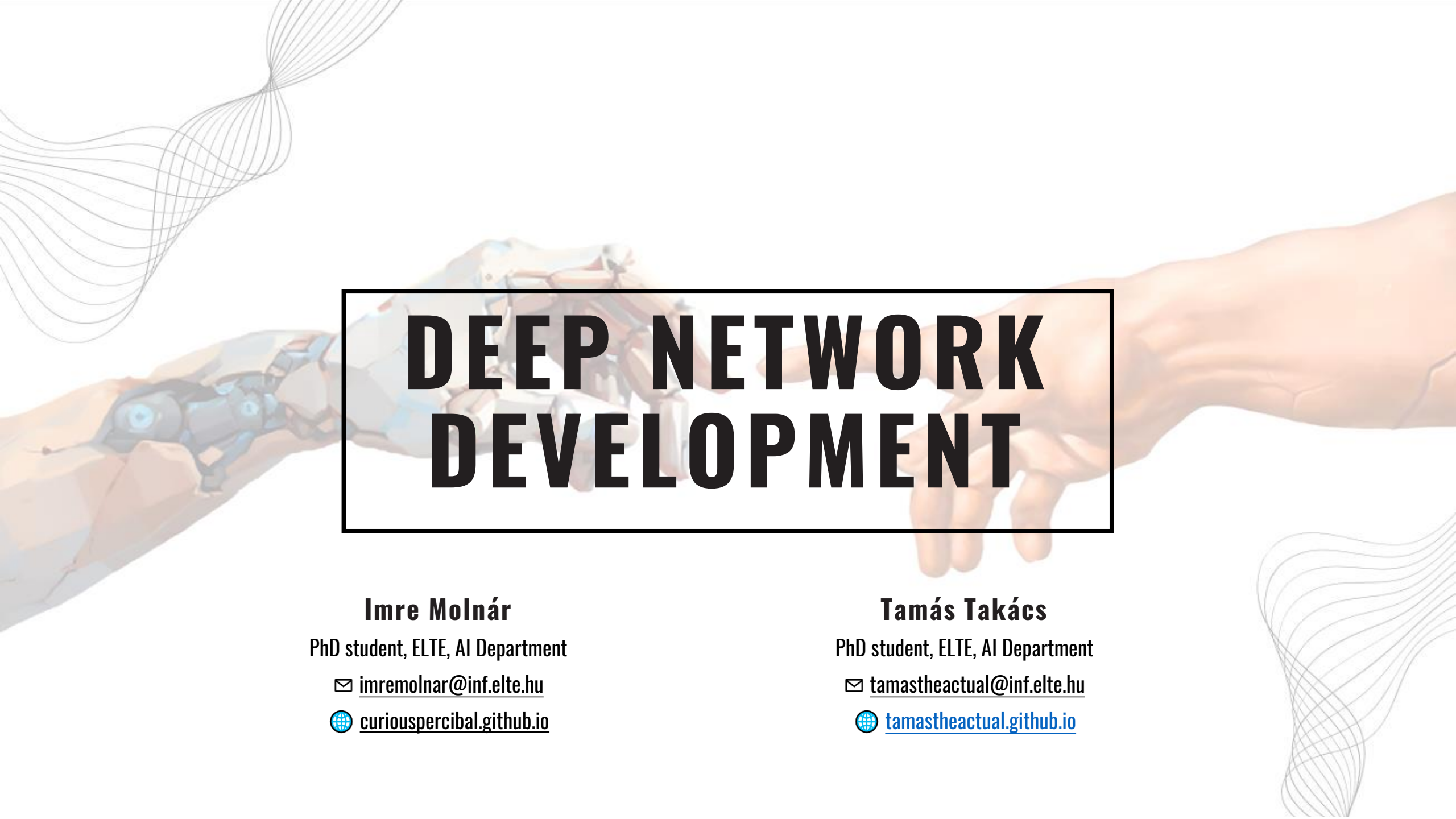


DEEP NETWORK DEVELOPMENT

Imre Molnár

PhD student, ELTE, AI Department

imremolnar@inf.elte.hu

curiouspercibal.github.io

Tamás Takács

PhD student, ELTE, AI Department

tamastheactual@inf.elte.hu

tamastheactual.github.io

Deep Network Development

Lecture 7.

Image Classification Convolutional Neural Networks Transfer Learning

Budapest, 7th October 2025

1 Image Classification

2 Convolutional Neural Networks

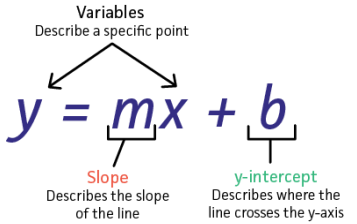
3 CNN Architectures

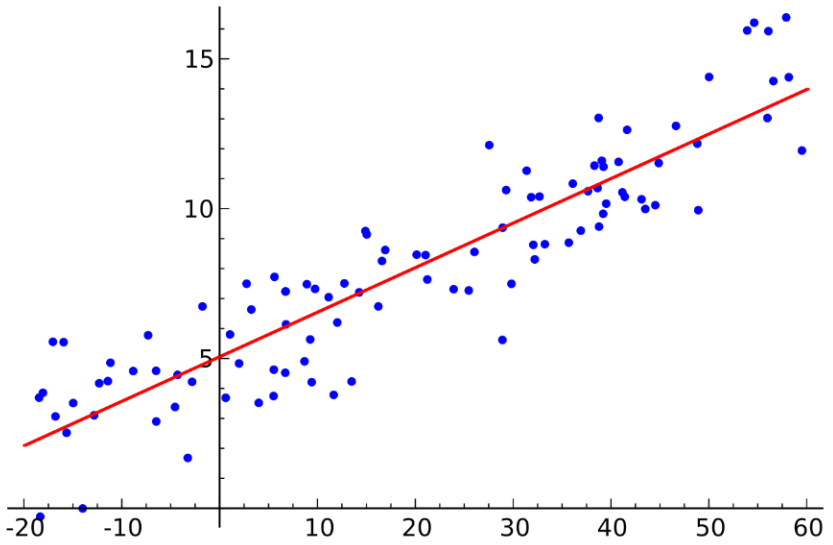
4 Transfer Learning

5 Autoencoders

Linear Regression

- Supervised learning
- Have: (x, y)
  - x – input
  - y – target
- *Goal: Learn a function to map* $\boldsymbol{x \rightarrow y}$.
  - $\boldsymbol{h(x)} = \hat{y}$
  - $\hat{y} = \boldsymbol{\theta_1 x + \theta_0}$
- Regression – Predict real-valued / continuous output:
  - $\hat{y} \in \mathbb{R}$





- Given the height and weight of a person, predict the age of the person.
- Age can be: 10,25,33,71,...

Linear Regression vs Classification

Linear Regression

- Supervised learning
- Have: (x, y)
  - x: input
  - y: target $\in \mathbb{R}$
- *Goal: Learn a function to map $x \rightarrow y$.*
  - $h(x) = \hat{y}$
  - $\hat{y} = \sum_{j=0}^{m} w_j x_j + b$
- Regression – Predict real-valued / continuous output:
  - $\hat{y} \in \mathbb{R}$

Binary Classification

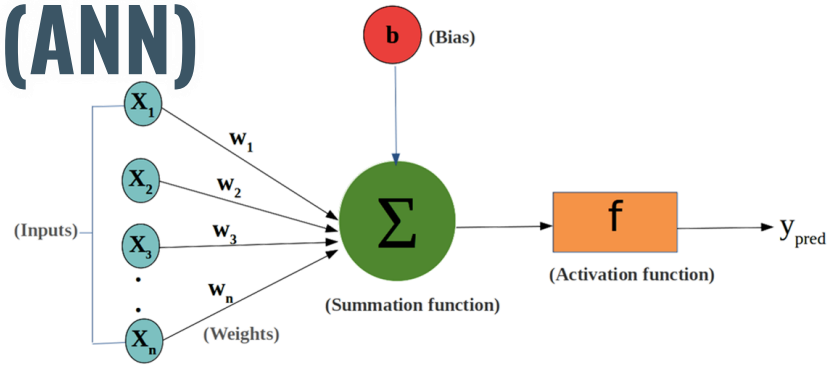
- Supervised learning
- Have: (x, y)
  - x: input
  - y: target $\in \{0, 1\}$
- *Goal: Learn a function to map $x \rightarrow y$.*
  - $h(x) = \hat{y}$
  - $\hat{y} = g\left(\sum_{j=0}^{m} w_j x_j + b\right)$ Add non-linear activation function
- Classification – Predict discrete set of values:
  - $\hat{y} \in \{0, 1\}$

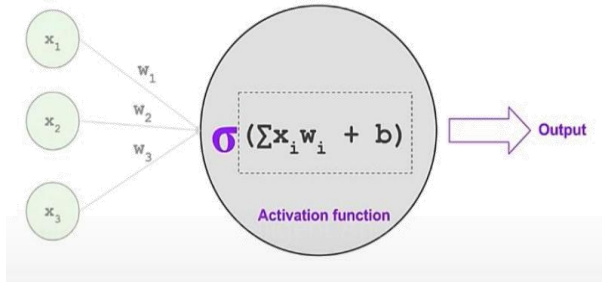
Binary Classification

Binary classification – which of the two classes does the input belong to?

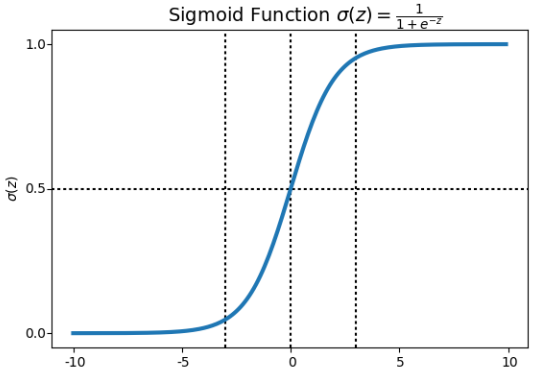
- Cat vs Dog
- Dog vs Mop

Artificial Neural Network (ANN)





- Using ANN for linear regression
- x_j – the inputs
- w_j – parameters we will train
- b – bias parameter
- **g – nonlinear activation function**
- o – the output
- One neuron with m inputs does the following:
  - $o = g\left(\sum_{j=0}^{m} w_j x_j + b\right)$



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

Activation function

- Adds non-linearity
- Output limited to a range (i.e. 0-1)
- Given the height and weight information, predict if it is a human or an animal.
- 0: human; 1: animal

Deep Network Development

Lecture 7.

Image Classification Convolutional Neural Networks Transfer Learning

Budapest, 7th October 2025

1 Image Classification

2 Convolutional Neural Networks

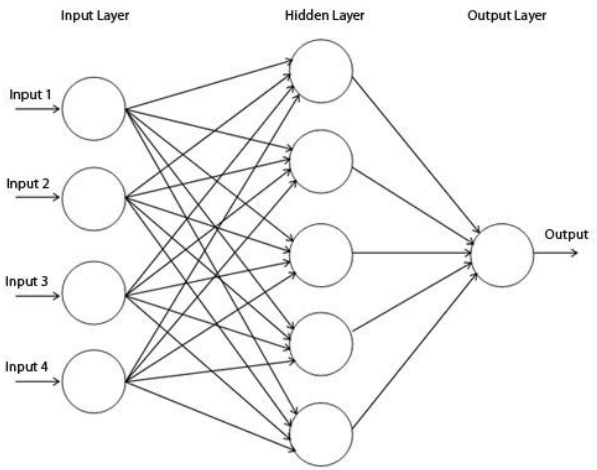
3 CNN Architectures

4 Transfer Learning

5 Autoencoders

Binary Classification

- Binary classification – Two classes
- Is x in the target class C?
- Output $P(x \in C)$, the probability of x is in C
- The network will have 1 output
- h(x) must be between 0 and 1



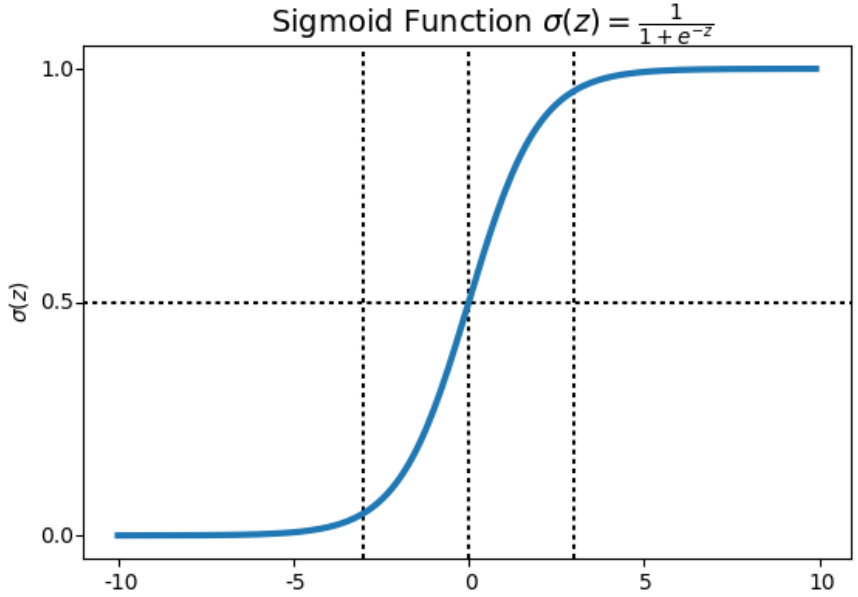
Dog vs Mop

Binary Classification

Binary Classification

- h(x) must be between 0 and 1
- Use the sigmoid activation on last layer

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



- Linear regression becomes Logistic Regression (Binary Classification), if we add a Sigmoid activation function to the output
- Linear Regression: $\boldsymbol{h}(\boldsymbol{x}) = Wx + b$ (continuous output)
- Binary Classification: $\boldsymbol{h}(\boldsymbol{x}) = g(Wx + b)$(discrete output between 0 and 1)

Linear Regression

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

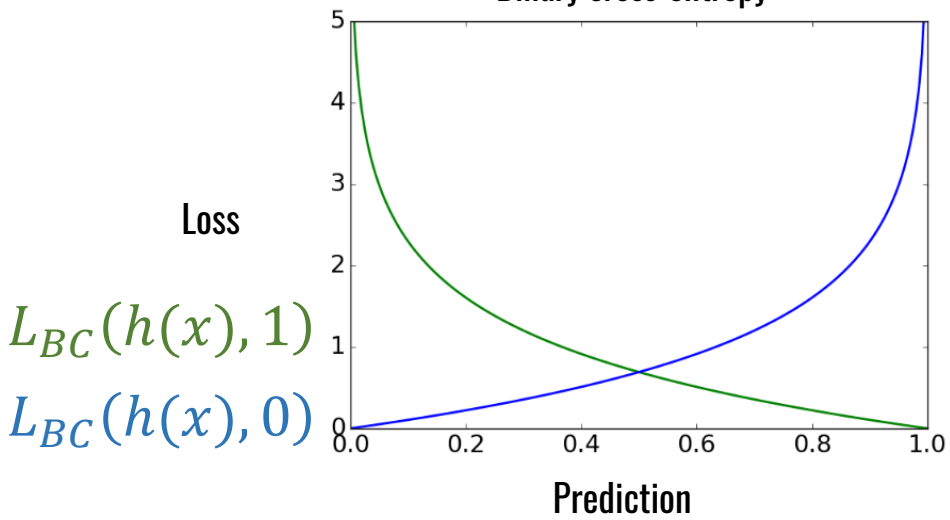
Binary Classification Loss

- Binary cross-entropy loss

$$L_{BC}(h(x), y) = -y \log h(x) - (1 - y) \log(1 - h(x))$$

- Assumptions:
  y is 0 or 1
  $h(x) = \hat{y}$ is between 0 and 1

Binary cross-entropy

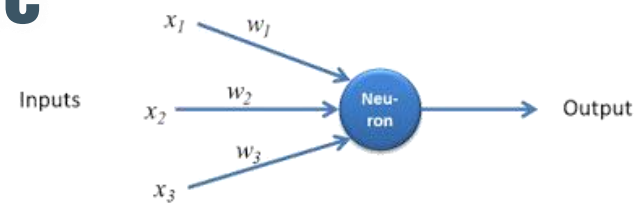


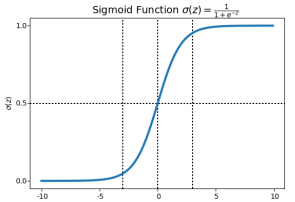
$$L_{BC}(h(x), y) = -y \log h(x) - (1 - y) \log(1 - h(x))$$
$$L_{BC}(1,1) = -1 \log 1 - (1 - 1) \log(1 - 1)$$
$$L_{BC}(1,1) = 0 - (0) \log(0)$$
$$L_{BC}(1,1) = 0$$

log(0) is undefined -> clipping

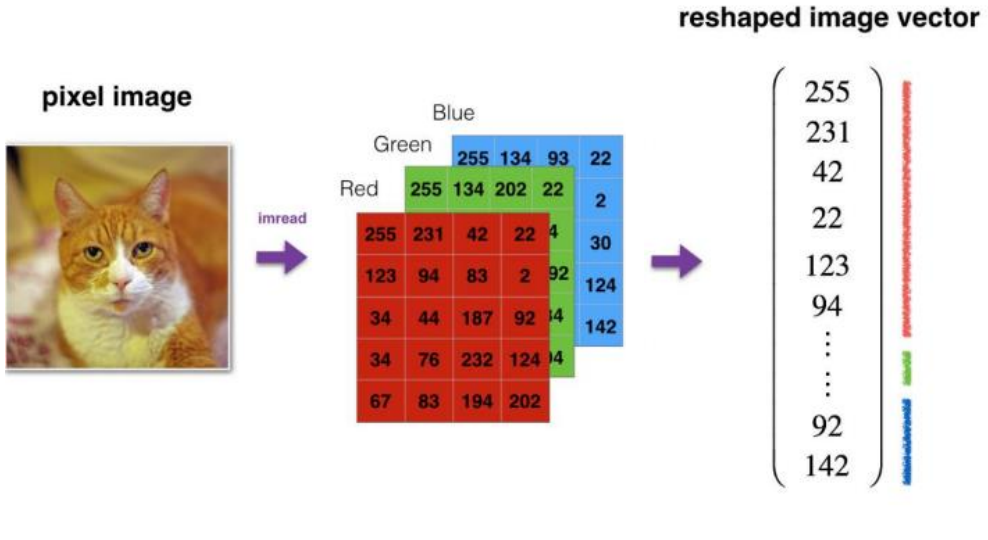
Binary Classification example

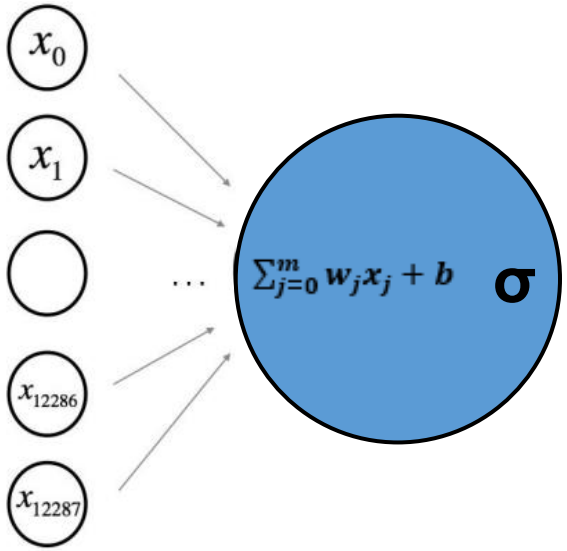
Classes C = {0: Dog, 1: Cat}

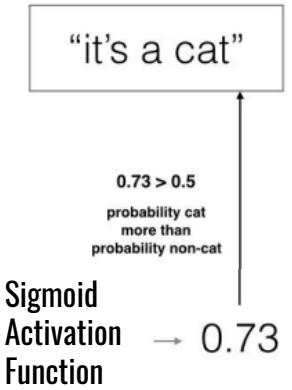




INPUT **NEURAL NETWORK** **OUTPUT** **TARGET**







Y = 1

LOSS

$$L_{BC}(h(x), y) = -y \log h(x) - (1-y) \log(1-h(x))$$
$$L_{BC}(0.73, 1) = -1 \log 0.73 - (1-1) \log(1-0.73)$$
$$L_{BC}(0.73, 1) = -1 * -0.13667714 - (0) \log(0.23)$$
$$L_{BC}(0.73, 1) = 0.13667714$$

$$\boldsymbol{L_{BC}(0.73, 1) = 0.14}$$

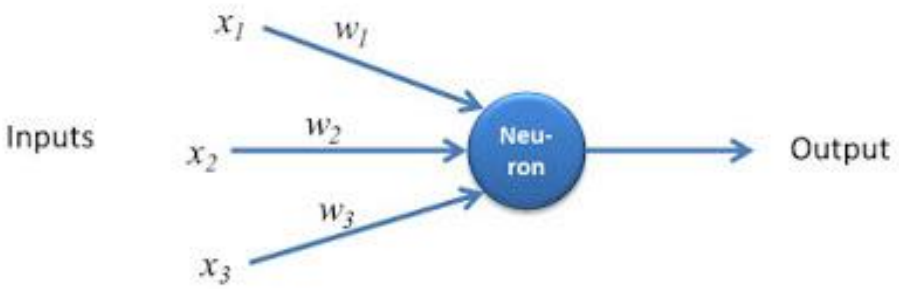
More neurons and categories

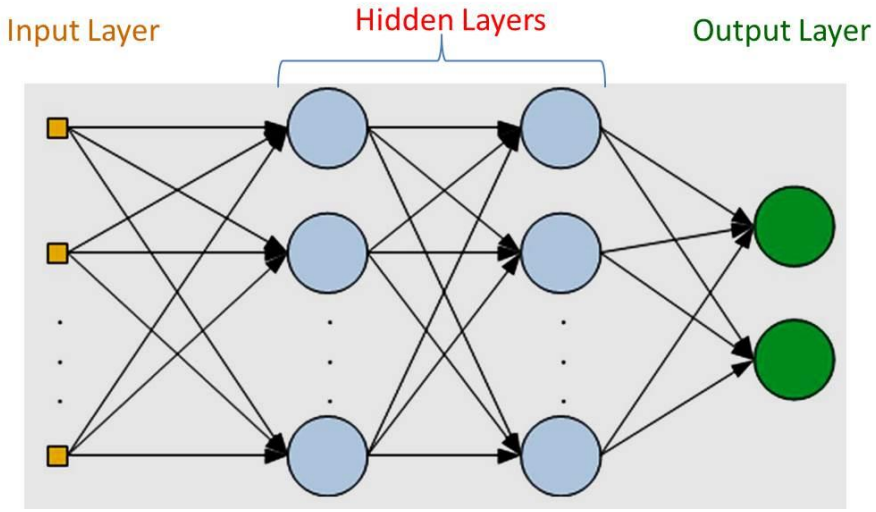
- Single artificial neuron
- Binary Classification (0 or 1)

- Deep Neural Network
- Multi-Class Classification (0,1,2,3…)

Deep Neural Networks

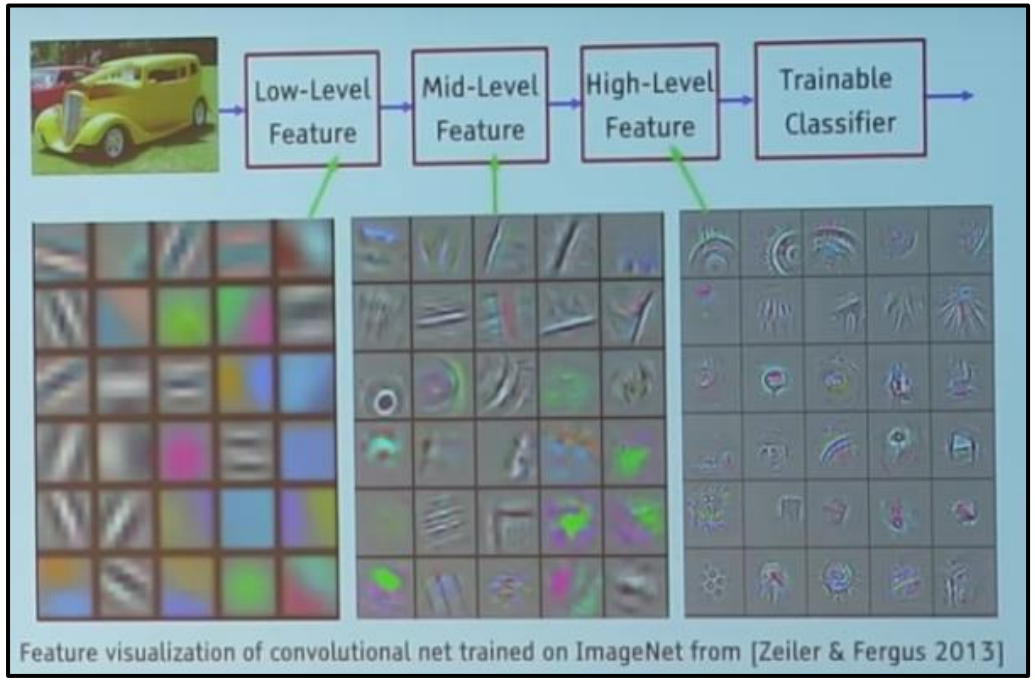
- Neural networks are built up from neurons, that have inputs and outputs
- Neurons are organised into layers
- Layers refine the output of the previous layers

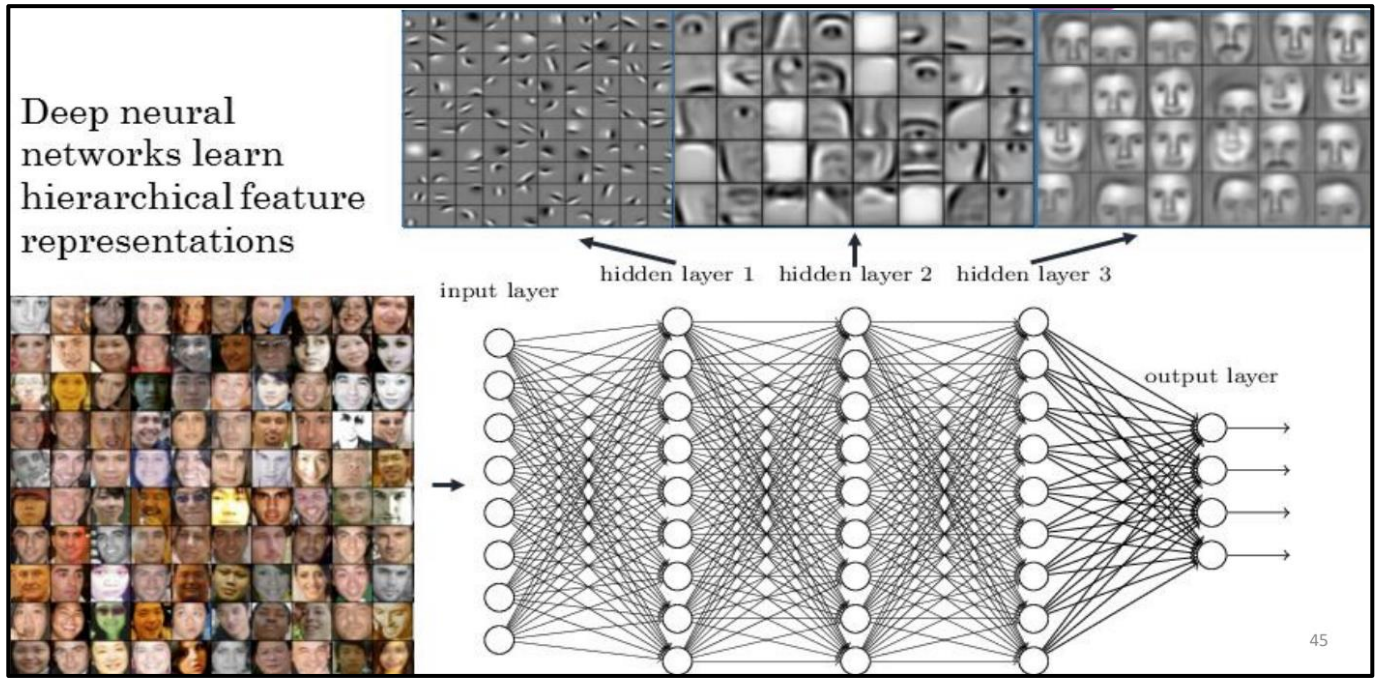




Deep Neural Networks

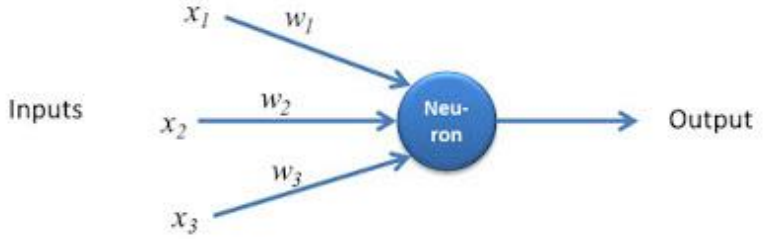
- Each layer refines the previous layer
- Visualization demo: https://adamharley.com/nn_vis/



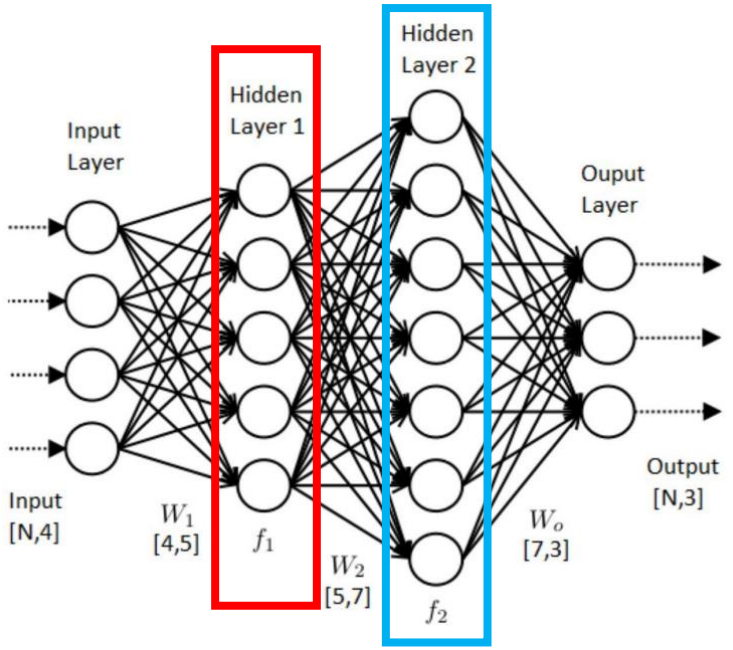


Deep Neural Networks

- Artificial Neural Networks ⟺ Feed Forward Neural Networks ⟺ Fully Connected Networks



- A neural network is built up from neurons
- Neurons are organised into layers
- Layers refine the output of the previous layers
  - $\boldsymbol{h_1(x)} = g\left(W^{(1)}x + b^{(1)}\right)$
  - $h_2(x) = g\left(W^{(2)}\boldsymbol{h_1} + b^{(2)}\right)$
  - $h_2(x) = g\left(W^{(2)}\boldsymbol{g}\left(\boldsymbol{W^{(1)}x} + \boldsymbol{b^{(1)}}\right) + b^{(2)}\right)$



Deep Neural Networks [1, 2]

- How to find *W* and *b*?
- Given $h = g\left(W^{(2)}\boldsymbol{g}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) + b^{(2)}\right)$

- Select a loss function L that measures how good *h* is:
  - the smaller L(h(x),y) , the better *h* is

- Update $\boldsymbol{W}^{(1)}$:
  $$\boldsymbol{W}_t{}^{(1)} = \boldsymbol{W}_{t\_1}{}^{(1)} - \alpha\nabla J(\boldsymbol{W}^{(1)})$$
- Update $\boldsymbol{W}^{(2)}$:
  $$\boldsymbol{W}_t{}^{(2)} = \boldsymbol{W}_{t\_1}{}^{(2)} - \alpha\nabla J(\boldsymbol{W}^{(2)})$$
- Update $\boldsymbol{b}^{(1)}$:
  $$\boldsymbol{b}_t{}^{(1)} = \boldsymbol{b}_{t\_1}{}^{(1)} - \alpha\nabla J(\boldsymbol{b}^{(1)})$$
- Update $\boldsymbol{b}^{(2)}$:
  $$\boldsymbol{b}_t{}^{(2)} = \boldsymbol{b}_{t\_1}{}^{(2)} - \alpha\nabla J(\boldsymbol{b}^{(2)})$$

Terminology

- L(h(x),y) – loss or error function for a single data point of the dataset
- C(θ), J(W) – cost or objective function for the entire dataset (usually average)
- The update rule can also be in the form:

$$w_i \leftarrow w_i + \Delta w_i$$

$$b \leftarrow b + \Delta b$$

$$\Delta w_i = -\alpha\frac{\partial L}{\partial w_i}$$

$$\Delta b = -\alpha\frac{\partial L}{\partial b}$$

[1] Backpropagation derivation: https://www.cs.put.poznan.pl/pliskowski/pub/teaching/eio/lab1/eio-supplementary.pdf
[2] Another backpropagation derivation (be aware of notation differences): https://www.cs.swarthmore.edu/~meeden/cs81/s10/BackPropDeriv.pdf

