Universidade do Minho
Escola de Engenharia
Departamento de Informática

Mestrado Integrado em Engenharia Informática
Mestrado em Engenharia Informática
Computação Natural
2020/2021

Filipe Gonçalves, Paulo Novais

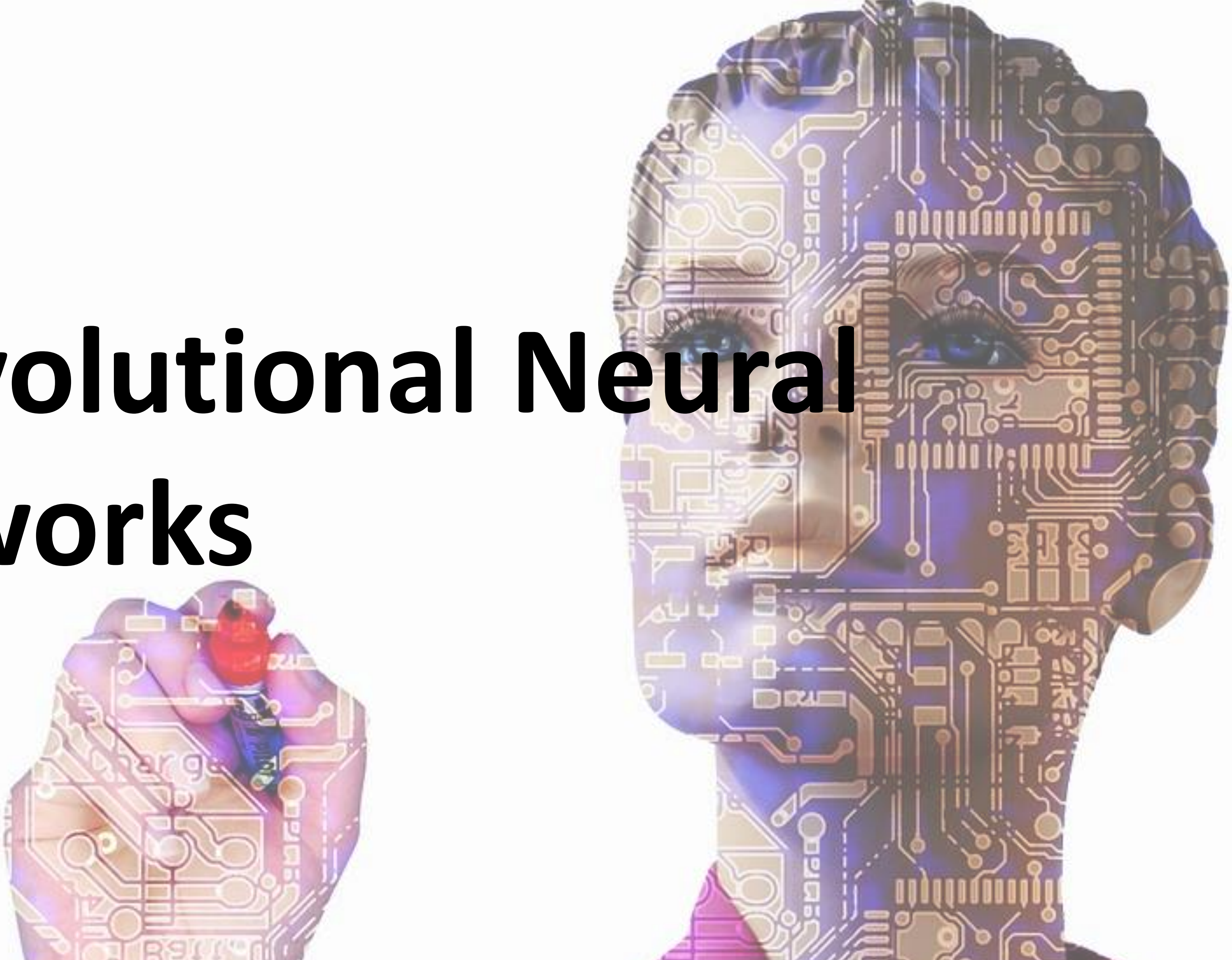
Synthetic Intelligence Lab

Computação Natural@ 2020/2021

- Paulo Novais – pjon@di.uminho.pt
- Filipe Gonçalves – fgoncalves@algoritmi.uminho.pt

- Departamento de Informática
  Escola de Engenharia
  Universidade do Minho
- Grupo ISLab – (Synthetic Intelligence Lab)
- Centro ALGORITMI
  Universidade do Minho

Convolutional Neural Networks

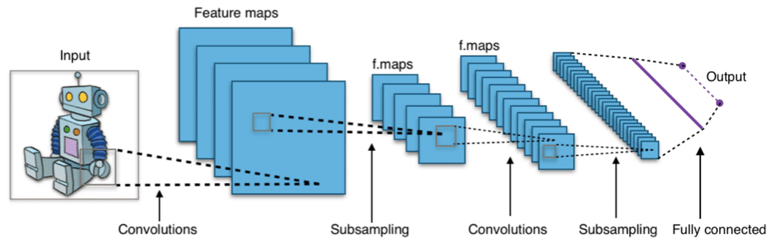


Convolutional Neural Networks (CNN's): what are they for?

- Applied when you have data that doesn't neatly align into columns
  - Images that you want to find features within
  - Machine translation
  - Sentence classification
  - Sentiment analysis
- They can find features that aren't in a specific spot
  - Like a stop sign in a picture
  - Or words within a sentence
- They are "feature-location invariant"

Convolutional Neural Networks (CNN's): how do they work?

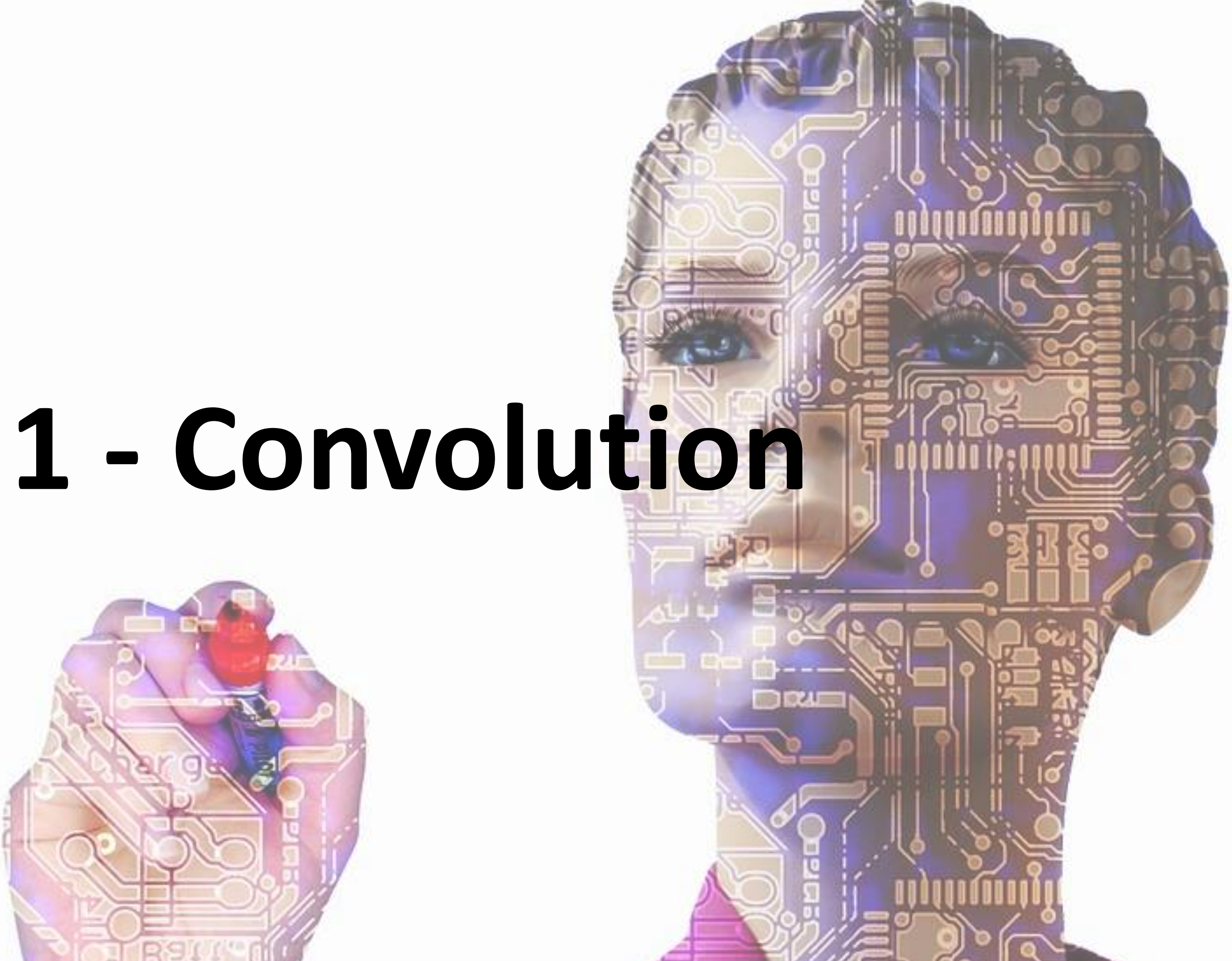
- Inspired by the biology of the visual cortex
  - Local receptive fields are groups of neurons that only respond to a part of what your eyes see (sub-sampling)
  - They overlap each other to cover the entire visual field (convolutions)
  - They feed into higher layers that identify increasingly complex images
    - Some receptive fields identify horizontal lines, lines at different angles, among other features (called feature maps or filters)
    - These would feed into a layer that identifies shapes
    - Which might feed into a layer that identifies objects
  - For color or RGB images, 3 layers are used to represent red, green and blue layers

