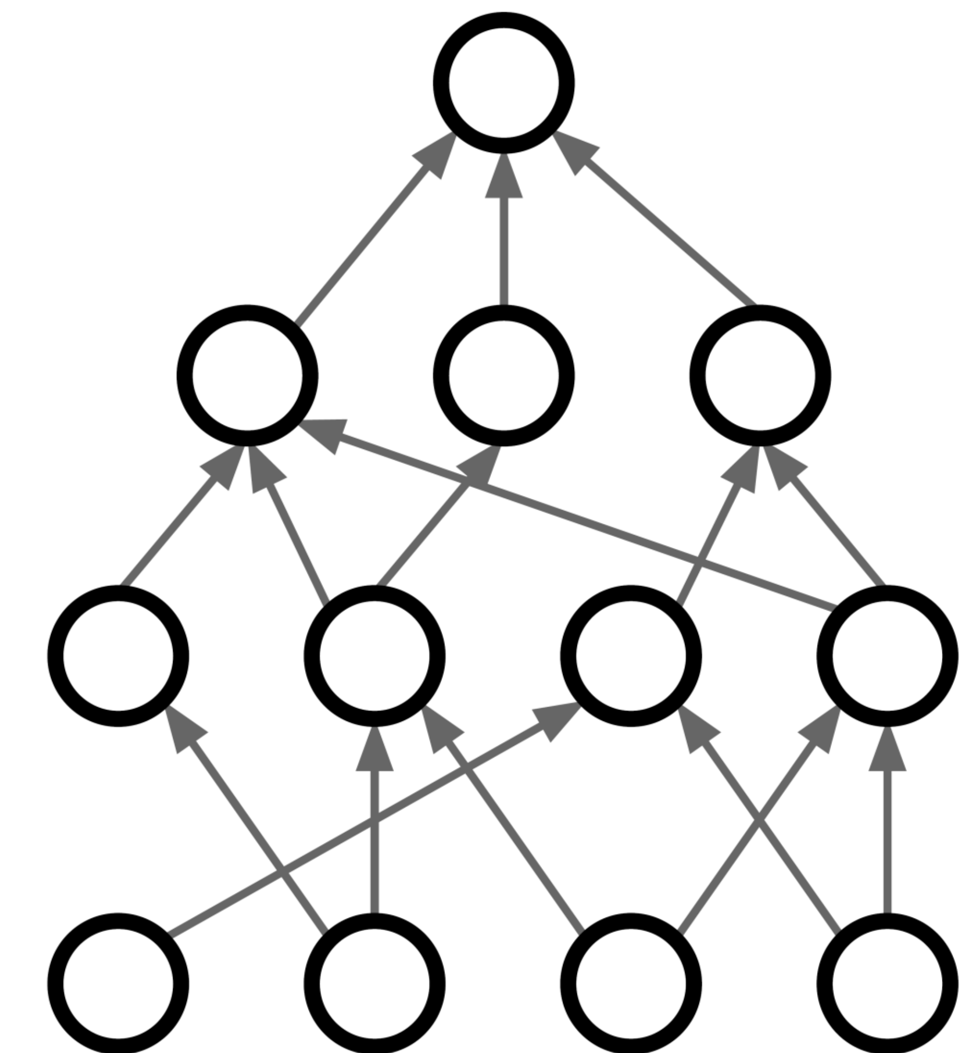
Training: Drop connections between neurons (set weights to 0)

Testing: Use all the connections

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect





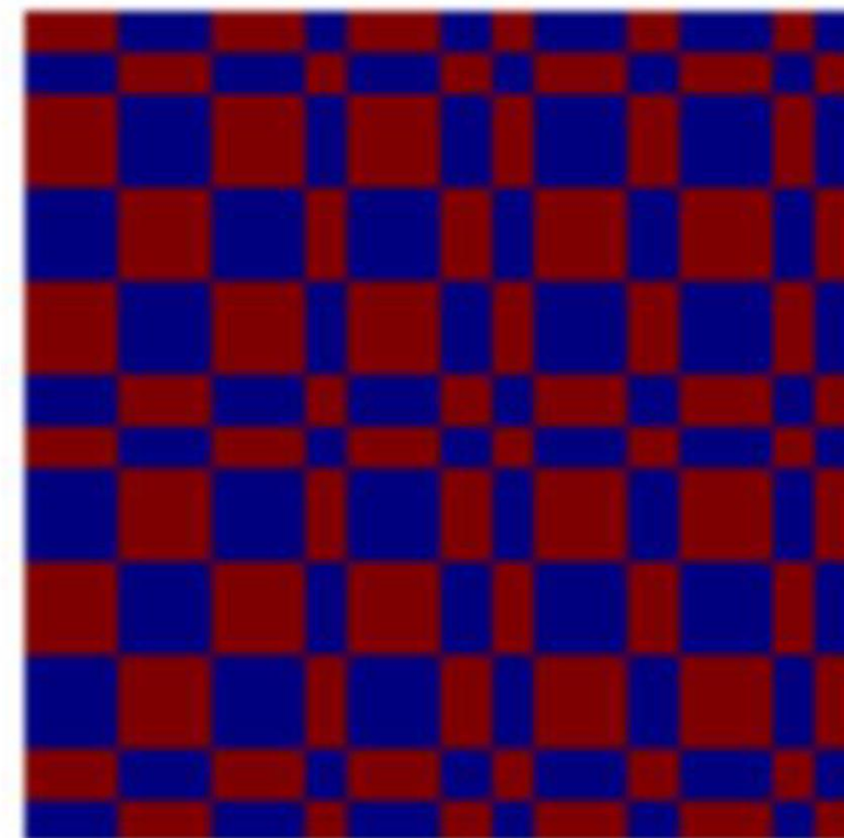
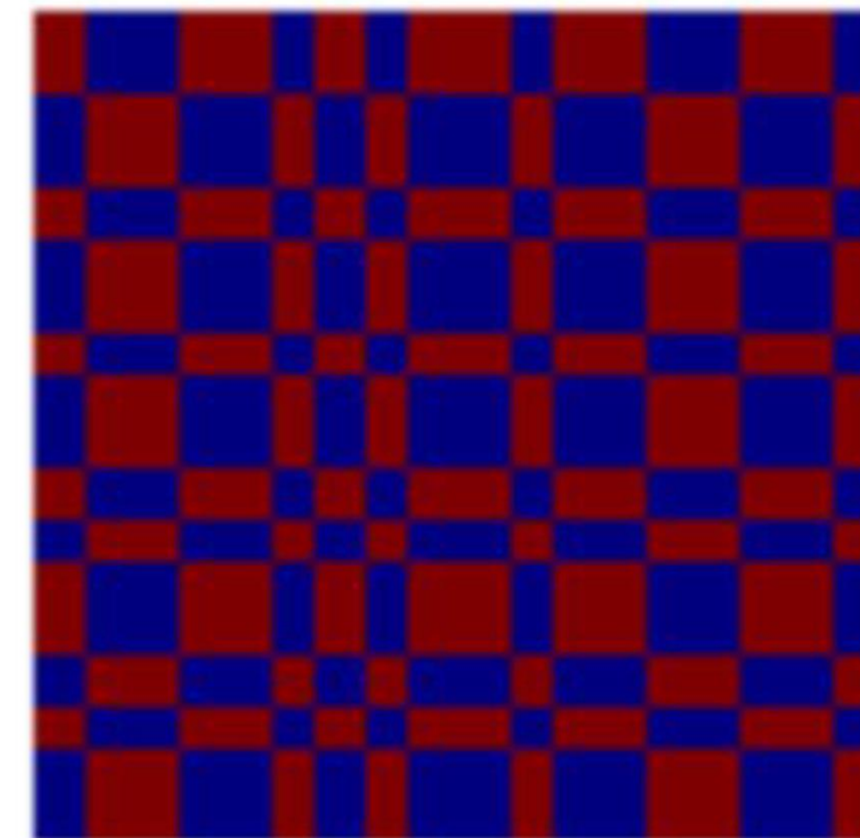
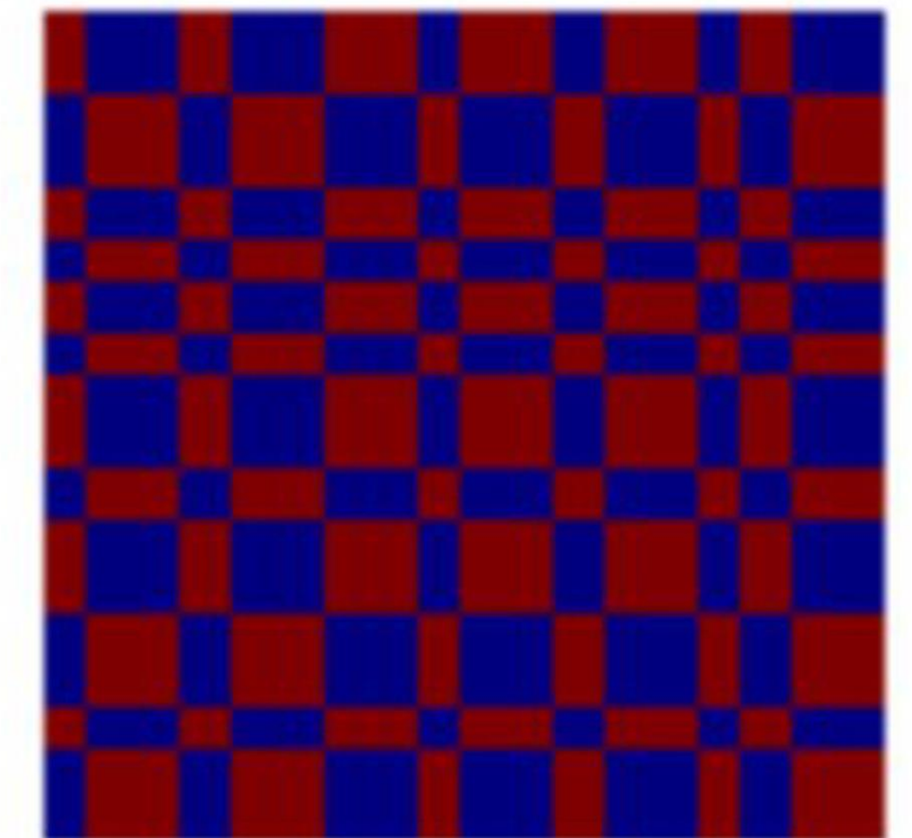
Regularization: *Fractional Pooling*

Training: Use randomized pooling regions

Testing: Average predictions from several regions

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling

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

Regularization: *Stochastic Depth*

Training: Skip some layers in the network

Testing: Use all the layer

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth

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

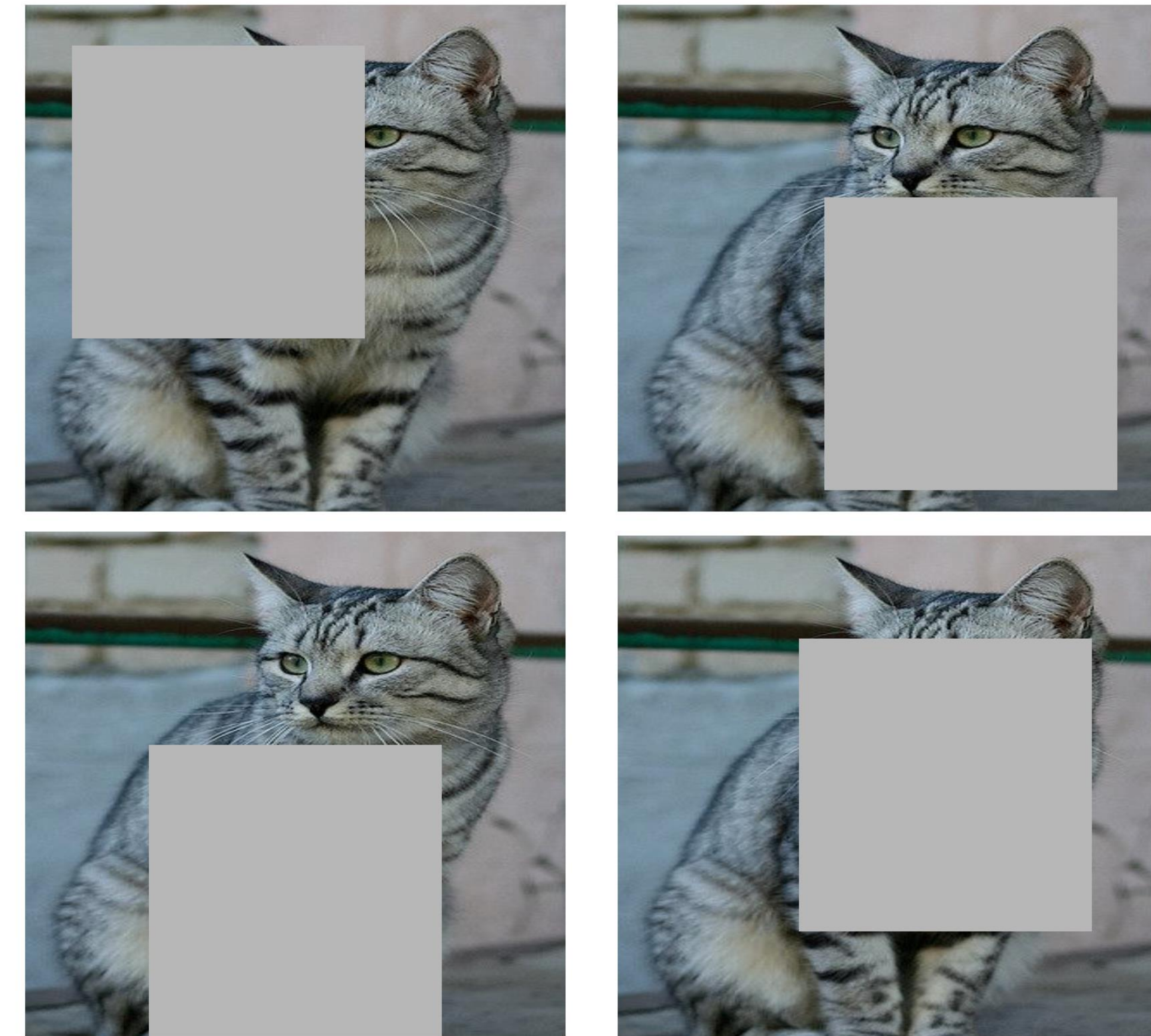
Regularization: *Cutout*

Training: Set random image regions to zero

Testing: Use full image

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout



Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

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

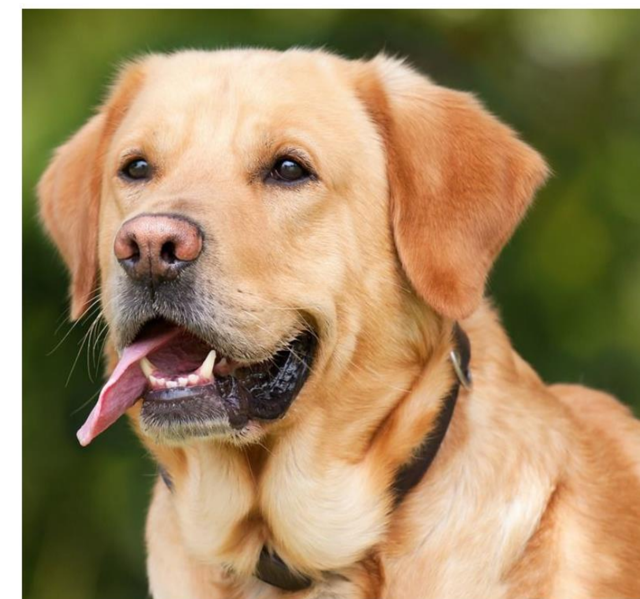
Regularization: *Mixup*

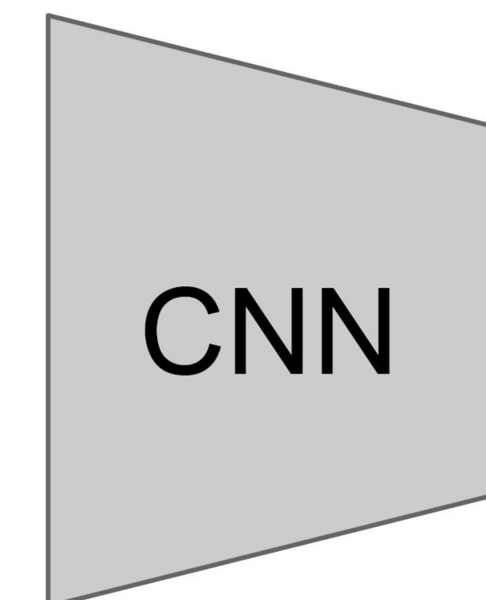
Training: Train on random blends of images

Testing: Use original images

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout
Mixup





Target label:
cat: 0.4
dog: 0.6

Randomly blend the pixels of pairs of training images, e.g. 40% cat, 60% 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

Regularization: *In practice*

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout
Mixup

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout and mixup especially for small classification datasets

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

Choosing Hyperparameters

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

Choosing Hyperparameters

Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization e.g. log(C) for softmax with C classes

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

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization

Loss not going down? LR too low, bad initialization Loss explodes to Inf or NaN? LR too high, bad initialization

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

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-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

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try: 1e-4, 1e-5, 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

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

Step 5: Refine grid, train longer

Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay

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

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

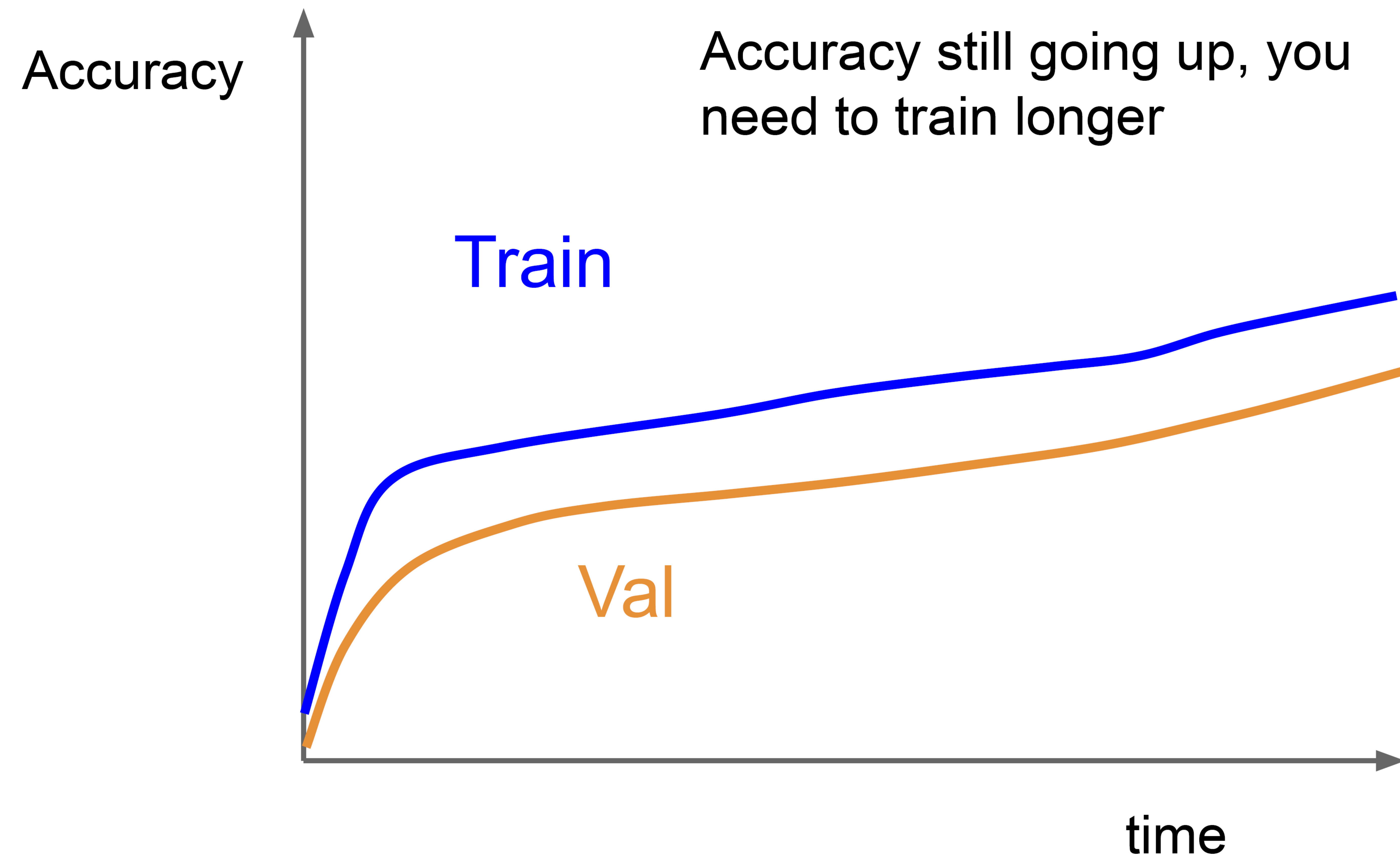
Step 4: Coarse grid, train for ~1-5 epochs