Deep Neural Networks

1. Select W_0 randomly
2. Calculate $h(X)$ for all training examples
3. Calculate the cost:

$$J(W) = \frac{1}{N}\sum_{i=1}^{N} L(h(x_i), y_i)$$

4. Update W:

$$W_t = W_{(t-1)} - \alpha \nabla J(W)$$

5. Repeat steps 2-4 until convergence

Deep Neural Networks

1. Select W_0 randomly
2. Calculate $h(X)$ for all training examples
3. Calculate the cost function:

$$J(W) = \frac{1}{N}\sum_{i=1}^{N} L(h(x_i), y_i)$$

4. Update W:

$$W_t = W_{(t-1)} - \alpha \nabla J(W)$$

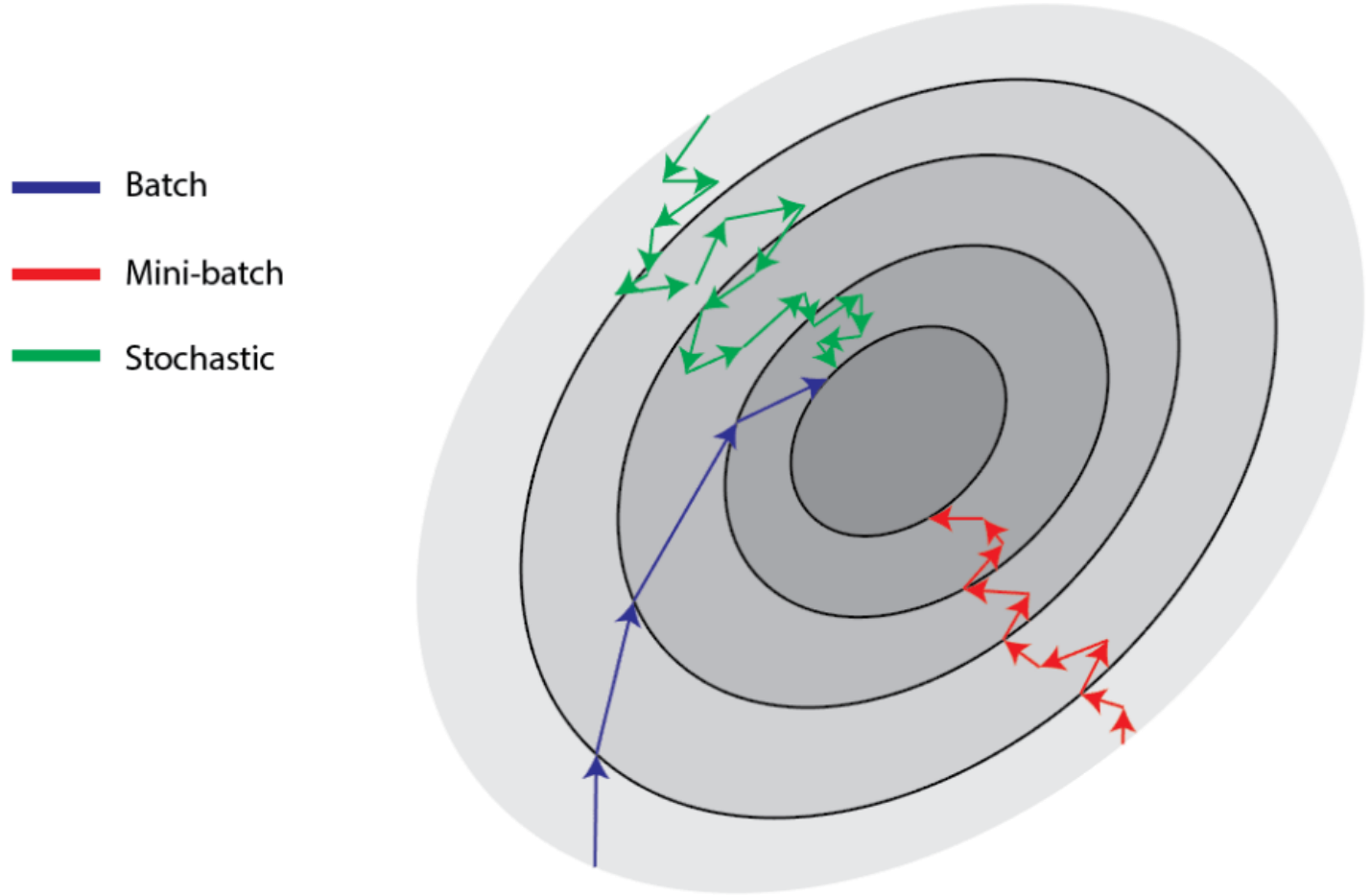
5. Repeat steps 2-4 until convergence

Batch size – the number of training samples used to optimize the model's parameters

Types (with Gradient Descent)

- **Batch Gradient Descent**
  - Batch size = Size of dataset
- **Mini-Batch Gradient Descent**
  - Batch size = a subset of the dataset examples; typically, 2^n , e.g., 8,16,32,128,...
  - In each iteration, a mini-batch is randomly sampled from the dataset
- **Stochastic Gradient Descent**
  - Batch size = 1
  - In each iteration, a single example is randomly sampled from the dataset

Training Deep Neural Networks



Hyperparameter vs Parameter

Hyperparameters

To create and train Neural Networks we must make certain decisions (we choose the values):

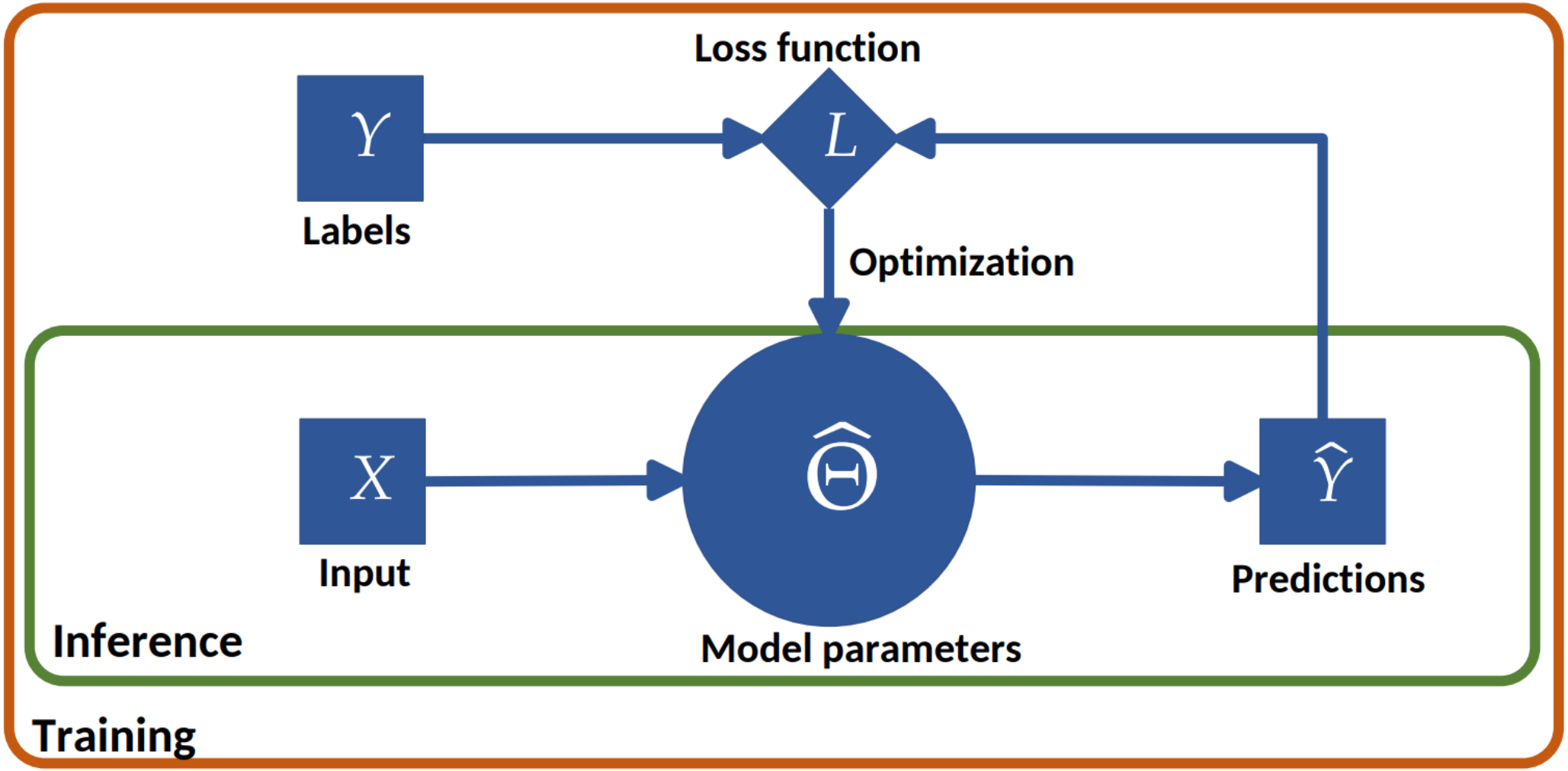
- Number of layers; Number of neurons in each layer
- Activation functions
- Learning rate
- Batch size
- Many more (optimization related)

Parameters

What the Neural Network learns. The weights and biases of the network are optimized with an algorithm (Gradient Descent, etc.). We do not choose the values for them.[3]

3. We can choose the initial values (weight initialization), it can be initialized with zeros, ones, random values, etc. However, the network learns the optimal values through training.

Deep Neural Networks



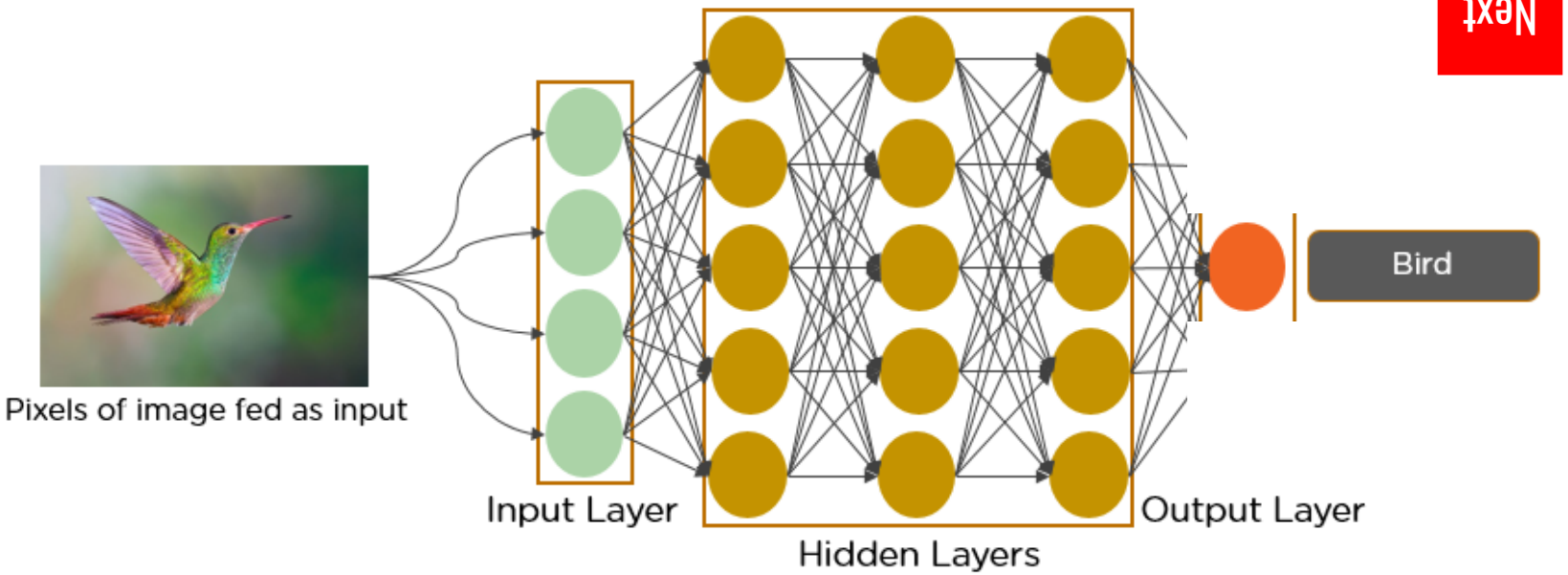
More neurons and categories

- **Single artificial neuron**
- Binary Classification (0 or 1)

→

- **Deep Neural Network**
- Multi-Class Classification (0,1,2,3…)

Next

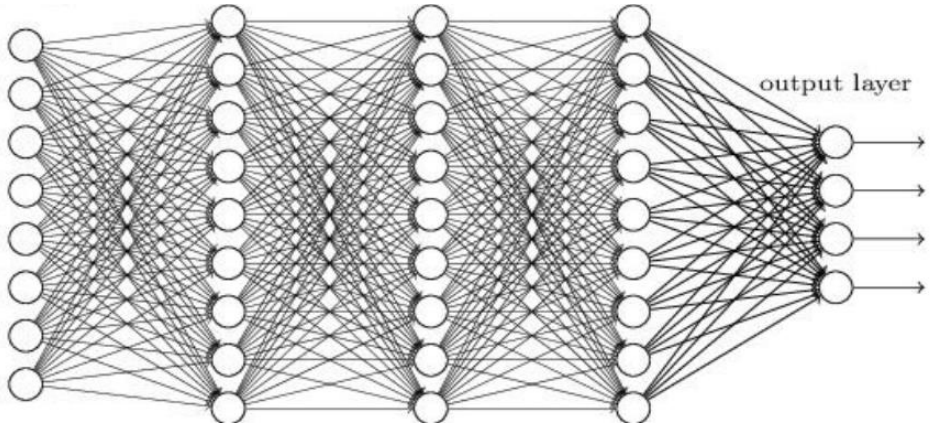


Multi-Class Classification

- With *K* classes
- The network will have K outputs
- Softmax activation for squashing outputs between 0 and 1

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}}$$

This is basically a probability distribution on the classes.
They sum up to 1.

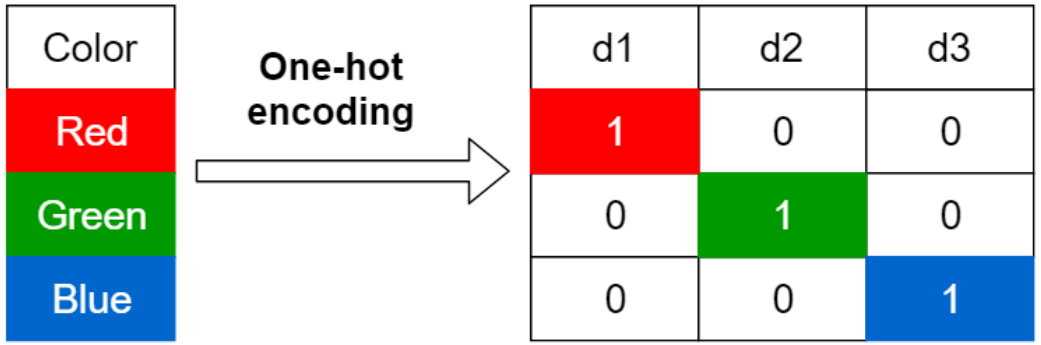


Multi-Class Classification

Output of the neural network is a K long vector

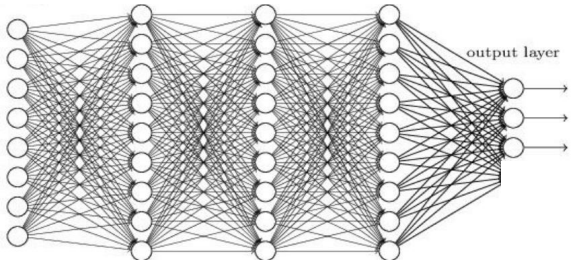
How should we encode the ground-truth values?
One-hot encoding (K=3):
(cat) 1 -> [1, 0, 0]
(dog) 2 -> [0, 1, 0]
(horse) 3 -> [0, 0, 1,]

Just as h(x): values between 0 and 1, sum up to 1

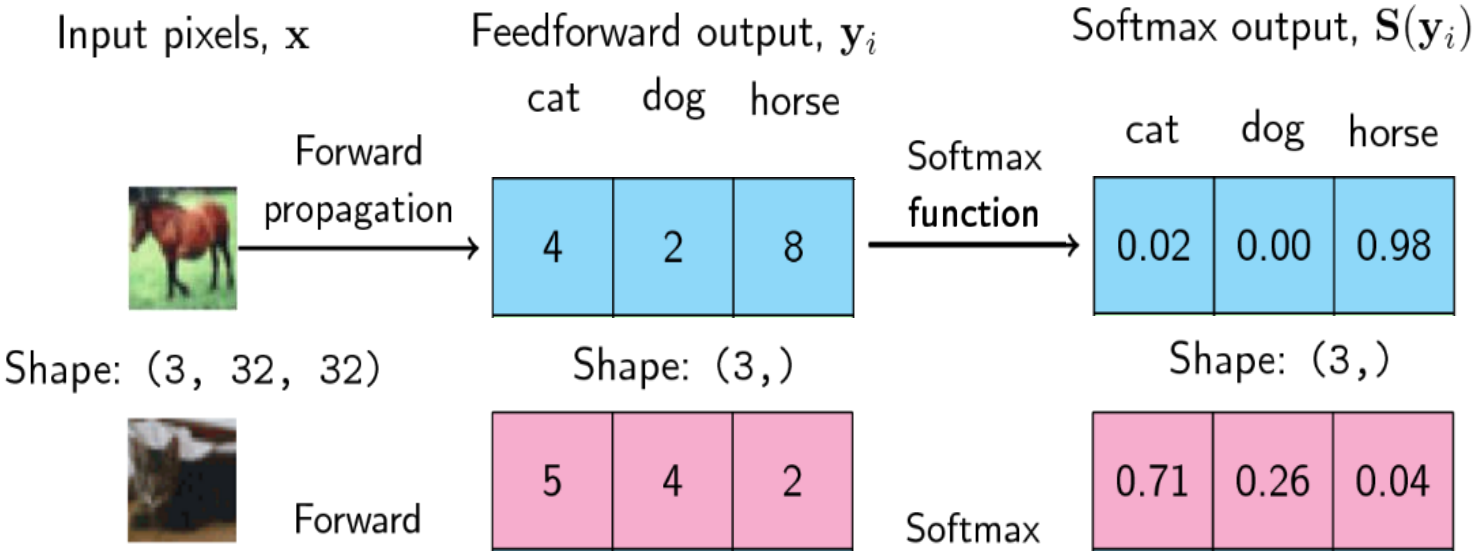


Multi-Class Classification

- K = 3
- Softmax activation $\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}}$
- Probability distribution on the classes. They sum up to 1.



(Cat, Dog, Horse)



Multi-Class Classification Loss

With *K* classes
Categorical Cross-entropy loss

$$L(h(x), y) = -\sum_{i=1}^{K} y_i \log h_i(x)$$

Assumptions:

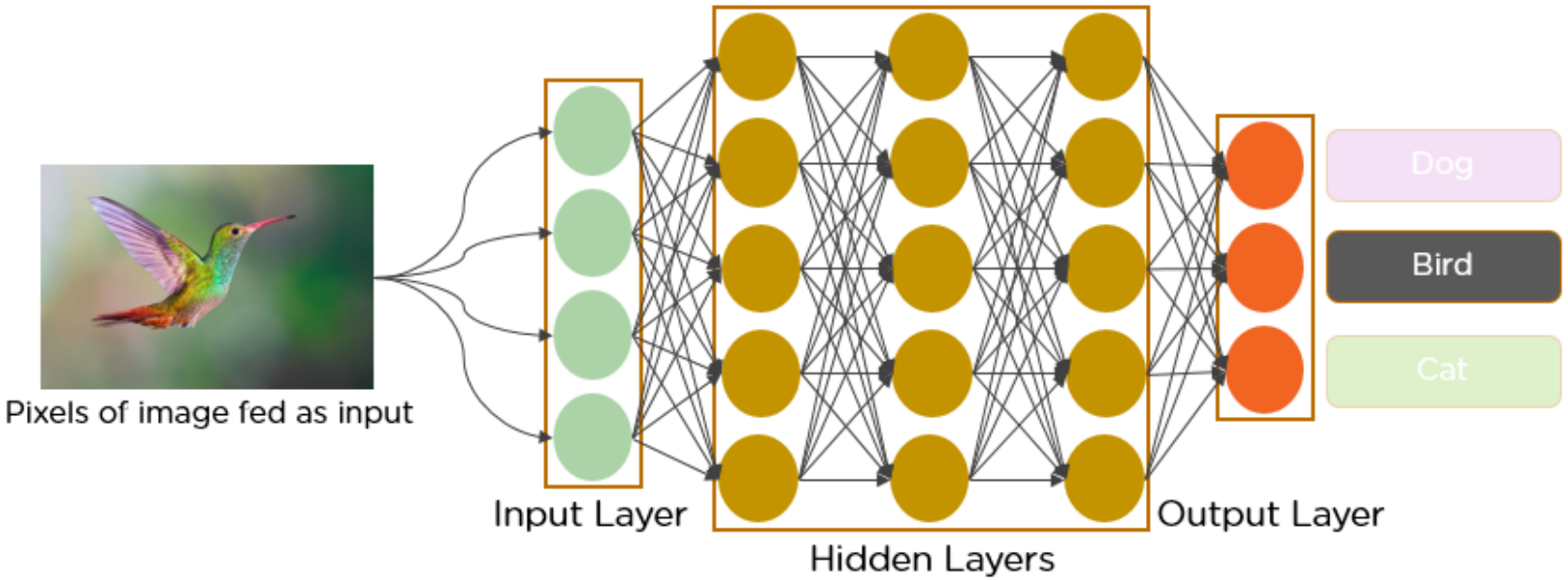
- *y* is a vector of *K elements with values* 0 or 1, exactly one element is 1
- $h_i(x)$ is a vector of K elements between 0 and 1, the sum of all elements is equal to 1

More neurons and categories

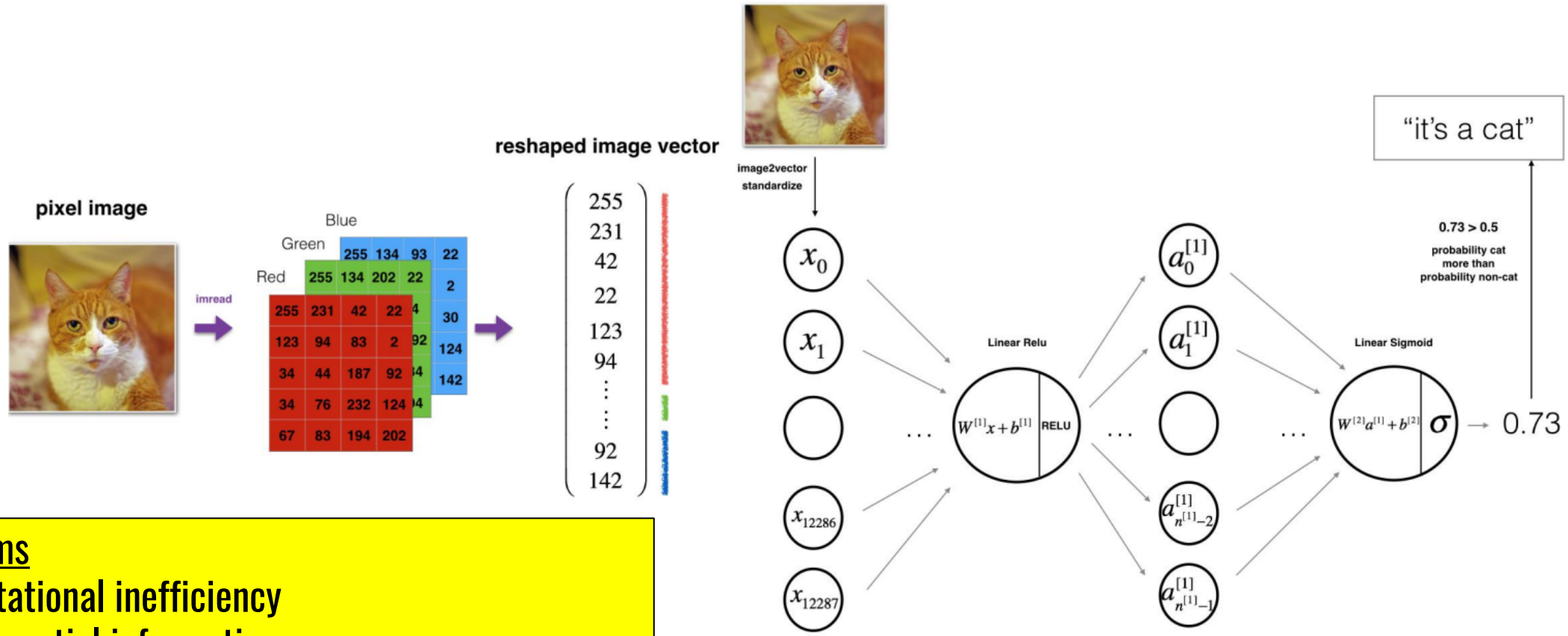
- Single artificial neuron
- Binary Classification (0 or 1)

→

- Deep Neural Network
- Multi-Class Classification (0,1,2,3...)



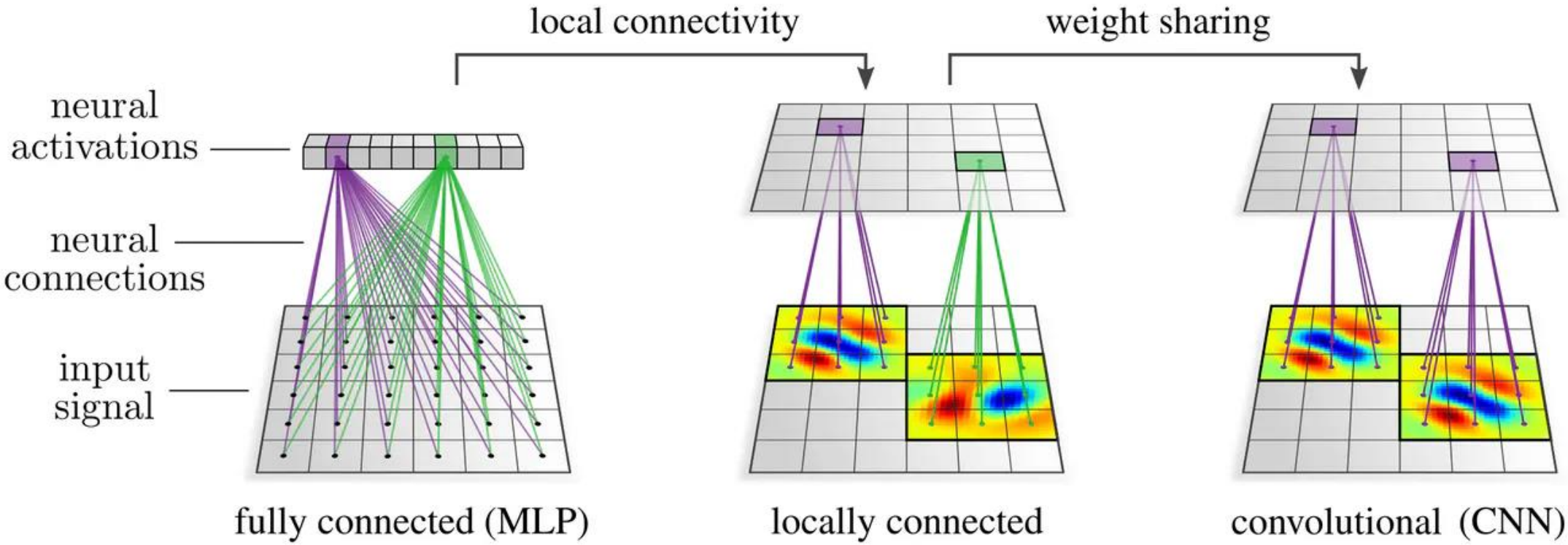
Disadvantages of Feed Forward Neural Networks



Problems

Computational inefficiency
Loss of spatial information
Independent weights/features

Disadvantages of Feed Forward Neural Networks (Intro to CNNs)



Deep Network Development

Lecture 7.

Image Classification Convolutional Neural Networks Transfer Learning

Budapest, 7th October 2025

1 Image Classification

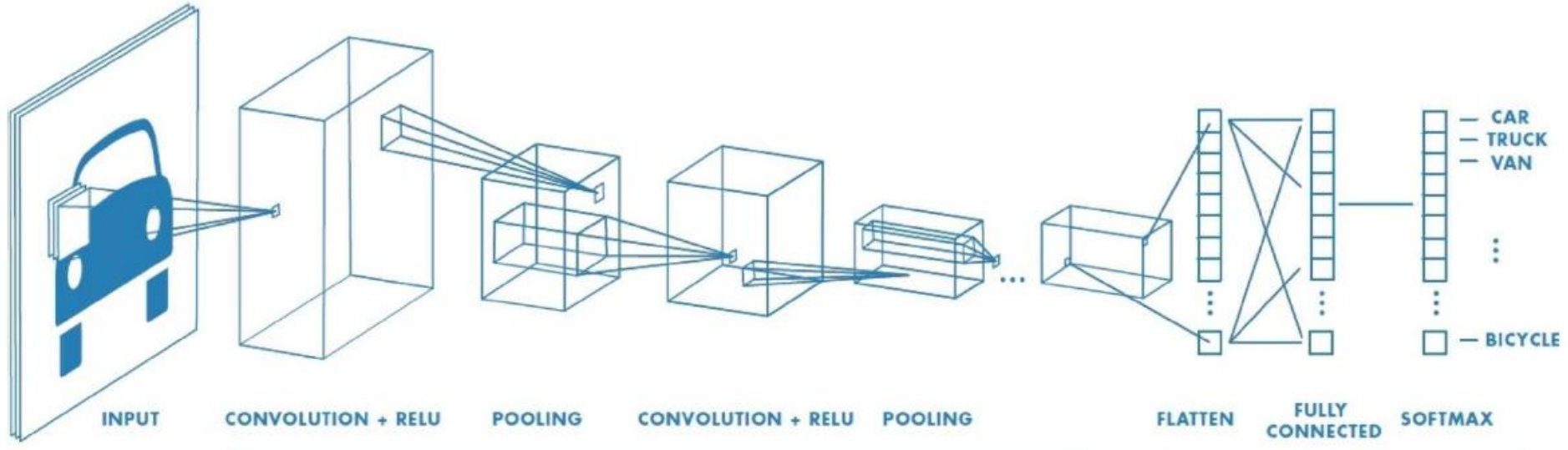
2 Convolutional Neural Networks

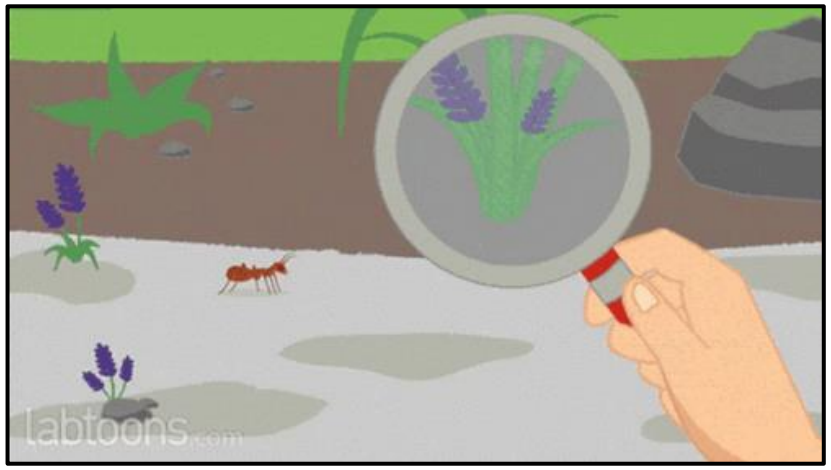
3 CNN Architectures

4 Transfer Learning

5 Autoencoders

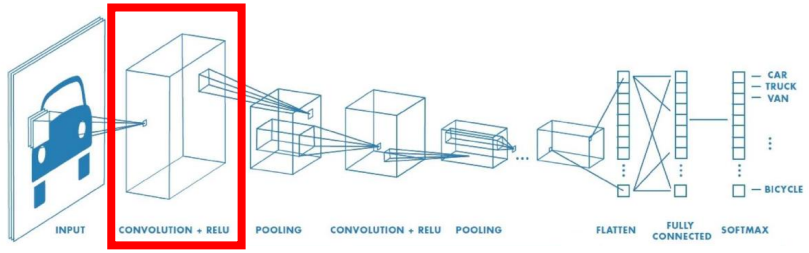
How do Convolutional Neural Networks (CNNs) work?





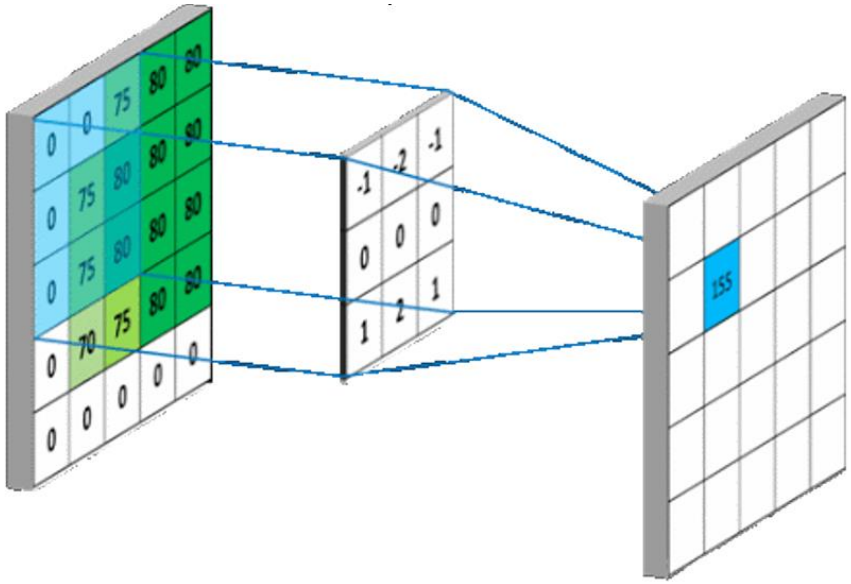
CNNs extract relevant features from an image.
Like a magnifying glass “enhancing” details of the image.

