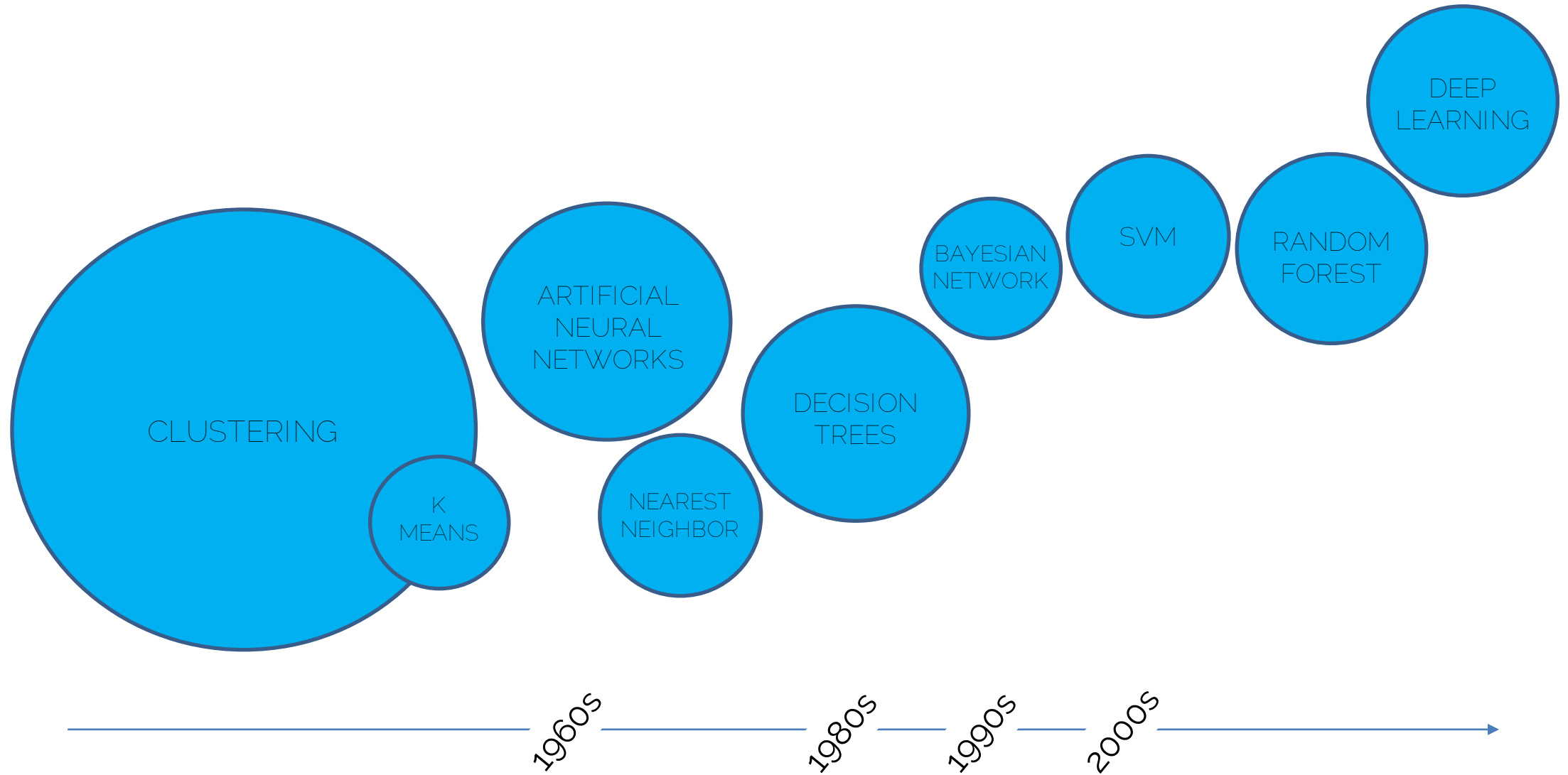


ML classifiers: Artificial Neural Networks and Deep Learning

Christian Salvatore
Scuola Universitaria Superiore IUSS Pavia

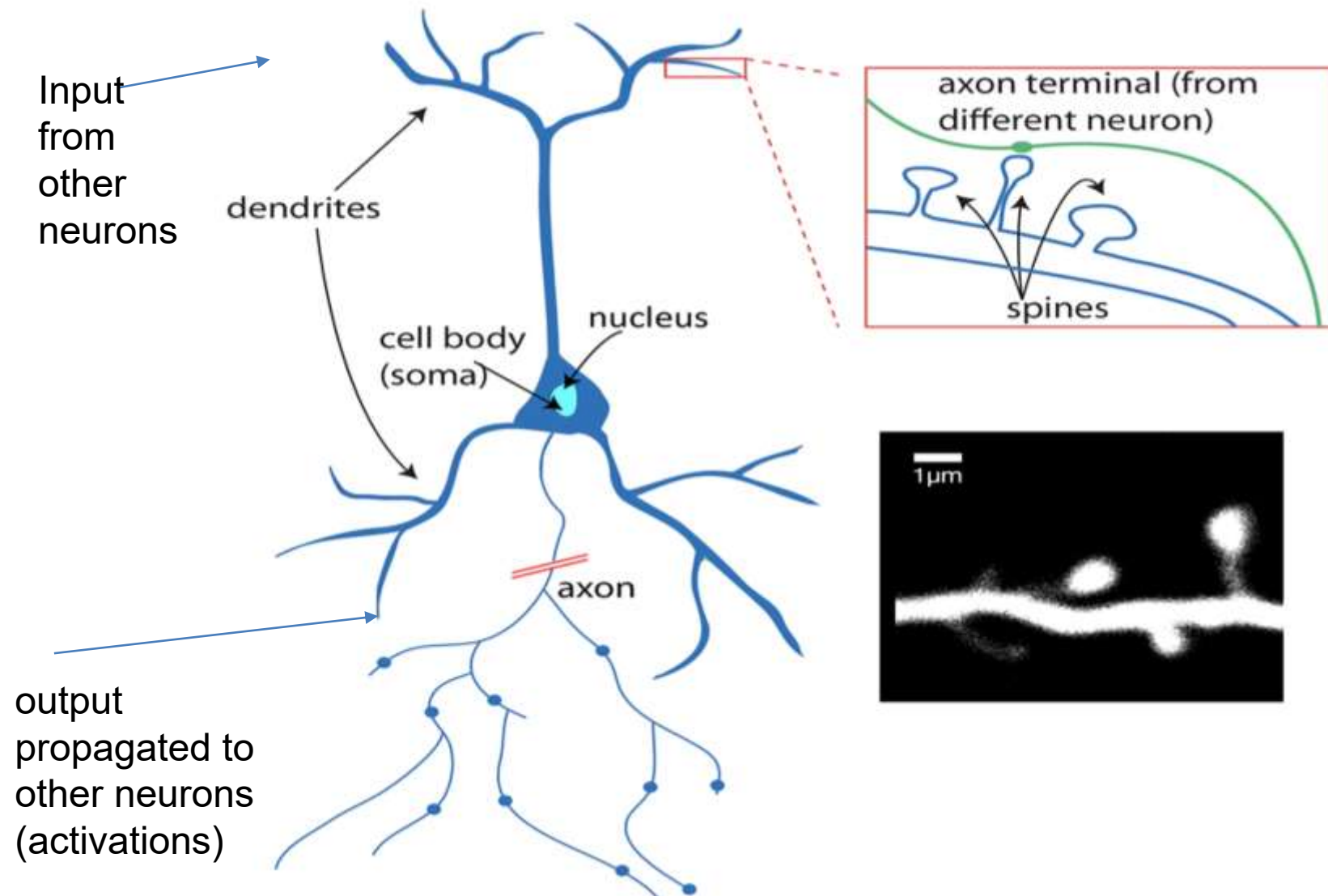
christian.salvatore@iusspavia.it

Machine learning

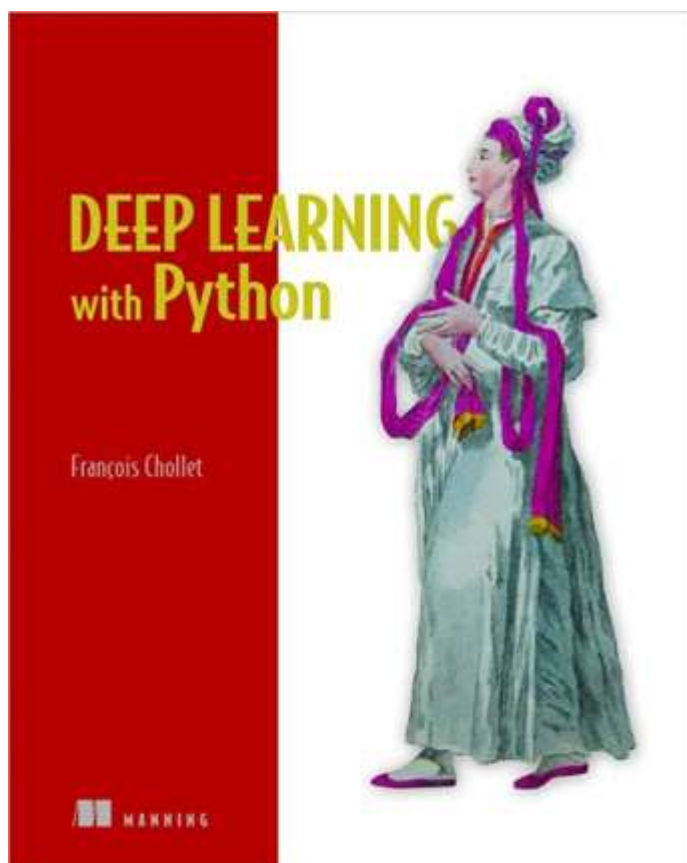


NEURAL NETWORKS

A Biological Neuron



Artificial Neural Networks



CS231n Convolutional Neural Networks for Visual Recognition

These notes accompany the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition. For questions/concerns/bug reports, please submit a pull request directly to our [git repo](#).

Spring 2023 Assignments

Assignment #1: Image Classification, kNN, SVM, Softmax, Fully Connected Neural Network

Assignment #2: Fully Connected and Convolutional Nets, Batch Normalization, Dropout, Pytorch & Network Visualization

(To be released) Assignment #3: Image Captioning with RNNs and Transformers, Network Visualization, Generative Adversarial Networks, Self-Supervised Contrastive Learning

Module 0: Preparation

Software Setup

Python / Numpy Tutorial (with Jupyter and Colab)

Module 1: Neural Networks

Image Classification: Data-driven Approach, k-Nearest Neighbor, train/val/test splits

[L1/L2 distances](#), [hyperparameter search](#), [cross-validation](#)

Linear classification: Support Vector Machine, Softmax

[parameteric approach](#), [bias trick](#), [hinge loss](#), [cross-entropy loss](#), [L2 regularization](#), [web demo](#)

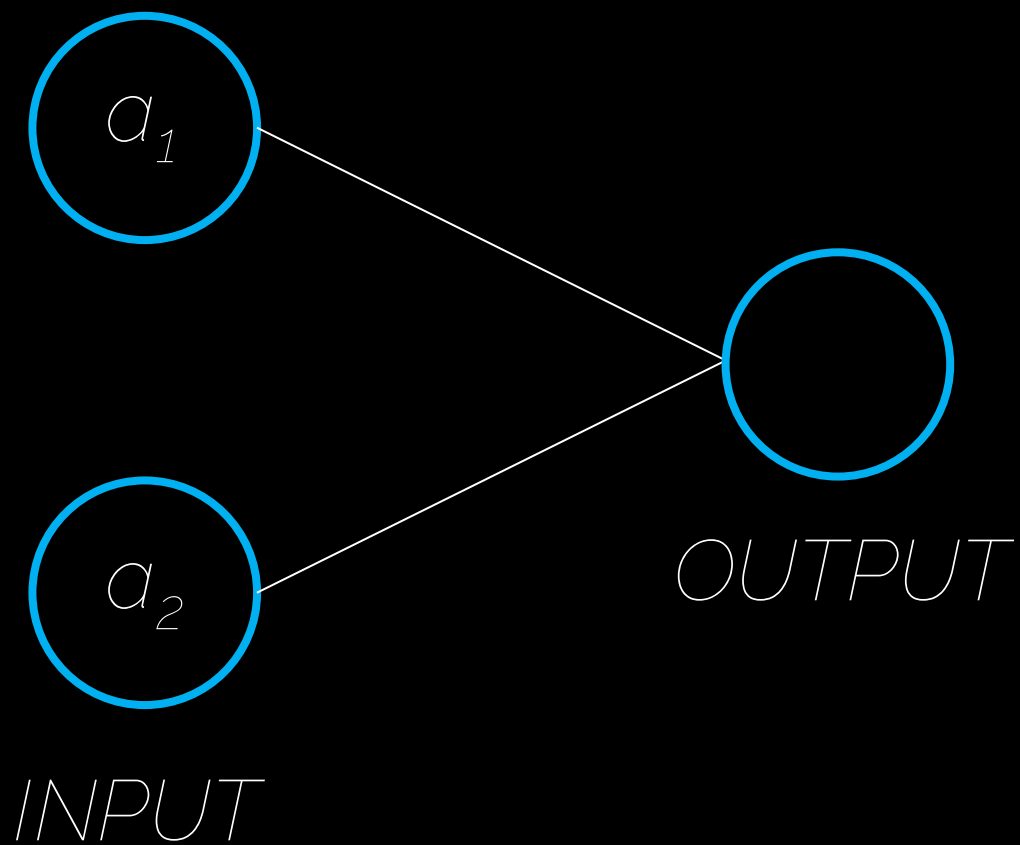
Optimization: Stochastic Gradient Descent

[optimization landscapes](#), [local search](#), [learning rate](#), [analytic/numerical gradient](#)

<https://cs231n.github.io/>



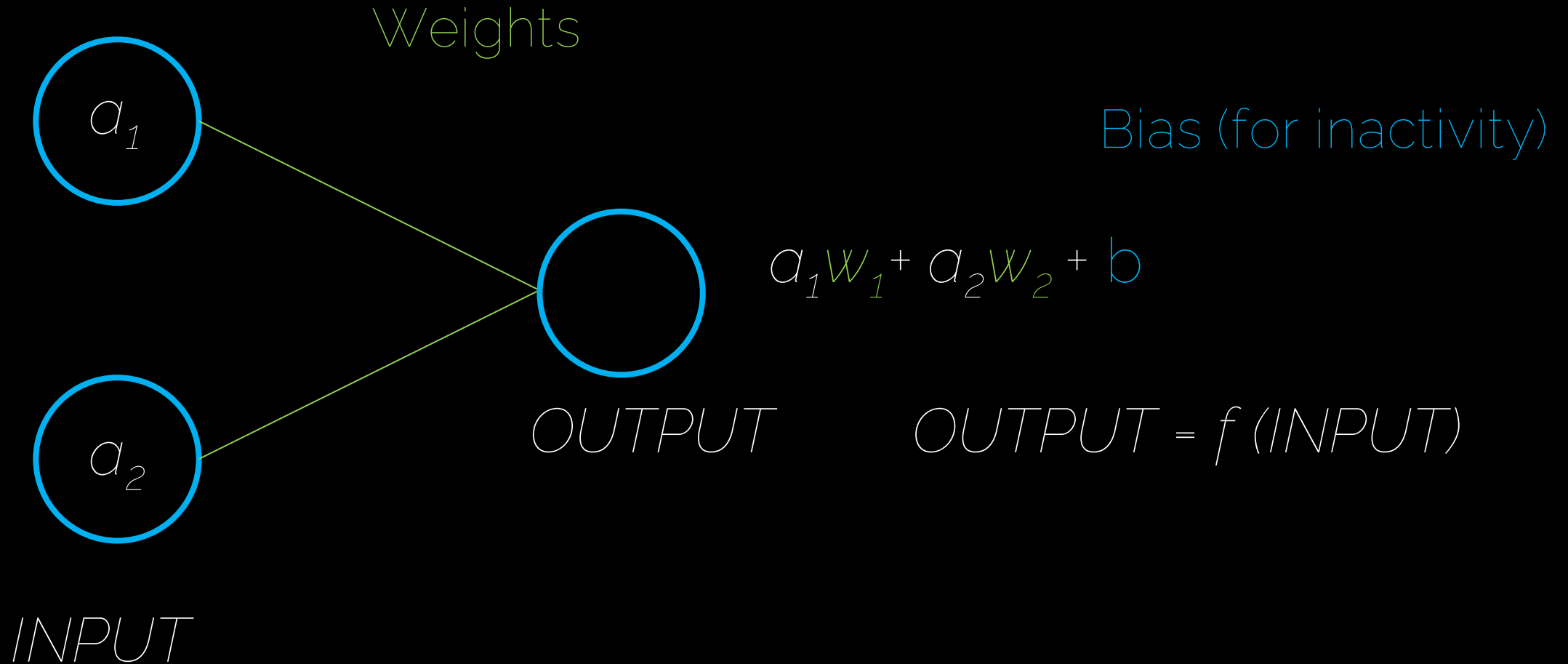
Neural Network



$$OUTPUT = f(INPUT)$$

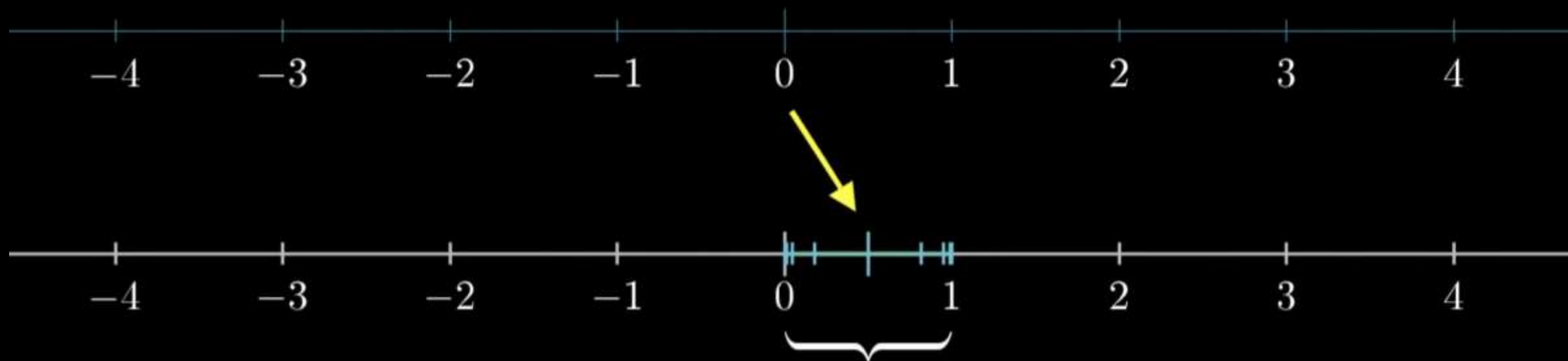
$$\begin{array}{ccc} 0 & & 1 \\ \hline \end{array}$$

Neural Network | Weights and Bias

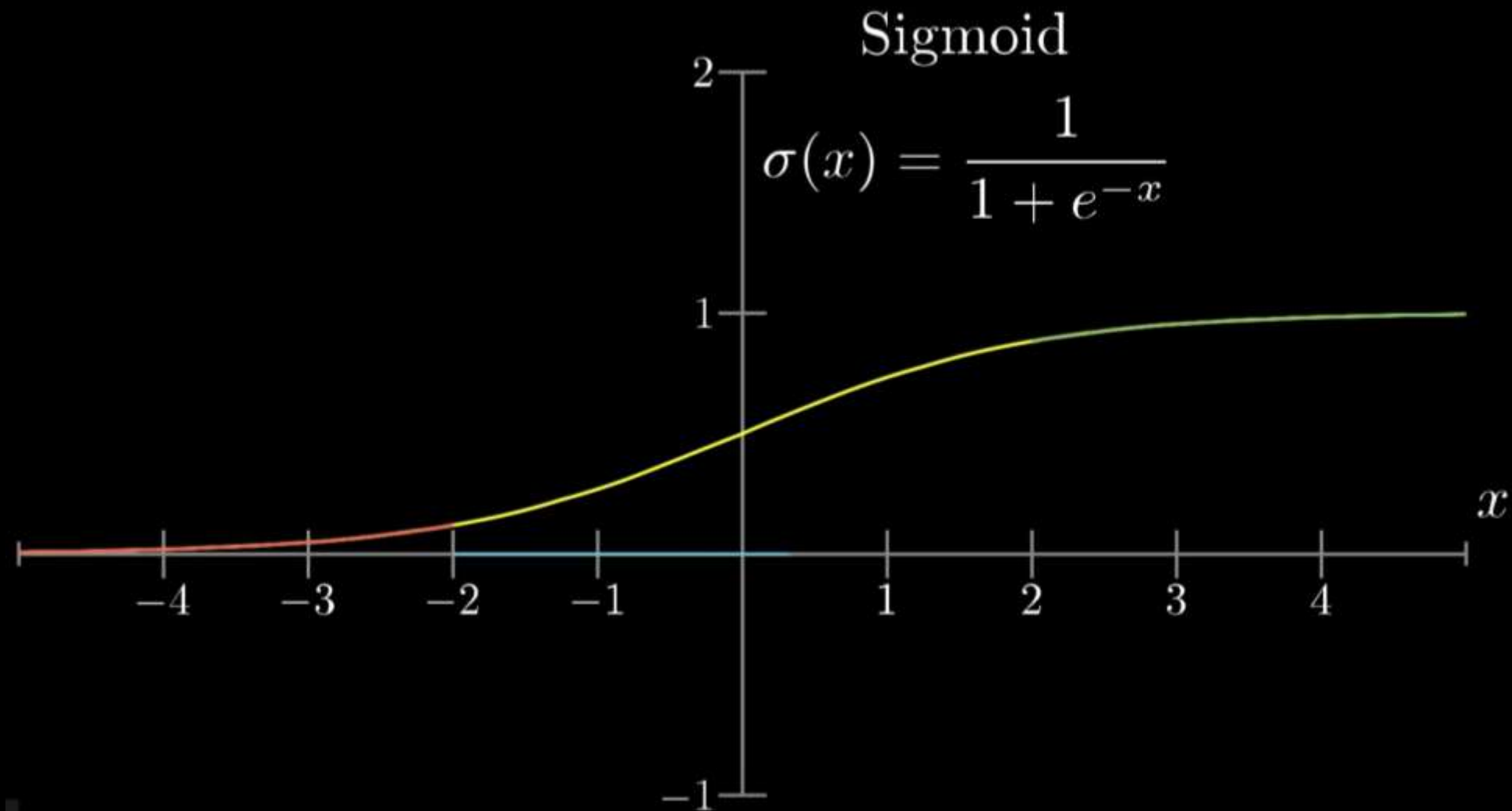




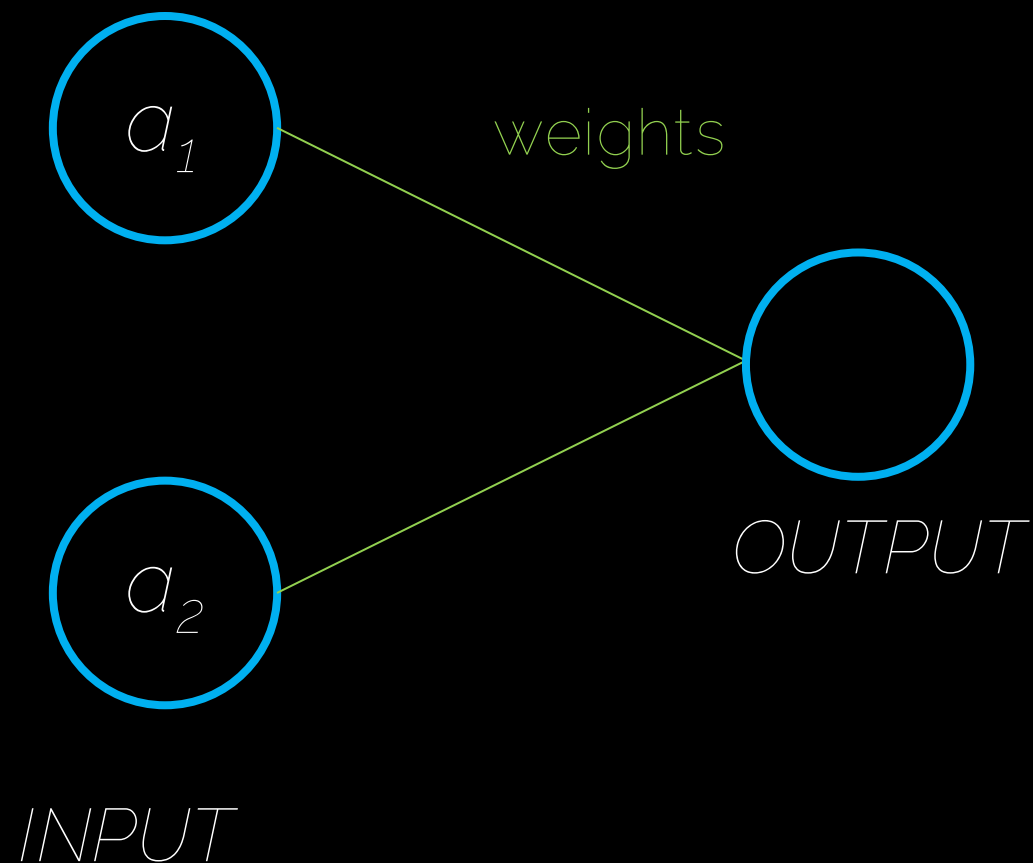
$$w_1 a_1 + w_2 a_2$$



Activations should be in this range



Neural Network | Activation function



Forward propagation

Activation function

$\text{Sigmoid}(a_1 w_1 + a_2 w_2 + b)$

$$\text{OUTPUT} = f(\text{INPUT})$$

$w_1?$ $w_2?$ $b?$

INPUT

a_1

a_2

Cost Function?