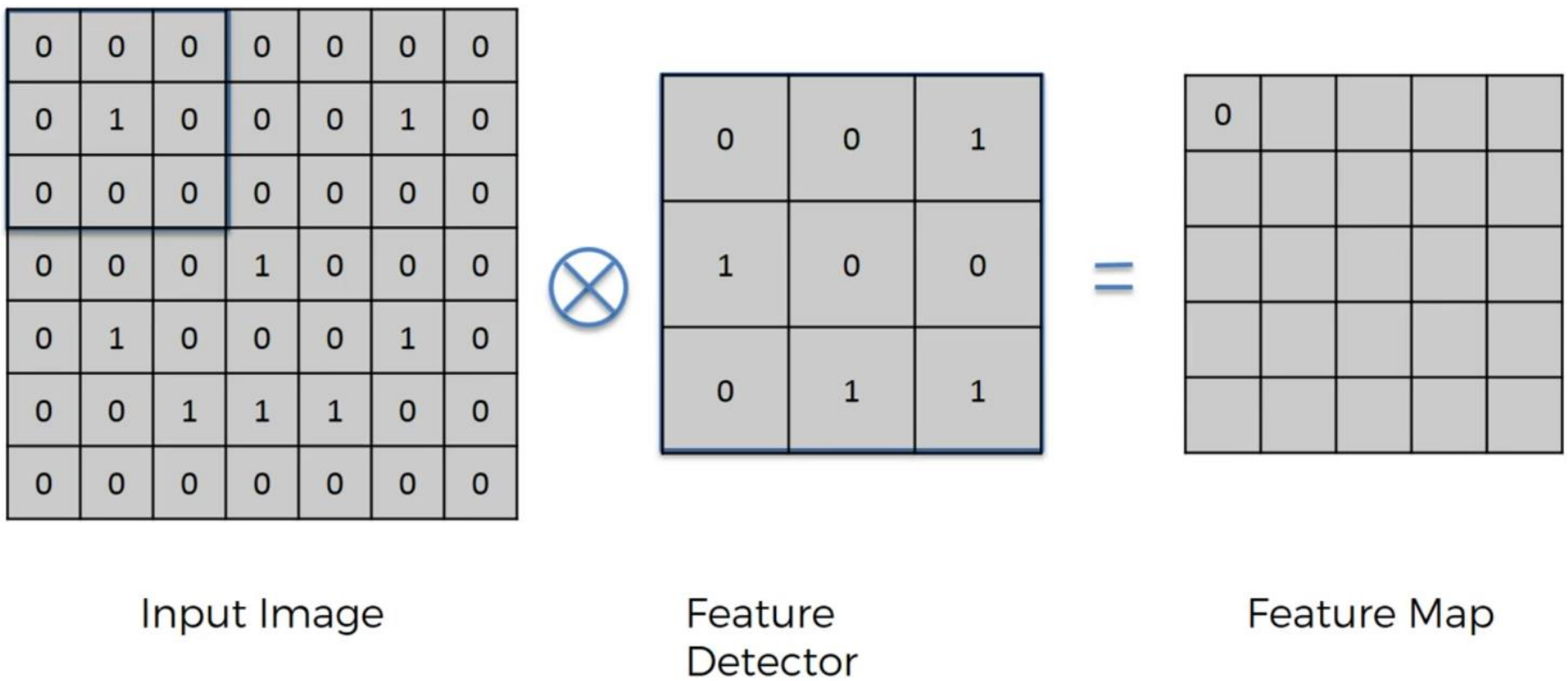


How do we know a traffic signal is a stop sign?

- Individual local receptive fields scan the image looking for edges, and pick up the edges of the stop sign in a layer
- Those edges are used by a higher-level convolution that identifies the stop sign's shape (among other features, e.g., letters)
- The shape then gets matches against the pattern of what a stop sign looks like, also using the strong red signal coming from the red layers
- The information keeps getting processed upward until a decision is made (i.e., classification)
- A CNN works the same way

Step 1 - Convolution




0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0
Input Image
Feature Detector
Feature Map

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

$\otimes$

0	0	1
1	0	0
0	1	1

Feature Detector

=

0	1			

Feature Map

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

$\otimes$

0	0	1
1	0	0
0	1	1

Feature Detector

=

0	1	0	0	0
0	1	1	1	0
1	0	1	2	

Feature Map

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

$\otimes$

0	0	1
1	0	0
0	1	1

Feature Detector

$=$

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

Feature Map

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

$\otimes$

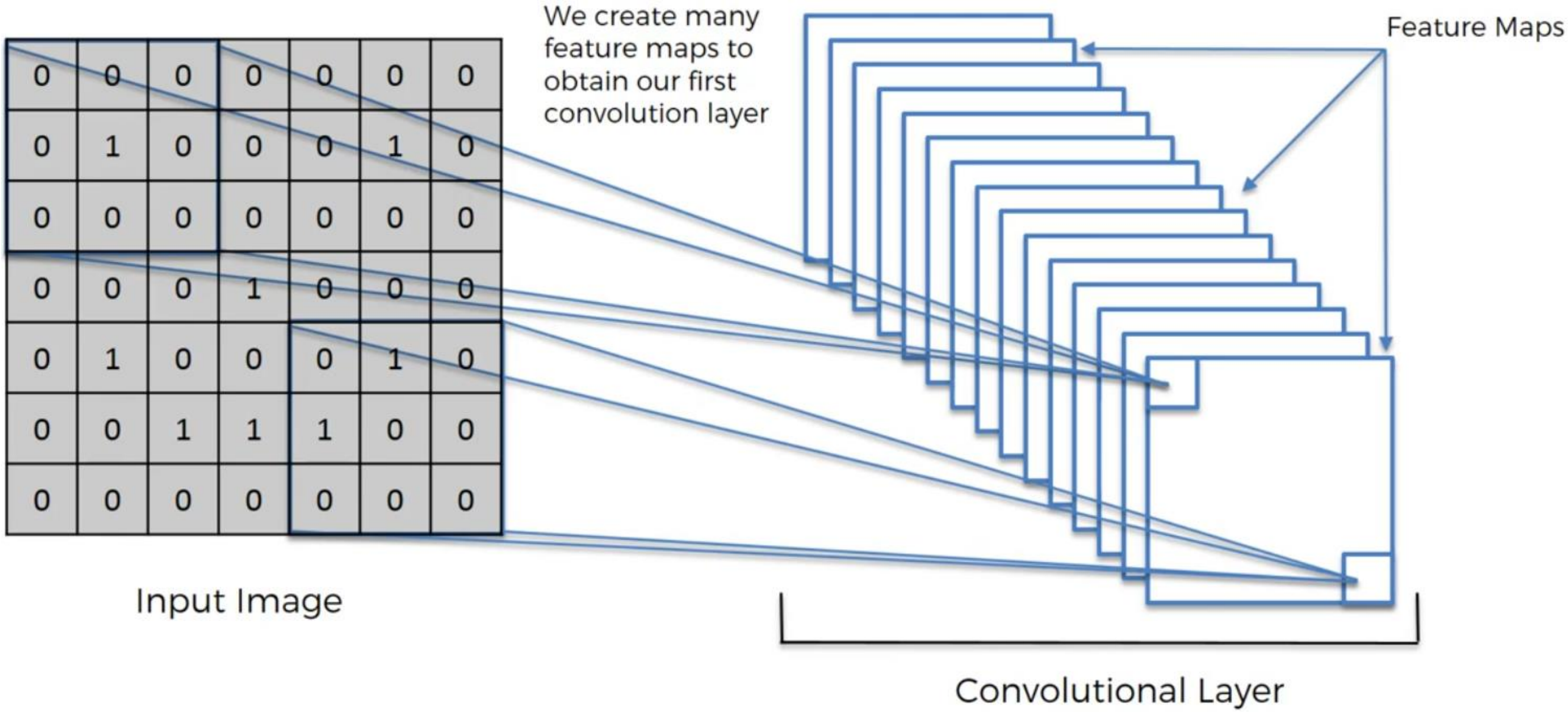
0	0	1
1	0	0
0	1	1

Feature Detector

=

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

Feature Map


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
We create many feature maps to obtain our first convolution layer
Feature Maps
Input Image
Convolutional Layer

Sharpen:

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

Image Source: docs.gimp.org/en/plug-in-convmatrix.html

Blur:

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Image Source: docs.gimp.org/en/plug-in-convmatrix.html

Emboss:

	-2	-1	0	
	-1	1	1	
	0	1	2	

Image Source: docs.gimp.org/en/plug-in-convmatrix.html

Edge Enhance:

	0	0	0	
	-1	1	0	
	0	0	0	

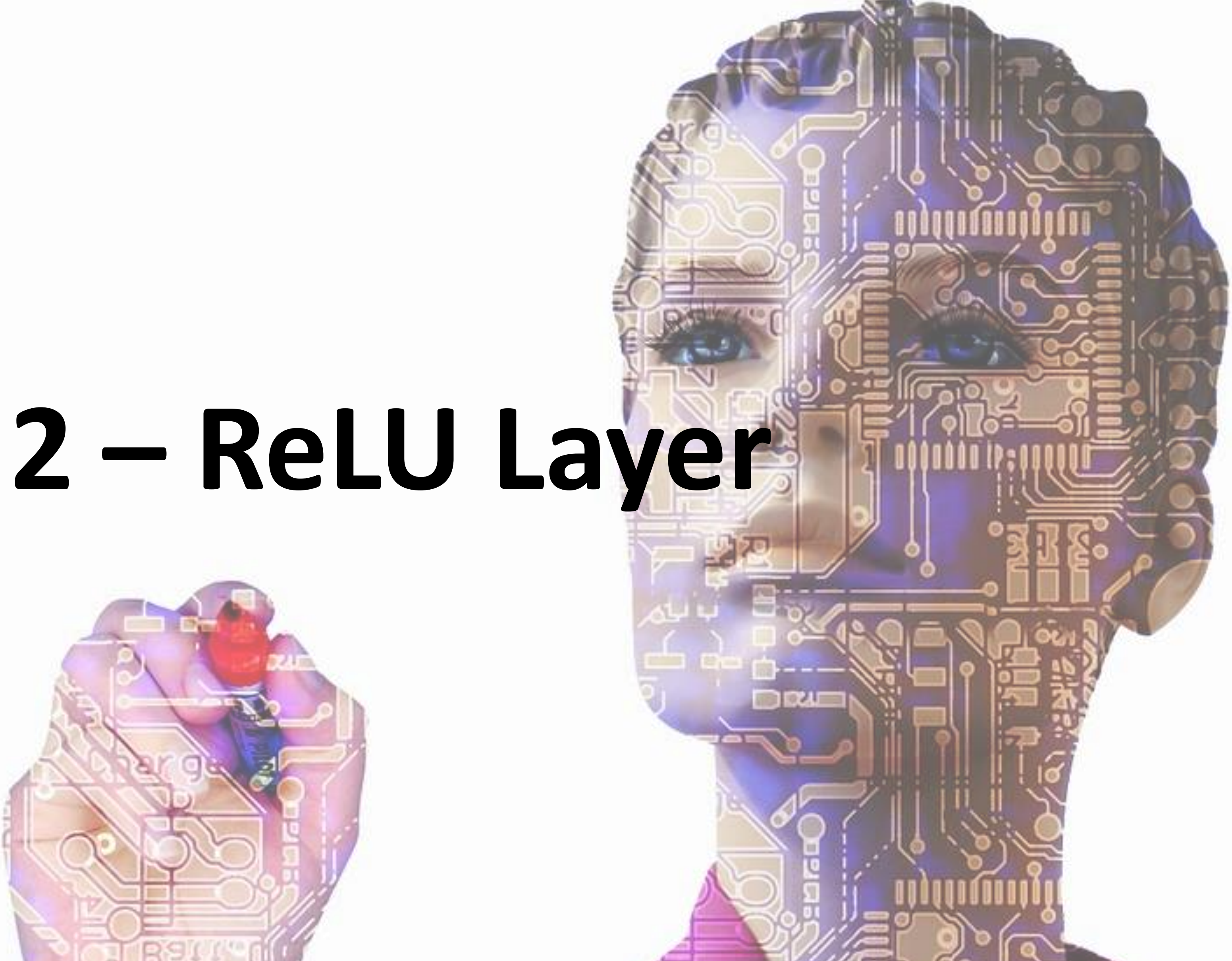
Image Source: docs.gimp.org/en/plug-in-convmatrix.html

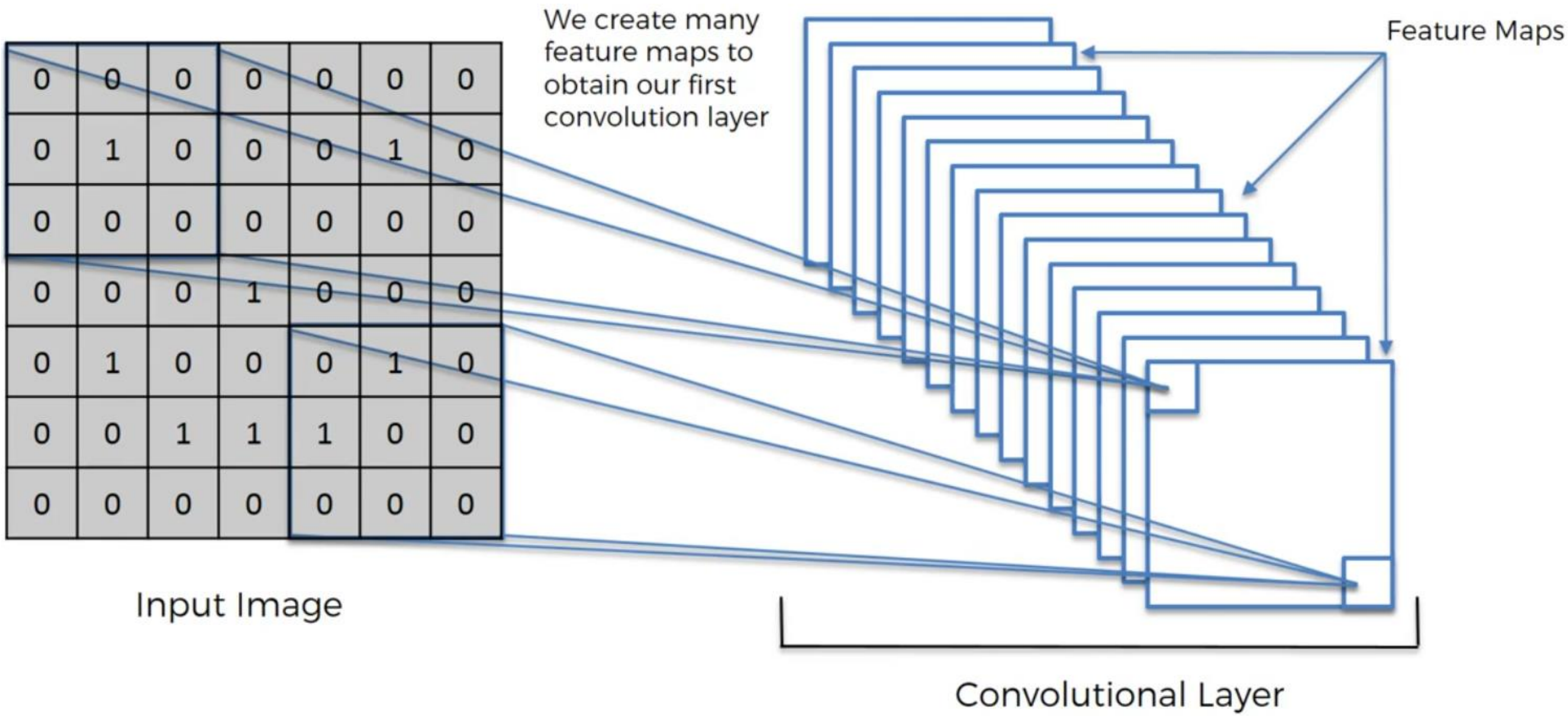
Edge Detect:

	0	1	0	
	1	-4	1	
	0	1	0	

Image Source: docs.gimp.org/en/plug-in-convmatrix.html

Step 2 – ReLU Layer




We create many feature maps to obtain our first convolution layer
Feature Maps
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolutional Layer

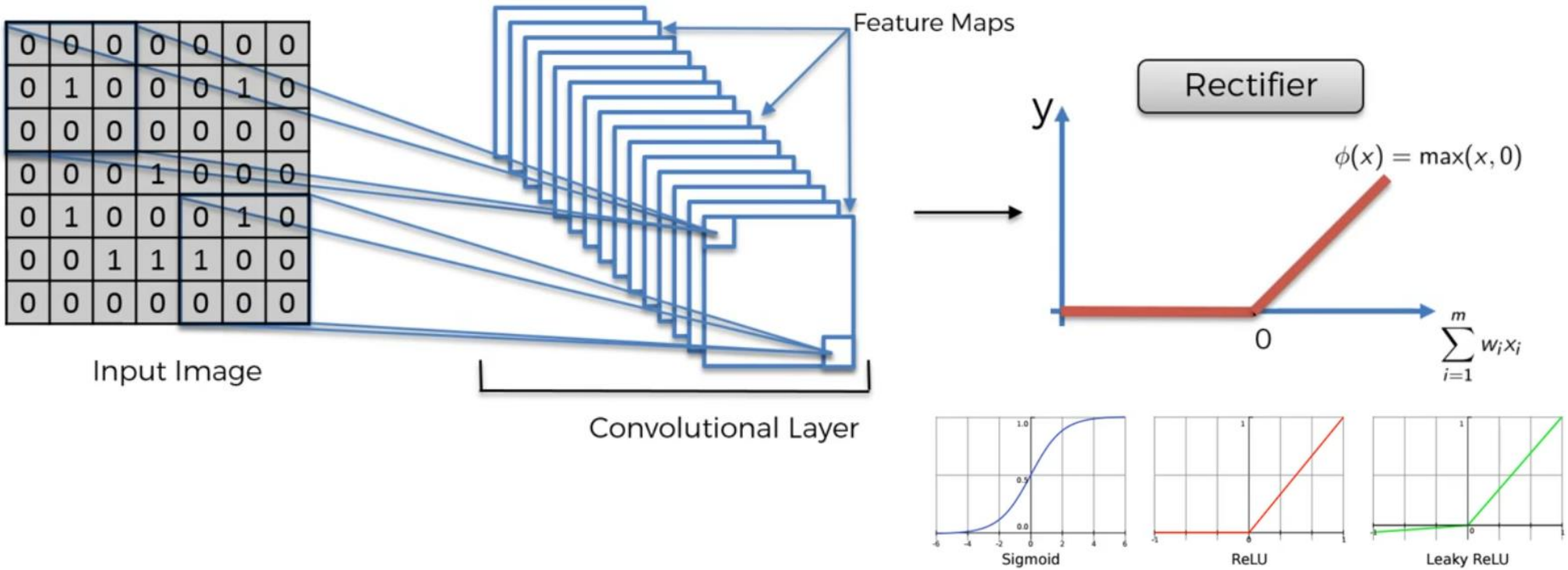
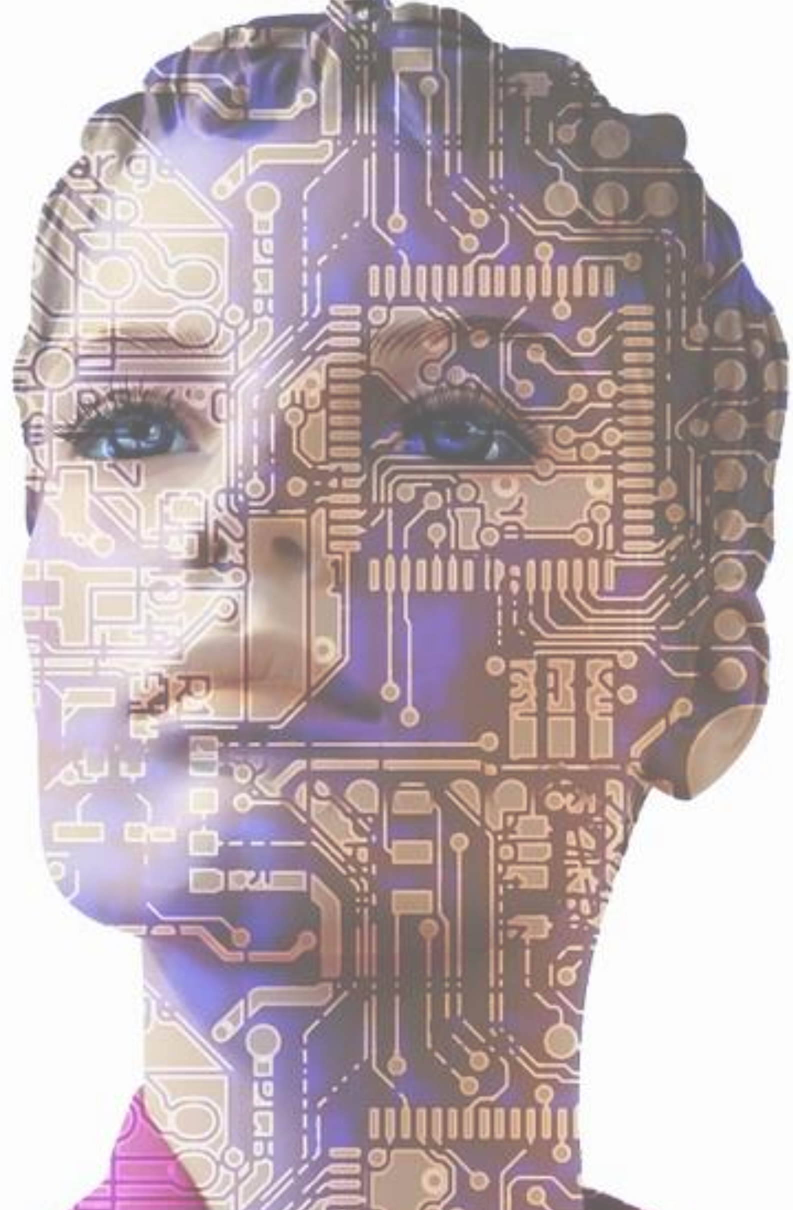