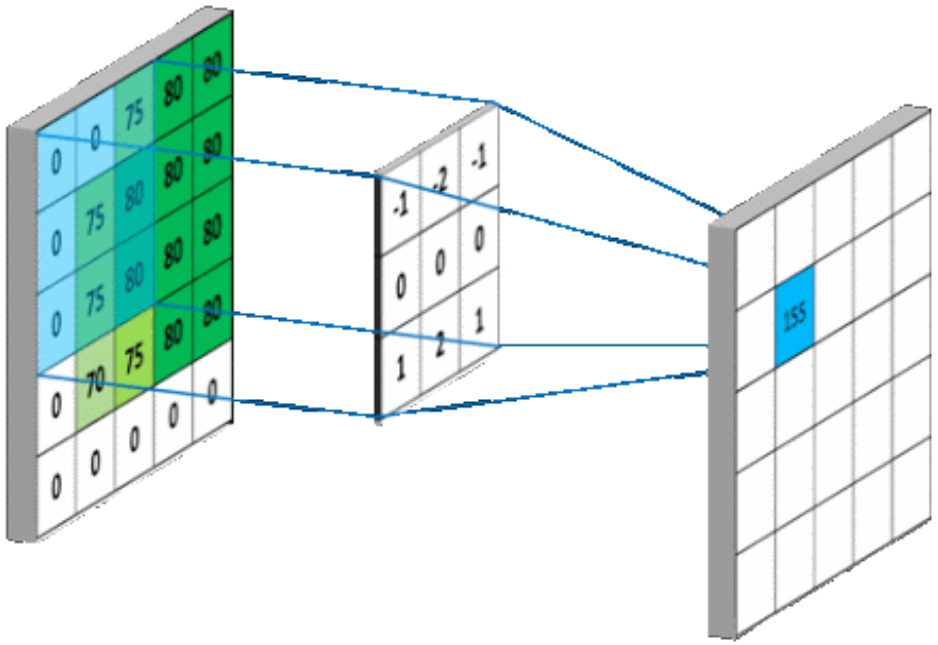
Convolutional layer



Convolution
Input: 5x5
Filter: 3x3
Output: 3x3

Filter / Kernel size = f x f





Convolutional layer

- Convolution operation
- Input: 5 x 5

- Filter / kernel size = 3 x 3

- Output = ?

10	2	4	1	1
0	4	0	2	2
1	8	16	0	0
8	1	1	0	0
0	0	2	6	1

*

2	2	2
1	0	1
2	2	2

Convolutional layer

- Convolution operation
- Input: 5 x 5
- Filter / kernel size = 3 x 3

10 x 2	2 x 2	4 x 2	1	1
0 x 1	4 x 0	0 x 1	2	2
1 x 2	8 x 2	16 x 2	0	0
8	1	1	0	0
0	0	2	6	1

$$10 \times 2 + 2 \times 2 + 4 \times 2 + 0 \times 1 + 4 \times 0 + 0 \times 1 + 1 \times 2 + 8 \times 2 + 16 \times 2 = 82$$

- Output = ?

82		

Convolutional layer

- Convolution operation
- Input: 5 x 5
- Filter / kernel size = 3 x 3

10	2 x 2	4 x 2	1 x 2	1
0	4 x 1	0 x 0	2 x 1	2
1	8 x 2	16 x 2	0 x 2	0
8	1	1	0	0
0	0	2	6	1

$$2 \times 2 + 4 \times 2 + 1 \times 2 + 4 \times 1 + 0 \times 0 + 2 \times 1 + 8 \times 2 + 16 \times 2 + 0 \times 2 = 68$$

- Output = ?

82	68	

Convolutional layer

- Convolution operation
- Input: 5 x 5
- Filter / kernel size = 3 x 3

10	2	4 x 2	1 x 2	1 x 2
0	4	0 x 1	2 x 0	2 x 1
1	8	16 x 2	0 x 2	0 x 2
8	1	1	0	0
0	0	2	6	1

$4 \times 2 + 1 \times 2 + 1 \times 2 +$
$0 \times 1 + 2 \times 0 + 2 \times 1 +$
$16 \times 2 + 0 \times 2 + 0 \times 2 = 46$

- Output = ?

82	68	46

Convolutional layer

- Convolution operation
- Input: 5 x 5
- Filter / kernel size = 3 x 3

10	2	4	1	1
0 x 2	4 x 2	0 x 2	2	2
1 x 1	8 x 0	16 x 1	0	0
8 x 2	1 x 2	1 x 2	0	0
0	0	2	6	1

$0 \times 2 + 4 \times 2 + 0 \times 2 + 1 \times 1 + 8 \times 0 + 16 \times 1 + 8 \times 2 + 1 \times 2 + 1 \times 2 = 46$

- Output = ?

82	68	46
45		

Convolutional layer

- Convolution operation
- Input: 5 x 5
- Filter / kernel size = 3 x 3

- Output = 3 x 3

10	2	4	1	1
0	4	0	2	2
1	8	16	0	0
8	1	1	0	0
0	0	2	6	1

*

2	2	2
1	0	1
2	2	2

=

82	68	46
45	24	26
63	65	51

Convolutional layer

Original

Sharpen

Edge Detect

"Strong" Edge Detect

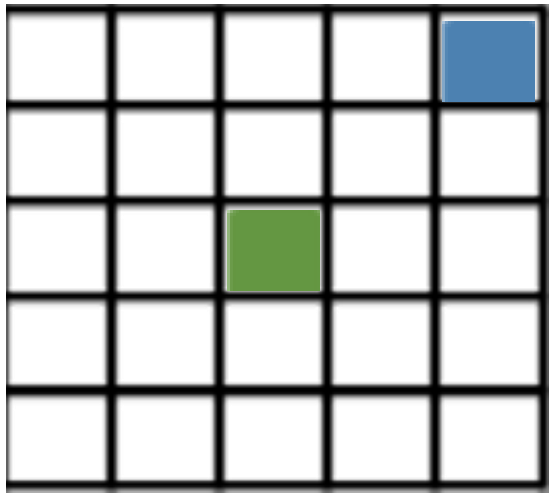
w1	w2	w3
w4	w5	w6
w7	w8	w9

Learnable filter[1]

1. The filter weights are like the weights in Feed Forward Networks (FFN). Similar to FFN, a bias term is added after applying the filter weights.

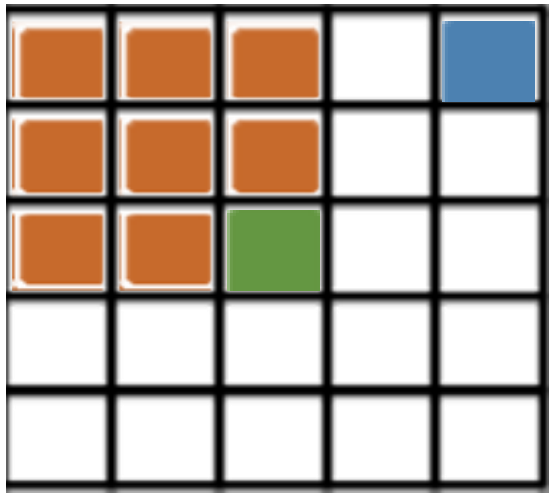
Convolutional layer - Padding

- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 0**
- **Number of times green pixel is in the filter’s receptive field: 0**



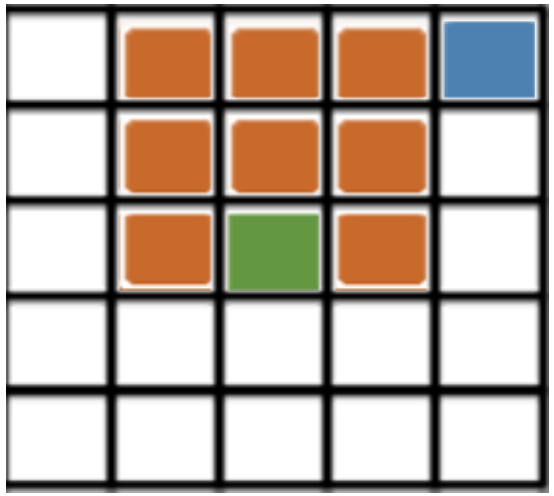
Convolutional layer - Padding

- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 0**
- **Number of times green pixel is in the filter’s receptive field: 1**



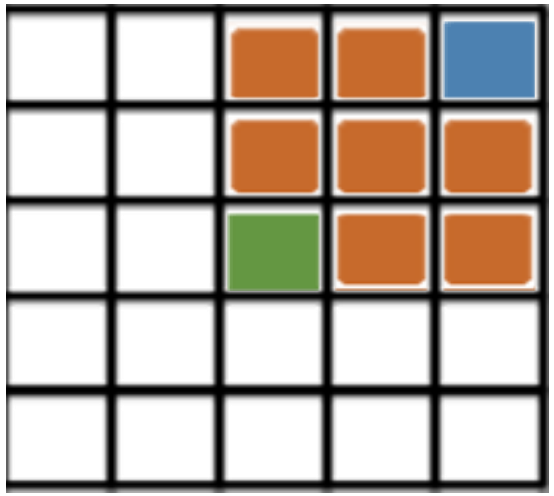
Convolutional layer - Padding

- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 0**
- **Number of times green pixel is in the filter’s receptive field: 2**



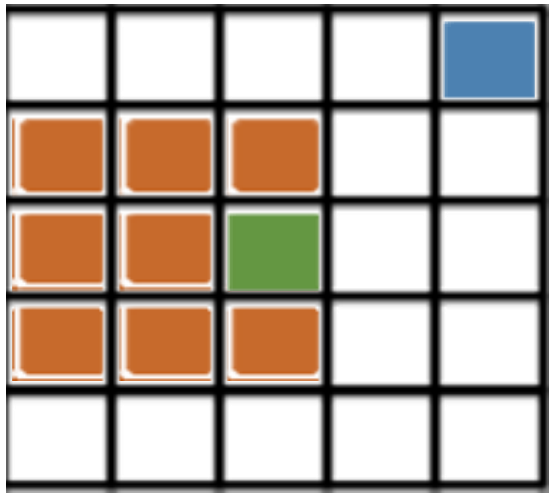
Convolutional layer - Padding

- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 1**
- **Number of times green pixel is in the filter’s receptive field: 3**



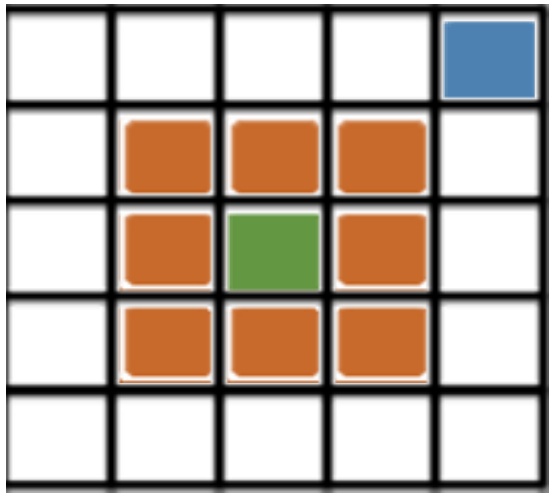
Convolutional layer - Padding

- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 1**
- **Number of times green pixel is in the filter’s receptive field: 4**



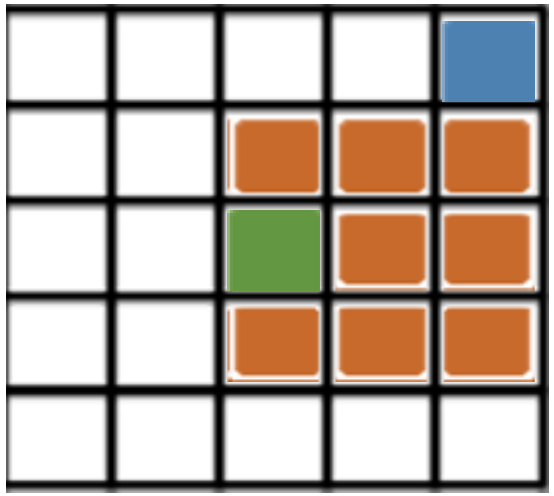
Convolutional layer - Padding

- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 1**
- **Number of times green pixel is in the filter’s receptive field: 5**



Convolutional layer - Padding

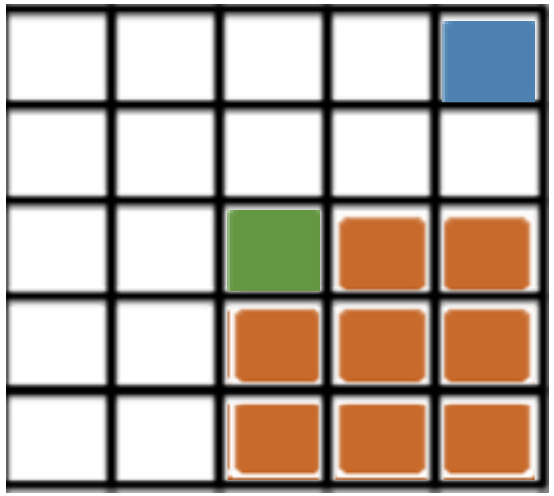
- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 1**
- **Number of times green pixel is in the filter’s receptive field: 6**



Convolutional layer - Padding

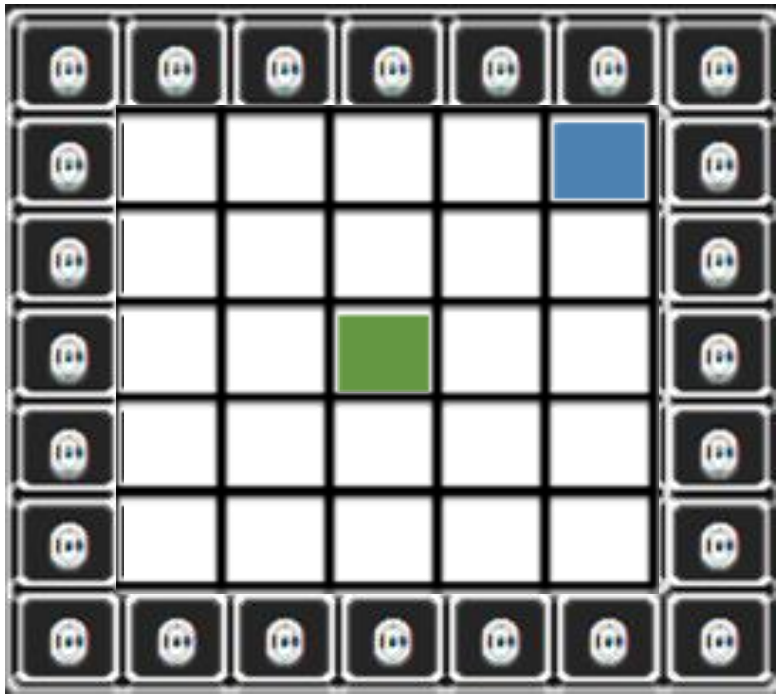
- Without Padding – corners / edges are not “seen” enough
- **Number of times blue pixel is in the filter’s receptive field: 1**
- **Number of times green pixel is in the filter’s receptive field: 9**

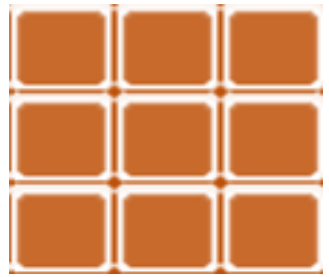
… Skip to the end



Convolutional layer - Padding

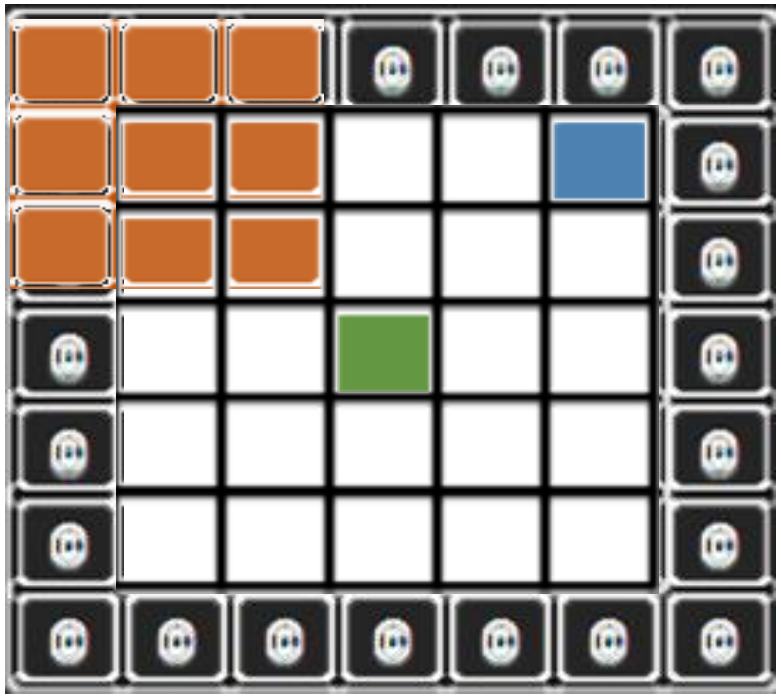
- **With Padding**
- Number of times blue pixel is in the filter's receptive field: 0
- Number of times green pixel is in the filter's receptive field: 0





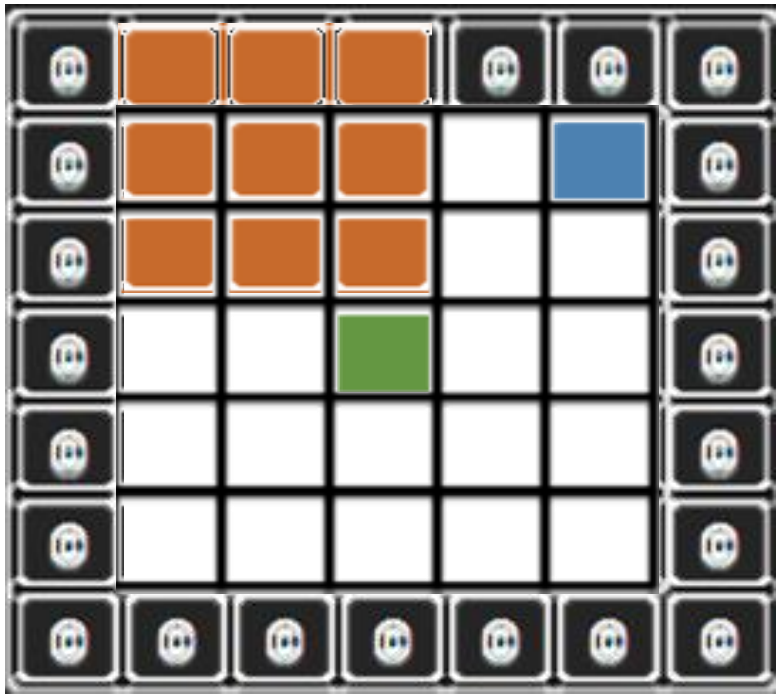
Convolutional layer - Padding

- **With Padding**
- Number of times blue pixel is in the filter's receptive field: 0
- Number of times green pixel is in the filter's receptive field: 0



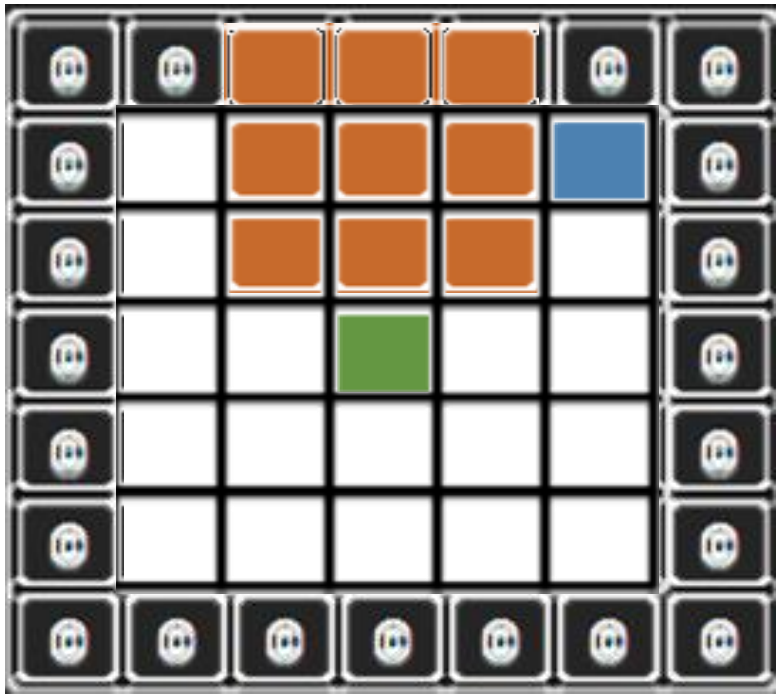
Convolutional layer - Padding

- **With Padding**
- Number of times blue pixel is in the filter's receptive field: 0
- Number of times green pixel is in the filter's receptive field: 0



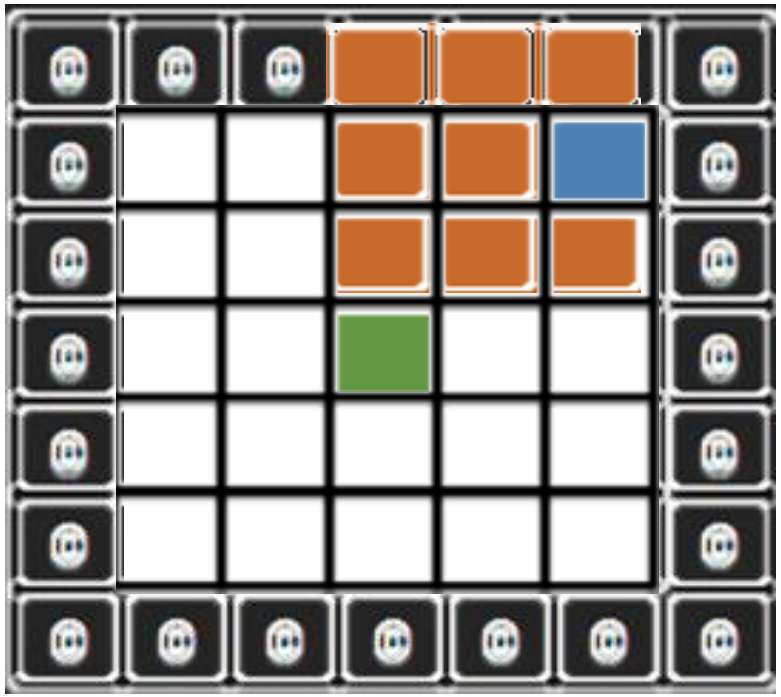
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 0**
- **Number of times green pixel is in the filter's receptive field: 0**



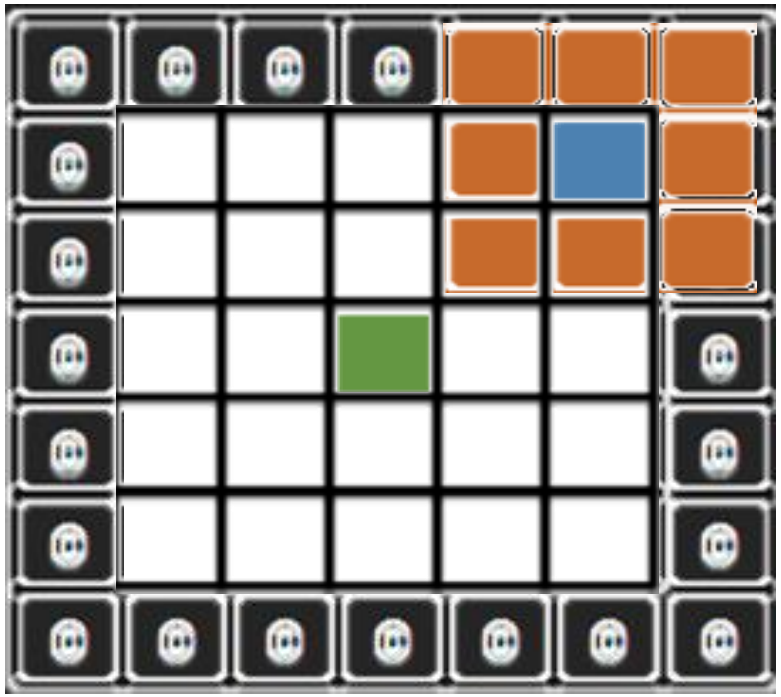
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 1**
- **Number of times green pixel is in the filter's receptive field: 0**



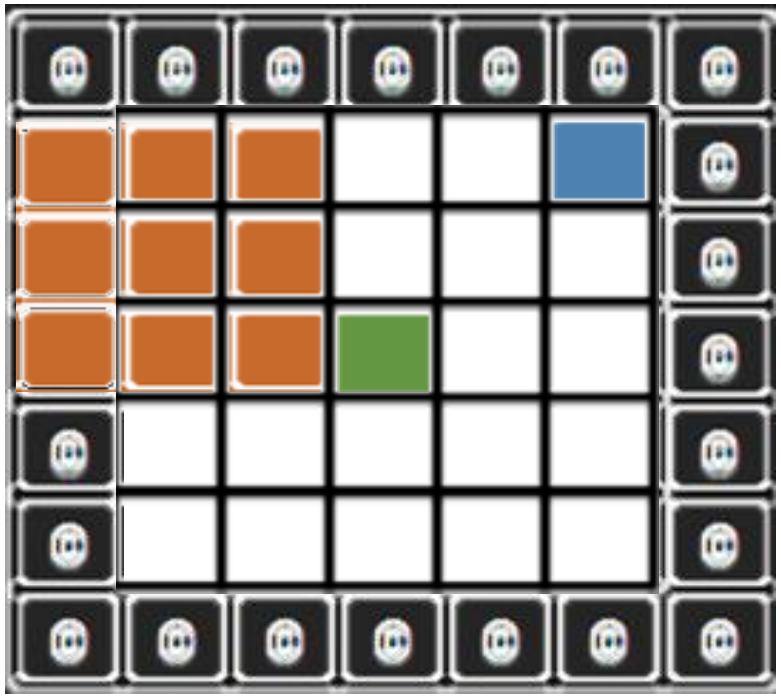
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 2**
- **Number of times green pixel is in the filter's receptive field: 0**



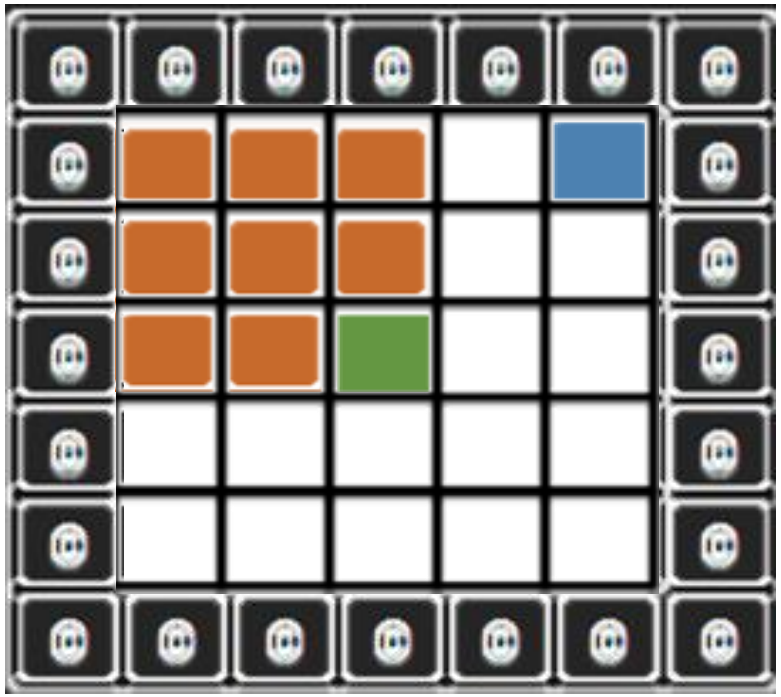
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 2**
- **Number of times green pixel is in the filter's receptive field: 0**



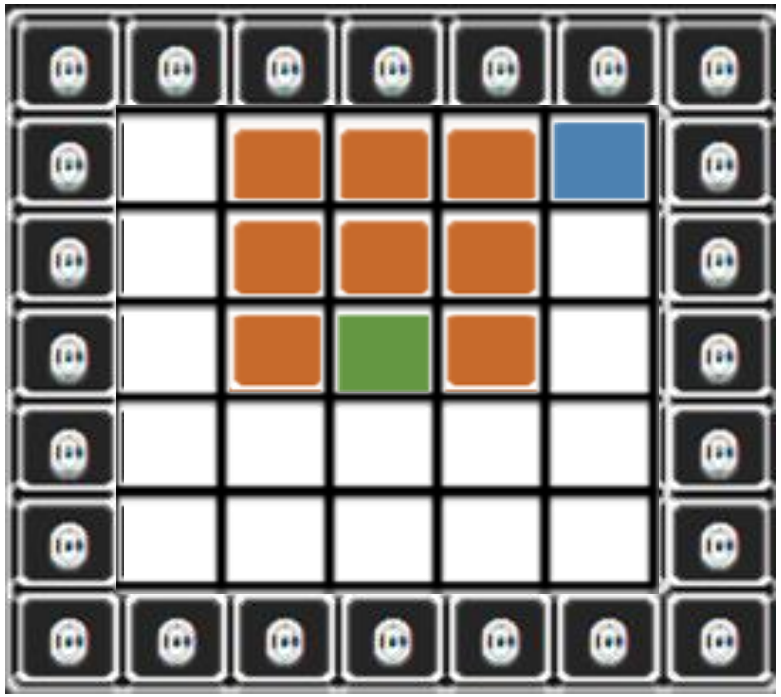
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 2**
- **Number of times green pixel is in the filter's receptive field: 1**



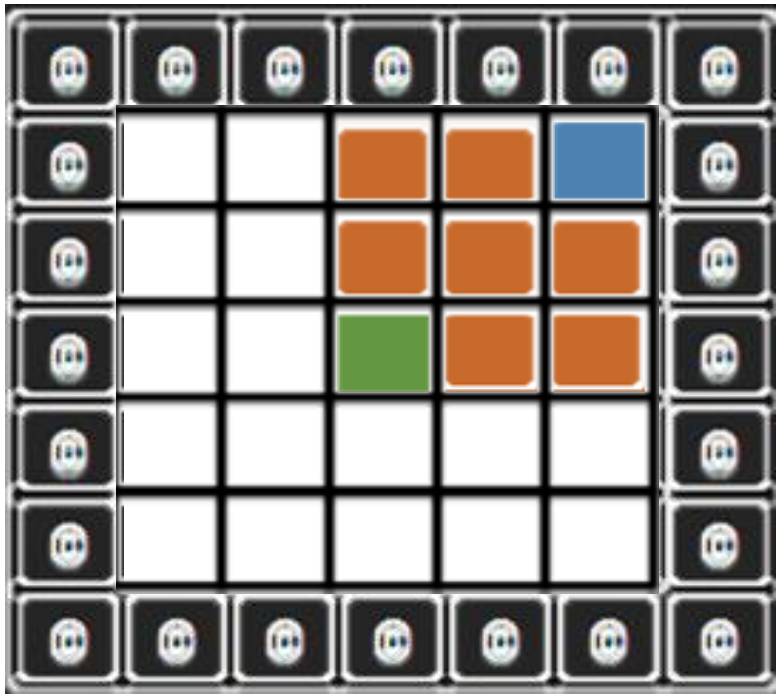
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 2**
- **Number of times green pixel is in the filter's receptive field: 2**



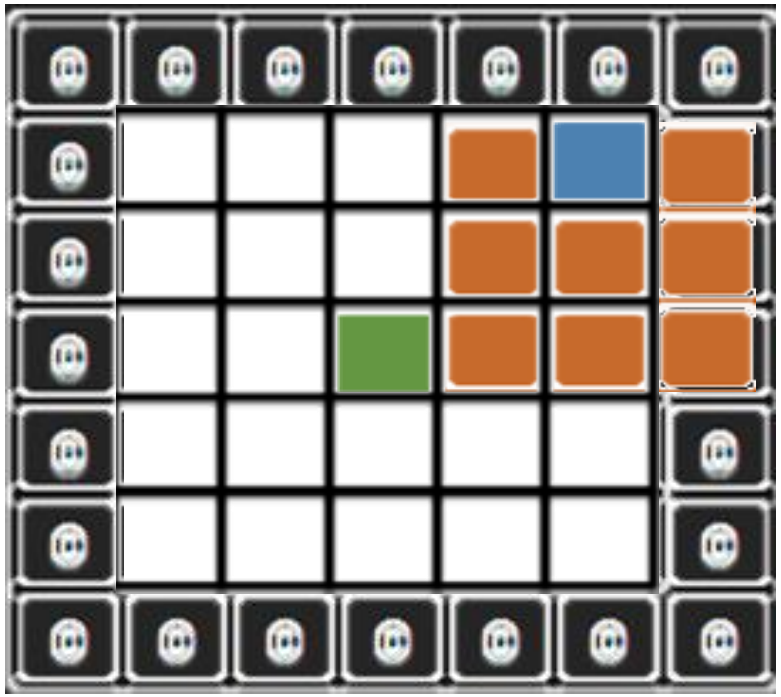
Convolutional layer - Padding

- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 3**
- **Number of times green pixel is in the filter's receptive field: 3**