Step 5: Refine grid, train longer

Step 6: Look at loss and accuracy curves

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

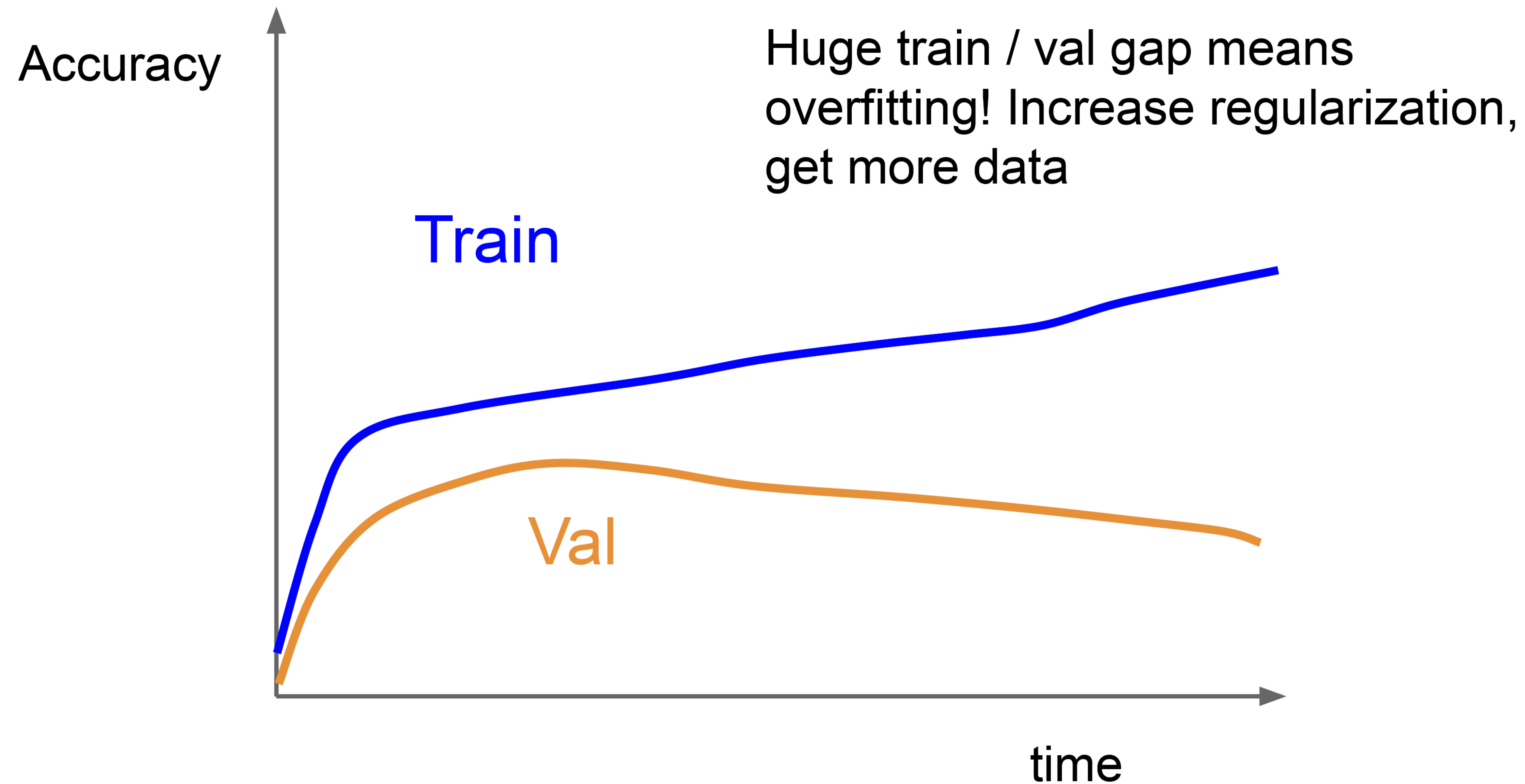
Choosing Hyperparameters



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

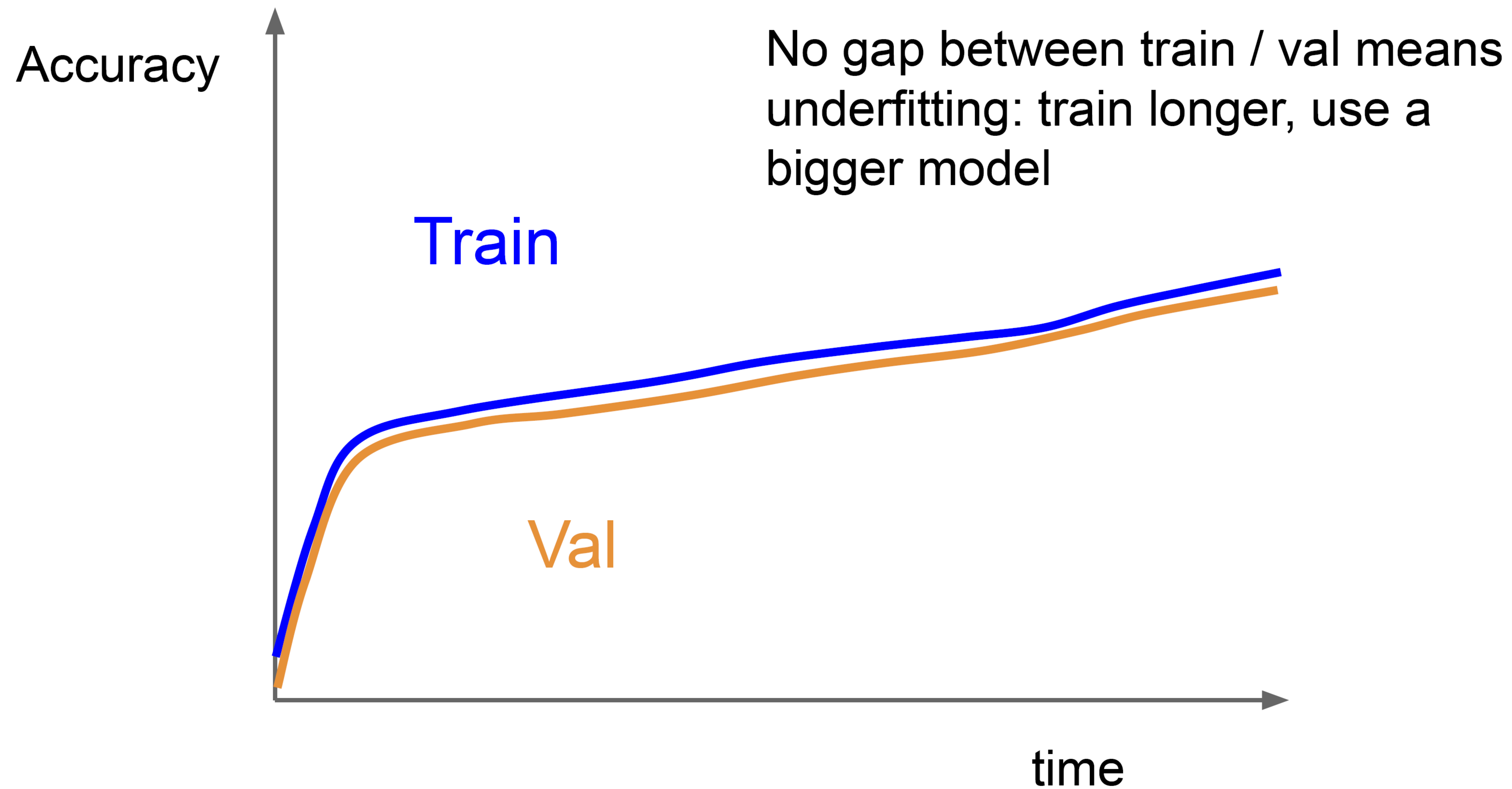
Choosing Hyperparameters



Huge train / val gap means overfitting! Increase regularization, get more data

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

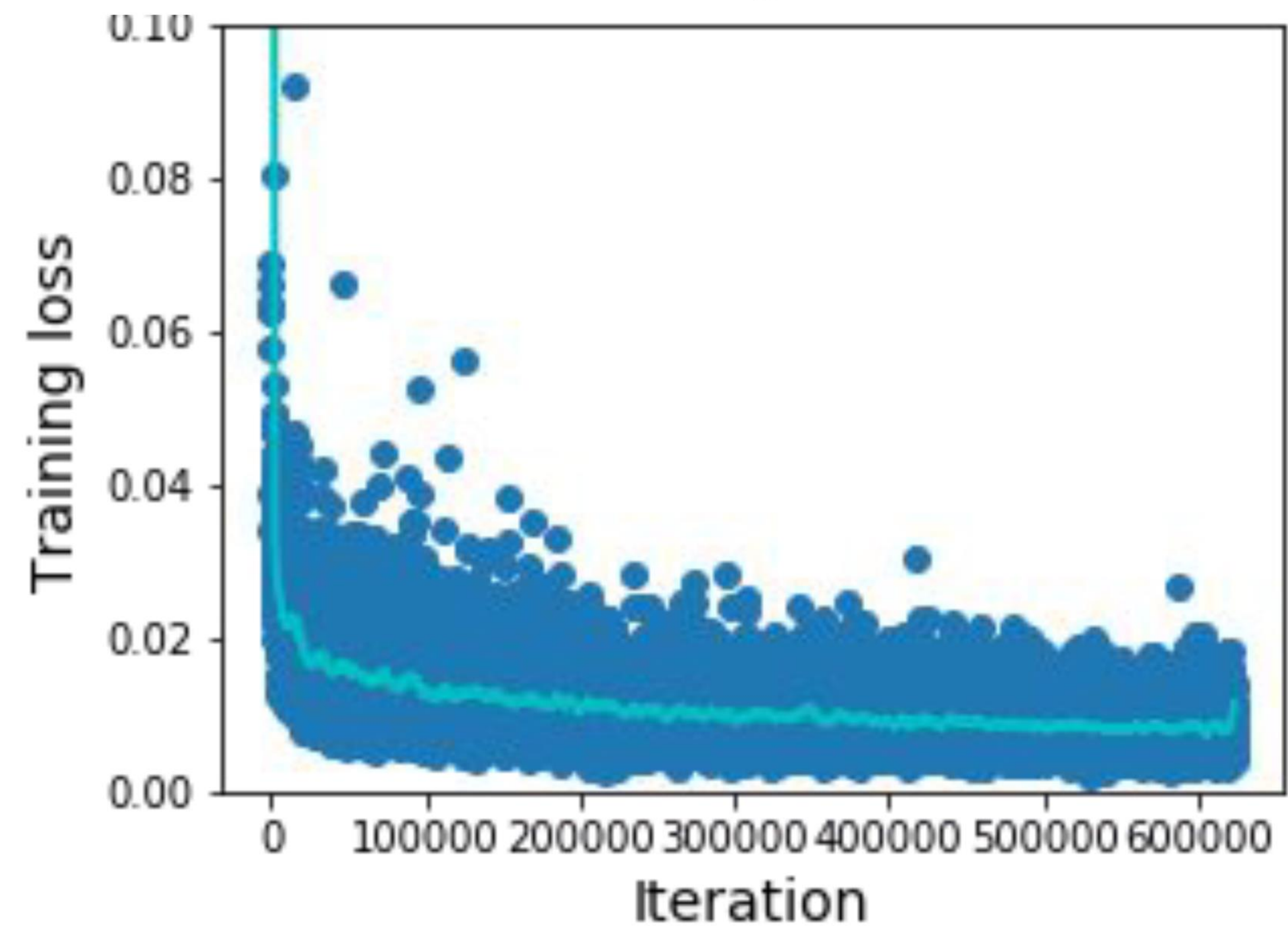
Choosing Hyperparameters

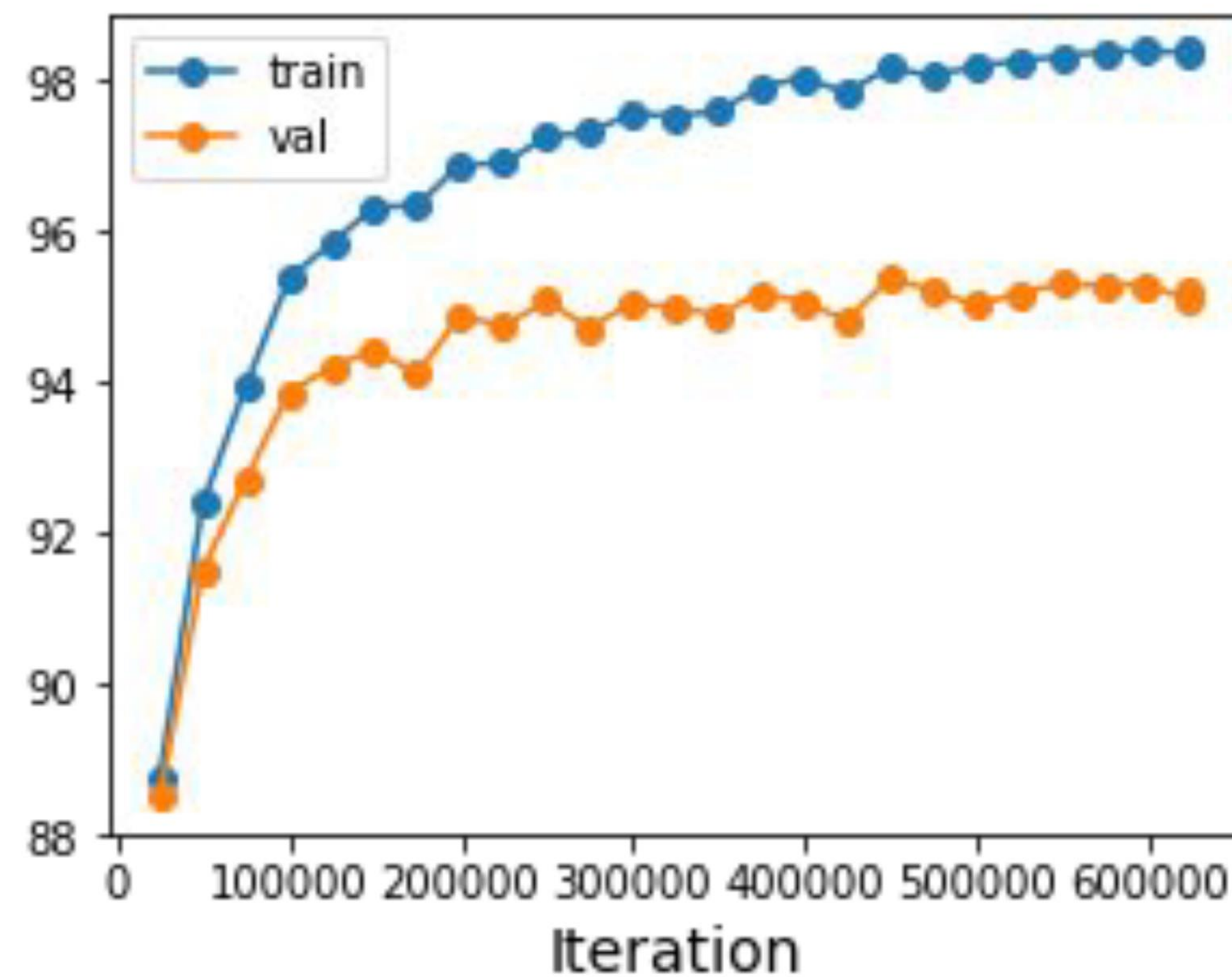


No gap between train / val means underfitting: train longer, use a bigger model

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

Look at learning curves!





Losses may be noisy, use a scatter plot and also plot moving average to see trends better

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

Visualizing & Understanding

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

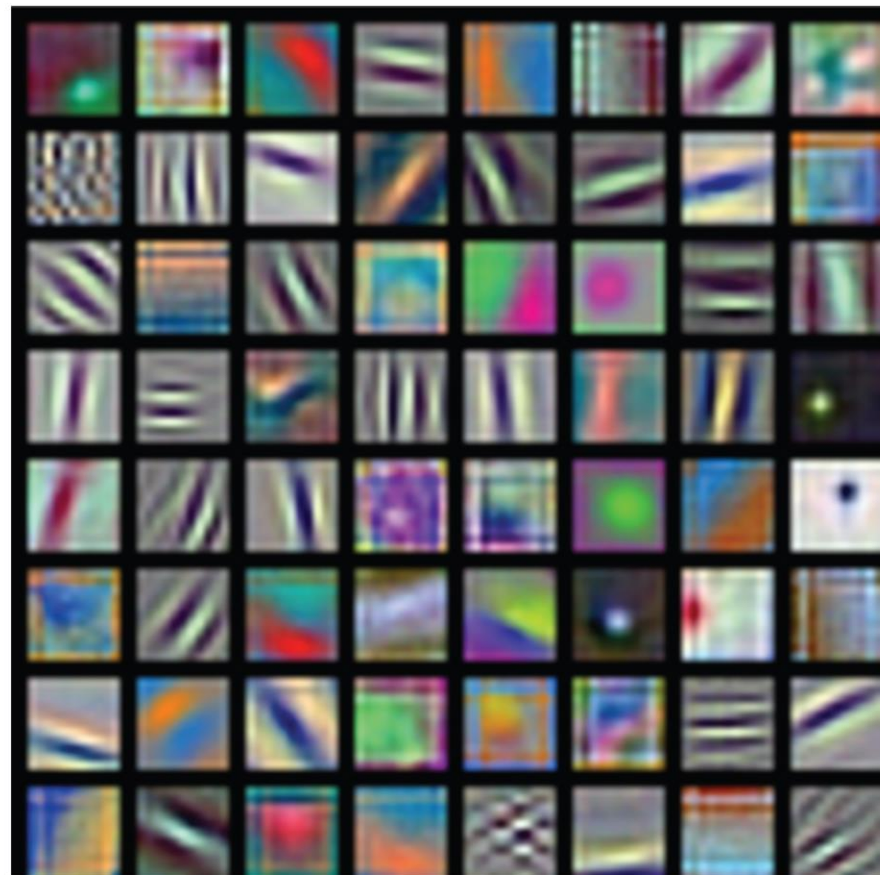
Visualizing what ConvNets learn

Several approaches for understanding and visualizing Convolutional Networks have been developed in the literature, partly as a response the common criticism that the learned features in a Neural Network are not interpretable. In this lecture we briefly survey some of these approaches and related work.

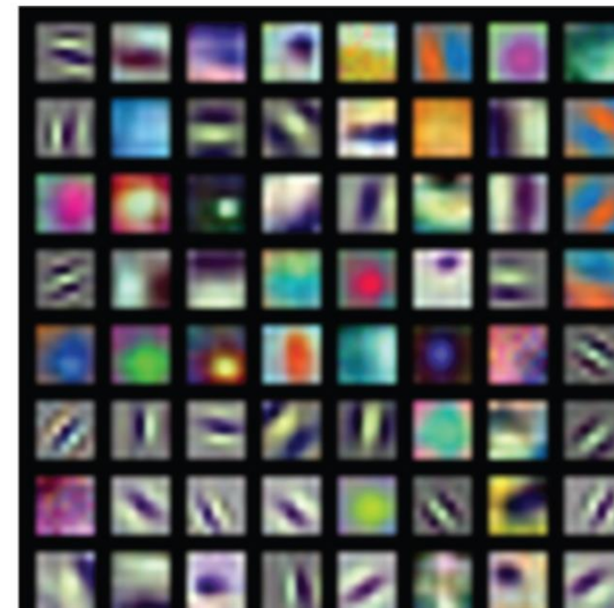
Co-financed by the European Union
Connecting Europe Facility

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

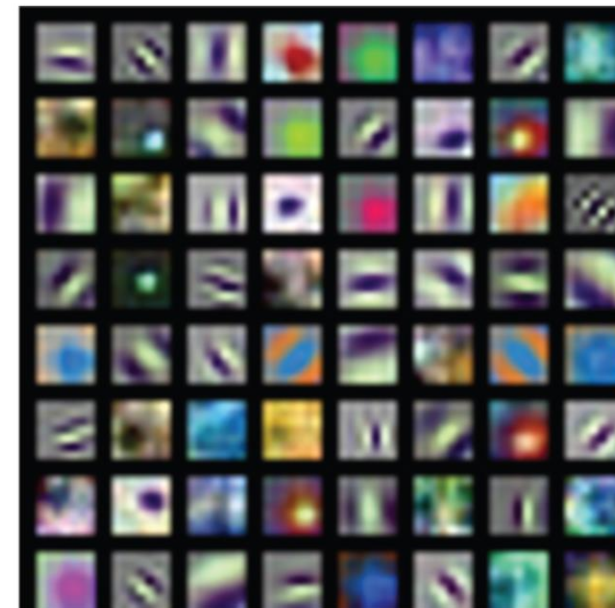
First Layer: *Visualize Filters*



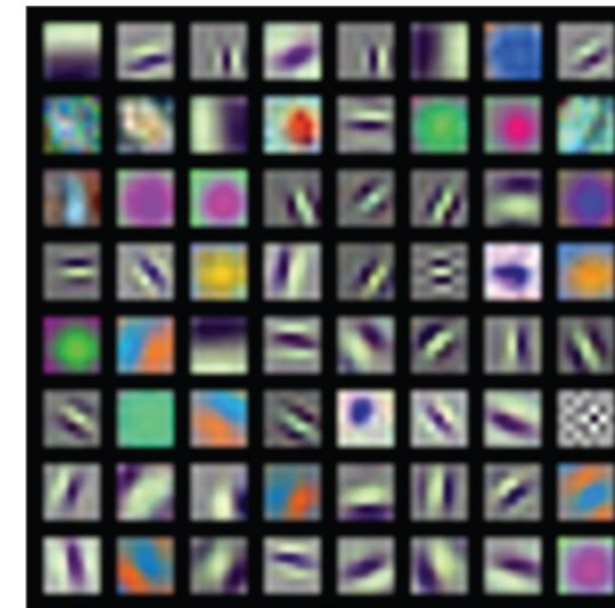
AlexNet:
64 x 3 x 11 x 11



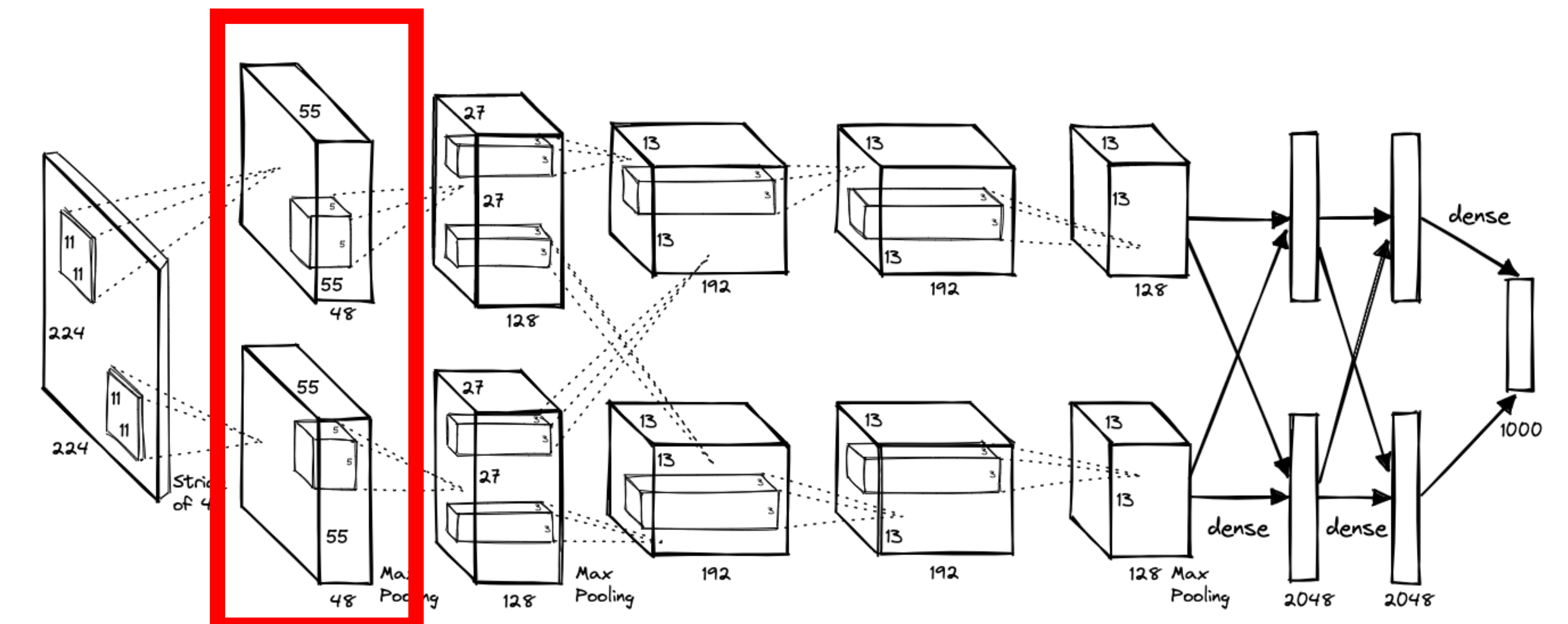
ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7



DenseNet-121:
64 x 3 x 7 x 7



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

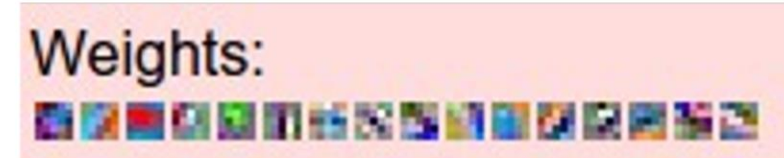
This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

First Layer: *Visualize Filters*

Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)



layer 1 weights

16 x 3 x 7 x 7

layer 2 weights

20 x 16 x 7 x 7

layer 3 weights

20 x 20 x 7 x 7

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

First Layer: *Visualize Filters*

Visualizing the filters or kernels can be useful for several reasons:

1. Understanding what the network is learning: CNNs are designed to learn features from input data automatically. By visualizing the filters, we can get an idea of what kind of features the network is detecting. For example, if a filter is detecting edges, we might see diagonal lines in the filter visualization.
2. Debugging the network: Visualizing the filters can help us understand why a network might not be performing well. If the filters appear to be detecting irrelevant features or noise in the input data, it may be an indication that the network is not learning the correct features.
3. Transfer learning: When using pre-trained CNNs for transfer learning, visualizing the filters can help us understand how the network was trained and what kind of features it has learned. This can inform us on how to fine-tune the network for our specific use case.
4. Improving network performance: Visualizing the filters can help us identify which filters are most important for the network's performance. This can inform us on how to adjust the network architecture, such as increasing the number of filters or changing the filter size.

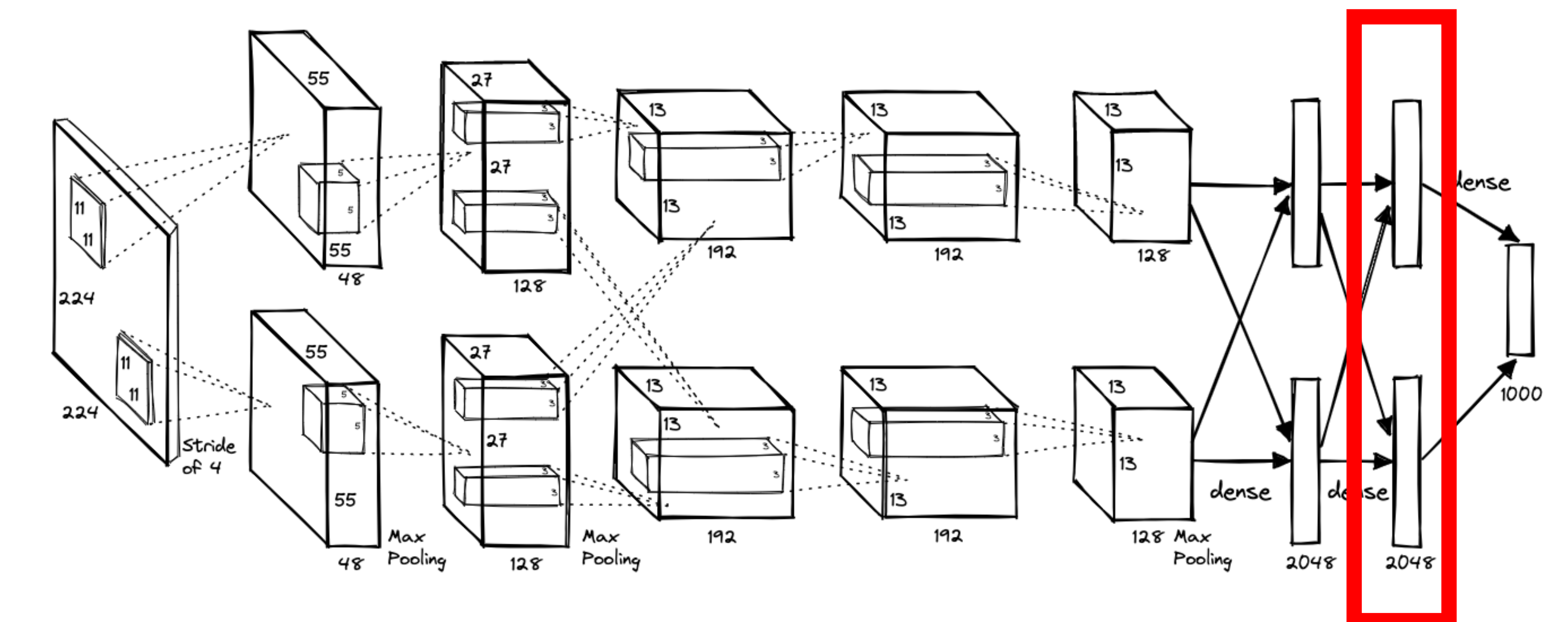
Co-financed by the European Union
Connecting Europe Facility

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

Last Layer: *Nearest Neighbors*

Recall: Nearest neighbors in pixel space

Test image L2 Nearest neighbors in feature space



Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.

Co-financed by the European Union
Connecting Europe Facility

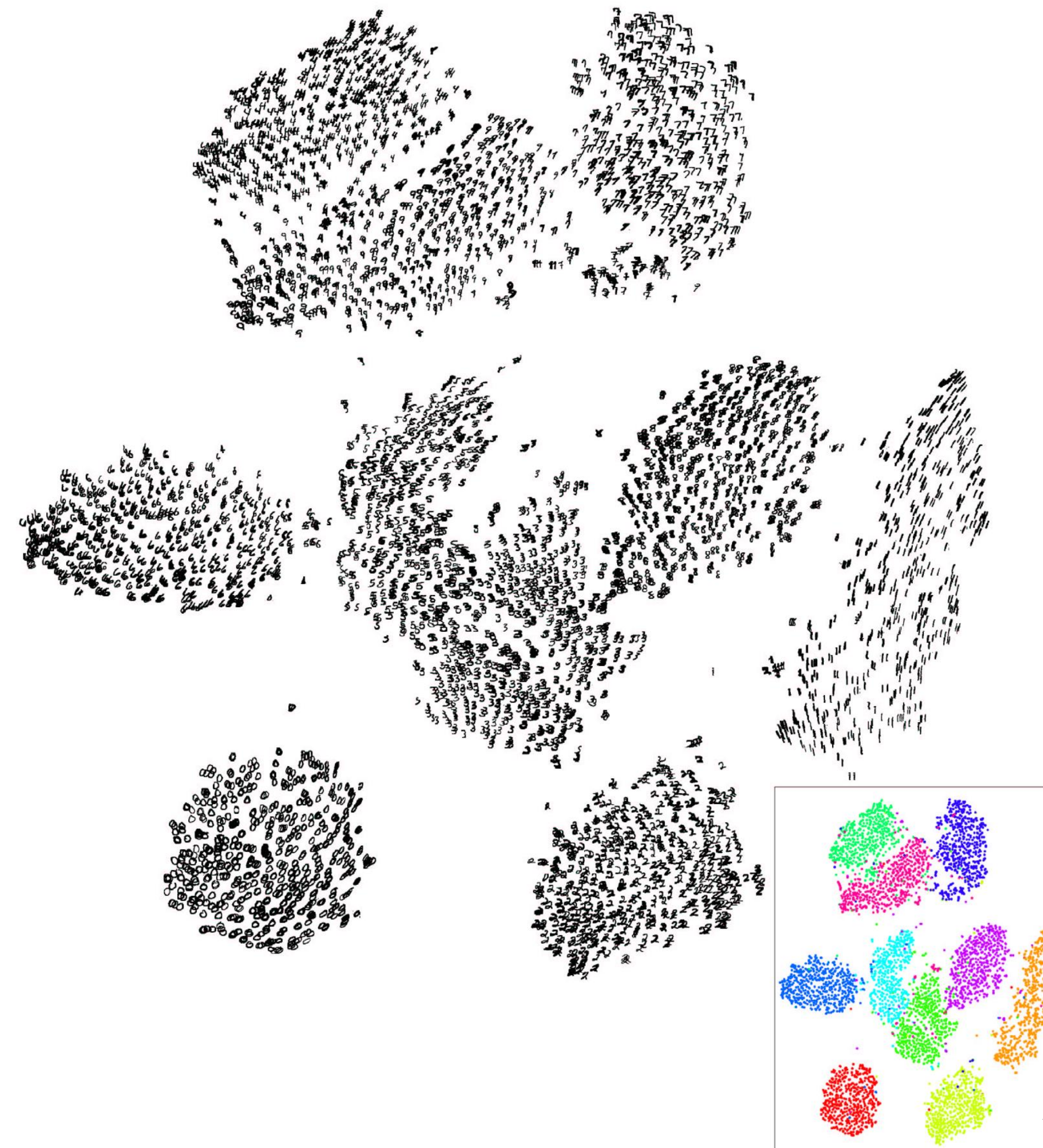
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