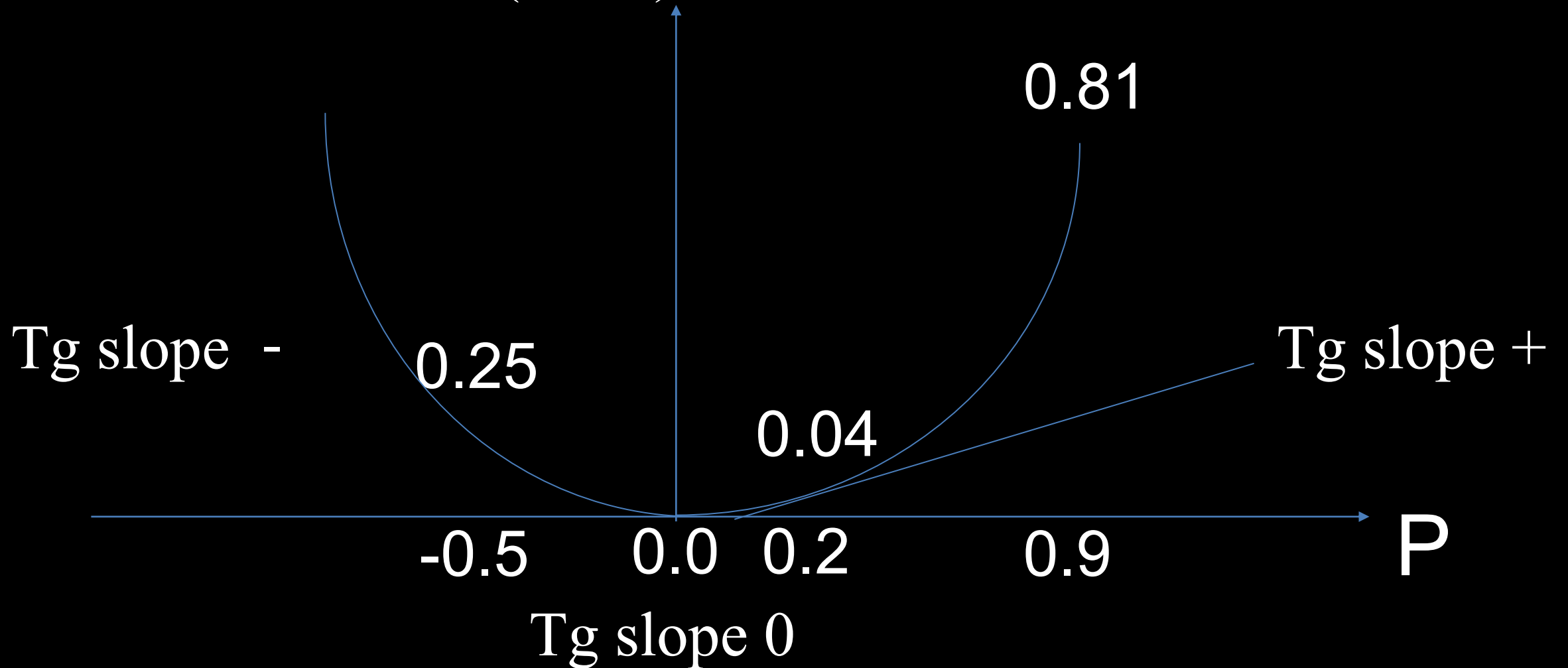
Sigmoid ($a_1 w_1 + a_2 w_2 + b$)

OUTPUT

Cost Function = (Predicted-Expected)²

Squared error cost

The Cost Function = $(P-0.0)^2$



Neural Network

If the slope is + we must decrease P of a fraction of the slope

If the slope is - we must increase P of a fraction of the slope

If the slope is 0 we have the solution

Neural Network

Training a single-neuron
neural network

-> backward propagation

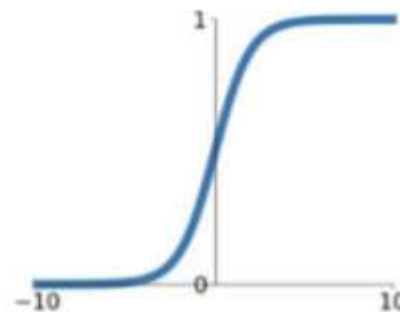


Neural Network

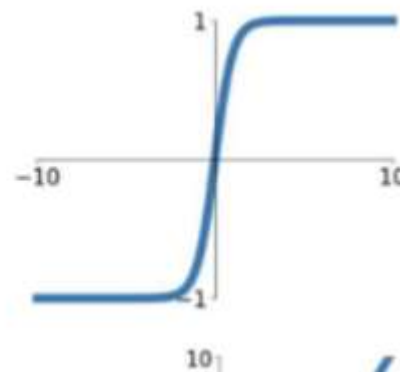
Other functions that progressively change from 0 to 1 with no discontinuity

Sigmoid

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



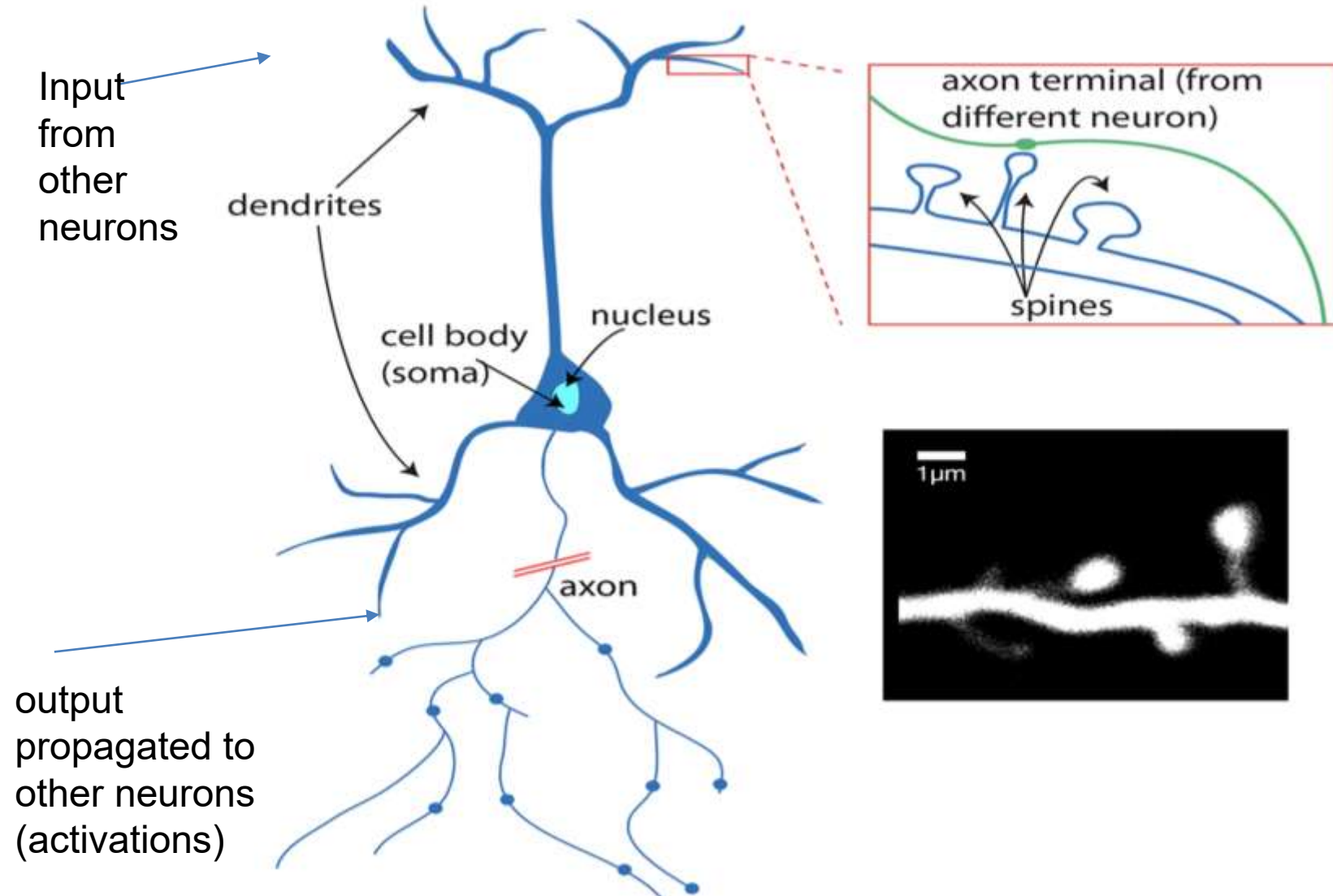
tanh
 $\tanh(x)$



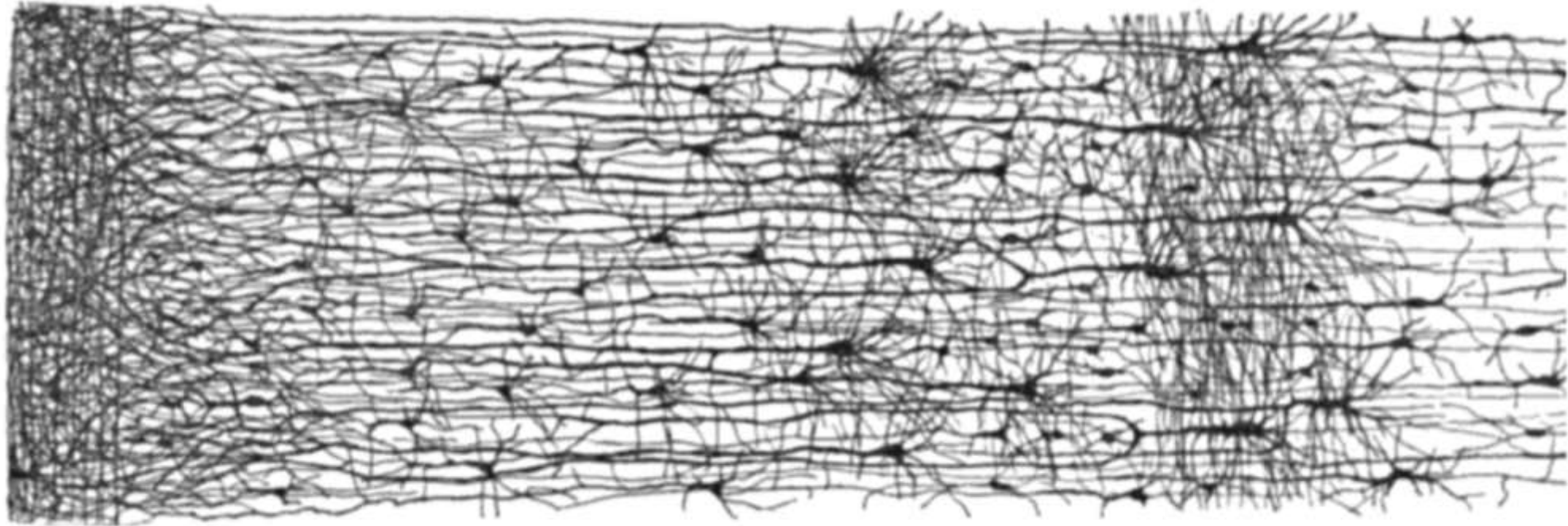
Hyperbolic
tangent
function

DEEP NEURAL NETWORKS (Deep Learning)

A Biological Neuron

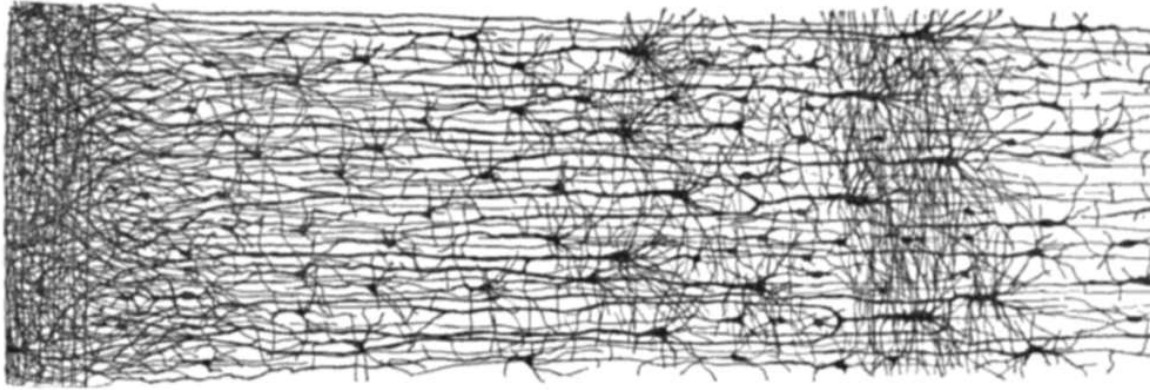


A Biological Neuron Network

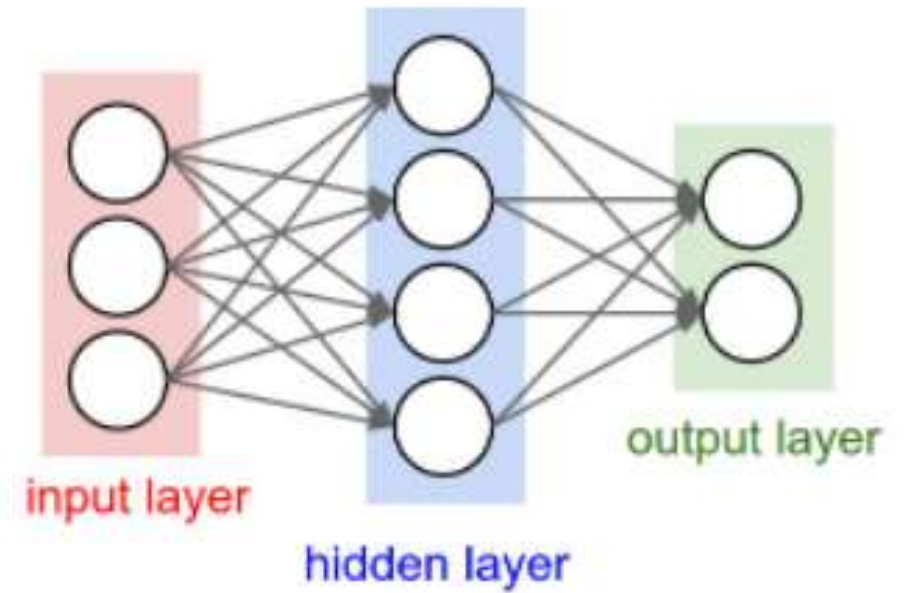


100 B neurons

A Biological Neuron Network



100 B neurons



Neural Network

Training a multi-layer / deep
neural network

-> generalized backward
propagation



Problems in training performance

- Vanishing Gradient
- Overfitting
- Computational load

Vanishing gradients

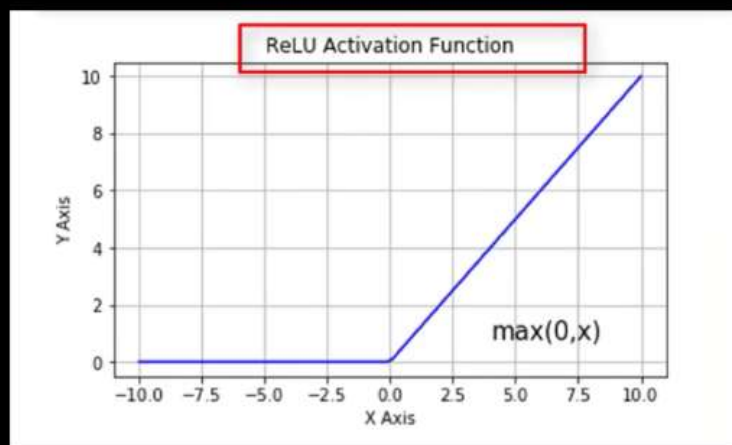
During training each weight receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training.

In some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.

As one example of the problem cause, traditional activation functions such as the hyperbolic tangent function have gradients in the range $(-1, 1)$, and backpropagation computes gradients by the chain rule. This has the effect of multiplying n of these small numbers to compute gradients of the "front" layers in an n -layer network, meaning that the gradient (error signal) decreases exponentially with n while the front layers train very slowly.

Vanishing gradients

Can be solved using Rectified Linear Unit function (ReLU) and its derivative as activation function



$$\varphi(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$
$$= \max(0, x)$$



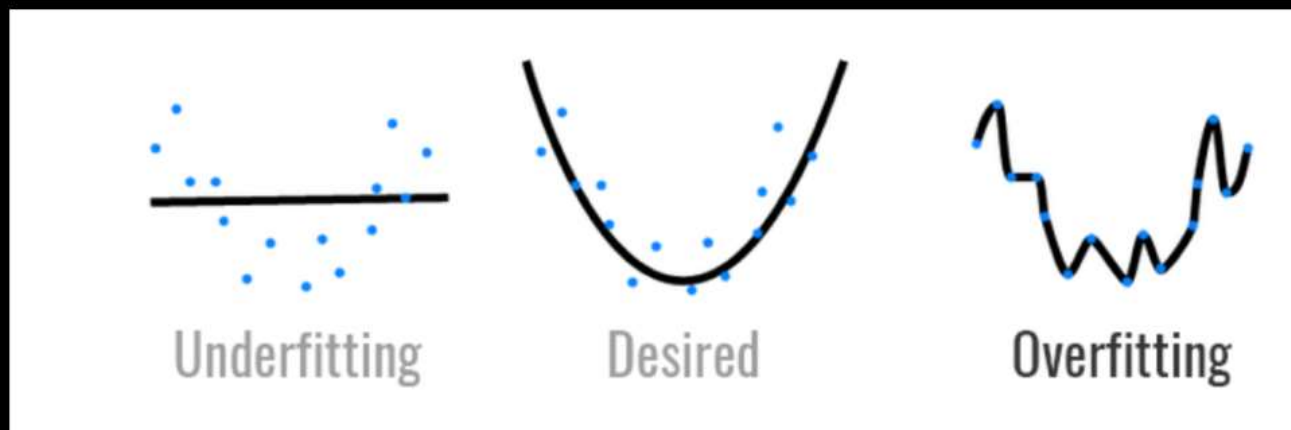
$$\varphi'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Computational load

Can be solved by GPU, Batch Normalization method

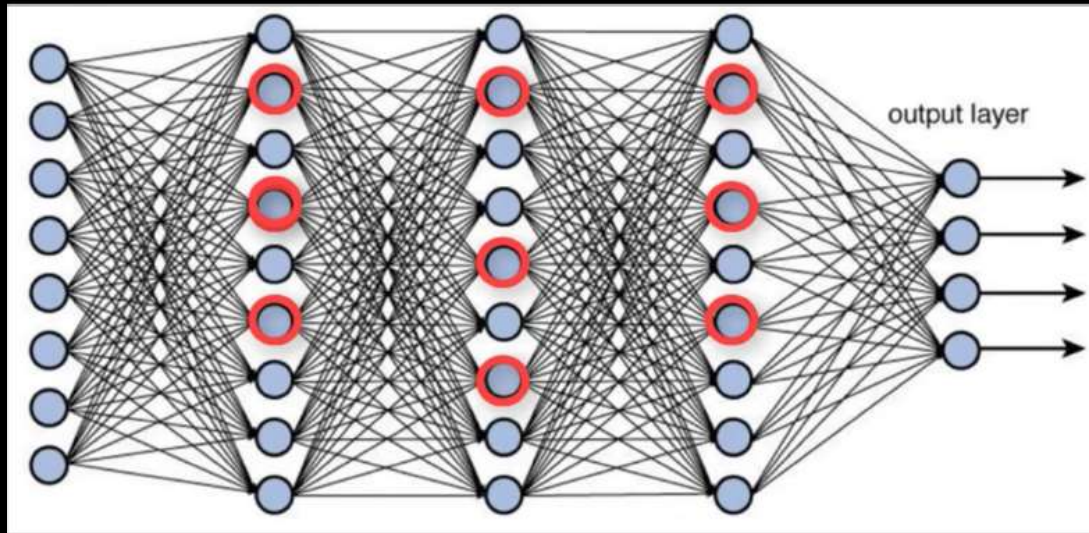
Overfitting

It refers to a model that models the training data too well. Instead of learning the genral **distribution** of the data, the model learns the *expected output* for every data point.

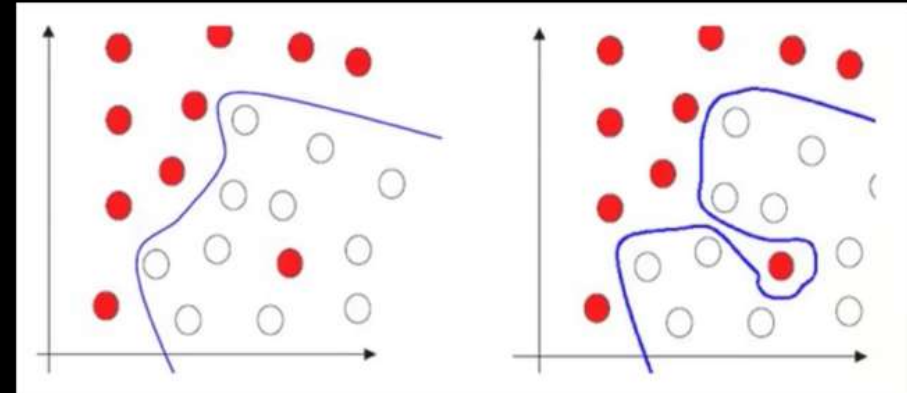


Overfitting

Can be solved using Dropout or Regularization

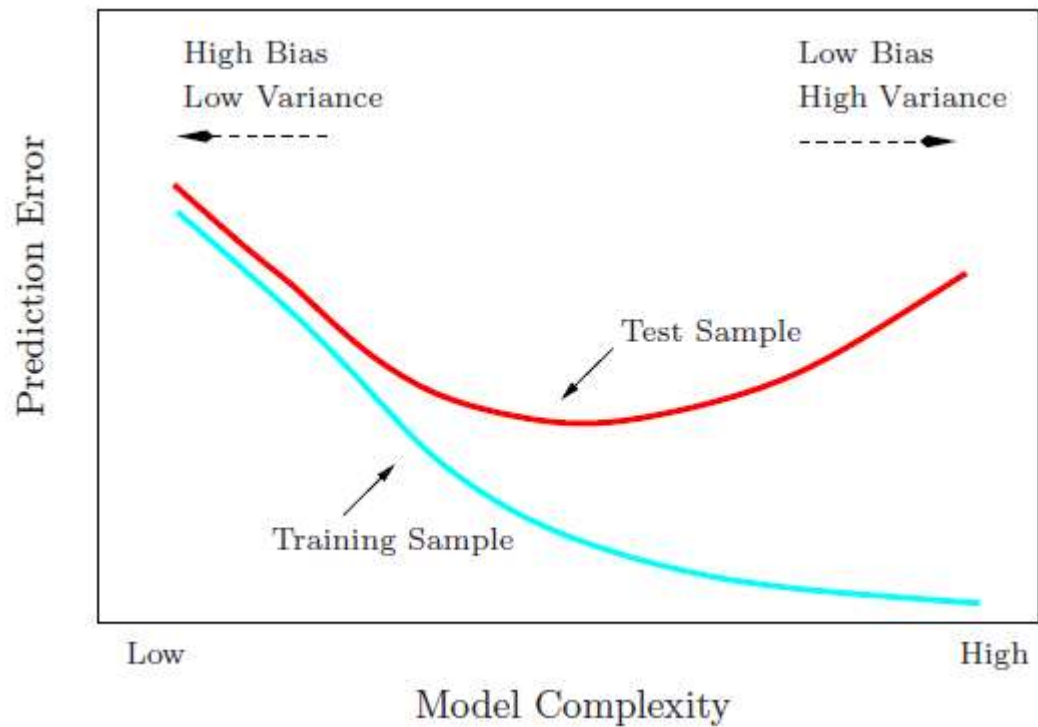


Dropout



Regularization

Deep Learning



	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none">• Complexity model• Add more features• Train longer		<ul style="list-style-type: none">• Perform regularization• Get more data

When weight update should be calculated?

Batch: error is calculated for all training data, each of the weight updates are calculated but the average of all weight updates are used only once in each epoch.

In **Stochastic Gradient Descent**, error is calculated for each training data, weights updated immediately.

With **Mini-Batch Gradient Descent**, we have a mix of the previous situations.

Mini-Batch method

1-10

11-20

21-40

41-60

61-80

5 weight update will be performed to complete the training process

Robustness of SGD
Efficiency of batch

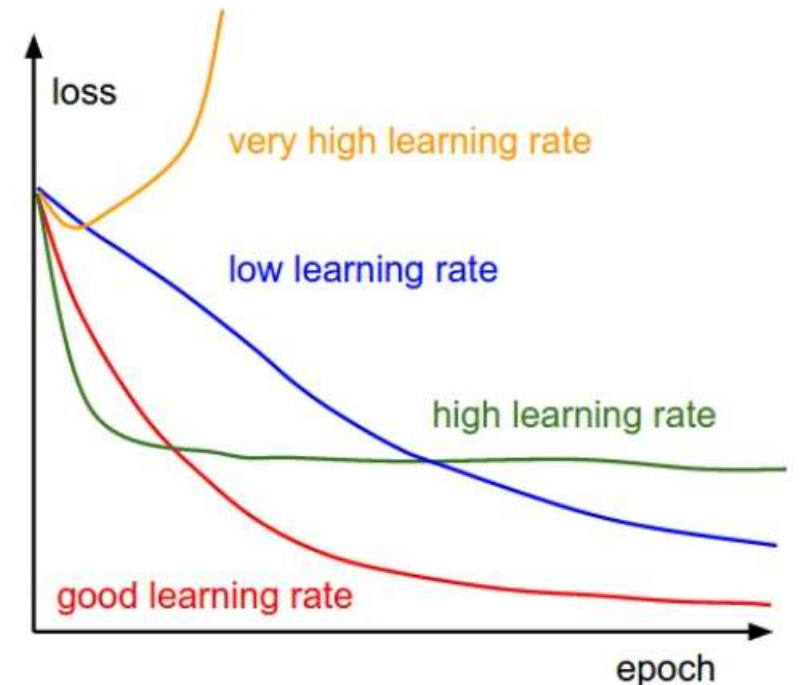
Learning rate and optimizer for weight update

Learning rate determines how much weights are changed every time

Too high → output wanders around the expected solution

Too low → output fails to converge to acceptable solution

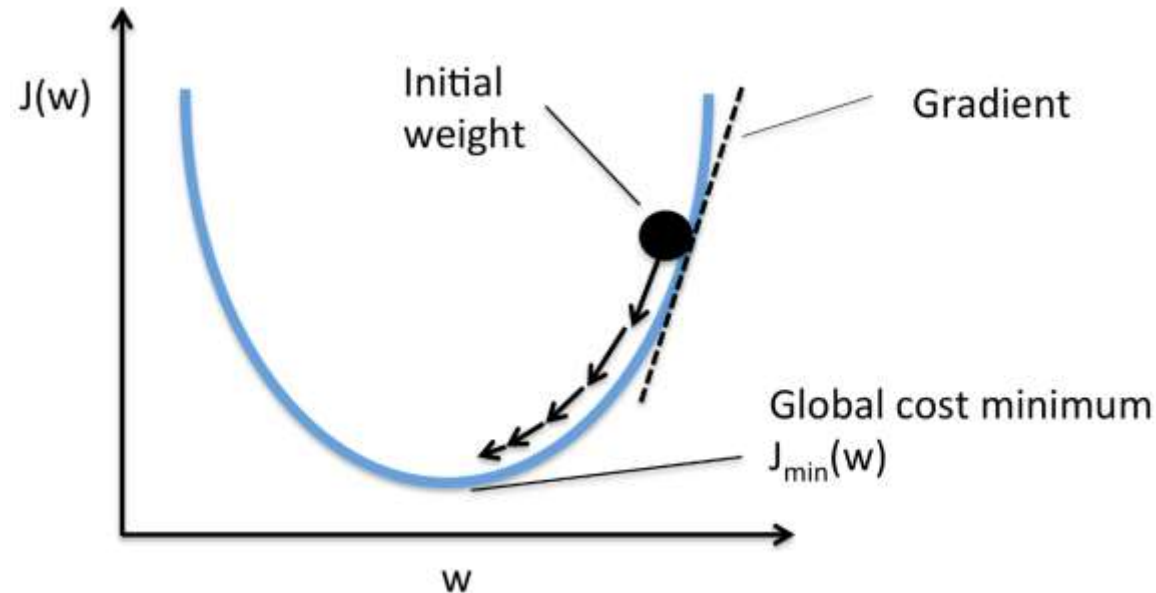
- > Constant learning rate
- > Learning rate schedule
 - (constant) step decay
 - exponential decay $\alpha = \alpha_0 e^{-kt}$
 - 1/t decay $\alpha = \alpha_0 / (1 + kt)$



Optimization methods are techniques used to improve the weight update in order to improve the performance of the network (for example, by reaching a solution in a faster way)

