



g

b

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

D

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

b)

Location shifted

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1



1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
1	1	-1	-1	-1
1	-1	-1	-1	-1



-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Location shifted



1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1



1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1



-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

Variation 1

-1	-1	1	-1	-1
-1	1	-1	1	-1
-1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1



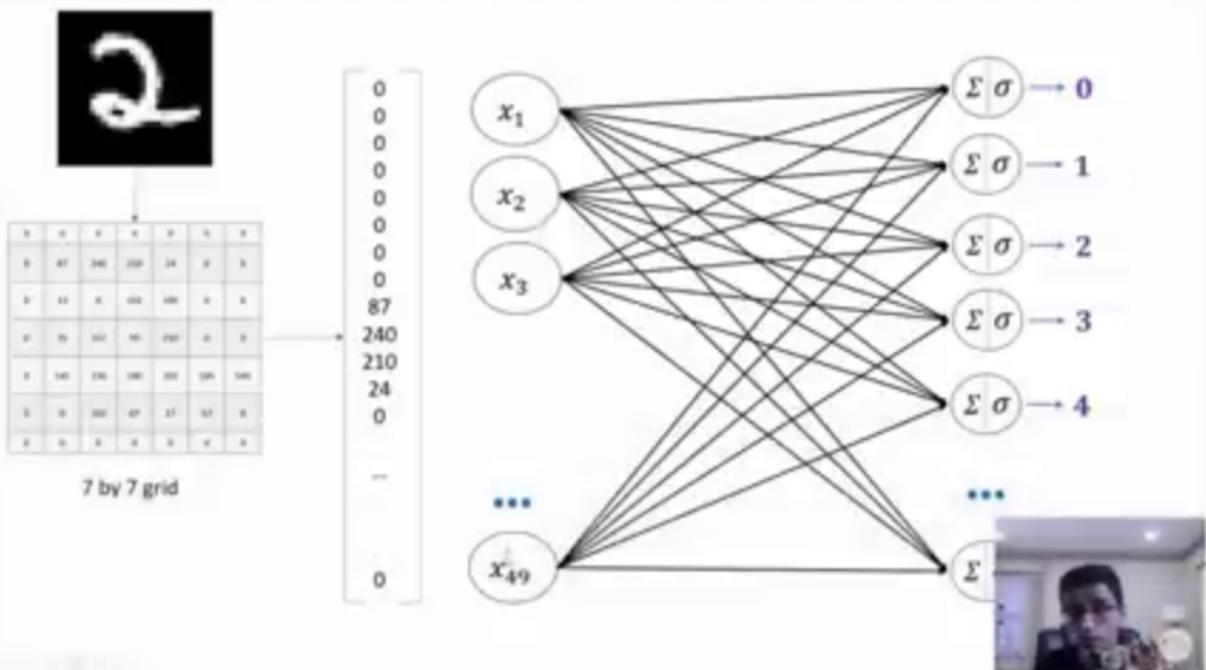
-1	-1	1	-1	-1
-1	1	-1	1	-1
-1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1



-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

To handle **variety** in digits we can use simple artificial neural network (ANN)





Neural Network For Handwritten Digits Classification | Deep Learning Tutorial 7 (Tensorflow2.0)

10,172 views • Published Jul 16, 2020

0:00 / 1:11

SHARE

SAVE

ANALYTICS

EDIT VIDEO



codebasics
475K subscribers

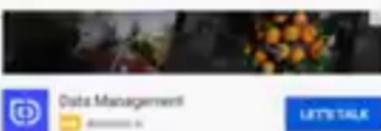
In this video we will build our first neural network in tensorflow and python for handwritten digits classification. We will first build a very simple neural network with only input and output layer. After that we will add a hidden layer and check how the performance of our model changes.

SHOW MORE

Deep Learning With Tensorflow 2.0, Keras and Python

codebasics - 7: 11

- 1 Neural Network | Deep Learning... 11:45
- 2 Install tensorflow 2.0 | Deep... 2:37
- 3 Pytorch vs Tensorflow vs Keras |... 2:17
- 4 Neural Network For Handwritten... 16:29
- 5 Activation Functions | Deep... 14:29
- 6 Derivatives | Deep Learning... 10:58
- 7 Back Propagation | Deep Learning... 10:58
- 8 Matrix Basics | Deep Learning... 10:58



Data Management

LET'S TALK



codebasics

Build a campaign in 5 steps
Start bringing new customers to the place where business is done
[Business LinkedIn.com](#)

visit site

2020 Machine Learning Roadmap

2020 Machine Learning Roadmap

David Howell

279K views

To handle **variety** in digits we can use simple artificial neural network (ANN)

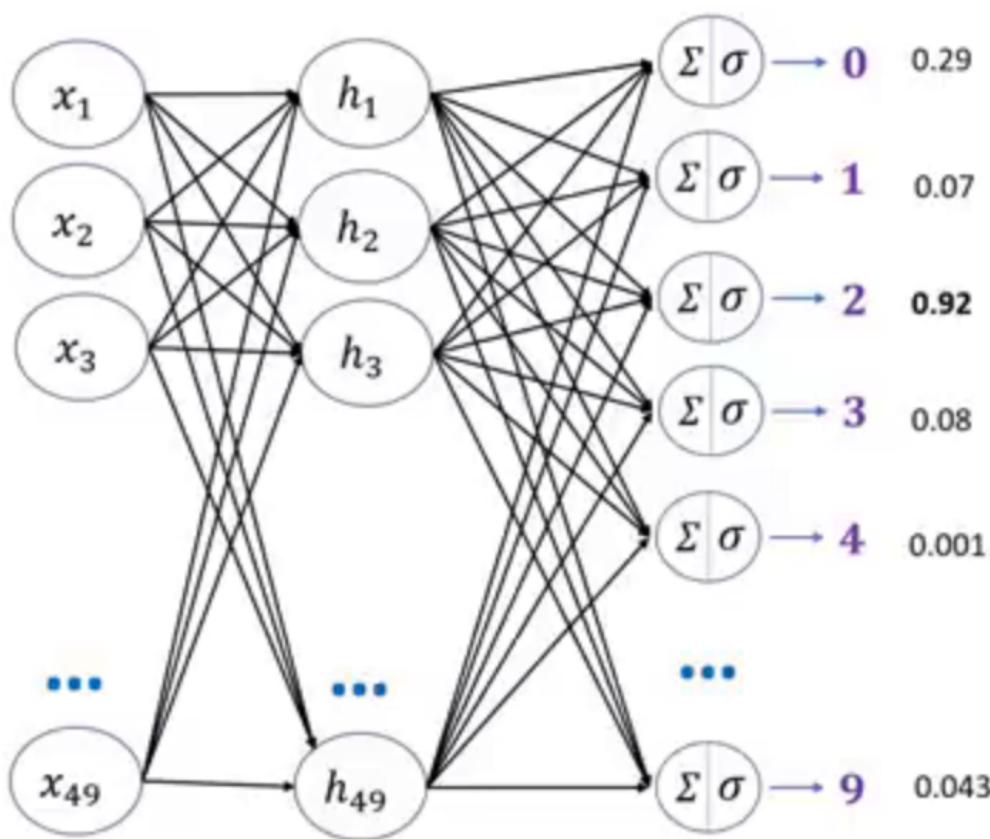




0	0	0	0	0	0	0	0
0	87	240	230	24	0	0	0
0	11	0	101	195	0	0	0
0	35	167	99	233	0	0	0
0	145	230	240	205	189	140	0
0	0	102	67	17	13	0	0
0	0	0	0	0	0	0	0

7 by 7 grid

0	0	0	0	0	0	0	0
0	87	240	230	24	0	0	0
0	11	0	101	195	0	0	0
0	35	167	99	233	0	0	0
0	145	230	240	205	189	140	0
0	0	102	67	17	13	0	0
0	0	0	0	0	0	0	0



$$49 * 49 = 2401$$

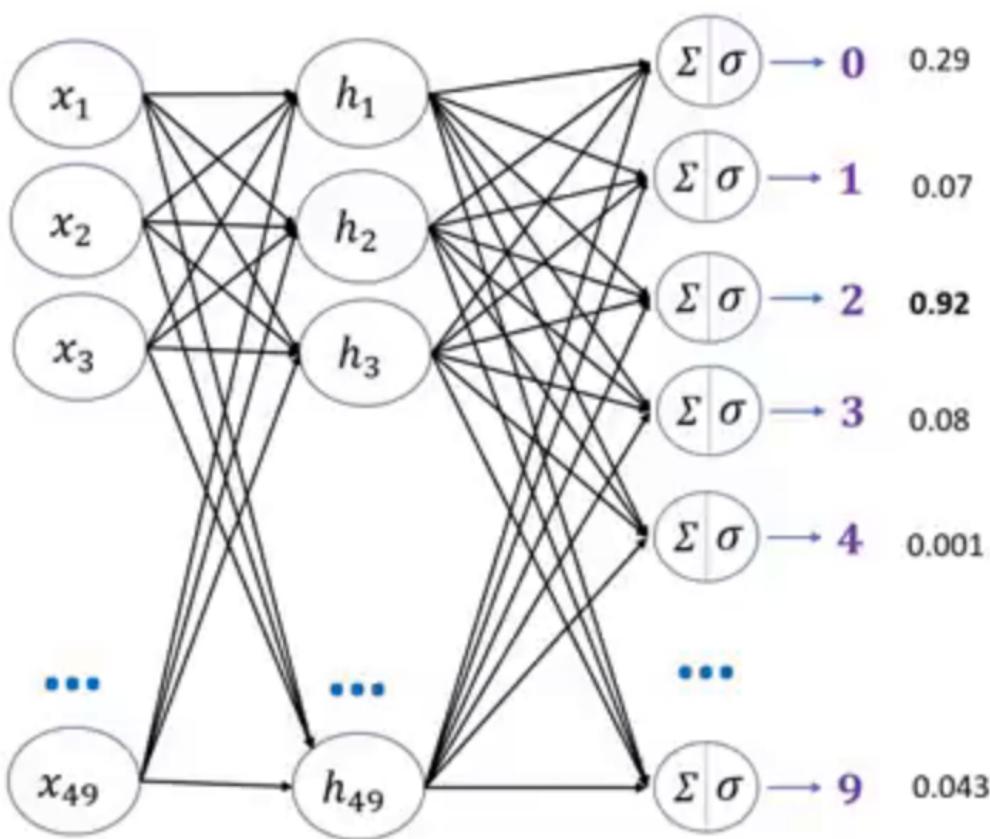
$$49 * 10 = 490$$



0	0	0	0	0	0	0	0
0	87	240	230	24	0	0	0
0	11	0	101	195	0	0	0
0	35	167	99	233	0	0	0
0	145	230	240	205	189	140	0
0	0	102	67	17	13	0	0
0	0	0	0	0	0	0	0

7 by 7 grid

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 87 \\ 240 \\ 210 \\ 24 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$



$$49 * 49 = 2401$$

$$49 * 10 = 490$$

Image size = 1920 x 1080 X 3



Image size = 1920 x 1080 X 3



First layer neurons = 1920 x 1080 X 3 ~ 6 million

b

A photograph of a koala sitting on a tree branch, facing slightly to the left. It has its characteristic white fur around its eyes and ears, and greyish-blue fur on its body.