Convolutional layer - Padding

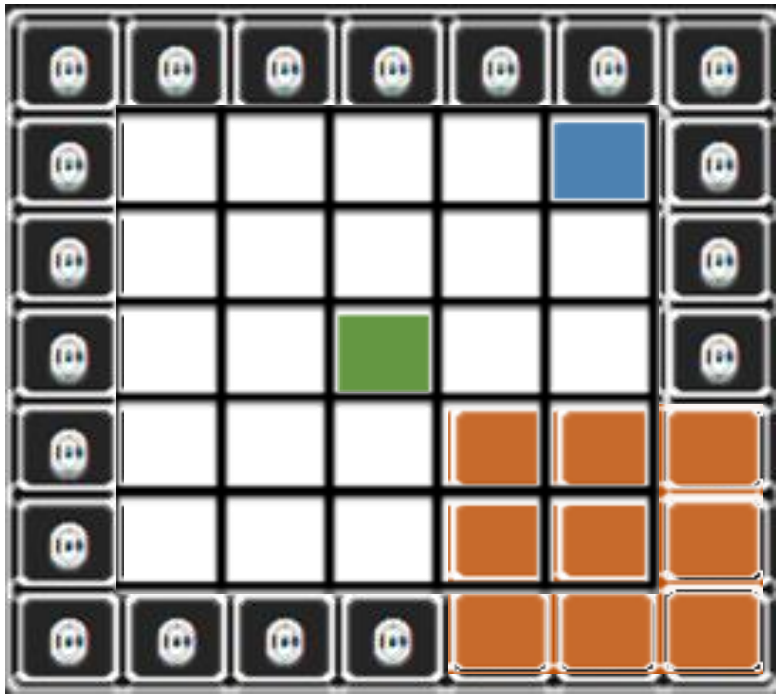
- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 4**
- **Number of times green pixel is in the filter's receptive field: 3**



Convolutional layer - Padding

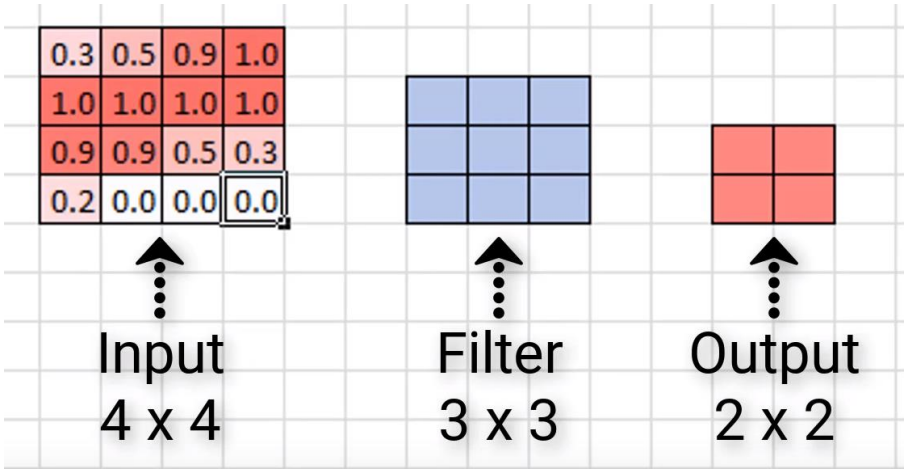
- **With Padding**
- **Number of times blue pixel is in the filter's receptive field: 4**
- **Number of times green pixel is in the filter's receptive field: 9**

... Skip to the end

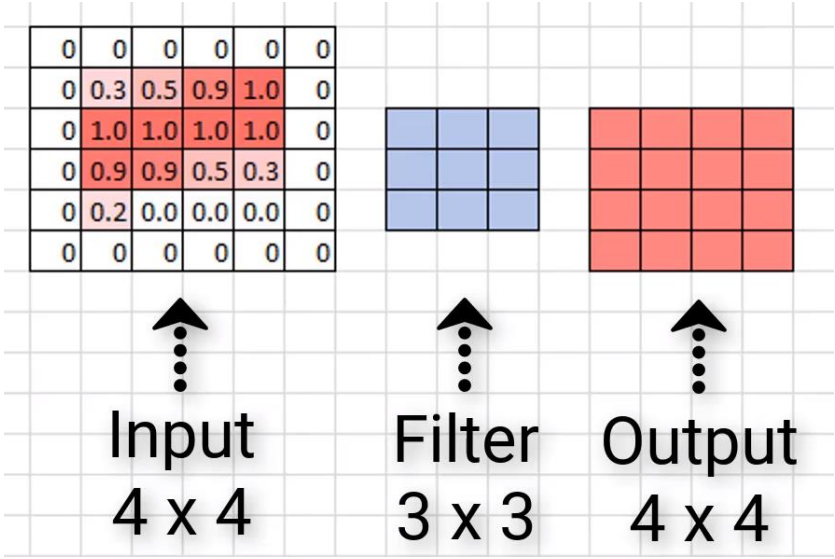


Convolutional layer - Padding

Valid Padding (No Padding)
p = 0

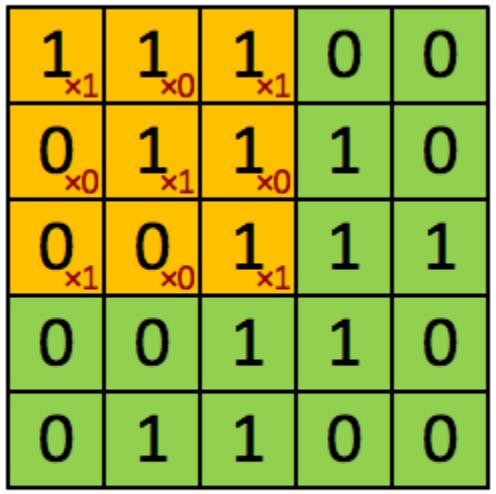


Same Padding
p ≠ 0

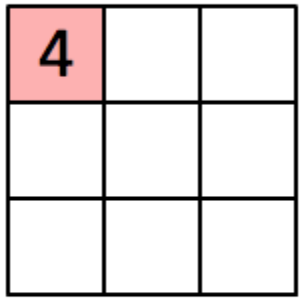


Convolutional layer - Stride

Stride
s = 1

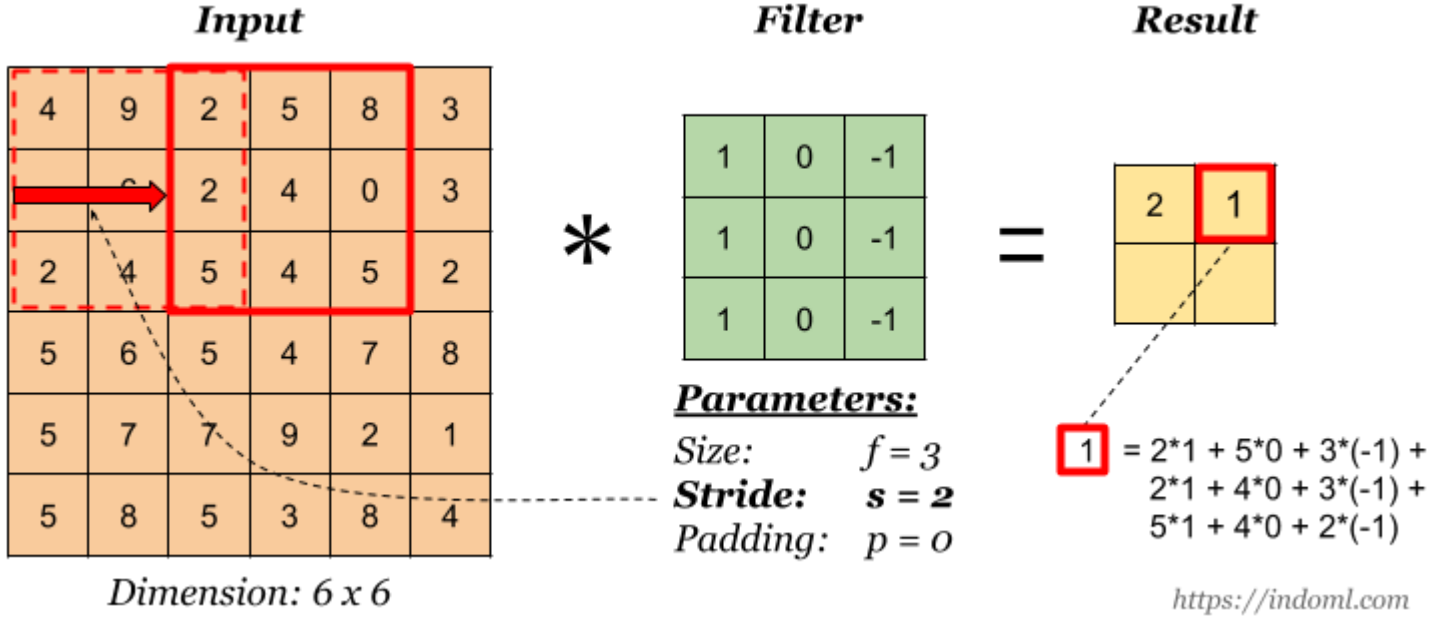


Image

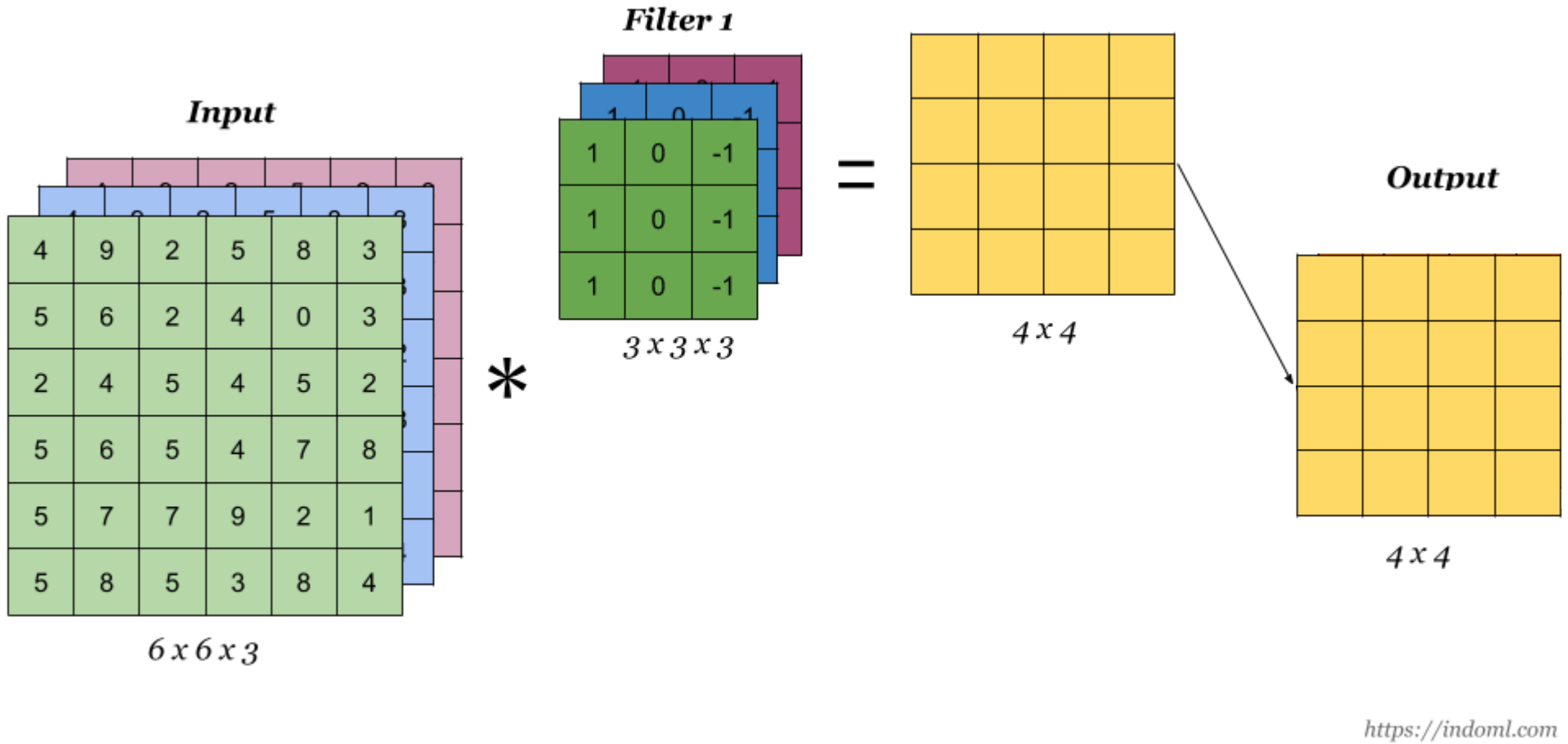


Convolved Feature

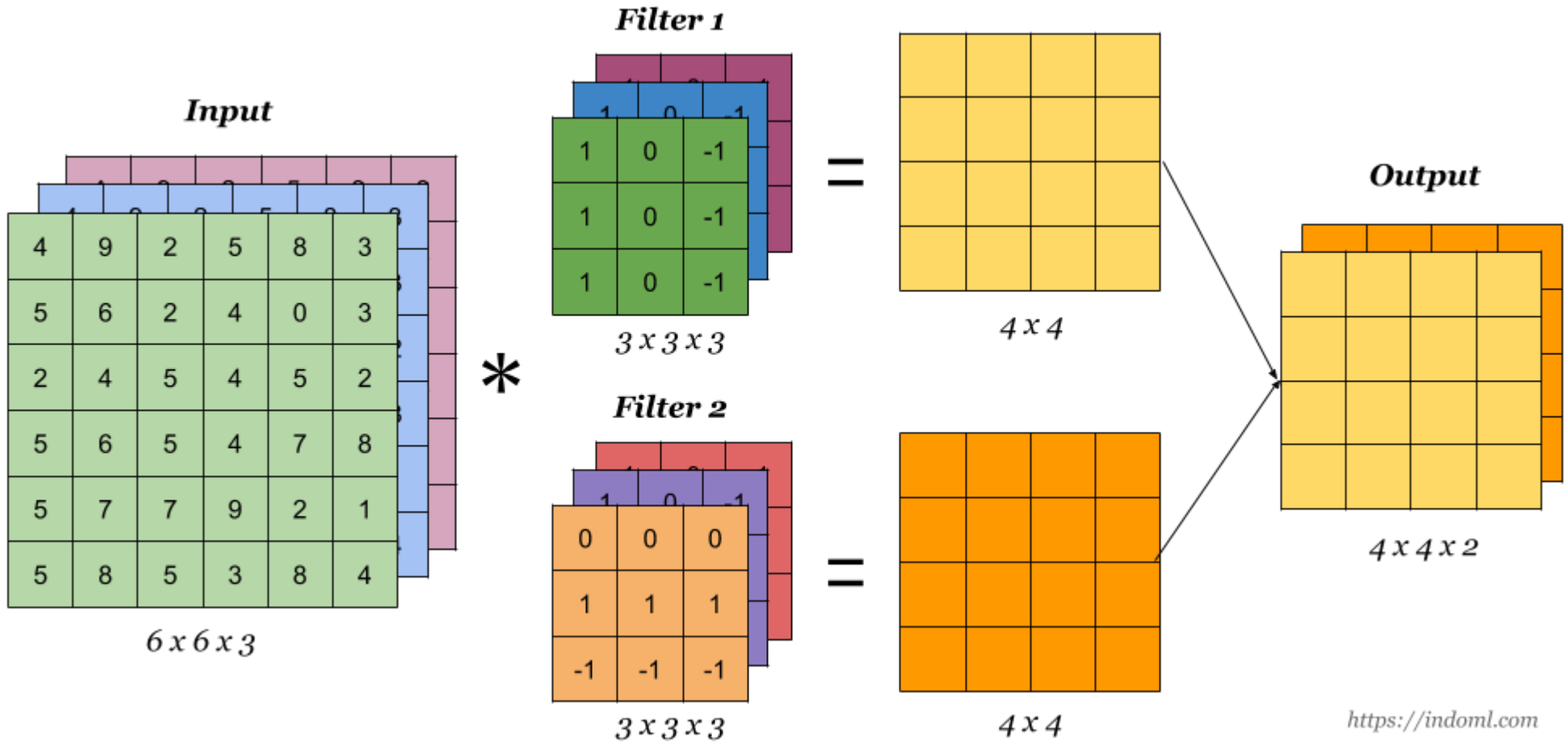
Stride
s = 2



Convolutional layer – Multiple filters



Convolutional layer – Multiple filters



Convolutional Layer

Summary of convolutions

n × m **image** f × f **filter** c **channels**

padding p **stride** s

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{m + 2p - f}{s} + 1 \right\rfloor \times \mathbf{c}$$

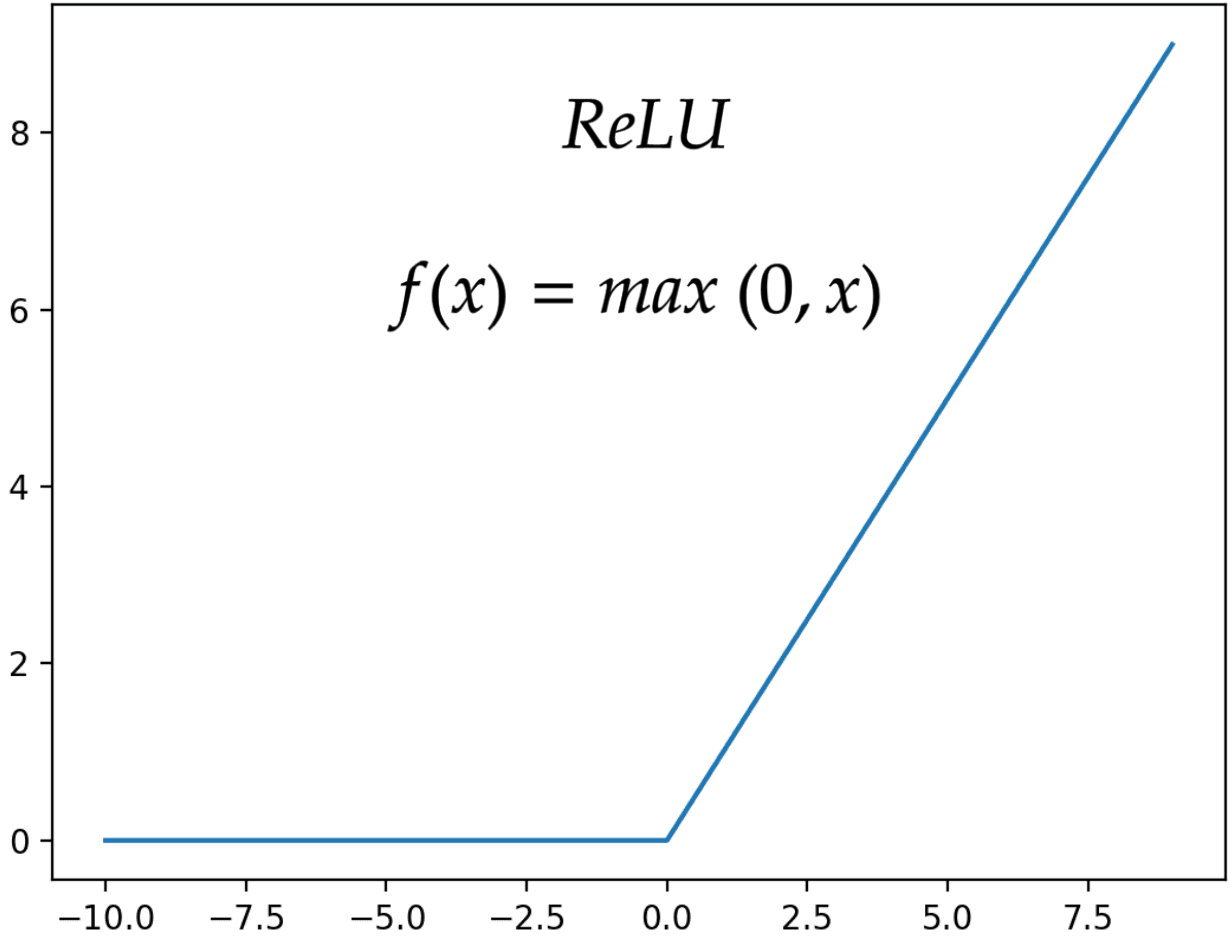
Example:

Input Image = 224x224x3

Filter = 3x3x3 (x32)
n = 224; p = 0; s = 1; f = 3; c_o = 32

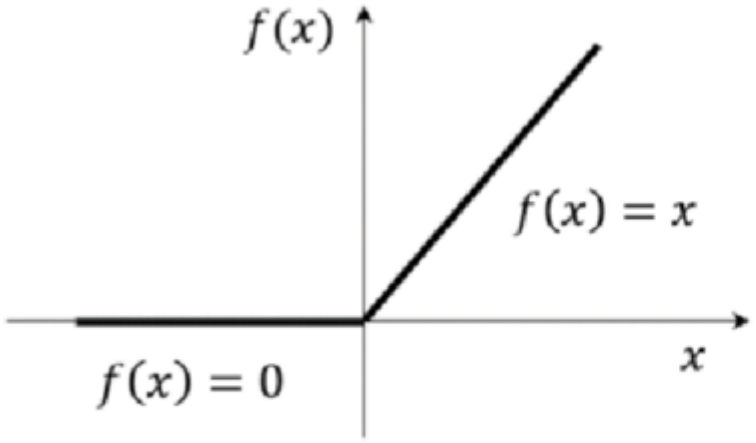
Output = 222x222x32

ReLU activation function



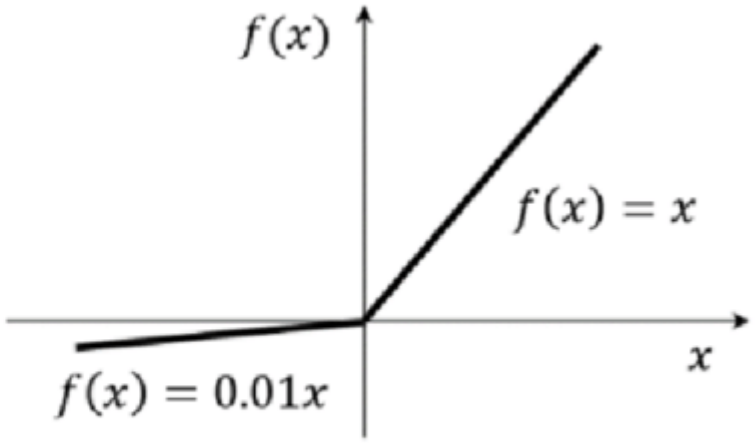
LeakyReLU activation function

$$ReLU = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$



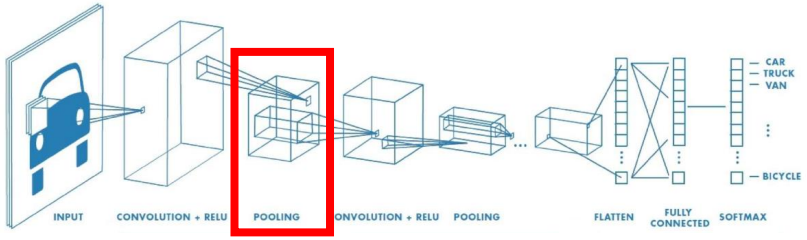
ReLU activation function

$$LealyReLU = \begin{cases} \gamma x, & x \leq 0 \\ x, & x > 0 \end{cases}$$



LeakyReLU activation function

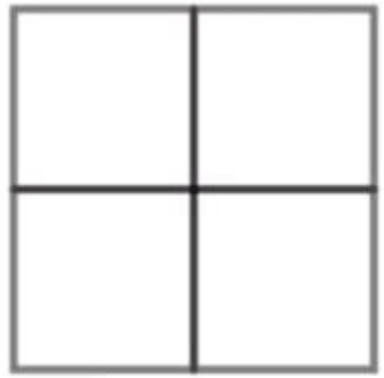
Pooling layer – Max Pooling



Max Pooling
Filter: 2x2
Stride: 2

4	6	1	1
1	3	1	3
4	0	0	8
8	5	4	0

Input (4x4)

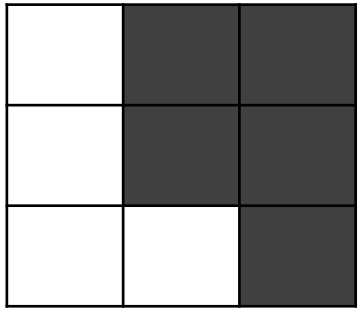
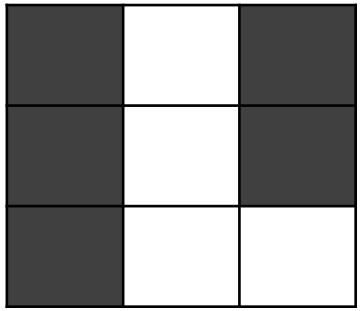


Output (2x2)

Convolutional layer – Convolution properties

- **Translation Equivariant**

$$f(t(x)) = t(f(x)),$$
where **t** is a transformation

	Input 3x3			Filter 2x2			Output 2x2	
	20	0	0	2	0	=	40	0
	20	0	0	0	2		80	0
	20	20	0					
	0	20	0	2	0	=	40	40
	0	20	0	0	2		40	80
	0	20	20					

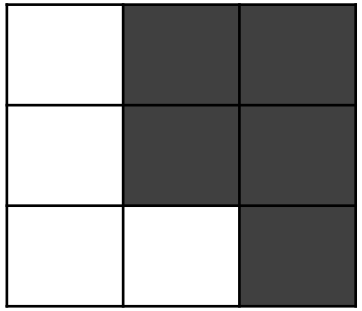
Both the **input** and the **output** shifted to the **right**!

1. Translation equivariance means that the model responds predictably to translations, while translation invariance means that the model produces the same output regardless of translations

Pooling layer – Max Pooling properties

- **Translation Invariant**

$f(t(x)) = f(x)$,
where **t** is a transformation



Input 3x3		
20	0	0
20	0	0
20	20	0

*

Filter 2x2	
2	0
0	2

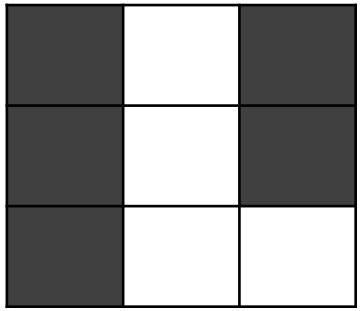
=

Output 2x2	
40	0
80	0

Pooling 2x2 →

80

Change in the **input** results in the **same output**



0	20	0
0	20	0
0	20	20

*

2	0
0	2

=

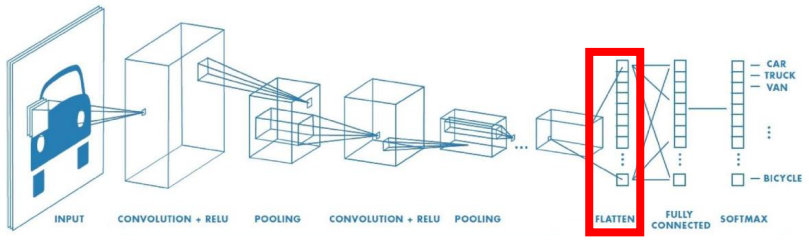
40	40
40	80

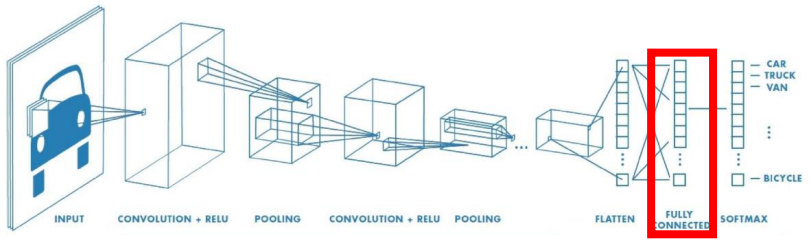
Pooling 2x2 →

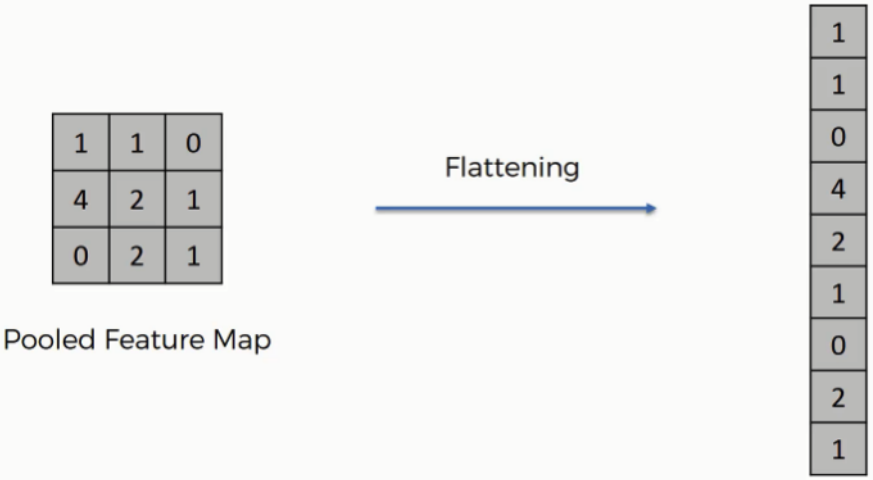
80

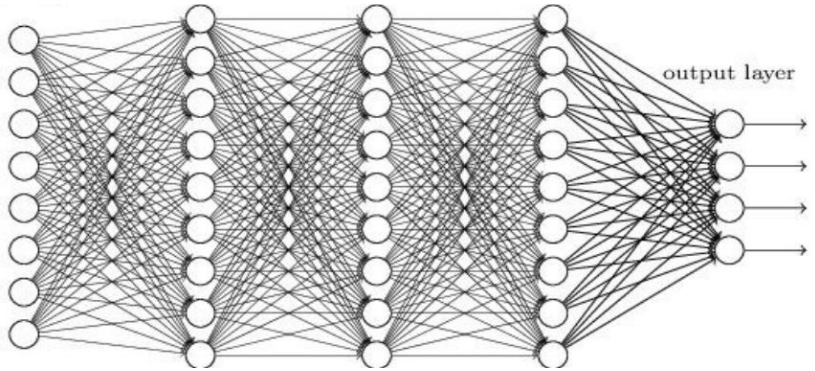
1. Translation equivariance means that the model responds predictably to translations, while translation invariance means that the model produces the same output regardless of translations

Flatten and Fully Connected layer





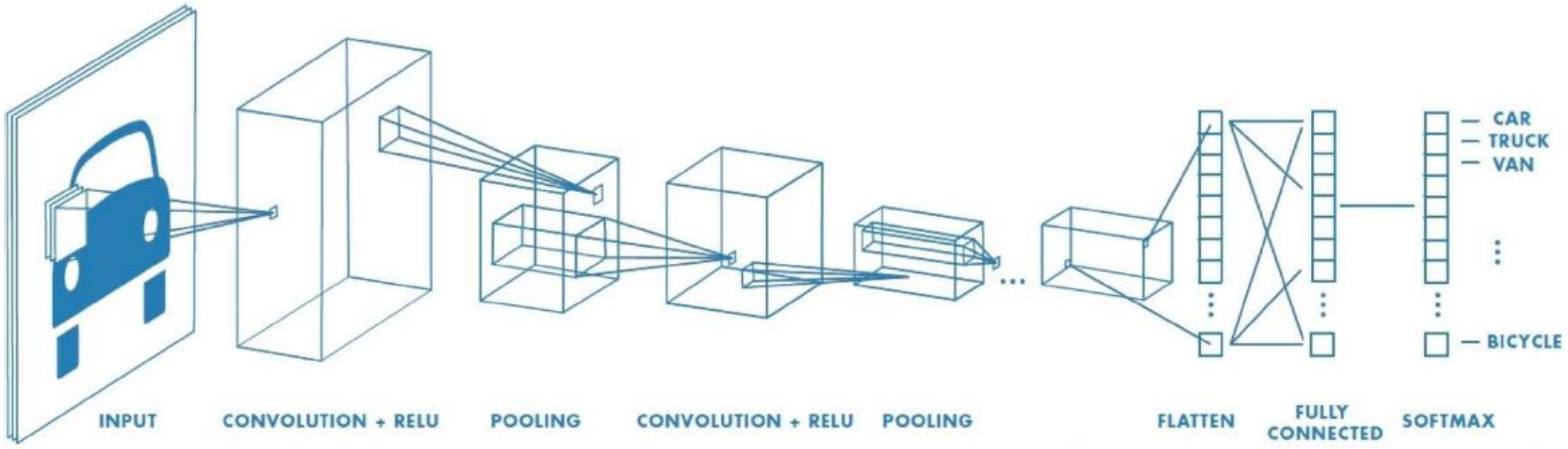




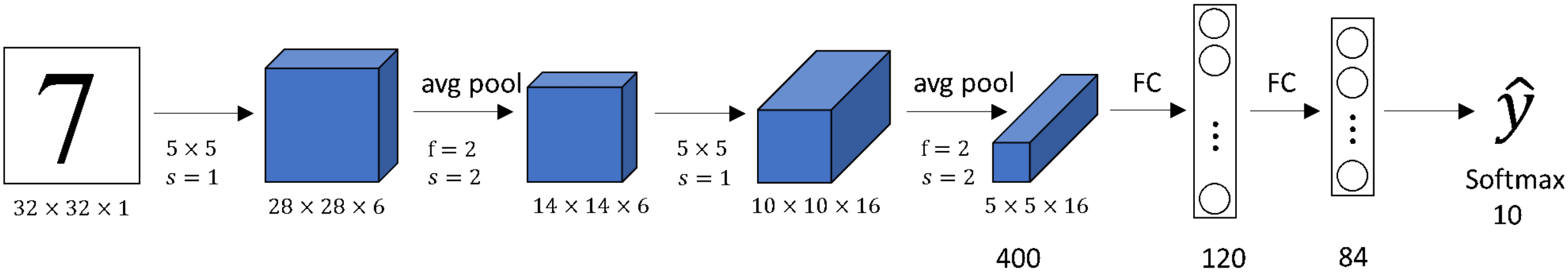
Convolutional Neural Network

Visualization:

- https://poloclub.github.io/cnn-explainer/
- https://distill.pub/2017/feature-visualization/

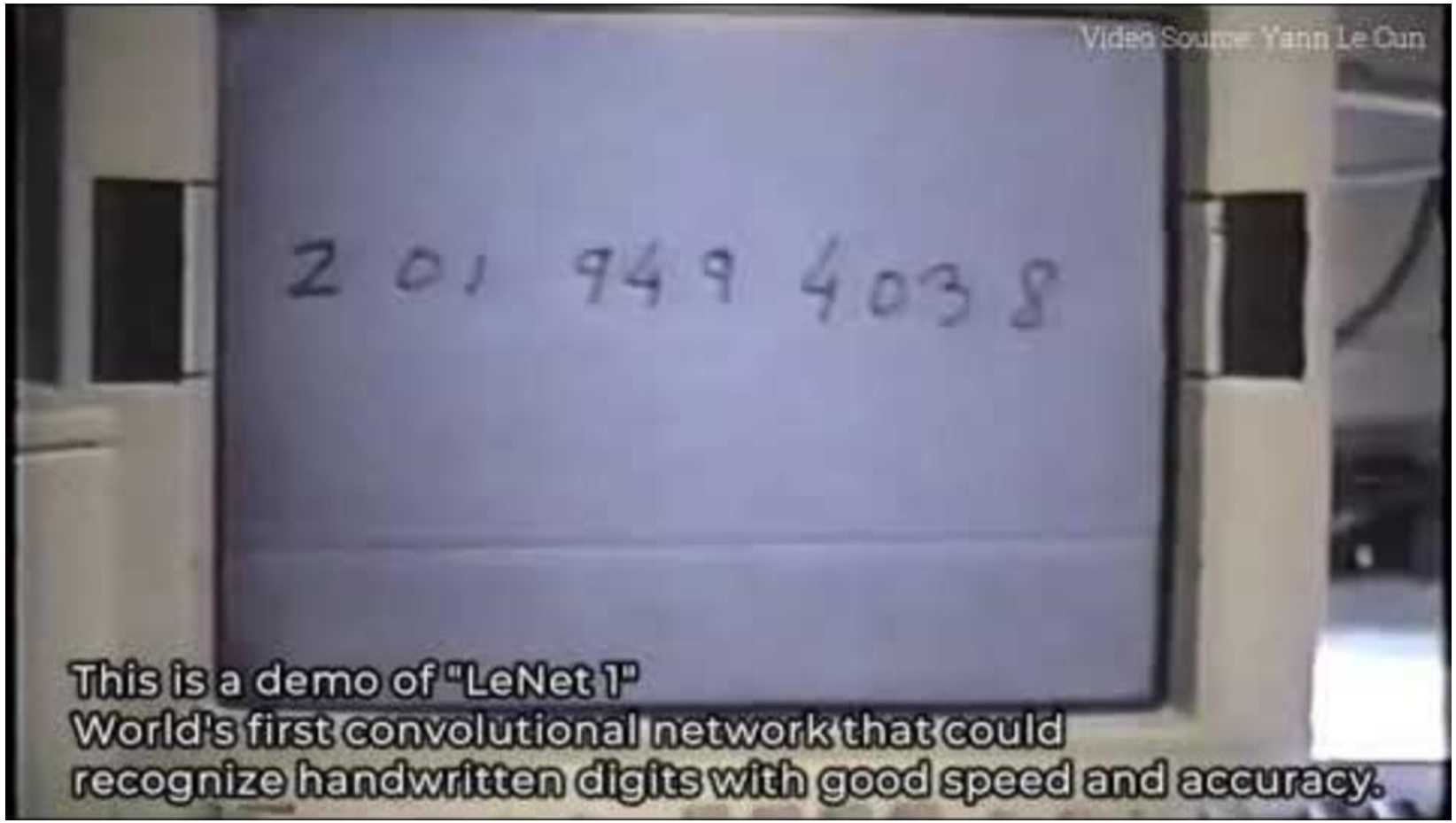


“LeNet 5” Y. Lecun et al. (1998) [3]



[3] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

“LeNet 5” Y. Lecun et al. (1998) [3]



[3] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

Deep Network Development

Lecture 7.