Stochastic Gradient Descent is the one described above

-> parameters are updated along the negative gradient direction

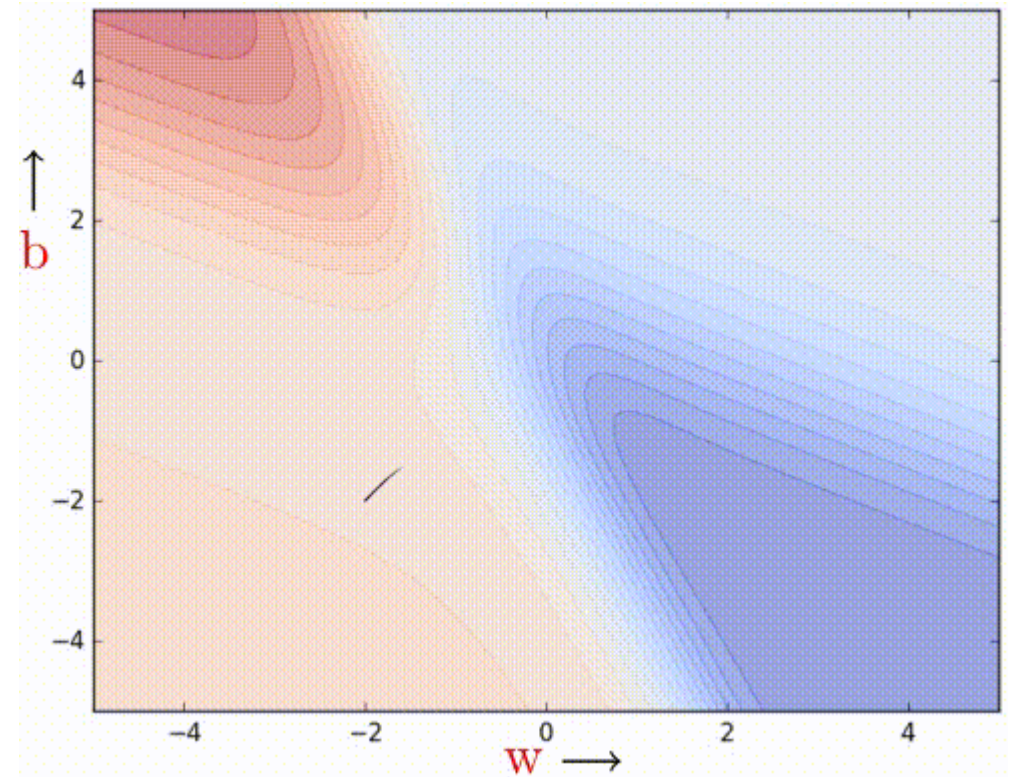


Neural Network

Momentum

Parameters are updated as a function not only of the present gradient, but also of previous steps (introducing a sort of “velocity” parameters)

```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```



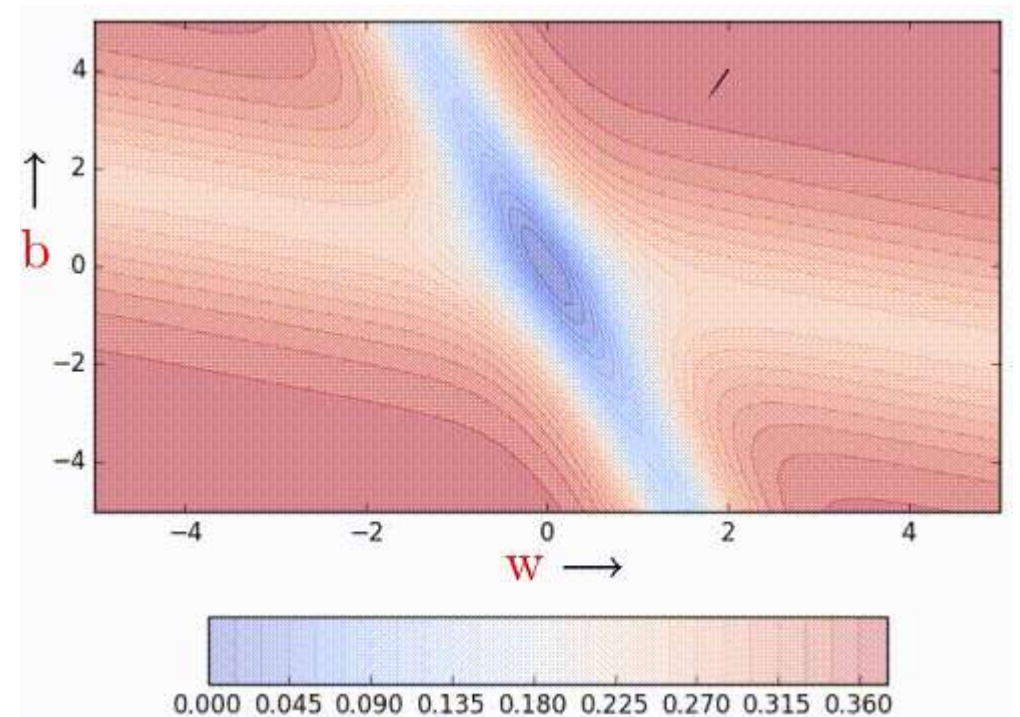
Vanilla SDG vs Momentum, 100 iterations

Neural Network

Momentum

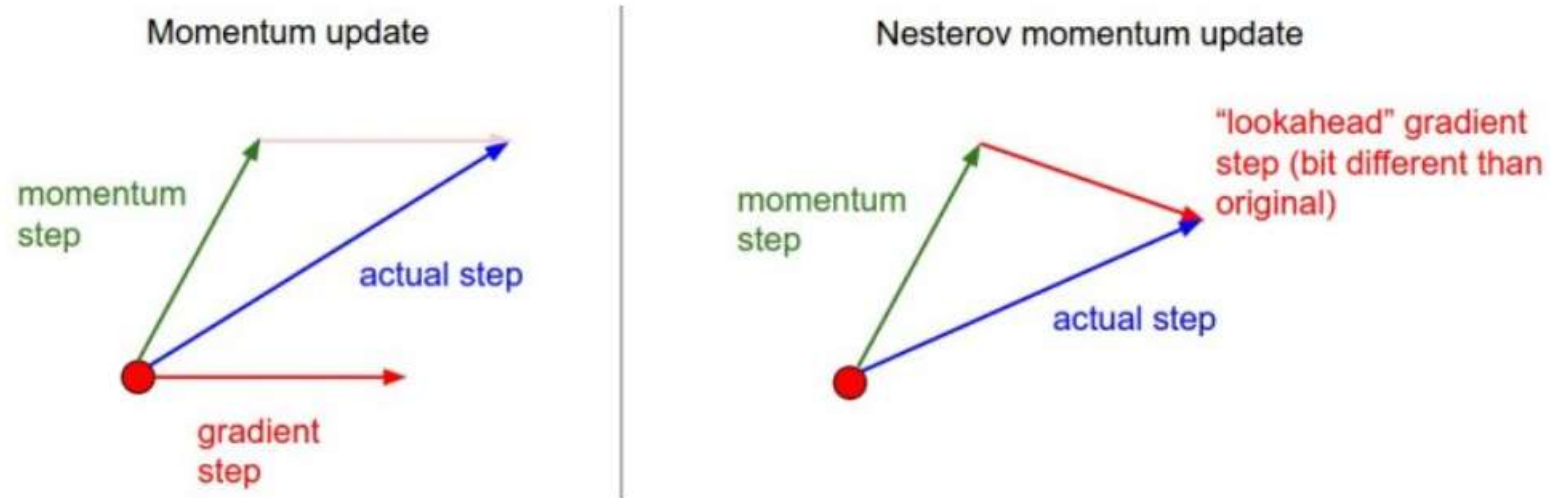
Parameters are updated as a function not only of the present gradient, but also of previous steps (introducing a sort of “velocity” parameters)

```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```



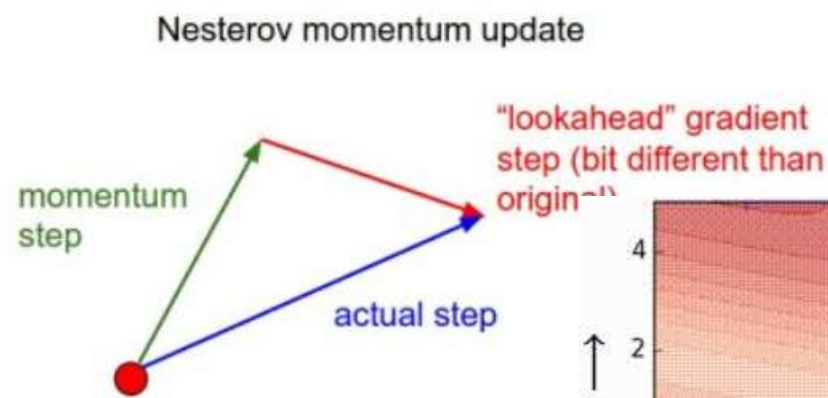
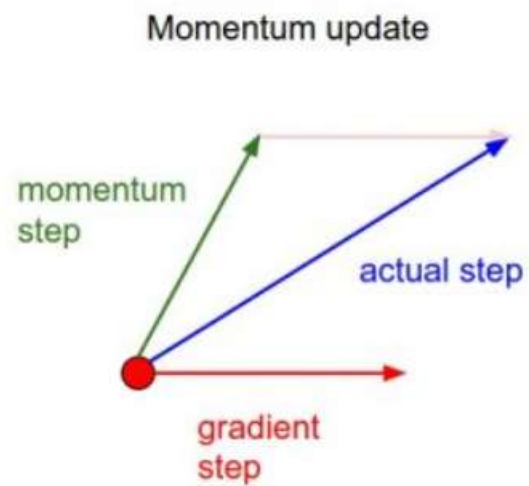
Vanilla SDG vs Momentum, 100 iterations

Nesterov Accelerated Gradient

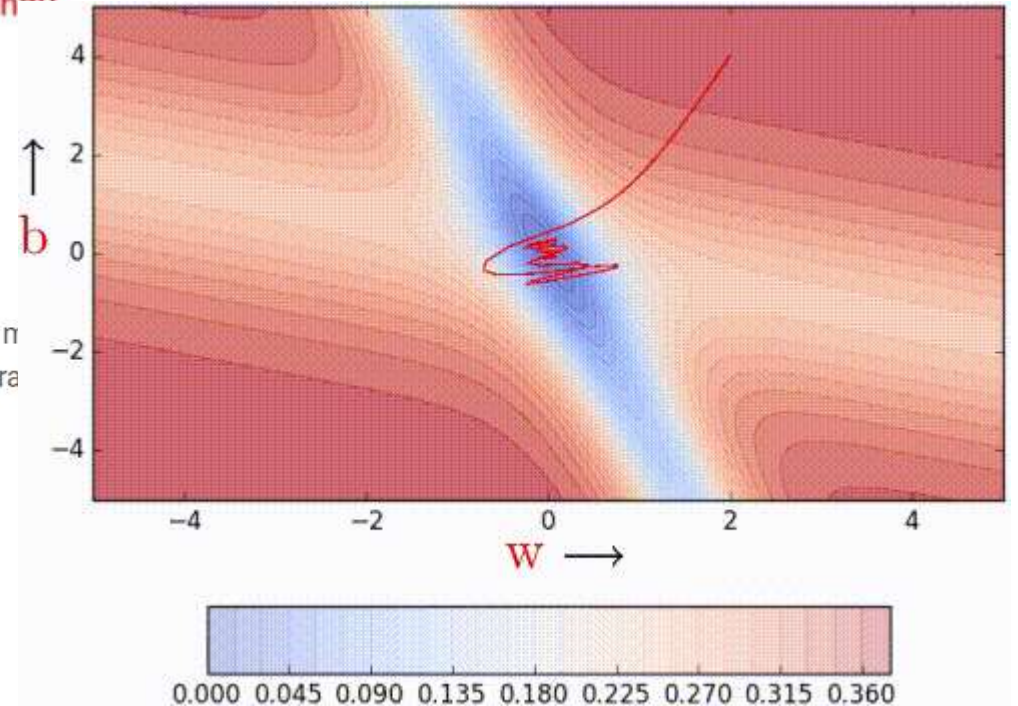


Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

Nesterov Accelerated Gradient



Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our next step will carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at the "lookahead" position.



Adagrad

```
cache += dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

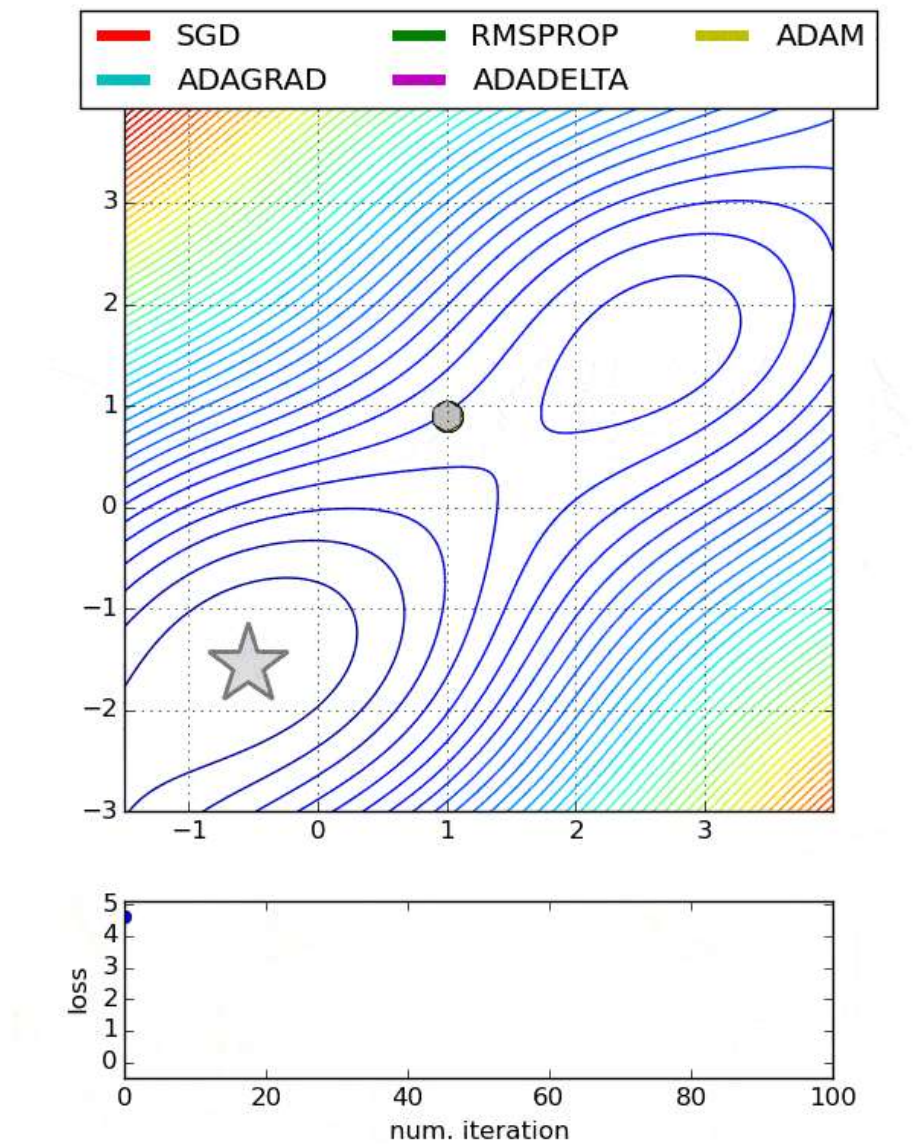
RMSprop

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

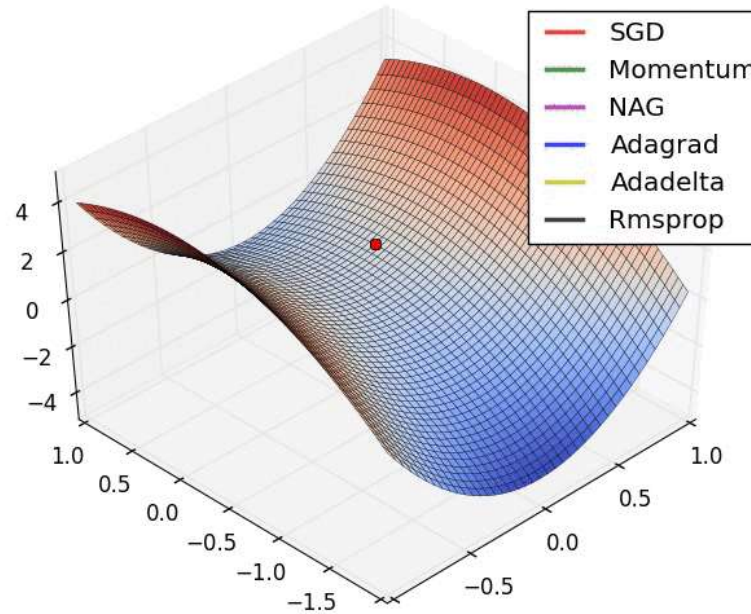
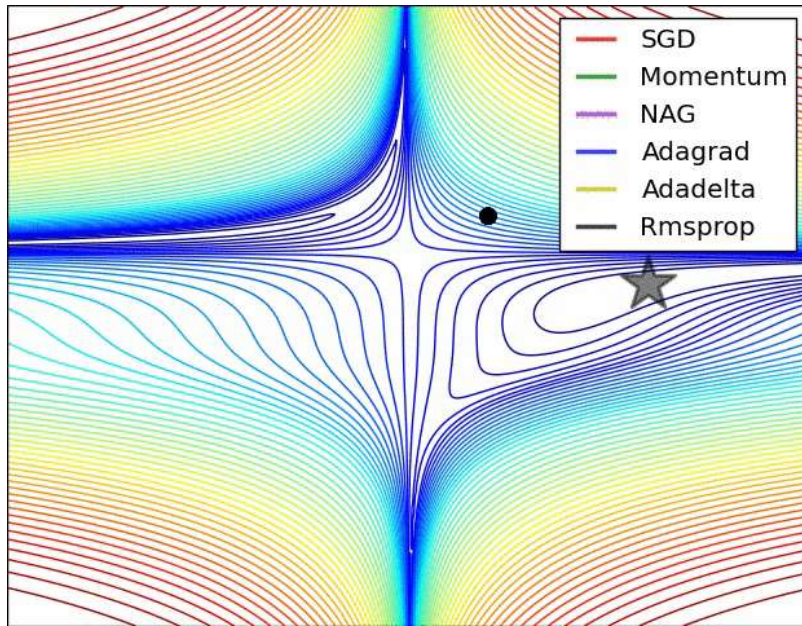
Adam (Adaptive Moment Estimation)