Image size = 1920 x 1080 X 3

First layer neurons = 1920 x 1080 X 3 ~ 6 million

b

Hidden layer neurons = Let's say you keep it ~ 4 million

Weights between input and hidden layer = $6 \text{ mil} * 4 \text{ mil}$
= 24 million



Image size = 1920 x 1080 X 3

First layer neurons = 1920 x 1080 X 3 ~ 6 million

b

Hidden layer neurons = Let's say you keep it ~ 4 million

Weights between input and hidden layer = $6 \text{ mil} * 4 \text{ mil}$
= 24 million



Image credit: <https://giphy.com/> (@rabbids)

Disadvantages of using ANN for image classification

1. Too much computation
2. Treats local pixels same as pixels far apart
3. Sensitive to location of an object in an image

Disadvantages of using ANN for image classification

1. Too much computation
2. Treats local pixels same as pixels far apart
3. Sensitive to location of an object in an image

Disadvantages of using ANN for image classification

1. Too much computation
2. Treats local pixels same as pixels far apart
3. Sensitive to location of an object in an image

Disadvantages of using ANN for image classification

1. Too much computation
2. Treats local pixels same as pixels far apart
3. Sensitive to location of an object in an image

**HOW DOES HUMANS
RECOGNIZE IMAGES SO EASILY?**



**HOW DOES HUMANS
RECOGNIZE IMAGES SO EASILY?**



Koala's **eye**? = Y



Koala's **nose**? = Y



Koala's **ears**? = Y



Koala's **eye**? = Y



Koala's **nose**? = Y



Koala's **ears**? = Y



Koala's **eye**? = Y



Koala's **nose**? = Y



Koala's **ears**? = Y





Koala's **eye**? = Y



Koala's **nose**? = Y



Koala's **ears**? = Y



Koala's **head**? = Y





Koala's **eye**? = Y



Koala's **nose**? = Y



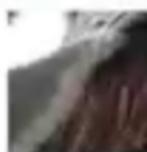
Koala's **ears**? = Y



Koala's **head**? = Y



Koala's **hands**? = Y



Koala's **legs**? = Y





Koala's **eye**? = Y



Koala's **nose**? = Y



Koala's **ears**? = Y



Koala's **hands**? = Y



Koala's **legs**? = Y



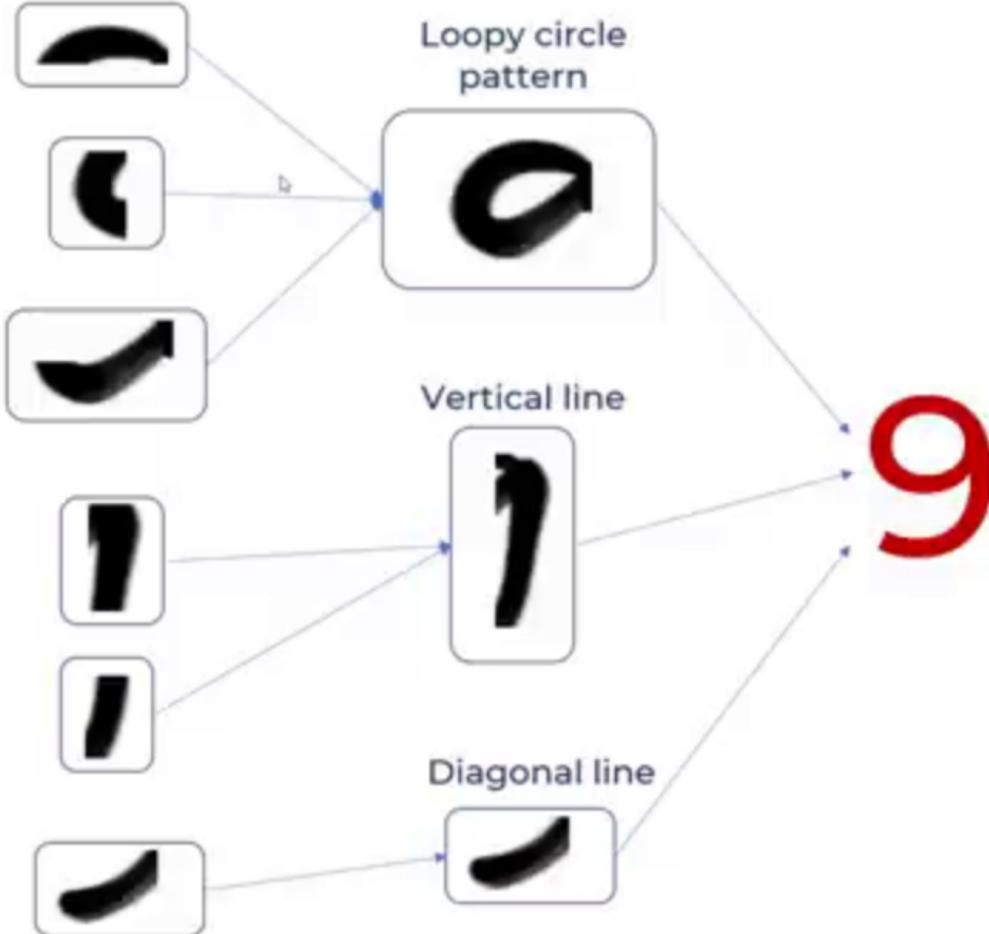
Koala's **head**? = Y

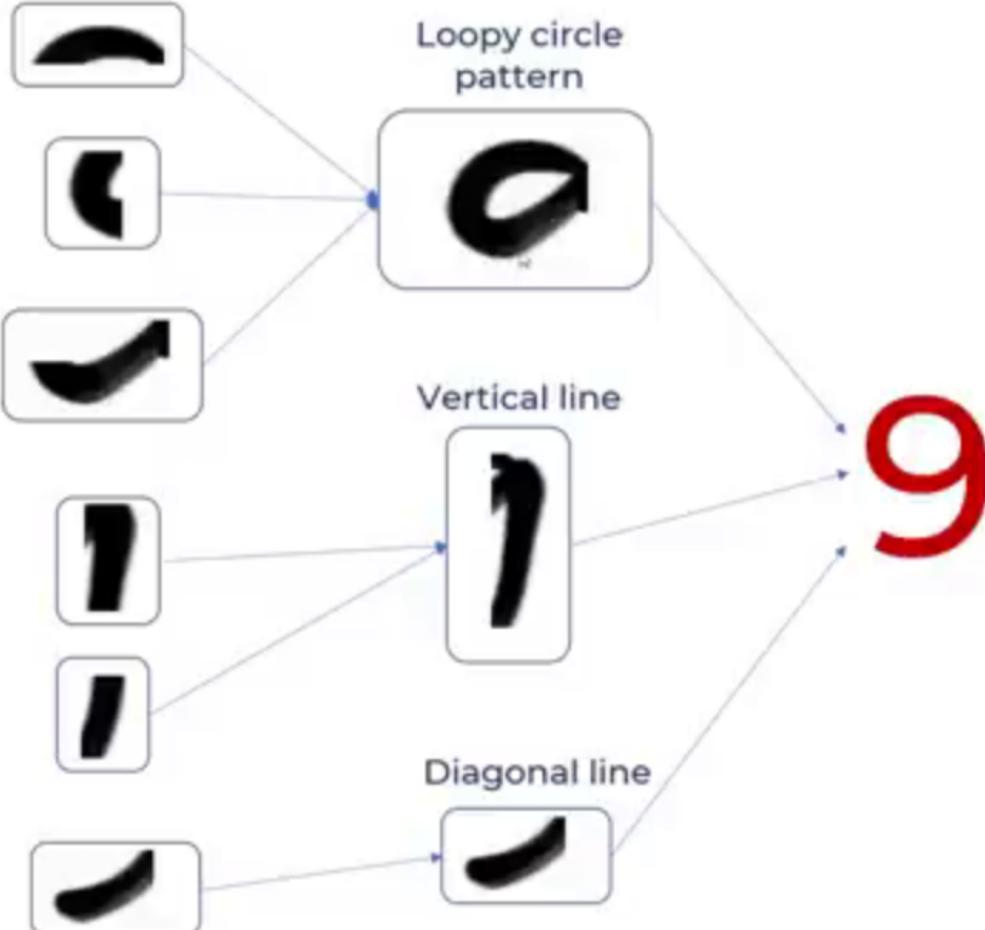


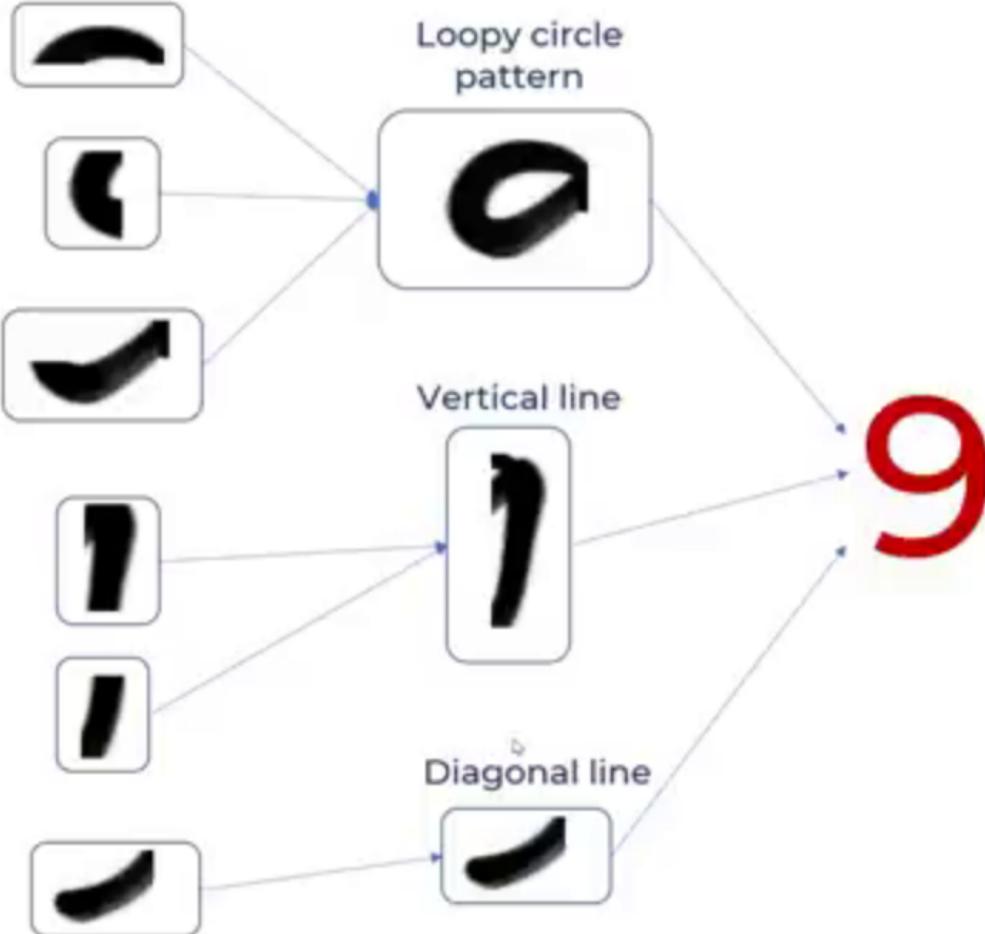
Koala's **body**? = Y

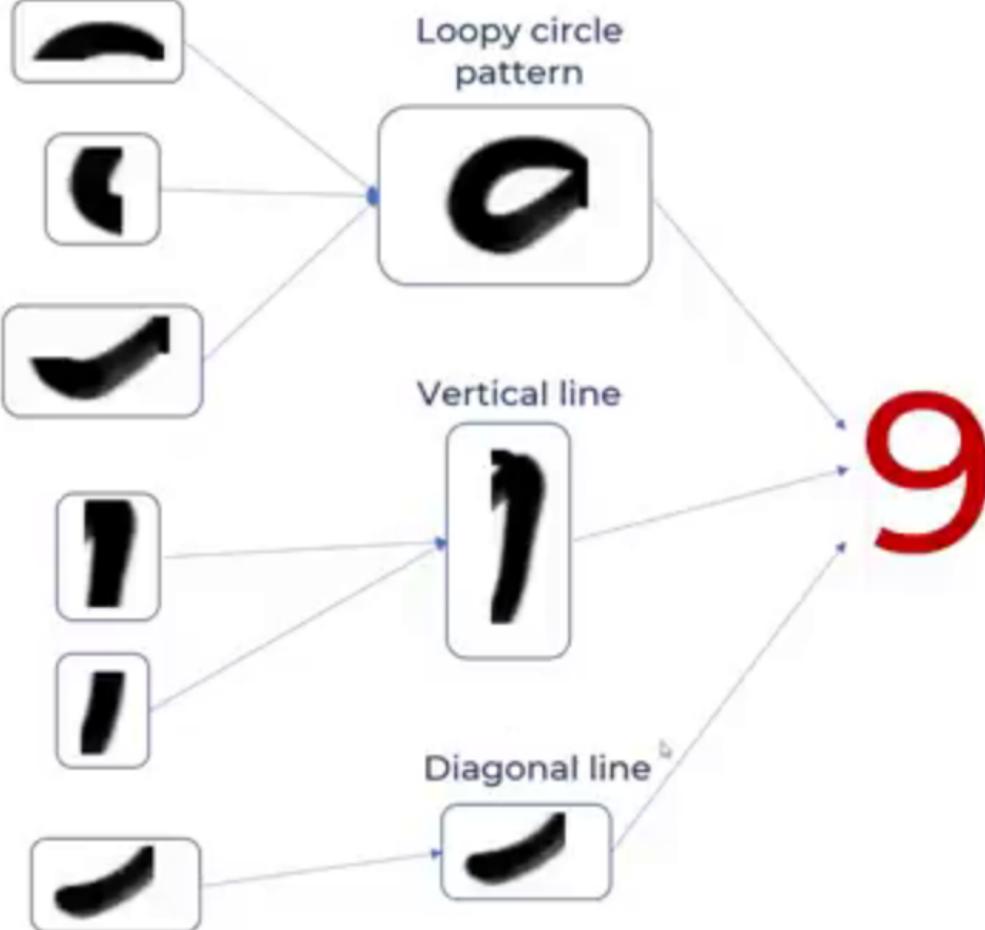


Is it **Koala**? = Y









-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Loopy pattern
filter

Vertical line
filter

Diagonal line
filter

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Loopy pattern
filter

Vertical line
filter

Diagonal line
filter

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

1	1	1
1	-1	1
1	1	1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1
1	-1	1
1	1	1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

1	1	1
1	-1	1
1	1	1

b

$$-1+1+1-1-1-1-1+1+1 = -1 \rightarrow -1/9 = -0.11$$

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11		

$$-1+1+1-1-1-1-1+1+1 = -1 \rightarrow -1/9 = -0.11$$

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11		

$$-1+1+1-1-1-1-1+1+1 = -1 \rightarrow -1/9 = -0.11$$

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11		

$$-1+1+1-1-1-1-1+1+1 = -1 \rightarrow -1/9 = -0.11$$

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11		

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11	1	

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

Feature Map

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

Feature Map

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

Feature Map

9

Loopy pattern
detector

$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline \end{array}$$

6

Loopy pattern
detector

$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline \end{array}$$

9

Loopy pattern
detector

$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & 1 & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

6

Loopy pattern
detector

$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array}$$

9

Loopy pattern
detector

$$9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & 1 \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

6

Loopy pattern
detector

$$6 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & 1 \\ \hline \end{array}$$

8

Loopy pattern
detector