Image Classification Convolutional Neural Networks Transfer Learning

Budapest, 7th October 2025

1 Image Classification

2 Convolutional Neural Networks

3 CNN Architectures

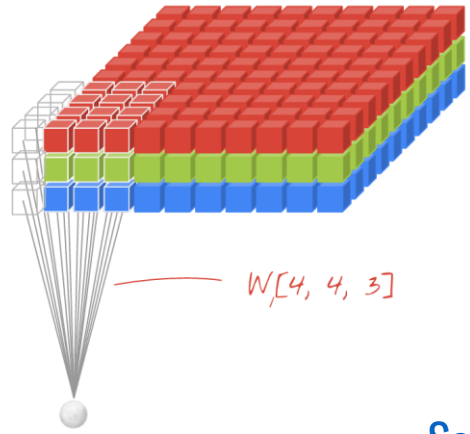
4 Transfer Learning

5 Autoencoders

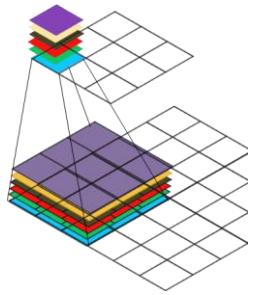
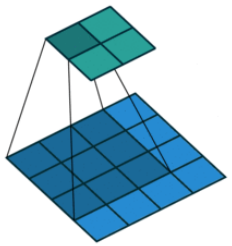
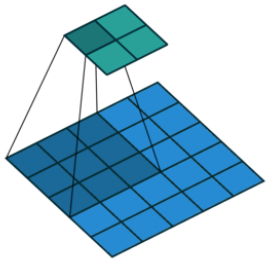
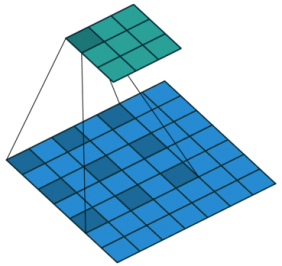
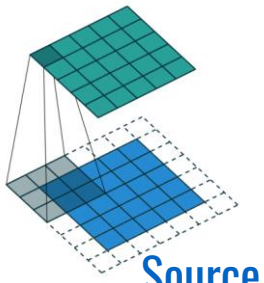
Convolutional Layer

- Purpose: To quantify the correspondence with the characteristics defined by the weights of the filters
- E.g.: how much of a (vertical edge)/nose/wheel/face is in the image part under consideration
- Play

Num. of filters	2 filters shown
Kernel size	4x4x3
Stride	1,1
Dilatation	1,1
Padding	0,0,1,2



Source

Filters: 6	**Kernel size** 3x3	**Stride** 2,2	**Dilatation** 2,2	**Padding** 1,1,1,1 (up, down, left, right)
				

Source

Convolutional Layer

Summary of convolutions

n × m **image** f × f **filter** c **channels** **padding** p **stride** s

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{m + 2p - f}{s} + 1 \right\rfloor \times \mathbf{C}$$

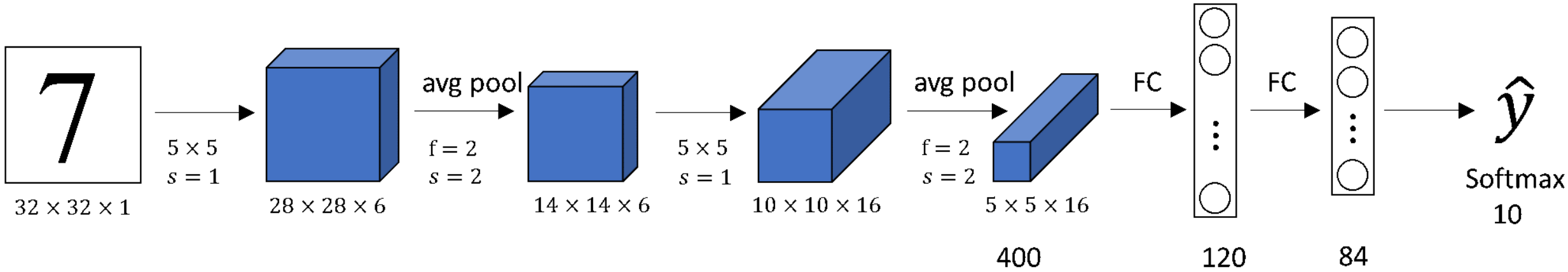
Example:

Input Image = 224x224x3

Filter = 3x3x3 (x32)
n = 224; p = 0; s = 1; f = 3; m = 32

Output = 222x222x32

“LeNet 5” Y. Lecun et al. (1998) [2]



[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

Going deeper

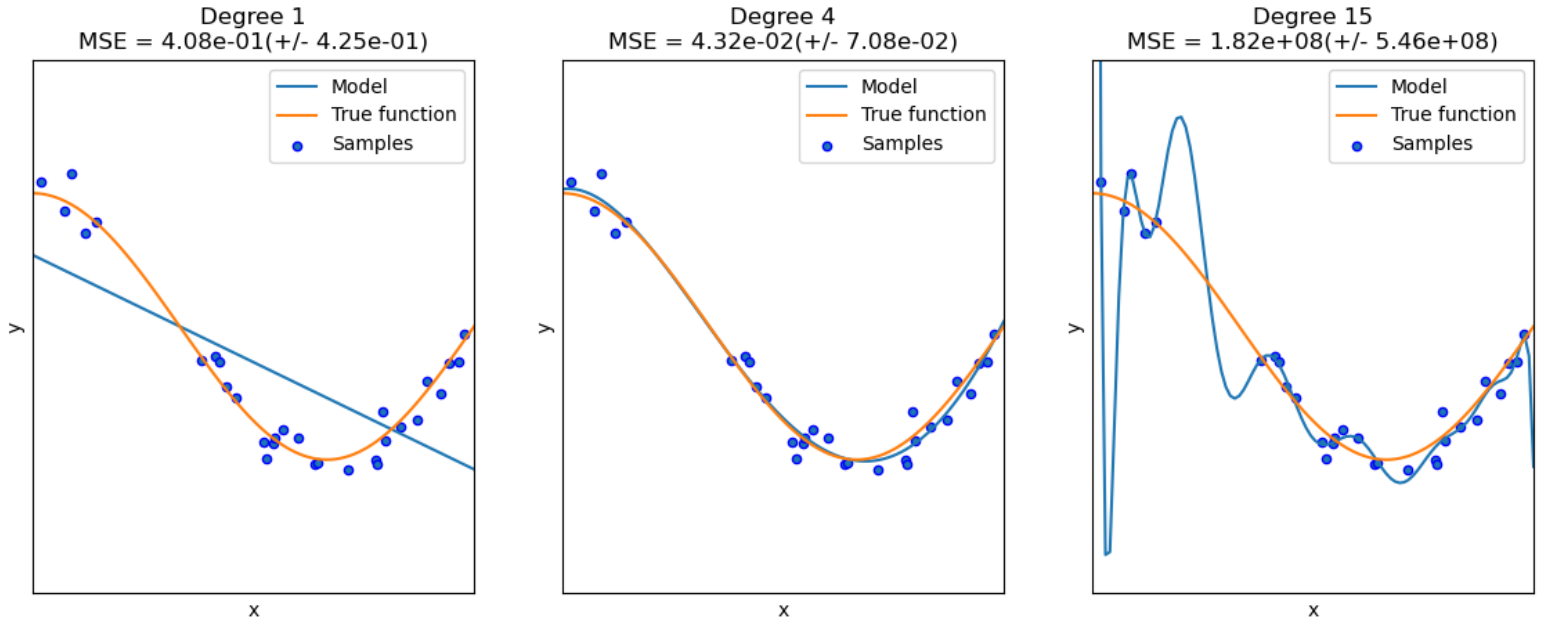
What prevents us from creating CNN with arbitrary depth?

- Underfitting/Overfitting
- Vanishing/Exploding gradient
- And many more...

Underfitting/Overfitting

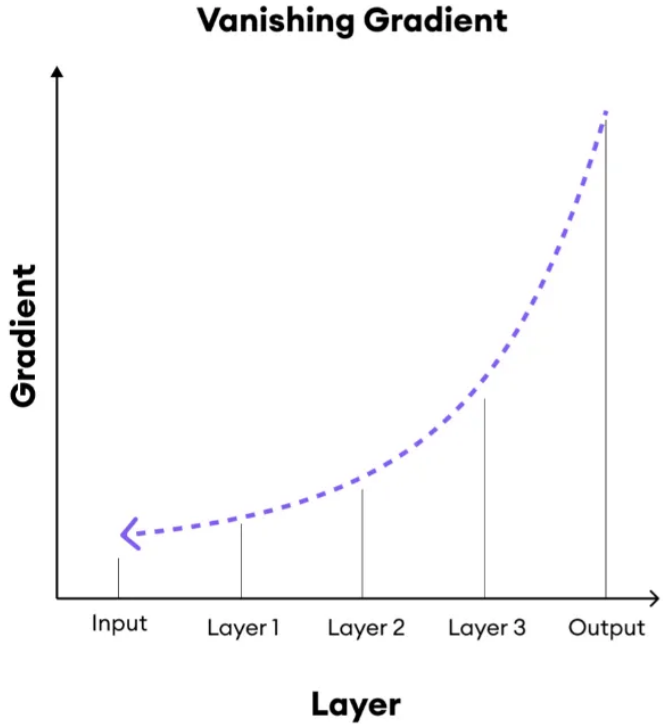
- Underfitting
  - we train the model for more epochs
  - If we have more parameter, we can counter this issue

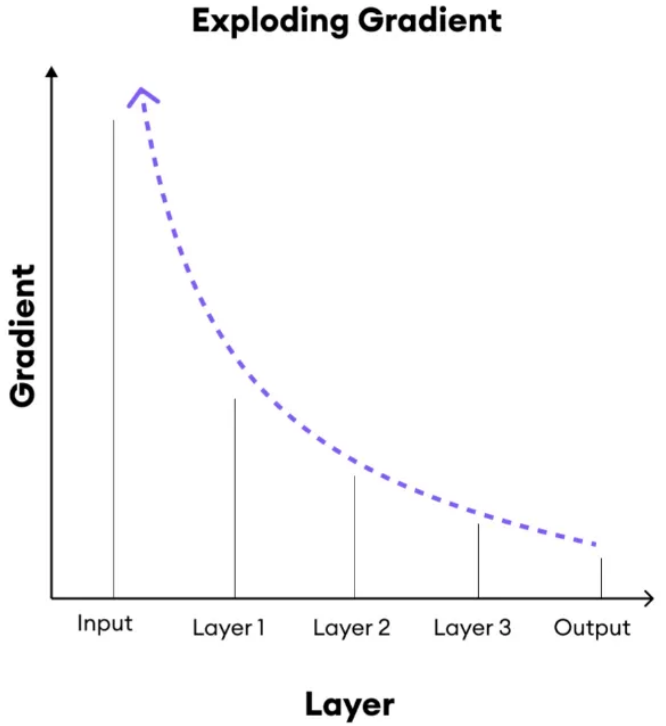
- Overfitting
  - Regularization (early-stopping, dropout, etc.)
  - If we have less parameters, it is less likely that our model will overfit



Vanishing/Exploding Gradient

- Weight initialization
- Activation function
- Regularization
  - Batch Normalization
  - Dropout
  - Weight decay
- “Skip connection”
- And many more…





Weight initialization

- As the name suggest specifies the method how we initialize the weight of our neural network
  - Sample from a uniform distribution in $[a, b]$
  - Sample from a normal distribution $N(\mu, \sigma)$ – usually with 0 mean and 1 standard deviation
  - Set it to some constant value
  - Or manually add the initial weights

- In pytorch the weights of the Convolution is sampled from $U(-\sqrt{k}, \sqrt{k})$, where $k = \frac{groups}{C_{in} * \prod_{i=0}^{1} kernel\_size[i]}$

- And in Linear layers is sampled from $U(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{in\_features}$

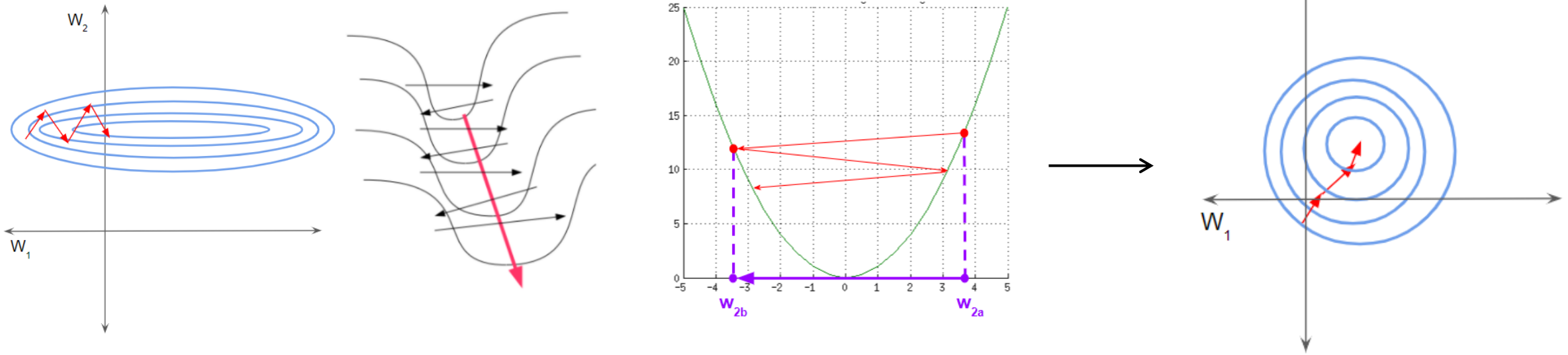
Regularization

- Weight related regularizations:
  - **L1 regularization** (also called LASSO) leads to sparse models by adding a penalty based on the absolute value of coefficients.
  - **L2 regularization** (also called ridge regression) encourages smaller, more evenly distributed weights by adding a penalty based on the square of the coefficients.
- Early stopping
- Data augmentation
- Batch normalization
- Dropout
- And many more...

Batch Normalization

- Batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs
- It basically normalizes the output of the previous layer in this batch
  - It also keeps track of moving averages, and can do some transformations too

$$y_B = \frac{x_B - \mathbf{E}[x_\mathrm{B}]}{\sqrt{\mathbf{Var}[x_\mathrm{B}]}}$$



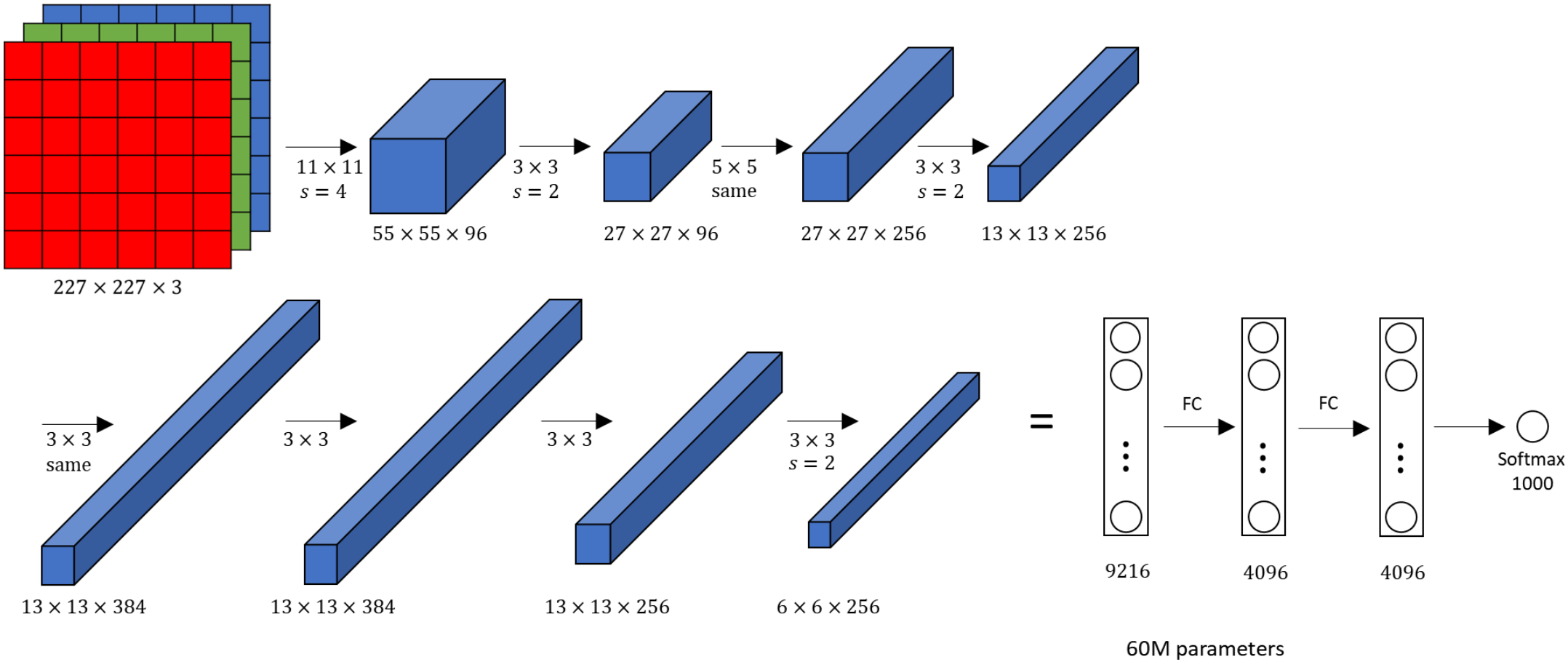
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [10]

- **1000 classes**
- **1M Images**

- **Serves as the common benchmark for neural nets**

[10] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

“AlexNet” A. Krizhevsky et al. (2012) [3]



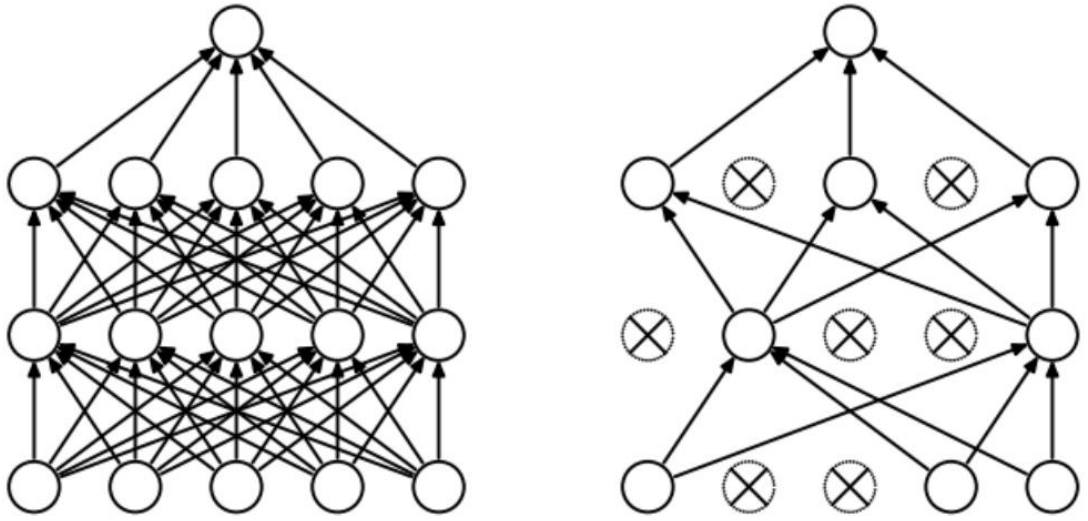
[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

“AlexNet” A. Krizhevsky et al. (2012) [3]

Dropout:

- We are dropping some neurons between layers with some probability **p**.
- The dropped out neurons do not contribute to the **forward pass** and do not participate in the **backpropagation.**
- “Every time we sample from a different architecture”

- Reduces neuron co-adaptation
- The model forced to learn more robust features



(a) Standard Neural Net

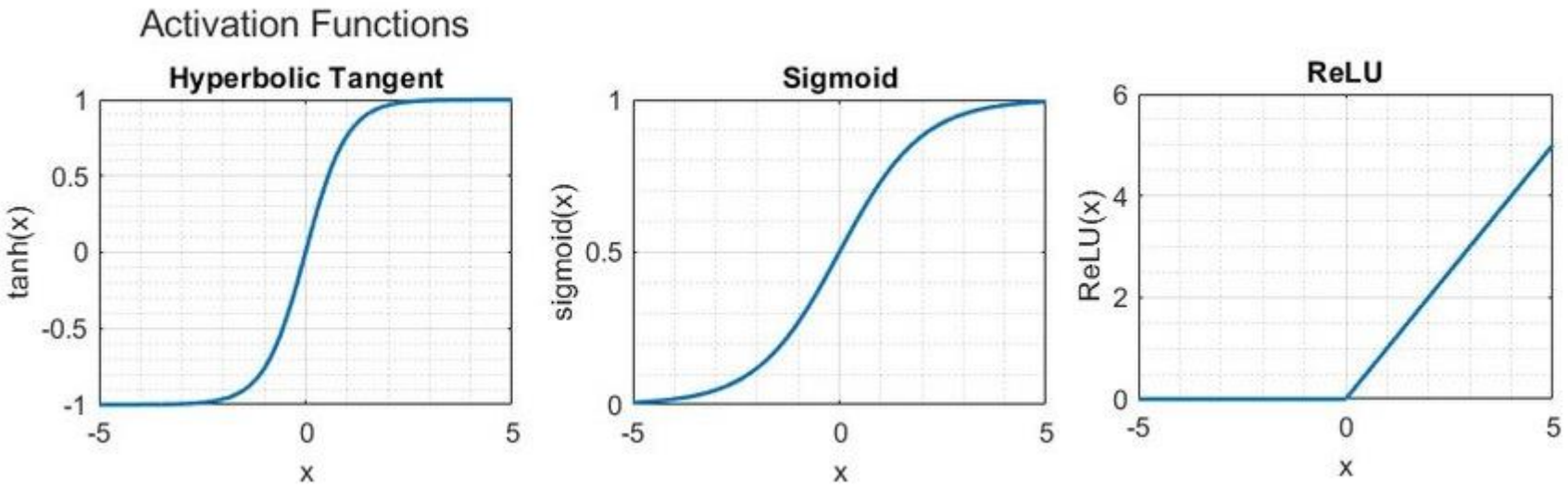
(b) After applying dropout.

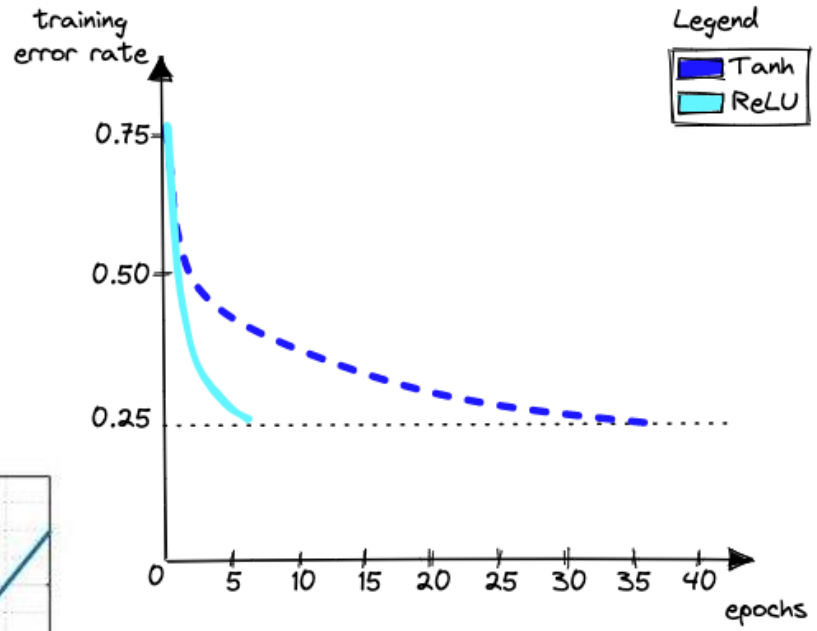
[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

“AlexNet” A. Krizhevsky et al. (2012) [3]

ReLU over Sigmoid or tanh:

- During training with **Gradient Descent** using **ReLU** (non-saturating nonlinearity) results in less training time compared to **Sigmoid** or **tanh** (saturating nonlinearity).
- Does not require input normalization
- Learning will happen if at least some training examples produce positive input

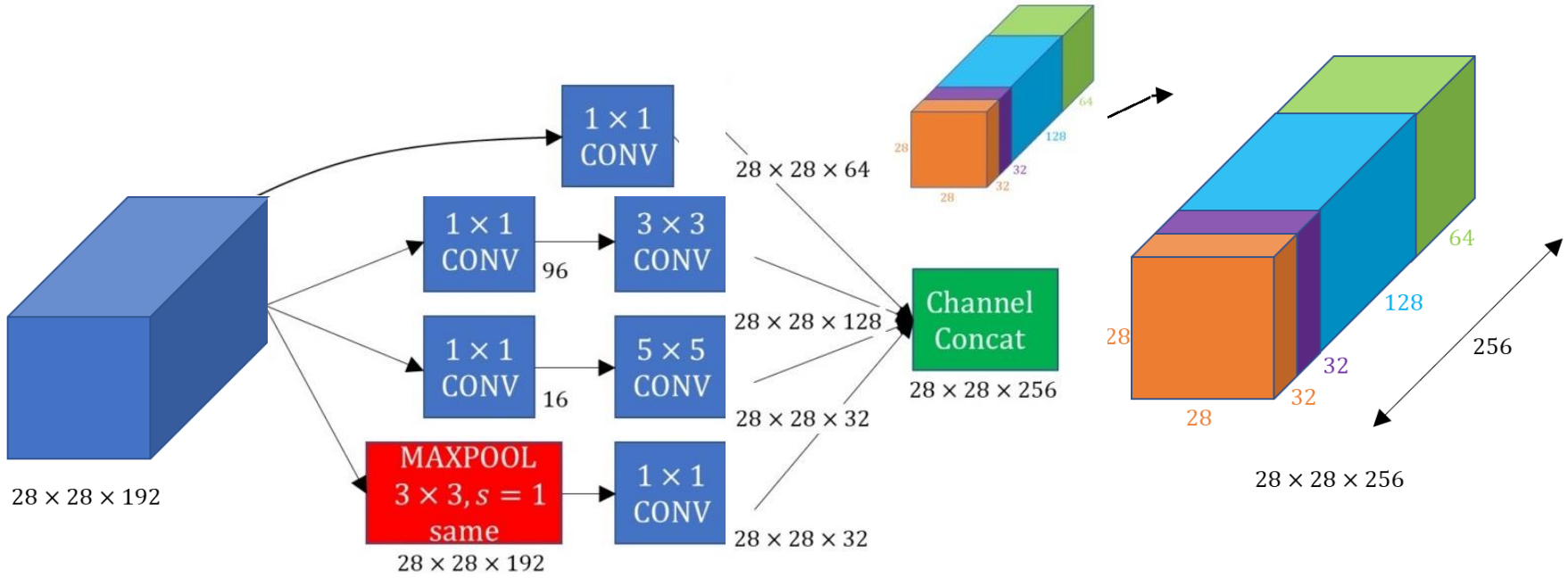




[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

“Inception / GoogLeNet” C. Szegedy et al. (2014) [5]

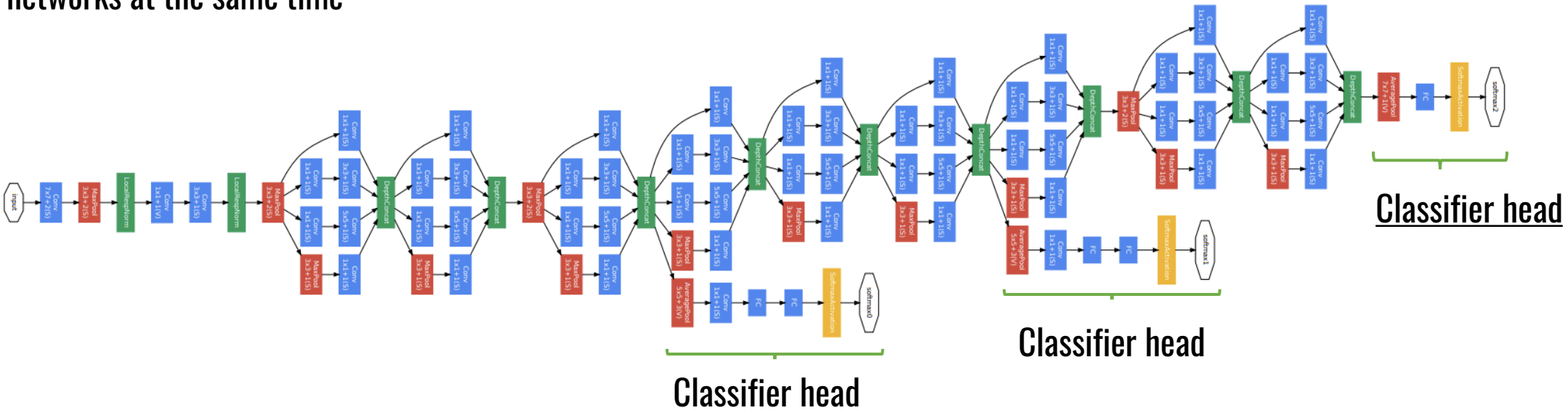
- Aligns with the intuition that visual information should be processed at various scales:
  - Be able to detect **bigger** and **smaller** objects on the image
- 1x1 Convolution blocks for dimensionality reduction



[5] Szegedy, C., et al. “Going Deeper with Convolutions”, arXiv e-prints, 2014. doi:10.48550/arXiv.1409.4842.