```
m = beta1*m + (1-beta1)*dx  
v = beta2*v + (1-beta2)*(dx**2)  
x += - learning_rate * m / (np.sqrt(v) + eps)
```

Optimizer



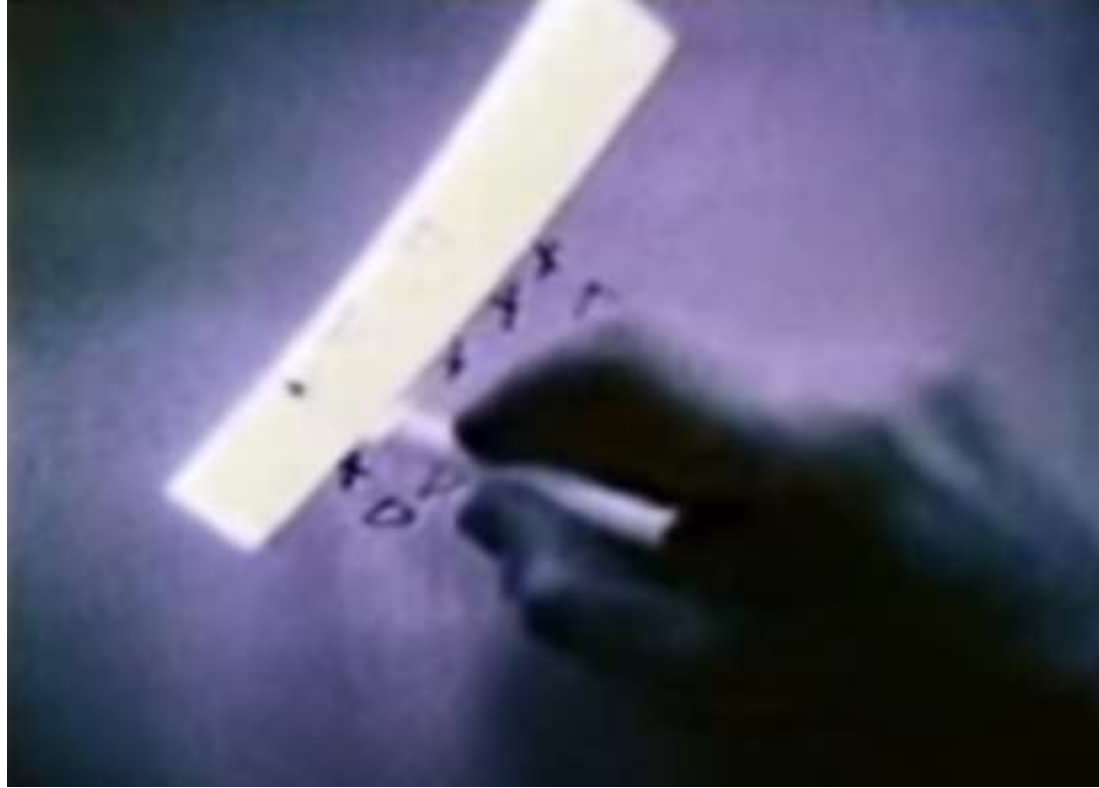
Optimizer



Animations that may help your intuitions about the learning process dynamics. **Left:** Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill. **Right:** A visualization of a saddle point in the optimization landscape, where the curvature along different dimension has different signs (one dimension curves up and another down). Notice that SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSprop proceed. Images credit: [Alec Radford](#).

(Deep) CONVOLUTIONAL NEURAL NETWORKS

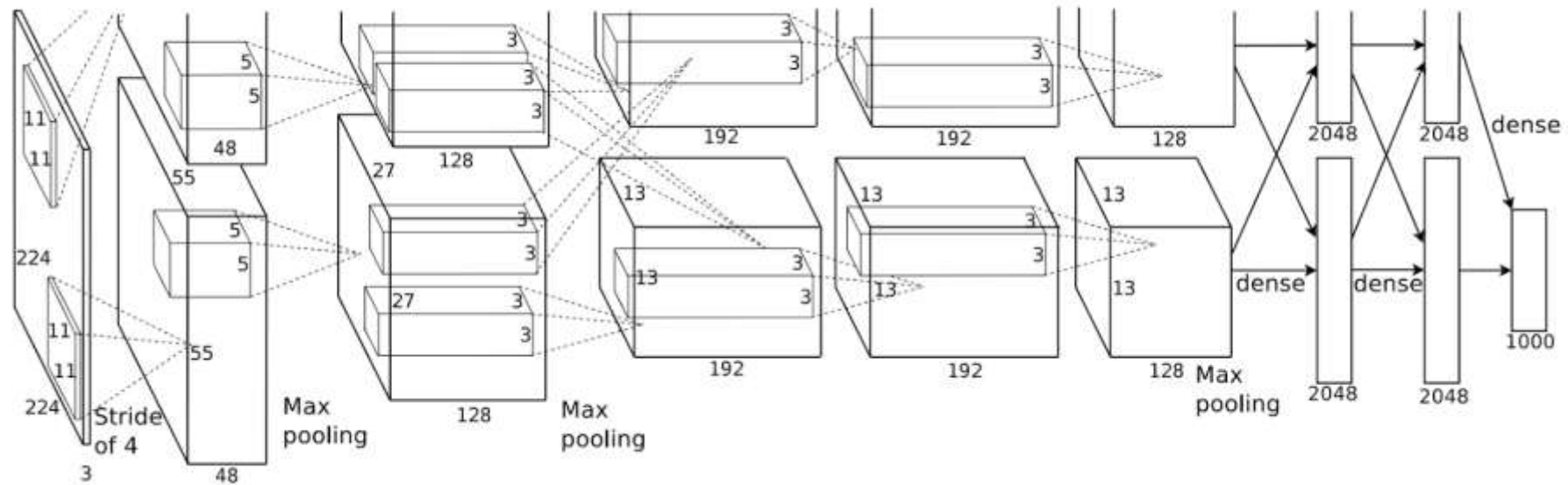
Convolutional Neural Networks



Hubel and Wiesel

<https://www.youtube.com/watch?v=Cw5PKVgRj3o>

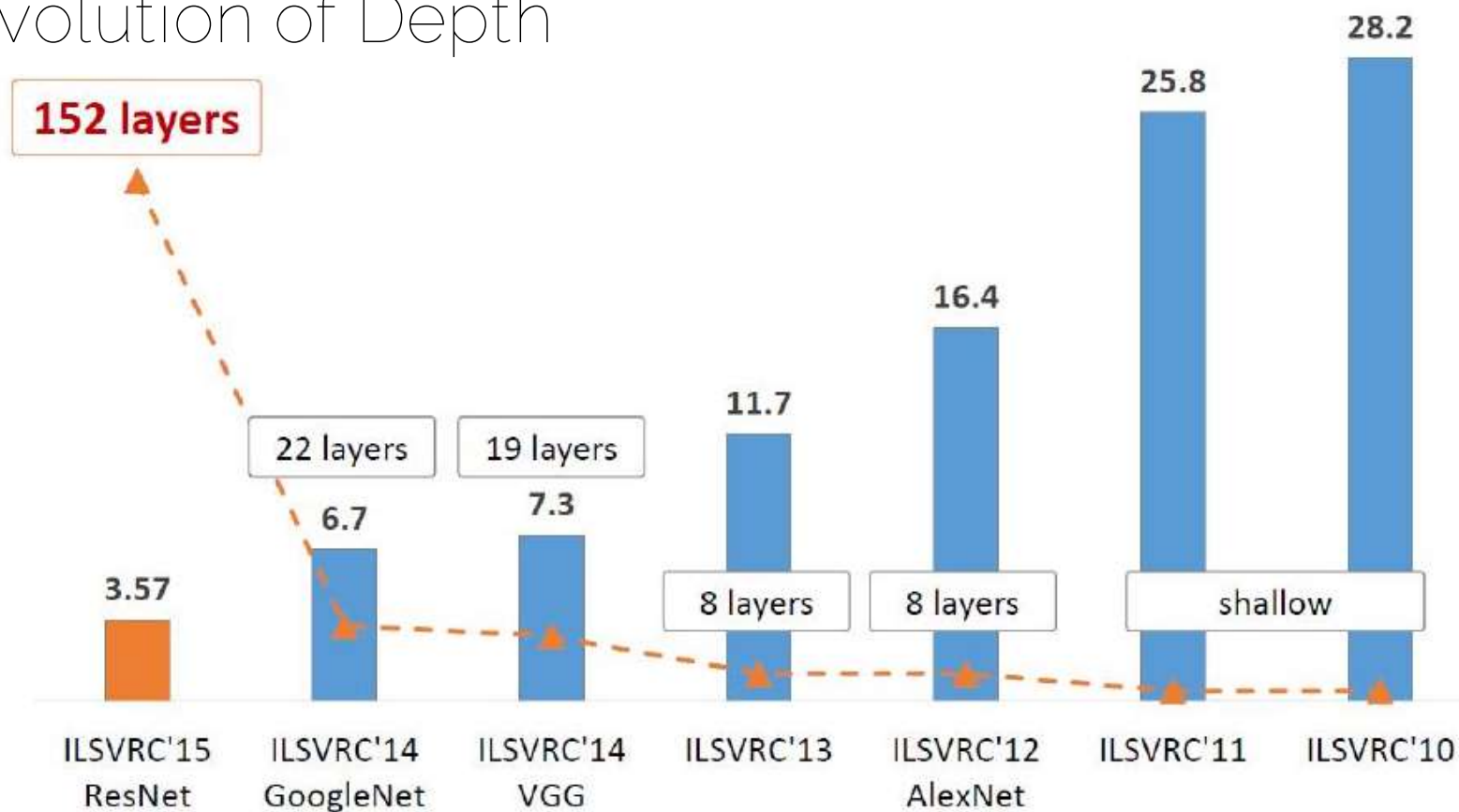
Convolutional Neural Networks



Krizhevsky, Sutskever, and Hinton, 2012

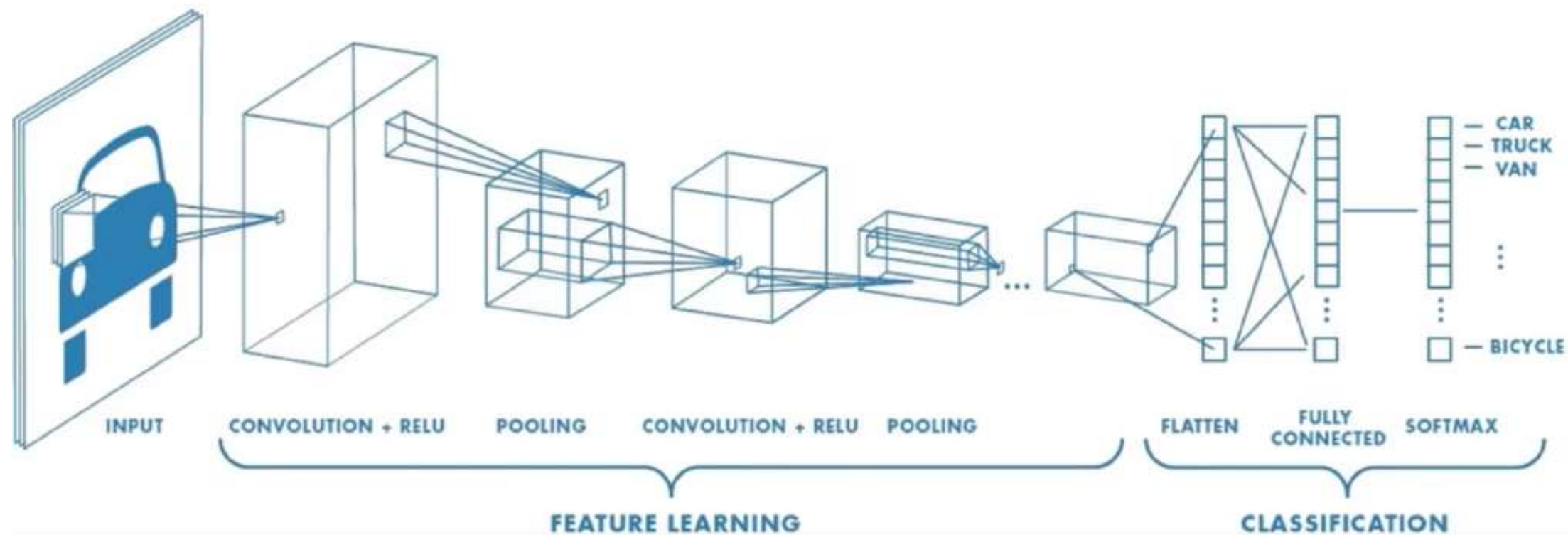
Convolutional Neural Networks

Revolution of Depth



ImageNet Classification top-5 error (%)

Convolutional Neural Networks



Convolutional Neural Networks

Convolution operation: a collection of digital filters (2D)

They convert images into feature maps

Different (convolutional) filters convert images into different feature maps

N convolutional layers by M convolutional filters
-> N by M feature maps

Convolutional Neural Networks

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

$$* \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Convolution filter

$$1 \times 1 + 1 \times 0 + 4 \times 0 + 6 \times 1 = 7$$

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

$$* \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

7	5	

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

$$* \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

7	5	9

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

$$* \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

7	5	9
4		

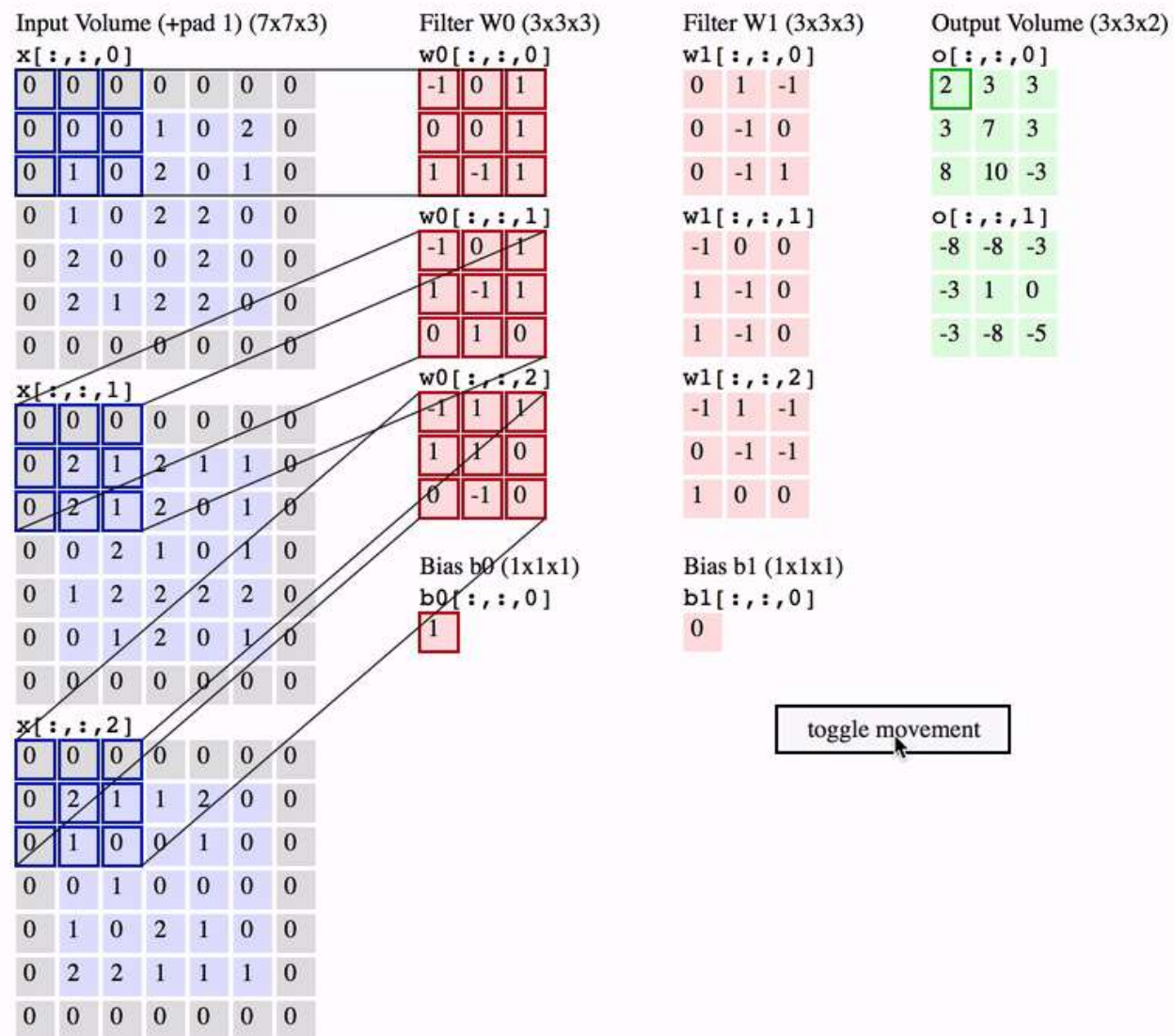


1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