$$8 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & 1 \\ \hline & & \\ \hline & & \\ \hline & 1 & \\ \hline \end{array}$$

96

Loopy pattern
detector

$$96 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & 1 \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & 1 \\ \hline \end{array}$$

9

Loopy pattern
detector

$$9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & 1 \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

6

Loopy pattern
detector

$$6 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & 1 \\ \hline \end{array}$$

8

Loopy pattern
detector

$$8 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & 1 \\ \hline & & \\ \hline & & \\ \hline & & 1 \\ \hline \end{array}$$

96

Loopy pattern
detector

$$96 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|} \hline & & 1 \\ \hline & & \\ \hline & & \\ \hline & & 1 \\ \hline \end{array}$$



$$\text{koala image} * \text{eye detector} = \boxed{\text{two green squares}}$$



$$\text{koala image} * \begin{matrix} \text{eye} \\ \text{detector} \end{matrix} = \boxed{\text{green squares at eye locations}}$$



$$\text{koala image} * \begin{matrix} \text{eye} \\ \text{detector} \end{matrix} = \boxed{\text{green squares at eye locations}}$$



$$\text{koala image} * \begin{matrix} \text{eye} \\ \text{detector} \\ \text{grid icon} \end{matrix} = \boxed{\text{green squares at eye locations}}$$

Location invariant: It can detect eyes in any location of the image



$$\text{koala image} * \begin{matrix} \text{eye} \\ \text{detector} \\ \text{grid icon} \end{matrix} = \boxed{\text{green squares at eye locations}}$$

Q



$$\text{koala} * \begin{matrix} \text{eye} \\ \text{detector} \\ \left[\begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} \right] \end{matrix} = \boxed{\begin{matrix} & 1 \\ & 1 \end{matrix}}$$

Location invariant: It can detect eyes in any location of the image



$$\text{koala} * \begin{matrix} \text{eye} \\ \text{detector} \\ \left[\begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} \right] \end{matrix} = \boxed{\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}}$$

b)



* hands
detector



Loopy pattern detector

$$g * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline & & & & & \\ \hline & & 1 & & & \\ \hline & & & & & \\ \hline & & & & & \\ \hline & & & & & \\ \hline \end{array}$$

Vertical line detector

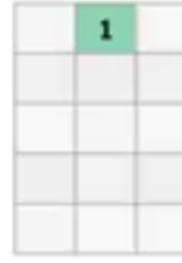
$$g * \begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} = \begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline & & 1 & & & \\ \hline & & & & & \\ \hline \end{array}$$

Diagonal line detector

$$g * \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} = \begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline & & 1 & & & \\ \hline & & & & & \\ \hline \end{array}$$

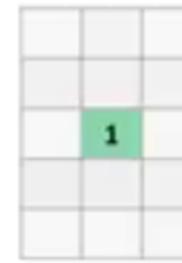
Loopy pattern detector



$$g * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$


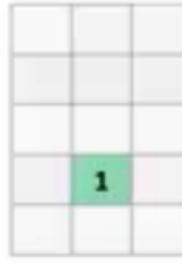
Vertical line detector



$$g * \begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} =$$


Diagonal line detector



$$g * \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} =$$


Loopy pattern detector

$$g * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & & & & \\ \hline & & 1 & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array}$$

b)

Vertical line detector

$$g * \begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} = \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline \end{array}$$

Feature Maps

Filters

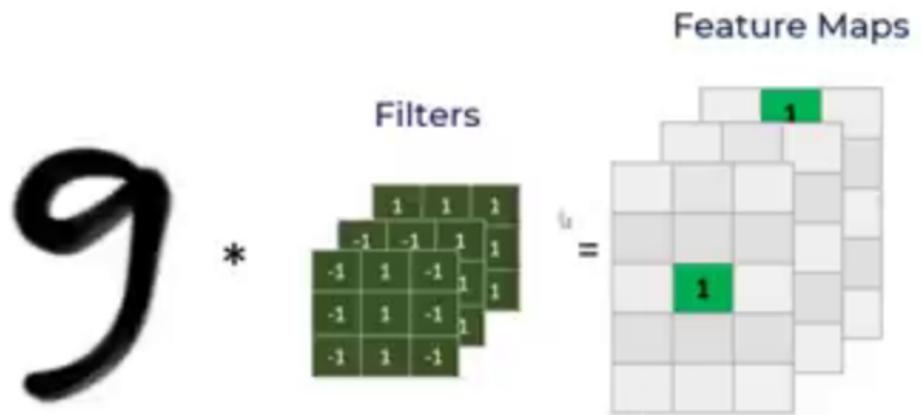
$\mathbf{g} * \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & & \end{bmatrix}$

The diagram illustrates a convolution operation. On the left, a large black handwritten digit 'g' is shown. To its right is a multiplication symbol (*). Next to the multiplication symbol is a 3x3 matrix representing a filter, with values: 1, 1, 1; -1, -1, 1; -1, 1, -1; -1, 1, -1. To the right of the filter is an equals sign (=). To the right of the equals sign is a 3x3 grid representing the resulting feature map. The top-right cell of the feature map contains the value '1', indicating the result of the convolution step.