“Inception / GoogLeNet” C. Szegedy et al. (2014) [5]

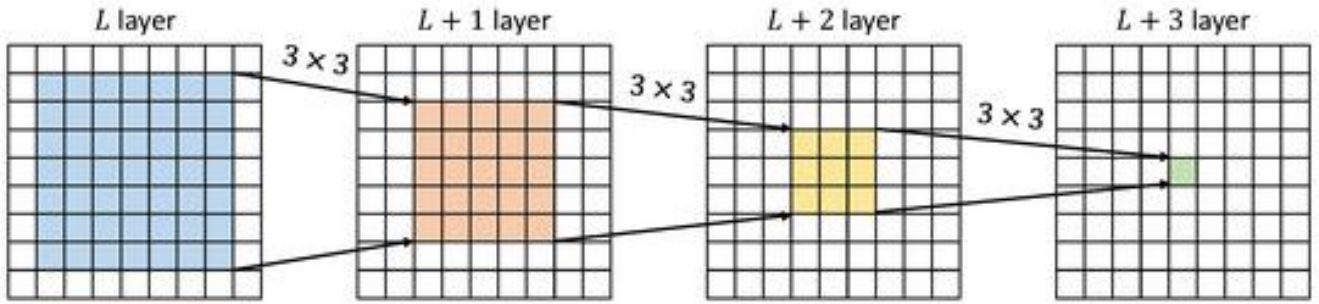
- Attached multiple classifier network during training that were discarded during inference
  - Multi-objective training
  - Minimize the loss function of all the classifier networks at the same time

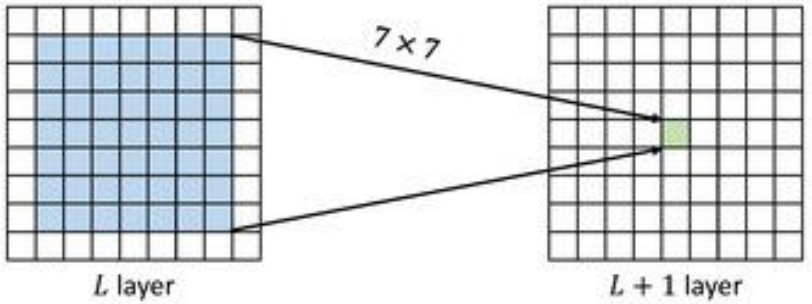


[5] Szegedy, C., et al. “Going Deeper with Convolutions”, arXiv e-prints, 2014. doi:10.48550/arXiv.1409.4842.

“VGG16” K. Simonyan and A. Zisserman (2014) [4]

- Increased the depth of the CNN by using smaller convolutions
- This enables the network the extract more complex hierarchical features
- Used multiple **3x3** convolutions instead of larger **(5x5 or 7x7)** convolutions:
  - More non-linearity
  - Same **receptive field**
  - Less parameters

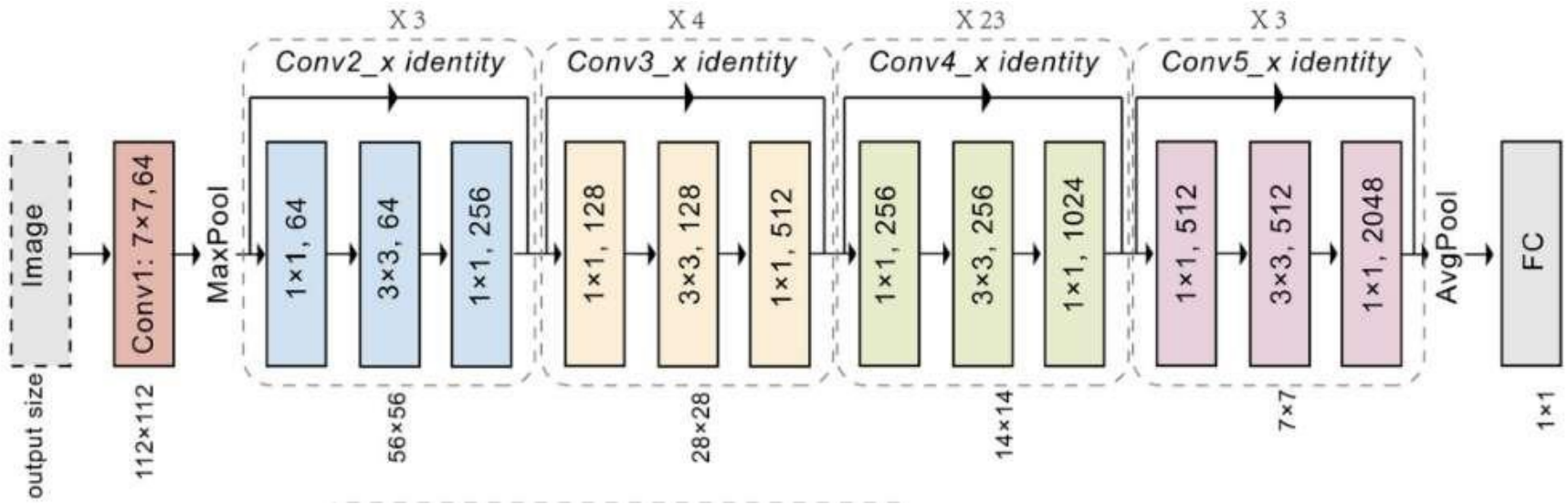




[4] Simonyan, K. and Zisserman, A., “Very Deep Convolutional Networks for Large-Scale Image Recognition”, arXiv e-prints, 2014. doi:10.48550/arXiv.1409.1556.

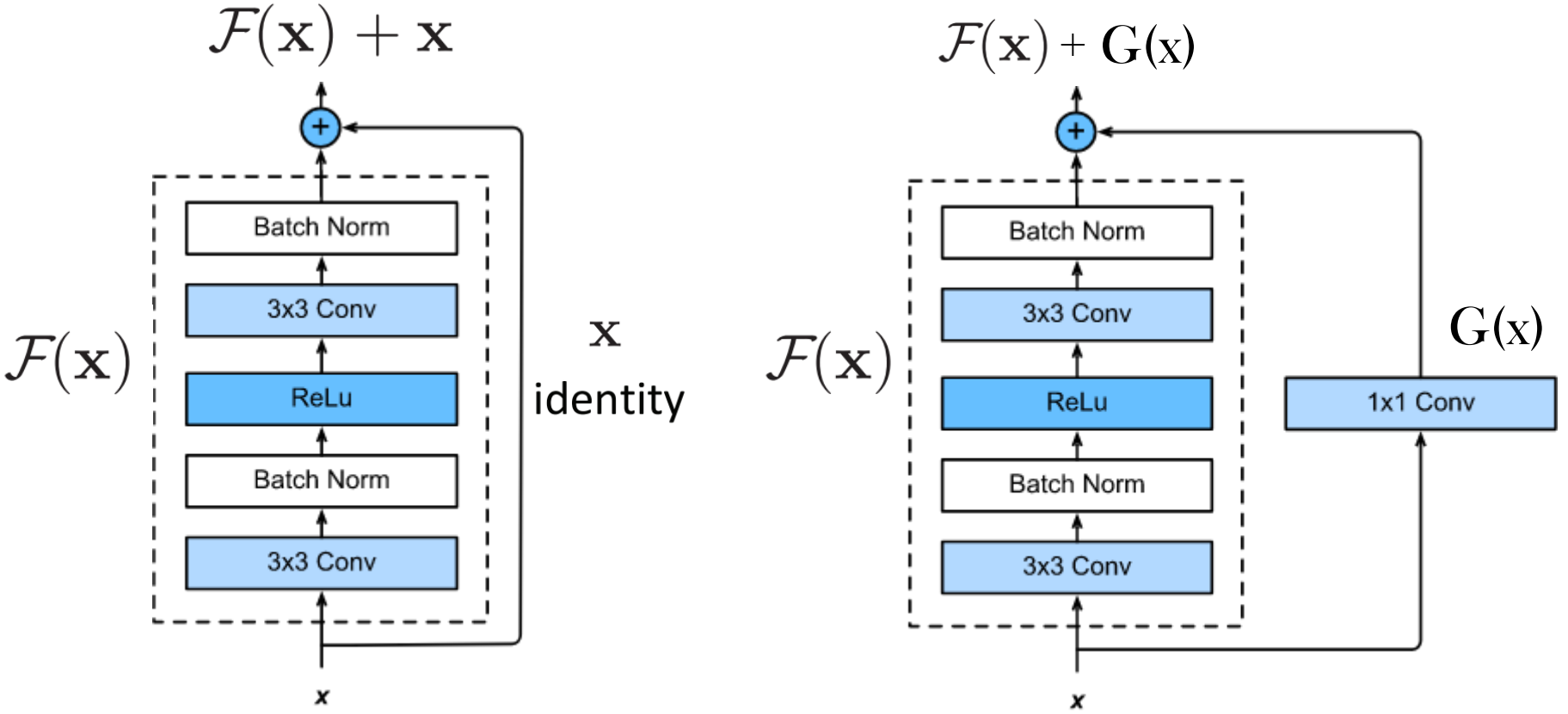
“Residual Network / ResNet” K. He et al. (2015) [6]

- Learn residual functions reference to layer inputs instead of unreferenced functions
- If we pretrain a smaller network and then we extend it with extra blocks it should have similar or better performance compared to the original network – **Identity mapping**



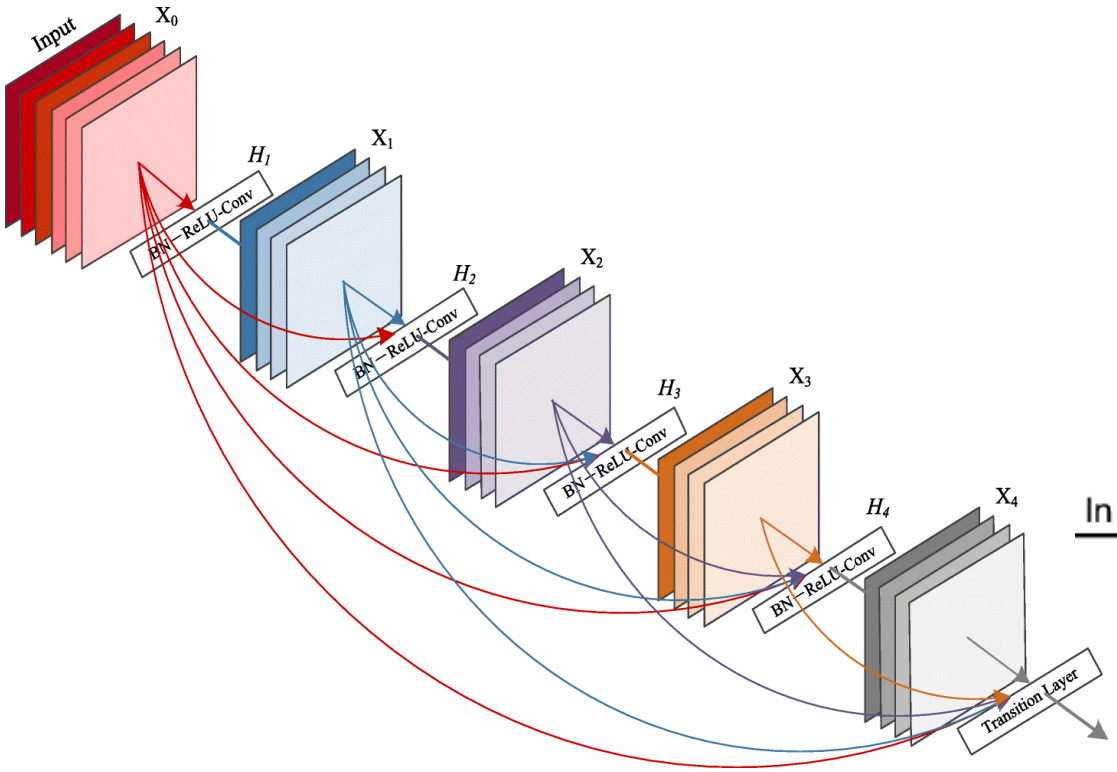
[6] He, K., Zhang, X., Ren, S., and Sun, J., “Deep Residual Learning for Image Recognition”, arXiv e-prints, 2015. doi:10.48550/arXiv.1512.03385.

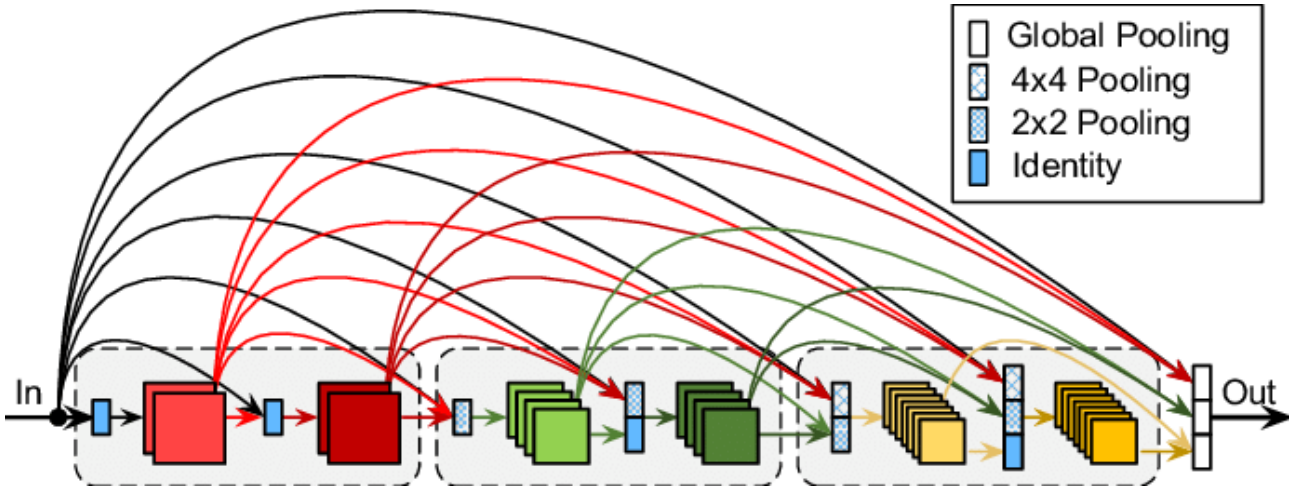
“Residual Network / ResNet” K. He et al. (2015) [6]



[6] He, K., Zhang, X., Ren, S., and Sun, J., “Deep Residual Learning for Image Recognition”, arXiv e-prints, 2015. doi:10.48550/arXiv.1512.03385.

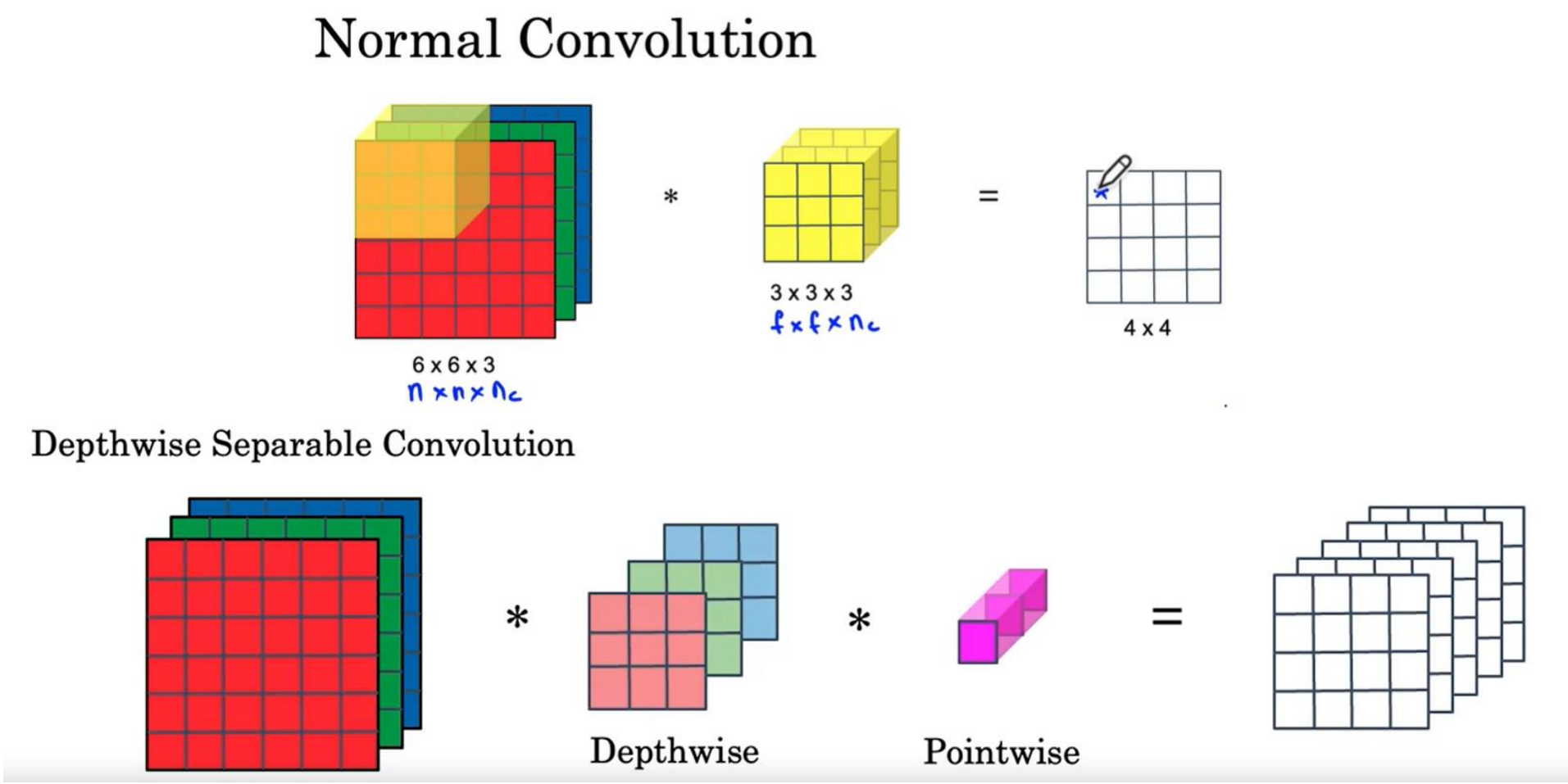
“DenseNet” G. Huang et al. (2016) [7]





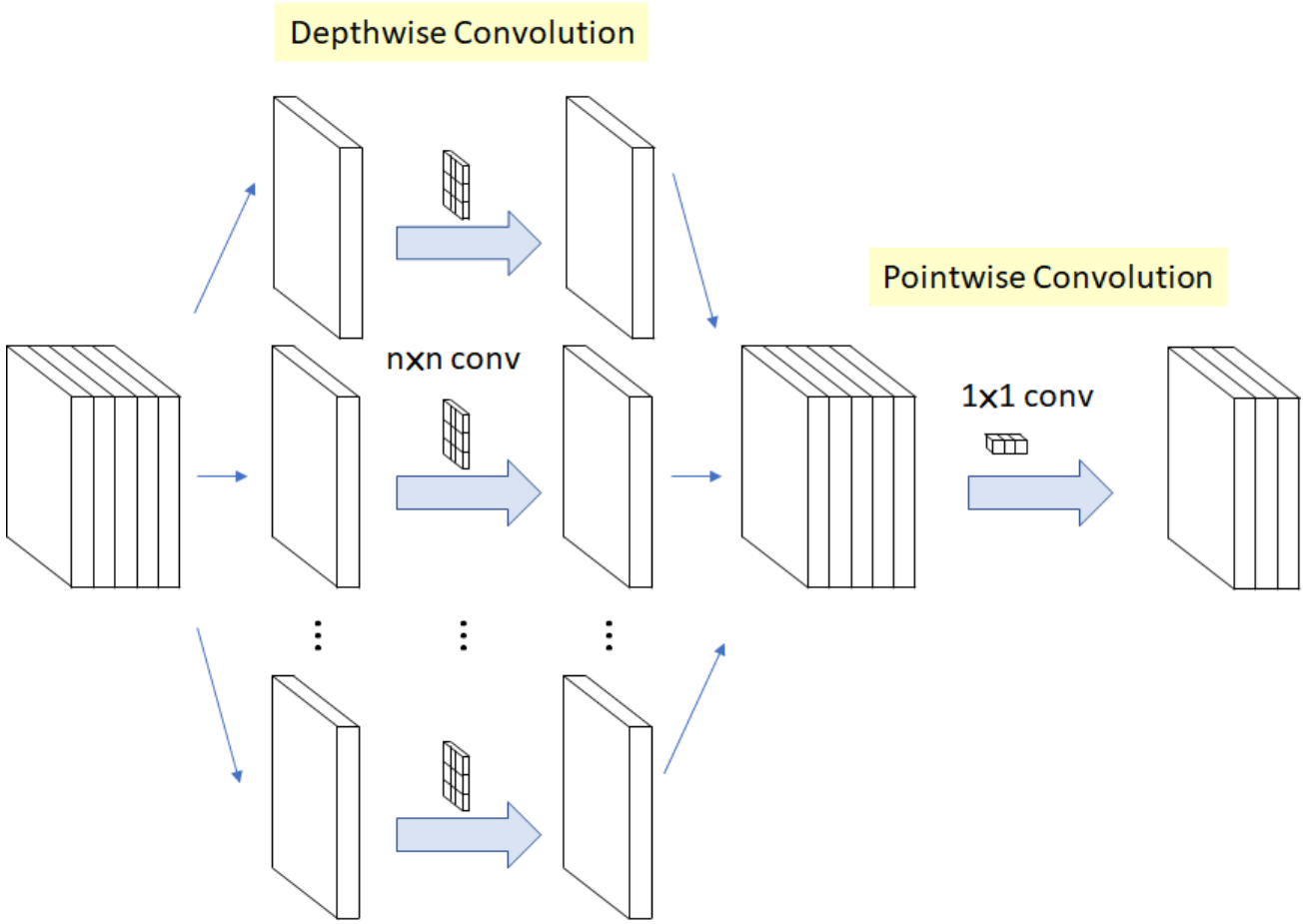
[7] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q., “Densely Connected Convolutional Networks”, arXiv e-prints, 2016. doi:10.48550/arXiv.1608.06993.

“MobileNet” A. G. Horward et al. (2017) [8]

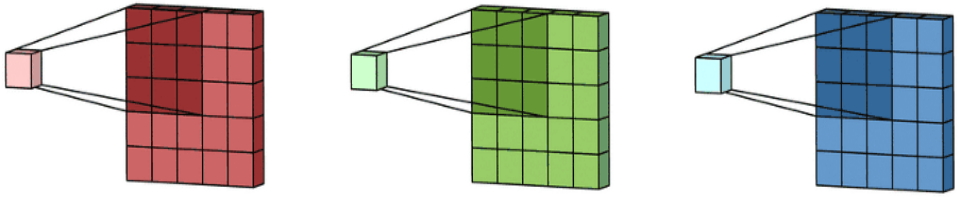


[8] Howard, A. G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv e-prints, 2017. doi:10.48550/arXiv.1704.04861.

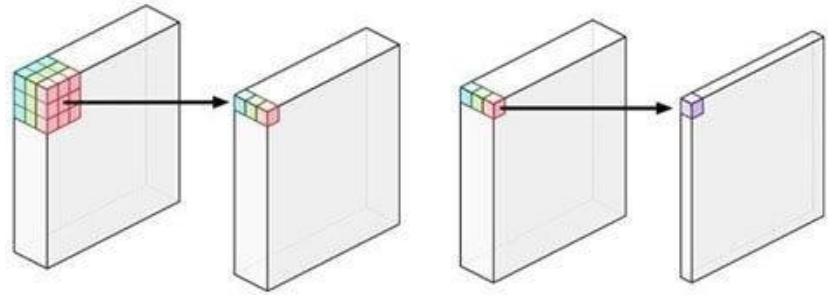
“MobileNet” A. G. Horward et al. (2017) [8]



Depthwise



Pointwise



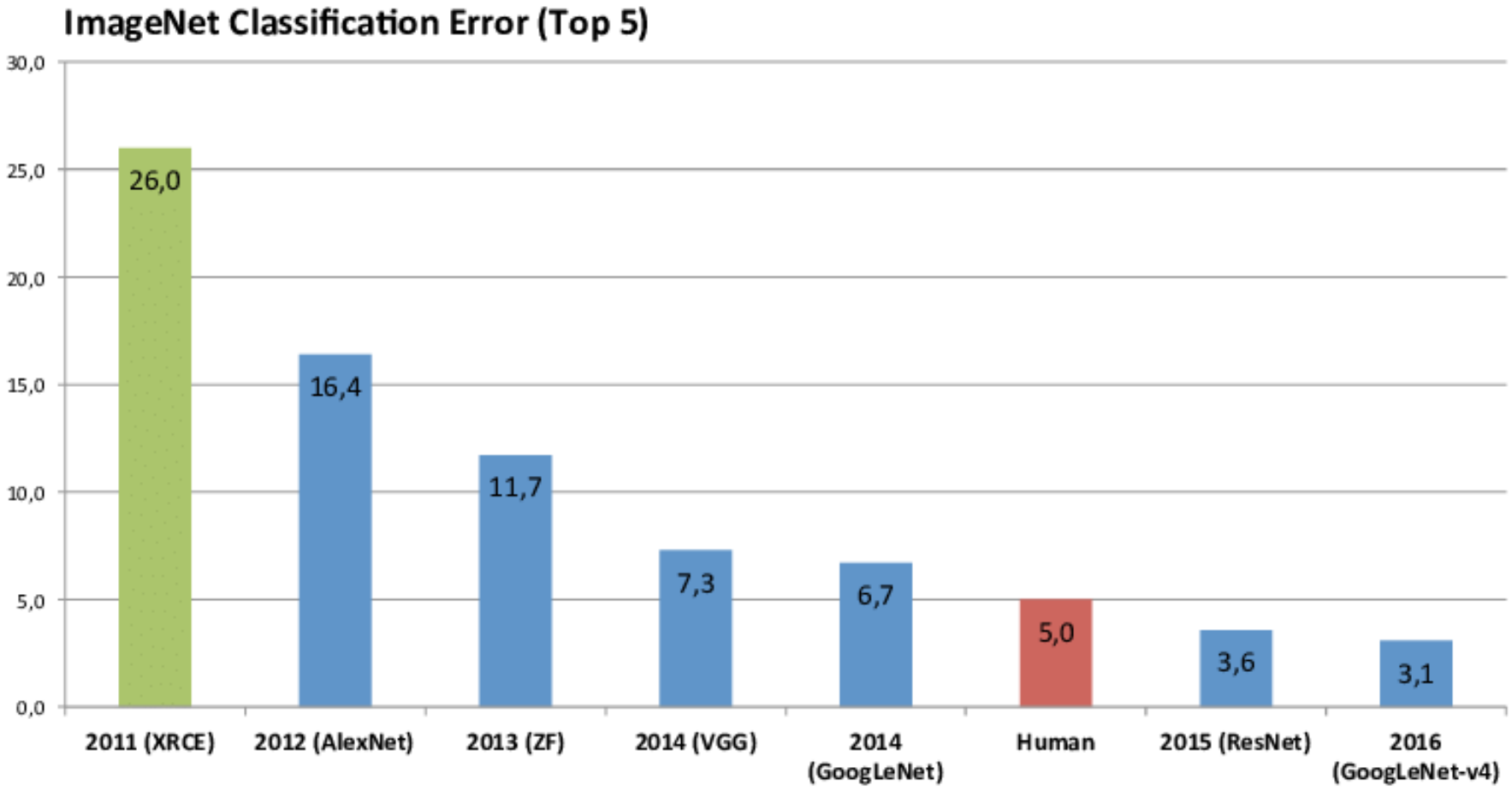
[8] Howard, A. G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv e-prints, 2017. doi:10.48550/arXiv.1704.04861.

More recent CNN architectures

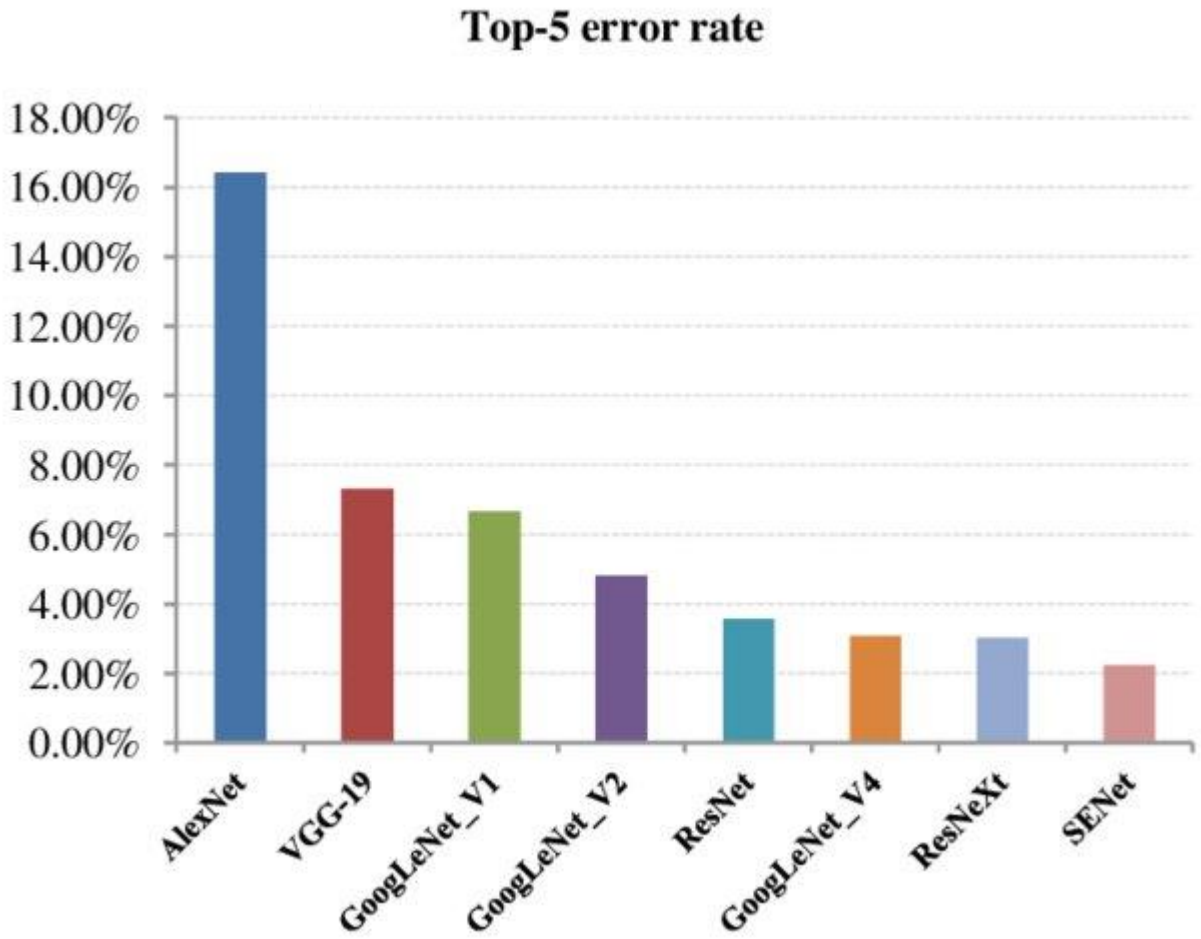
- **ZfNet**
- **Xception**
- **InceptionV2, V3, V4**
- **Inception-ResNet**
- **MobileNet V3**
- **FractalNet**
- **WideResNet**
- **PyramidalNet**
- **Residual Attention Net**
- **EfficientNet**
- **Etc**

- Currently, there are many models using the Transformer architecture (we will explore this later)

ImageNet – Architectures Top 5 error (2016)



ImageNet – Architectures Top 5 error (2019)



- **More recently:**
https://paperswithcode.com/sota/image-classification-on-imagenet

- **ImageNet website:**
https://www.image-net.org/

Deep Network Development

Lecture 7.