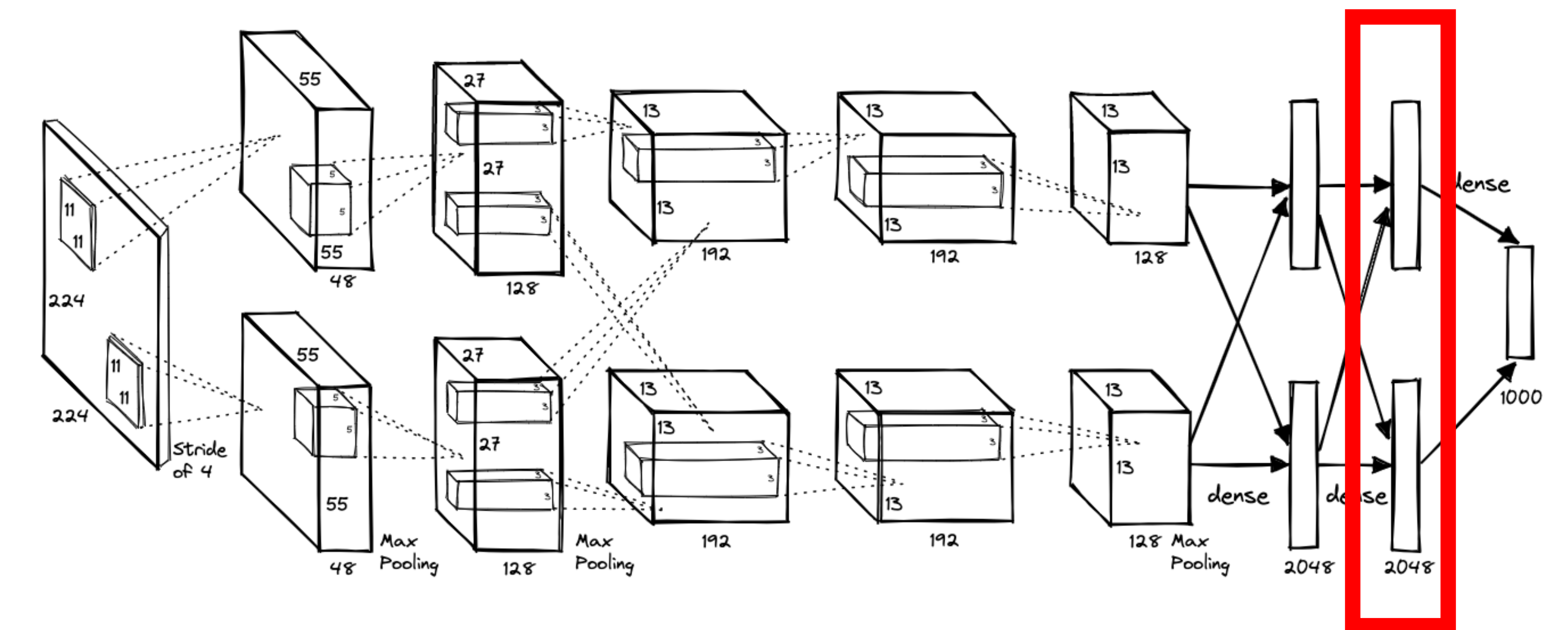
Last Layer: *Nearest Neighbors*

Visualize the "space" of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

More complex: **t-SNE**



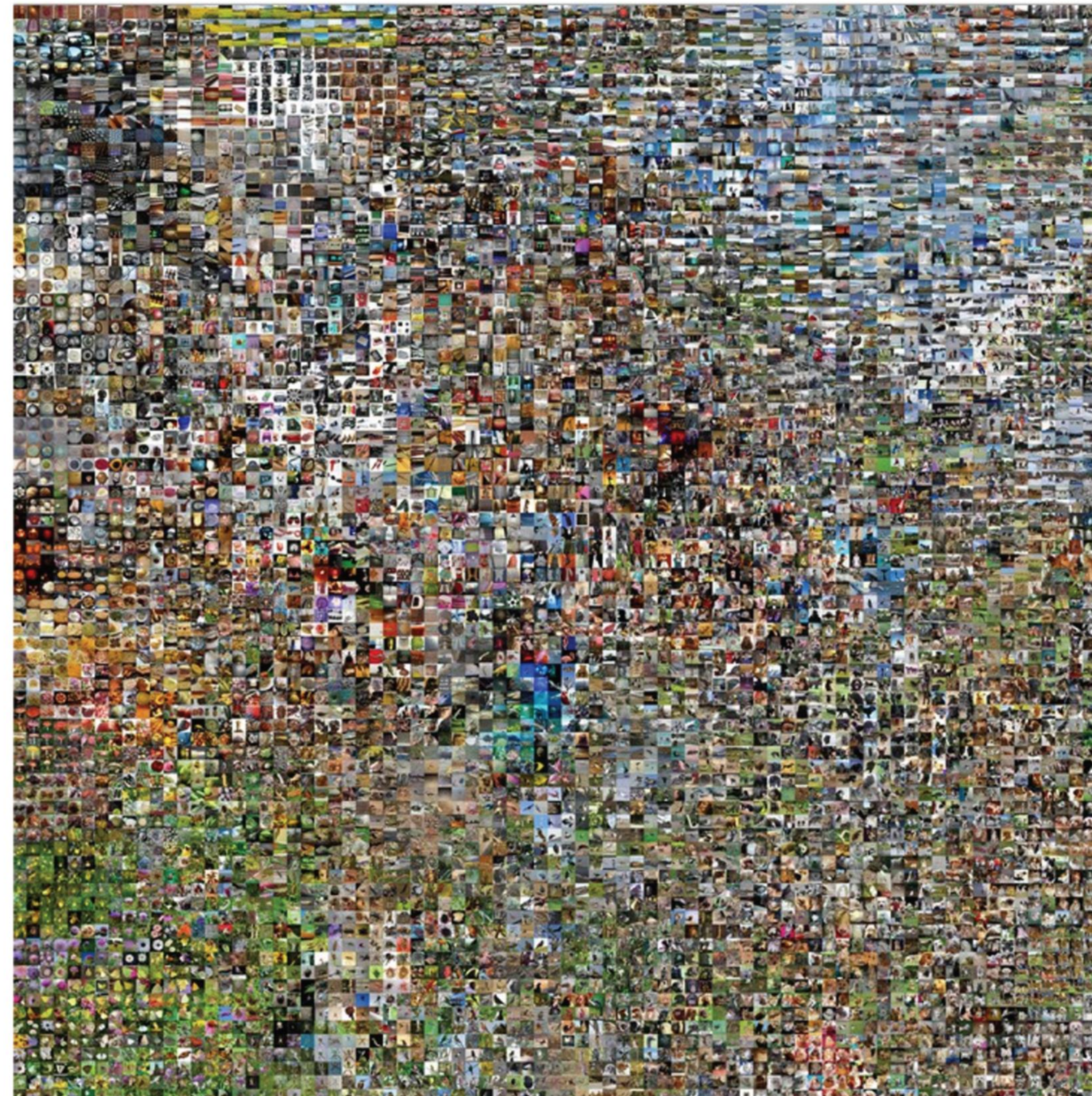


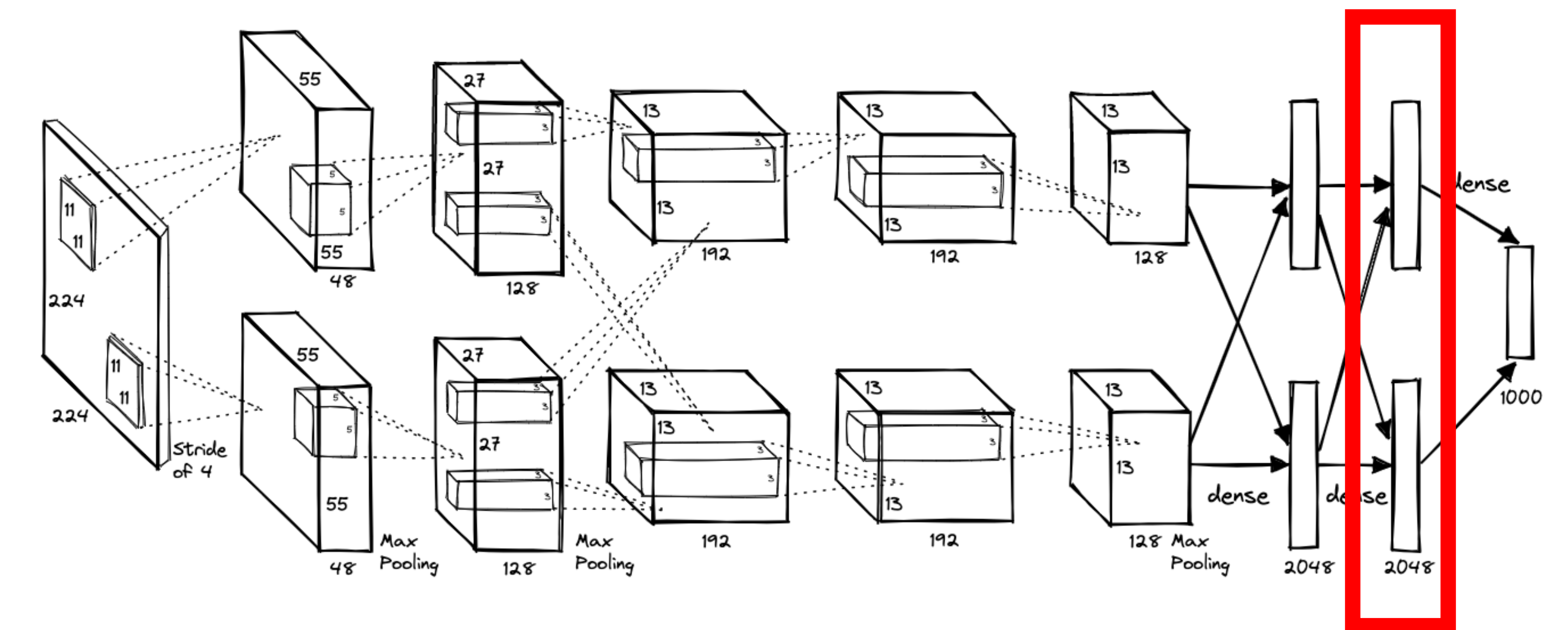
Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008

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

Last Layer: *Dimensionality Reduction*





See high-resolution versions at http://cs.stanford.edu/people/karpathy/cnnembed/

Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.

Last Layer: *Dimensionality Reduction*

Visualization: Reducing the dimensionality of the feature vectors to 2 dimensions makes it possible to visualize the high-dimensional feature space in a 2D scatter plot. This can help us gain insight into the relationship between different features in the space and identify patterns that may not be apparent in the high-dimensional space.

Interpretability: By visualizing the feature space in 2D, we can gain a better understanding of how different features are related to each other and identify clusters of similar features. This can help us interpret the output of the network and understand how it is making its predictions.

Comparison: By visualizing the feature space for different inputs or classes, we can compare the distribution of features between them. This can help us identify differences between classes and understand what features the network is using to distinguish between them.

Optimization: Reducing the dimensionality of the feature vectors can also be useful for optimization purposes, as it makes it easier to perform operations such as clustering or classification in the feature space.

Co-financed by the European Union
Connecting Europe Facility

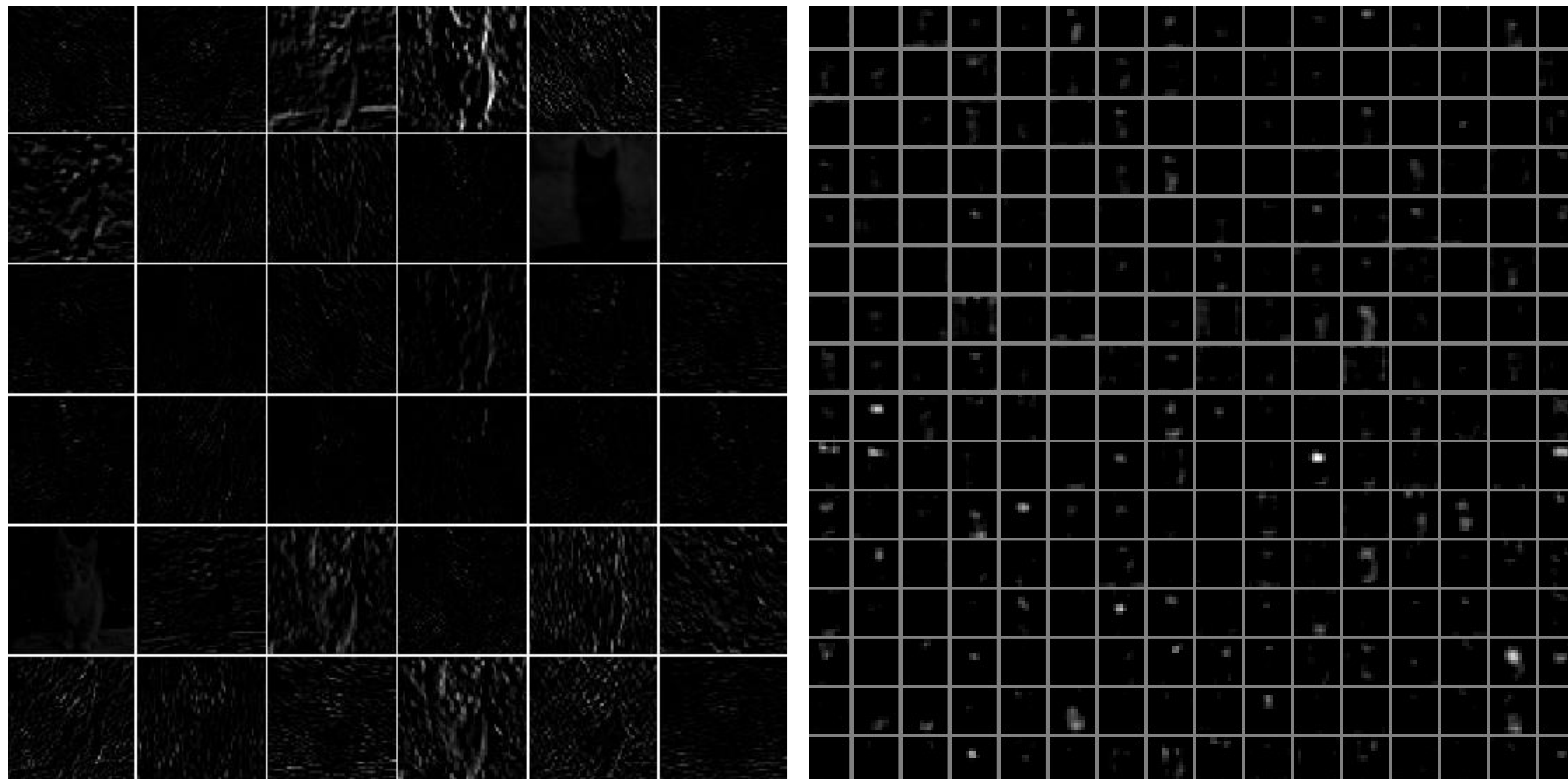
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Visualizing Activations

Layer Activations. The most straight-forward visualization technique is to show the activations of the network during the forward pass. For ReLU networks, the activations usually start out looking relatively blobby and dense, but as the training progresses the activations usually become sparser and more localized. One dangerous pitfall that can be easily noticed with this visualization is that some activation maps may be all zero for many different inputs, which can indicate *dead* filters, and can be a symptom of high learning rates.

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

Visualizing Activations



Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Every box shows an activation map corresponding to some filter. Notice that the activations are sparse (most values are zero, in this visualization shown in black) and mostly local.

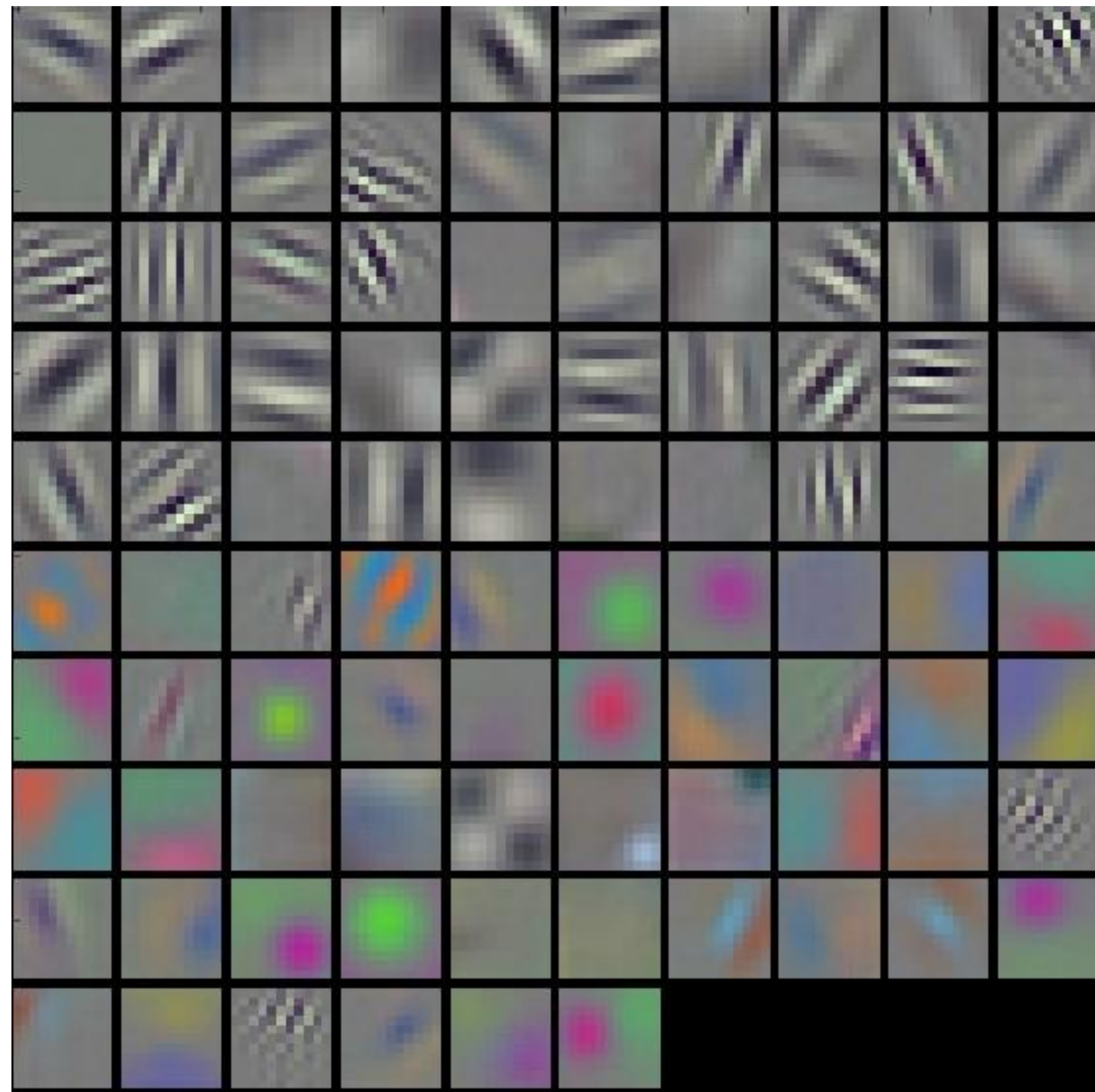
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

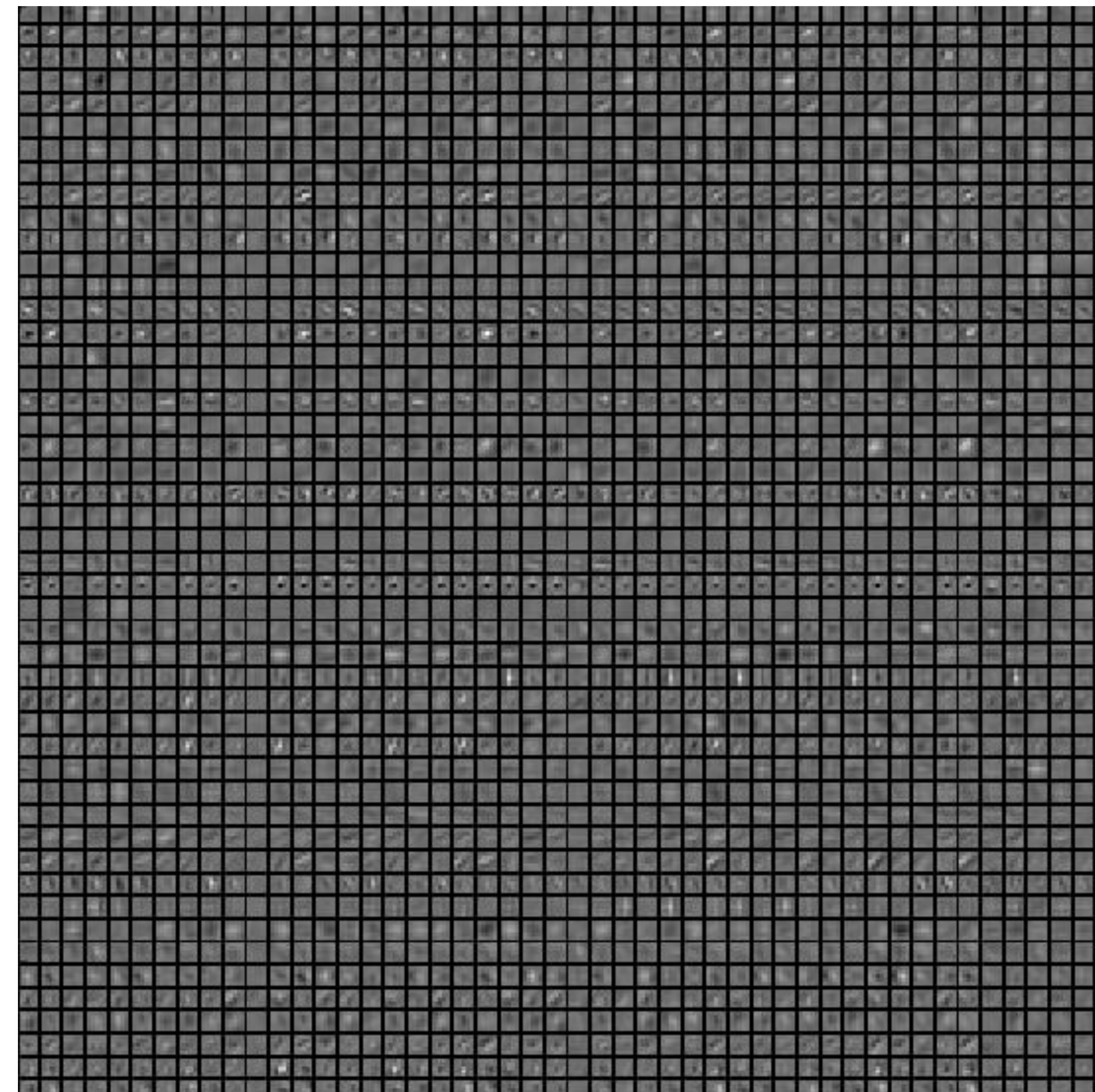
Visualizing Activations

Conv/FC Filters. The second common strategy is to visualize the weights. These are usually most interpretable on the first CONV layer which is looking directly at the raw pixel data, but it is possible to also show the filter weights deeper in the network. The weights are useful to visualize because well-trained networks usually display nice and smooth filters without any noisy patterns. Noisy patterns can be an indicator of a network that hasn't been trained for long enough, or possibly a very low regularization strength that may have led to 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

Visualizing Activations



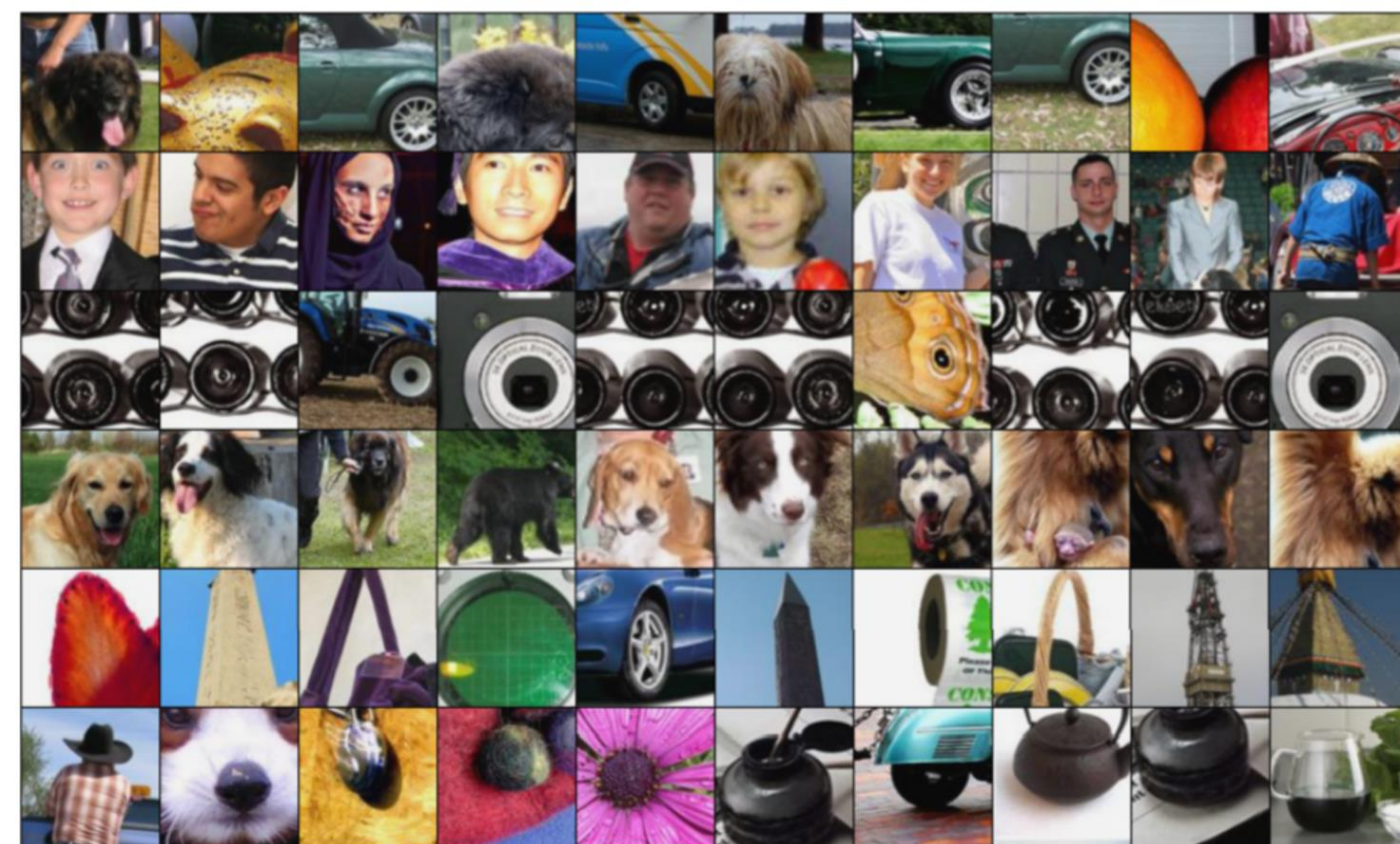


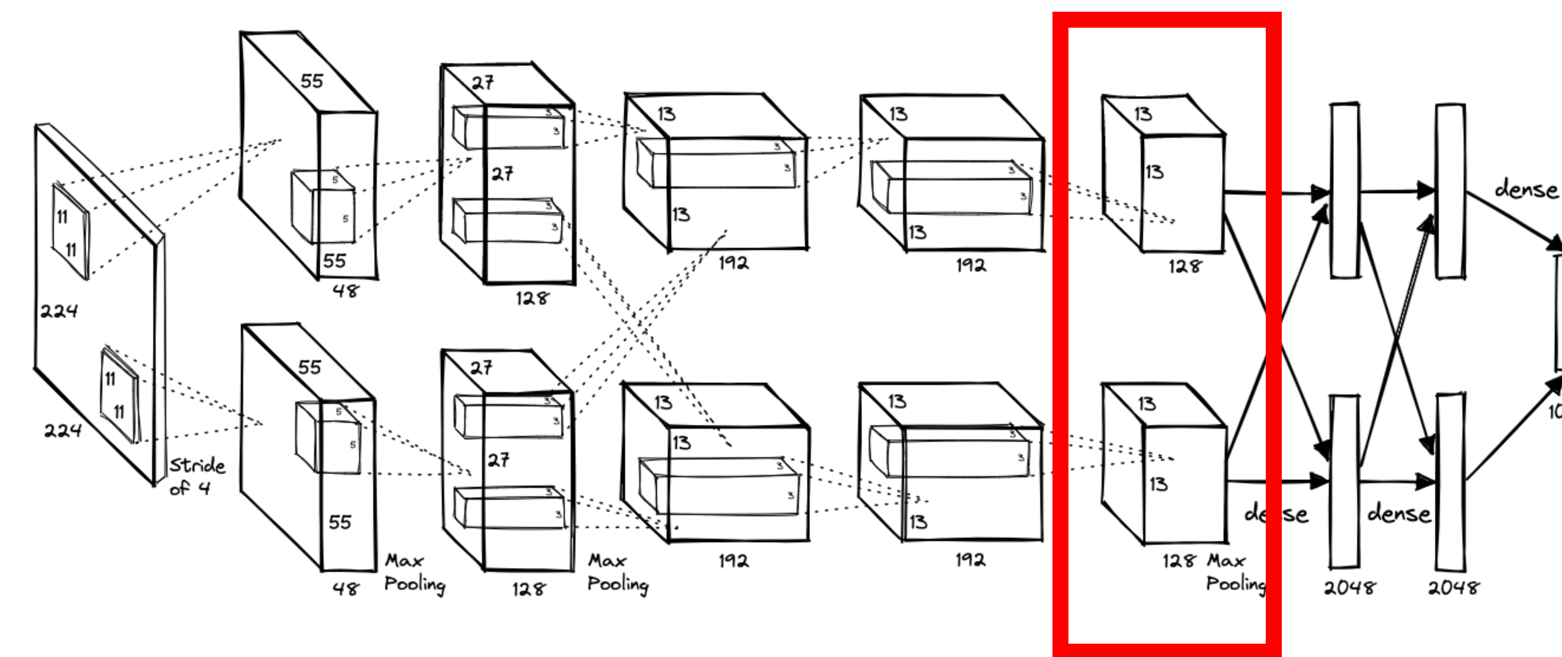
Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained AlexNet. Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

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

Maximally Activating Patches





Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

Run many images through the network, record values of chosen channel

Visualize image patches that correspond to maximal activations

Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

Saliency maps

Saliency maps are a visualization technique that highlights the most important regions of an input image that are relevant to a specific output of a neural network. Saliency maps can be used to understand the features and patterns that a neural network is using to make its predictions.

Saliency maps are typically generated by backpropagating the gradient of the output with respect to the input image. This produces a map that indicates how each pixel in the input image contributes to the output. By highlighting the pixels with the highest contributions, we can generate a saliency map that shows which regions of the input image are most important for the output.

Saliency maps are a powerful visualization tool that can be used to gain insights into the workings of neural networks and diagnose problems with their performance.

Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 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

Saliency maps

Saliency maps can be used for a variety of tasks, including:

Interpretation: Saliency maps can help us understand how a neural network is making its predictions by highlighting the regions of the input image that are most relevant to the output. This can provide insights into the features and patterns that the network is using to make its predictions.

Debugging: Saliency maps can also be used to diagnose problems with a neural network. For example, if the saliency map highlights irrelevant regions of the input image, it may be an indication that the network is not learning the correct features.

Localization: Saliency maps can be used to identify the location of objects or features in an image. By highlighting the regions of the input image that are most important for a specific output, we can identify the regions of the image that contain the relevant objects or features.

Adversarial attacks: Saliency maps can be used to generate adversarial examples that are designed to fool a neural network. By modifying the input image to maximize the saliency map for a specific output, we can generate images that are classified incorrectly by the network.

Springenberg et al. "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 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

Which pixels matter: *Saliency via Occlusion*

Example of visualization of features in a fully trained model. For layers 2-5 it shows the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. The reconstructed patterns from the validation set is shown that cause high activations in a given feature map. For each feature map we also show the corresponding image patches.