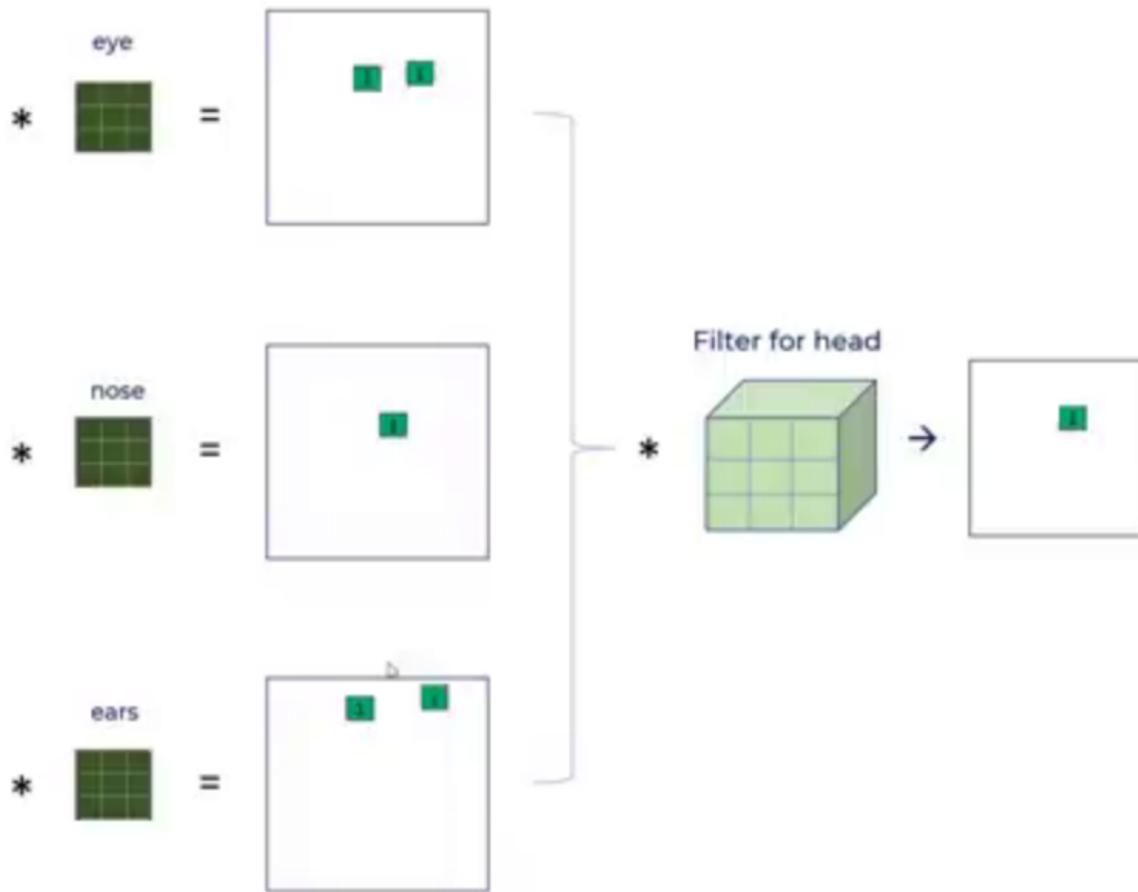
Feature Maps

Filters

$g * \begin{matrix} 1 & 1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{matrix} = \begin{matrix} 1 & & \\ & 1 & \\ & & \end{matrix}$



The diagram illustrates a convolution operation. On the left, a handwritten digit 'g' is multiplied by a 3x3 kernel (filter). The result is a 3x3 feature map. The values in the feature map are highlighted in green at positions (1,1) and (2,2), indicating the presence of the kernel's receptive field at those locations.





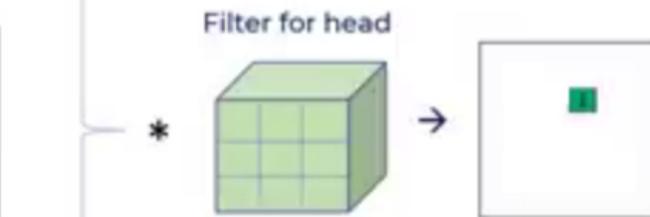
$$\text{eye} * \begin{array}{|c|c|}\hline \text{green} & \text{green} \\ \hline \end{array} = \boxed{\text{green green}}$$

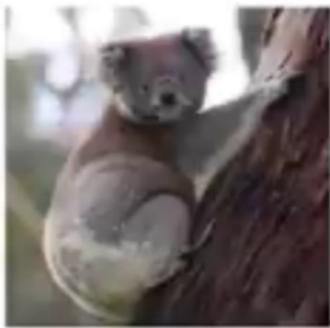


$$\text{nose} * \begin{array}{|c|c|}\hline \text{green} & \text{green} \\ \hline \end{array} = \boxed{\text{green}}$$



$$\text{ears} * \begin{array}{|c|c|}\hline \text{green} & \text{green} \\ \hline \end{array} = \boxed{\text{green green}}$$





$$* \quad \text{eye} \quad = \quad \begin{array}{|c|} \hline \text{green square} & \text{green square} \\ \hline \end{array}$$

$$* \quad \text{nose} \quad = \quad \begin{array}{|c|} \hline \text{green square} \\ \hline \end{array}$$

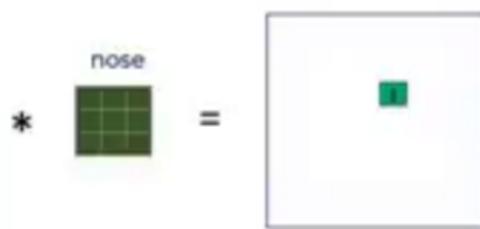
$$* \quad \text{ears} \quad = \quad \begin{array}{|c|} \hline \text{green square} & \text{green square} \\ \hline \end{array}$$

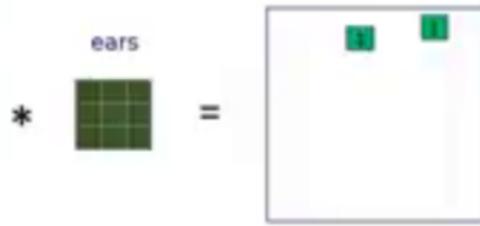
Filter for head

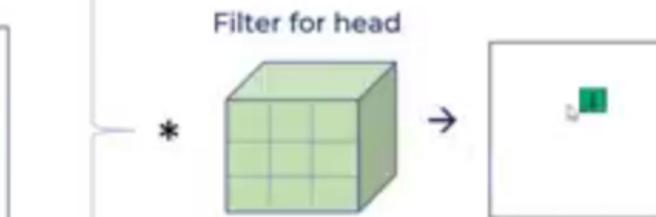
$$* \quad \begin{array}{|c|c|c|} \hline \text{green cube} & & \\ \hline & \text{green cube} & \\ \hline & & \text{green cube} \\ \hline \end{array} \rightarrow \quad \begin{array}{|c|} \hline \text{green square} \\ \hline \end{array}$$

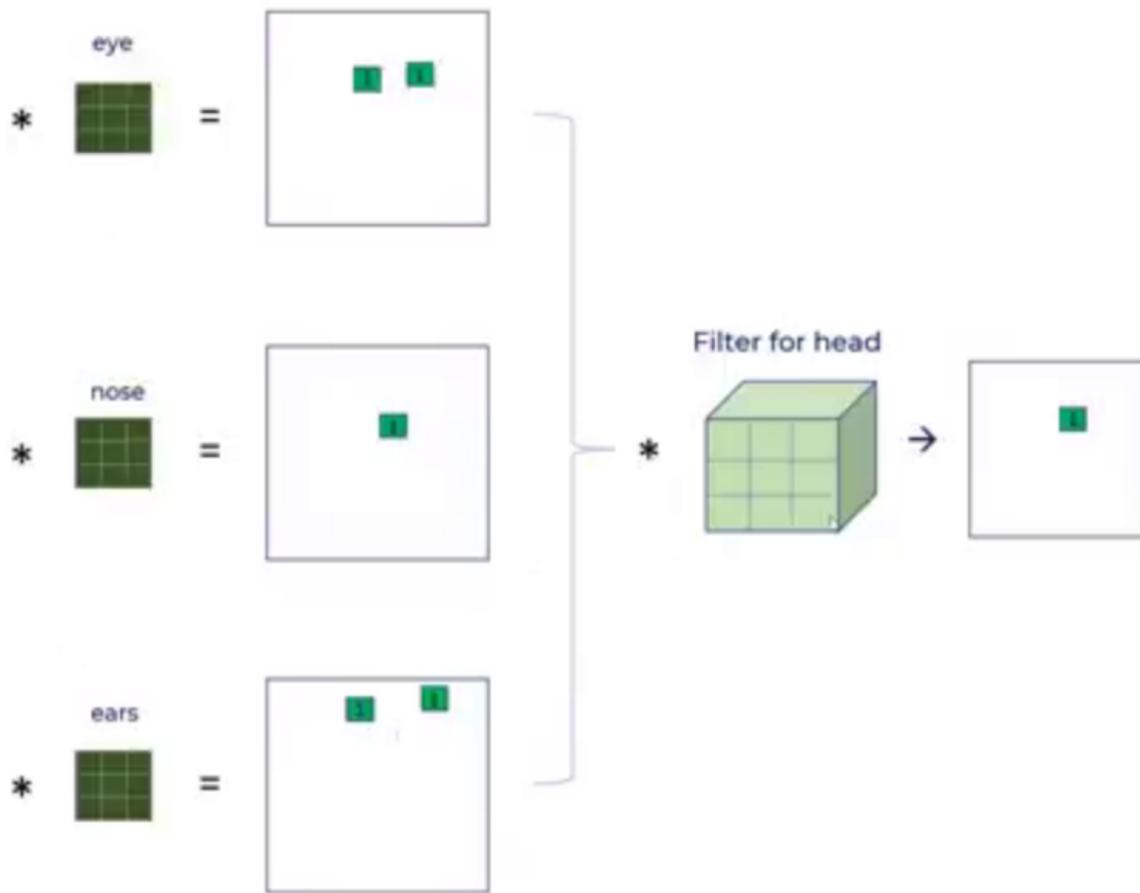


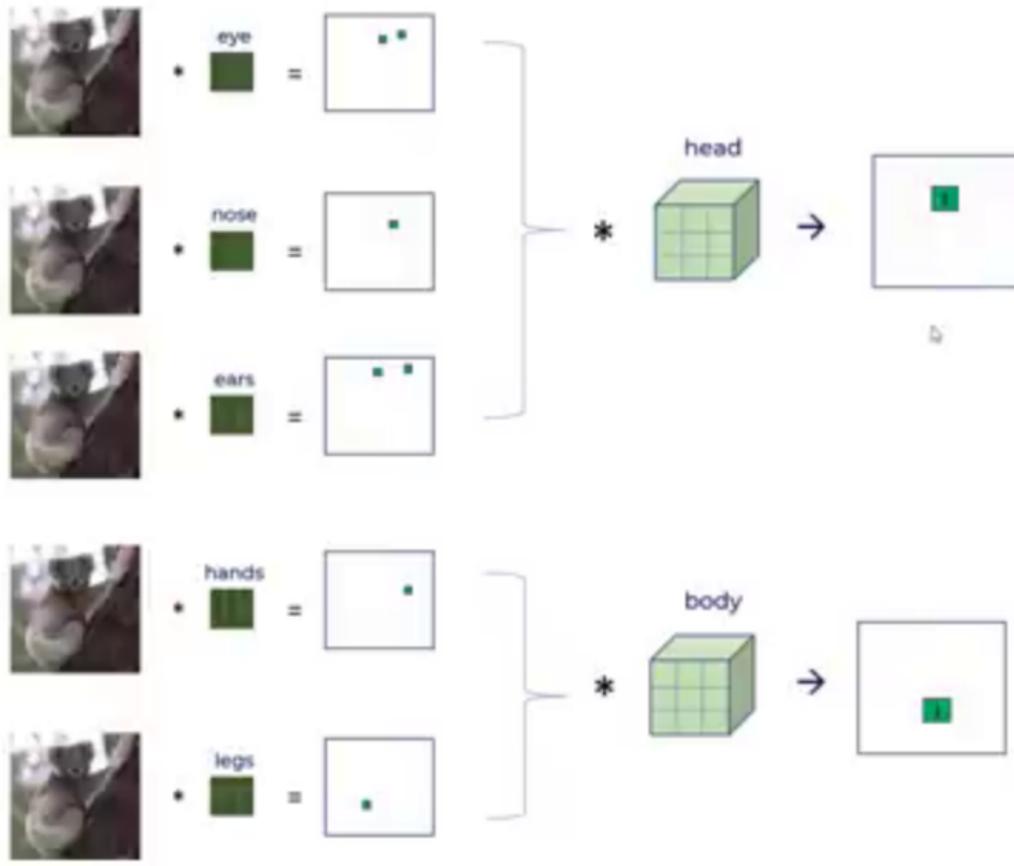
$$* \text{ eye } =$$
A diagram showing a convolution operation. On the left is a 3x3 input grid with green squares at positions (1,1), (1,2), and (2,1). An asterisk (*) indicates the convolution operator followed by an equals sign. To the right is a 2x2 output grid with green squares at positions (1,1) and (2,1).