Image Classification Convolutional Neural Networks Transfer Learning

Budapest, 7th October 2025

1 Image Classification

2 Convolutional Neural Networks

3 CNN Architectures

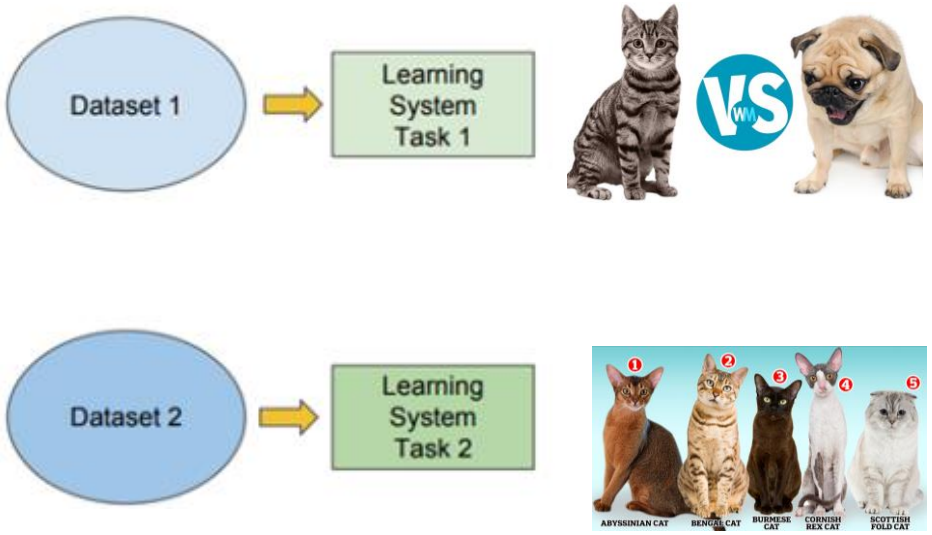
4 Transfer Learning

5 Autoencoders

Traditional ML vs Transfer Learning

Traditional ML

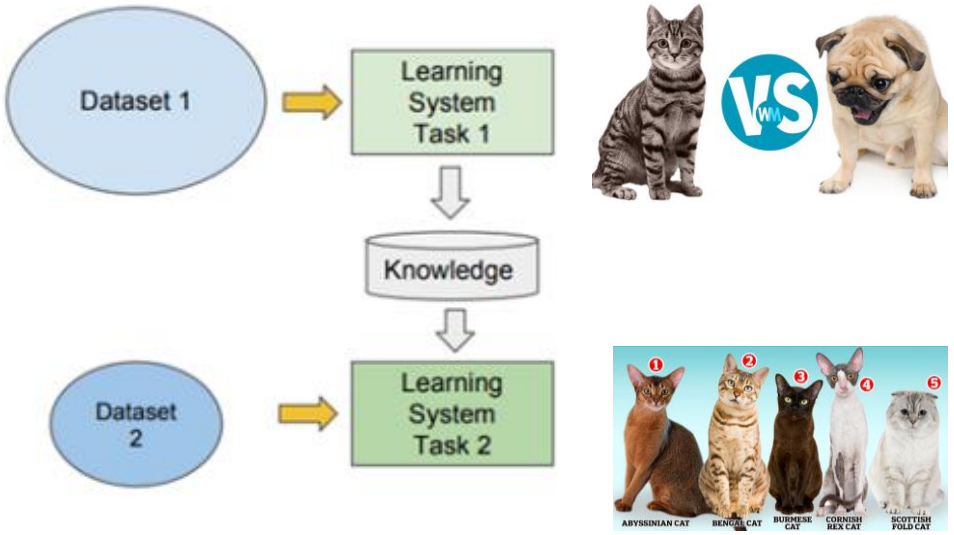
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

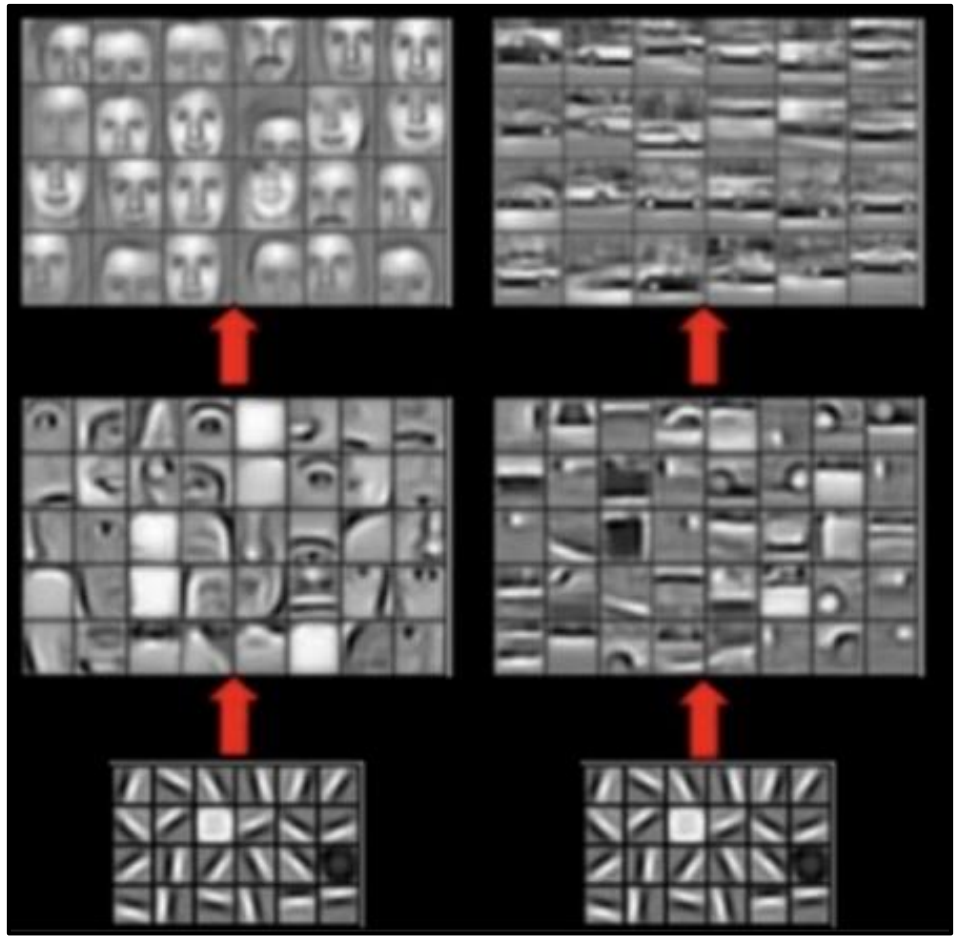
Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

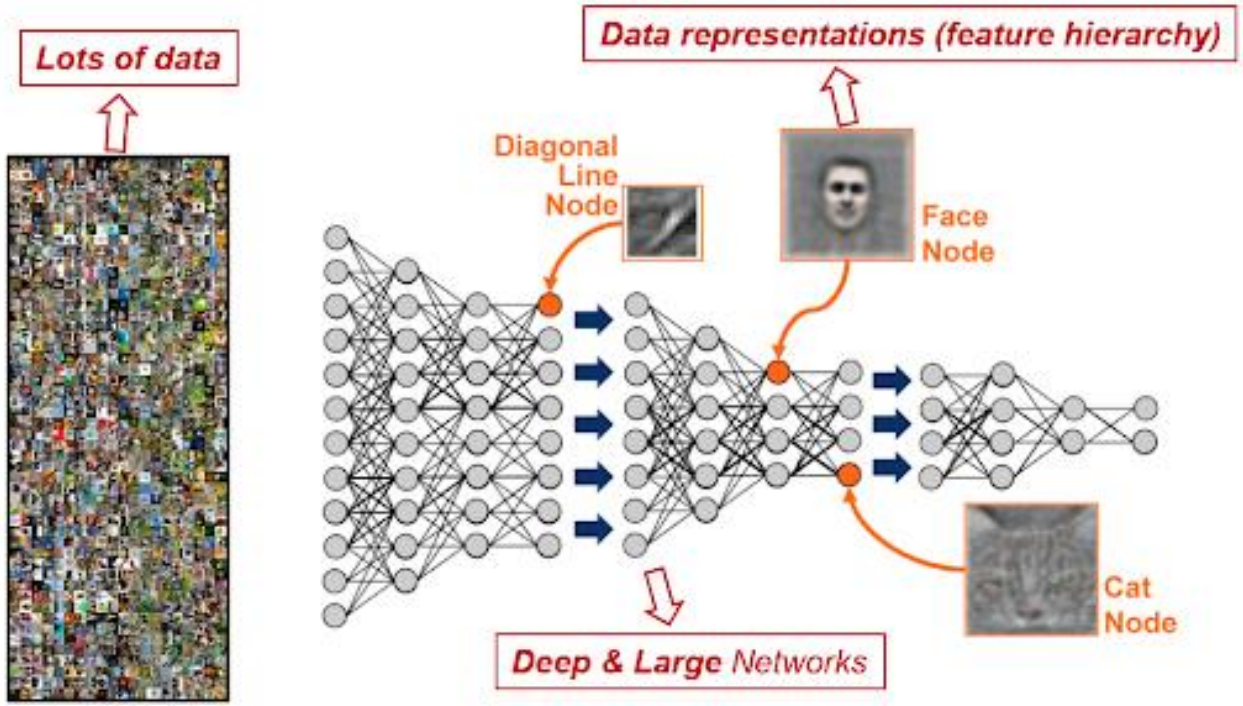


Learned / Transferrable features

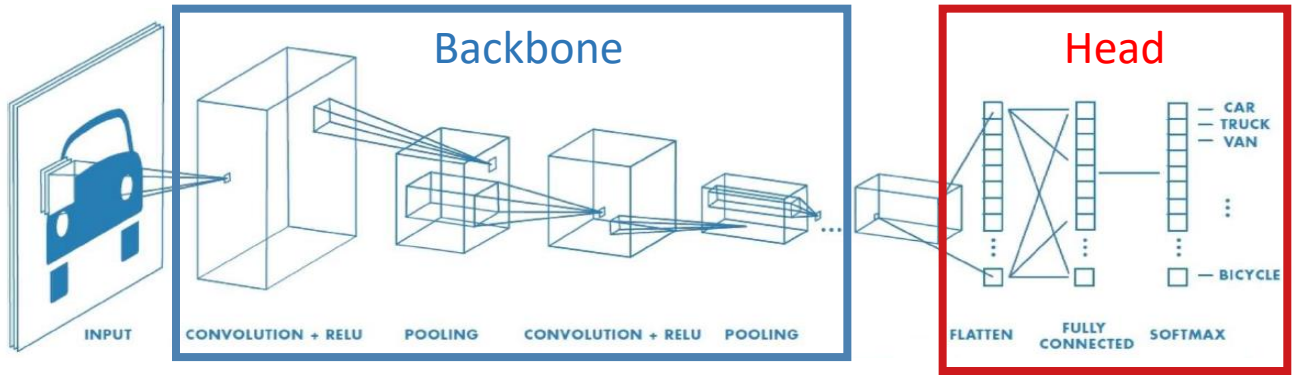
- The characteristics are built up in a pyramidal way
- Initial edge filtering with different orientation, "color" and cut-off values
- Simpler shapes, forms
- Part components (nose, eye, wheel,...)
- Faces, cars

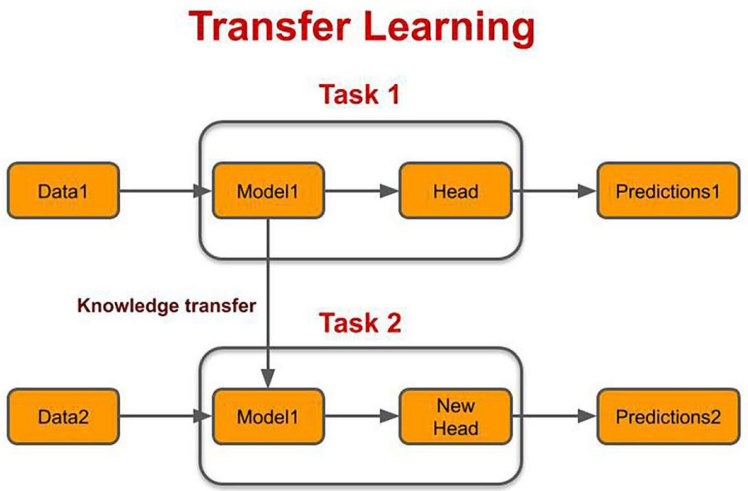


Learned / Transferrable features



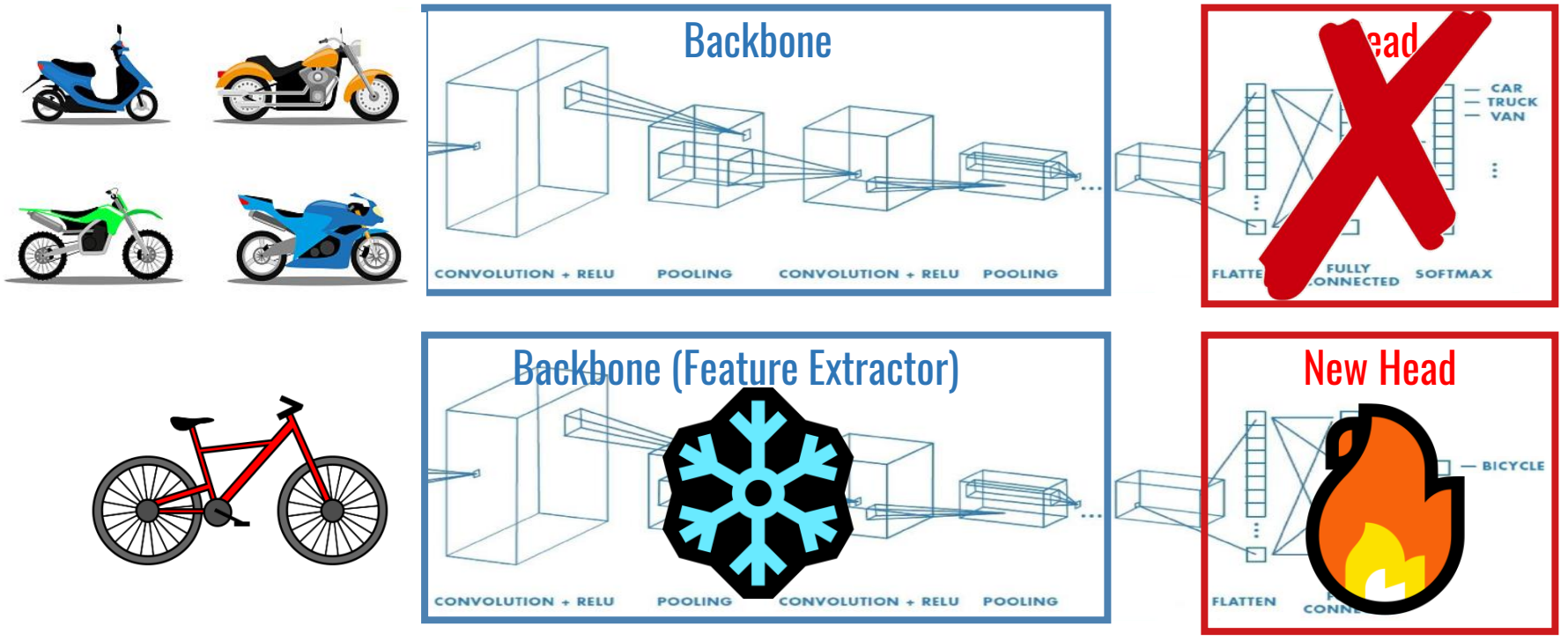
Backbone vs head





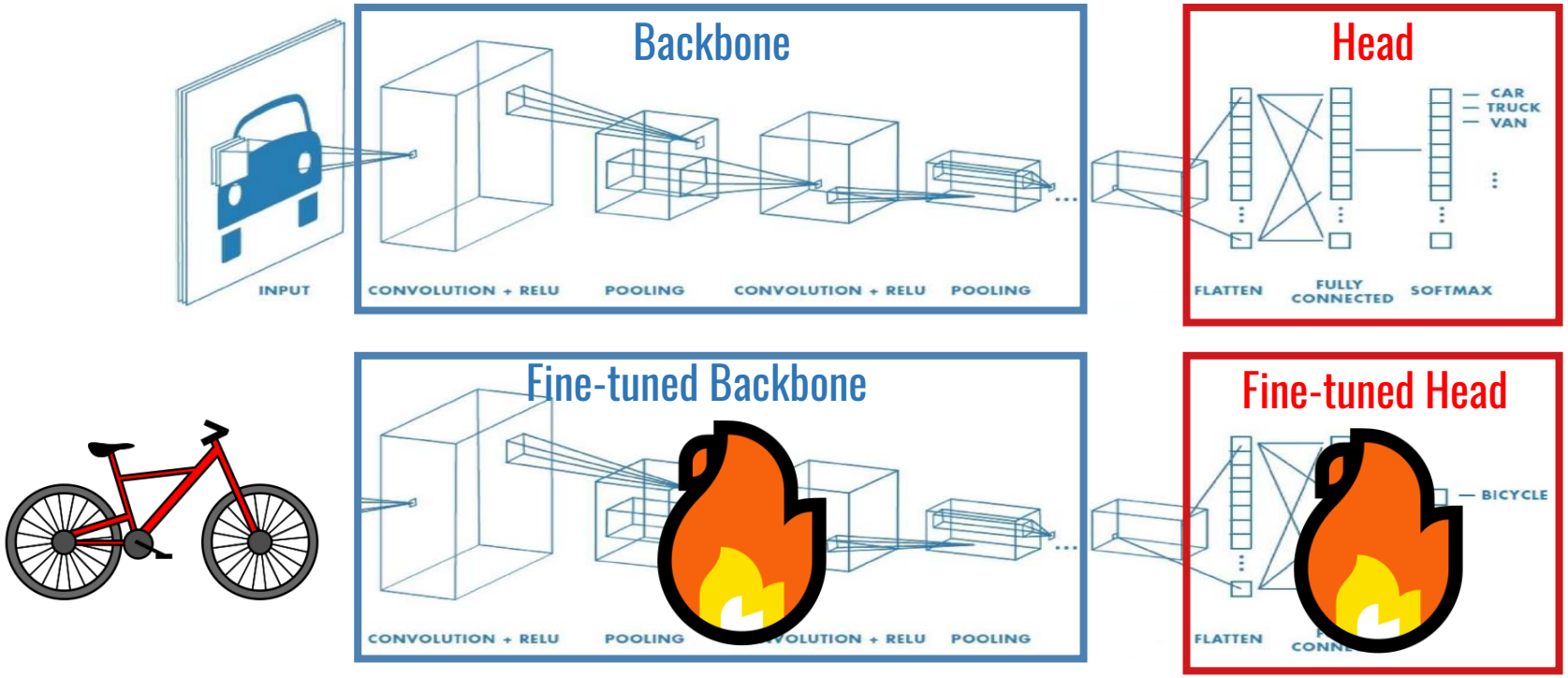
Feature extraction

- We initialize the network with pre-trained weights from a pre-trained model. We freeze all the weights from the backbone, and we replace the head with a new one, initialized with random weights. We only train the new head, and not the backbone.



Fine-tuning

- Instead of randomly initializing the weights for training , we initialize the network with pretrained weights from a pretrained model trained on a bigger dataset such as ImageNet (1M images and 1K class) and then we can fine-tune (train) the whole network.



Performance comparison

Dataset A
(Cats vs Dogs)

Dataset B
(Cat breeds)

Dataset C
(Kangaroos)

Model A
(trained on dataset A)

Model B
(Using Model A for Feature Extraction)

Good performance;
Fast Training;
Less data needed.

Model D
(Using Model A for Feature Extraction)

Worse performance

Model C
(Fine-Tuned from Model A)

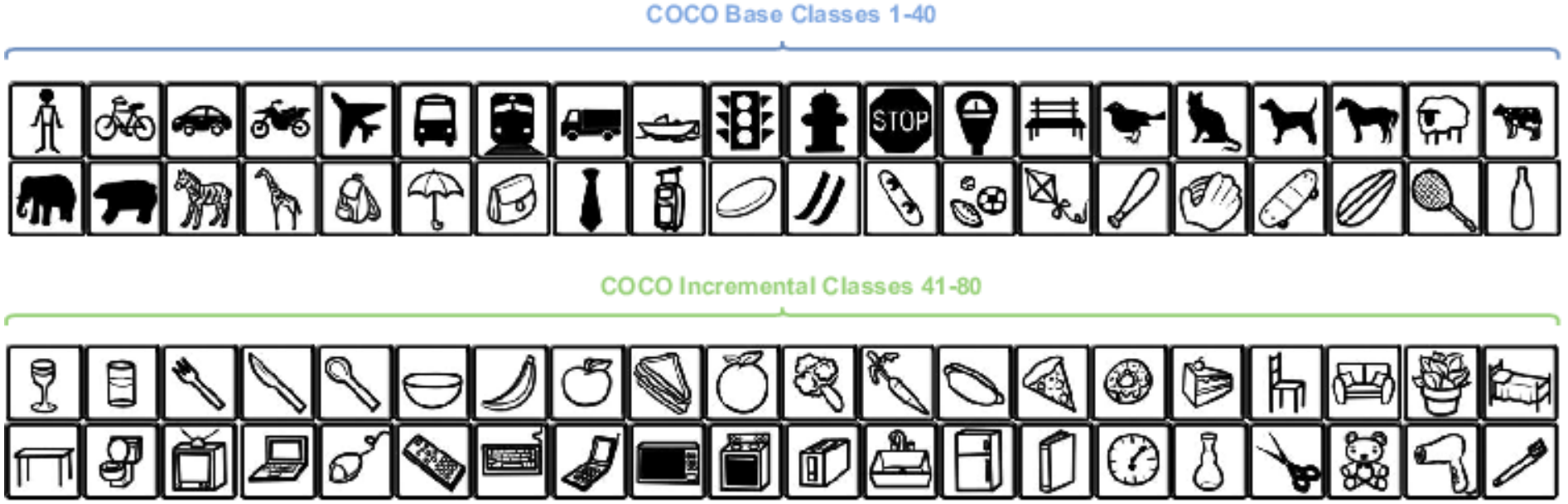
Good performance;
Slower Training;
More data needed.

Model E
(Fine-Tuned from Model A)

Good performance;
Slow Training;
More data needed.

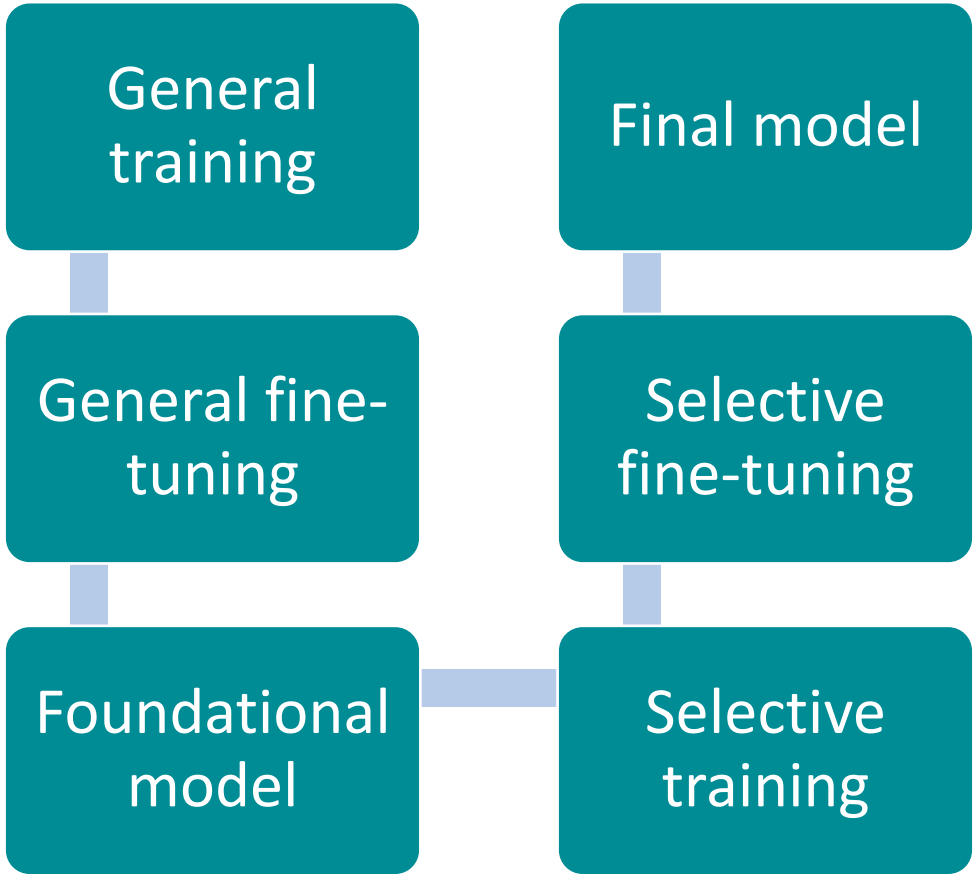
When to use Transfer Learning?

- Task A and B have the same input x
- When you have a lot of data for the problem you are transferring from (A) and few data for the problem you are transferring to (B)
- Low level features from A could be helpful for learning B
- Faster training. Use pre-trained weights as initialization point whether than randomly initializing weights



Deep Learning common steps - Parameter learning

- General
  - large dataset
  - Training
  - Fine-tuning
  - Foundational model



Deep Network Development

ELTE EÖTVÖS LORÁND UNIVERSITY

Lecture 7.

Image Classification Convolutional Neural Networks Transfer Learning

Budapest, 7th October 2025

1 Image Classification

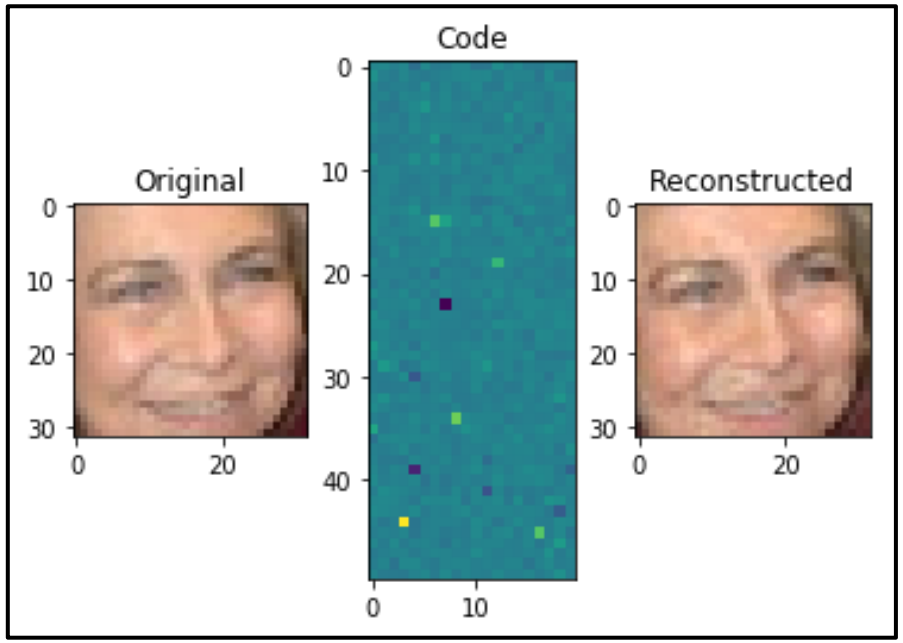
2 Convolutional Neural Networks

3 CNN Architectures

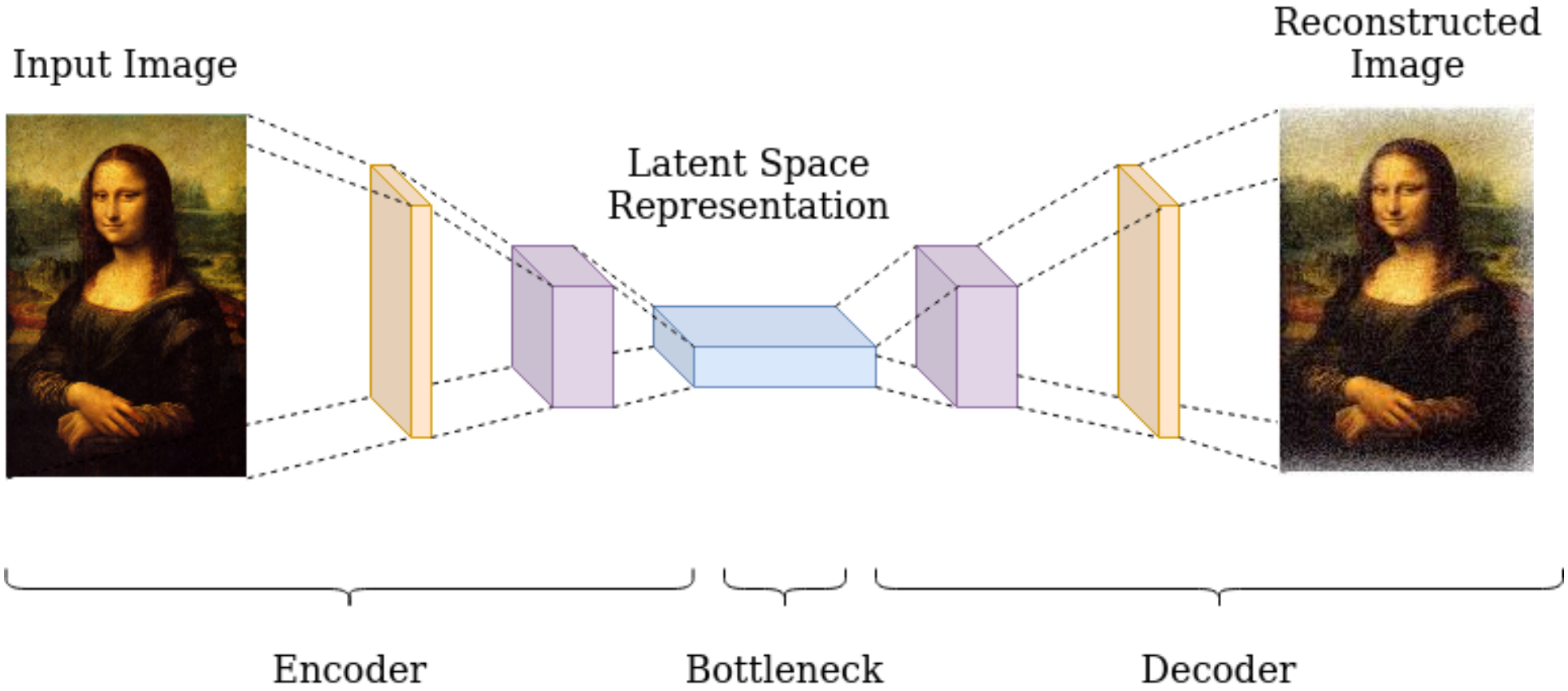
4 Transfer Learning

5 Autoencoders

Feature learning: Autoencoders



Reconstruction Task



Reconstruction Task

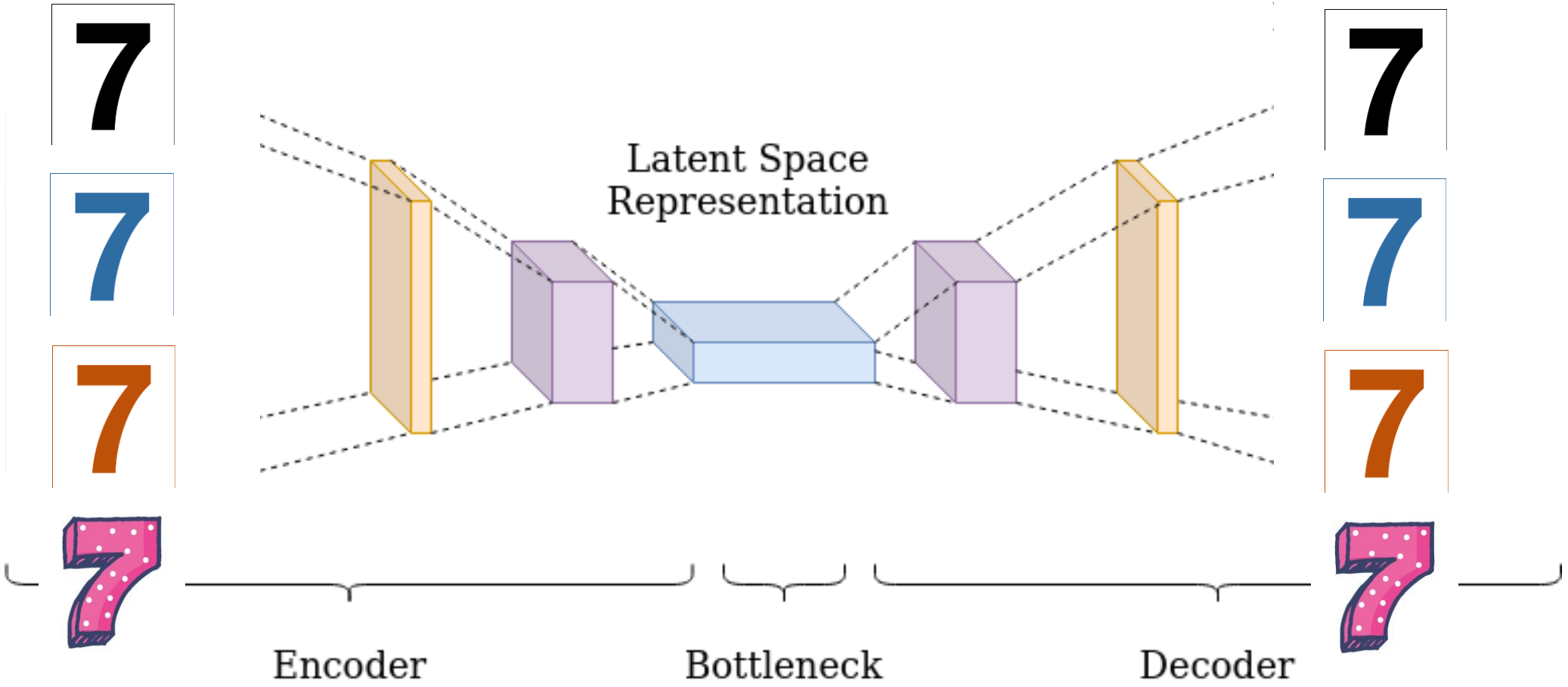
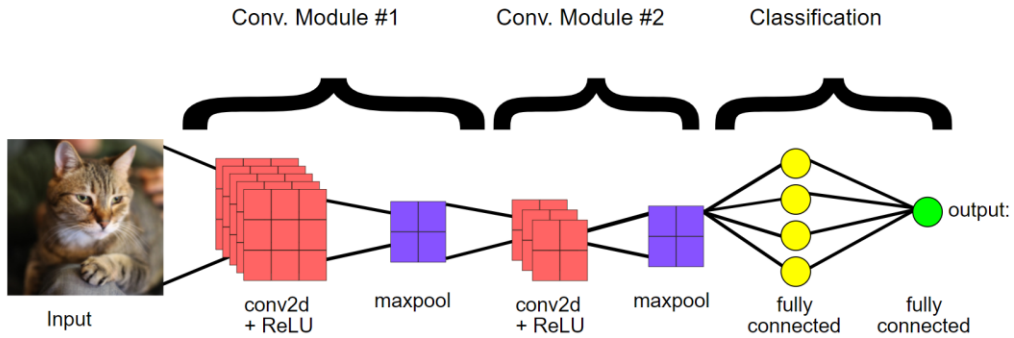