Input Image
Feature Maps
Convolutional Layer
Rectifier
y
$\phi(x) = \max(x, 0)$
0
$\sum_{i=1}^{m} w_i x_i$
Sigmoid
ReLU
Leaky ReLU

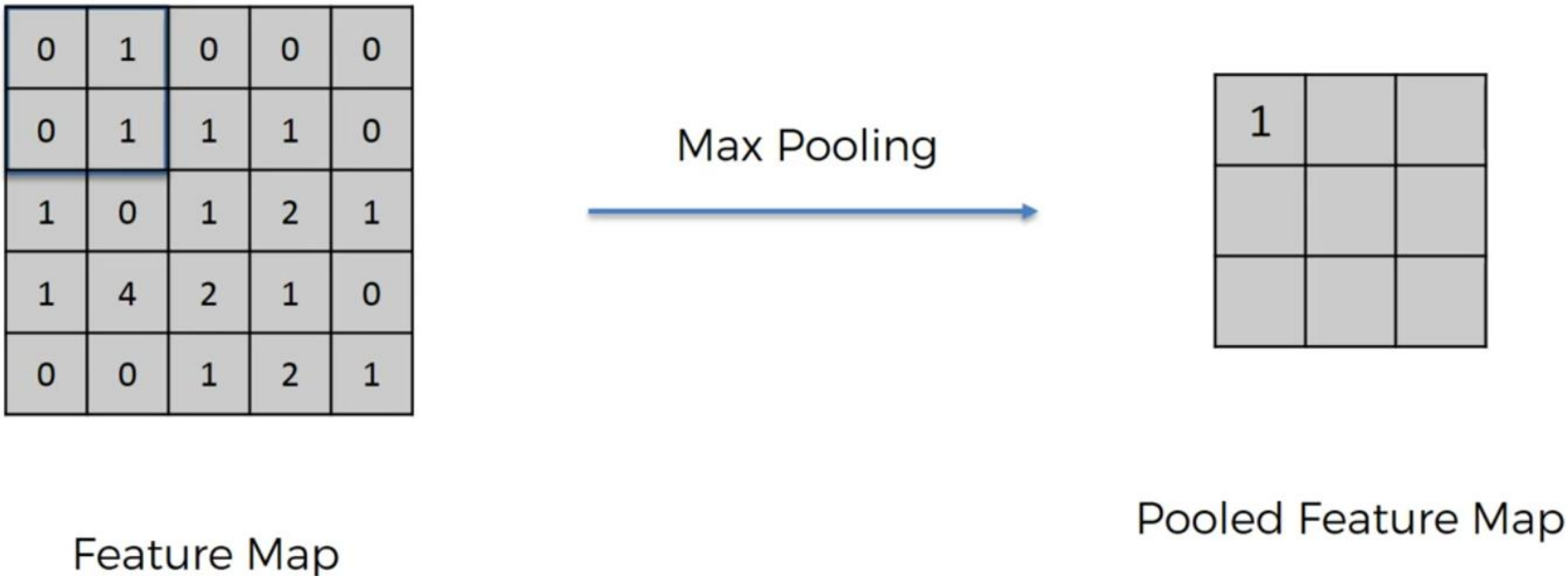
Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

Step 3 – Pooling




0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Max Pooling
1
Feature Map
Pooled Feature Map

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

Feature Map

Max Pooling $\longrightarrow$

1	1	

Pooled Feature Map

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

Feature Map

Max Pooling

1	1	0

Pooled Feature Map

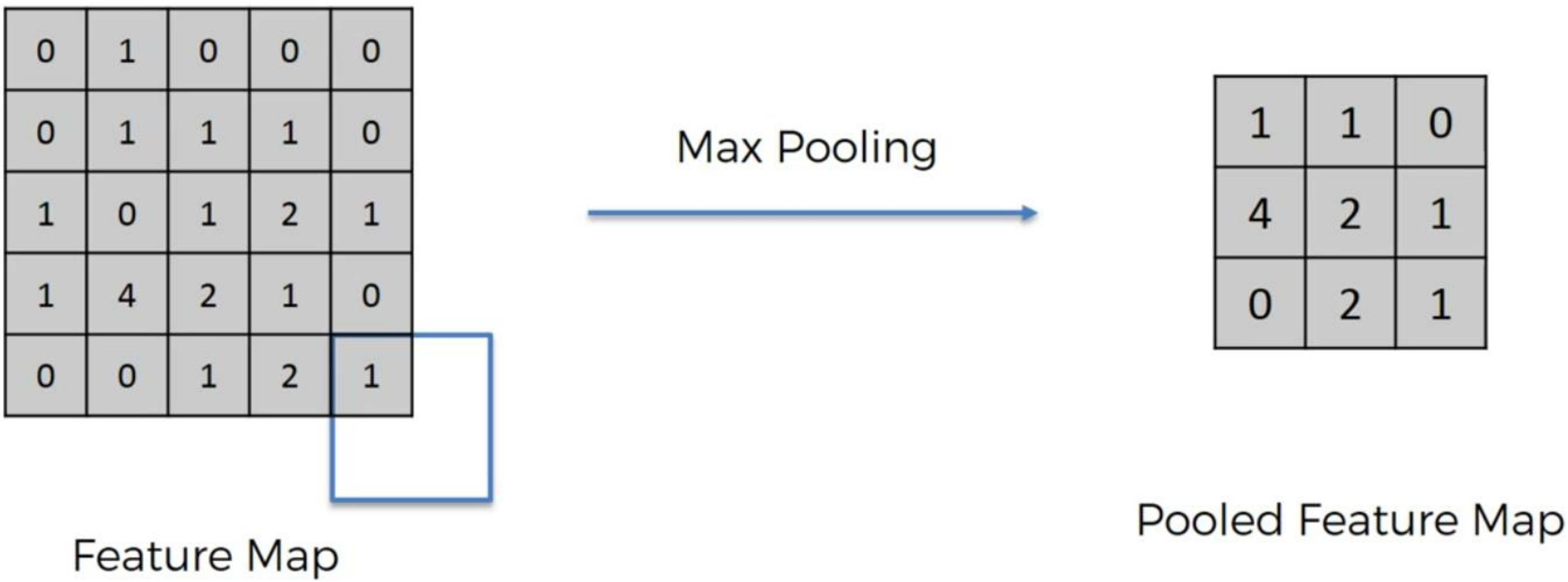
0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