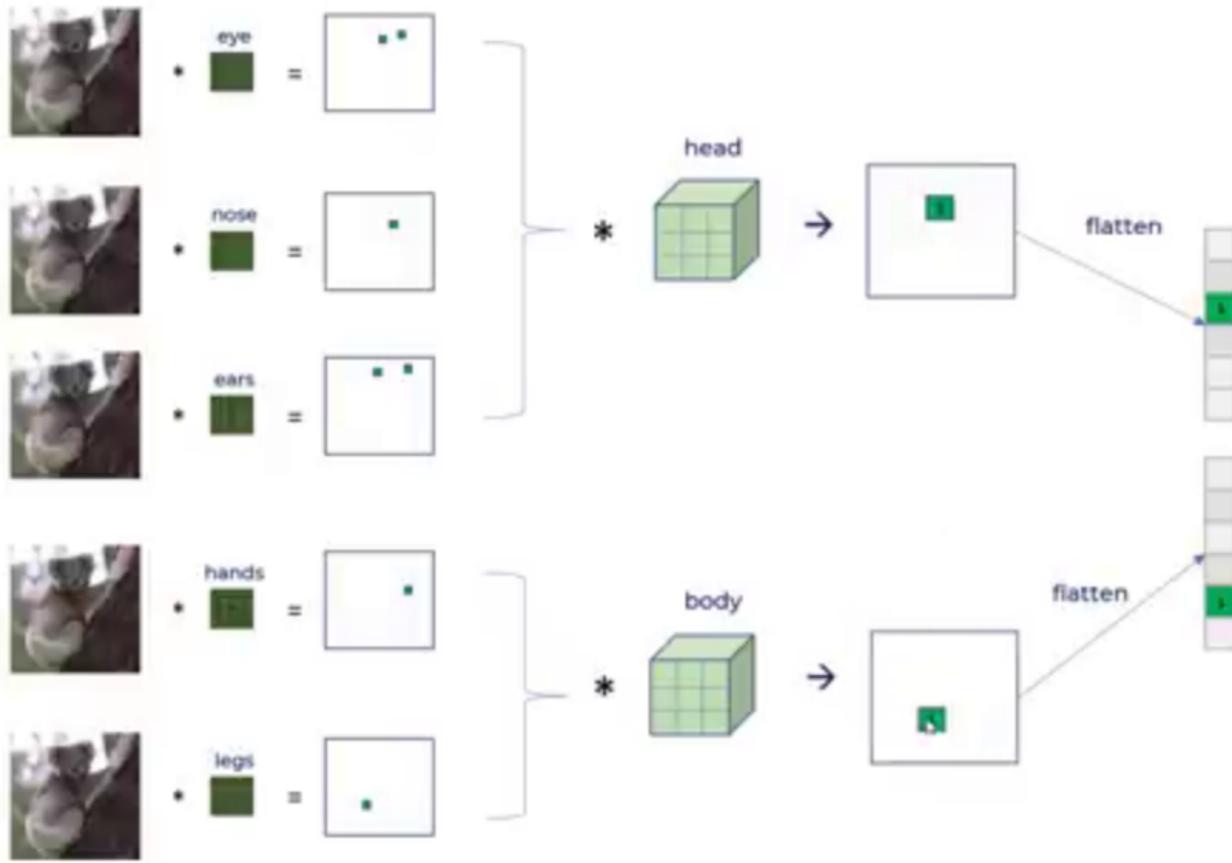
$$* \text{ nose } =$$
A diagram showing a convolution operation. On the left is a 3x3 input grid with a single green square at position (2,2). An asterisk (*) indicates the convolution operator followed by an equals sign. To the right is a 2x2 output grid with a single green square at position (1,1).

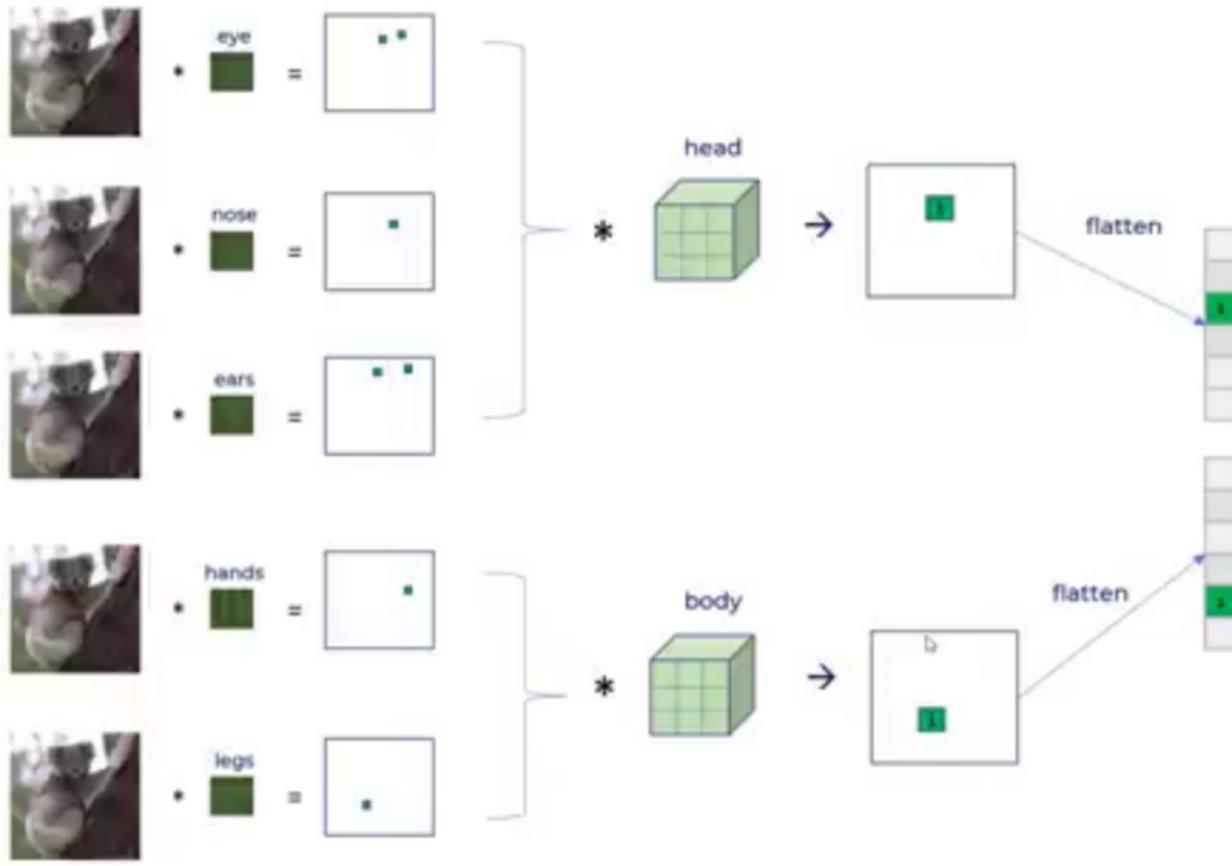
$$* \text{ ears } =$$
A diagram showing a convolution operation. On the left is a 3x3 input grid with green squares at positions (1,1), (1,2), and (2,1). An asterisk (*) indicates the convolution operator followed by an equals sign. To the right is a 2x2 output grid with green squares at positions (1,1) and (2,1).