$$* \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

7	5	9
4	7	9
32	2	5

Convolutional Neural Networks



Convolutional Neural Networks

Convolution filter: first example

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

 $*$

1	0
0	1

 $=$

7	5	9
4	7	9
32	2	5

High value when filter match image patch

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

 $*$

1	0
0	1

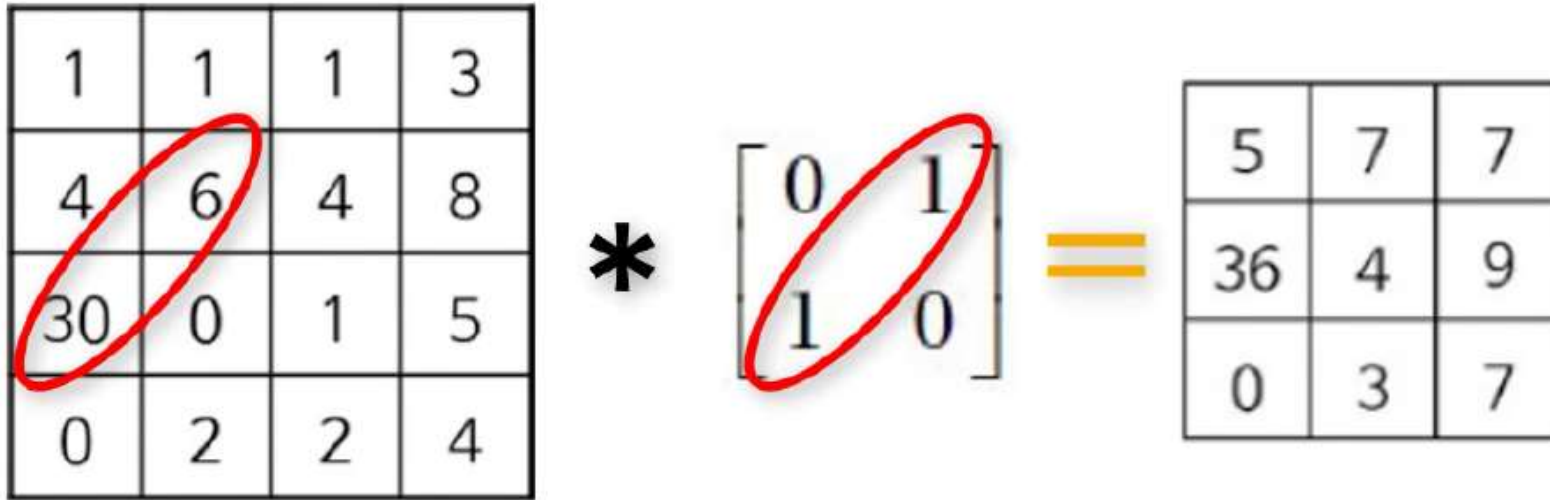
 $=$

7	5	9
4	7	9
32	2	5

Low value when filter does not match image patch


Convolutional Neural Networks

Convolution filter: second example

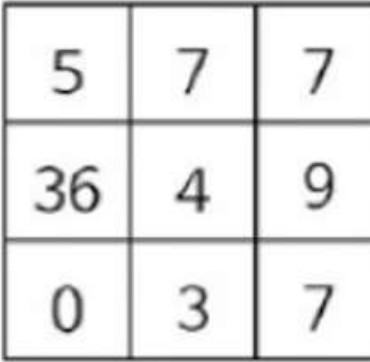


1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

*

 $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

=



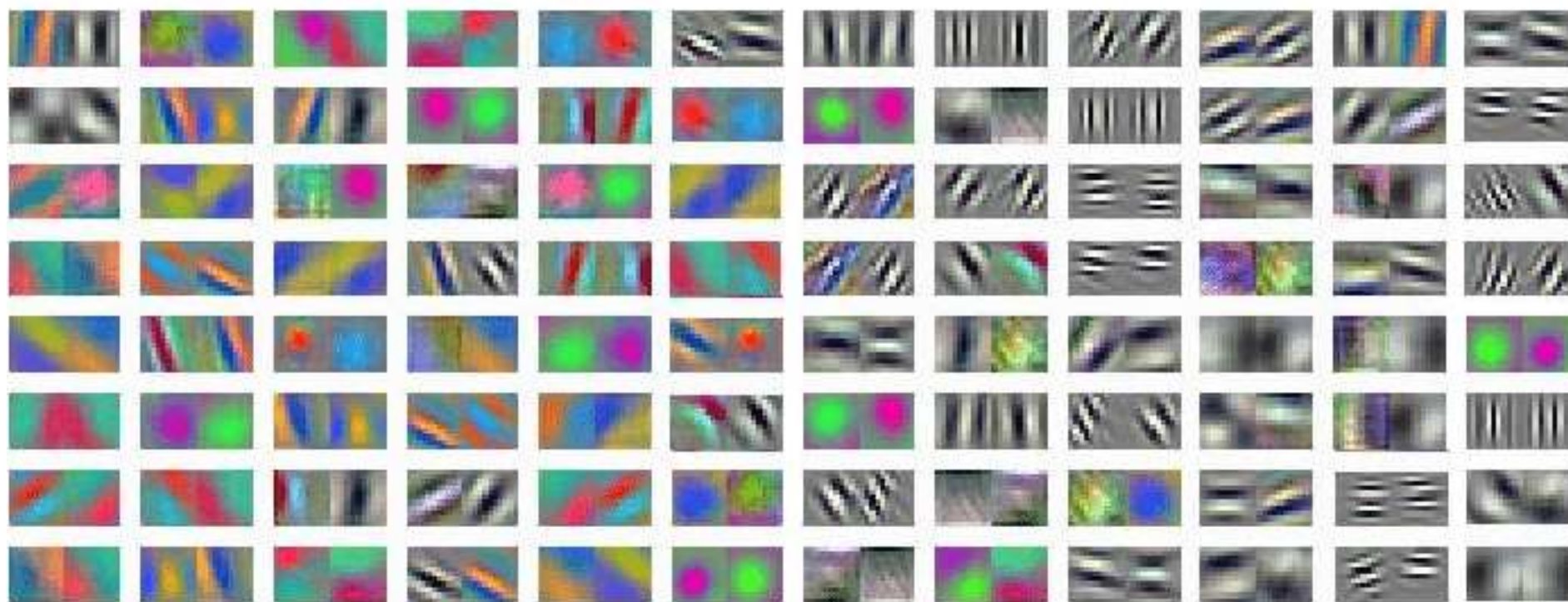
5	7	7
36	4	9
0	3	7

High value when filter match image patch

Low value when filter does not match image patch

Convolutional Neural Networks

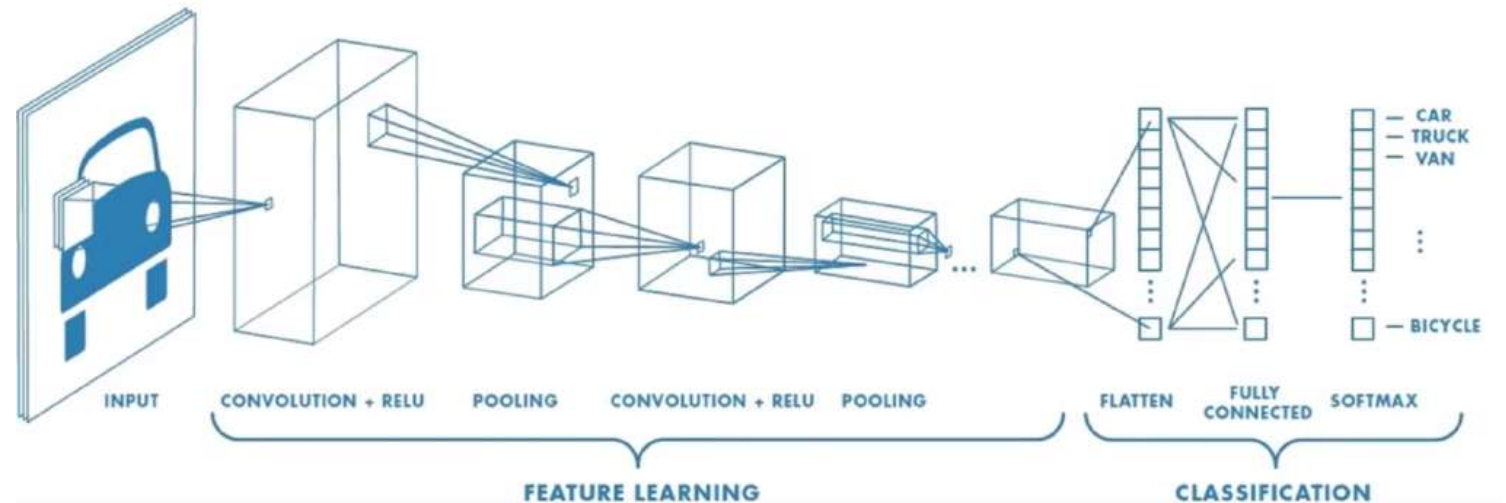
ALEXNET CONV1 LEARNED FILTERS



Convolutional Neural Networks

TYPES OF LAYERS

- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer



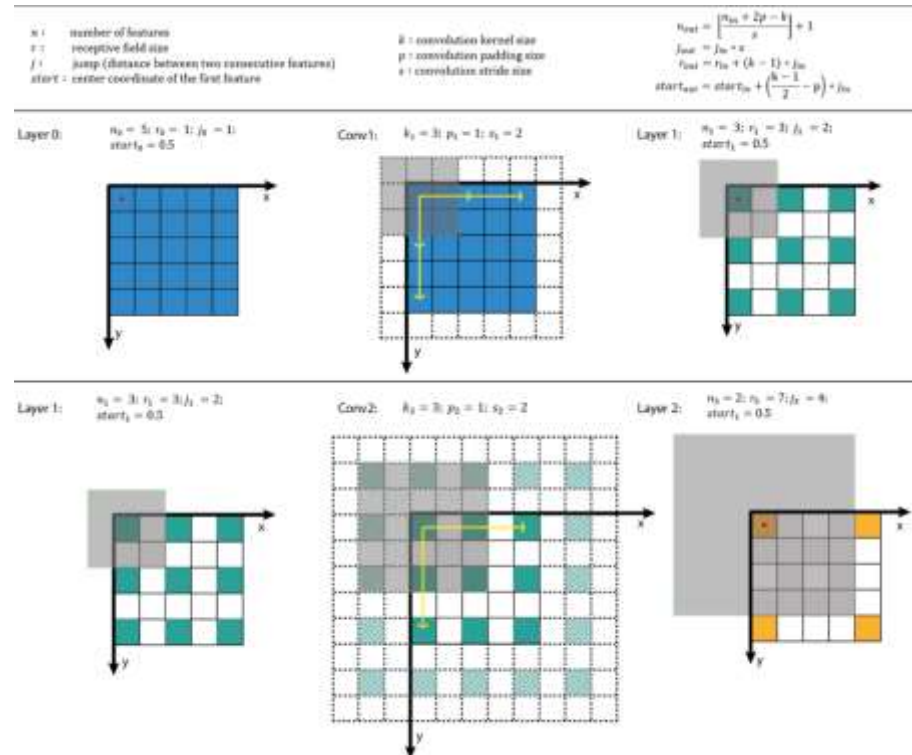
Convolutional Neural Networks

TYPES OF LAYERS

- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer

CONVOLUTION HYPERPARAMETERS

DEPTH / STRIDE / ZERO-PADDING

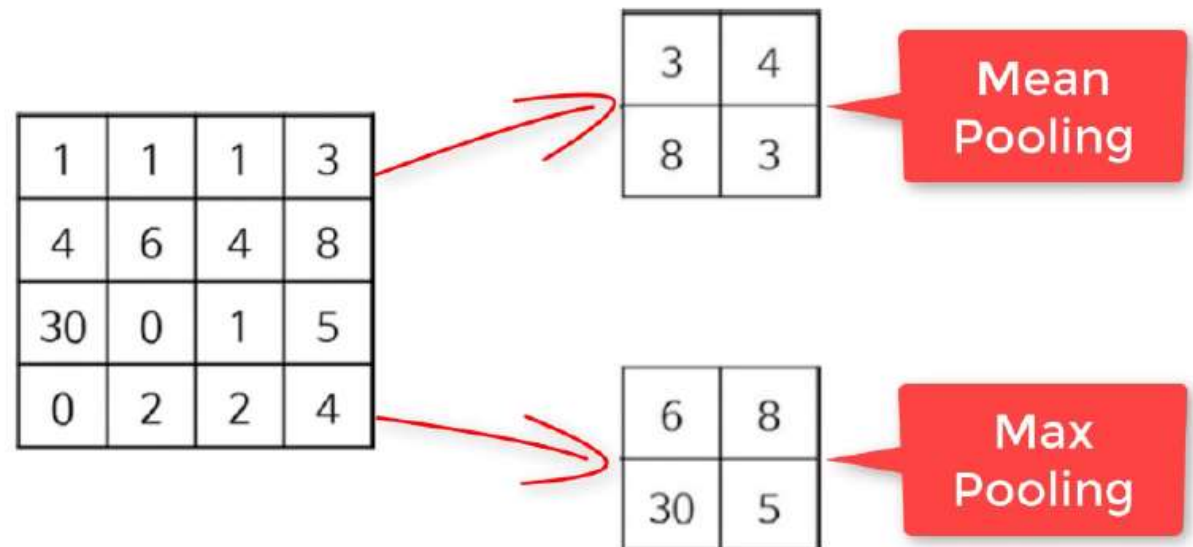


Convolutional Neural Networks

TYPES OF LAYERS

- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer

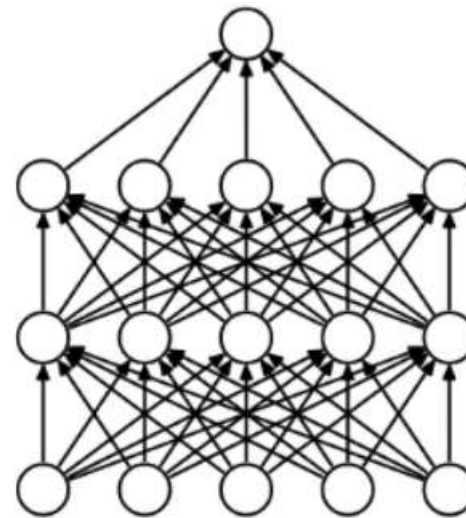
Pooling operation combines adjacent pixels into one pixel (2D) (to reduce the size of the image)



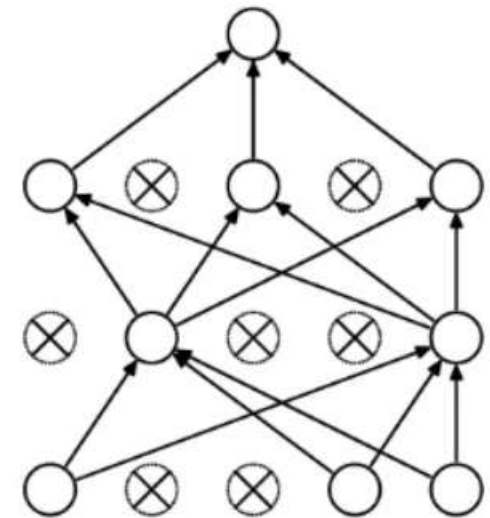
Convolutional Neural Networks

TYPES OF LAYERS

- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer



(a) Standard Neural Net



(b) After applying dropout.

Convolutional Neural Networks

TYPES OF LAYERS

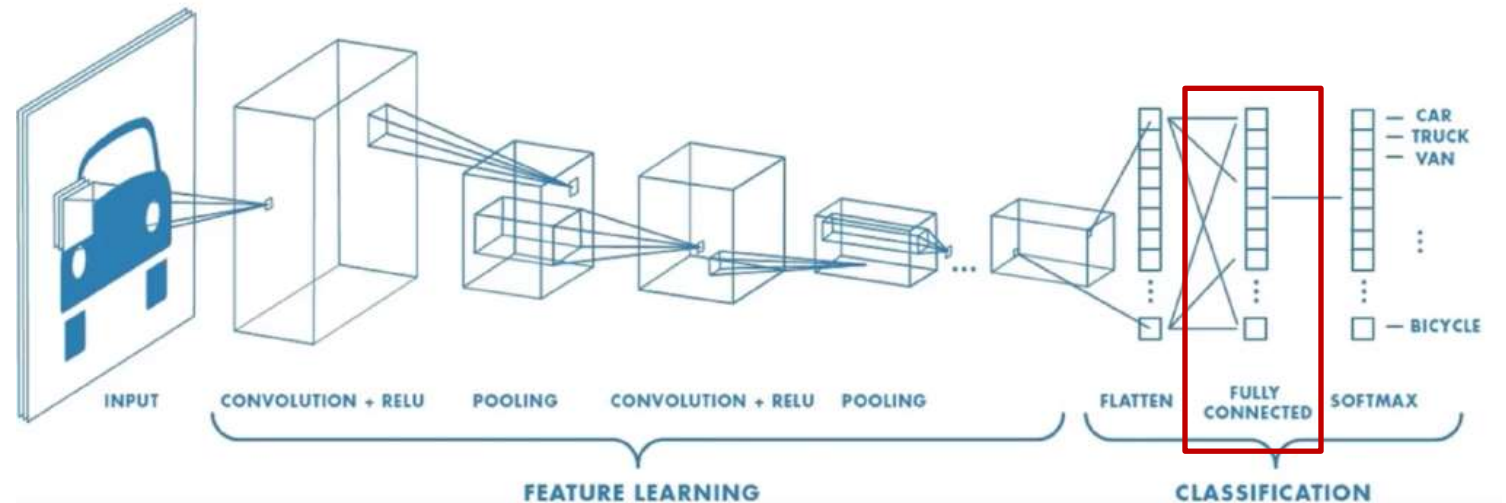
- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer

Flattening is converting the data into a 1-dimensional array for inputting it to the next **layer**

Convolutional Neural Networks

TYPES OF LAYERS