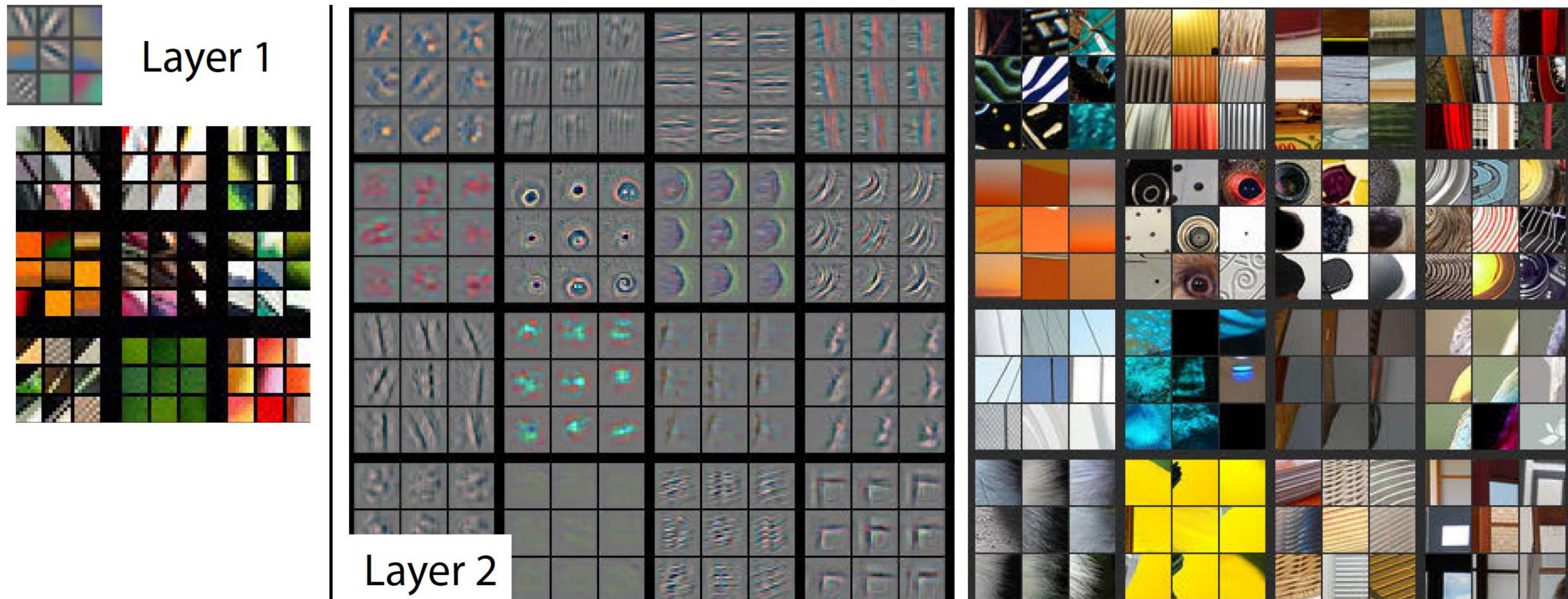
Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1).

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 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

Which pixels matter: *Saliency via Occlusion*

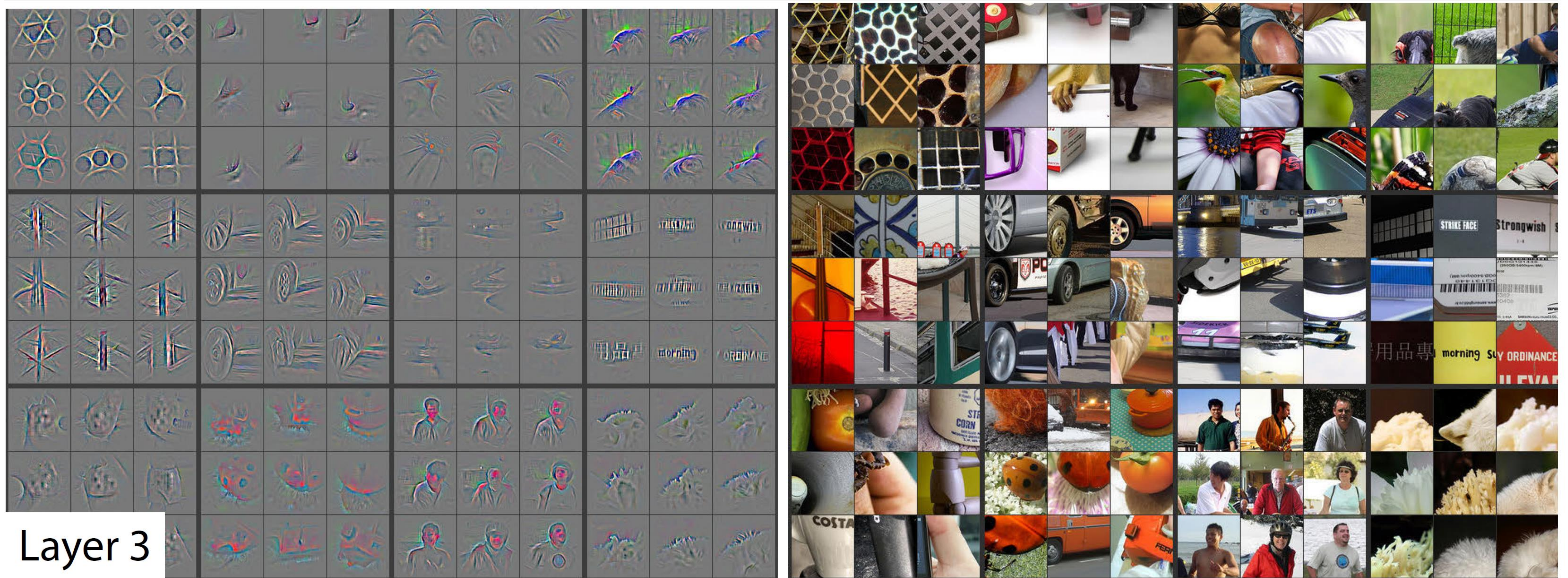


Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

Co-financed by the European Union
Connecting Europe Facility

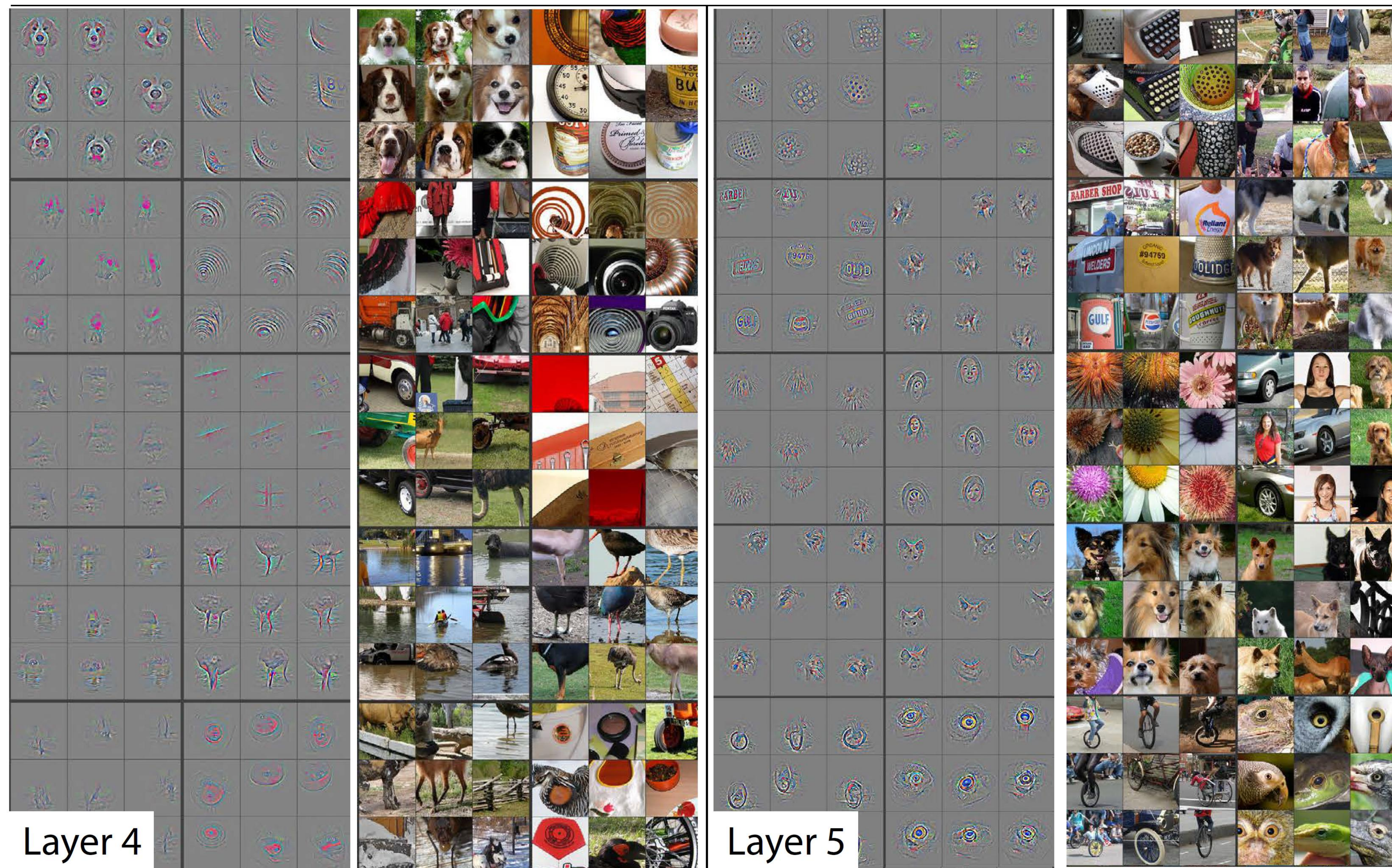
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Which pixels matter: *Saliency via Occlusion*



Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

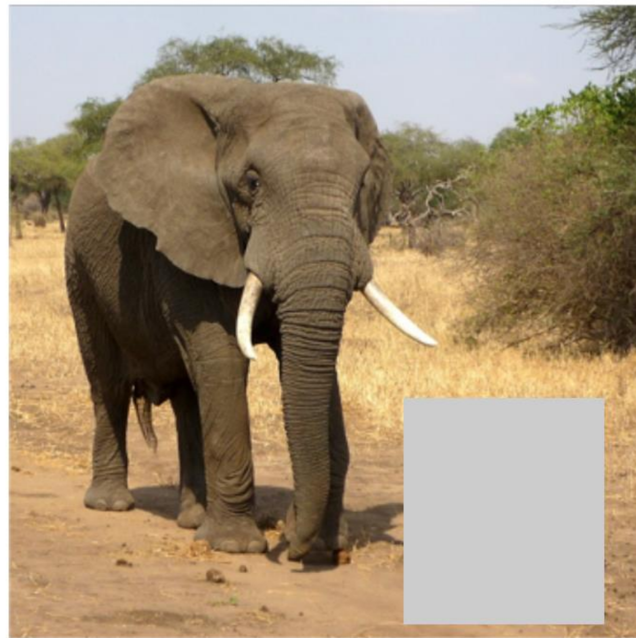
Which pixels matter: *Saliency via Occlusion*

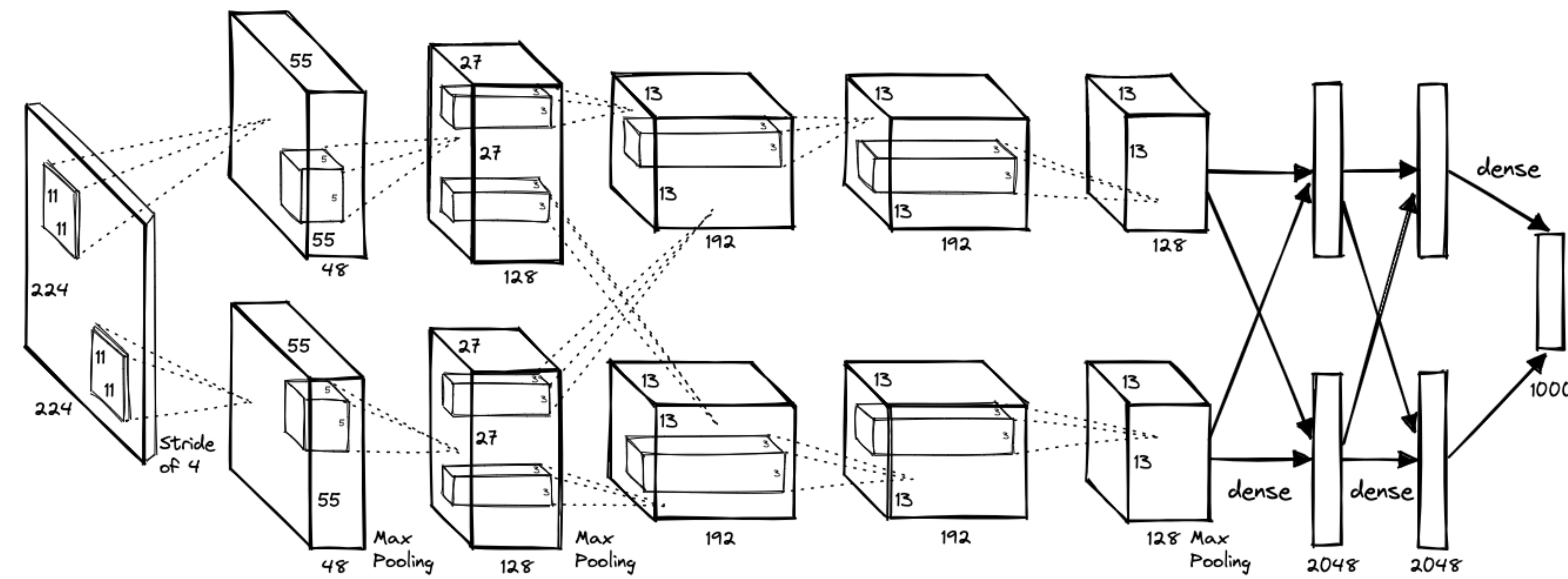


Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

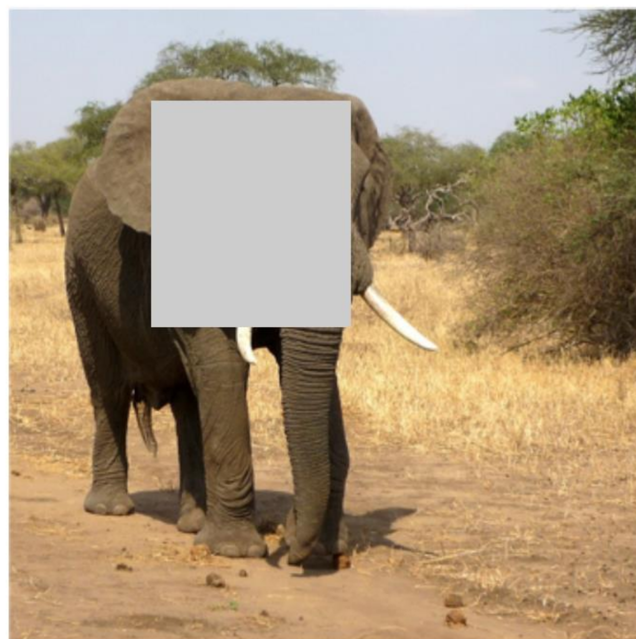
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

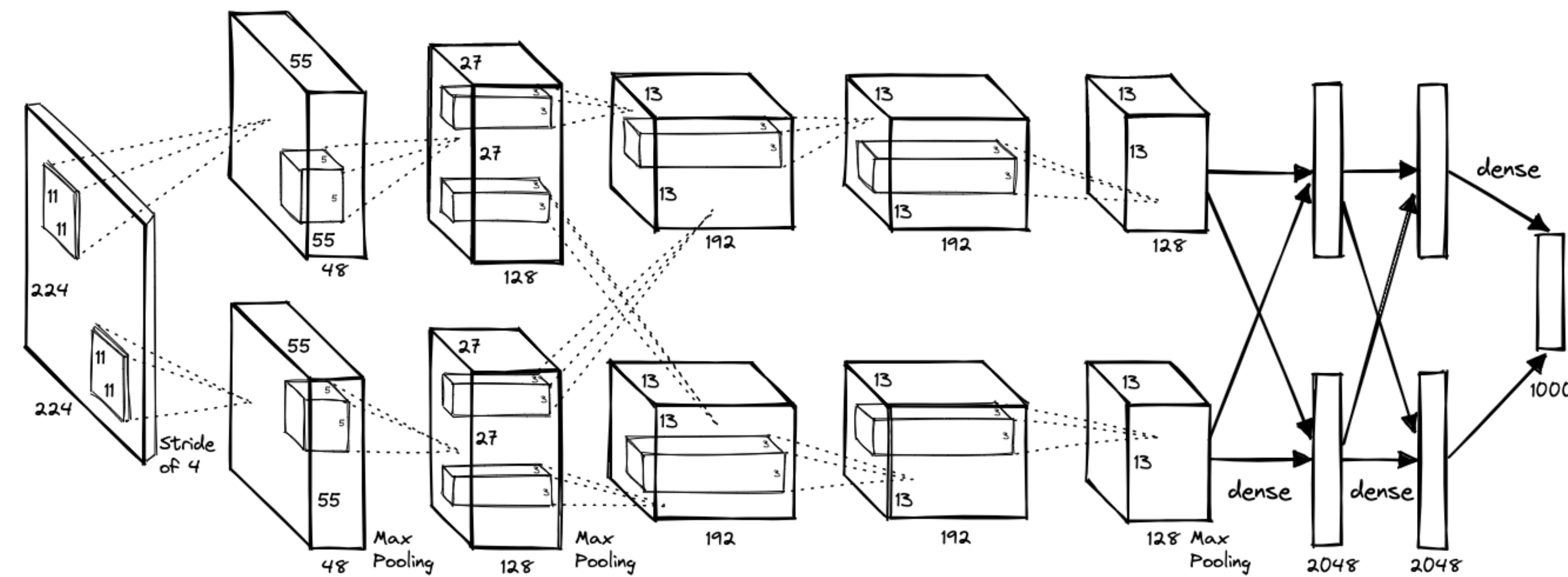
Which pixels matter: *Saliency via Occlusion*





P(elephant) = 0.95



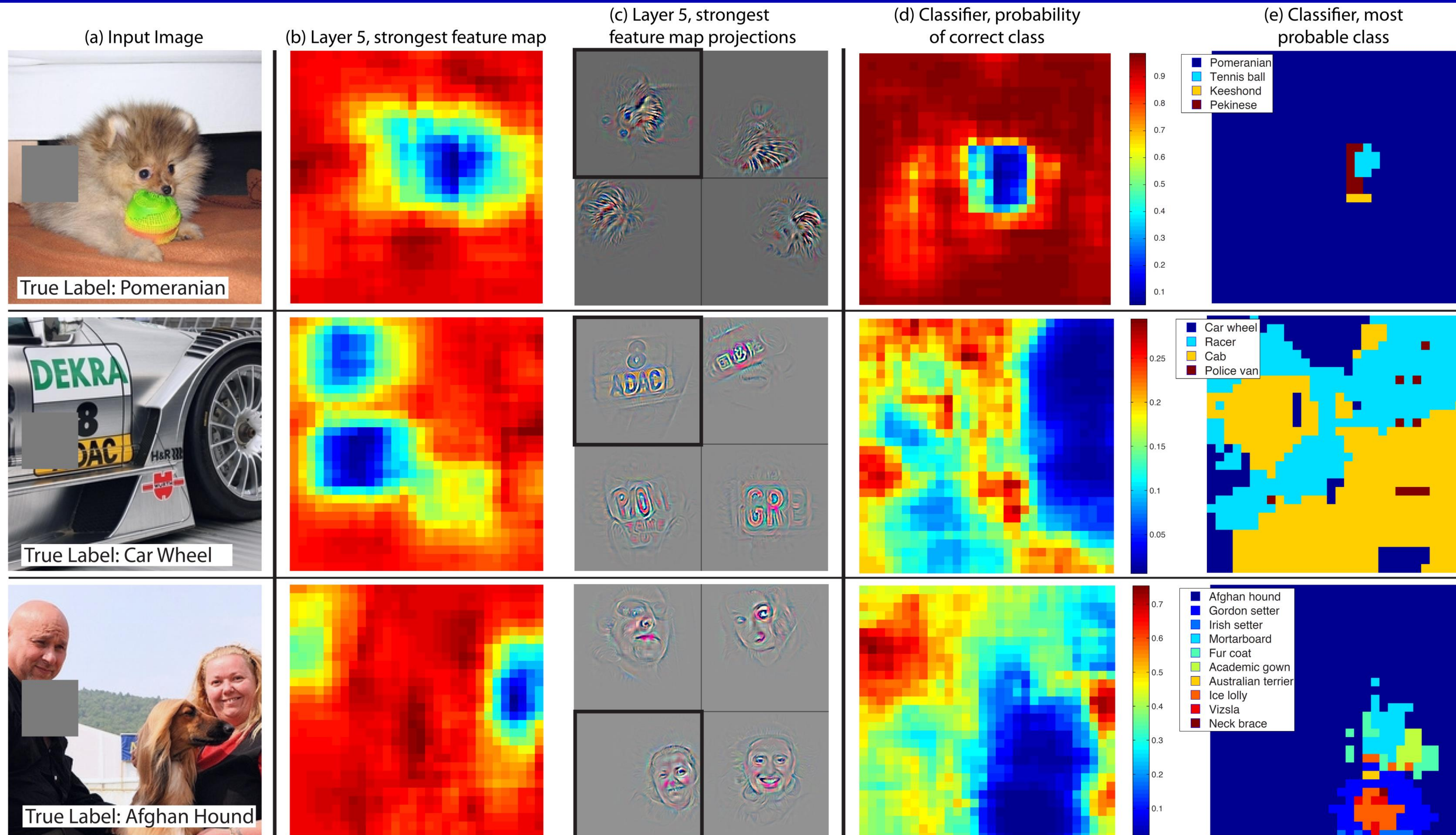


P(elephant) = 0.75

Mask part of the image before feeding to CNN, check how much predicted probabilities change

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Which pixels matter: *Saliency via Occlusion*



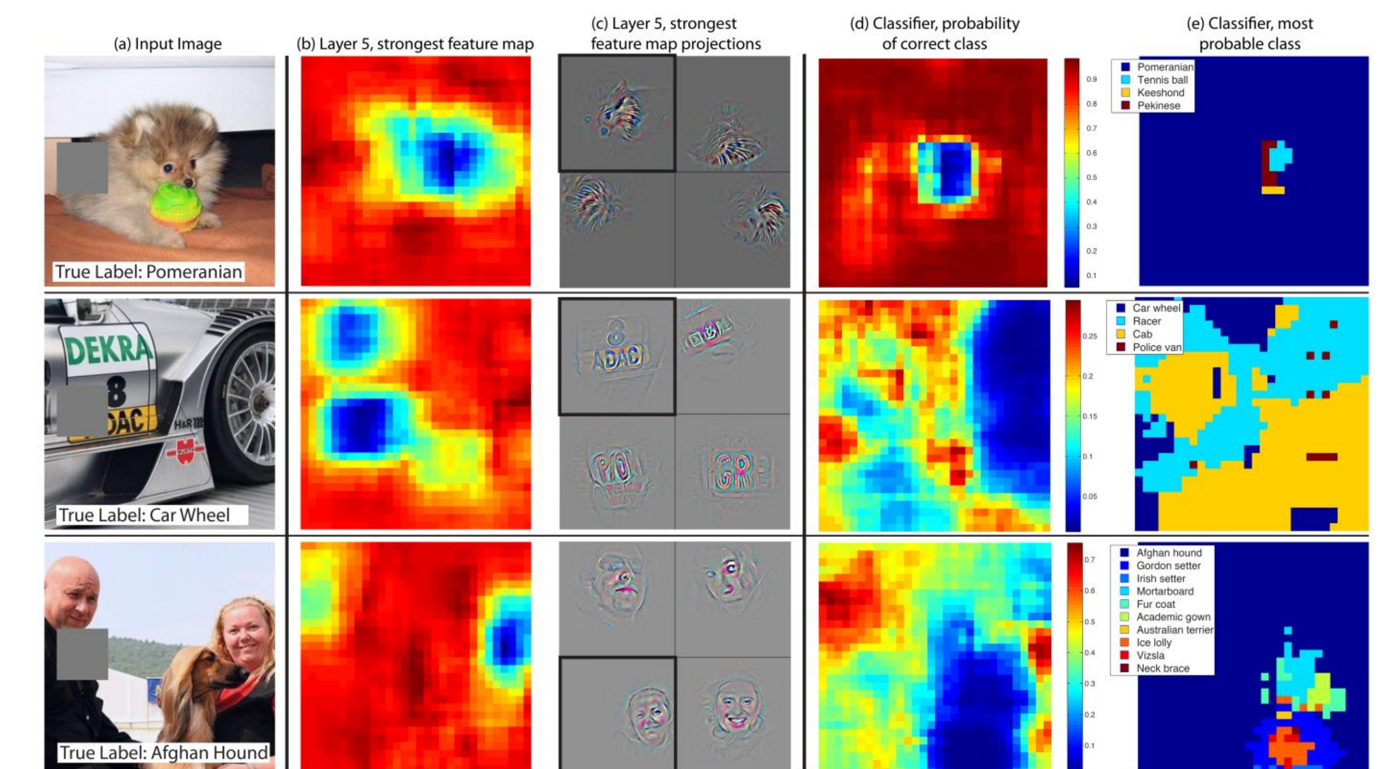
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

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

Which pixels matter: *Saliency via Occlusion*

Three test examples where we systematically cover up different portions of the scene with a gray square (1st column) and see how the top (layer 5) feature maps ((b) & (c)) and classier output ((d) & (e)) changes.
(b): for each position of the gray scale, we record the total activation in one layer 5 feature map (the one with the strongest response in the un-occluded image).
(c): a visualization of this feature map projected down into the input image (black square), along with visualizations of this map from other images. The first-row example shows the strongest feature to be the dog's face. When this is covered-up the activity in the feature map decreases (blue area in (b)).



(d): a map of correct class probability, as a function of the position of the gray square. E.g. when the dog's face is obscured, the probability for "pomeranian" drops signifficantly. (e): the most probable label as a function of occluder position. E.g. in the 1st row, for most locations it is "pomeranian", but if the dog's face is obscured but not the ball, then it predicts \tennis ball". In the 2nd example, text on the car is the strongest feature in layer 5, but the classier is most sensitive to the wheel. The 3rd example contains multiple objects. The strongest feature in layer 5 picks out the faces, but the classier is sensitive to the dog (blue region in (d)), since it uses multiple feature maps.