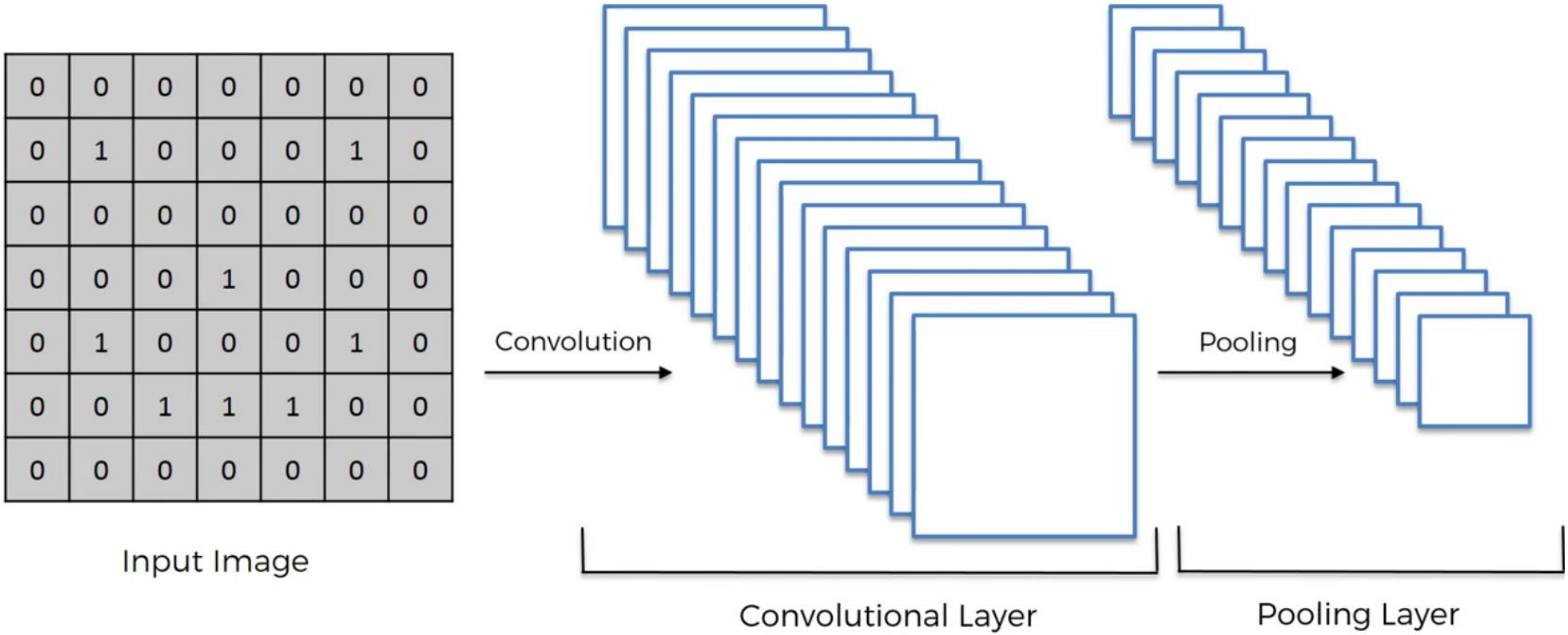
Feature Map

Max Pooling →

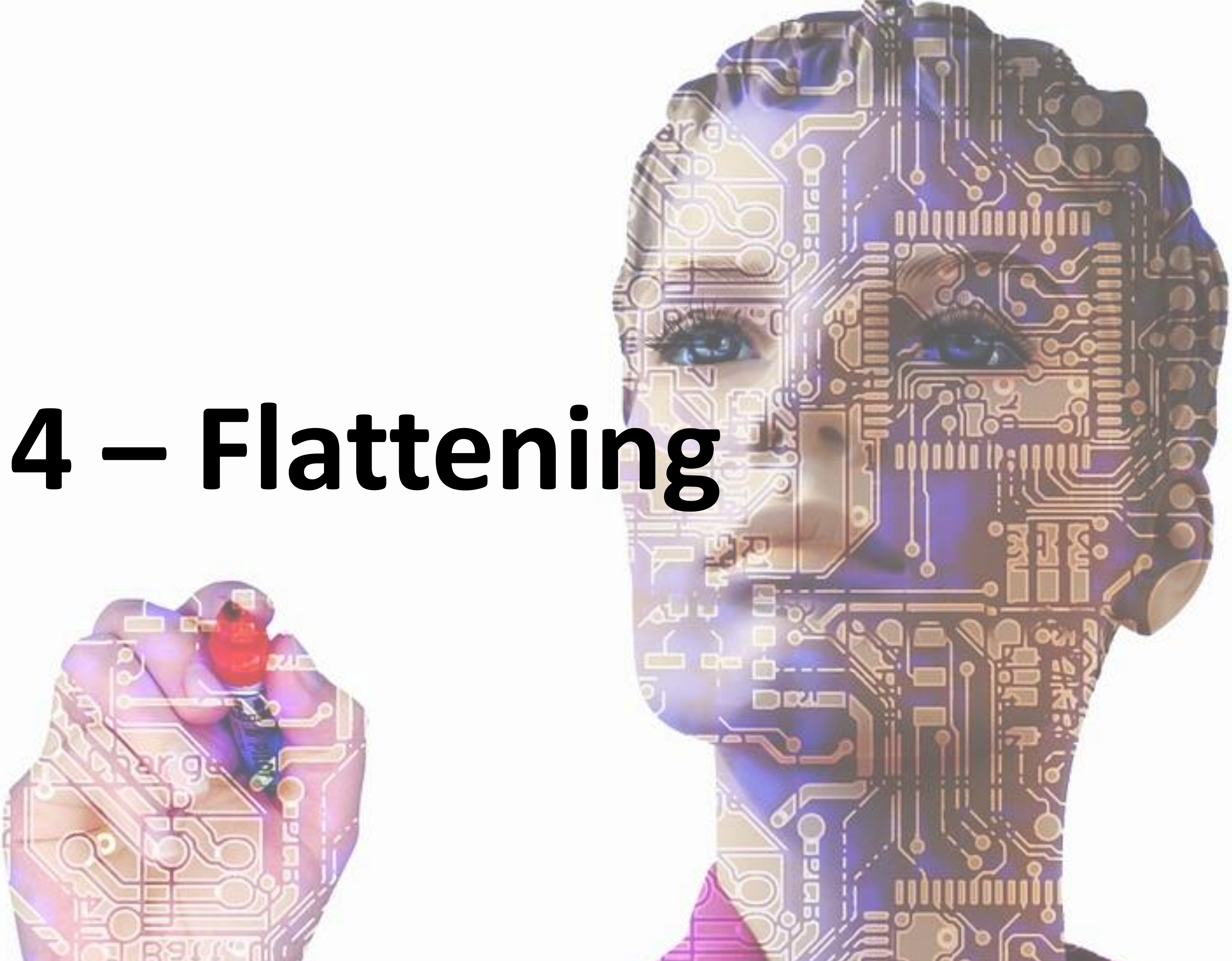
1	1	0
4		

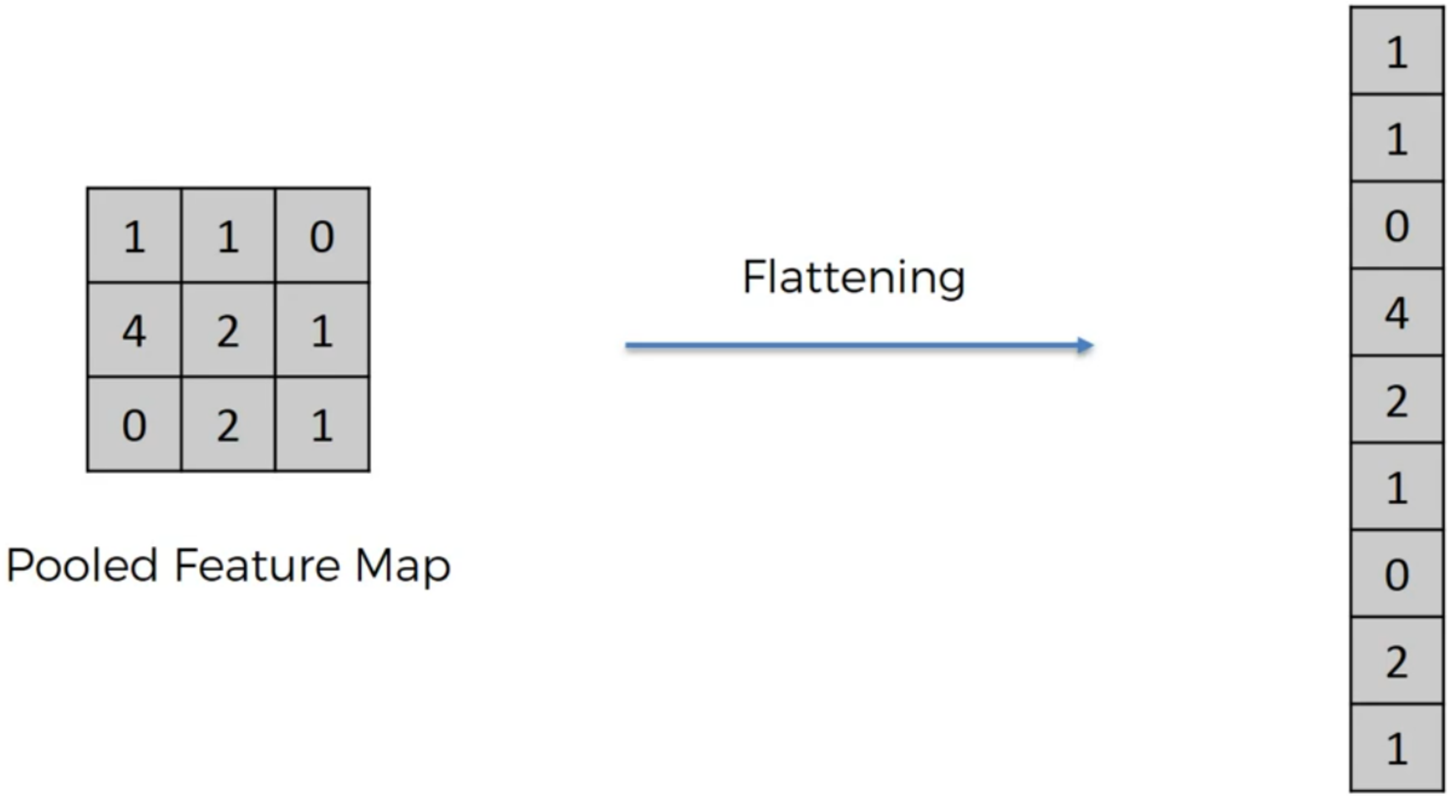
Pooled Feature Map

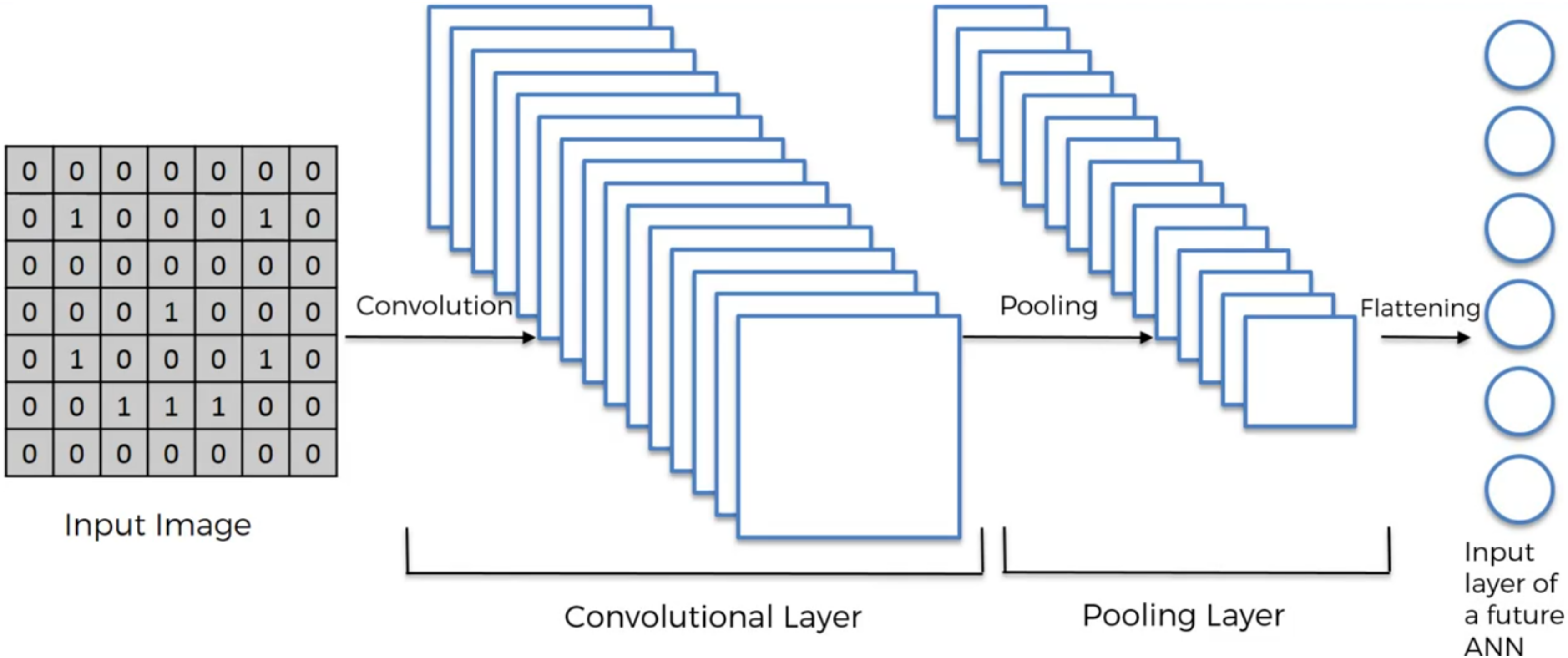

0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Feature Map
Max Pooling
1 1 0
4 2 1
0 2 1
Pooled Feature Map


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Convolutional Layer
Pooling
Pooling Layer