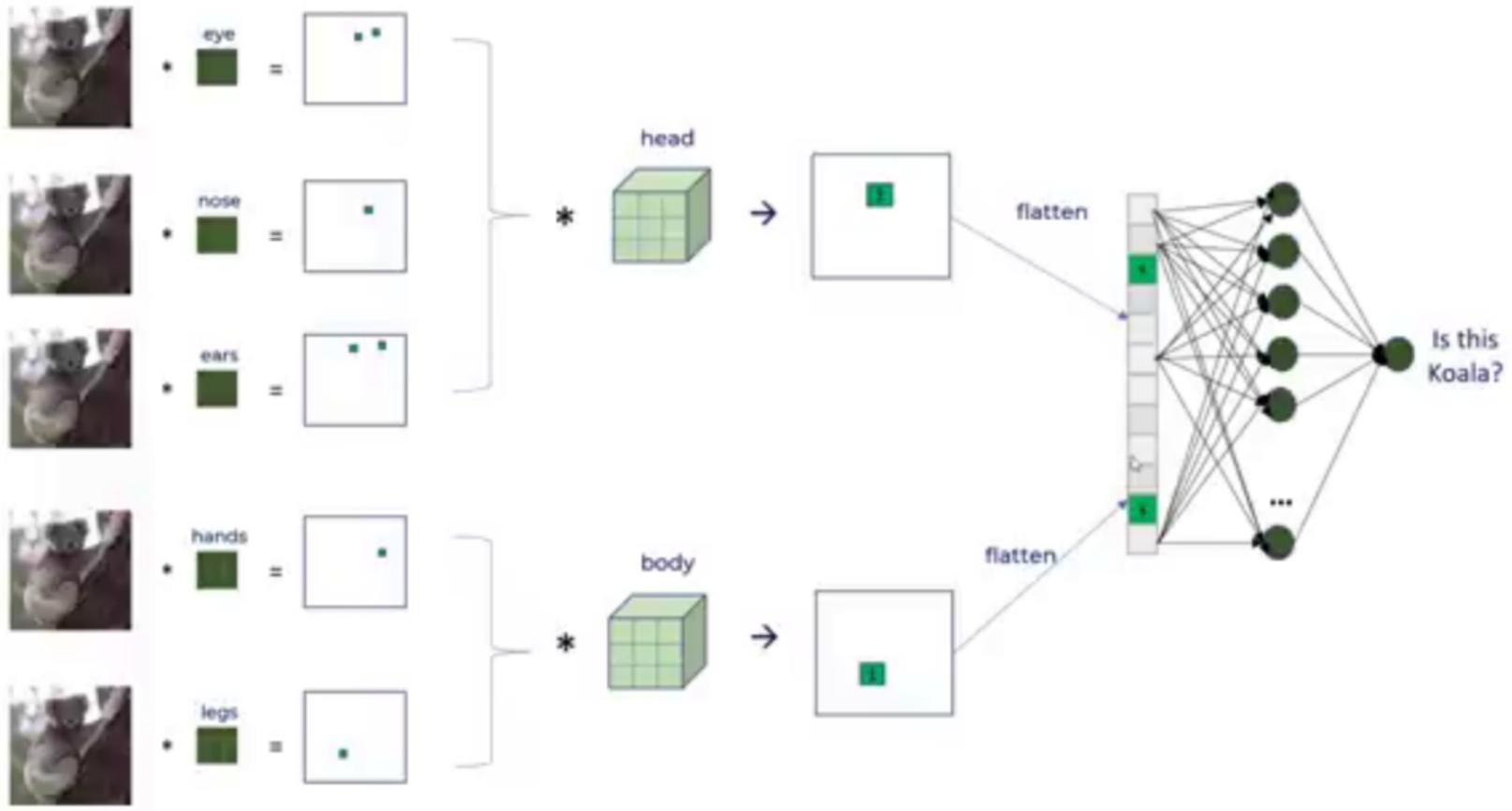


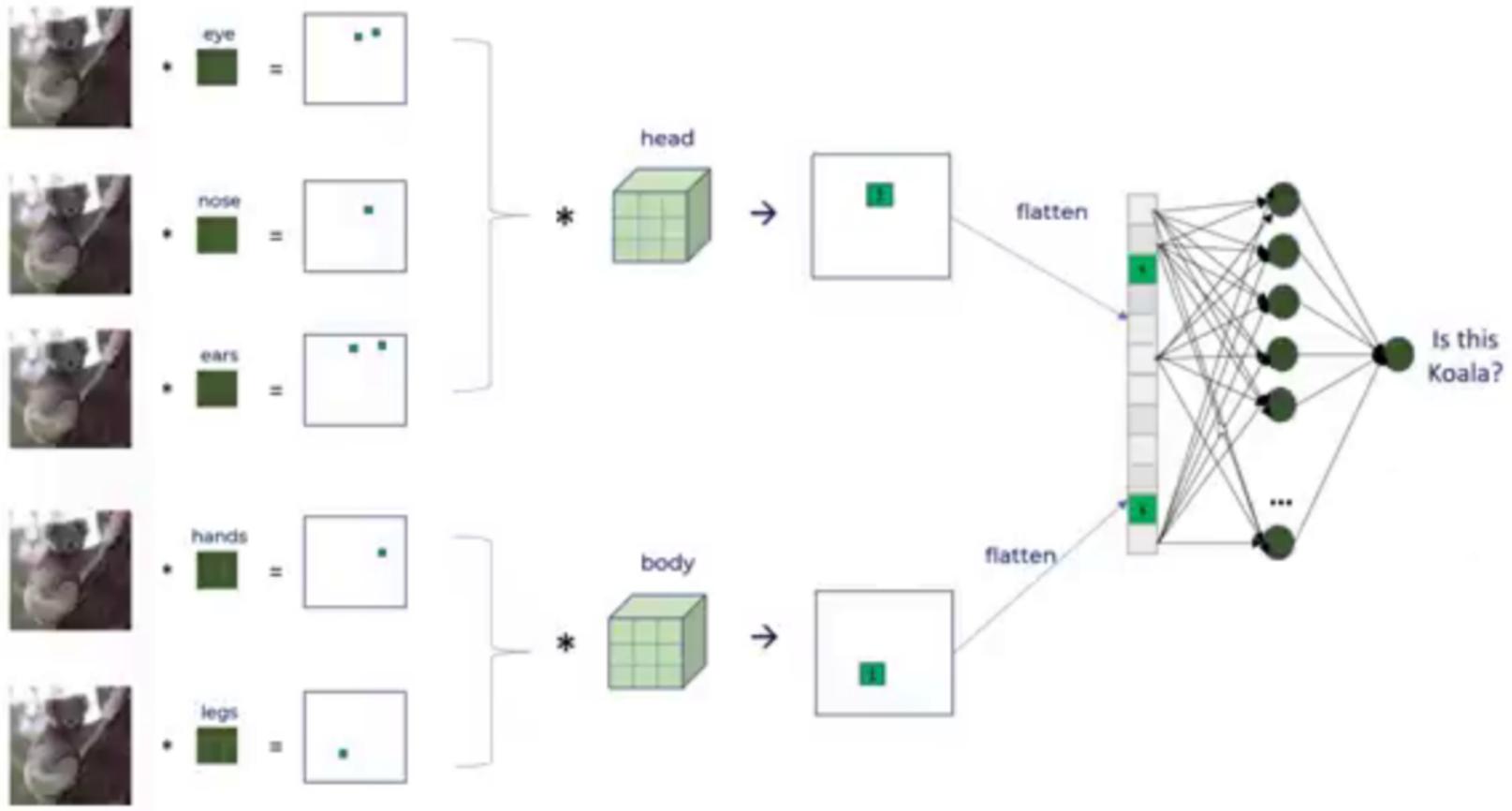


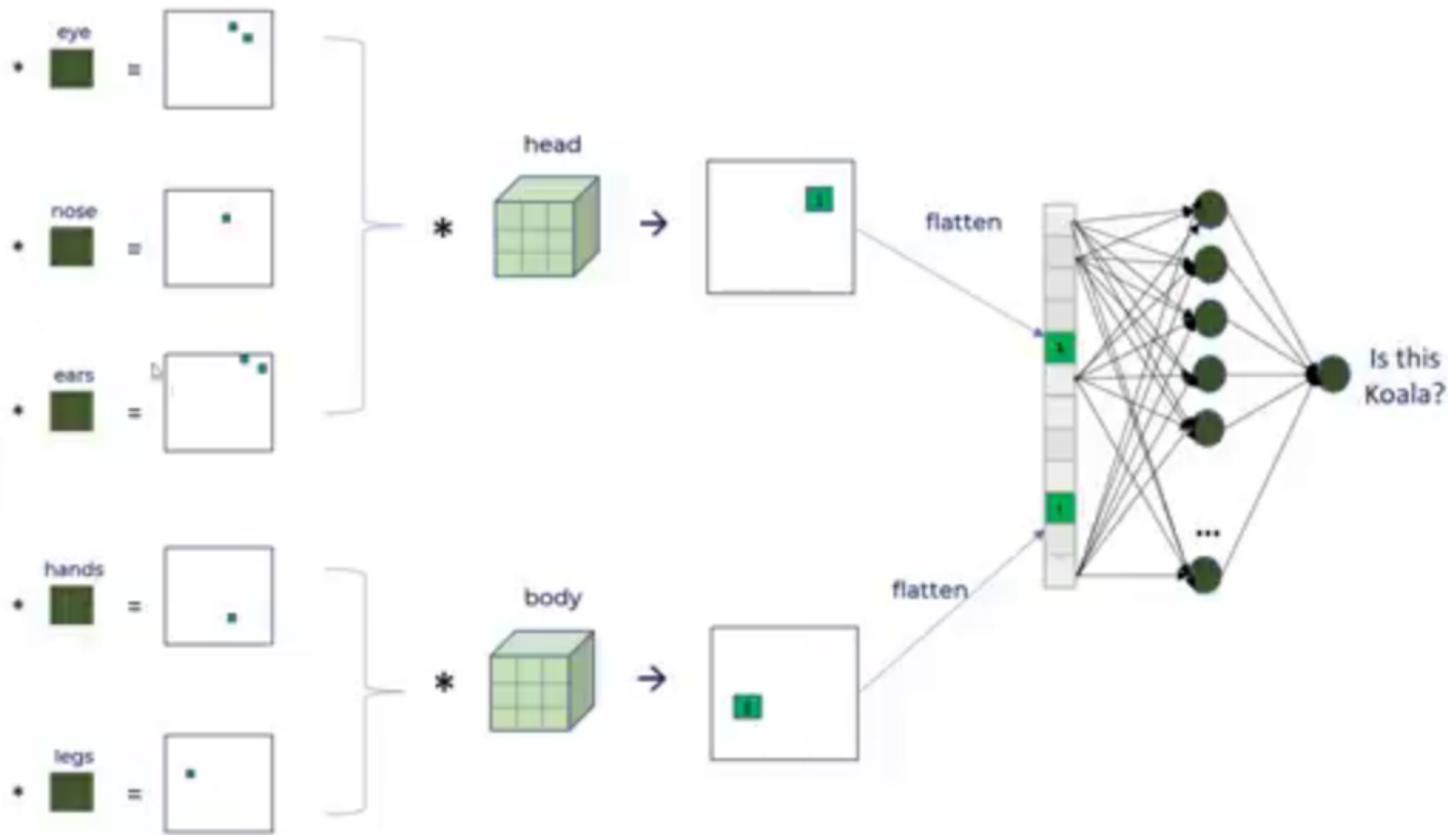