Image Classification: Supervised Learning (labelled data is needed)

Data:
X = image
Y = label / target (1)



0.73

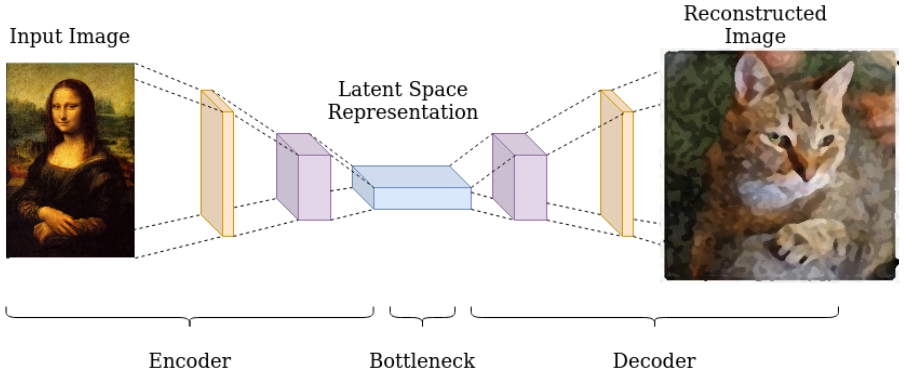
Loss between prediction and target

$$L_{BC}(h(x), y) = -y \log h(x) - (1-y)\log(1-h(x))$$
$$L_{BC}(0.73,1) = -1 \log 0.73 - (1-1)\log(1-0.73)$$
$$L_{BC}(0.73,1) = -1 * -0.13667714 - (0)\log(0.23)$$
$$L_{BC}(0.73,1) = 0.13667714$$

$$\boldsymbol{L_{BC}(0.73, 1) = 0.14}$$

Reconstruction: Self-Supervised Learning (no need for labelled data)

Data:
X = image
Y = X



Loss between prediction and input

( - $L_{BC}($)**2

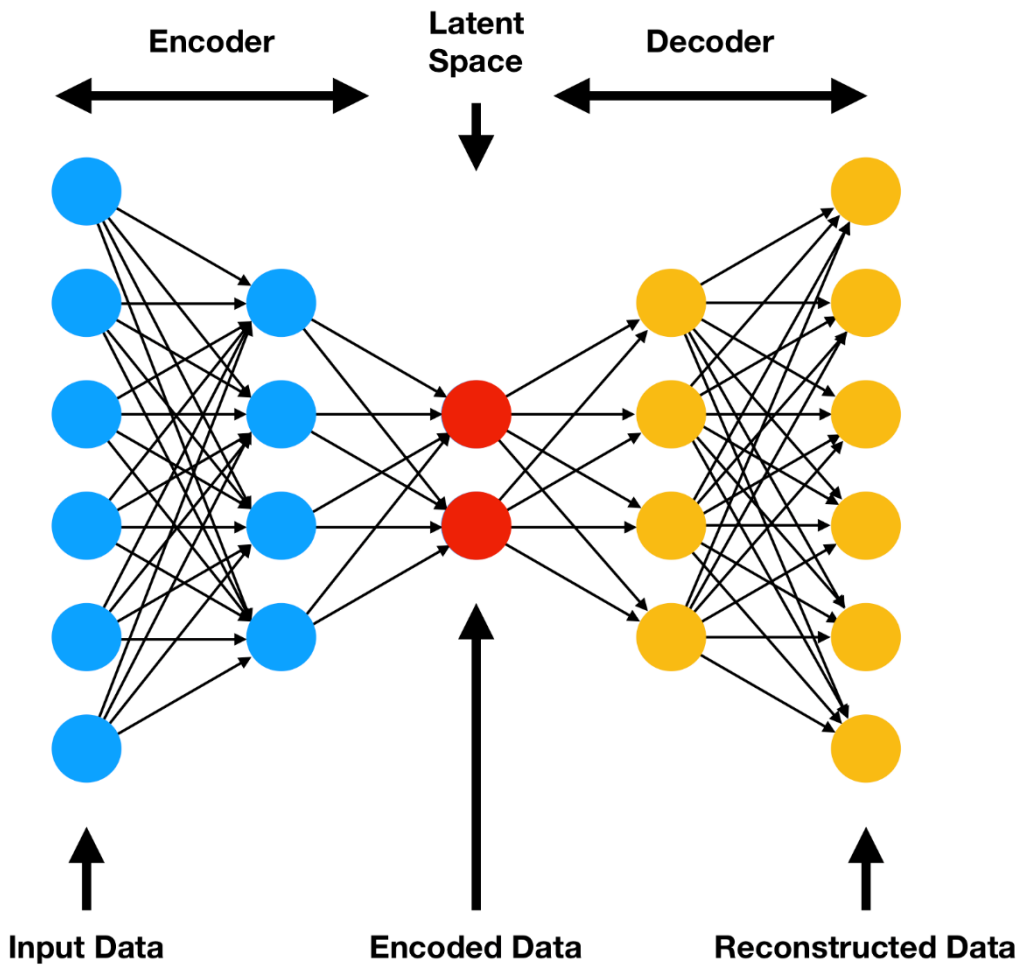
= 0.2

Autoencoder structure

Autoencoder is a machine learning algorithm that **learns a compressed representation of an input**. Because it requires no label it is an unsupervised method[1]. However, the **output is a reconstruction of the input**, thus it is also considered as a self-supervised method.

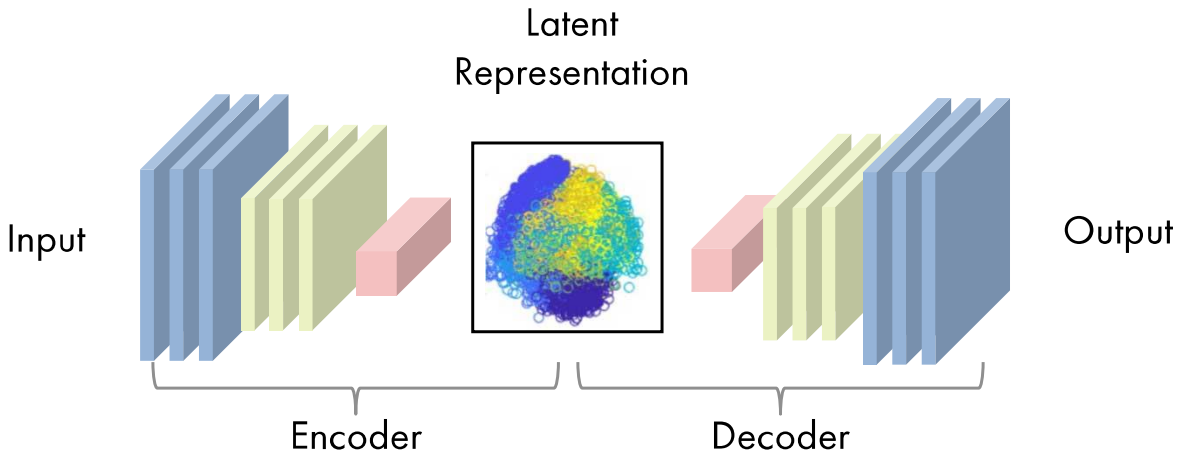
An autoencoder consists of three parts:

- **Encoder**: the part of the network that compresses the input into a latent-space representation of reduced dimension. It can be represented by an encoding function h=f(x).
- **Latent space (bottleneck/code)**: the part of the network which contains the reduced representation of the input.
- **Decoder**: the part that aims to reconstruct the input from the latent space representation. It has a similar structure to the encoder and can be represented by a decoding function r=g(h).



Autoencoder structure

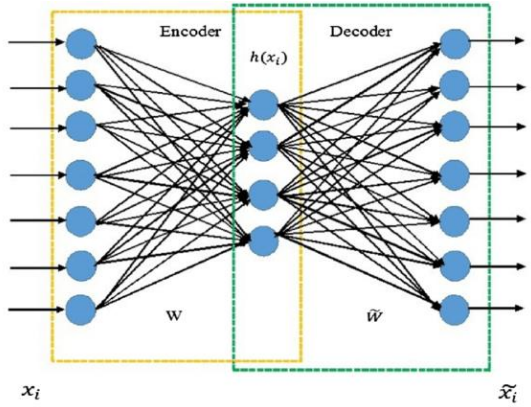
- Autoencoders are learned automatically from data examples, **using the input as output target**, making them easy to train. However, they do not generalize to new data.
- If the only purpose of autoencoders was to copy the input to the output, they would be useless. The main idea is that by training the autoencoder to copy the input to the output, **the latent representation will learn the most important features of the input**.
- The bottleneck is designed in such a way that the maximum information possessed by an input is captured in it, therefore, the bottleneck **forms a knowledge-representation of the input**.



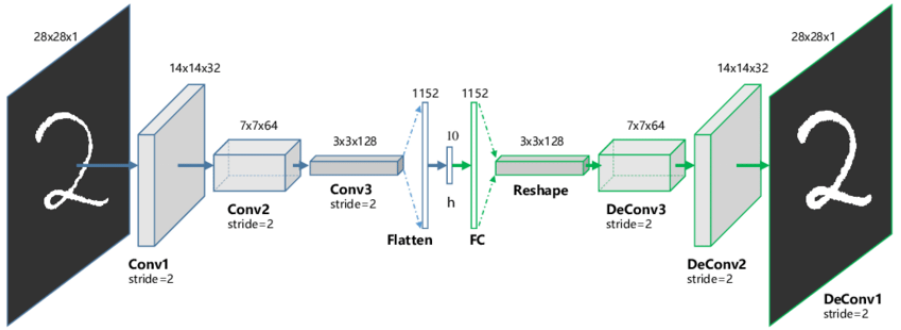
Autoencoder structure

The input and output layers of the autoencoder can be formed with:

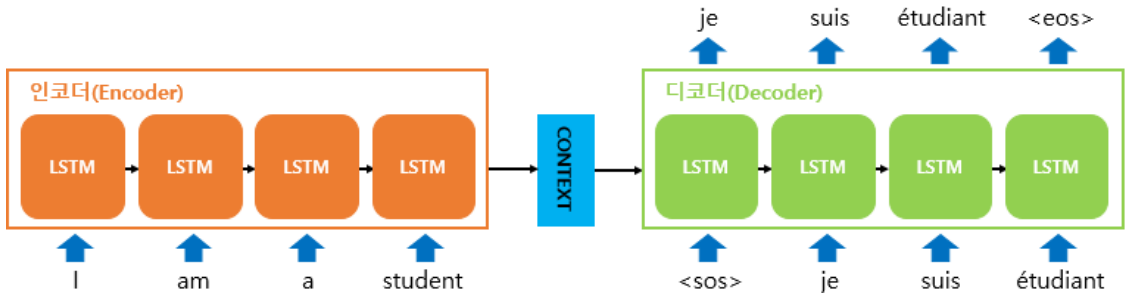
- Feed Forward neural networks (Dense / Fully connected layers)



- Convolutional neural networks



- Recurrent neural networks



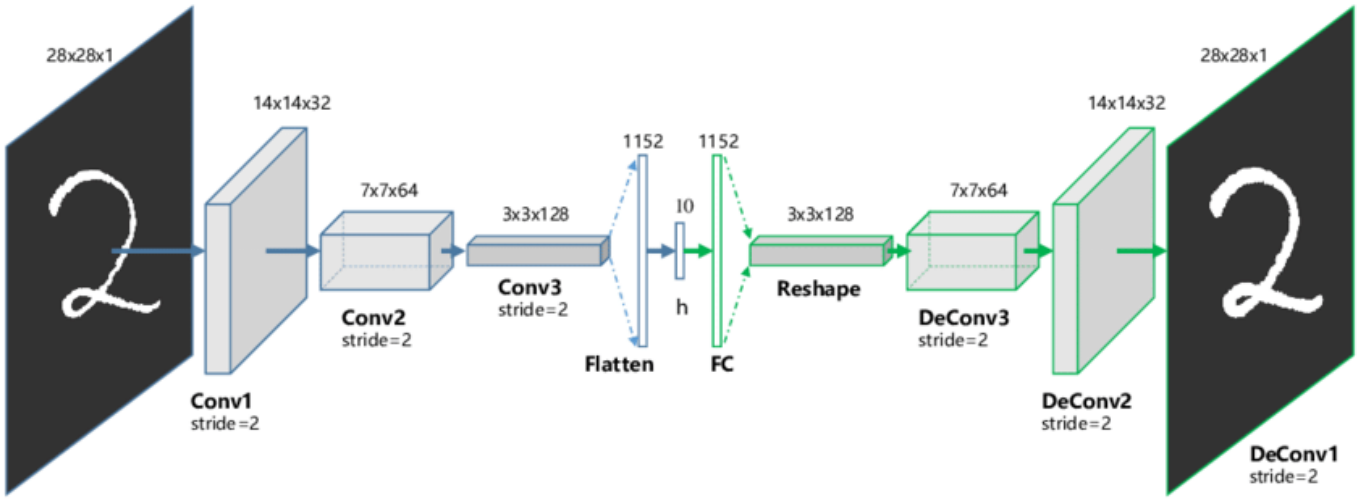
- Others (Restricted Boltzmann Machine, ...)

Autoencoder Types

- Convolutional Autoencoder
- Denoising Autoencoder
- Sparse Autoencoder
- Variational Autoencoder
- and more

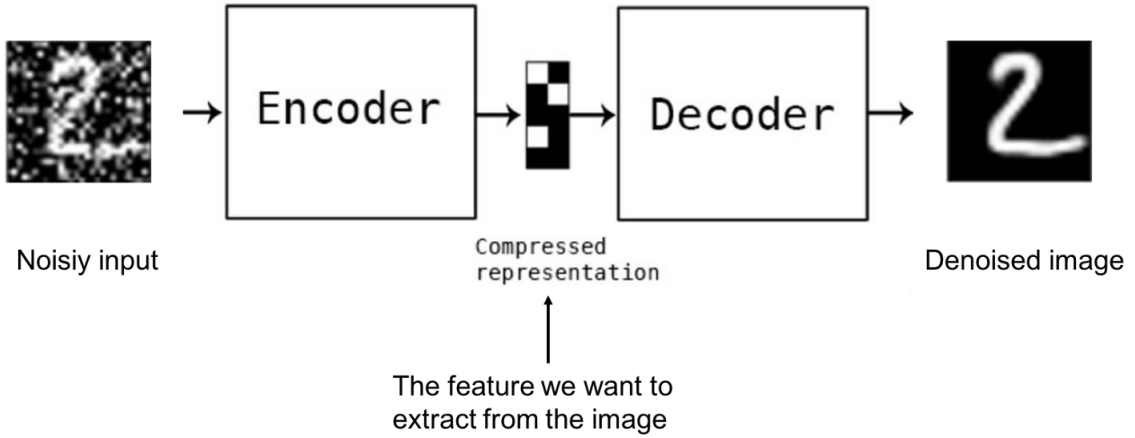
Convolutional Autoencoders

- Specifically designed for image data.
- They employ convolutional layers in both the **encoder** and **decoder** parts of the network.
- This architecture allows them to capture spatial dependencies and hierarchical features effectively.
- The reconstruction of the input image is often blurry and of lower quality due to compression during which information is lost.



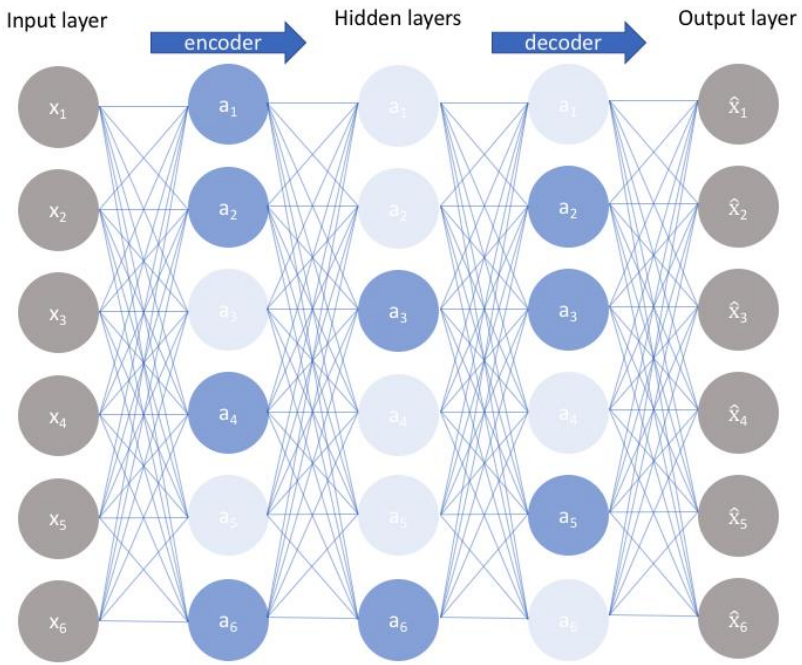
Denoising Autoencoders

- Designed to remove noise from input data.
- Noise can be added to the input, by modifying different parts of the input:
  - Gaussian noise; Salt-and-Pepper noise; Random Masking; Perturbation noise; etc
- During training, a noisy version of the input is provided, and the network learns to reconstruct the clean, noise-free data.
- This encourages the model to capture robust and meaningful features while filtering out irrelevant or noisy information.



Sparse Autoencoders

- Designed to impose sparsity constraints on the representations learned by the autoencoder.
- Sparsity means that only a small subset of neurons in the hidden layer is active at a time
- They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.
- Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer and to the reconstruction error. This prevents overfitting.



Applications

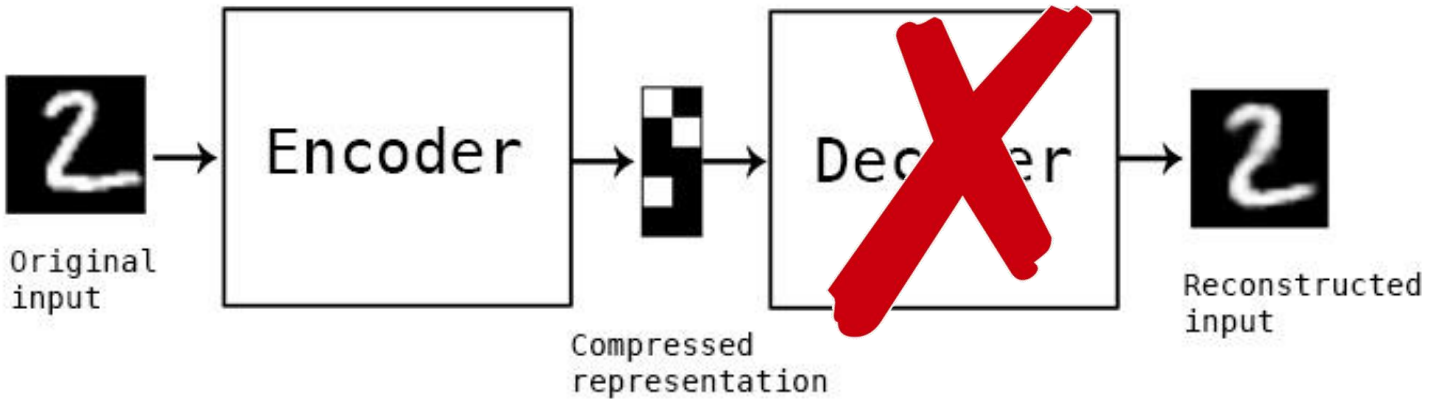
(Not for) Data Compression

- Although autoencoders can compress the input into a latent representation, it is **usually not used for data compression**. The reasons are:
  - **Lossy compression**: The output of the autoencoder is not the same as the input, it is a close but degraded representation because information is lost in the latent representation.
  - **Data-specific**: Autoencoders are only able to meaningfully compress data like what they have been trained on. Since they learn features specific for the given training data. Hence, we can't expect an autoencoder trained on handwritten digits to compress landscape photos.

Applications

Dimensionality Reduction

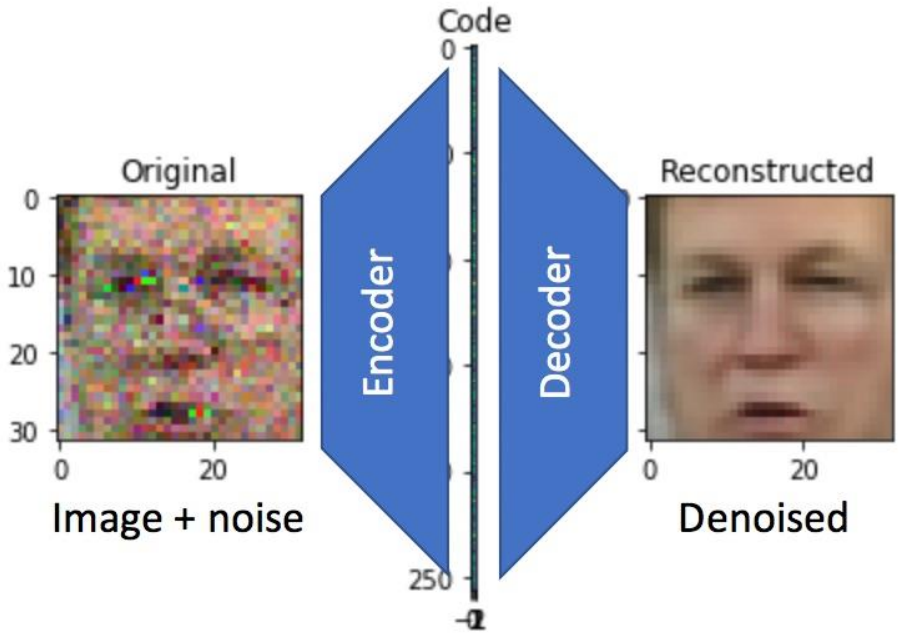
- The autoencoders convert the input into a **reduced representation which is stored in the latent representation space**. This is where the information from the input has been compressed. Thus by **removing the decoder part**, an autoencoder can be used for dimensionality reduction with **the output being the latent representation vector**.

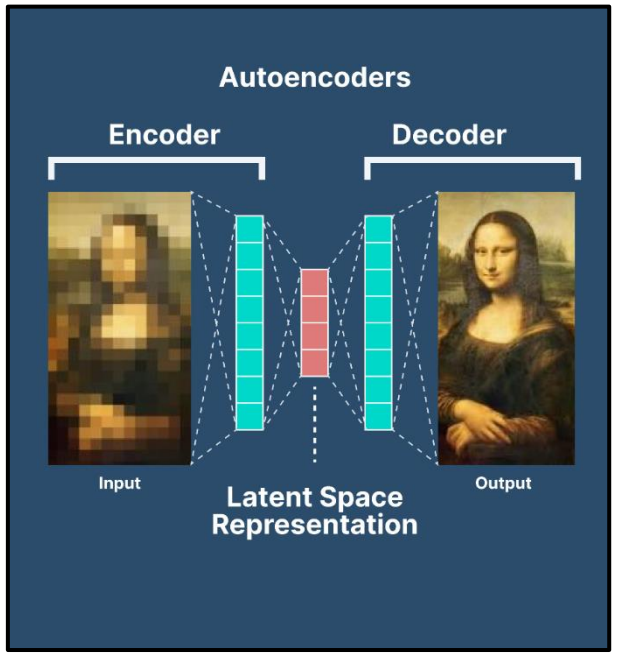


Applications

Image Denoising

- Example:
  - https://huggingface.co/spaces/Xintao/GFPGAN
  - https://huggingface.co/spaces/aryadytm/photo-colorization

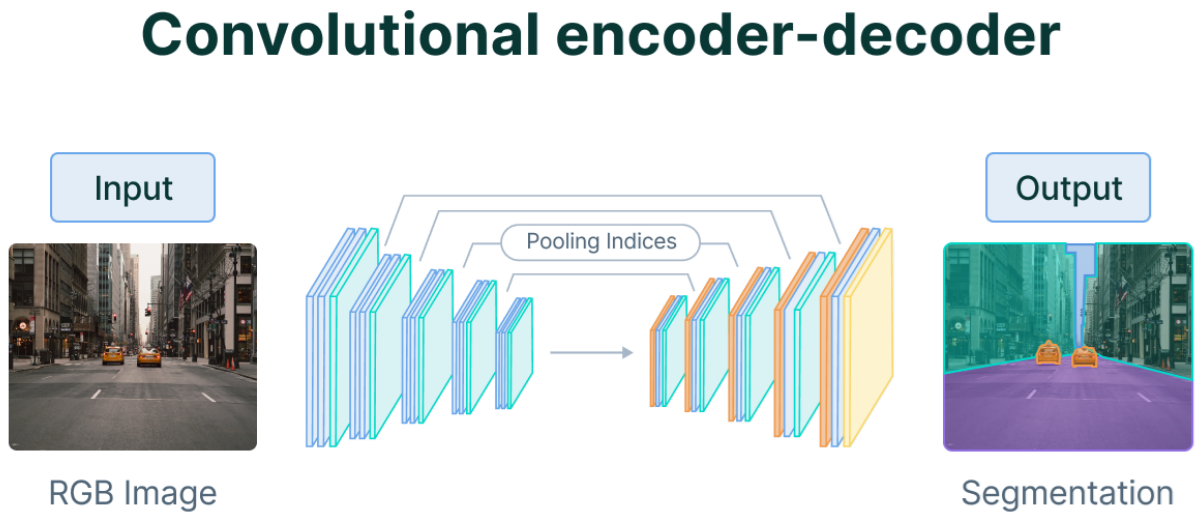




Applications

Image Segmentation

- Image segmentation is the process of partitioning an image into multiple segments each belonging to a class. The goal is to simplify and/or change the representation of an image by **grouping pixel values according to the class they belong**.

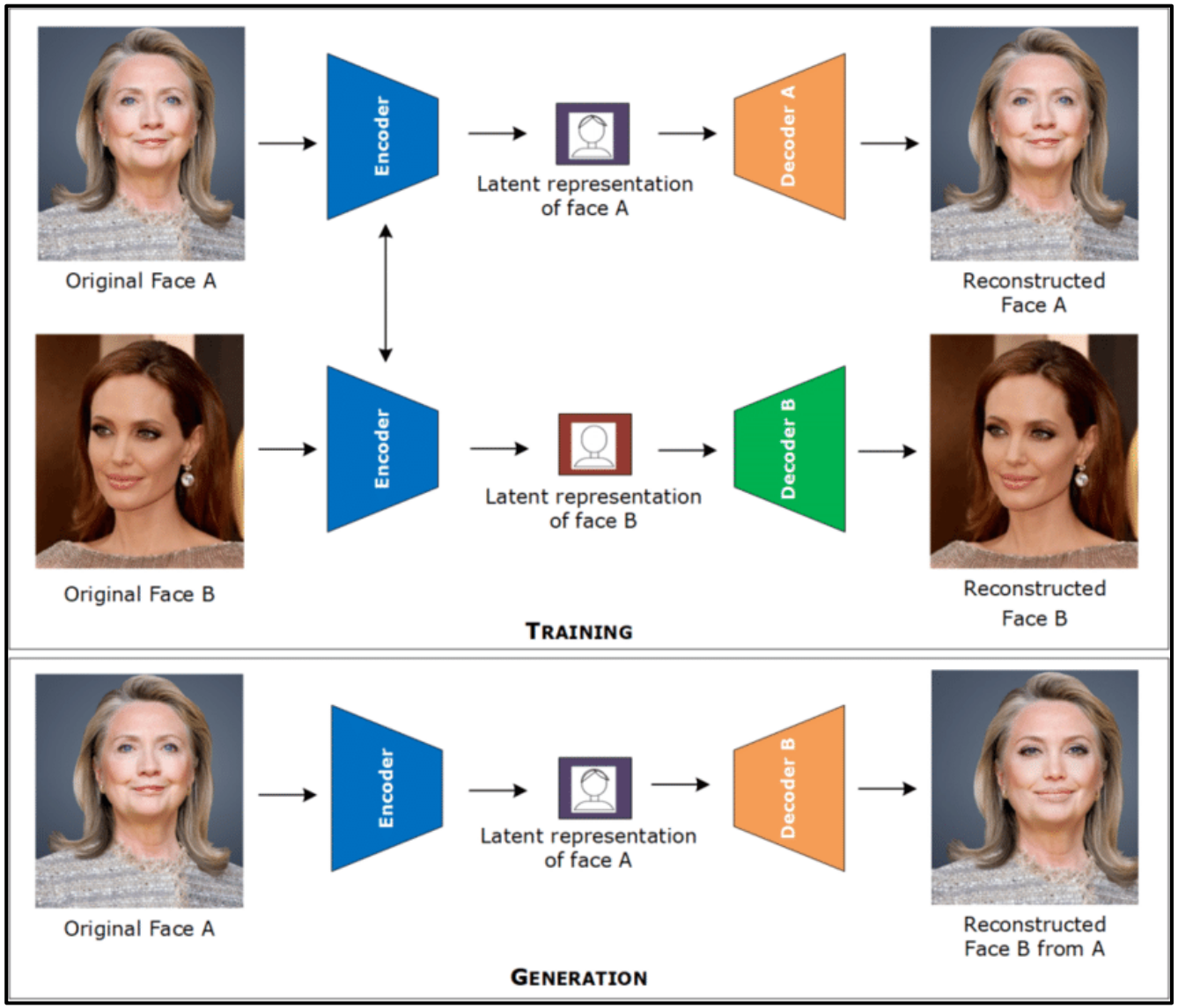


Applications

Image Generation

Encoder-Decoder swap

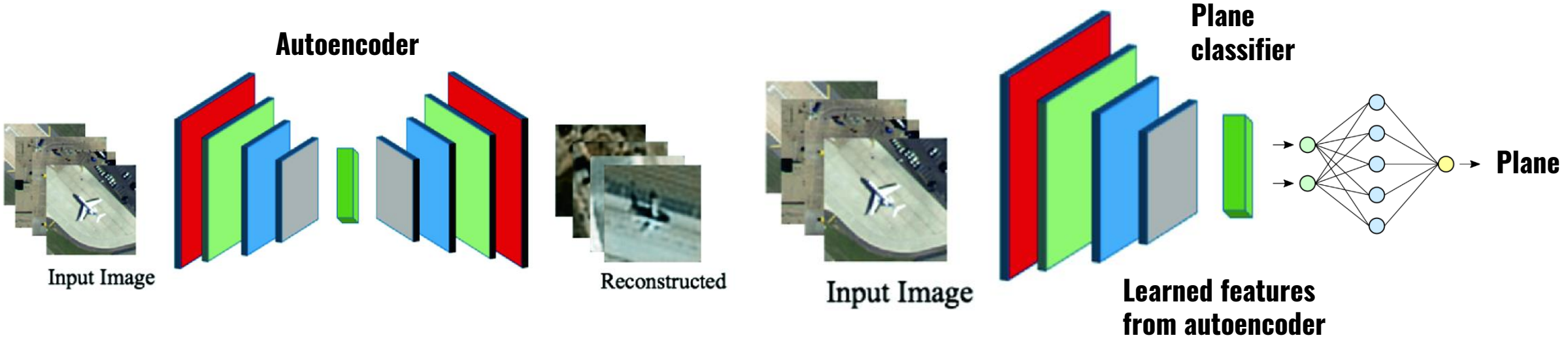
Latent Space Manipulation



Applications

Feature Extraction

- The Encoding part of Autoencoders helps to **learn important hidden features** present in the input data, in the process to reduce the reconstruction error. During encoding, a new set of combinations of original features is generated. **By removing the decoder, we can use the encoded features as input features for a network to use**, for example, in the classification task.



Summary

- **Fully Connected Networks** (Feed Forward Networks) calculate a weighted sum of the inputs
- We can add non-linear activation functions like **Sigmoid** to transform linear regression to logistic regression (classification)

- **Classification** is a supervised learning task
  - Binary: the target is 0 or 1
  - Multiclass: the target is one element out of a discrete set of elements (usually use one-hot encoding)

- **Convolutional Neural Networks** (CNNs) are efficient algorithms for images
  - Weight sharing
  - Capture spatial patterns

- By changing the CNN configurations, we create **different architectures**

Summary

- Different CNN hyperparameter choices result in different CNN architectures

- Transfer Learning is used when we want to utilize the learned features from a model A, that can be useful for a task that model B is trying to solve.
- We can retrain the whole model or freeze part of the model and only retrain a small part

- Autoencoders are unsupervised / self-supervised methods
- By reconstructing the input, they learn to encode the input into a lower dimensional latent space
- By removing the decoder from a trained autoencoder, the encoder can be used for feature learning

Resources

Books:

- Courville, Goodfellow, Bengio: Deep Learning
  Freely available: https://www.deeplearningbook.org/
- Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J.: Dive into Deep Learning
  Freely available: https://d2l.ai/

Courses:

- Deep Learning specialization by Andrew NG
- https://www.coursera.org/specializations/deep-learning

That’s all for today!