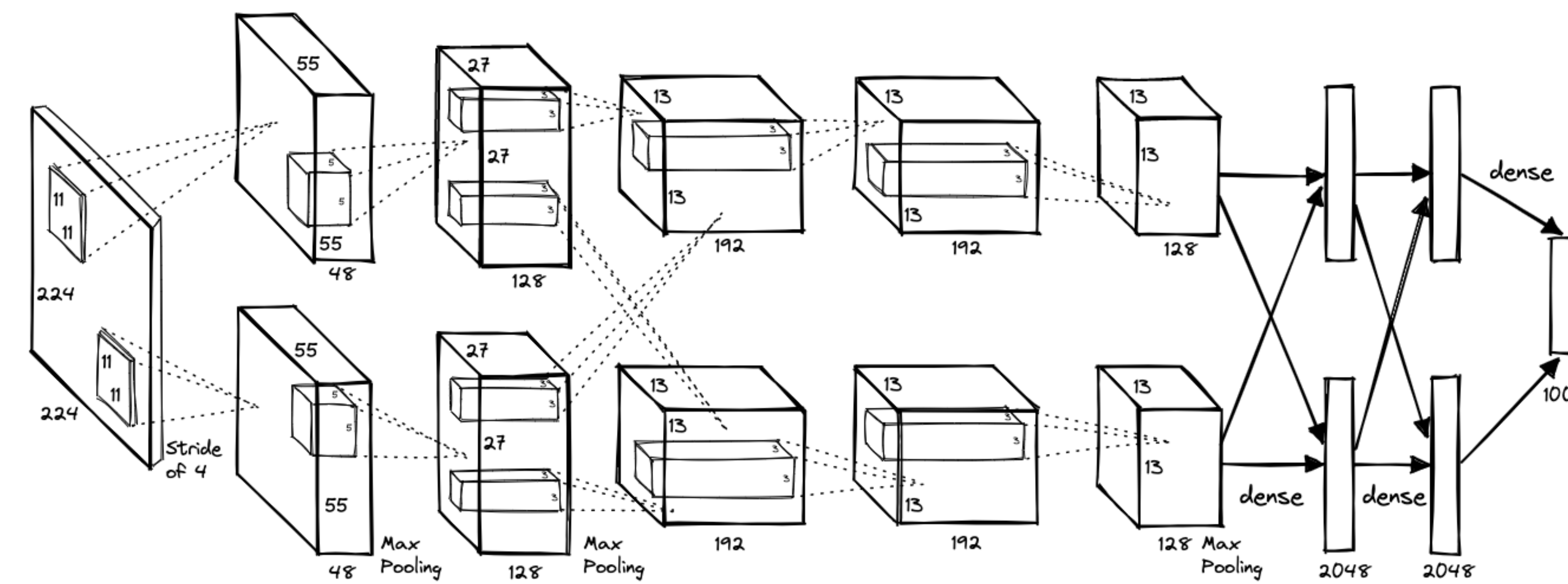
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

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

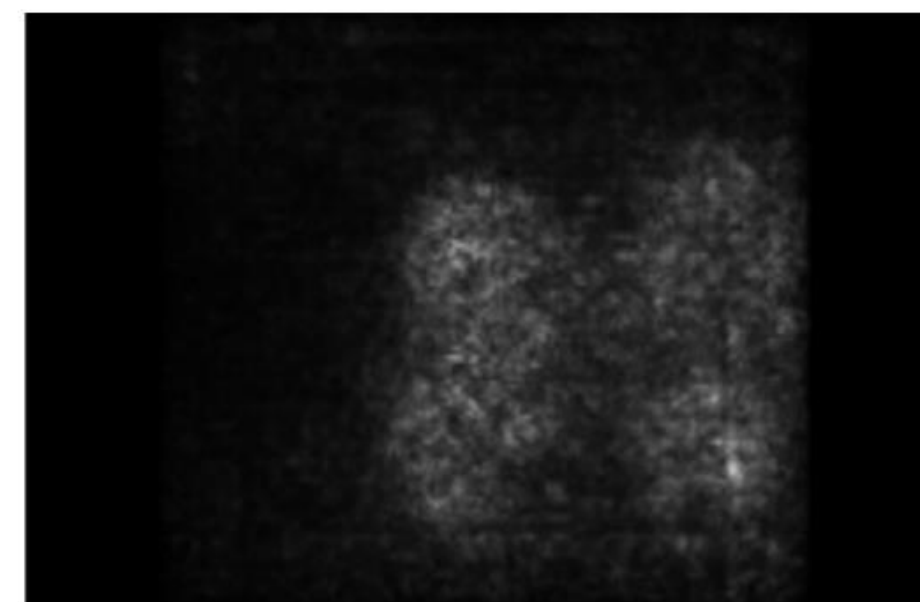
Which pixels matter: *Saliency via Backprop*

Forward pass: Compute probabilities



Dog

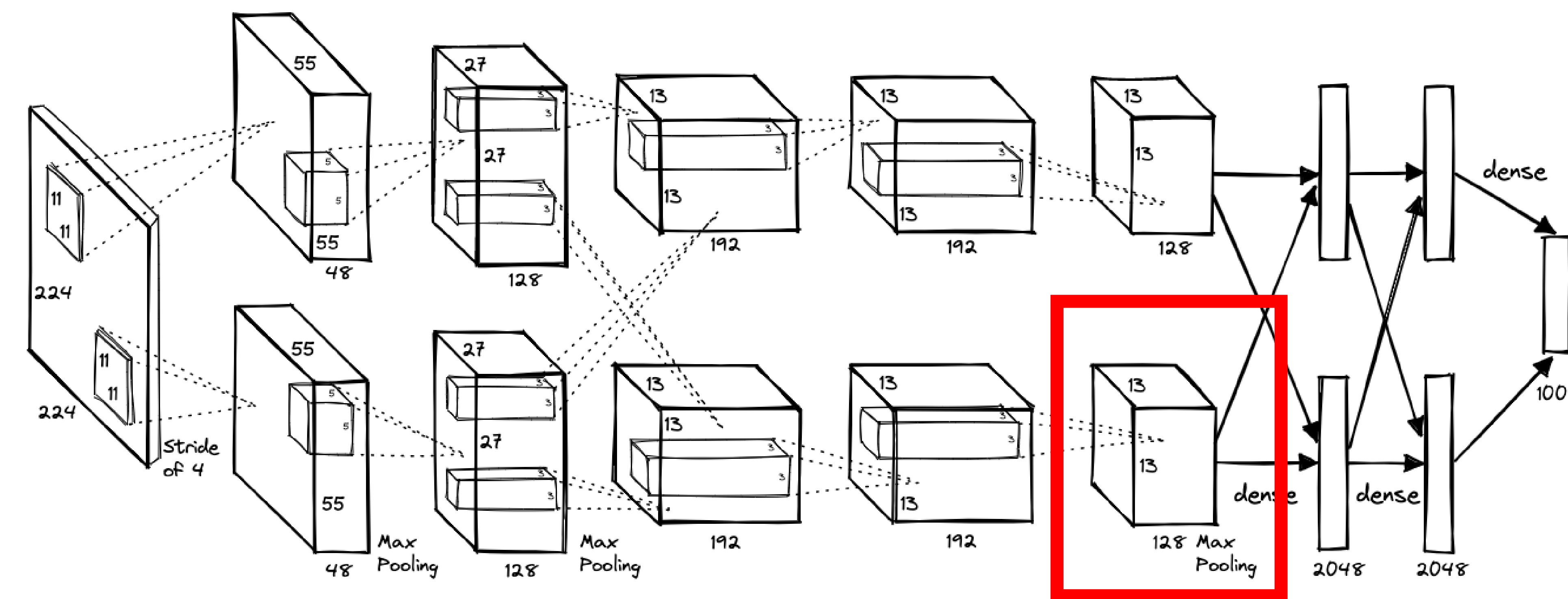
Compute gradient of **(unnormalized) class score** with respect to image pixels, take absolute value and max over RGB channels



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

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

Which pixels matter: *Saliency via Backprop*



Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

ReLU

b)

Forward pass

$$\begin{bmatrix} 1 & -1 & 5 \\ 2 & -5 & -7 \\ -3 & 2 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 5 \\ 2 & 0 & 0 \\ 0 & 2 & 4 \end{bmatrix}$$

Backward pass: backpropagation

$$\begin{bmatrix} -2 & 0 & -1 \\ 6 & 0 & 0 \\ 0 & -1 & 3 \end{bmatrix} \leftarrow \begin{bmatrix} -2 & 3 & -1 \\ 6 & -3 & 1 \\ 2 & -1 & 3 \end{bmatrix}$$

Backward pass: *guided backpropagation*

$$\begin{bmatrix} 0 & 0 & 0 \\ 6 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix} \leftarrow \begin{bmatrix} -2 & 3 & -1 \\ 6 & -3 & 1 \\ 2 & -1 & 3 \end{bmatrix}$$

Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 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

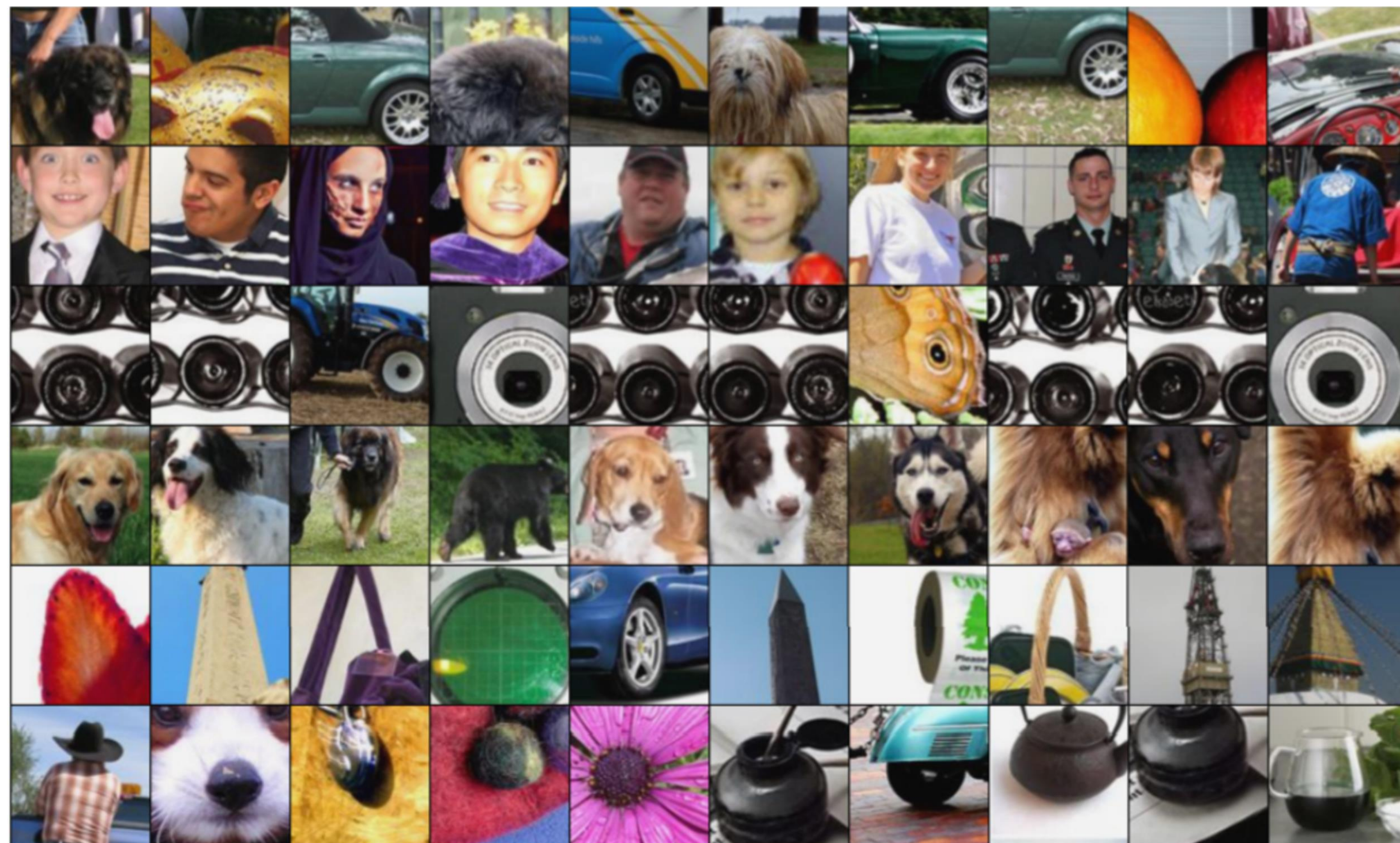
Intermediate features via (guided) backprop

Maximally activating patches
(Each row is a different neuron)

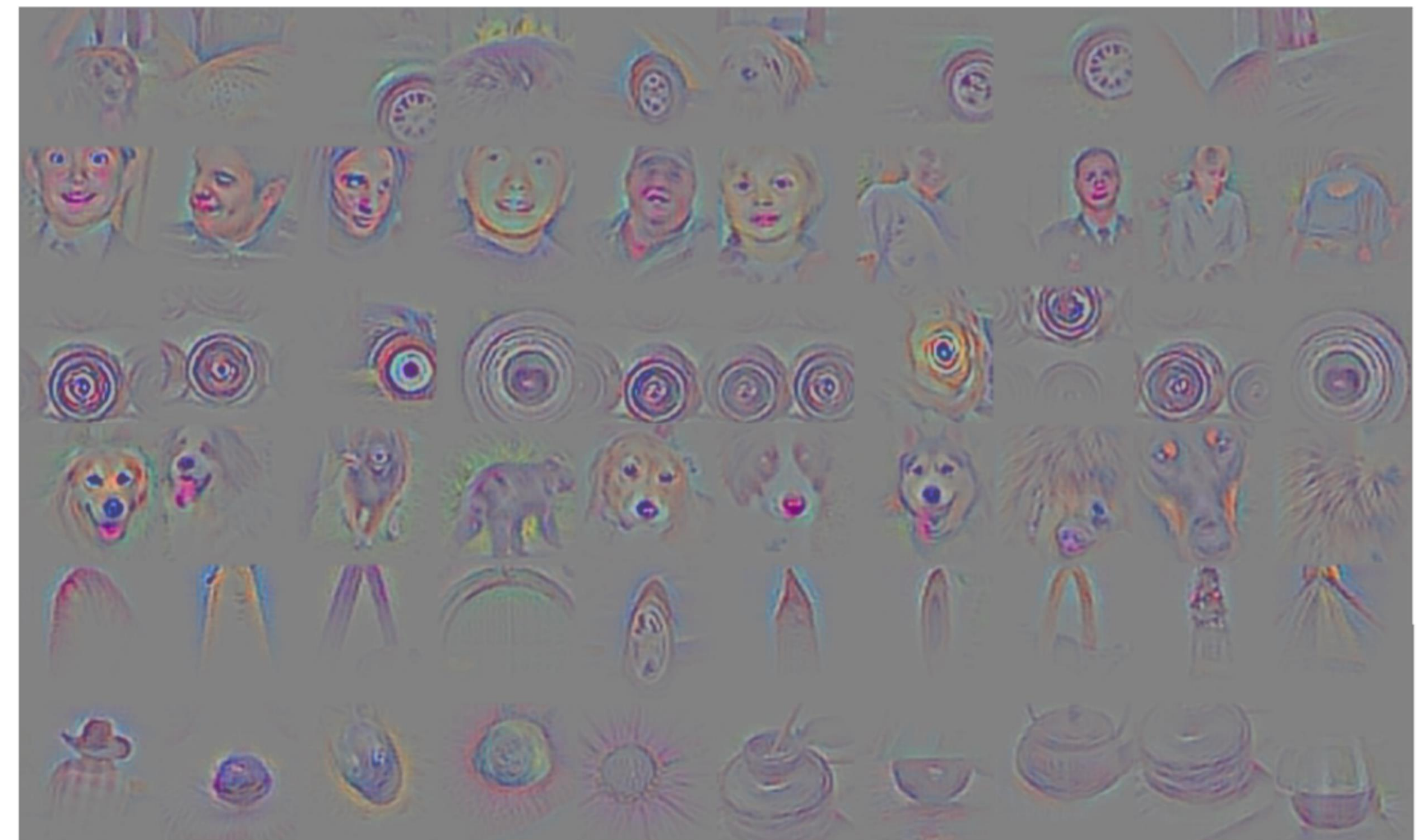
Guided Backprop

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

Intermediate features via (guided) backprop



Maximally activating patches
(Each row is a different neuron)



Guided Backprop

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

Gradient Ascent

Gradient ascent is used to generate images that maximize the activation of a specific neuron or set of neurons in the network. The basic idea behind gradient ascent is to compute the gradient of the function with respect to its input, and then update the input in the direction of the gradient. By repeating this process iteratively, we can gradually increase the output of the function and find an input that maximizes it.

It is used to generate images that maximize the activation of a specific neuron or set of neurons in the network. To do this, we first select the neuron or set of neurons that we want to maximize the activation of. We then start with a random image as the input and compute the gradient of the activation of the chosen neuron(s) with respect to the input image. We then update the input image in the direction of the gradient, by adding a small fraction of the gradient to the image. This produces a new image that has a slightly higher activation for the chosen neuron(s). We repeat this process for a certain number of iterations or until we reach a threshold for the activation of the neuron(s).

Thus, we can generate images that maximize the activation of a specific neuron or set of neurons in a neural network. This can be useful for visualizing the features and patterns that the neuron(s) are sensitive to, as well as for generating adversarial examples that are designed to fool 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

Gradient Ascent

(Guided) backprop:
Find the part of an image that a neuron responds to

Gradient ascent:
Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I \; f(I) + R(I)$$

Neuron value — Natural image regularizer

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

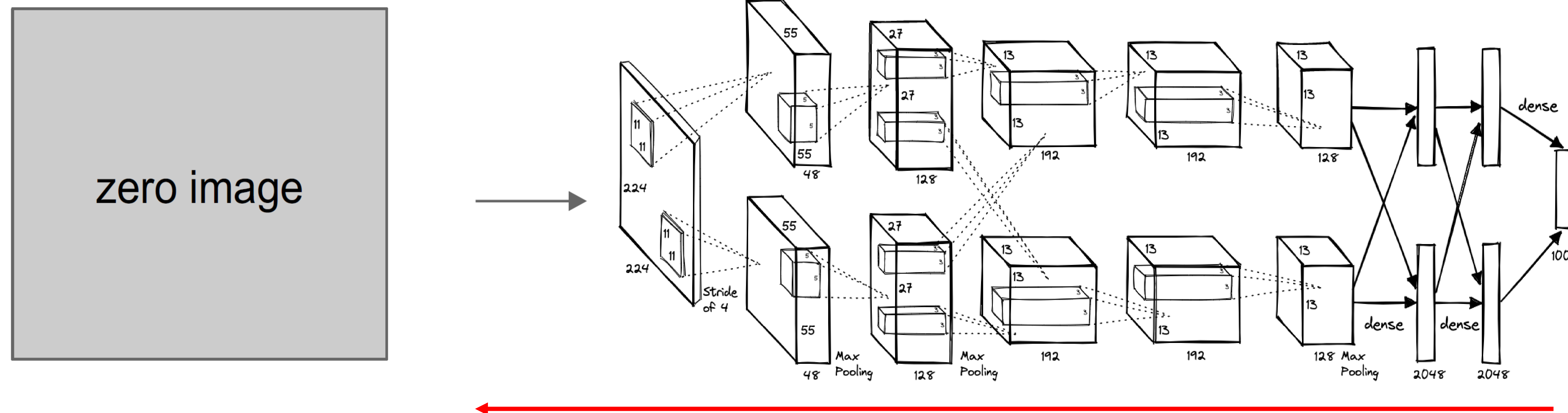
Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

score for class c (before Softmax)

1. Initialize image to zeros

zero image



Repeat:

2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

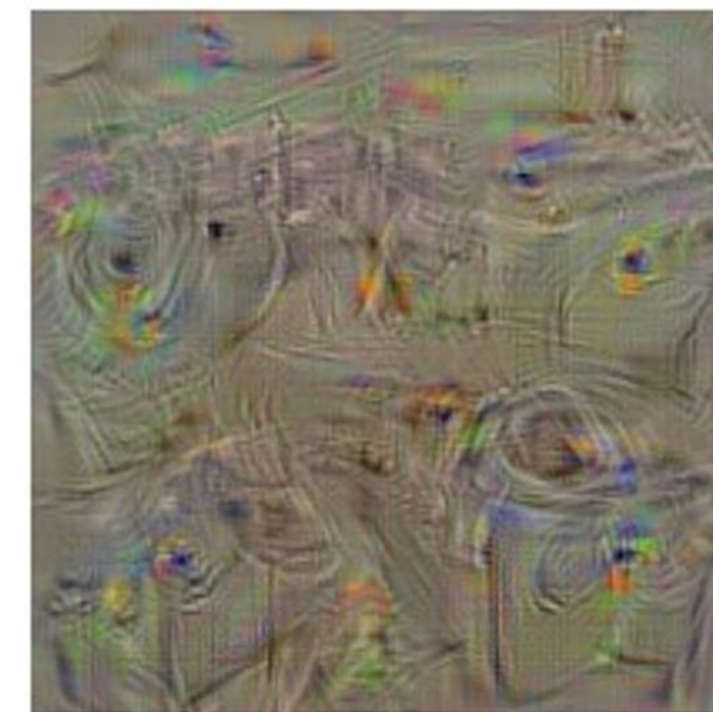
Co-financed by the European Union
Connecting Europe Facility

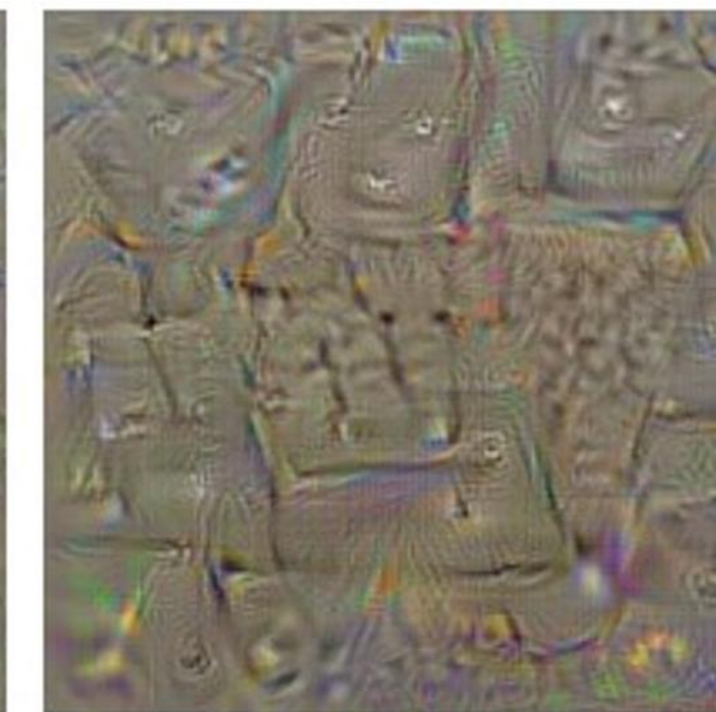
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

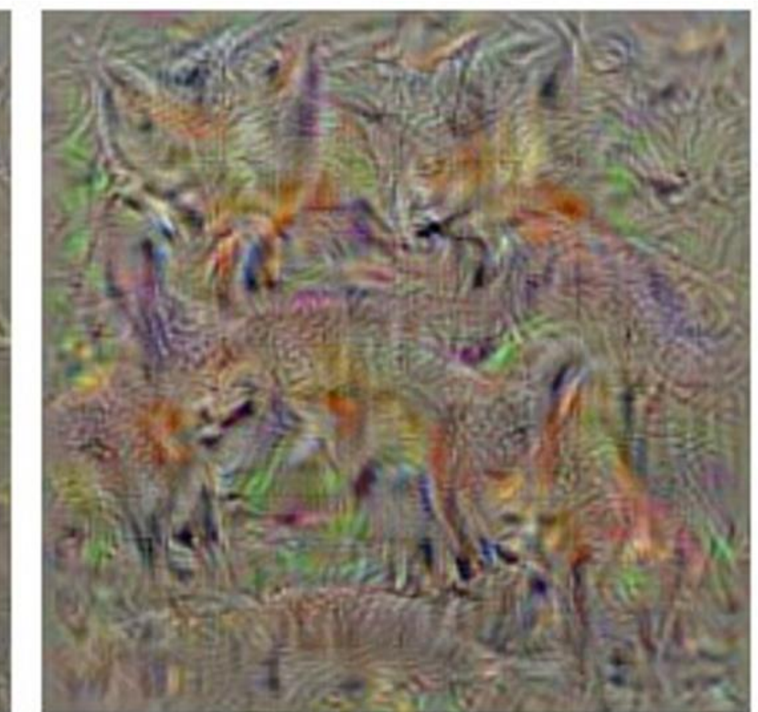
Gradient Ascent

$$\arg\max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

Simple regularizer: Penalize L2 norm of generated image

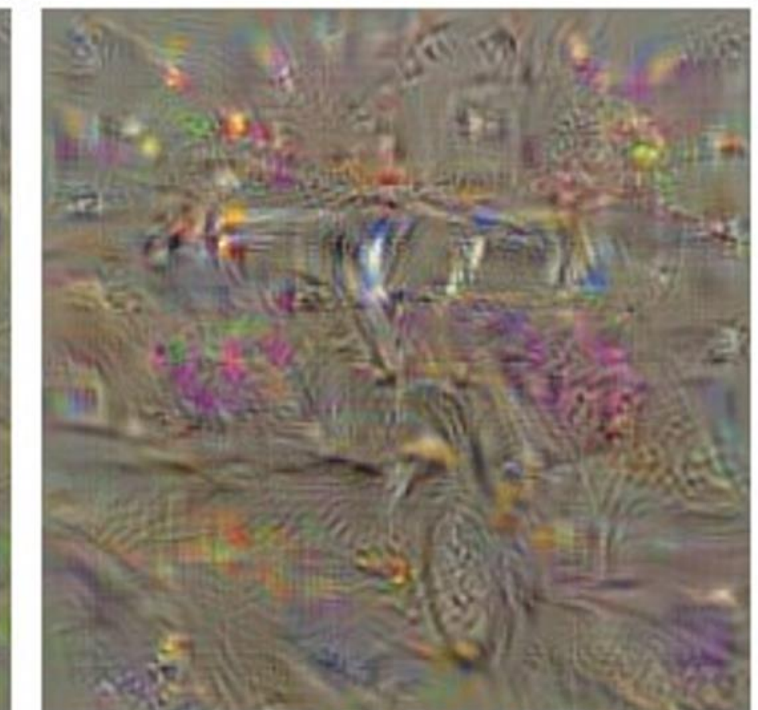

washing machine


computer keyboard


kit fox

goose

ostrich


limousine

Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014.

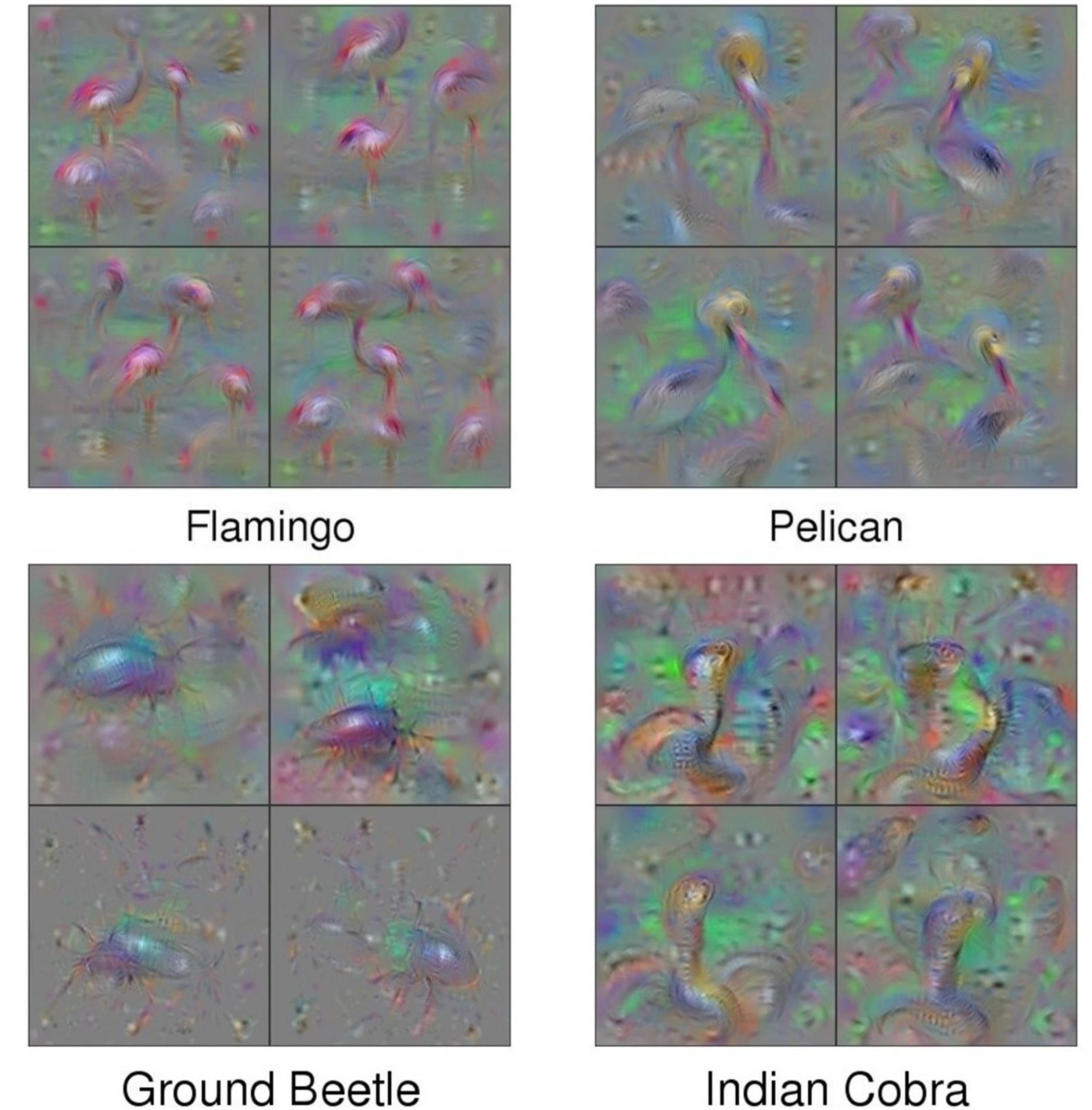
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Gradient Ascent

$$\arg\max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

(1) Gaussian blur image
(2) Clip pixels with small values to 0
(3) Clip pixels with small gradients to 0