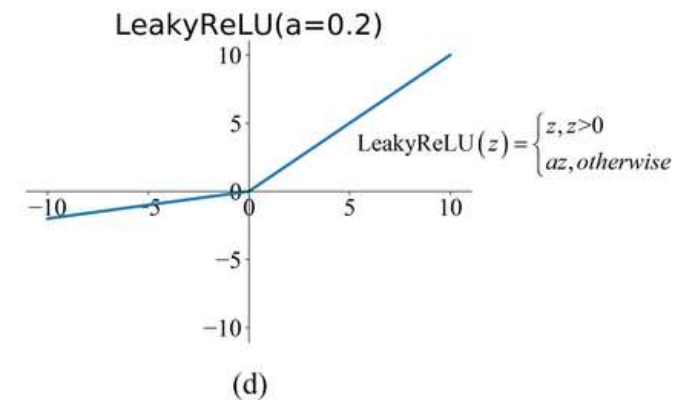
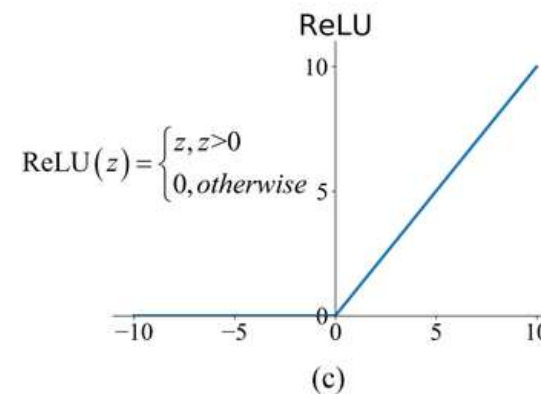
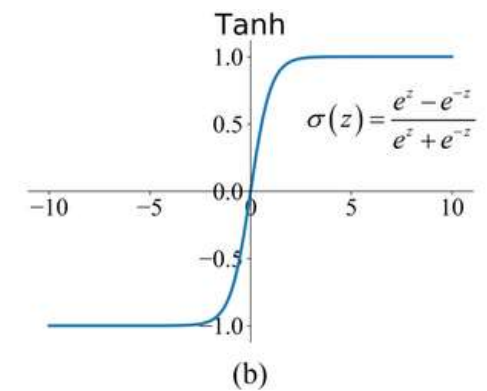
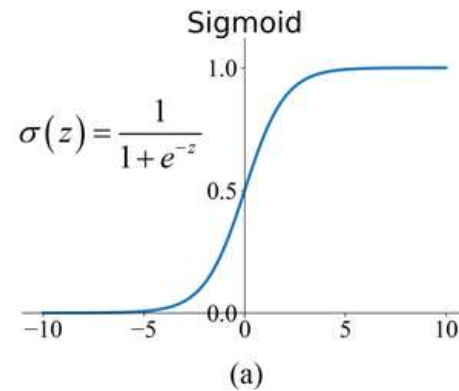
- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer



Convolutional Neural Networks

TYPES OF LAYERS

- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer



Convolutional Neural Networks

TYPES OF LAYERS

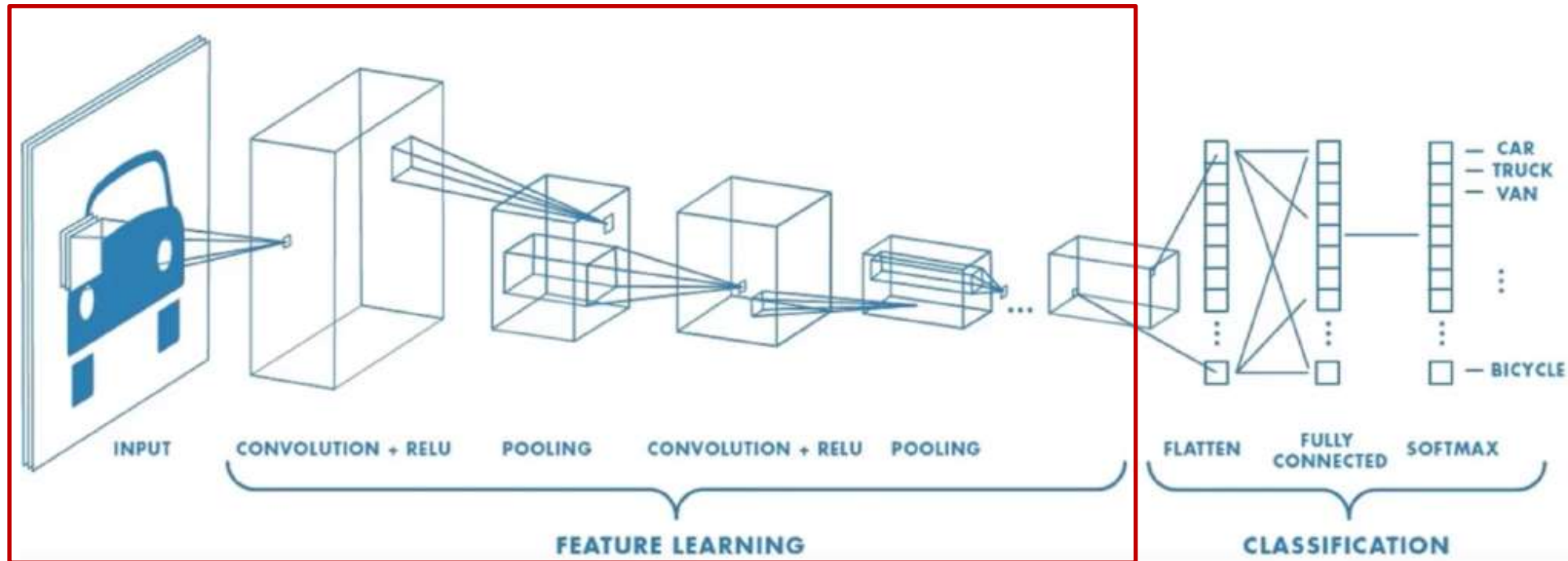
- Convolutional layer
- Pooling layer
- Dropout layer
- Flatten layer
- Fully-connected layer
- Activation layer

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Softmax

+ loss
calculation

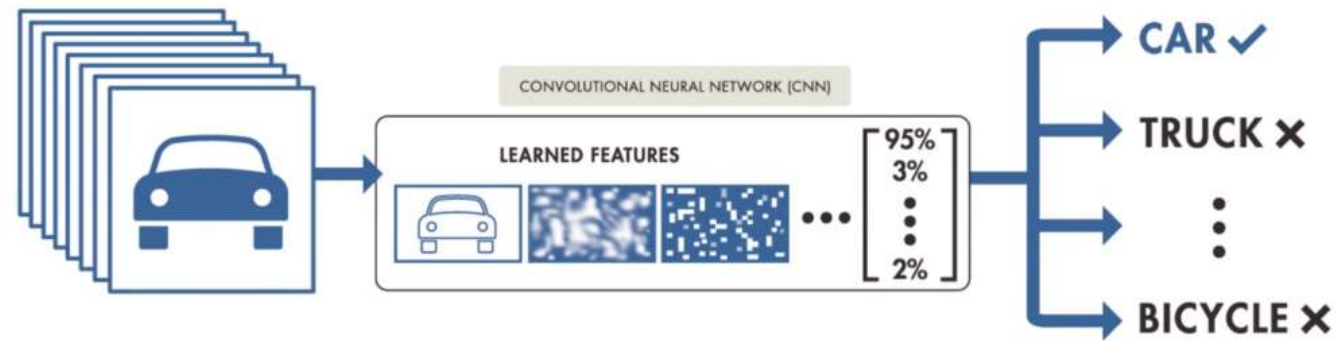
Convolutional Neural Networks as feature extractors



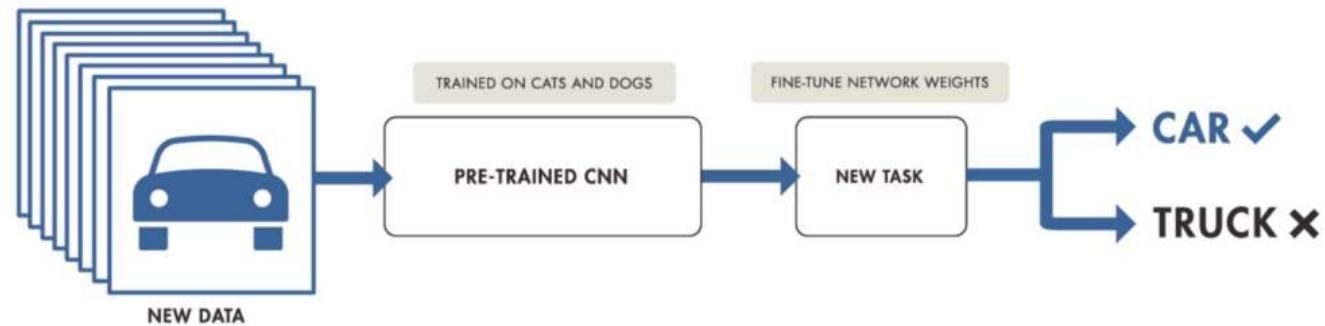
Conventional-ML
classifier

Convolutional Neural Networks | Transfer learning

TRAINING FROM SCRATCH

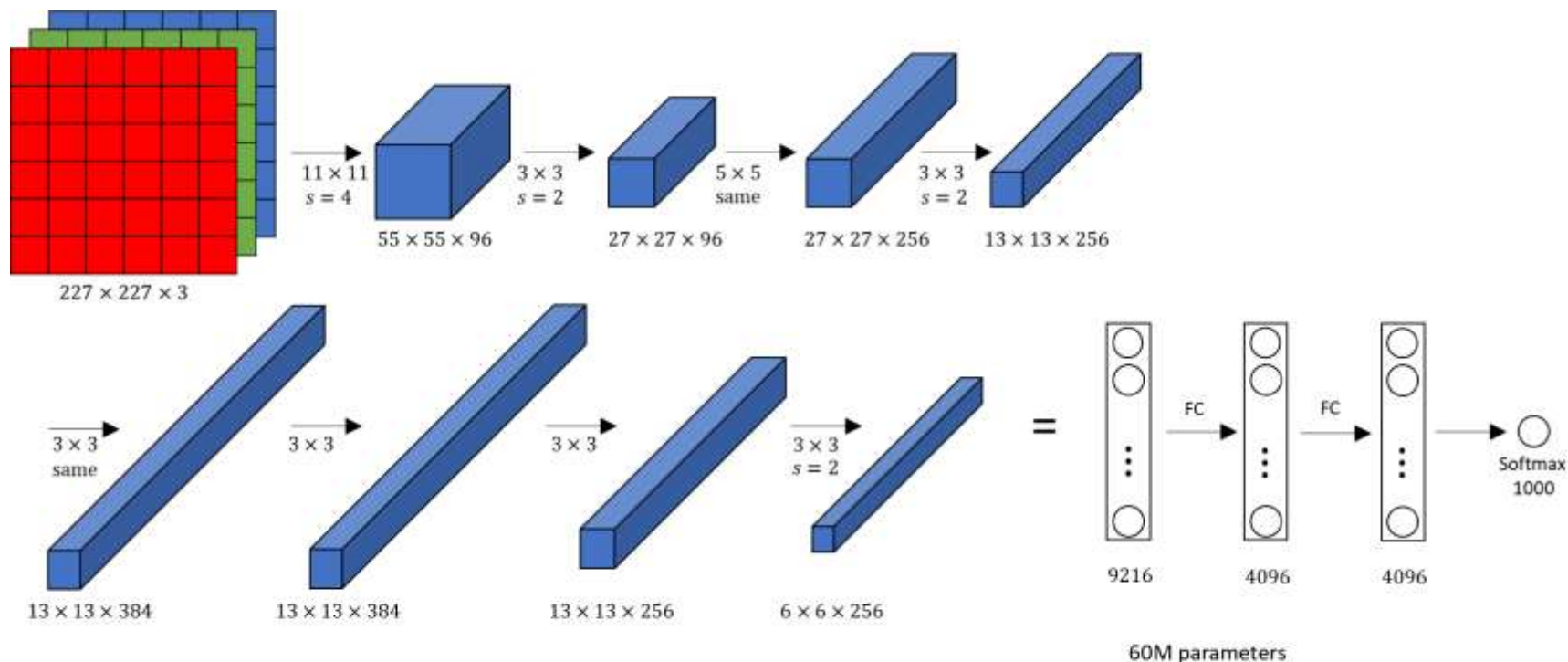


TRANSFER LEARNING

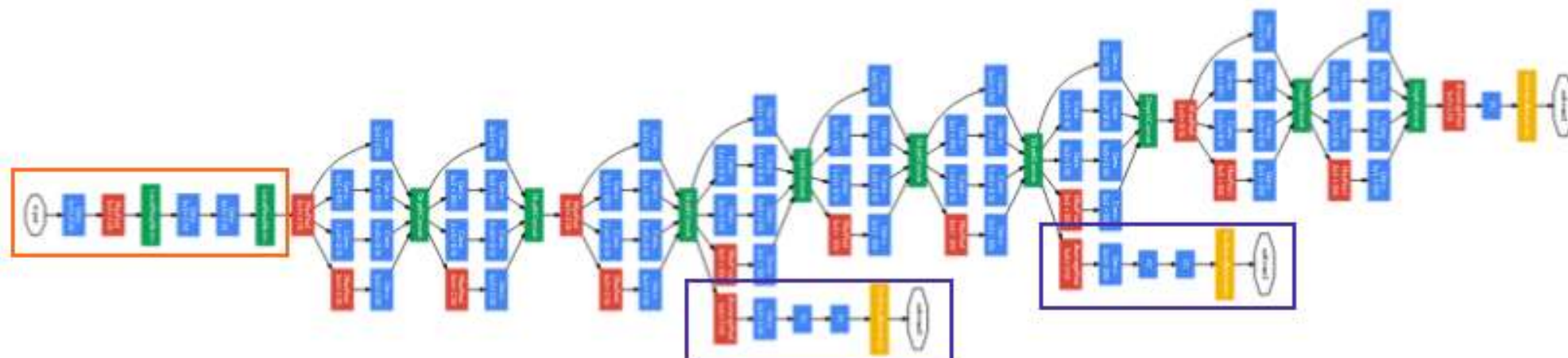


Convolutional Neural Networks | From LeNet to ...

AlexNet [winner of the ImageNet ILSVRS challenge in 2012; (17)] is composed of both stacked and connected layers and includes five convolutional layers followed by three fully-connected layers, with max-pooling layers in between. A rectified linear unit nonlinearity is applied to each convolutional layer along with a fully-connected layer to enable faster training.

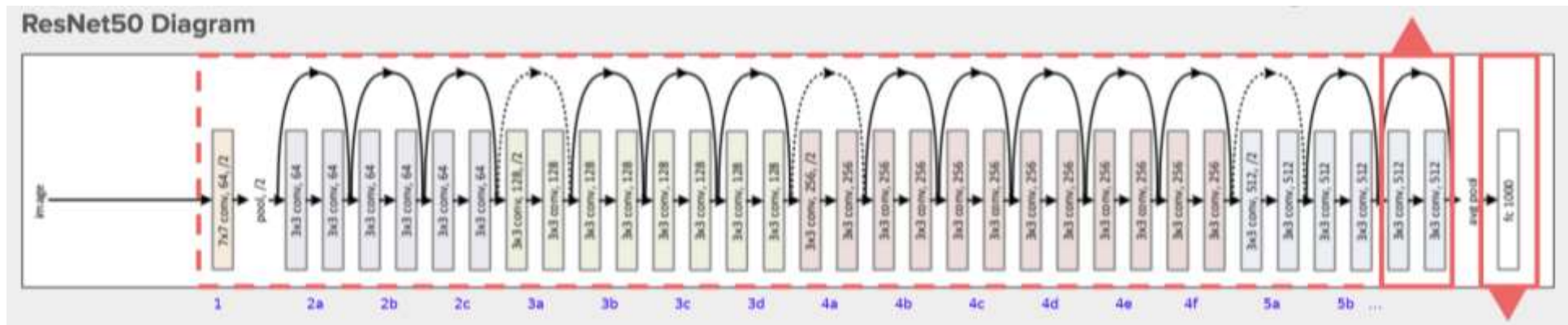
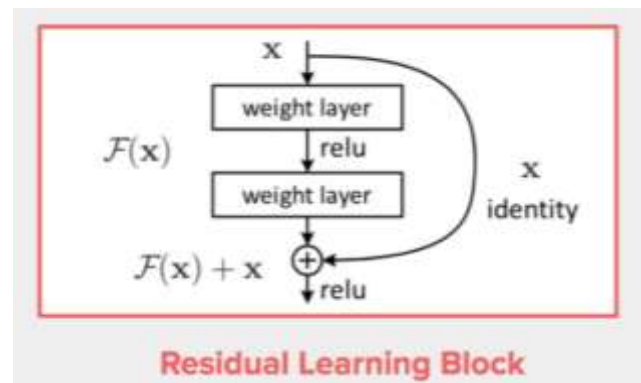


GoogleNet [winner of the ImageNet ILSVRS challenge in 2014; (41)] is the deep-learning algorithm whose design introduced the so-called Inception module, a subnetwork consisting of parallel convolutional filters whose outputs are concatenated. Inception greatly reduces the number of required parameters. GoogleNet is composed by 22 layers that require training (for a total of 27 layers when including the pooling layers).



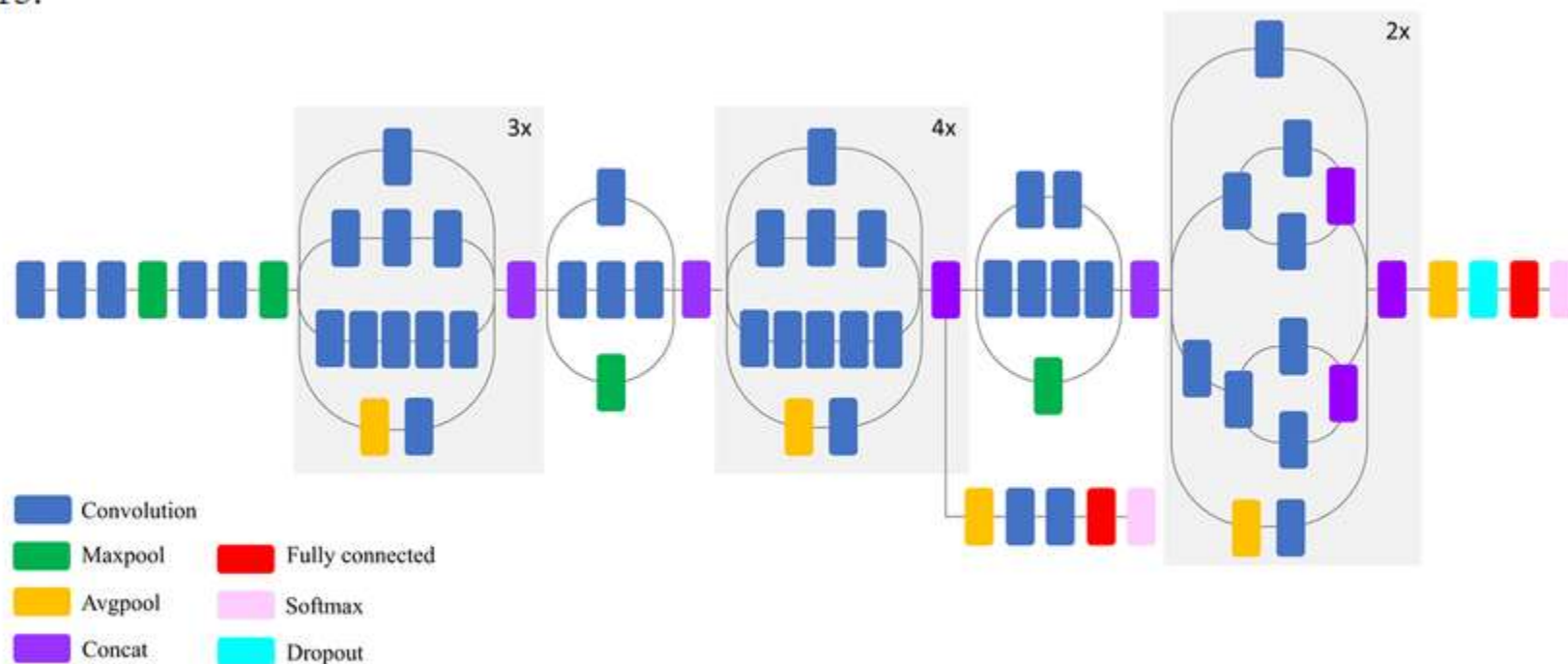
Convolutional Neural Networks | From LeNet to ...