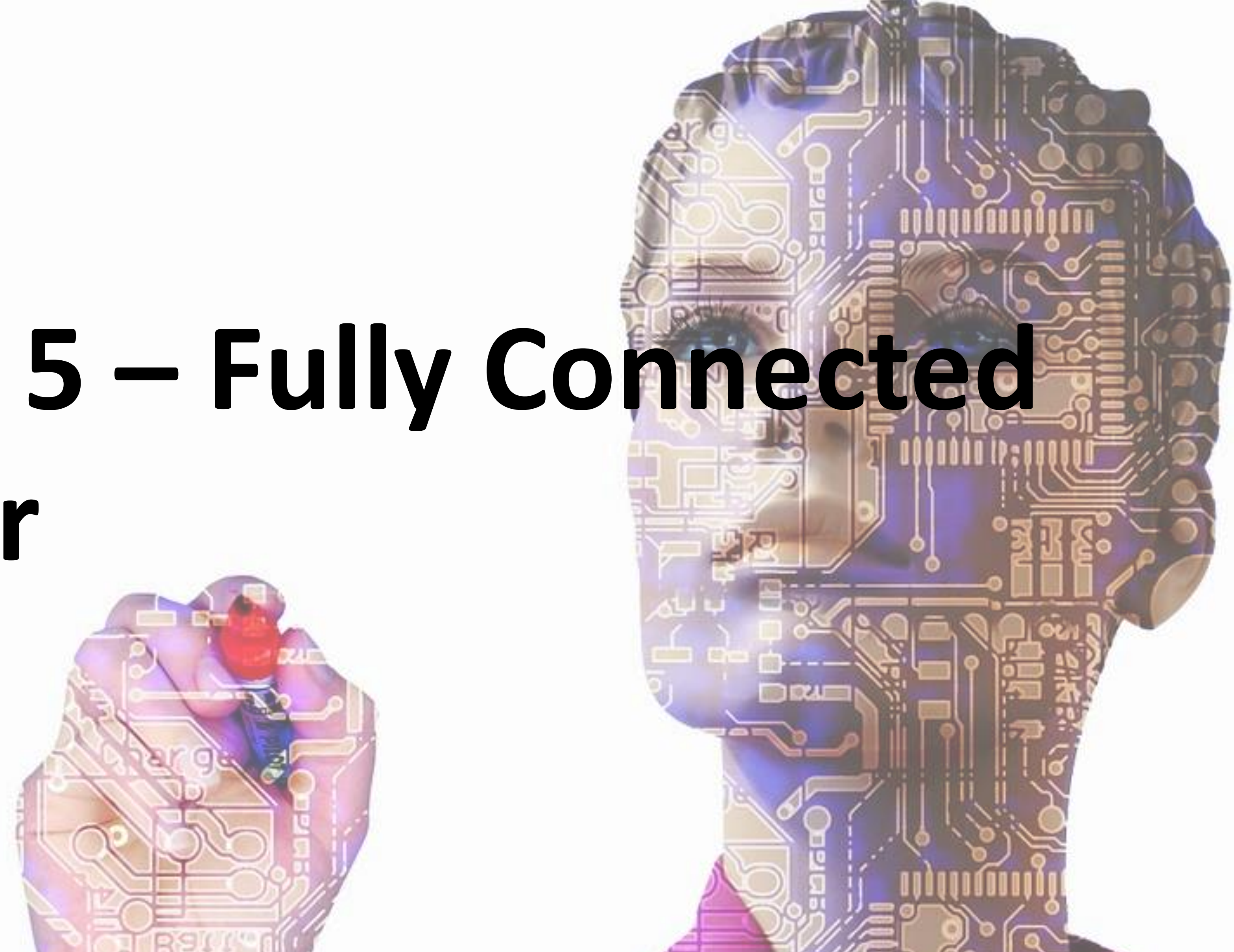
Step 4 – Flattening

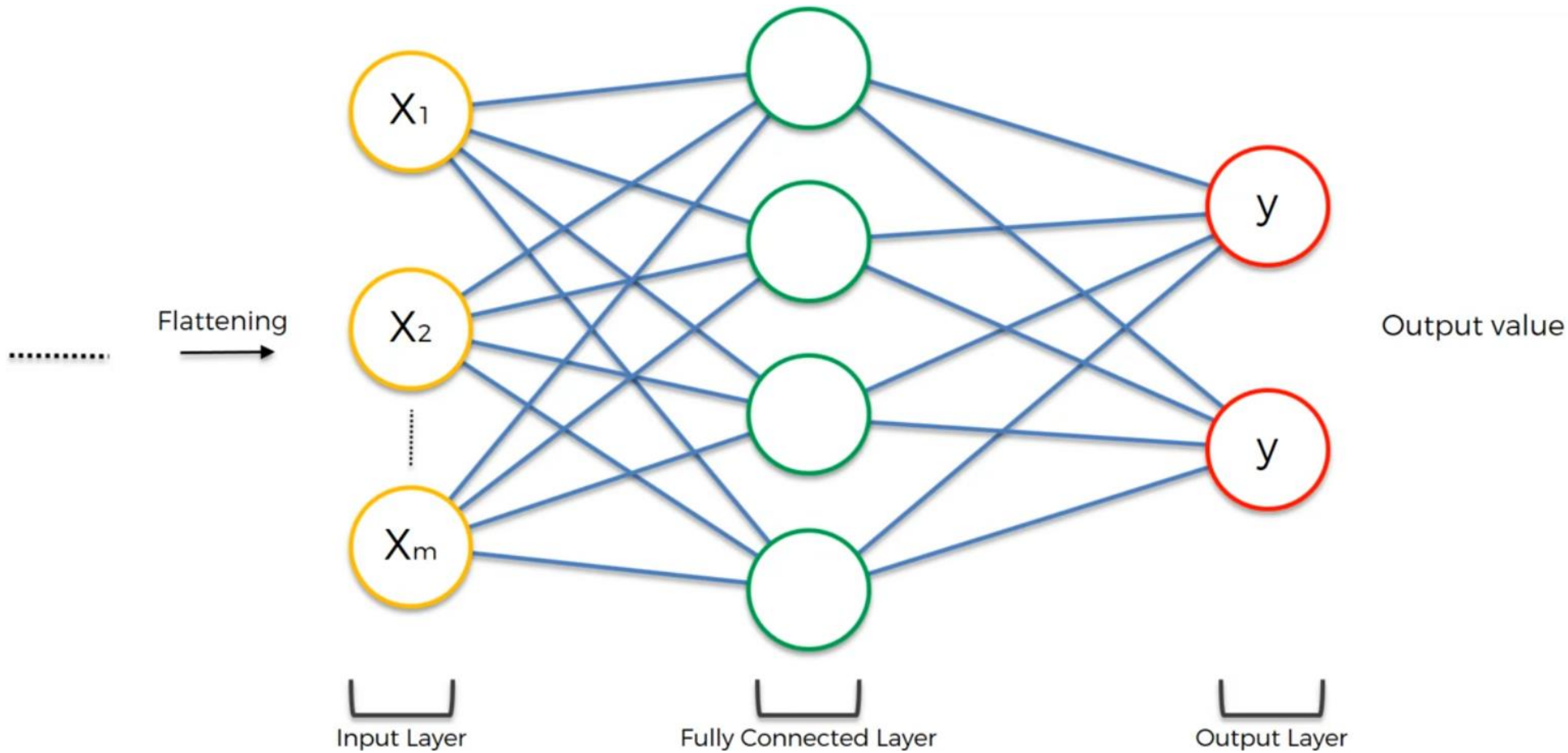


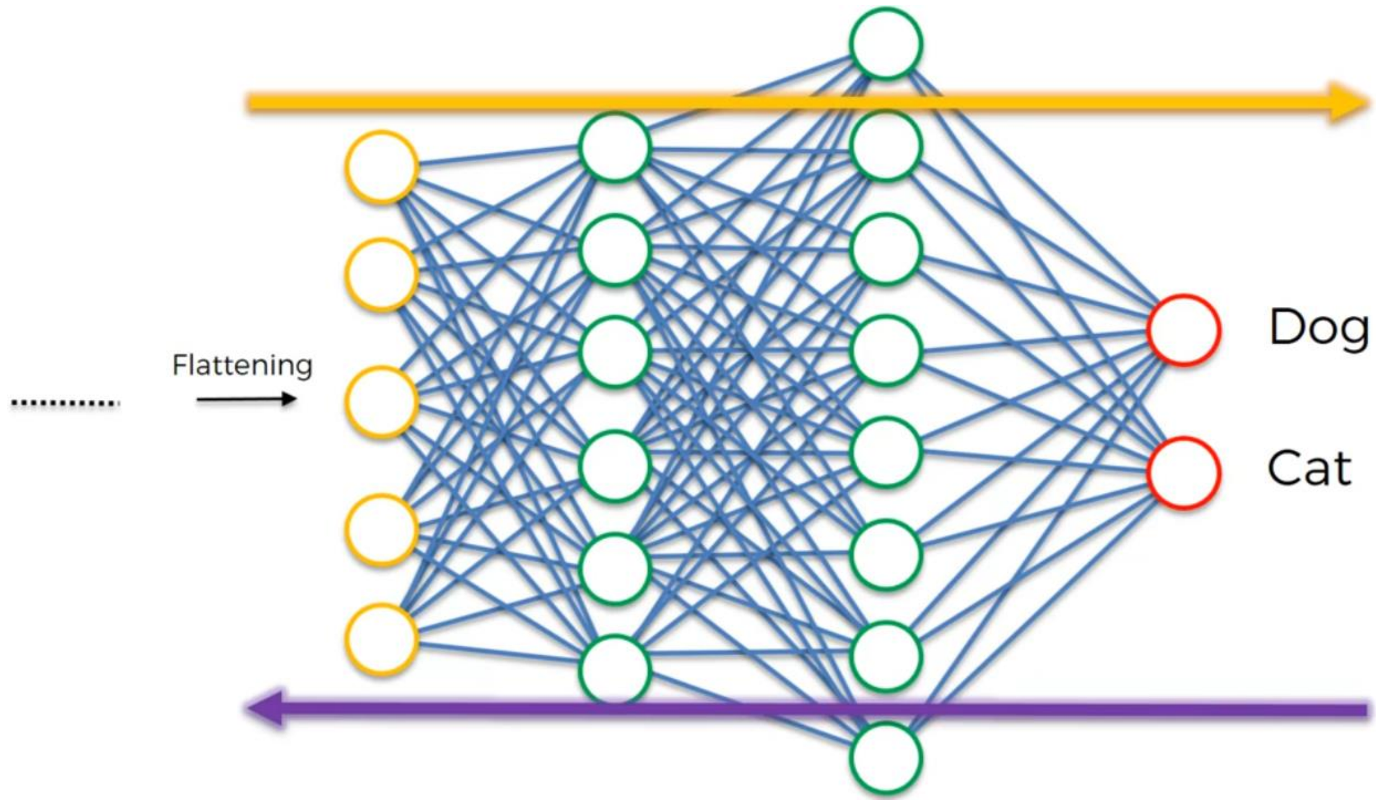

1 1 0
4 2 1
0 2 1
Pooled Feature Map
Flattening