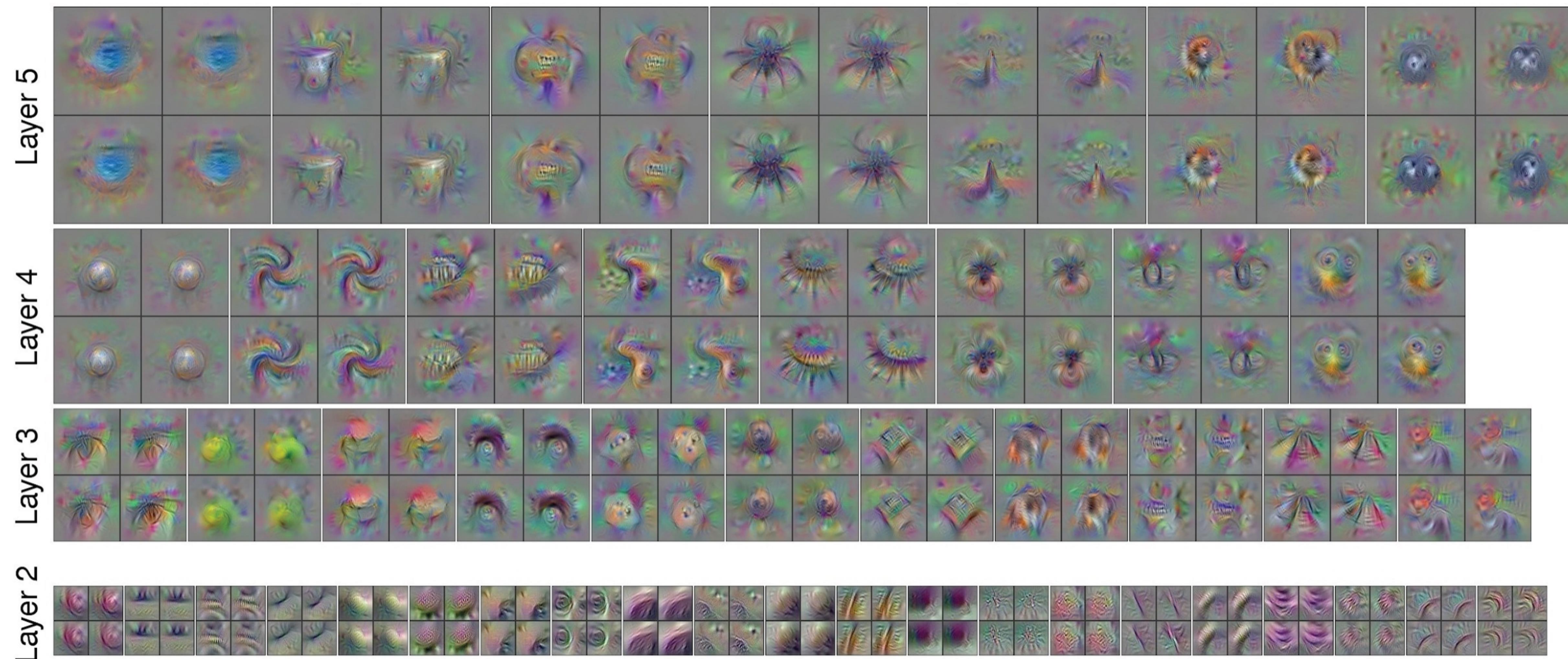


Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

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

Gradient Ascent



Use the same approach to visualize intermediate features

Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

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

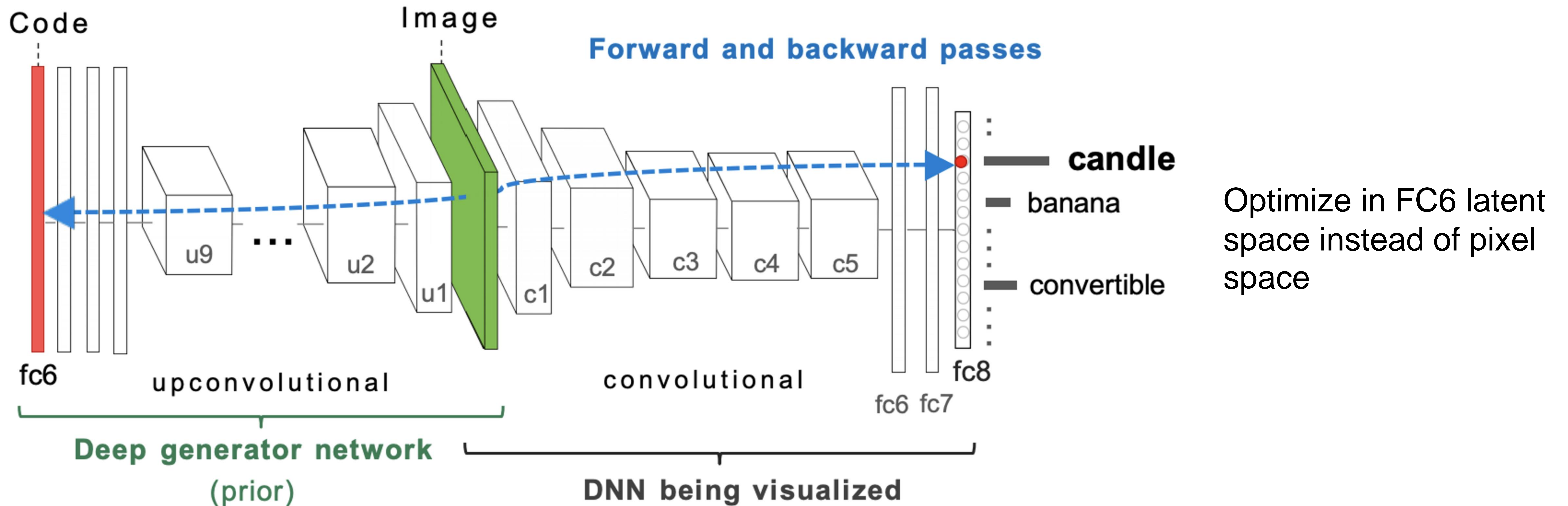
Gradient Ascent

Nguyen et al, "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks", ICML Visualization for Deep Learning Workshop 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

Gradient Ascent



Optimize in FC6 latent space instead of pixel space

Nguyen et al, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

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

Gradient Ascent

Optimize in FC6 latent space instead of pixel space

Nguyen et al, “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks,” NIPS 2016

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

Adversarial perturbations

Adversarial perturbations are small, often imperceptible, changes made to an input (such as an image or audio signal) that are designed to cause a neural network to misclassify it. These perturbations can be added to an input in a targeted or untargeted way.

In a targeted attack, the adversarial perturbation is designed to cause the network to misclassify the input as a specific target class. For example, an image of a panda might be perturbed to cause a neural network to classify it as a gibbon. In an untargeted attack, the goal is simply to cause the network to misclassify the input, without specifying a particular target class.

Adversarial perturbations are generated using optimization techniques, such as gradient ascent, to find the smallest perturbation that causes the network to misclassify the input. The perturbation is typically constrained to be small and imperceptible, to ensure that it does not significantly alter the input

Moosavi-Dezfooli, Seyed-Mohsen, et al. "Universal adversarial perturbations." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

Co-financed by the European Union
Connecting Europe Facility

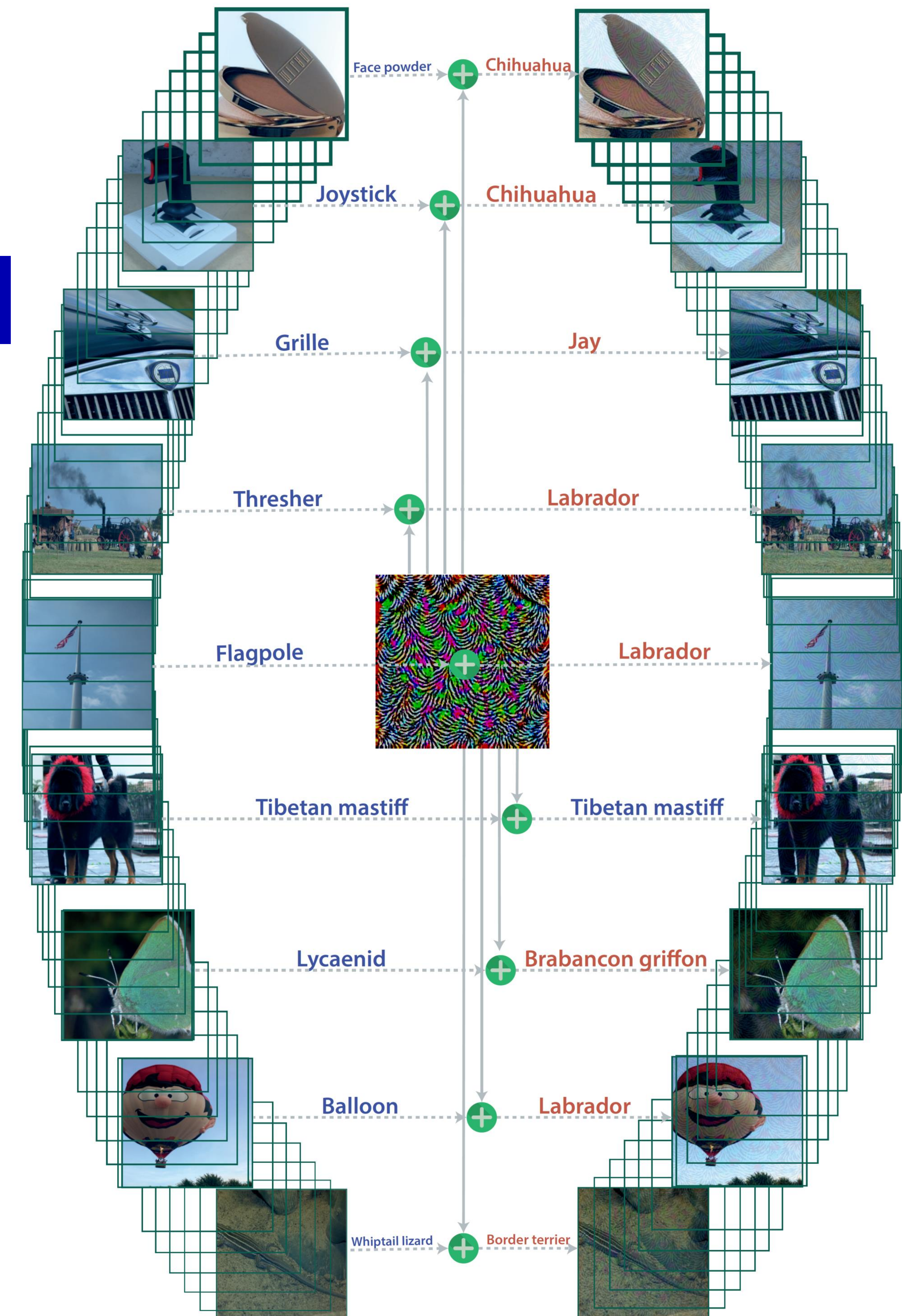
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Adversarial perturbations

Can we find a single small image perturbation that fools a state-of-the-art deep neural network classifier on all natural images?

→ lead to misclassify natural images with high probability.

By adding a quasi-imperceptible perturbation to natural images, the label estimated by the deep neural network is changed with high probability. Such perturbations are dubbed universal, as they are image agnostic. The existence of these perturbations is problematic when the classifier is deployed in real-world (and possibly hostile) environments, as they can be exploited by adversaries to break the classifier.



Moosavi-Dezfooli, Seyed-Mohsen, et al. "Universal adversarial perturbations." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

Feature Inversion

Feature inversion is a technique used to reconstruct an image that maximally activates a particular set of neurons in a neural network. The idea behind feature inversion is to find an image that has similar features to the set of neurons being targeted, in order to understand what the network is looking for in the input data.

The feature inversion process starts by selecting a set of neurons in the network that we want to maximize the activation of. We then initialize a random image and feed it forward through the network, recording the activations of the selected neurons. We then compute the gradient of the activations with respect to the input image, and adjust the image to increase the activation of the selected neurons. This process is repeated iteratively, gradually improving the image until it maximally activates the selected neurons.

Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 2015

Feature Inversion

Feature inversion can be used to visualize what the network is looking for in the input data. By generating an image that maximally activates a set of neurons, we can gain insight into the features and patterns that are important for the network's predictions. Feature inversion can also be used for generating visualizations of the network's internal representations, which can be useful for interpretability and debugging.

One limitation of feature inversion is that it may not always result in realistic or meaningful images, as it can be difficult to generate an image that simultaneously maximizes the activations of a set of neurons and looks realistic. However, with appropriate regularization techniques and additional constraints, such as style transfer or image reconstruction, feature inversion can be used to generate more realistic and interpretable images.

Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 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

Feature Inversion

Given a CNN feature vector for an image, find a new image that:

- Matches the given feature vector
- “looks natural” (image prior regularization)

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^{H\times W\times C}}{\operatorname{argmin}} \ \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda\mathcal{R}(\mathbf{x})$$

Given feature vector (Φ_0)

Features of new image ($\Phi(\mathbf{x})$)

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j}\left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2\right)^{\frac{\beta}{2}}$$

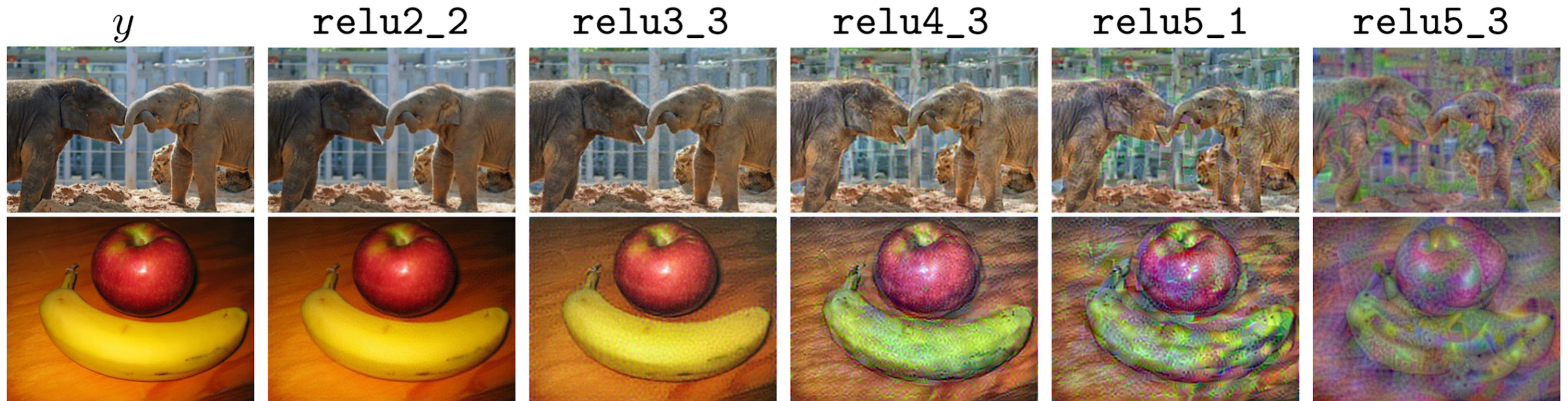
Total Variation regularizer (encourages spatial smoothness)

Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 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

Feature Inversion



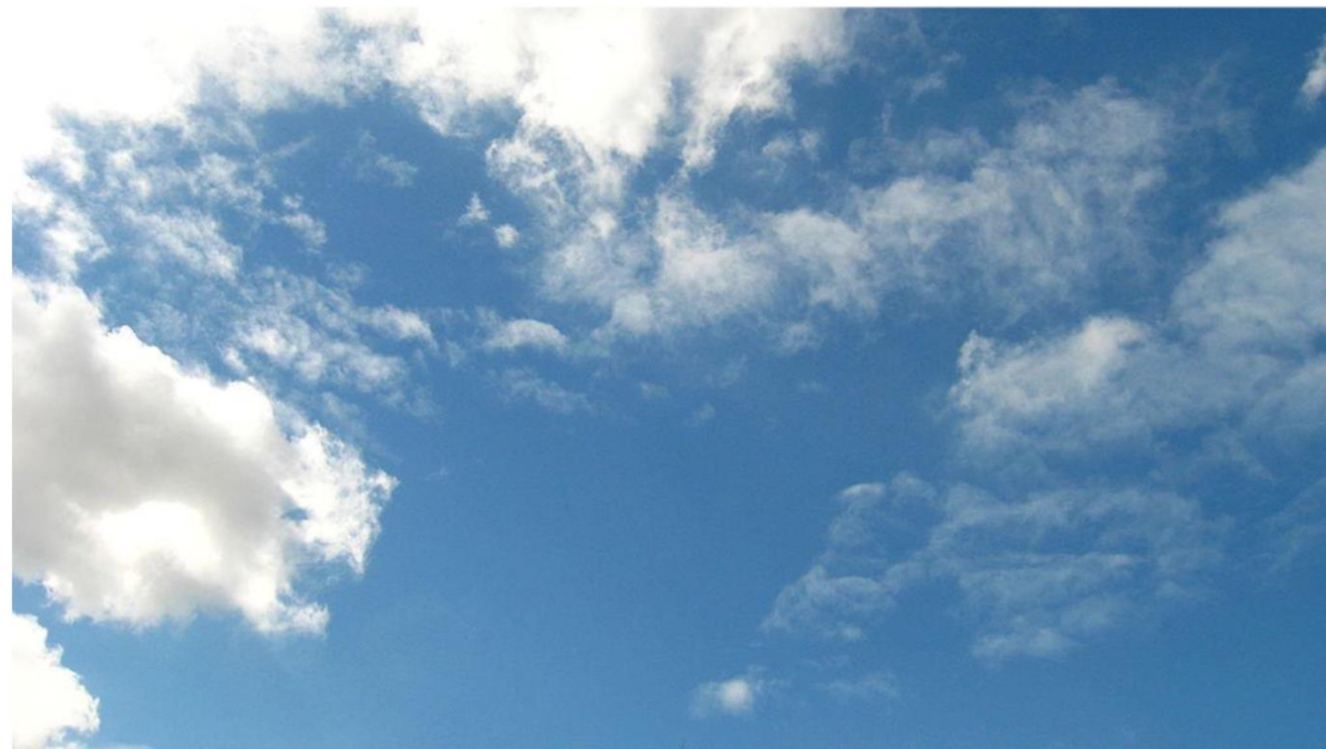
Reconstructing from different layers of VGG-16

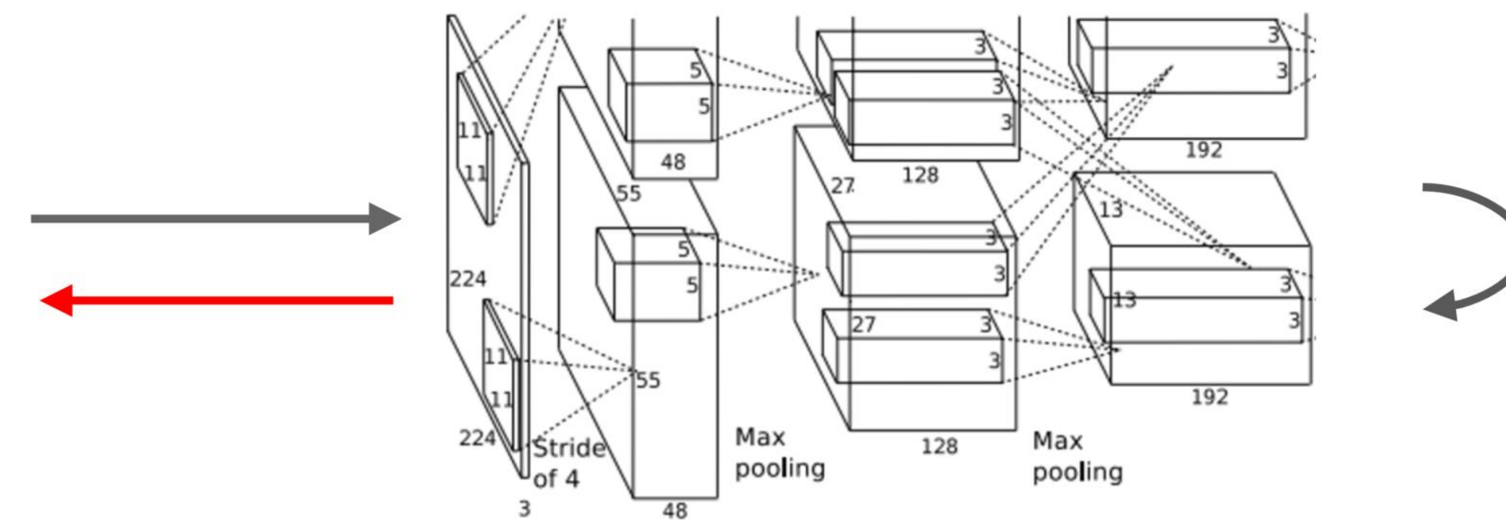
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

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

Feature Inversion: *DeepDream - Amplify existing features*

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network.





Choose an image and a layer in a CNN; repeat:
1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer equal to its activatic
3. Backward: Compute gradient on image
4. Update image

Equivalent to:
$I^* = \arg\max_I \sum_i f_i(I)^2$

Mordvintsev, Olah, and Tyka, “Inceptionism: Going Deeper into 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

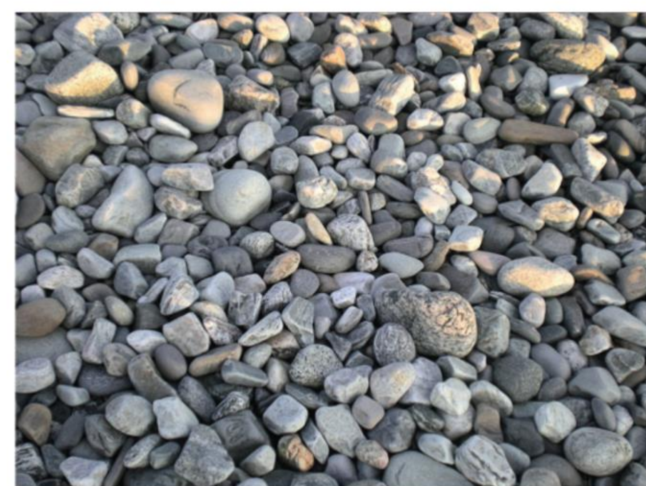
Neural Texture Synthesis: *Gram Matrix*

Neural texture synthesis is a technique that uses deep neural networks to generate new textures that have similar visual characteristics to a given input texture. A pre-trained convolutional neural network is used to analyze the features of the input texture. The network is typically a variant of VGG (Visual Geometry Group) network, which is trained on large-scale image recognition tasks. The network is used to extract the feature maps from a set of convolutional layers, which can be thought of as representations of the input texture at different levels of abstraction.

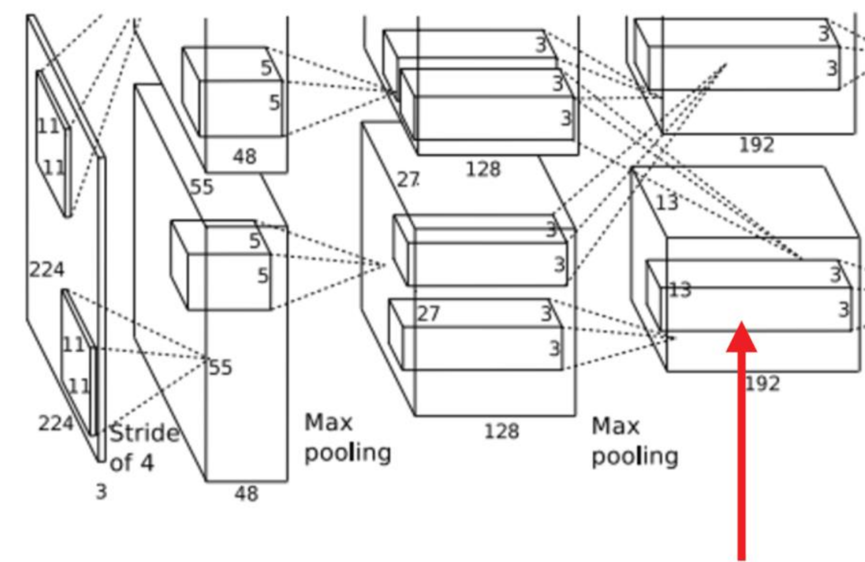
The **Gram matrix** is then computed for each set of feature maps. The Gram matrix is a matrix of dot products between the feature vectors of the original texture. This matrix captures the correlation between the different features in the texture and is used as a measure of texture style. To generate a new texture with a similar style to the input texture, the Gram matrices of the input texture and a randomly initialized noise image are computed. The algorithm then optimizes the noise image to minimize the difference between its Gram matrix and the Gram matrix of the input texture, while also preserving certain statistical properties of the noise image. This optimization process generates a new image that has similar texture characteristics as the input texture.

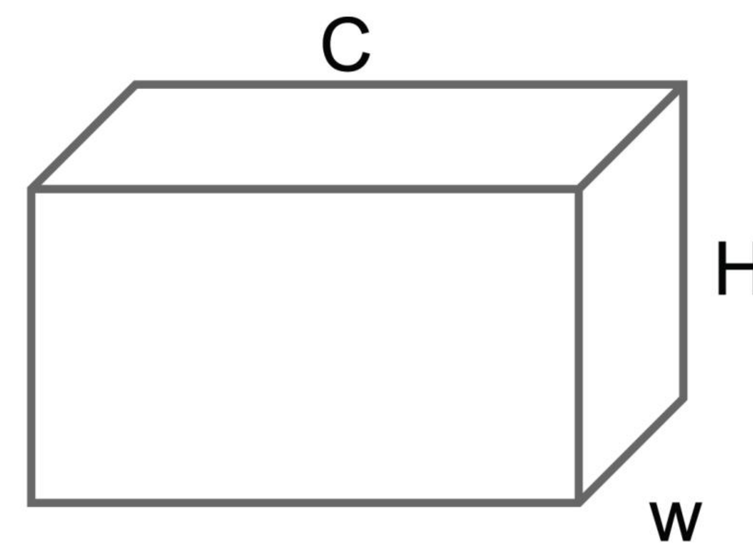
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Neural Texture Synthesis: *Gram Matrix*



This image is in the public domain.



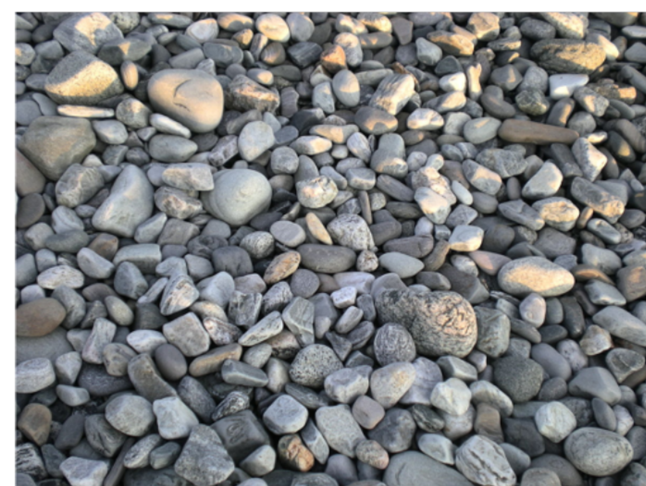


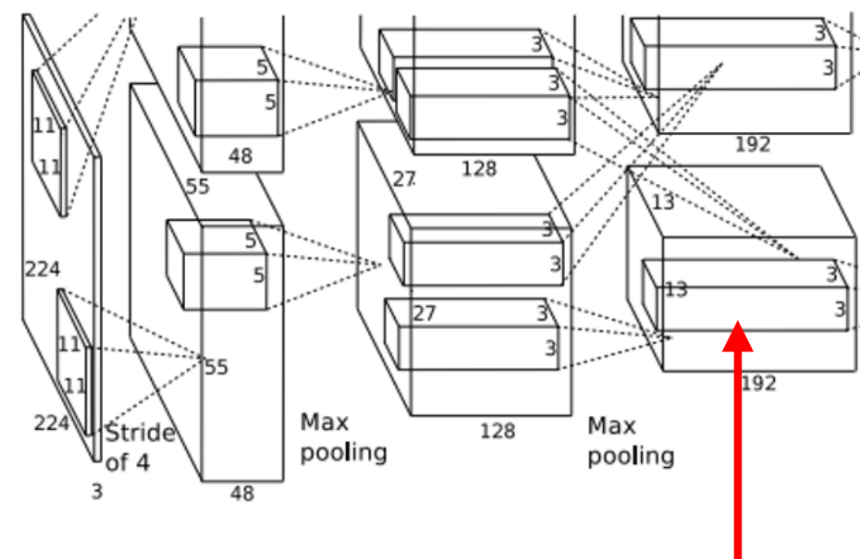
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

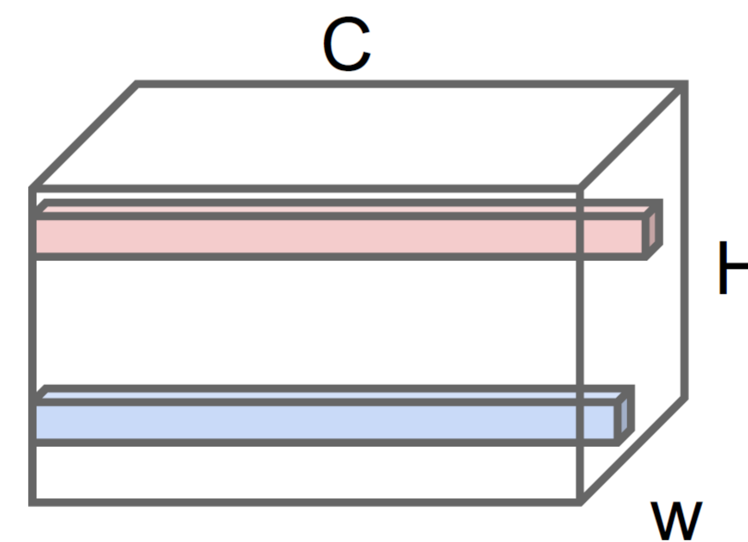
Co-financed by the European Union
Connecting Europe Facility

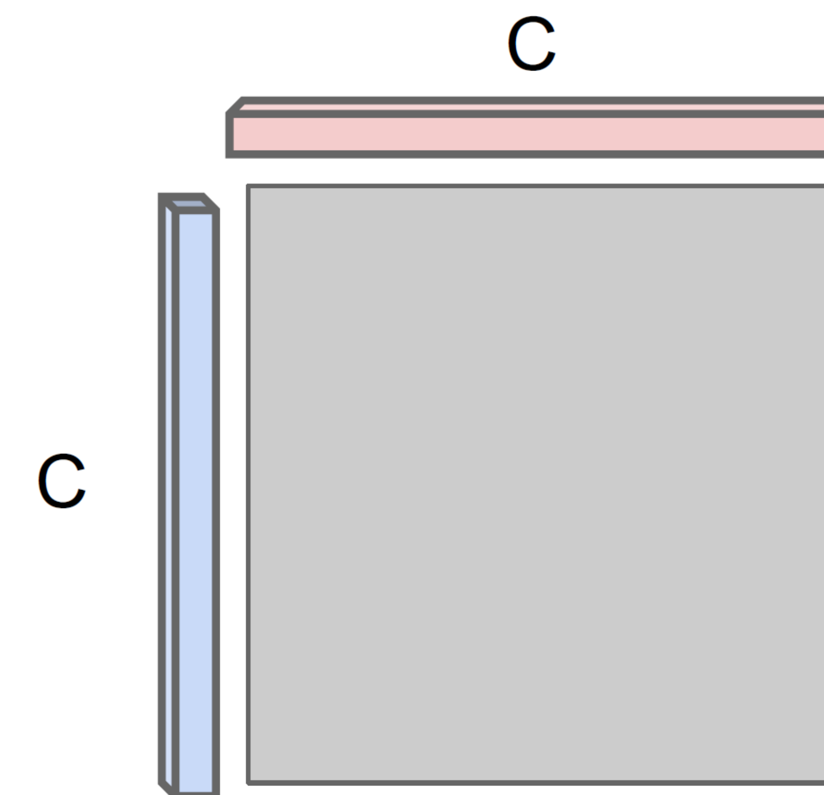
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Neural Texture Synthesis: *Gram Matrix*


This image is in the public domain.