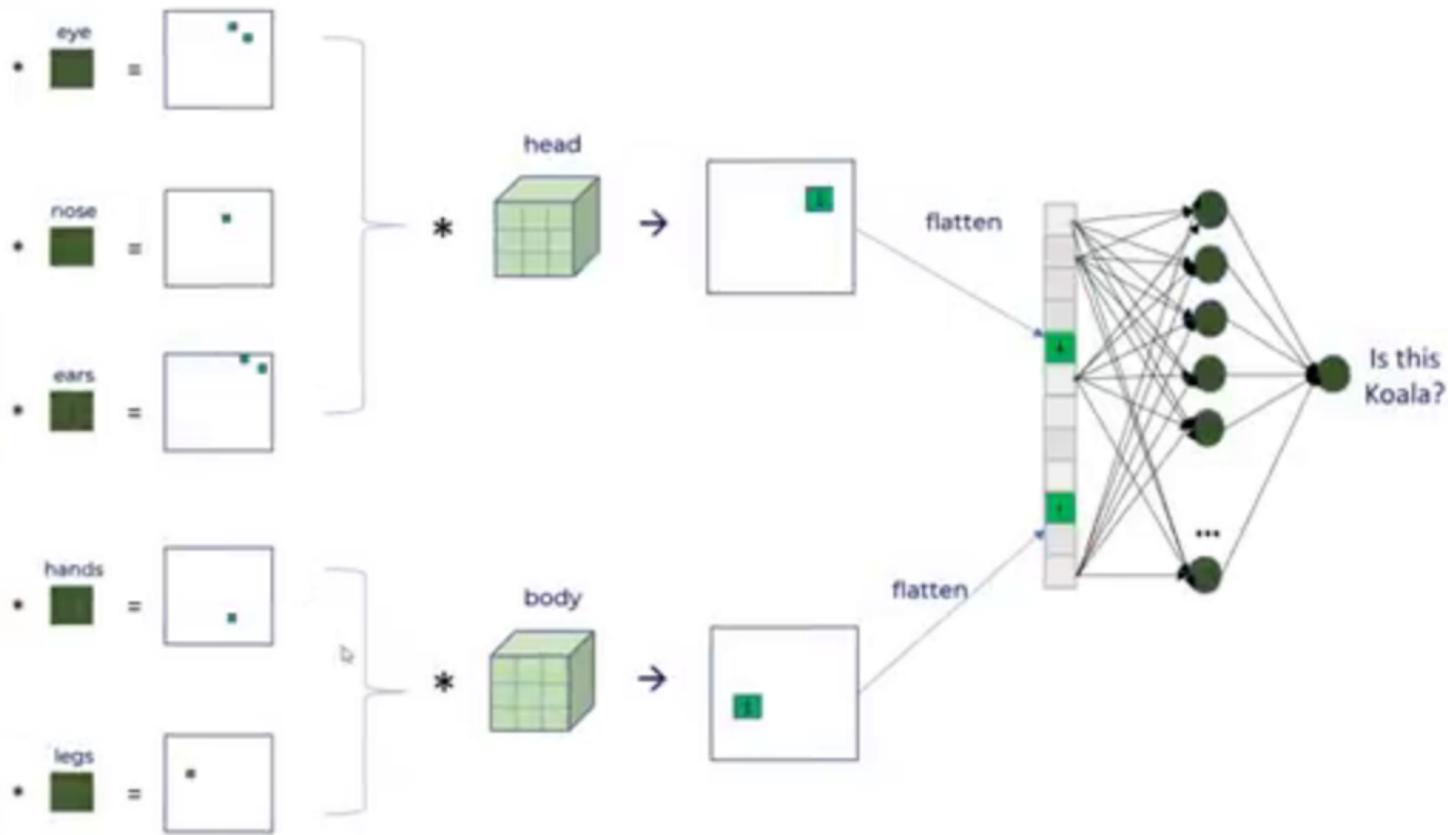


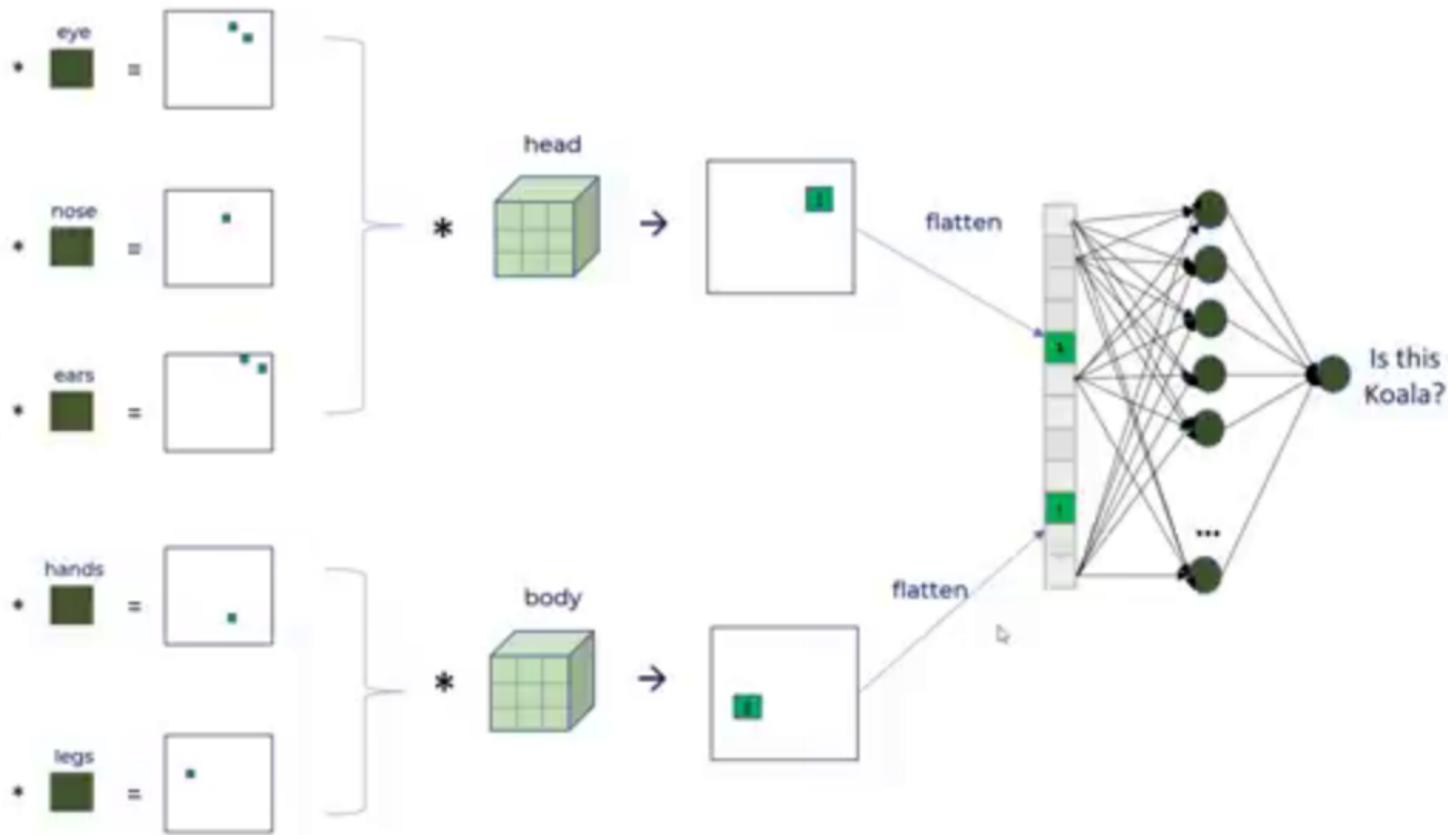


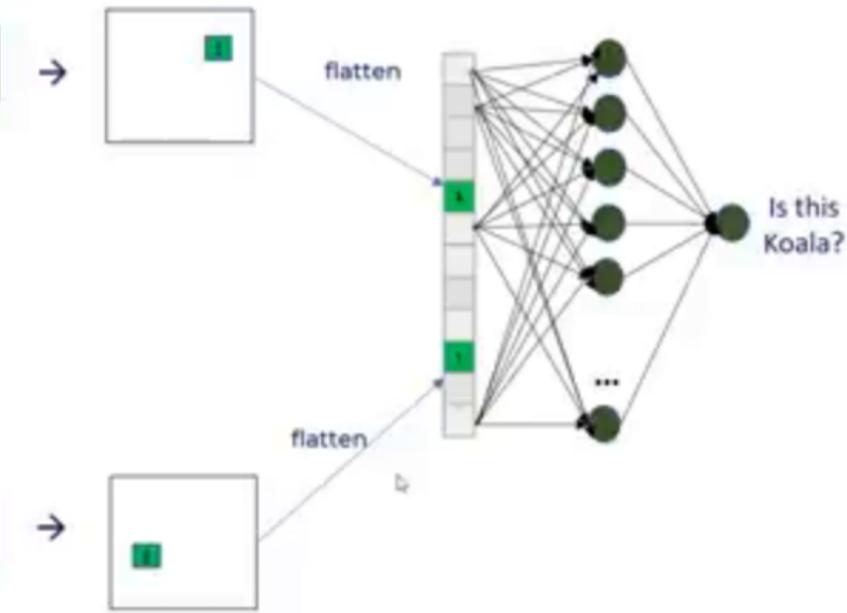
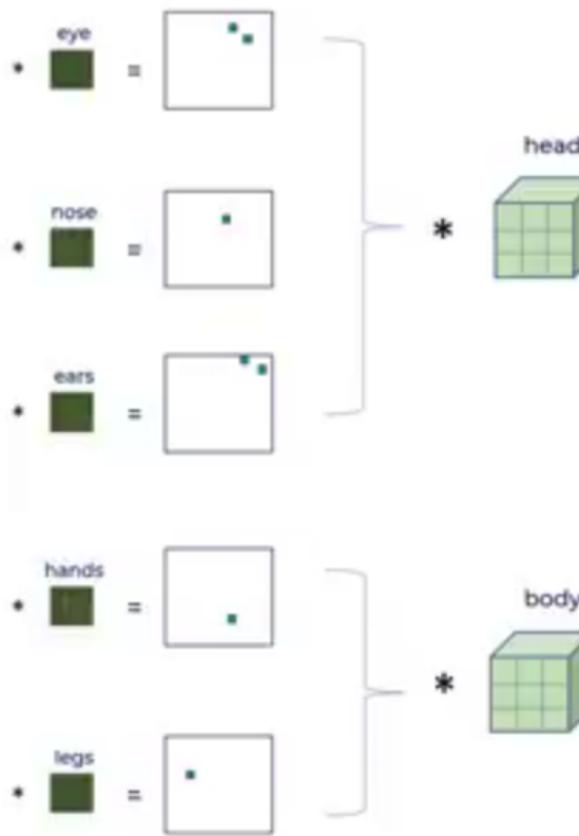