ResNet [winner of ILSVRC 2015; (42)], an architecture that is approximately twenty times deeper than AlexNet; its main novelty is the introduction of residual layers, a kind of network-in-network architecture that forms building blocks to construct the network. ResNet uses special skip connections and batch normalization, and the fully-connected layers at the end of the network are substituted by global average pooling. Instead of learning unreferenced functions, ResNet explicitly reformulates layers as learning residual functions with reference to the input layer, which makes the model smaller in size and thus easier to optimize than other architectures.

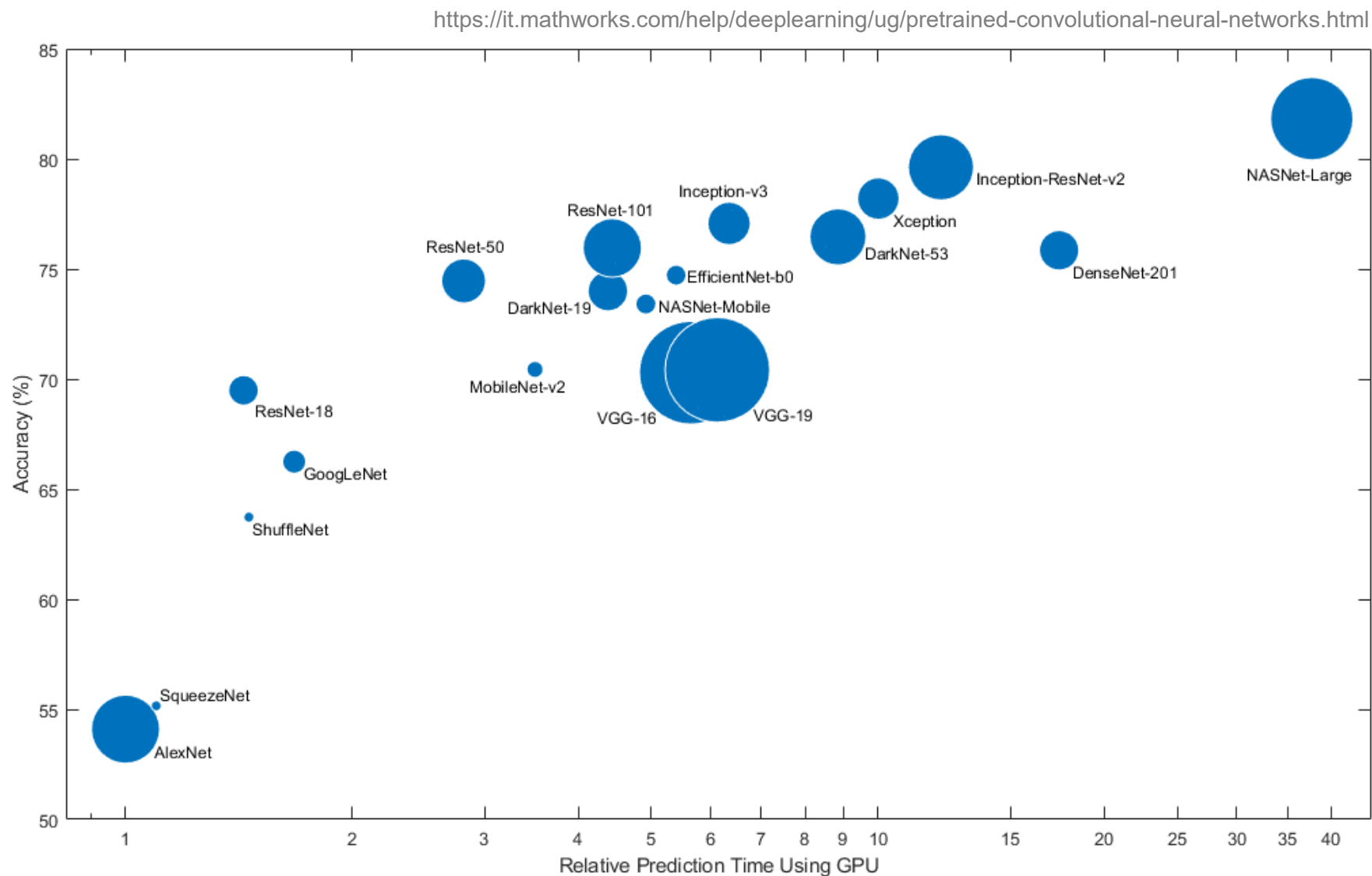


Convolutional Neural Networks | From LeNet to ...

Inception-v3 (43), a deep architecture of 48 layers able to classify images into 1,000 object categories; the net was trained on more than a million images obtained from the ImageNet database (resulting in a rich feature representation for a wide range of images). Inception-v3 classified as the first runner up for the ImageNet ILSVRC challenge in 2015.



Convolutional Neural Networks | From LeNet to ...



Convolutional Neural Networks | From LeNet to . . .

Network	Depth	Size	Parameters (Millions)	Image Input Size
squeeze net	18	5.2 MB	1.24	227-by-227
goog le net	22	27 MB	7.0	224-by-224
inception v3	48	89 MB	23.9	299-by-299
dense net201	201	77 MB	20.0	224-by-224
mobile netv2	53	13 MB	3.5	224-by-224
res net18	18	44 MB	11.7	224-by-224
res net50	50	96 MB	25.6	224-by-224
res net101	101	167 MB	44.6	224-by-224
xception	71	85 MB	22.9	299-by-299
inception resnetv2	164	209 MB	55.9	299-by-299
shuff le net	50	5.4 MB	1.4	224-by-224
nas netmobile	*	20 MB	5.3	224-by-224
nas netlarge	*	332 MB	88.9	331-by-331
dark net19	19	78 MB	20.8	256-by-256
dark net53	53	155 MB	41.6	256-by-256
efficient netb0	82	20 MB	5.3	224-by-224
alex net	8	227 MB	61.0	227-by-227
vgg 16	16	515 MB	138	224-by-224
vgg 19	19	535 MB	144	224-by-224

Convolutional Neural Networks | From LeNet to . . .

AlexNet (2012)



VGG-M (2013)



VGG-VD-16 (2014)



FROM A. VEDALDI

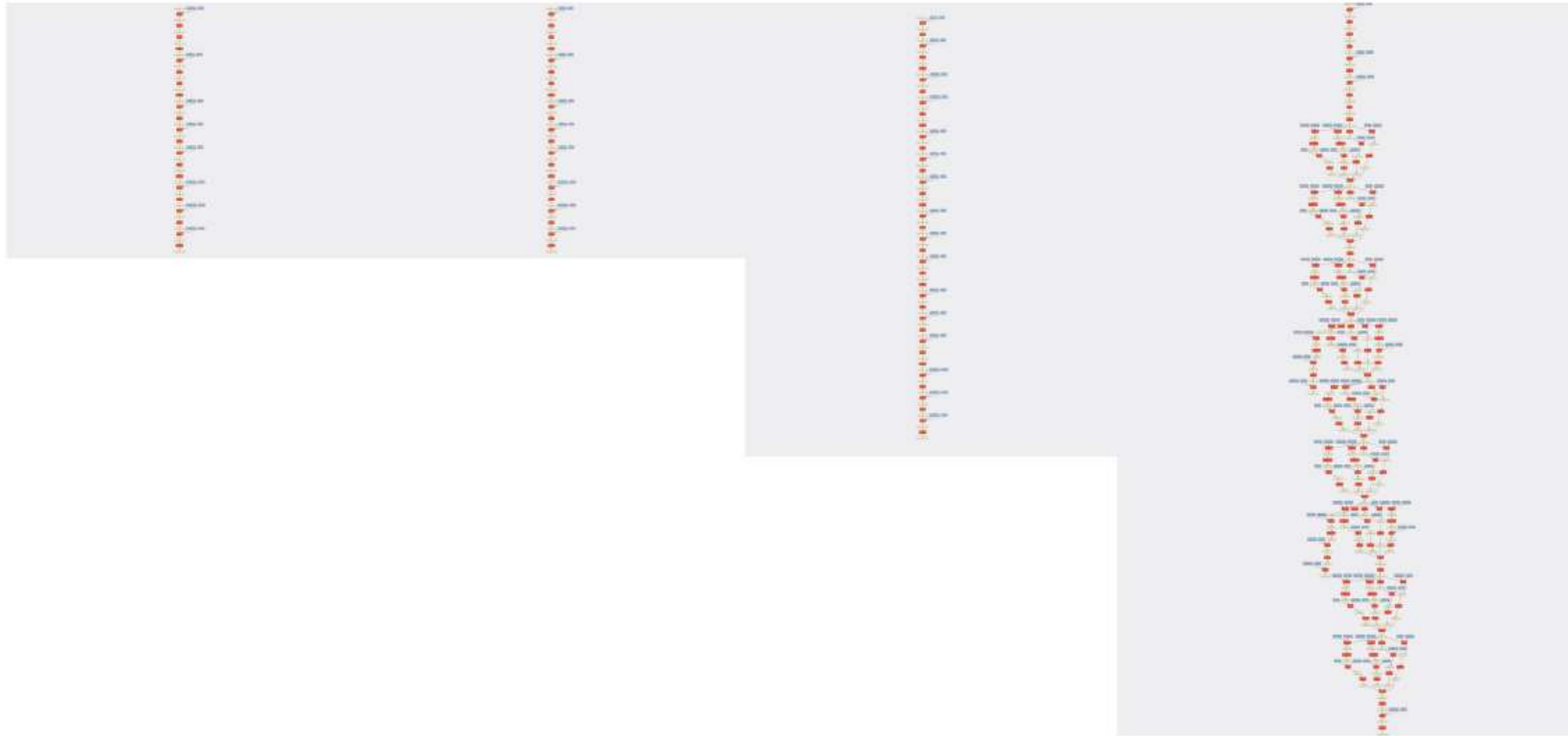
Convolutional Neural Networks | From LeNet to ...

AlexNet (2012)