1
1
0
4
2
1
0
2
1

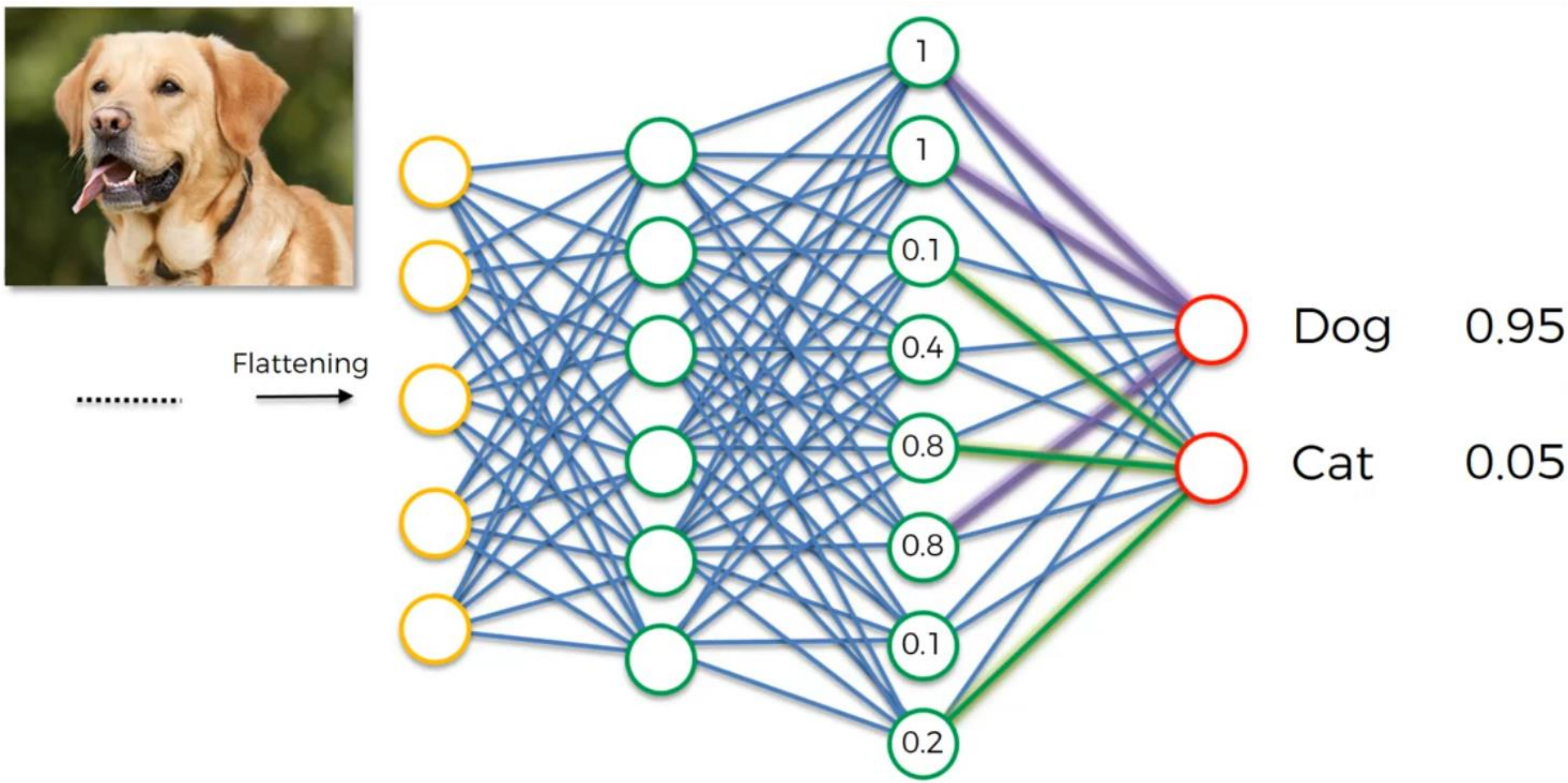

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Pooling
Flattening
Convolutional Layer
Pooling Layer
Input layer of a future ANN