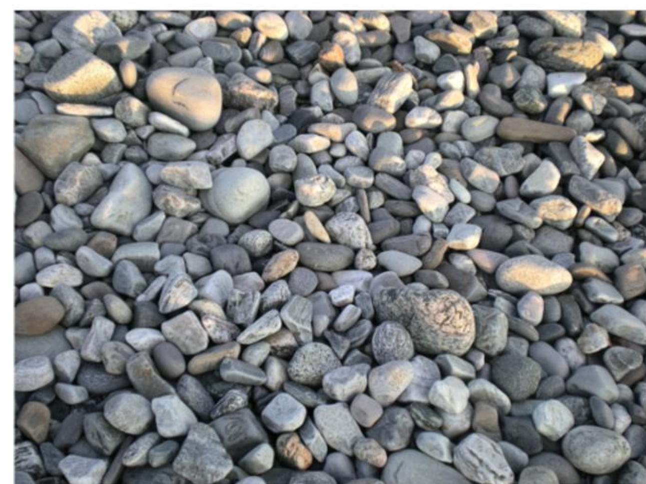


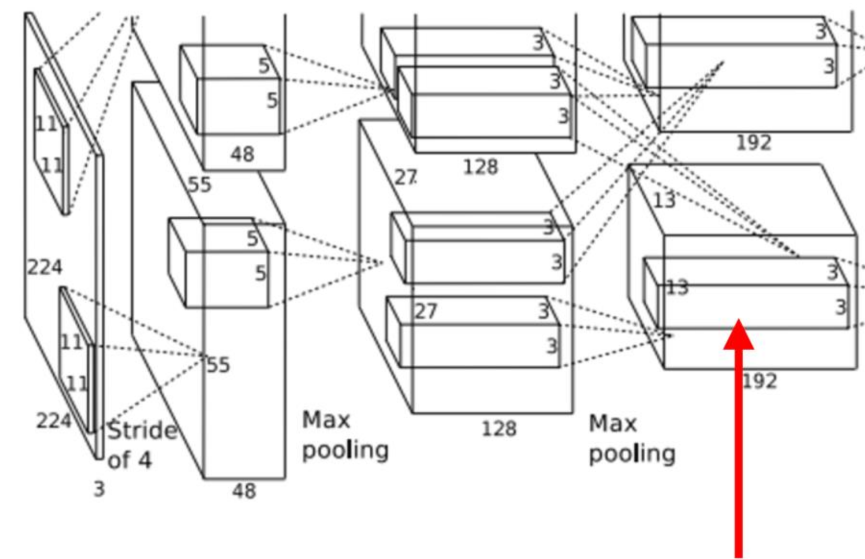


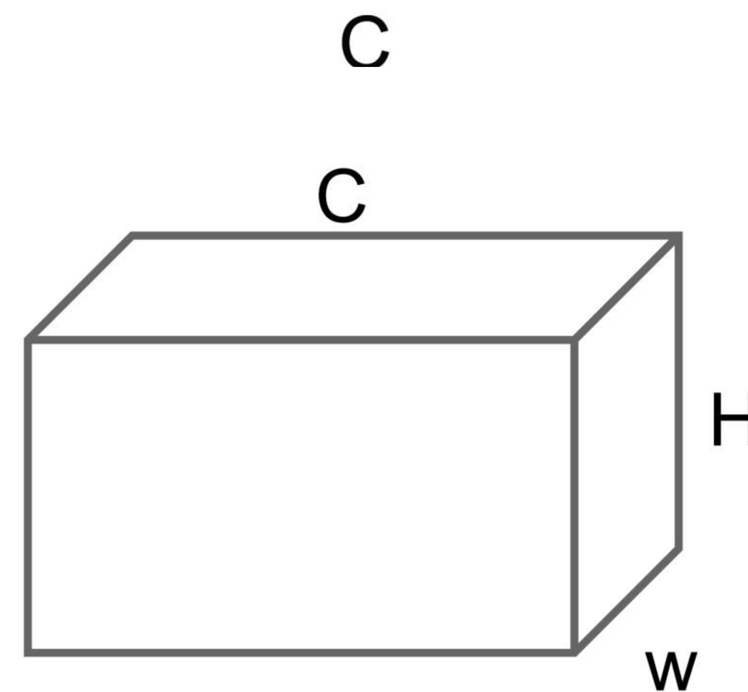
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

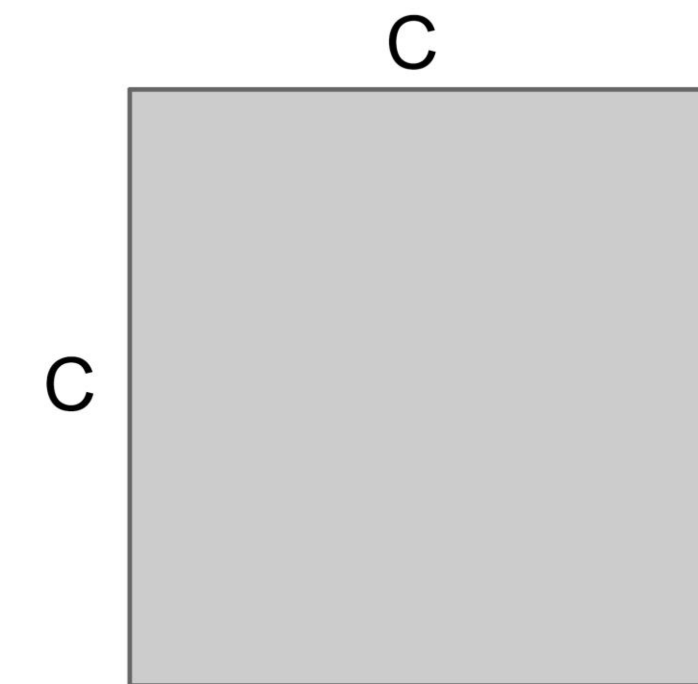
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Neural Texture Synthesis: *Gram Matrix*


This image is in the public domain.







Gram Matrix

Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Average over all pairs of vectors, giving **Gram matrix** of shape C x C

Efficient to compute; reshape features from

C x H x W to =C x HW

then compute $G = FF^T$

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

Neural Texture Synthesis

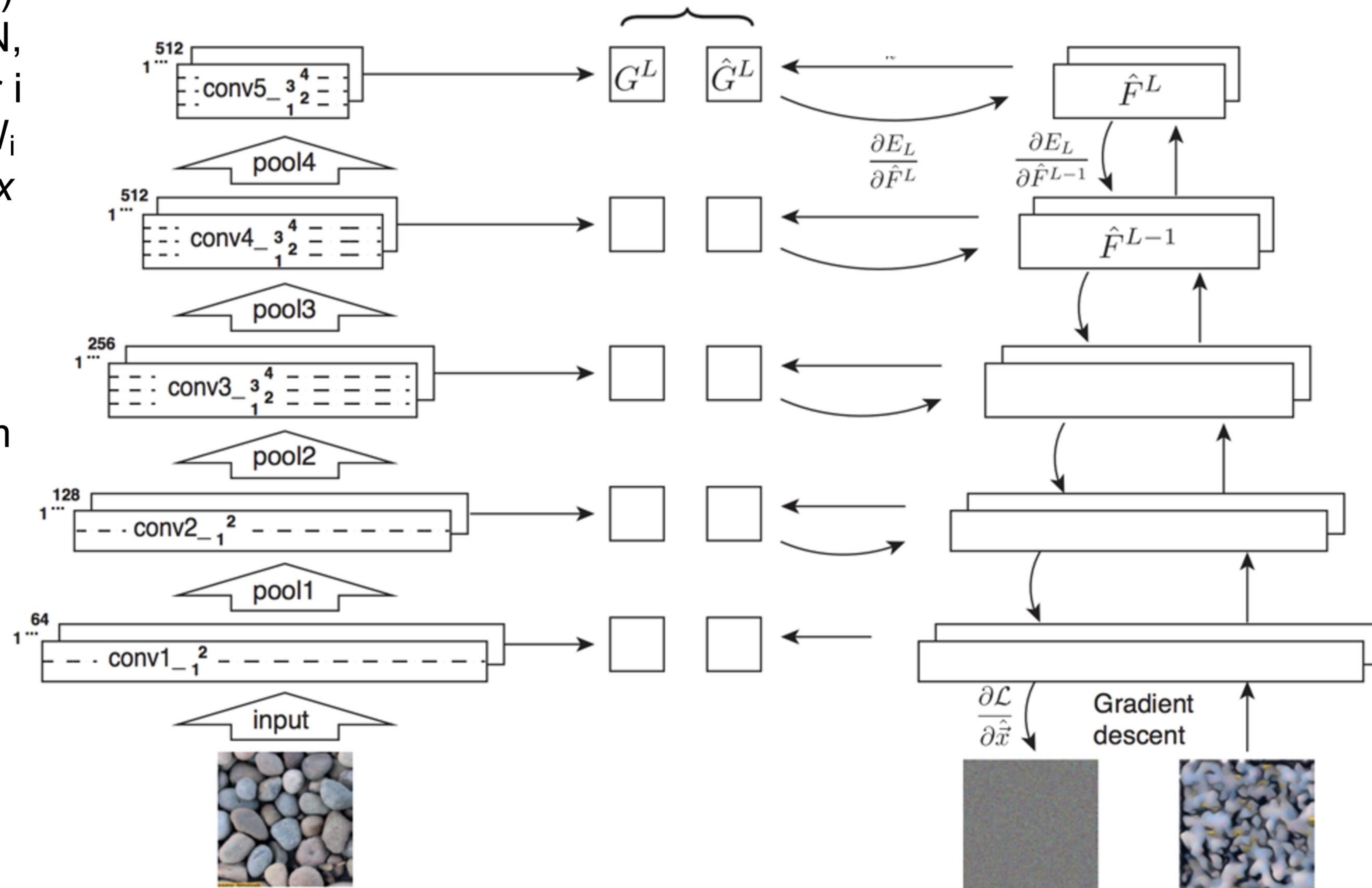
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l \right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i)$$

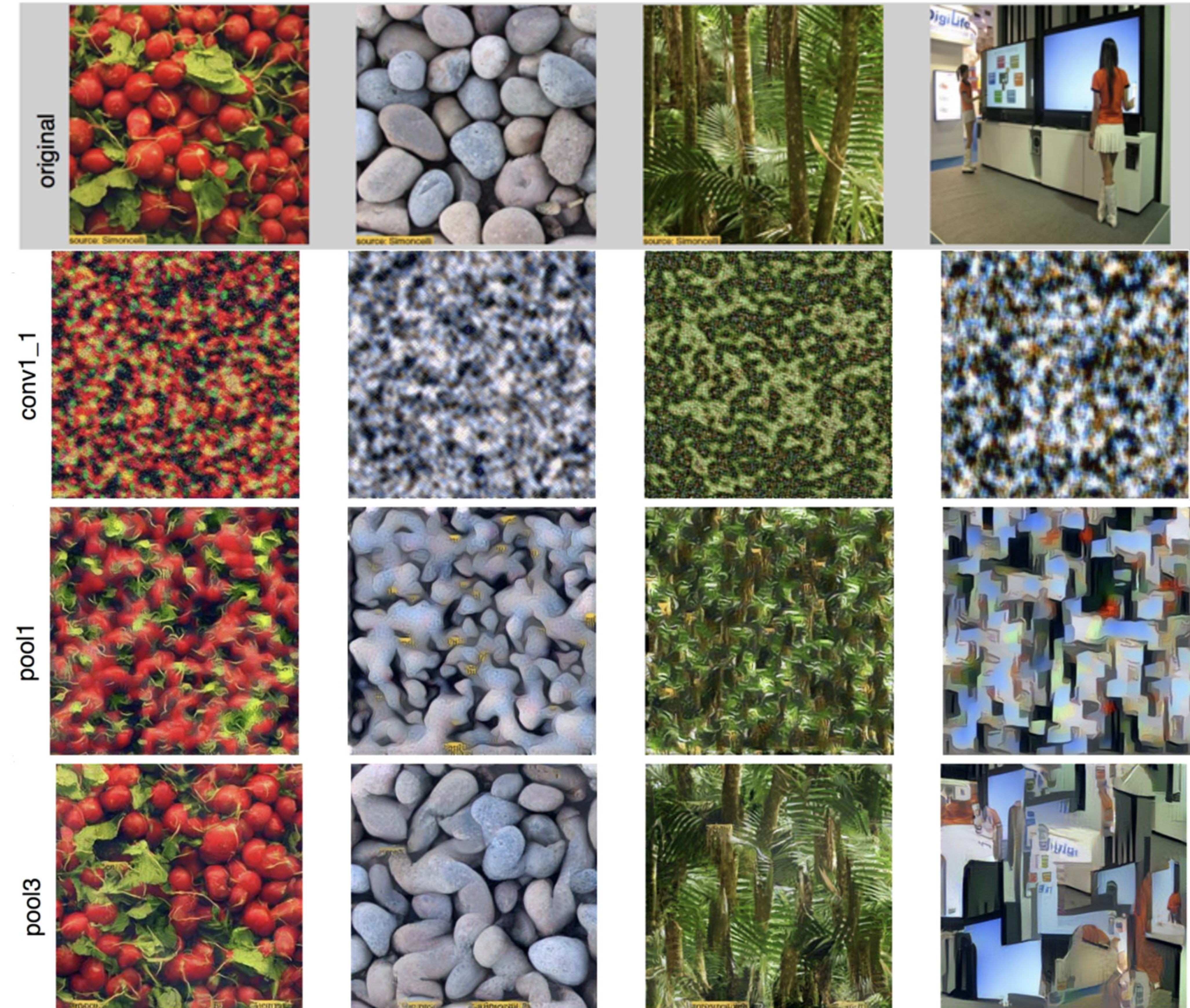
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Neural Texture Synthesis

Reconstructing texture from higher layers recovers larger features from the input texture



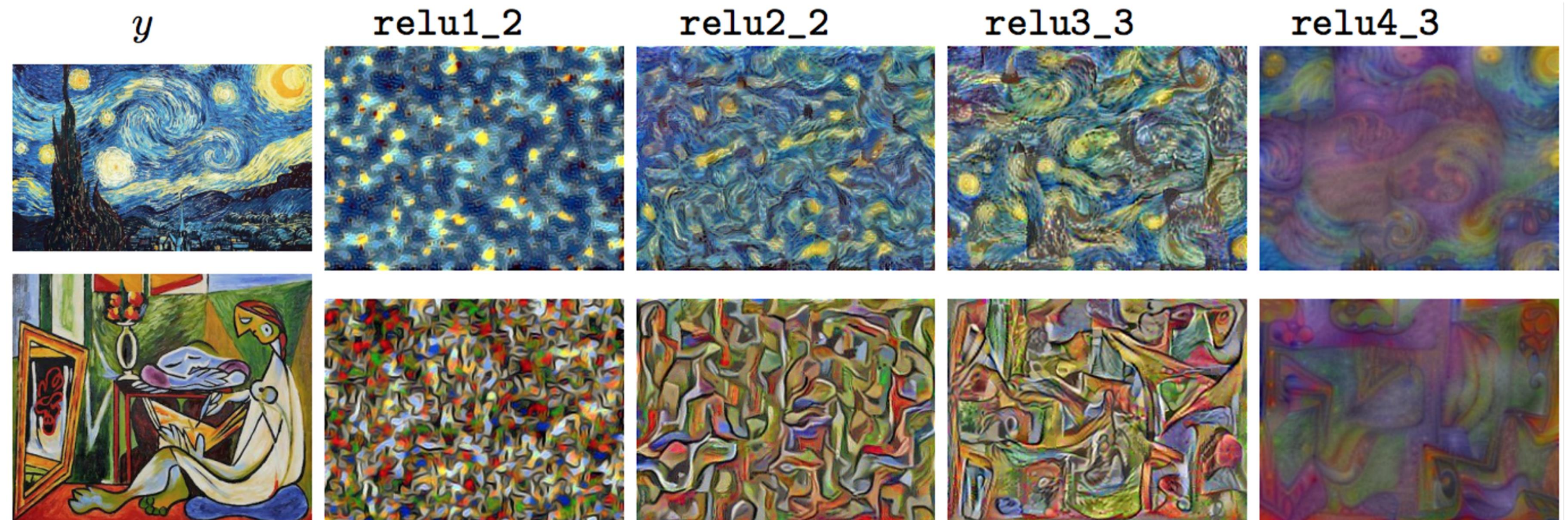
Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Co-financed by the European Union
Connecting Europe Facility

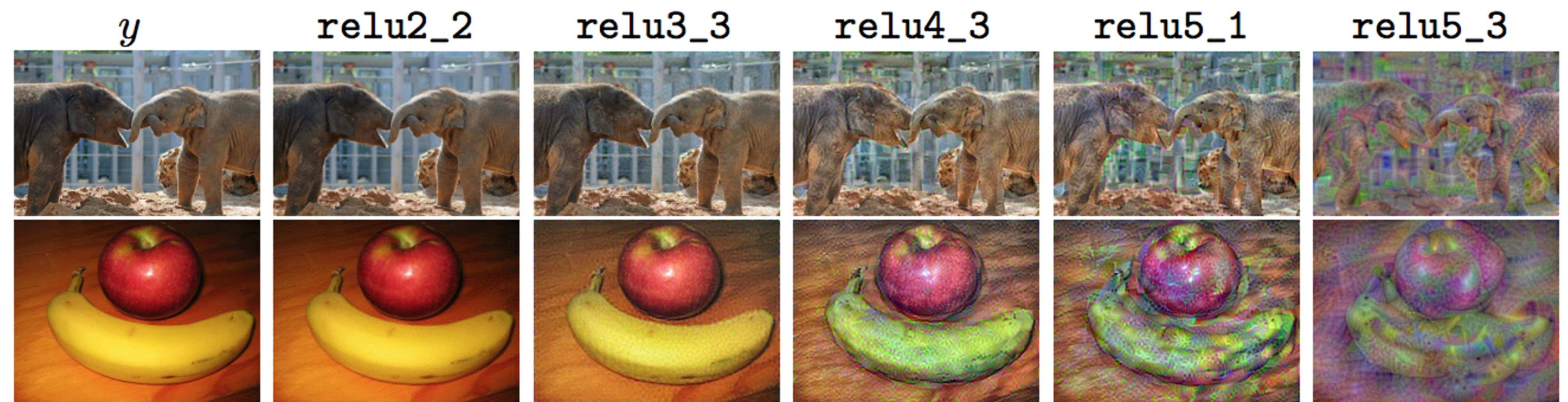
This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Neural Texture Synthesis: *Texture = Artwork*

Texture synthesis
(Gram reconstruction)



Feature
reconstruction



Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 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

Neural Texture Synthesis: *Style transfer*

Content Image

+

Style Image

=

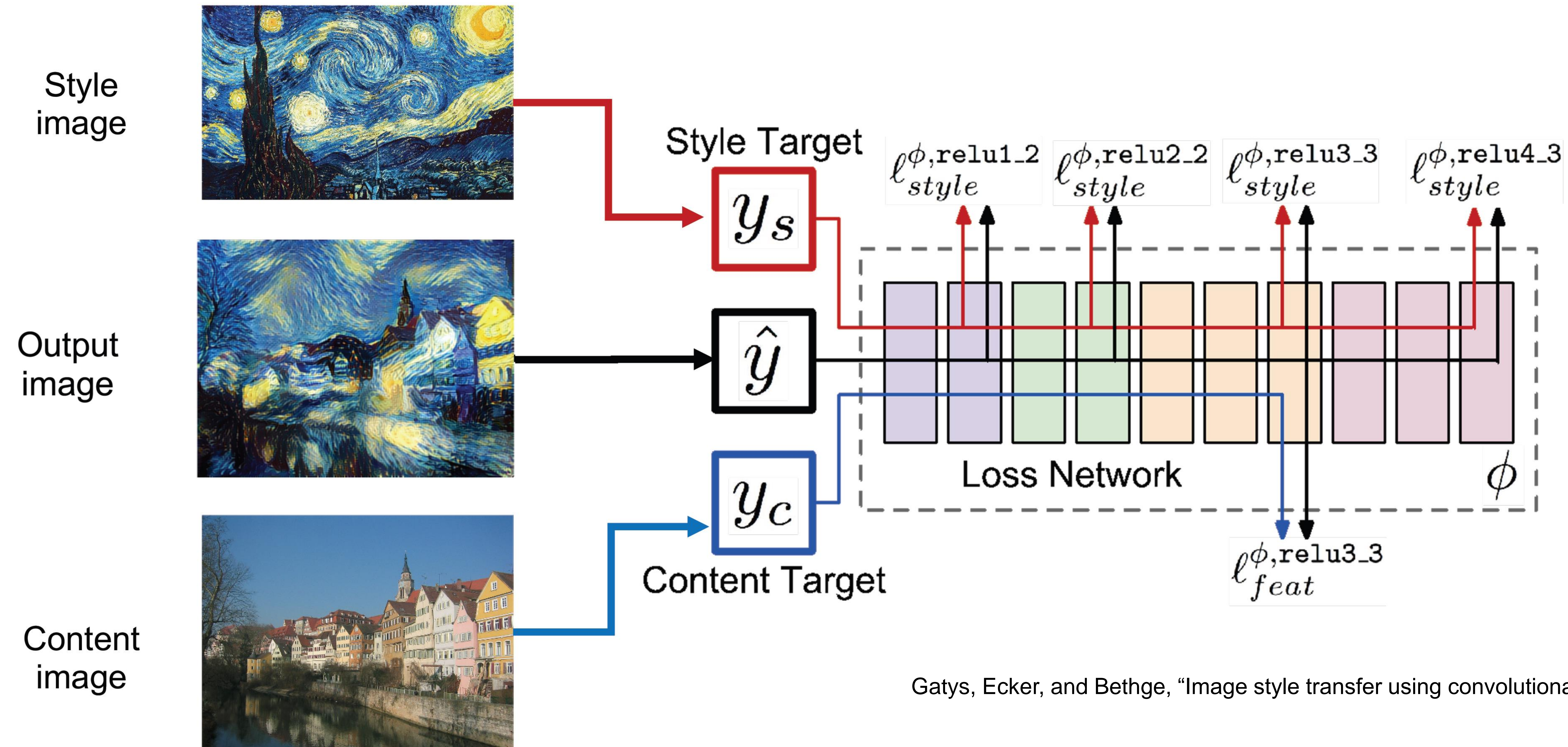
Style Transfer!

Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, 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

Neural Texture Synthesis: *Style transfer*

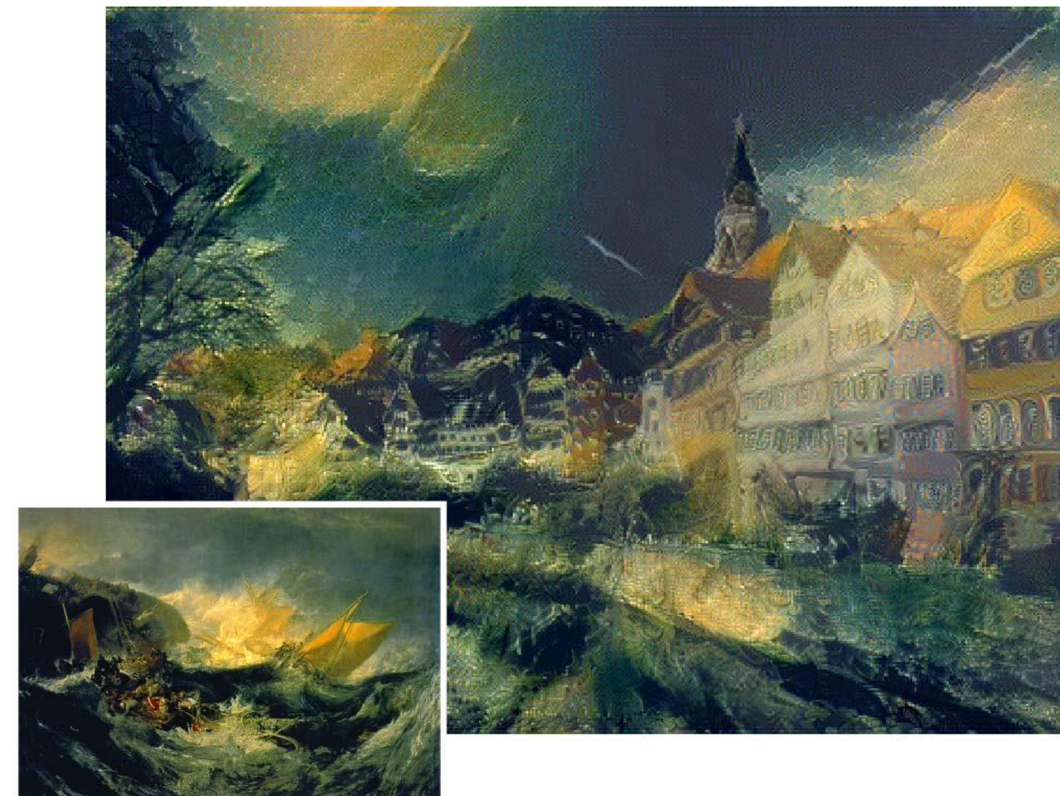
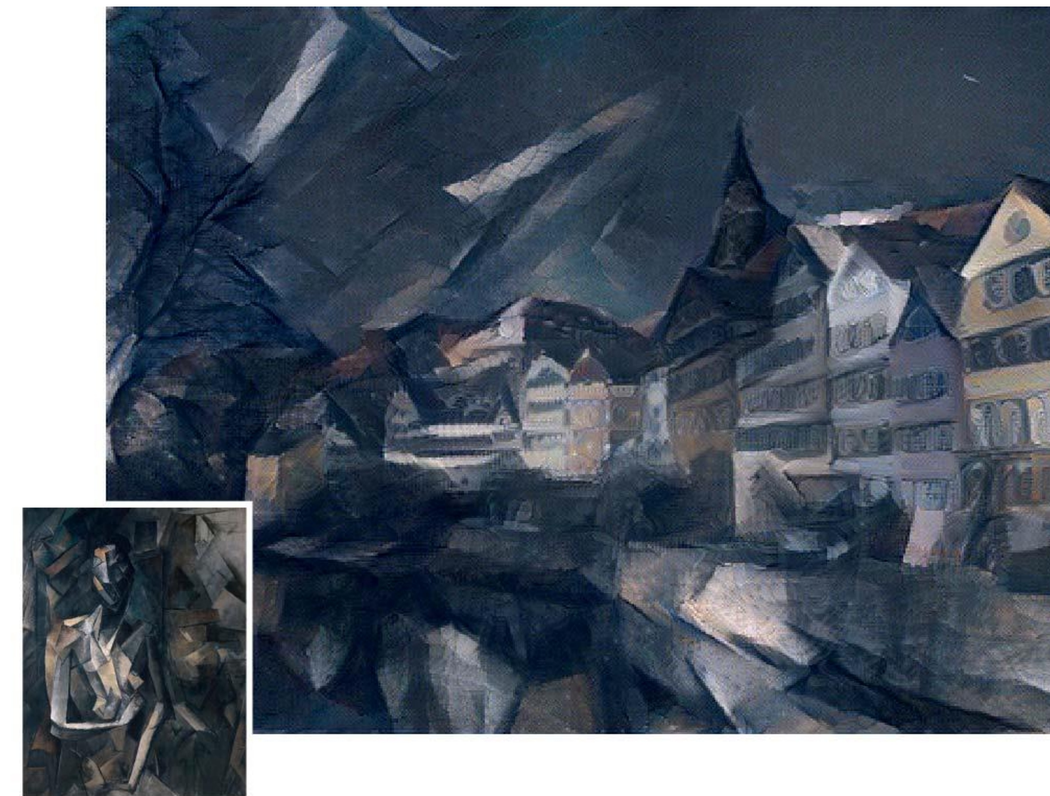