VGG-M (2013)

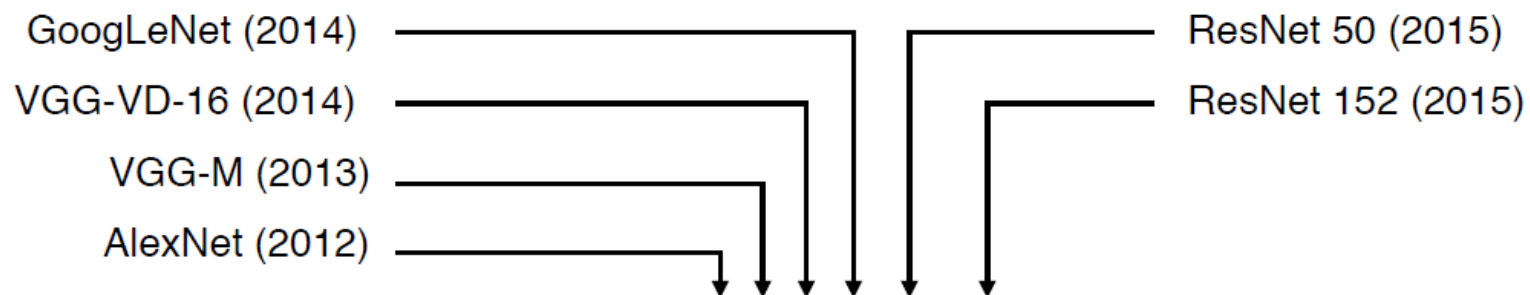
VGG-VD-16 (2014)

GoogLeNet (2014)



FROM A. VEDALDI

Convolutional Neural Networks | From LeNet to ...



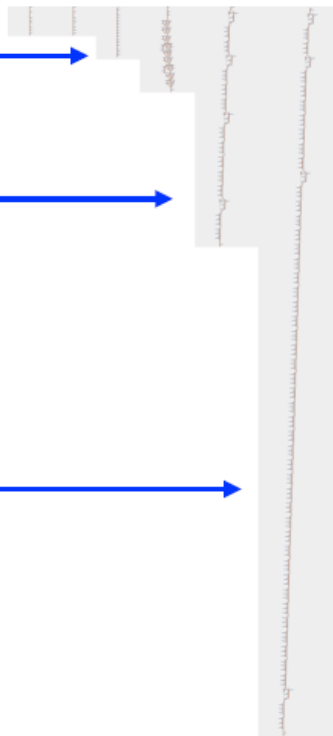
16 convolutional layers



50 convolutional layers



152 convolutional layers



Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

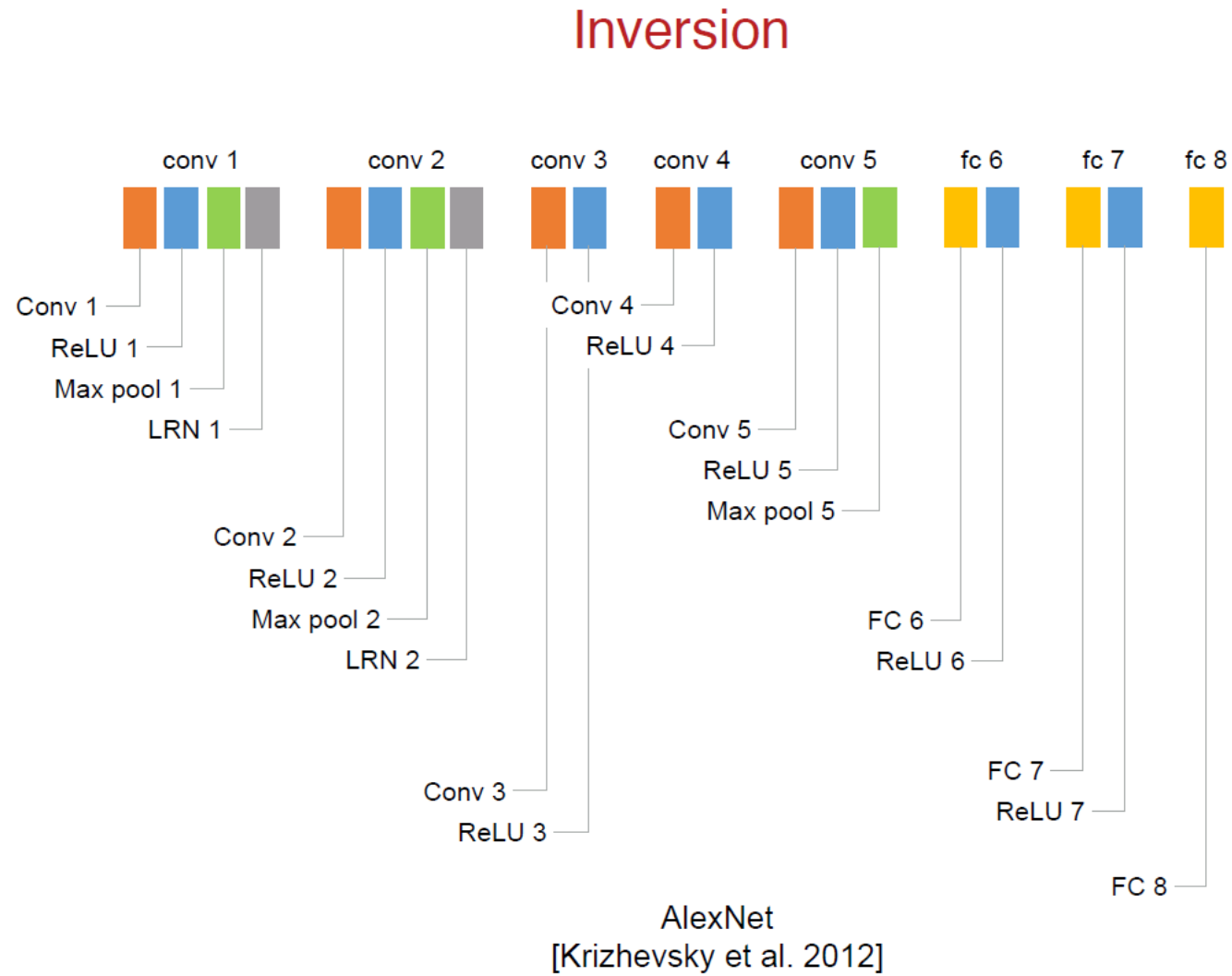
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

FROM A. VEDALDI

A CONSIDERATION ON EXPLAINABILITY AND (OVER)FITTING

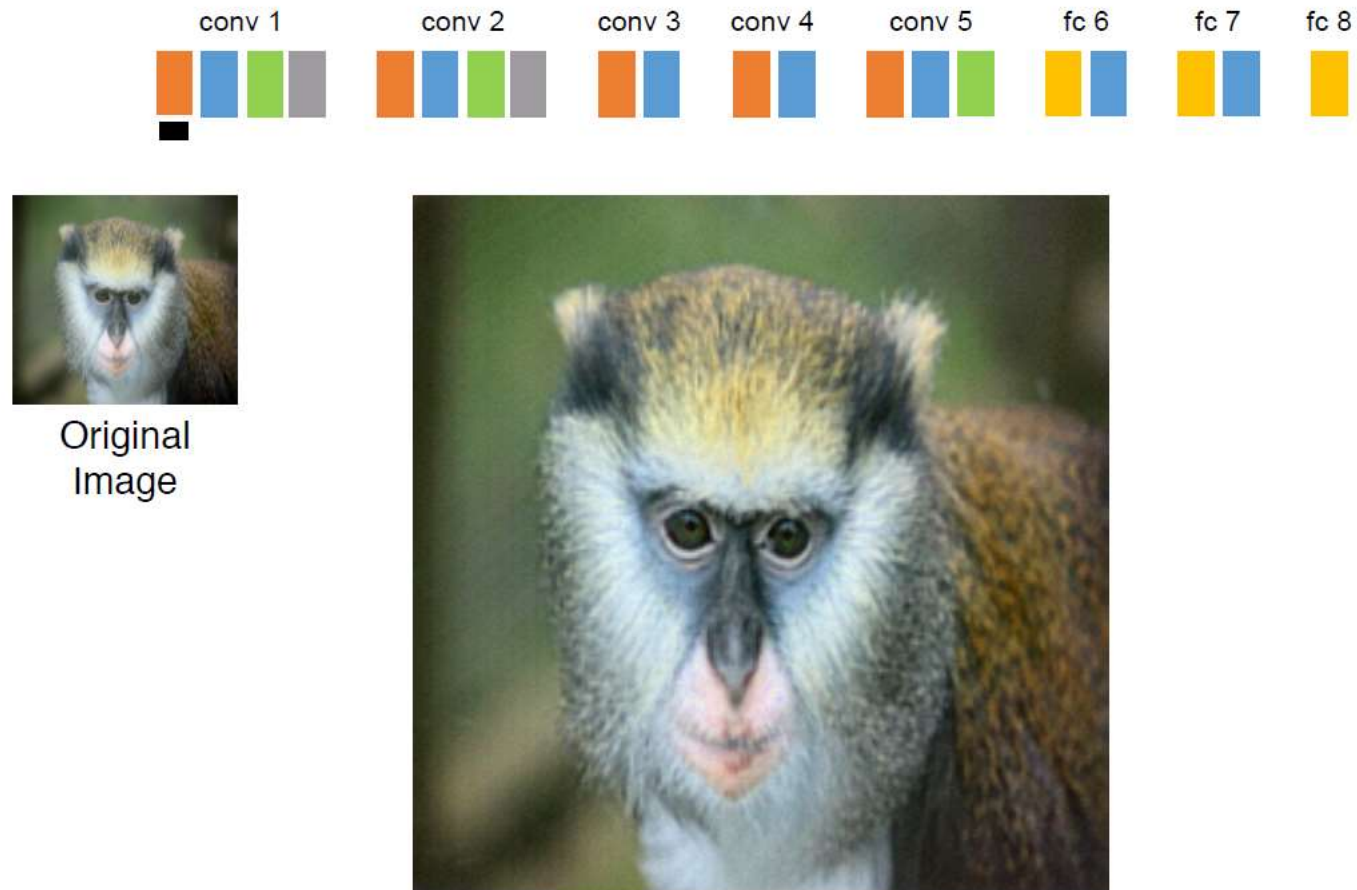
UNDERFITTING, OVERFITTING AND BEST FITTING



10

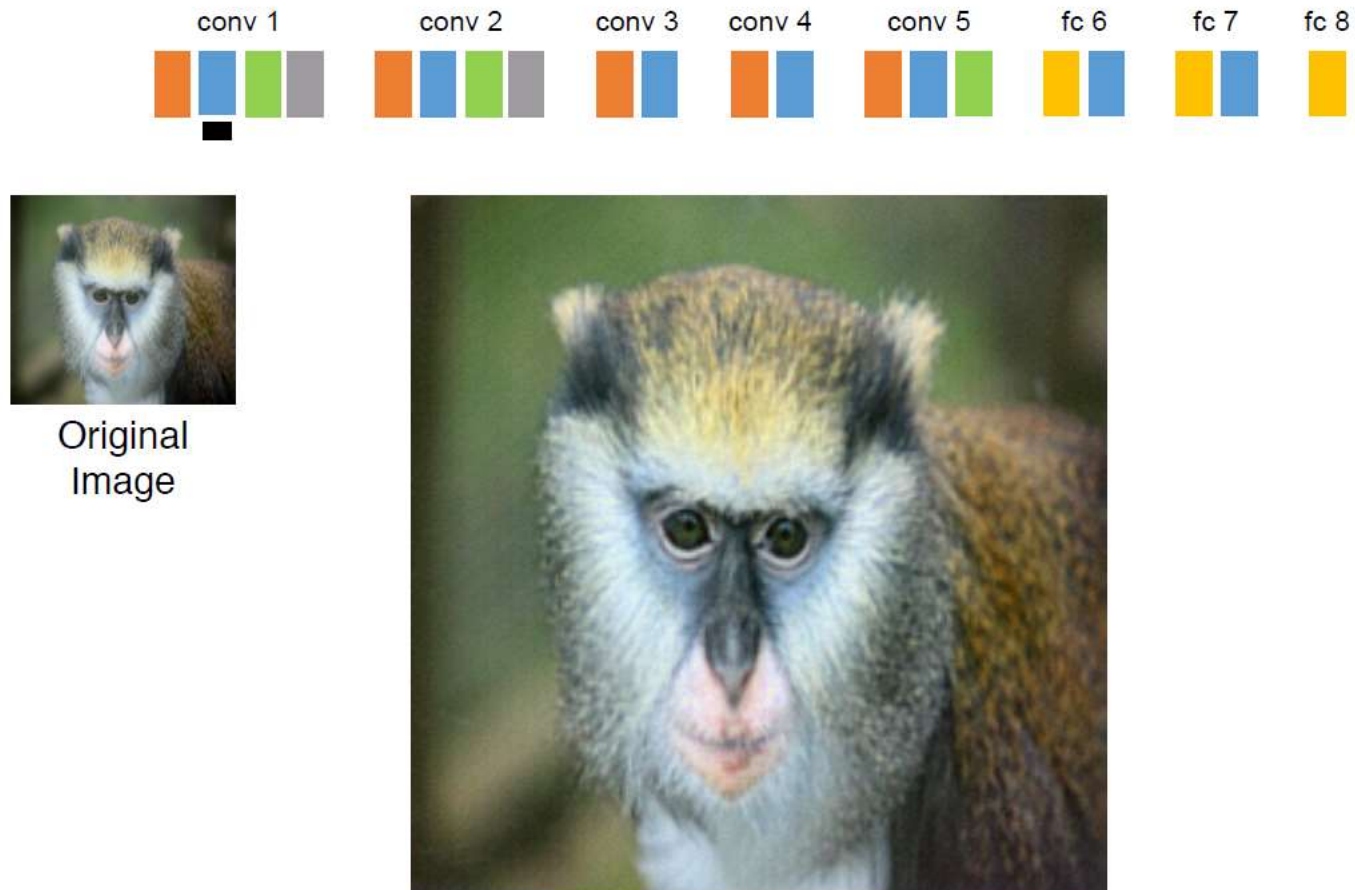
Inversion

11



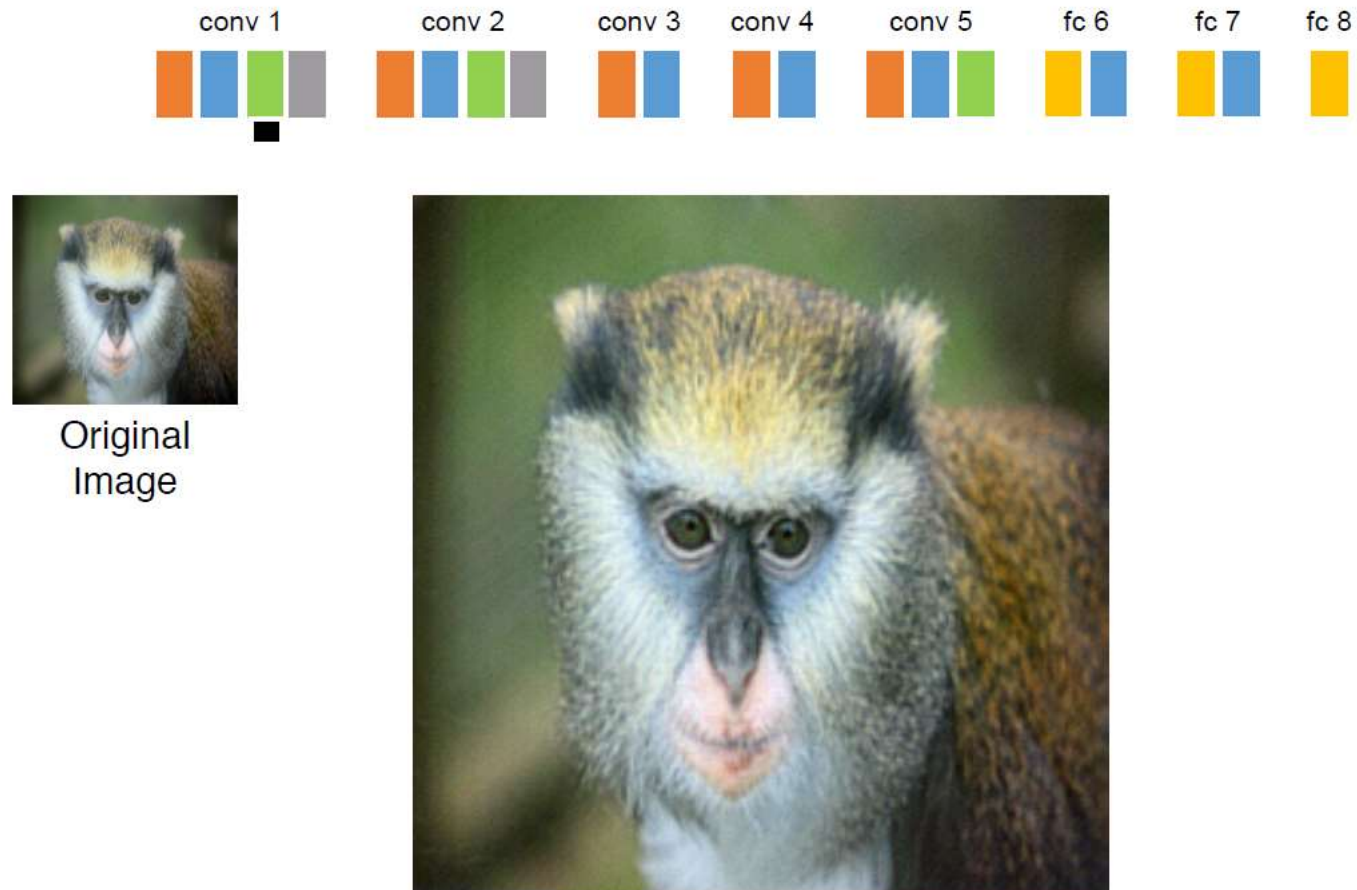
Inversion

12



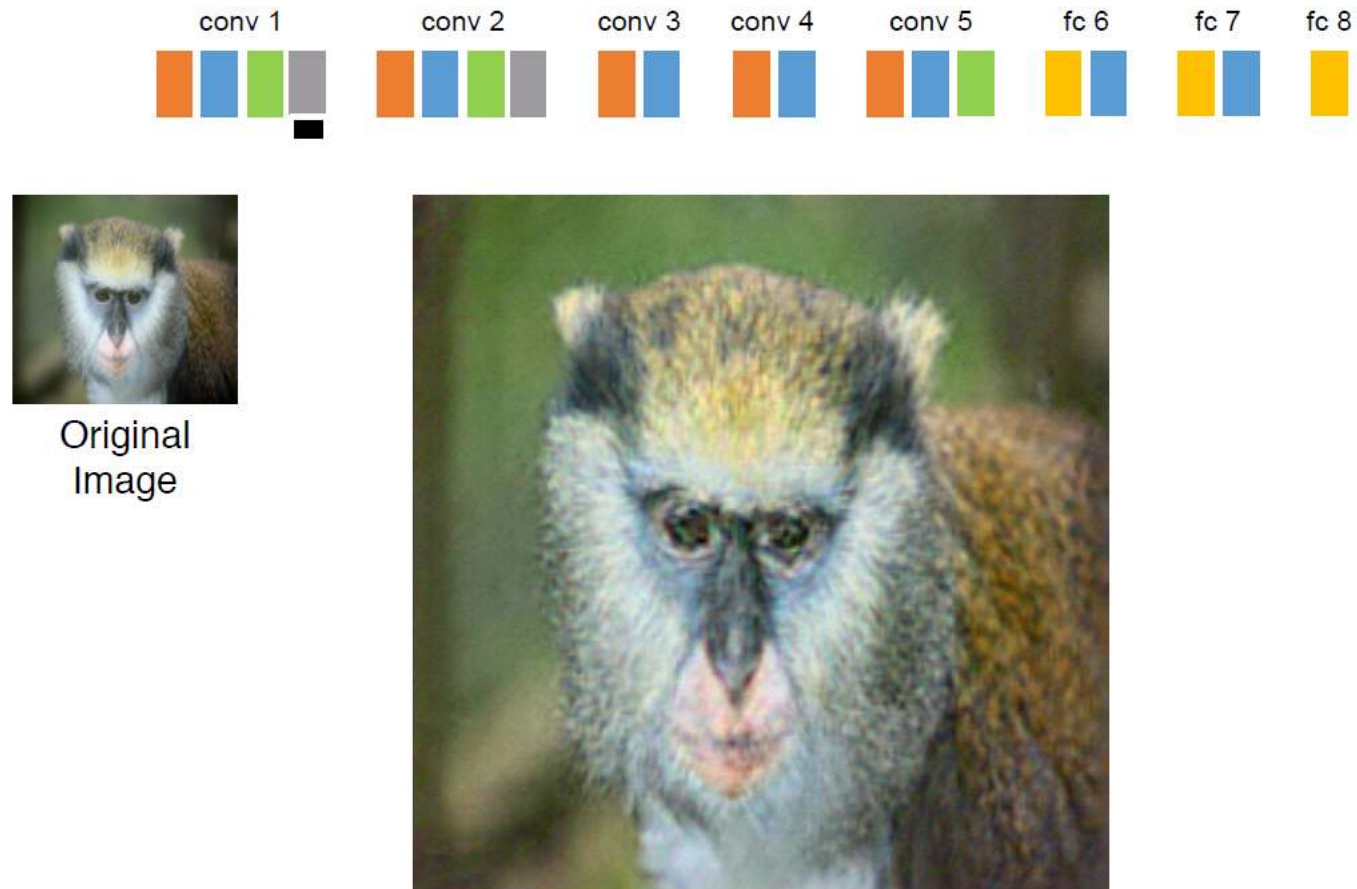
Inversion

13



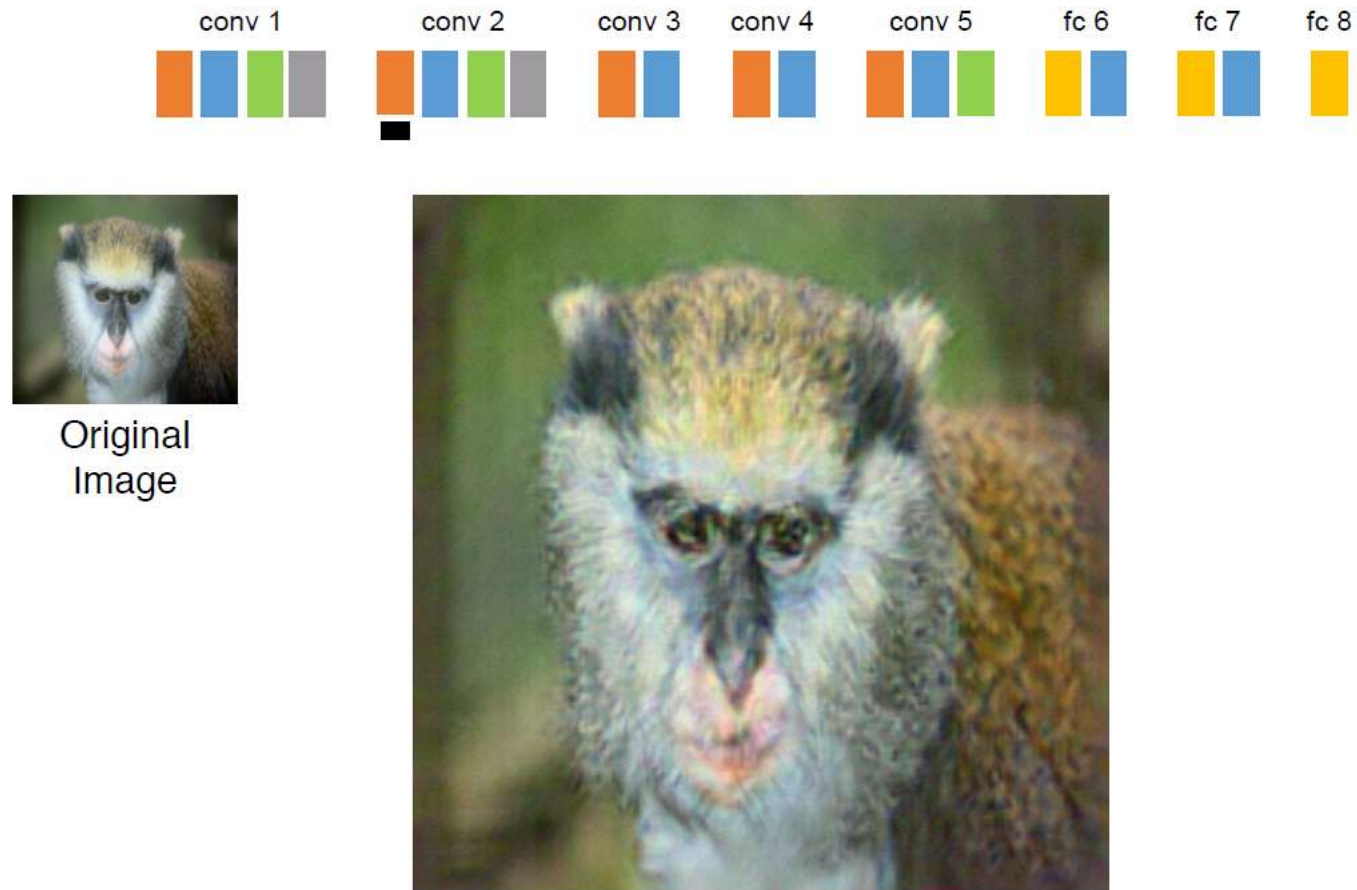
Inversion

14



Inversion

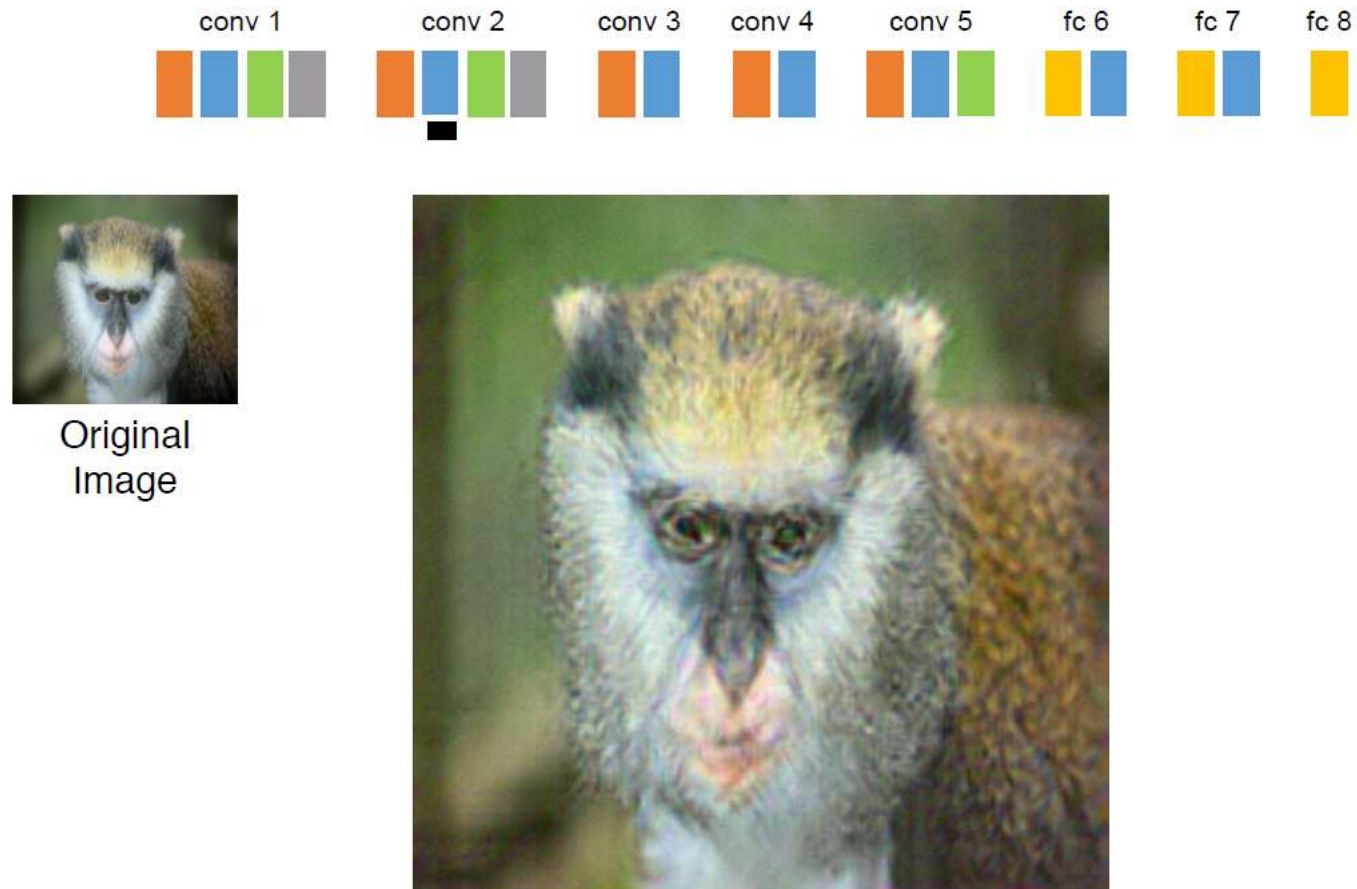
15



UNDERFITTING, OVERFITTING AND BEST FITTING

Inversion

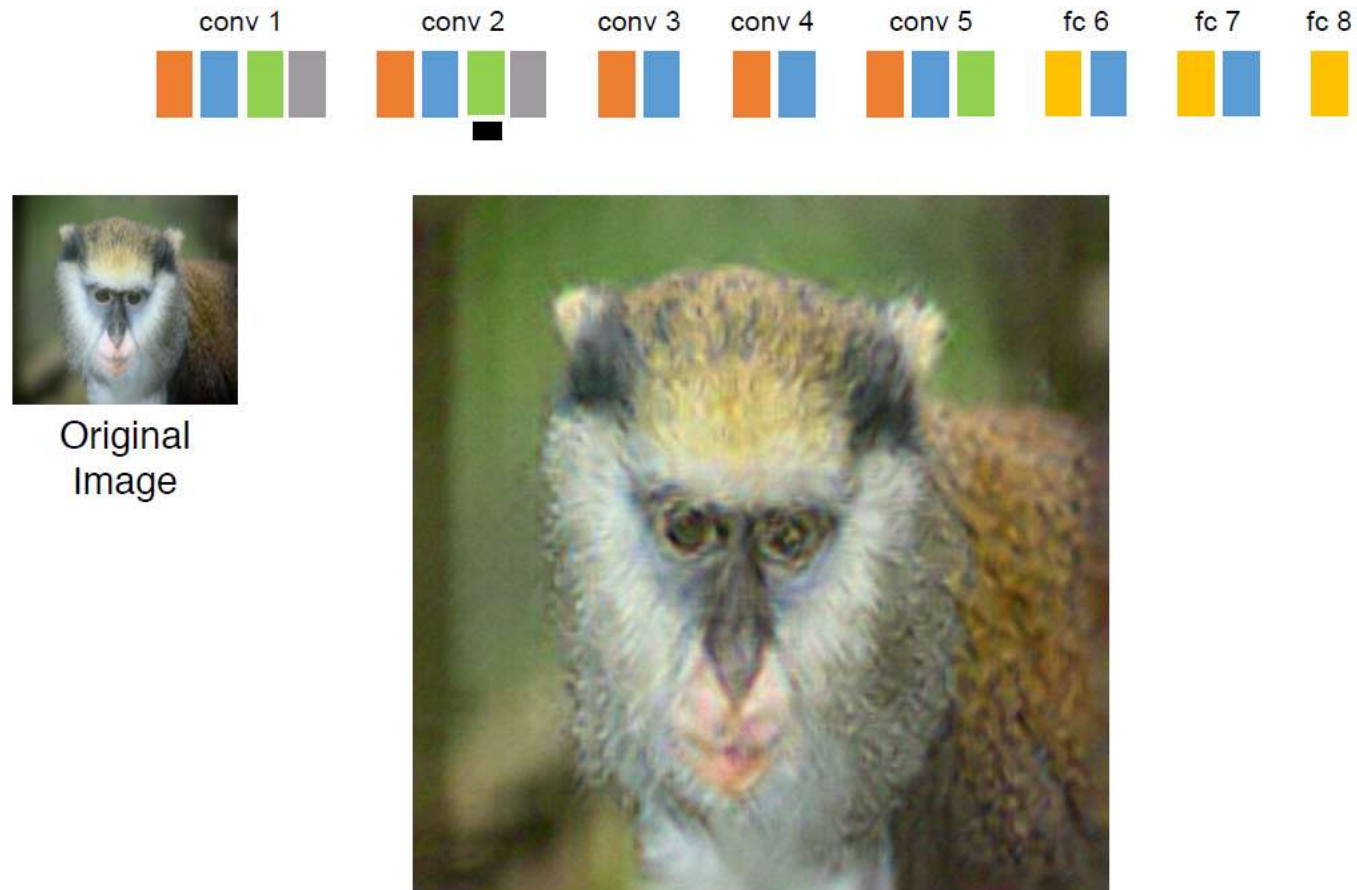
16



UNDERFITTING, OVERFITTING AND BEST FITTING

Inversion

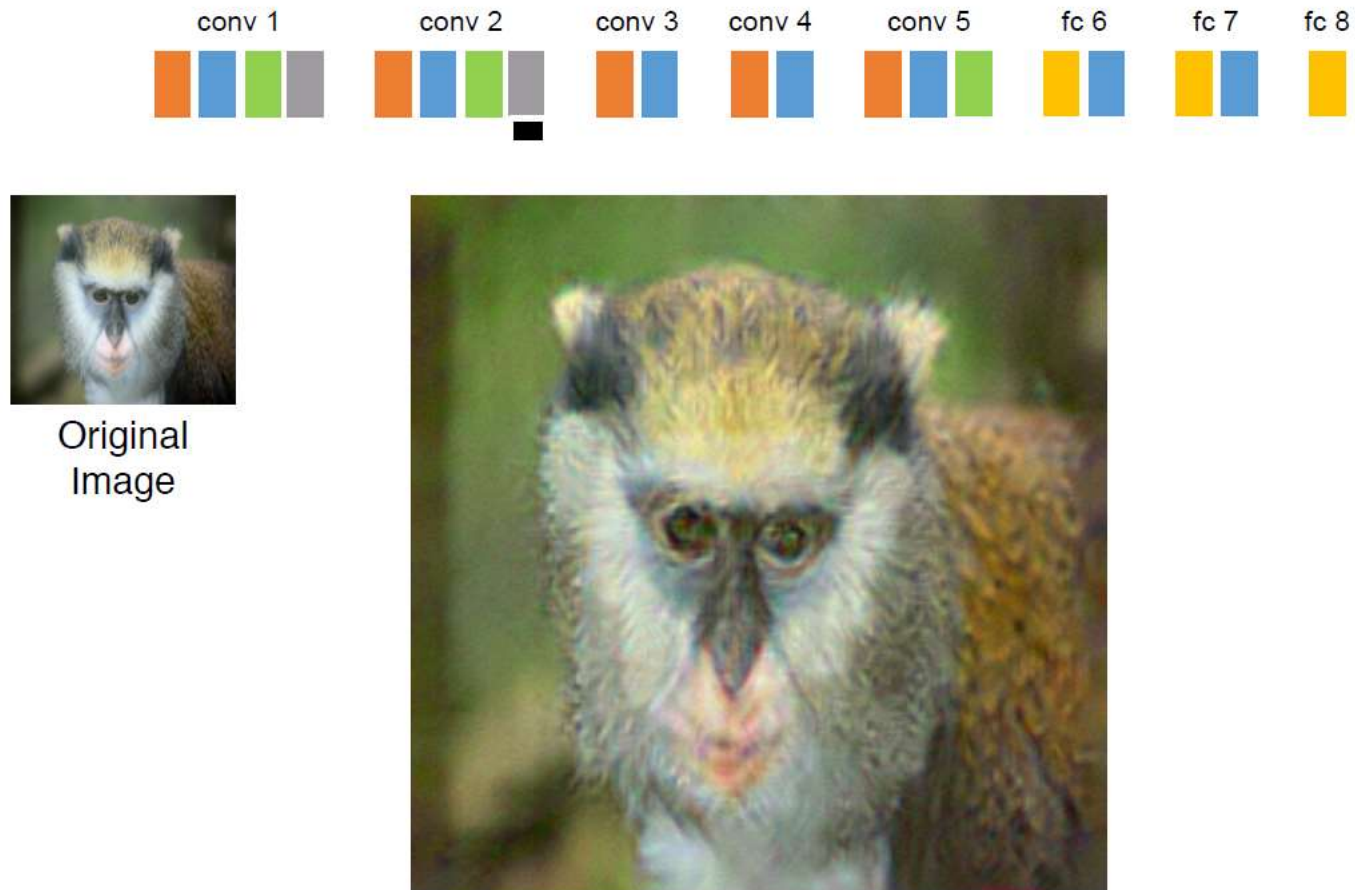
17



UNDERFITTING, OVERFITTING AND BEST FITTING

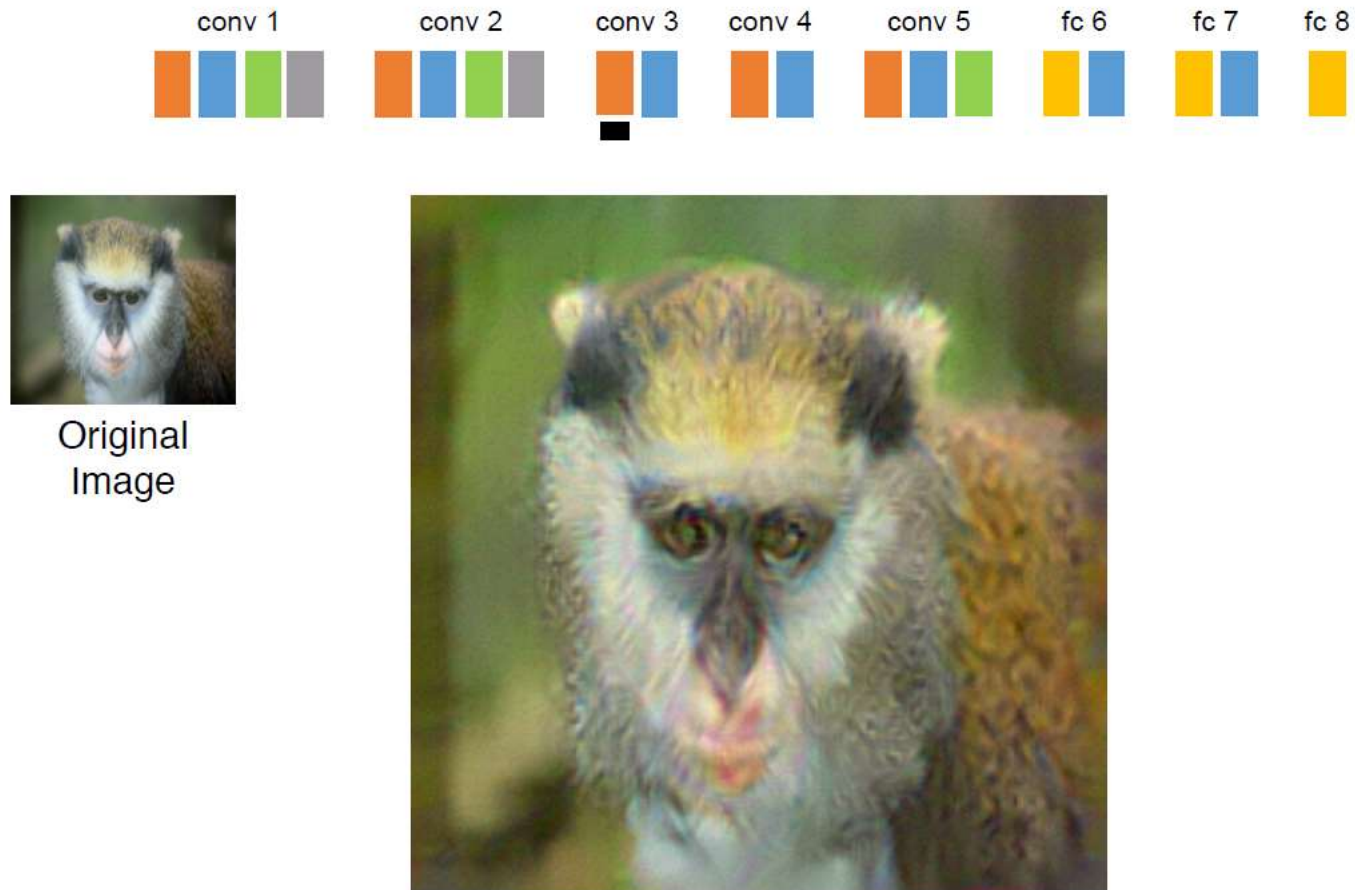
Inversion

18



Inversion

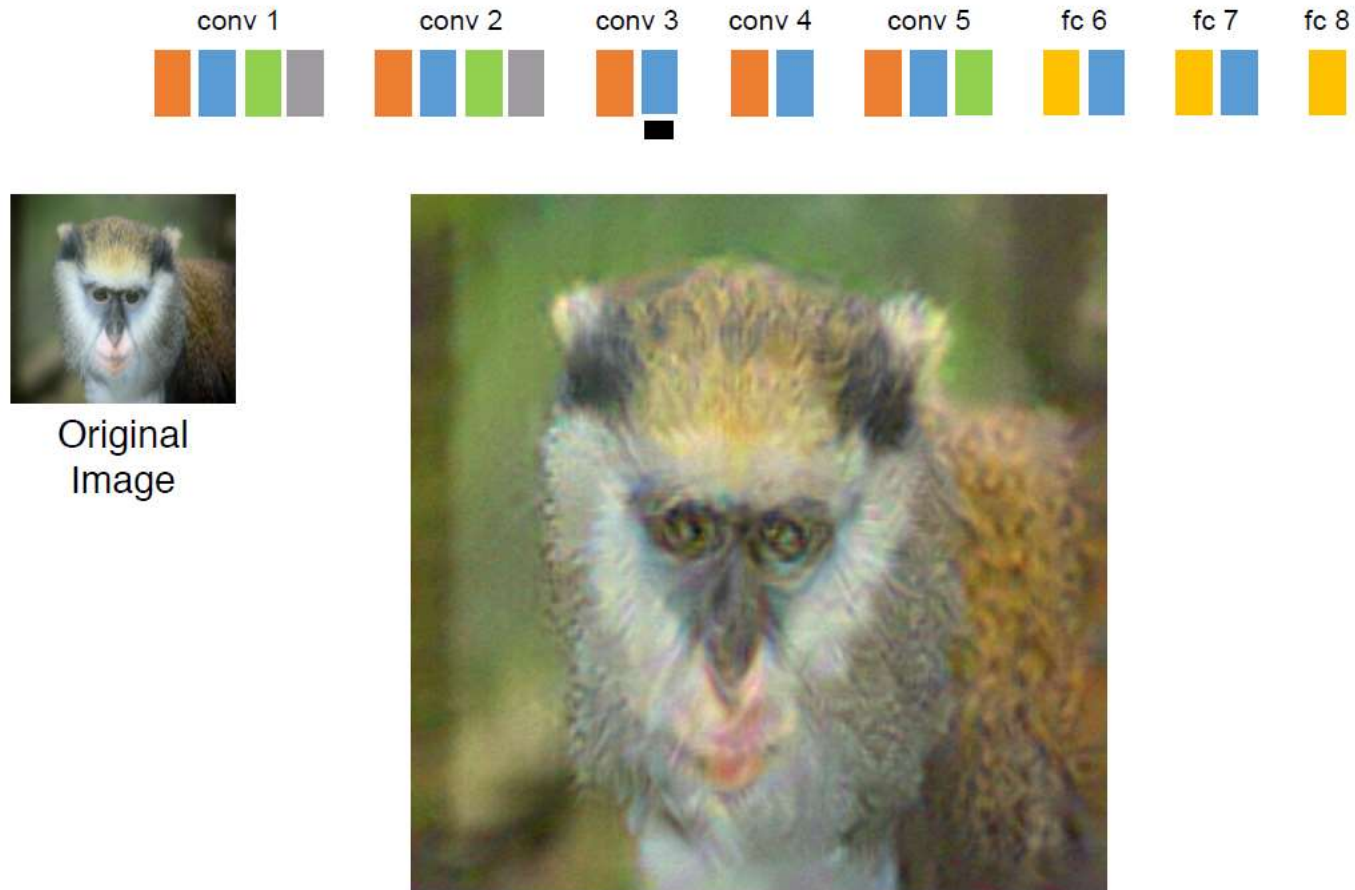
19



UNDERFITTING, OVERFITTING AND BEST FITTING

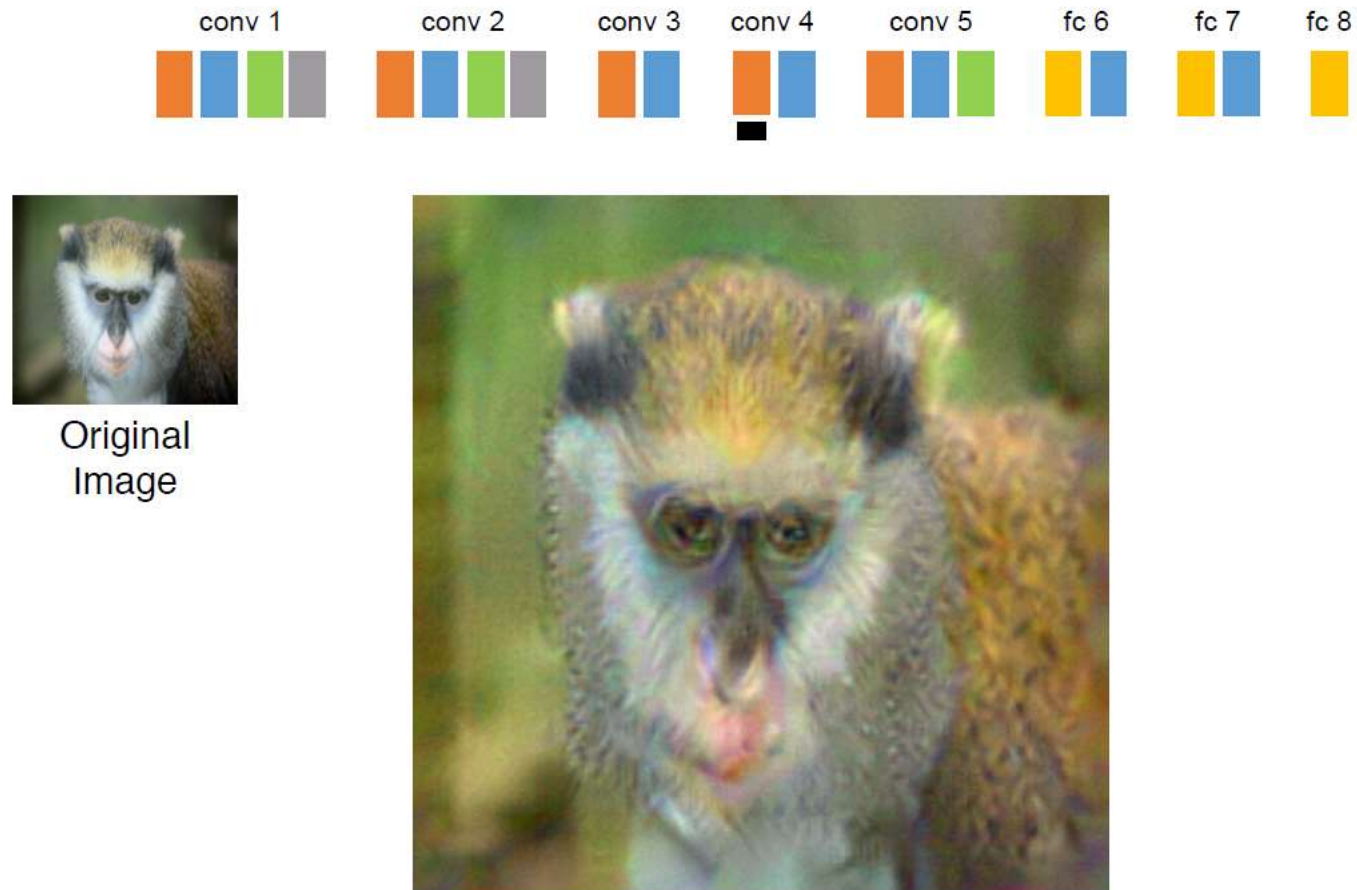
Inversion

20



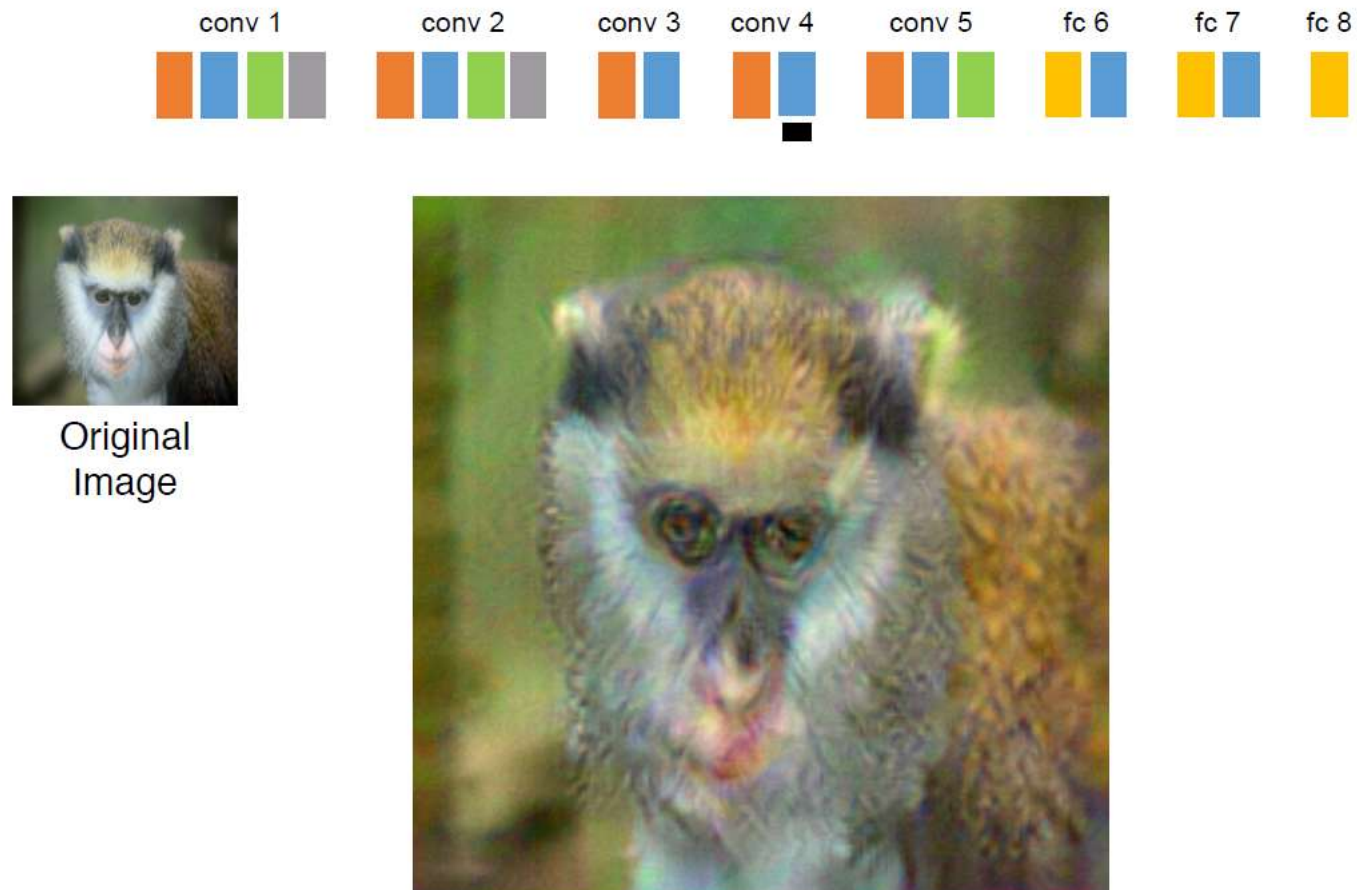
Inversion

21



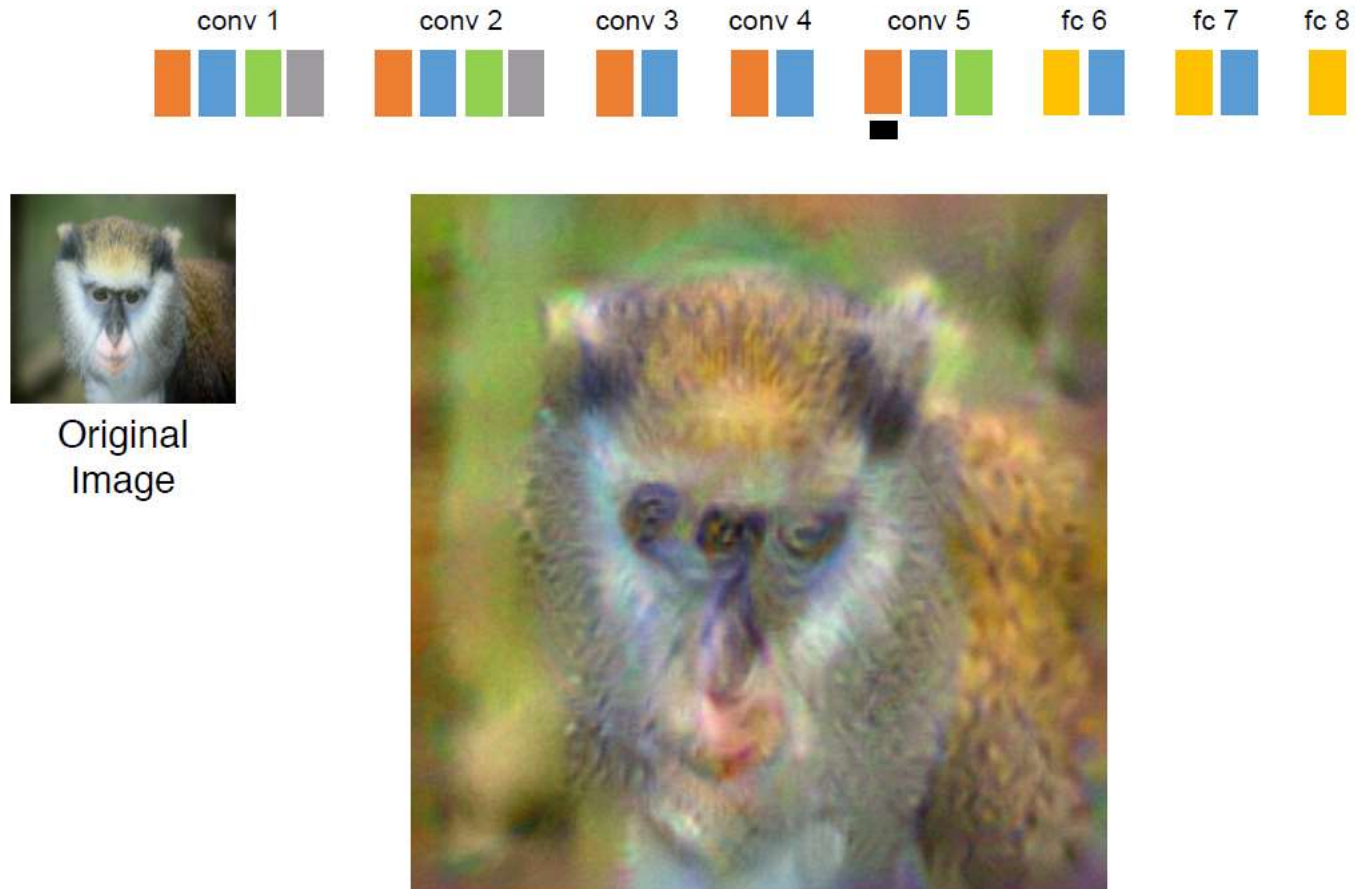
Inversion

22



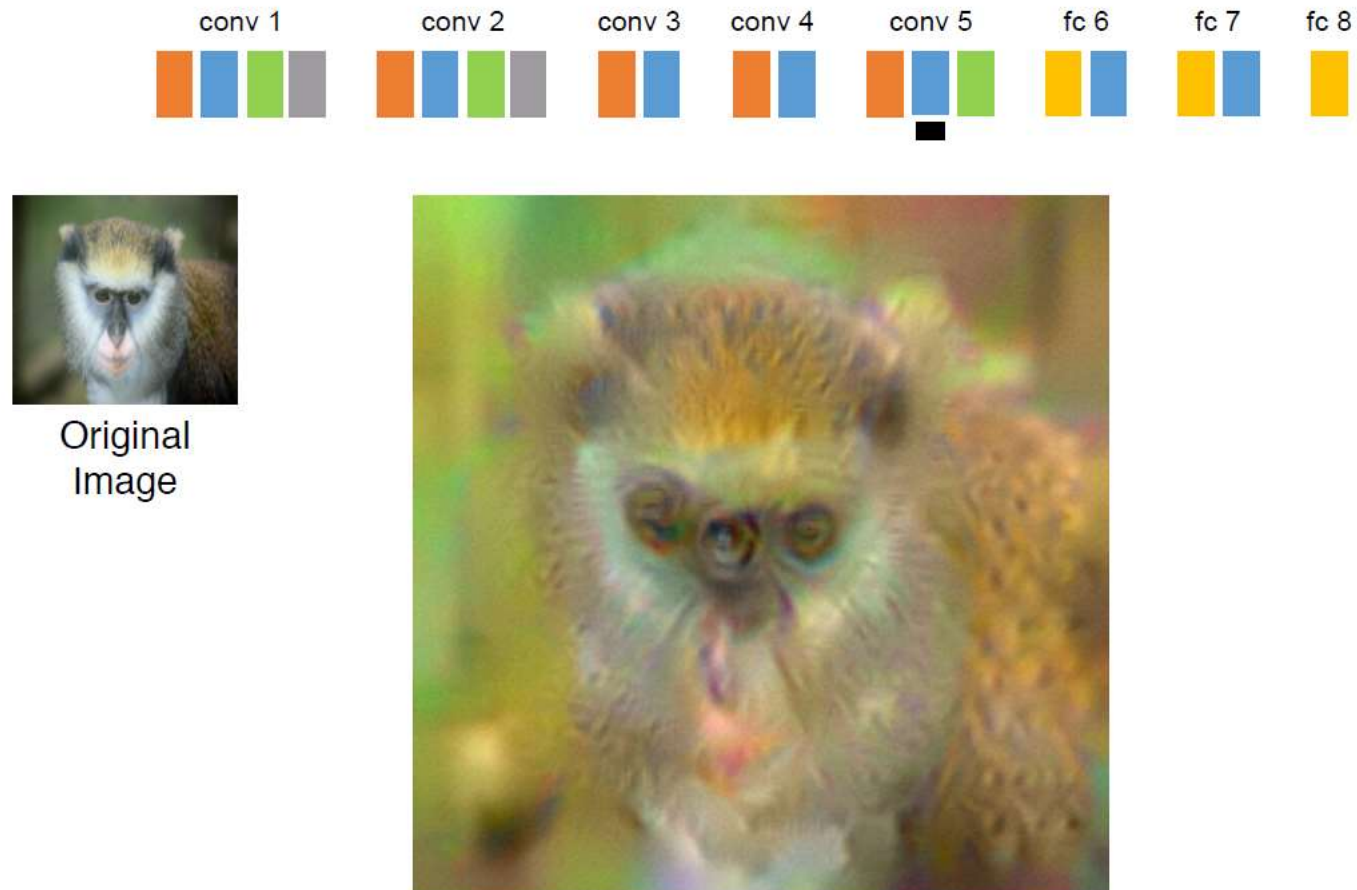
Inversion

23



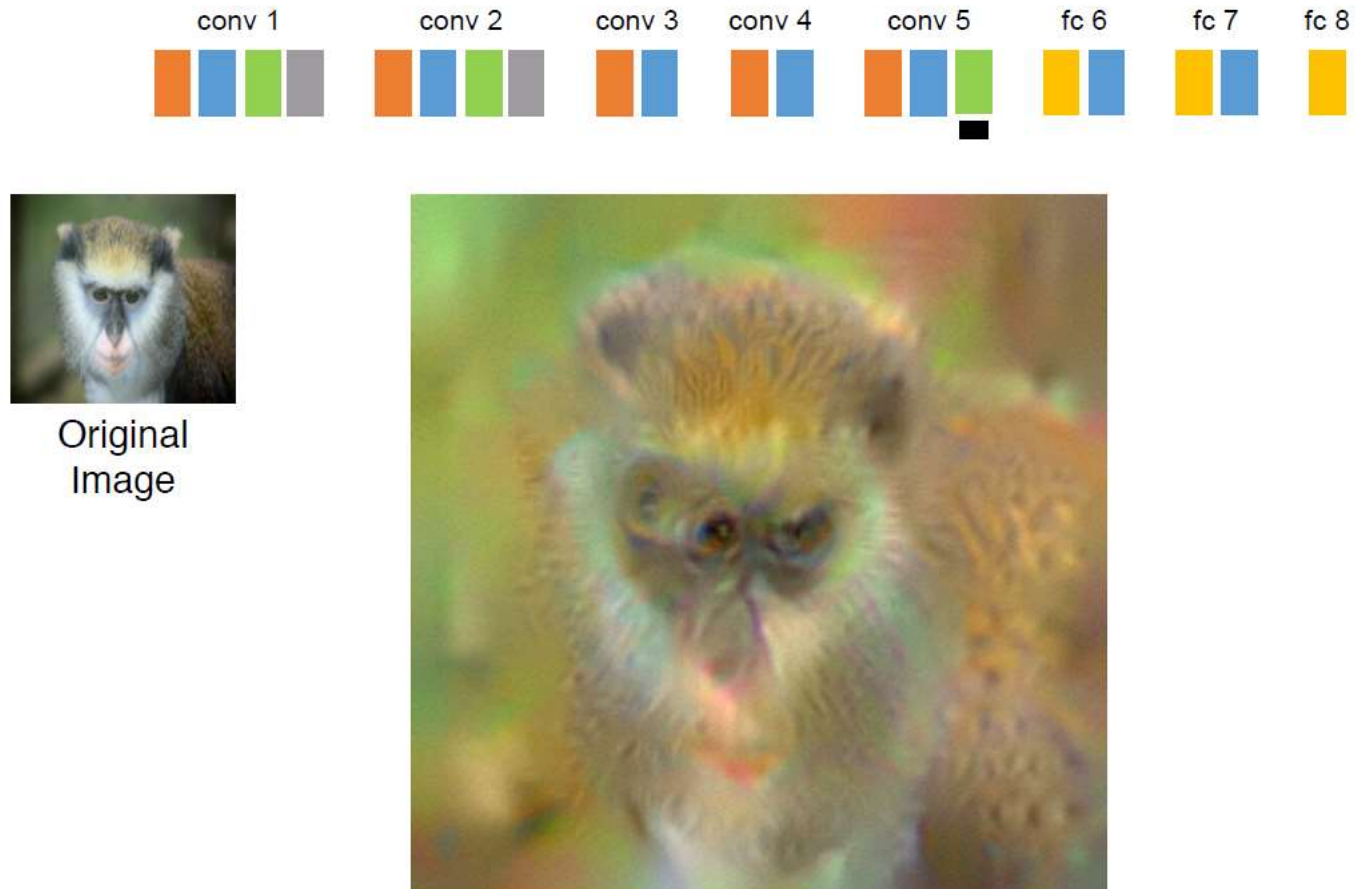
Inversion

24



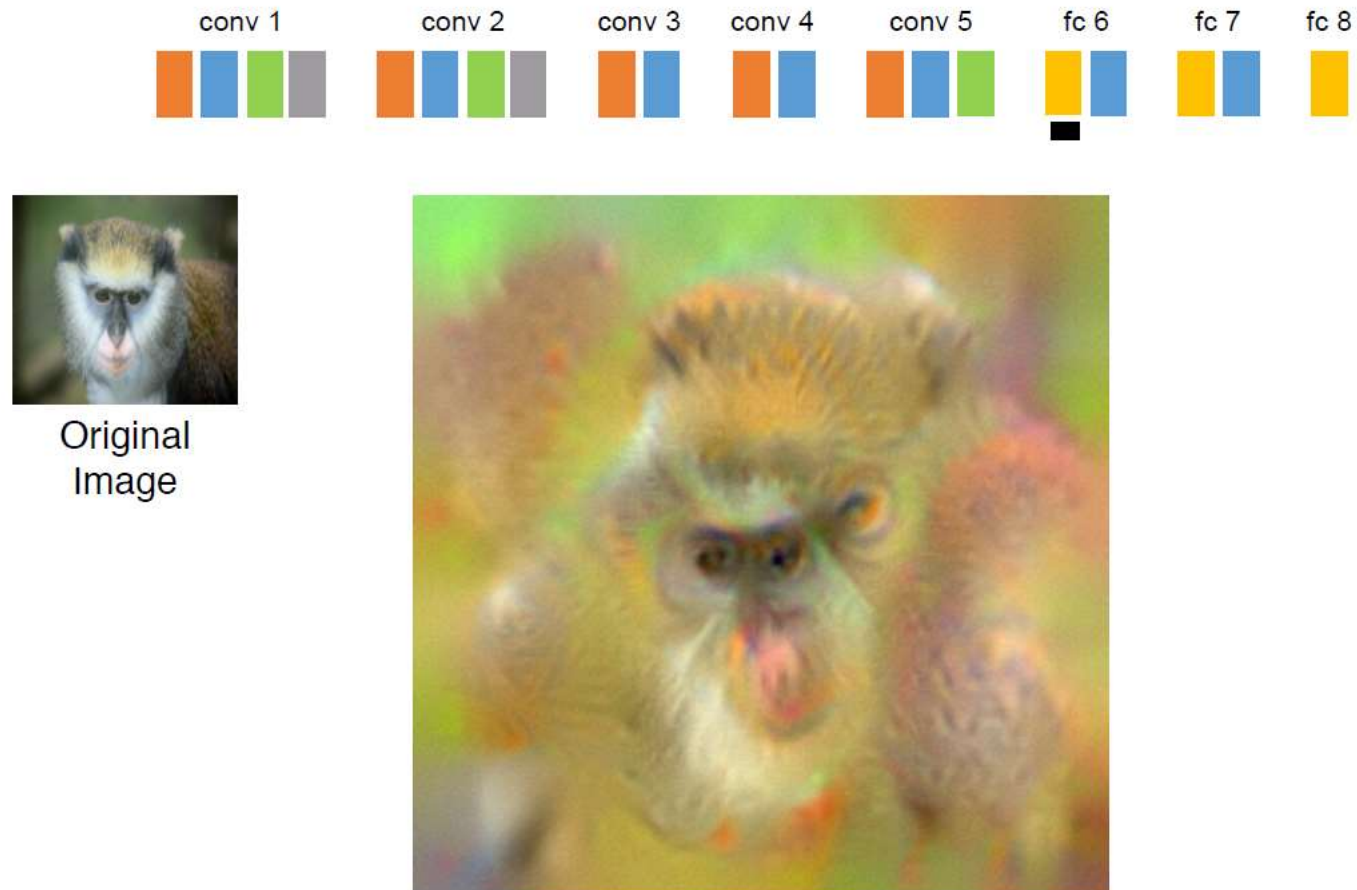
Inversion

25



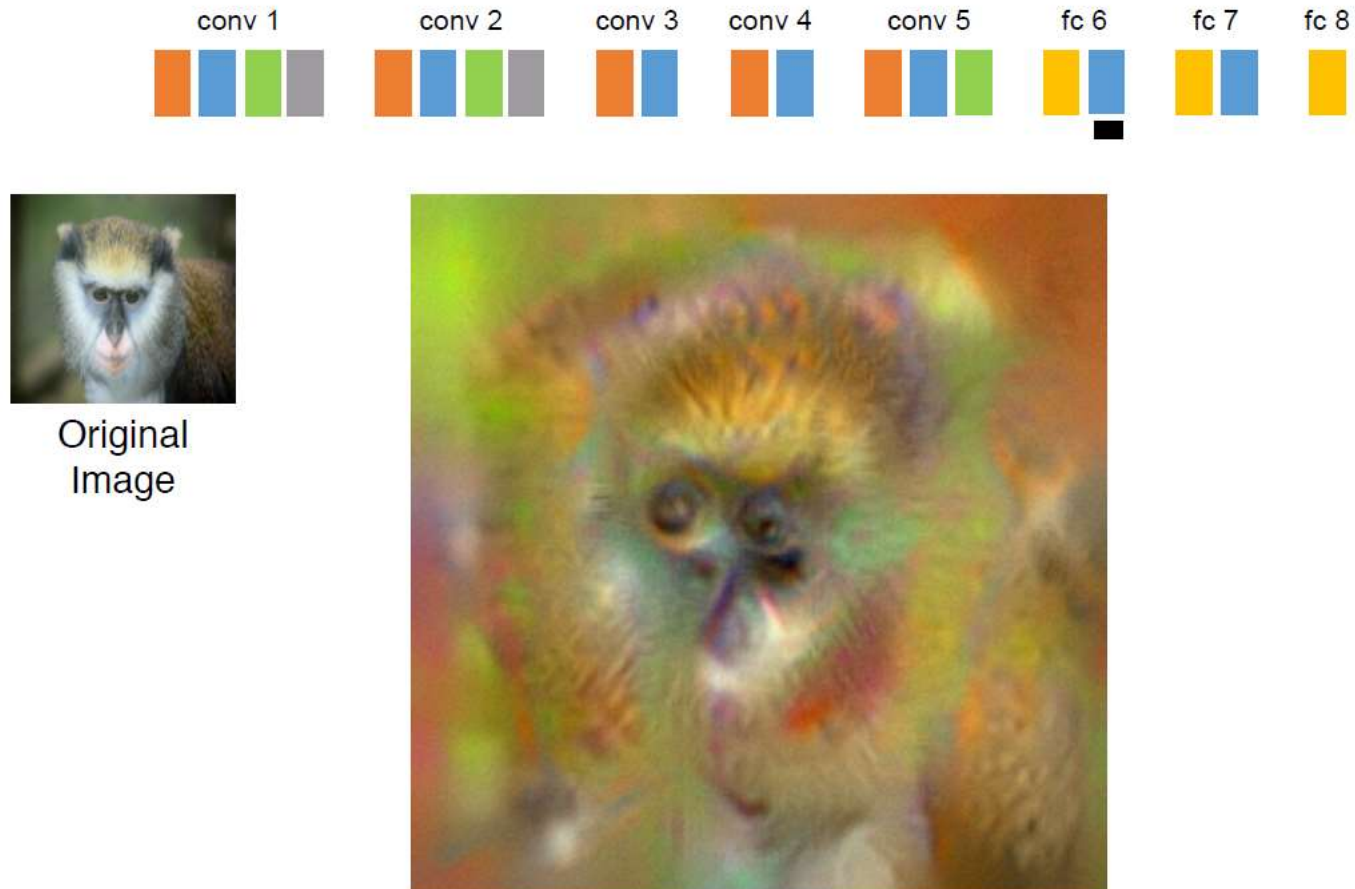
Inversion

26



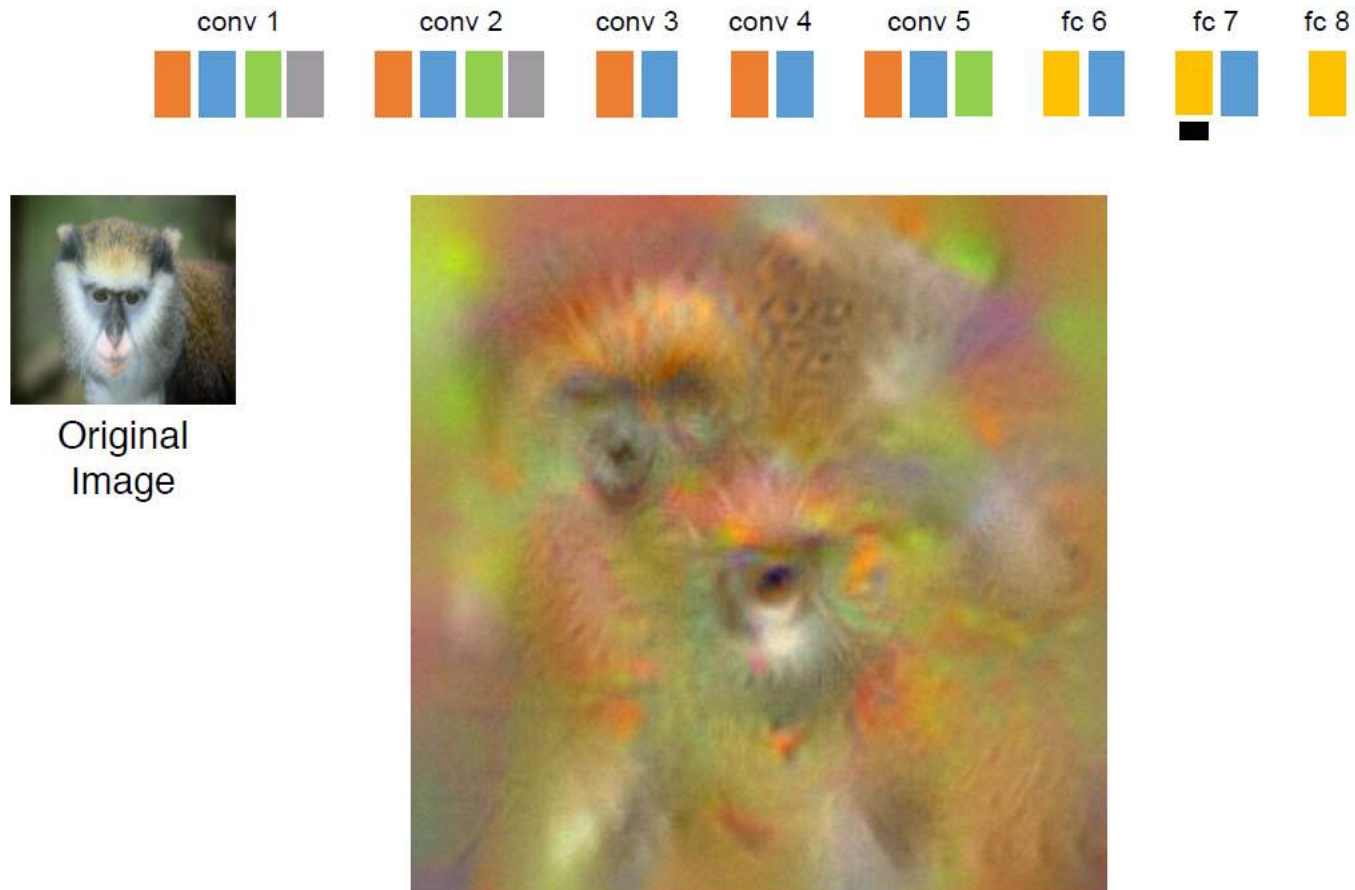
Inversion

27



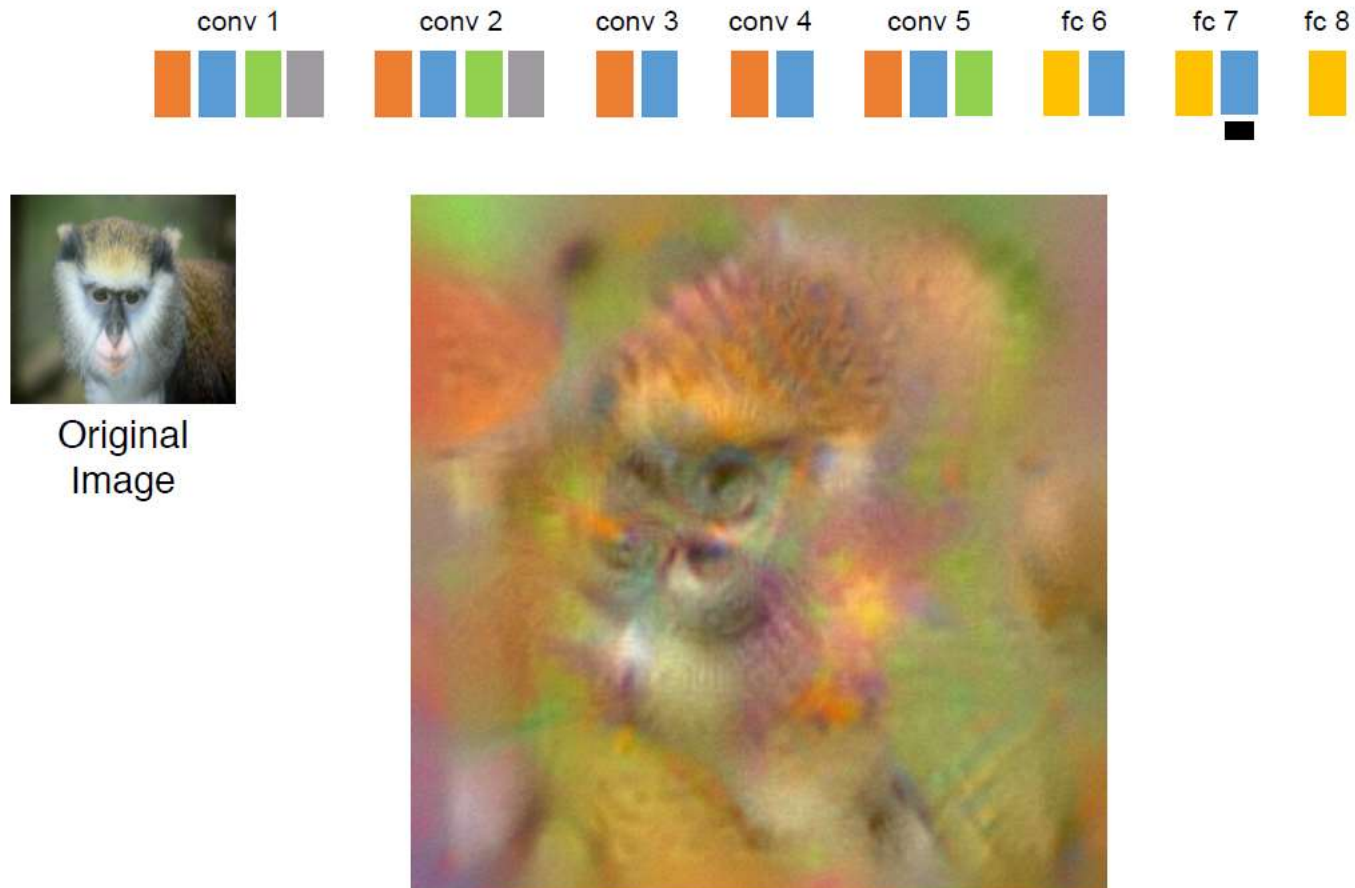
Inversion

28



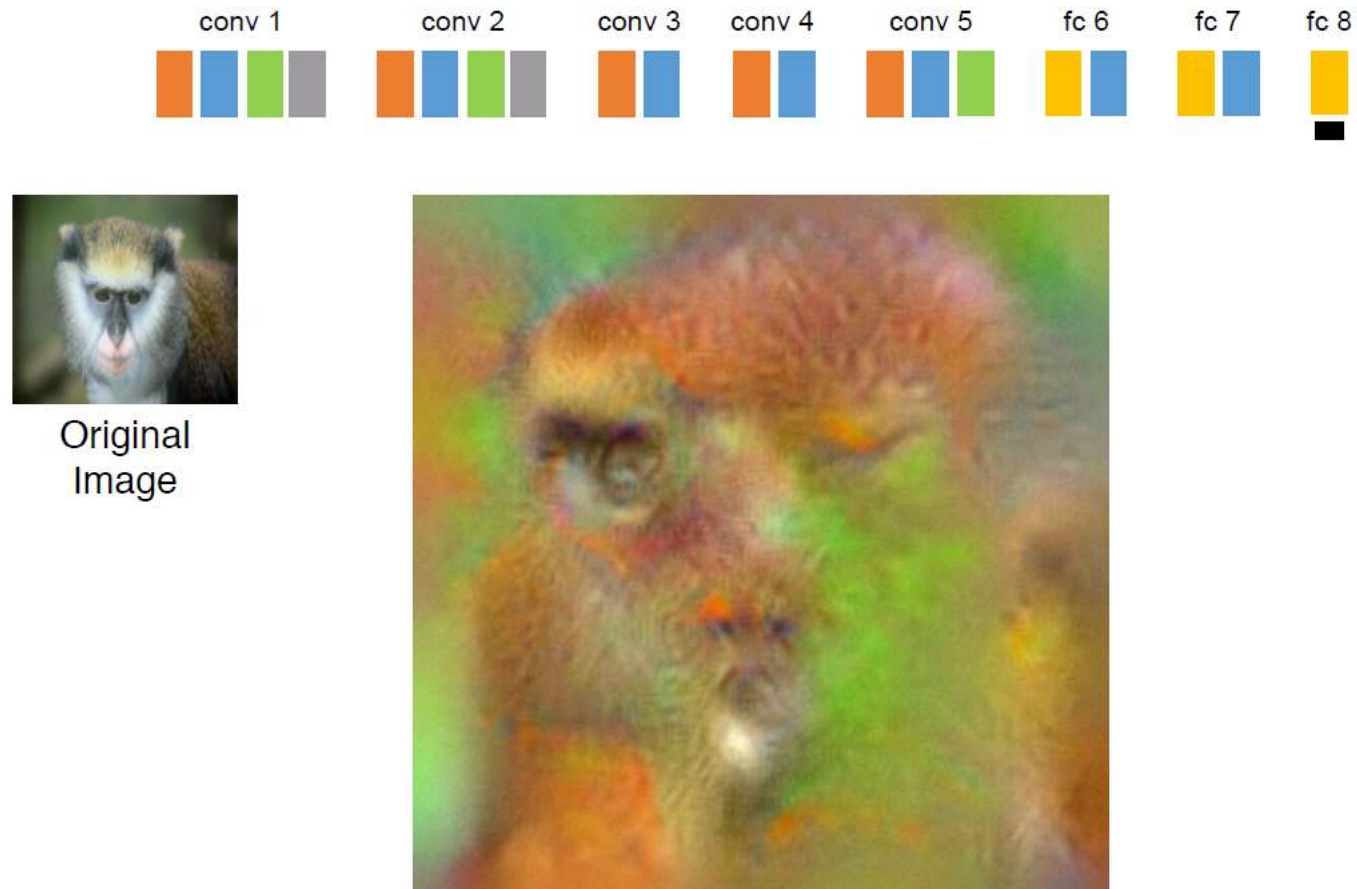
Inversion

29



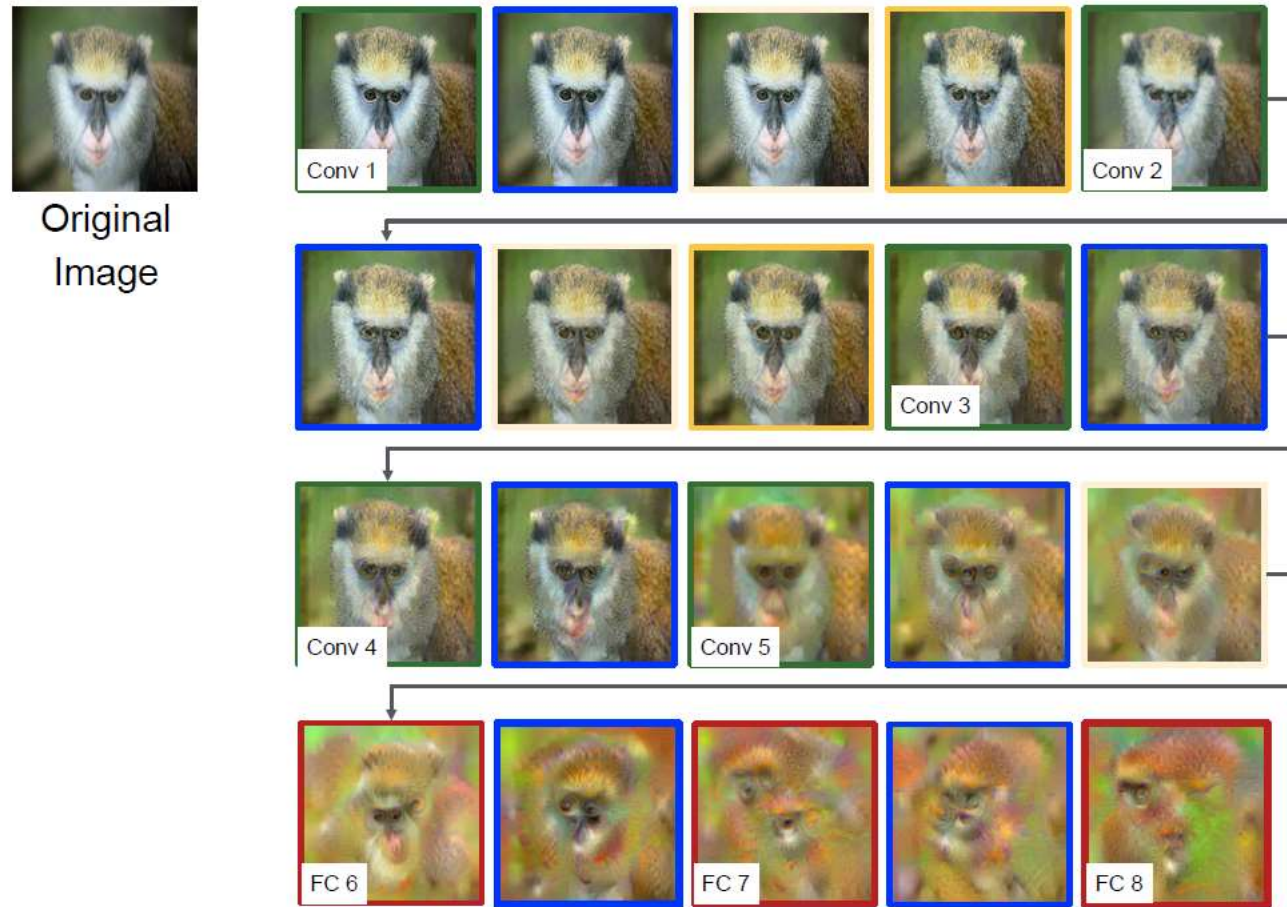
Inversion

30



Inverting a Deep CNN

31



UNDERFITTING, OVERFITTING AND BEST FITTING

Another interesting source:

<https://distill.pub/2017/feature-visualization/>