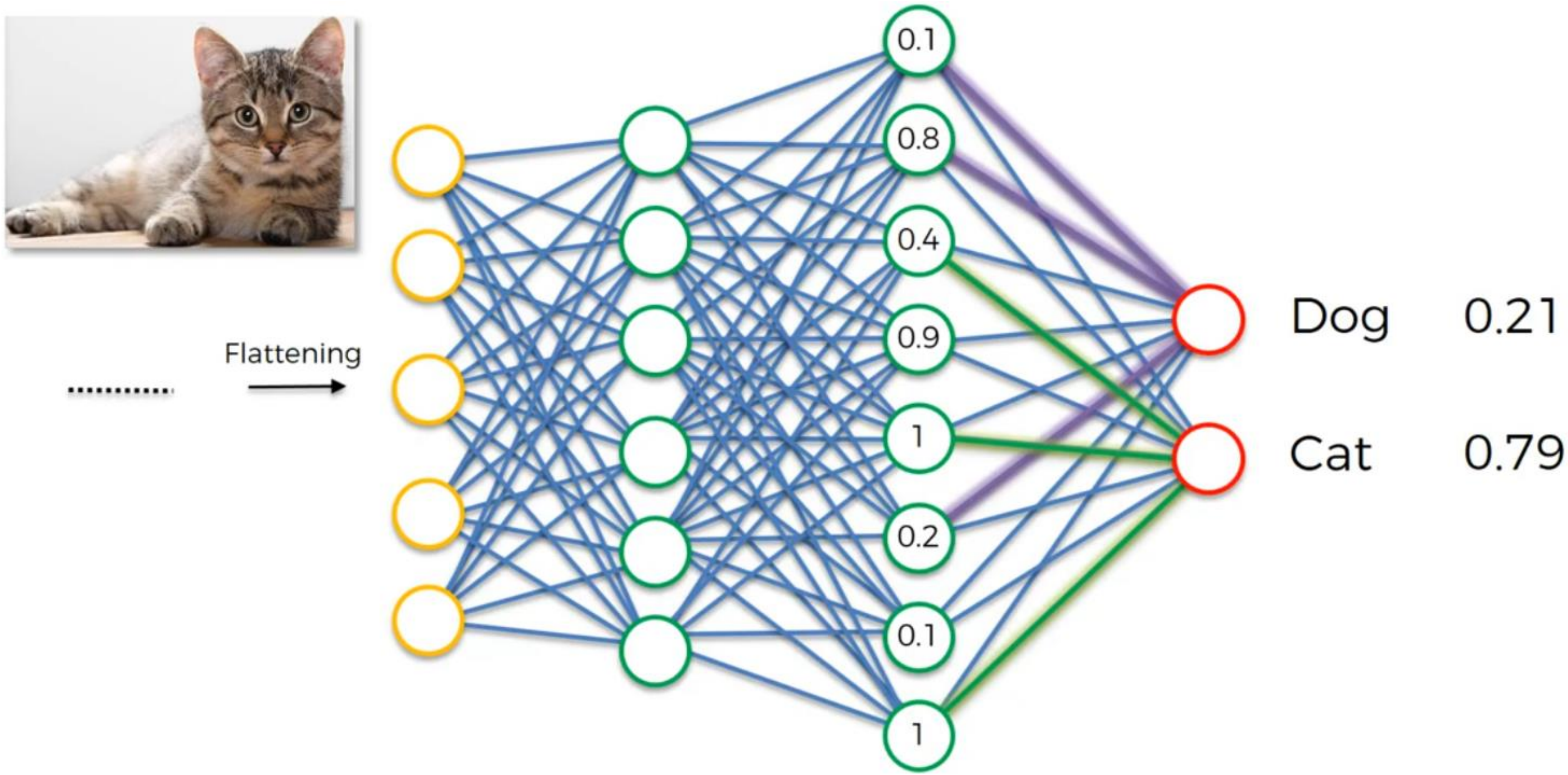
Step 5 – Fully Connected Layer




Flattening
X1
X2
Xm
y
y
Output value
Input Layer
Fully Connected Layer
Output Layer


Flattening
Dog
Cat


Flattening
1
1
0.1
0.4
0.8
0.8
0.1
0.2
Dog 0.95
Cat 0.05


Flattening
0.1
0.8
0.4
0.9
1
0.2
0.1
1
Dog
0.21
Cat
0.79

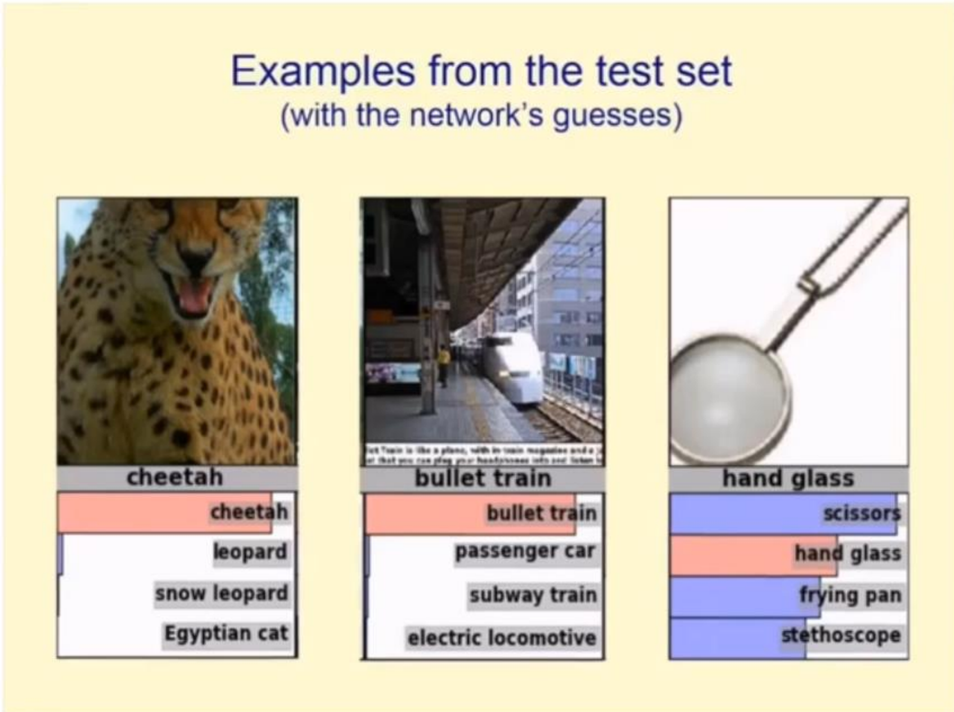
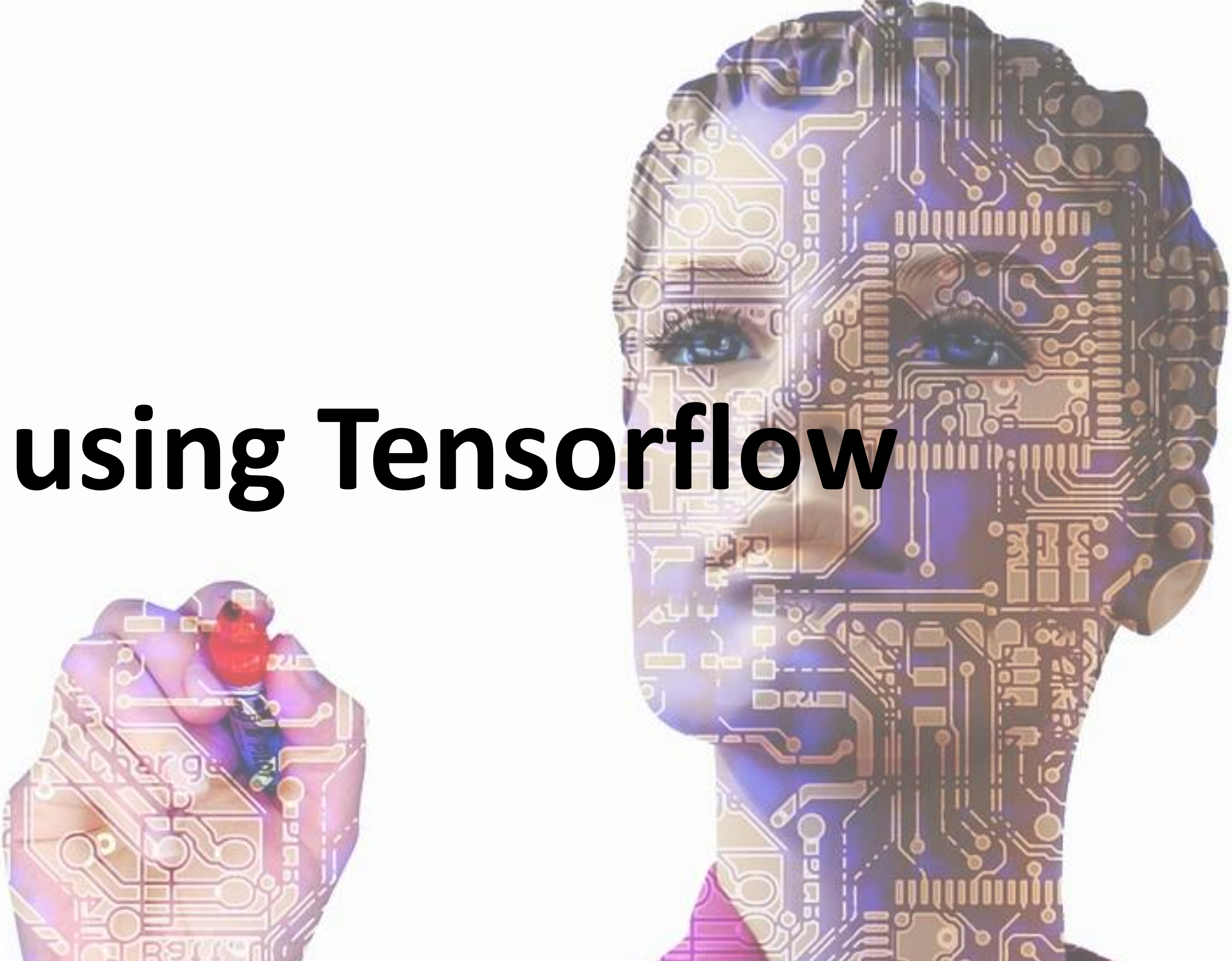


Image Source: a talk by Geoffrey Hinton

CNN using Tensorflow



CNN's with Tensorflow

- Source data must be of appropriate dimensions
  - i.e., width x height x color channels
- Conv2D layer type does the actual convolution on a 2D image
  - Conv1D and Conv3D also available – doesn't have to be image data, e.g., Signal Data and Video Data
- MaxPooling2D layers can be used to reduce a 2D layer down by taking the maximum value in a given kernel
- Flatten layers will convert the 2D layer to a 1D layer for passing into a flat hidden layer of neurons
- Typical architecture use:
  - Conv2D -> MaxPooling2D -> Dropout -> Flatten -> Dense -> Dropout -> Softmax

CNN’s are resource-intensive

- Uses a lot of computational resources (CPU, GPU and RAM)
- Lots of hyper-parameters
    - Kernel sizes, multiple layers with different number of units, amount of pooling, number of layers, choice of optimizers, etc.
- Getting the training data is often the hardest part (as well as storing and accessing it)

Specialized CNN architectures

- Defines specific arrangement of layers, padding, and hyper-parameters
- LeNet-5
  - Good for handwriting recognition
- AlexNet
  - Image Classification, deeper than LeNet
- VGG
  - Upgrade version of AlexNet
  - Used in multiple contexts with good overall performance
- GoogLeNet
  - Even deeper, but with better performance
  - Introduces inception modules (groups of convolution layers)
- ResNet (Residual Network)
  - Even deeper – maintains performance via skip connections

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Mestrado Integrado em Engenharia Informática
Mestrado em Engenharia Informática
Computação Natural
2020/2021

Filipe Gonçalves, Paulo Novais