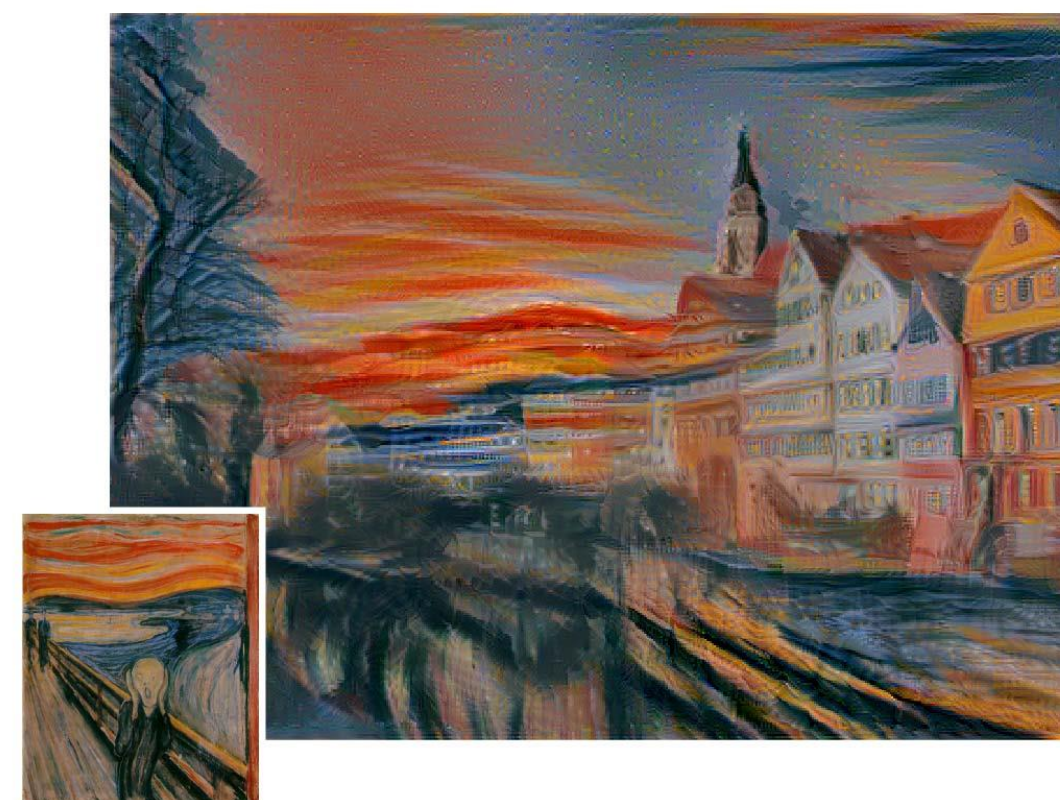
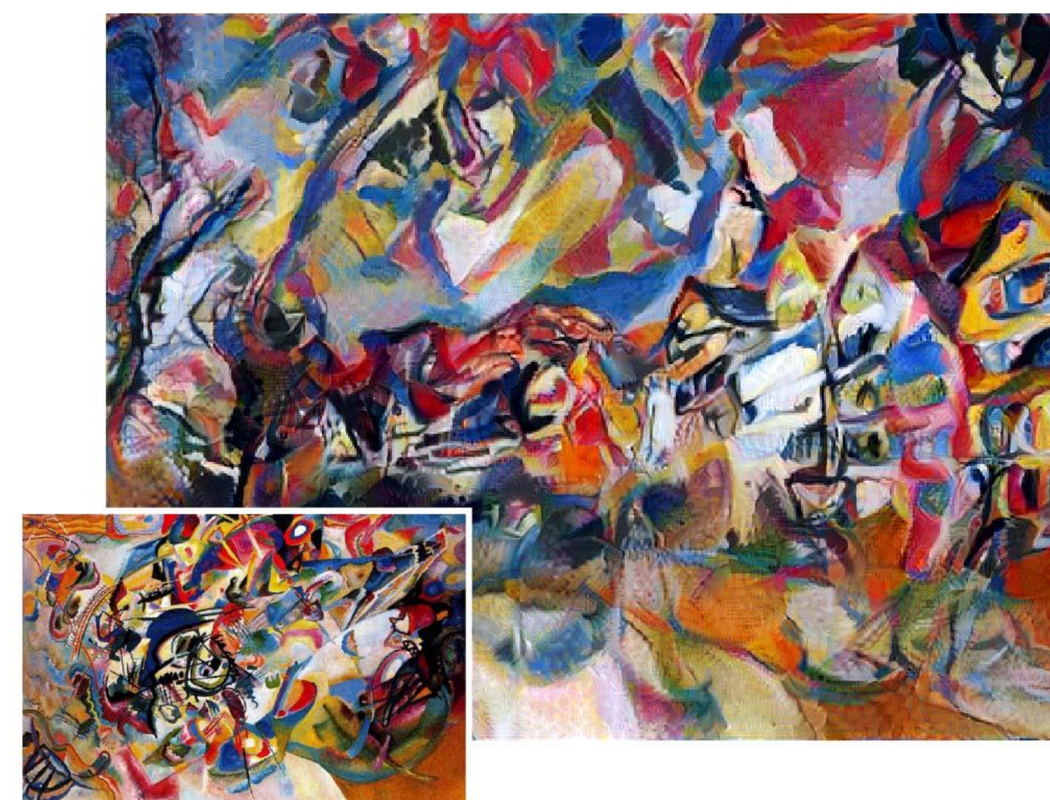
Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016

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

Neural Texture Synthesis: *Style transfer*

Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, 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

Neural Texture Synthesis: *Style transfer*

More weight to content loss

More weight to style loss

Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, 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

Neural Texture Synthesis: *Style transfer*

Problem: Style transfer requires many forward / backward passes through VGG; very slow!

Solution: Train another neural network to perform style transfer for us!

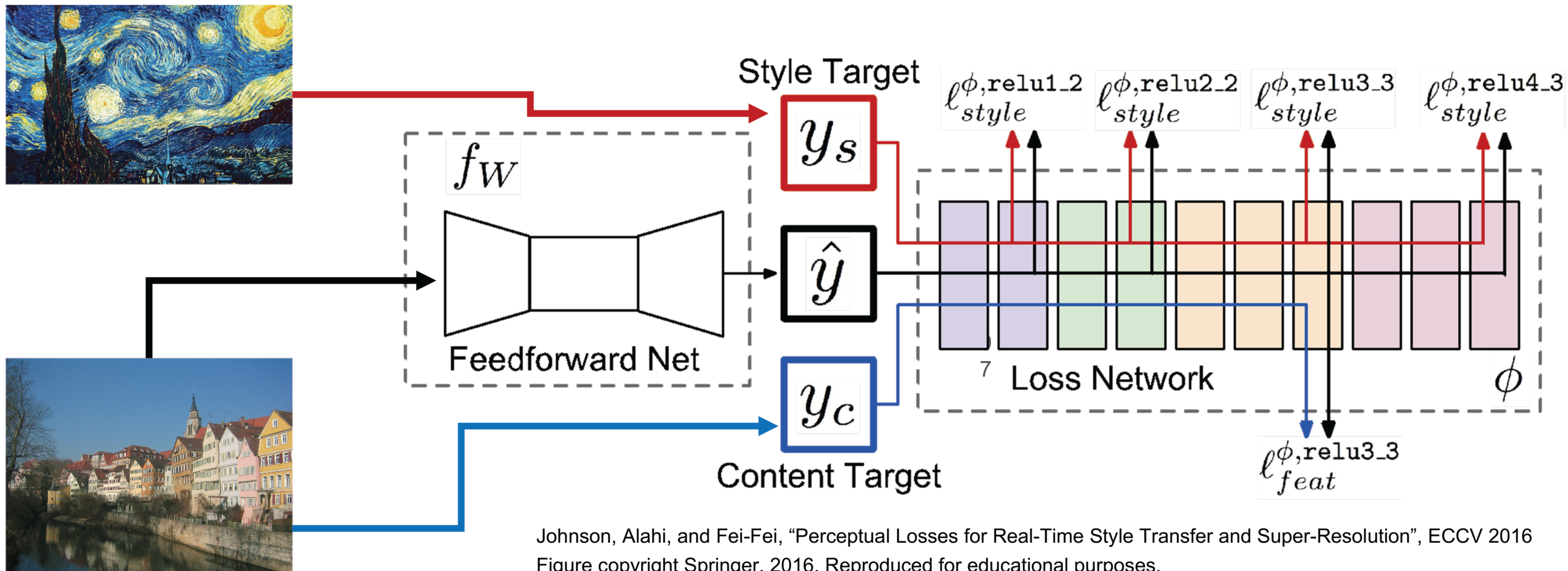
Co-financed by the European Union
Connecting Europe Facility

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

Neural Texture Synthesis: *Style transfer*

Fast Style Transfer

(1) Train a feedforward network for each style
(2) Use pretrained CNN to compute same losses as before
(3) After training, stylize images using a single forward pass

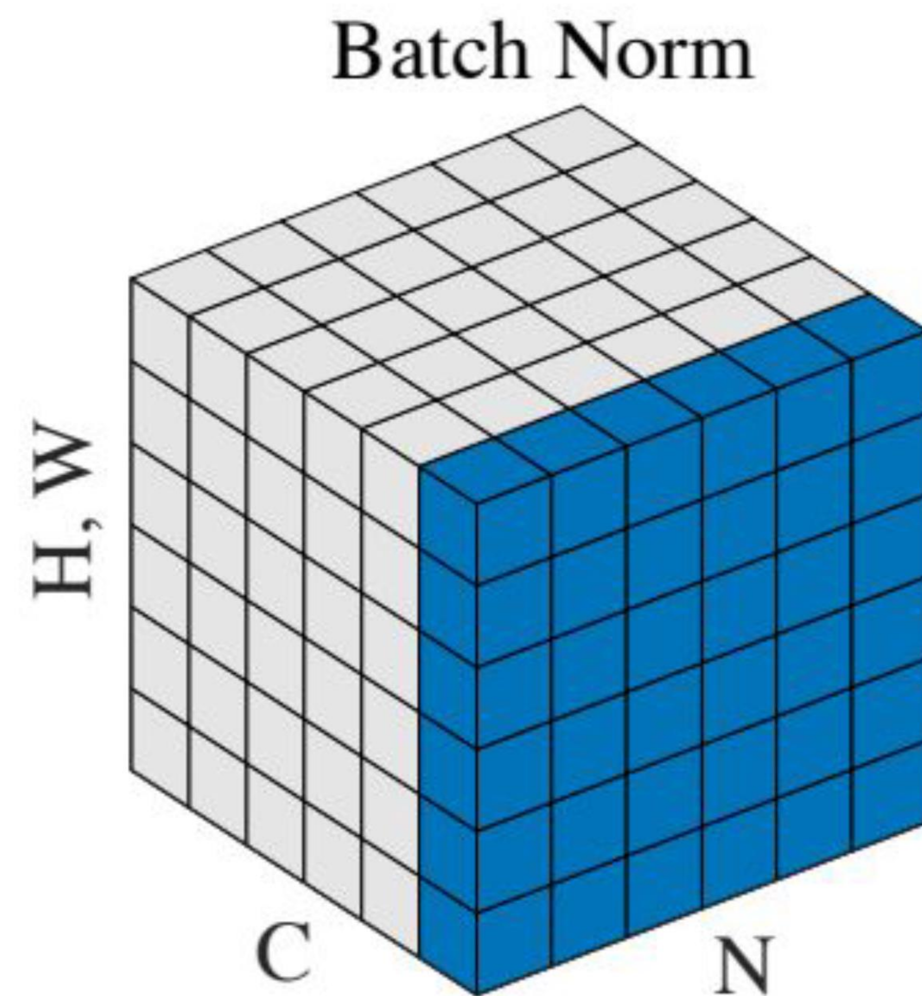


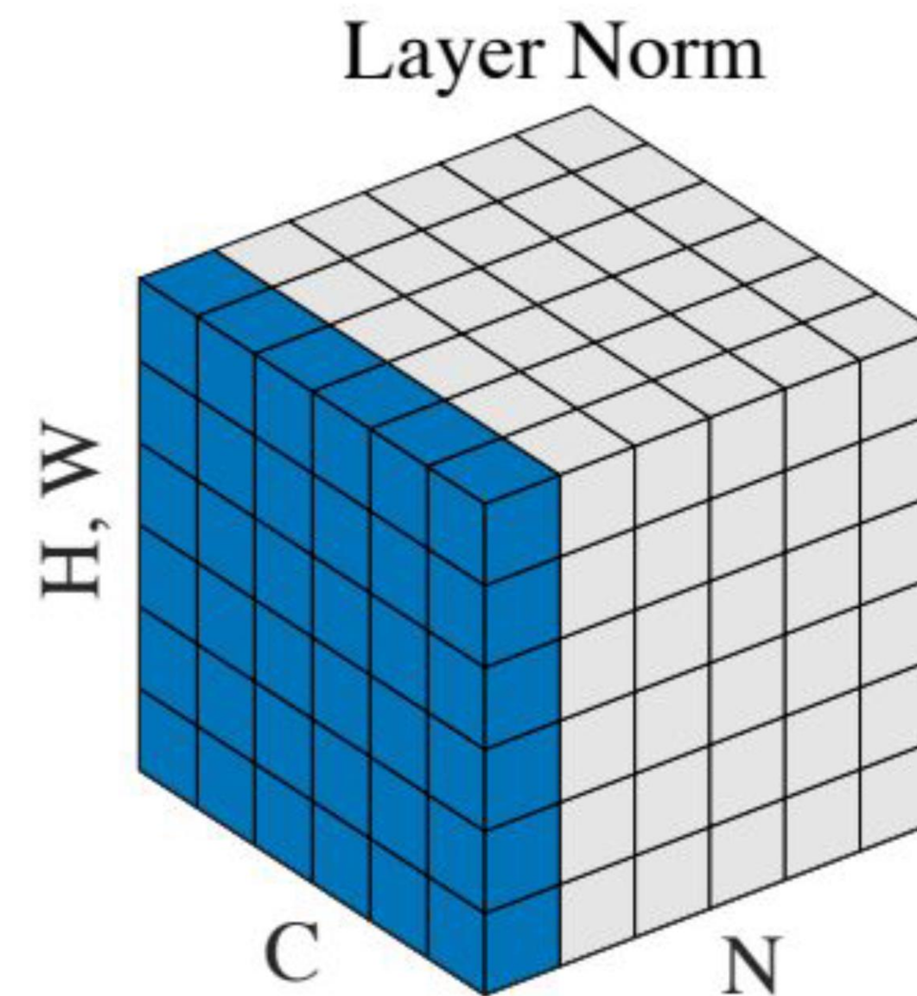
Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016
Figure copyright Springer, 2016. Reproduced for educational purposes.

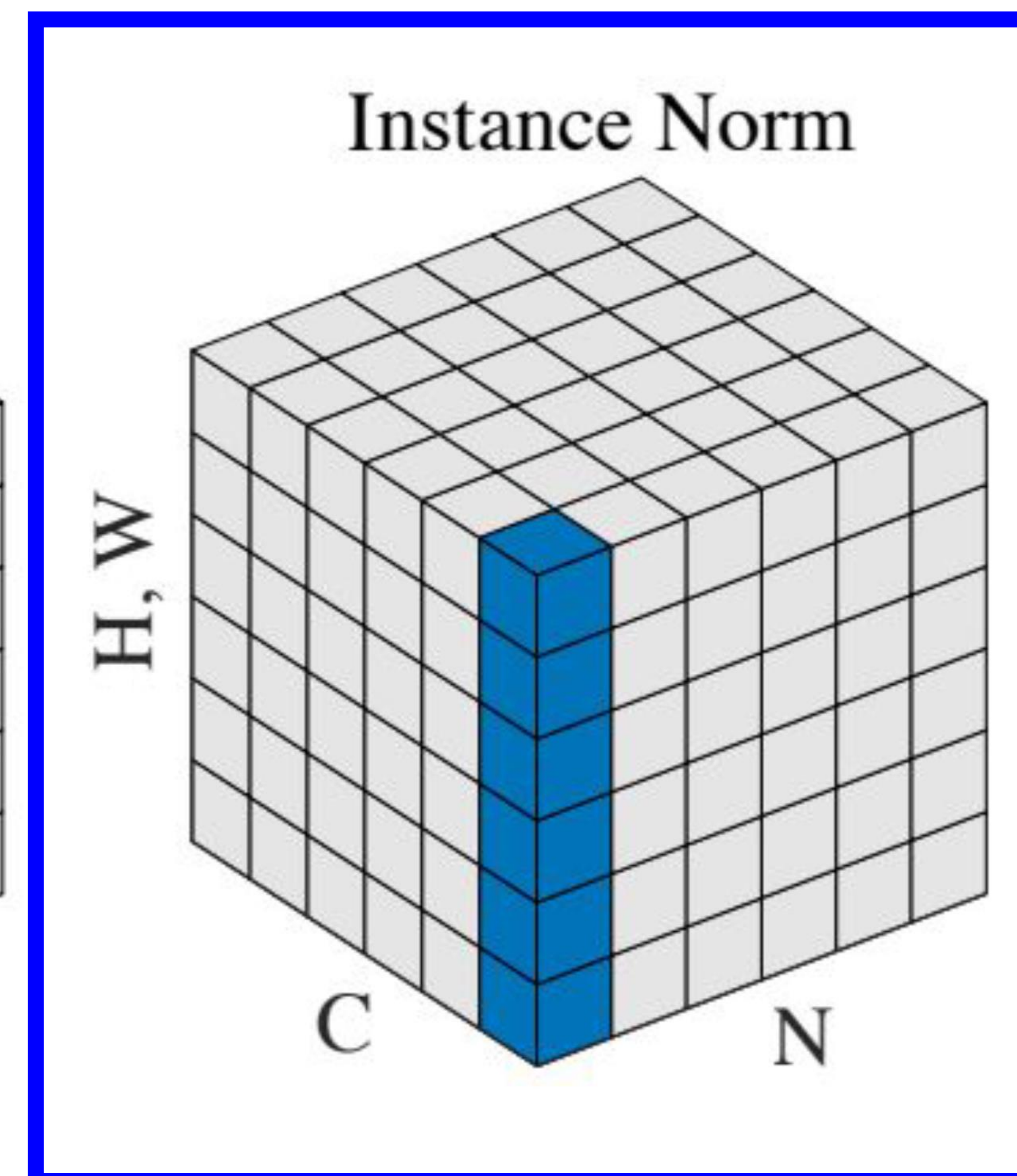
Neural Texture Synthesis: *Fast Style transfer*

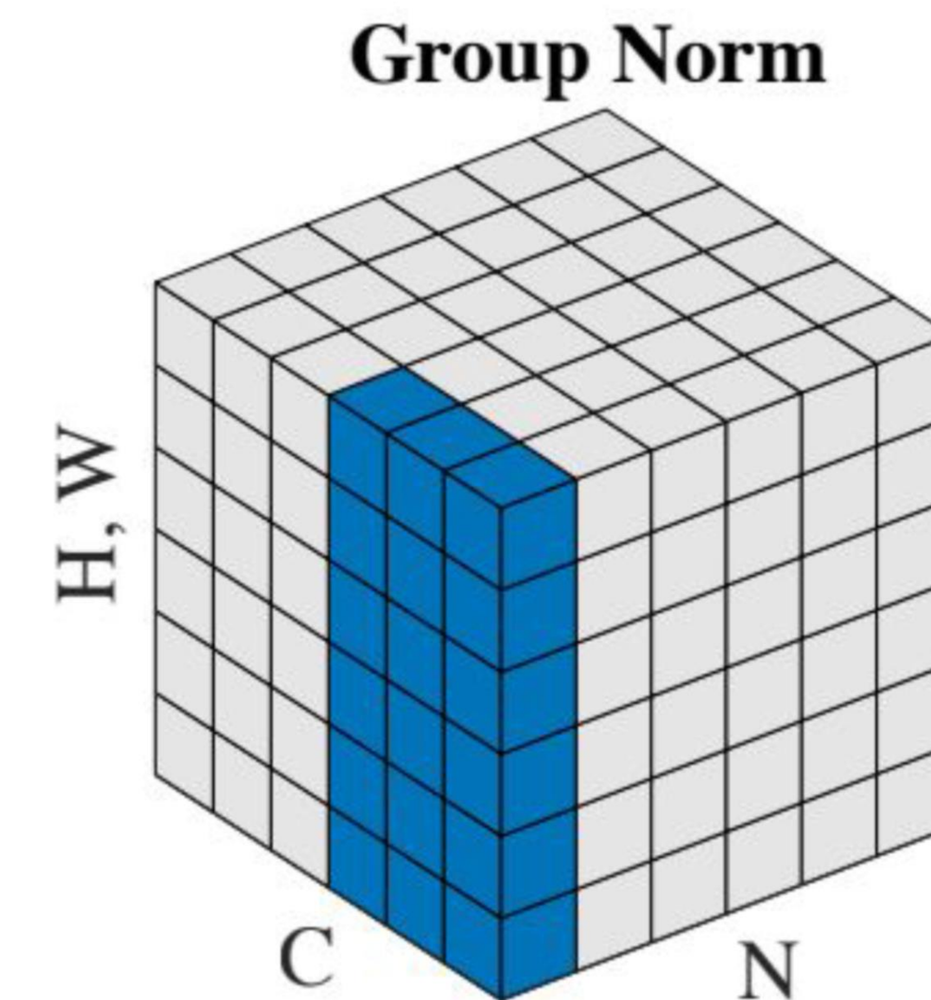
https://github.com/jcjohnson/fast-neural-style

Remember Normalization Methods?









Wu and He, “Group Normalization”, ECCV 2018

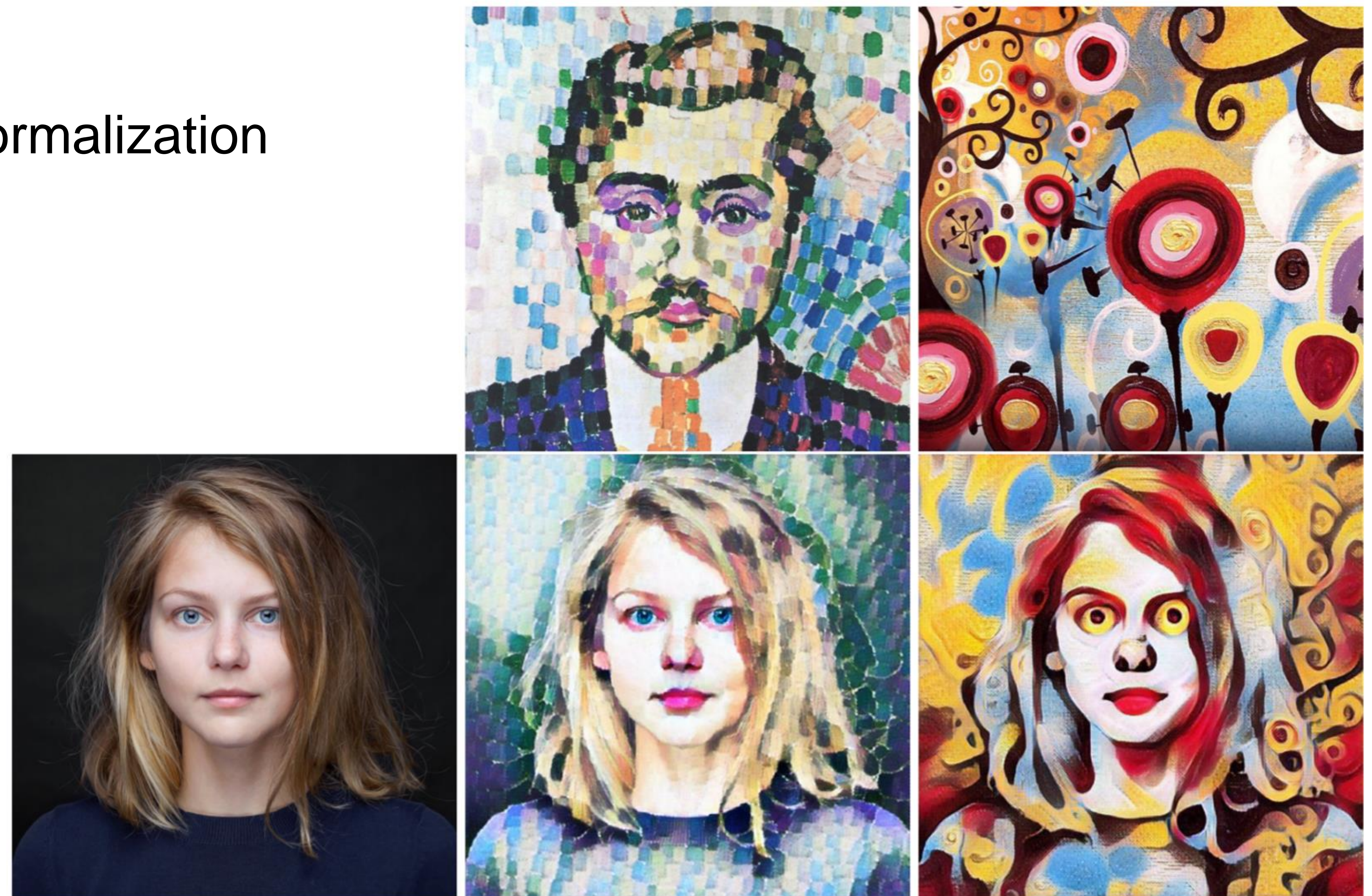
Instance Normalization was developed for style transfer!

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

Neural Texture Synthesis: *Fast Style transfer*

Replacing batch normalization with Instance Normalization improves results



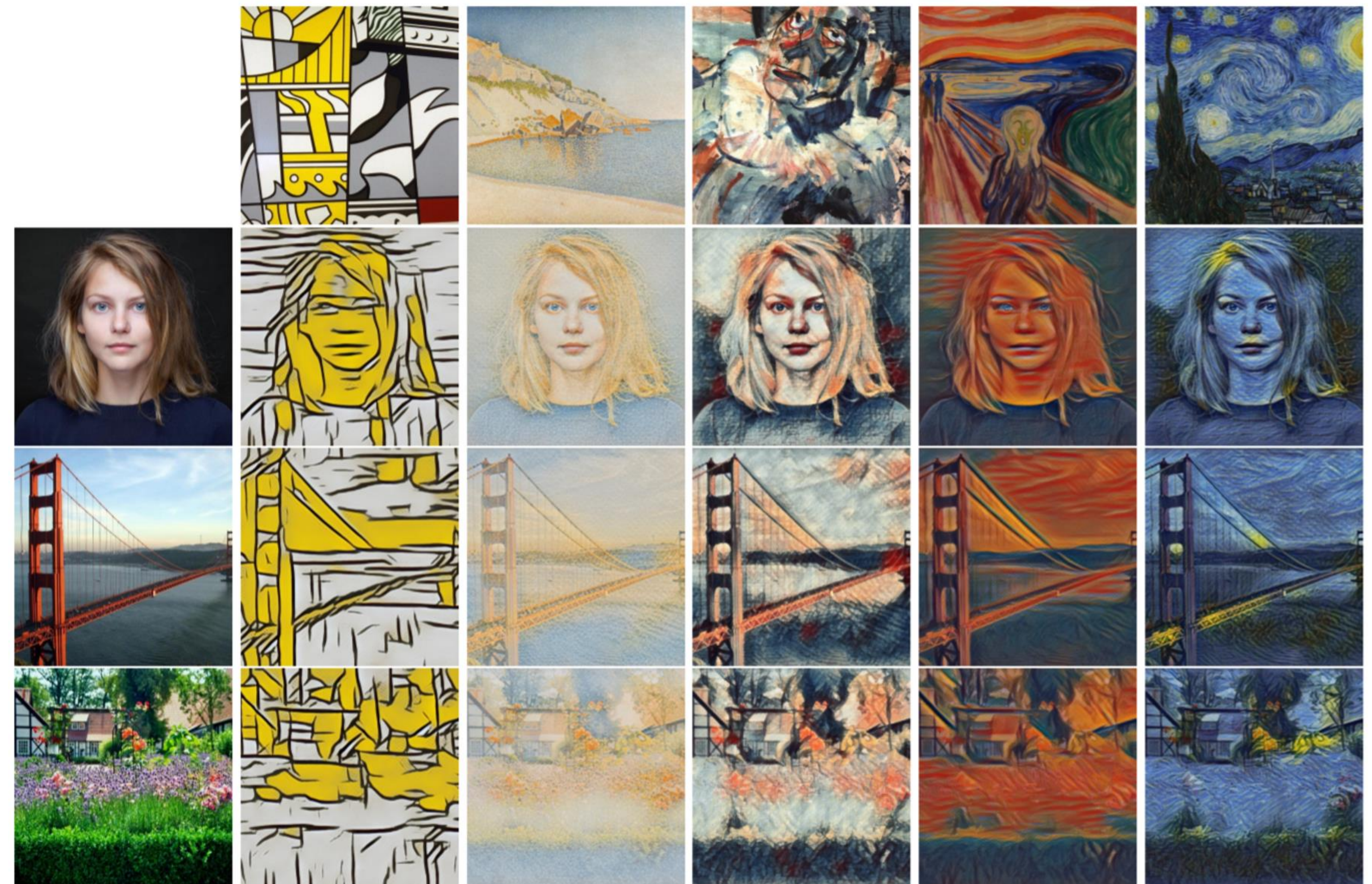
Ulyanov et al, "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images", ICML 2016
Ulyanov et al, "Instance Normalization: The Missing Ingredient for Fast Stylization", arXiv 2016

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

Neural Texture Synthesis: *Fast Style transfer*

One Network, Many Styles



Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.

Neural Texture Synthesis: *Fast Style transfer*

Single network can blend styles after training

Dumoulin, Shlens, and Kudlur, “A Learned Representation for Artistic Style”, ICLR 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

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