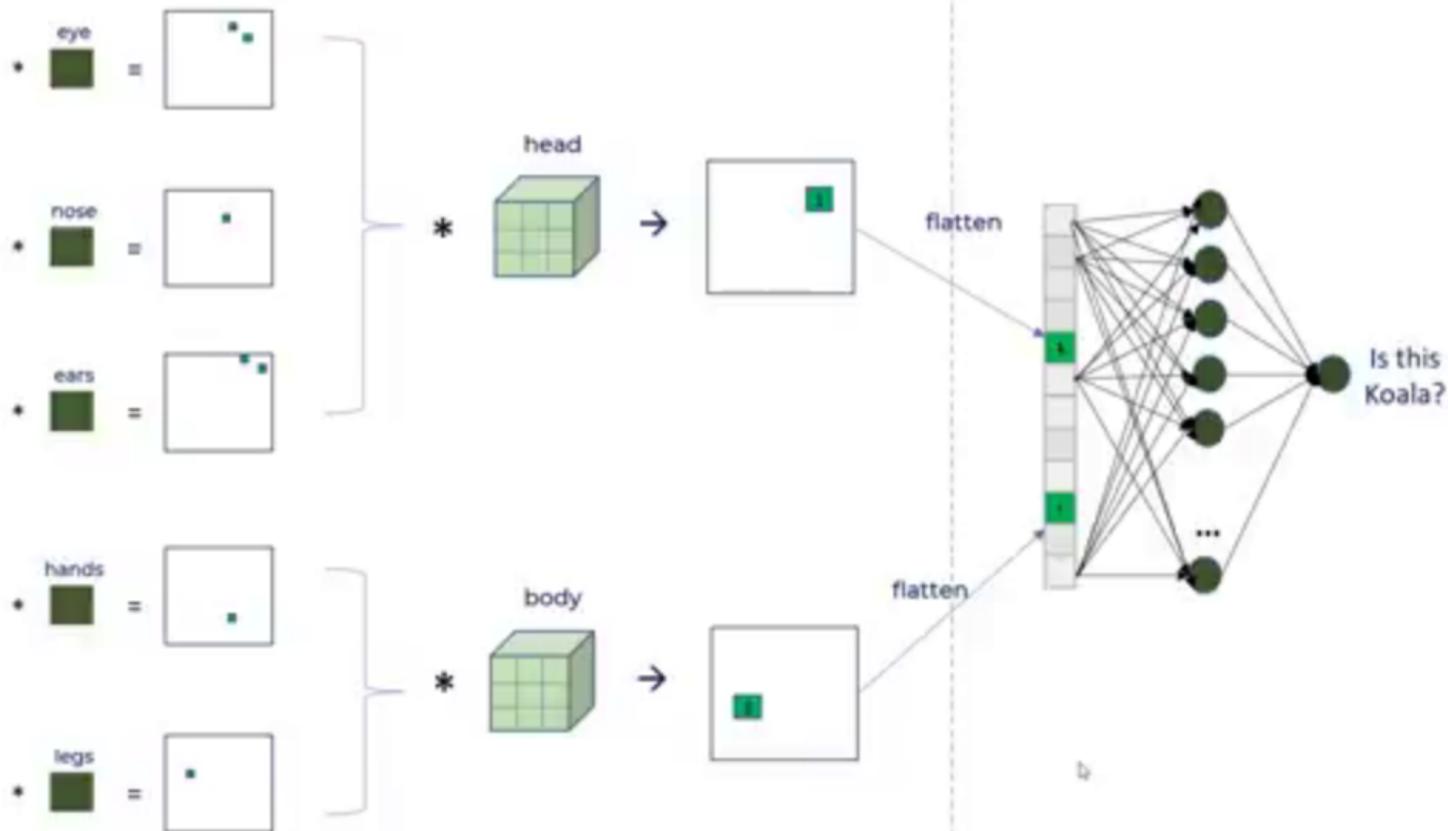


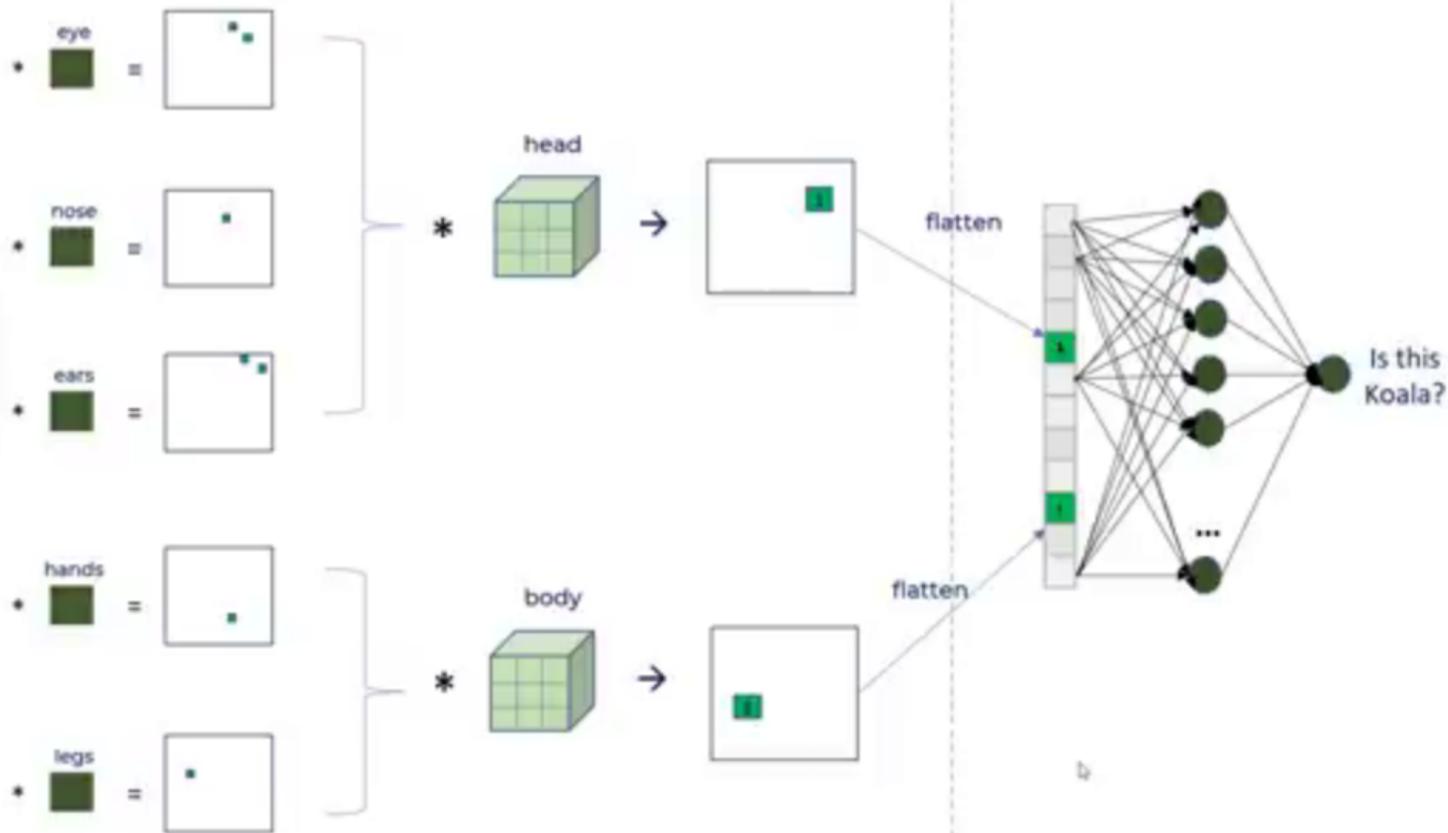












-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

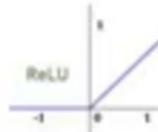
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

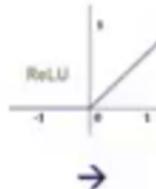
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

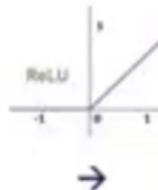
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

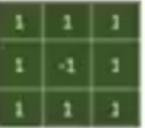


0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

b)

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Loopy pattern
 filter

\ast

→

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

ReLU helps with making
the model nonlinear

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

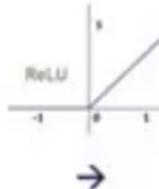
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



→

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

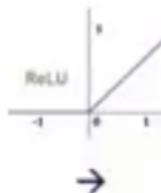
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

b₂

We didn't
address the issue
of too much
computation



Image credit: <https://giphy.com/> (@rabbids)



Image size = 1920 x 1080 X 3

First layer neurons = 1920 x 1080 X 3 ~ 6 million

Hidden layer neurons = Let's say you keep it ~ 4 million

Weights between input and hidden layer = $6 \text{ mil} * 4 \text{ mil}$
= 24 million



$$\text{eye} * \begin{matrix} \blacksquare \\ \blacksquare \end{matrix} = \boxed{\text{eye}} \quad 1920 \times 1080$$



$$\text{nose} * \begin{matrix} \blacksquare \\ \blacksquare \end{matrix} = \boxed{\text{nose}} \quad 1920 \times 1080$$



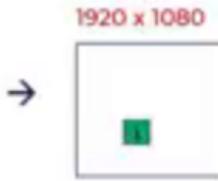
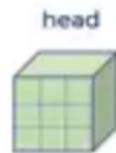
$$\text{ears} * \begin{matrix} \blacksquare \\ \blacksquare \end{matrix} = \boxed{\text{ears}} \quad 1920 \times 1080$$



$$\text{hands} * \begin{matrix} \blacksquare \\ \blacksquare \end{matrix} = \boxed{\text{hands}} \quad 1920 \times 1080$$

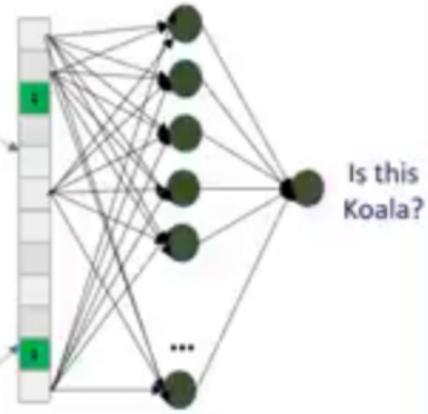


$$\text{legs} * \begin{matrix} \blacksquare \\ \blacksquare \end{matrix} = \boxed{\text{legs}} \quad 1920 \times 1080$$



→

flatten



Is this
Koala?

Pooling layer is used to
reduce the size

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

8	9
3	2

4

2 by 2 filter with stride = 2

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

8	9
3	2

2 by 2 filter with stride = 2

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

8	9
3	2

2 by 2 filter with stride = 2

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

4

8	9
3	2

2 by 2 filter with stride = 2

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

8	9
3	2

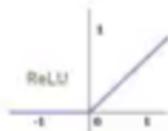
2 by 2 filter with stride = 2

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

* Loopy pattern filter →

1	1	1
1	-1	1
1	1	1

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Max pooling →

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

8	9
3	2

2 by 2 filter with stride = 2



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

1	1

2 by 2 filter with stride = 1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1

* →

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



→

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Max
pooling

→

1	1
0.33	0.33
0.33	0.33
0	0

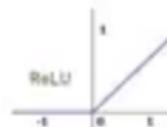
Shifted 9 at
different position

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

Loopy pattern
filter

$$* \quad \begin{matrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{matrix} \quad \rightarrow$$

1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55



1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

Max
pooling
 \rightarrow

1	0.33
0.33	0.33
0.33	0
0	0

There is average pooling also...

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

4	4.5
2	0.75

b

Shifted 9 at
different position

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

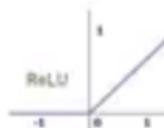
Loopy pattern
filter

*

1	1	1
1	-1	1
1	1	1



1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55



1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

Max
pooling



1	0.33
0.33	0.33
0.33	0
0	0

There is average pooling also...

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

4	4.5
2	0.75

There is average pooling also...

5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

4	4.5
2	0.75

Benefits of pooling

Reduces dimensions & computation

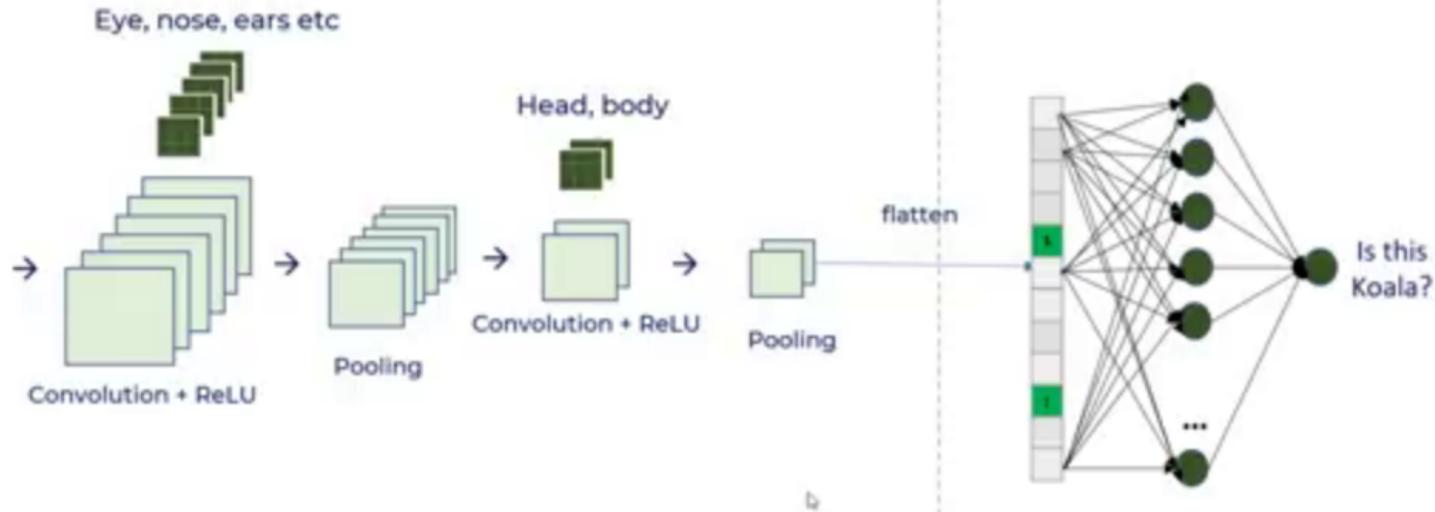
Reduce overfitting as there are less parameters

Benefits of pooling

Reduces dimensions & computation

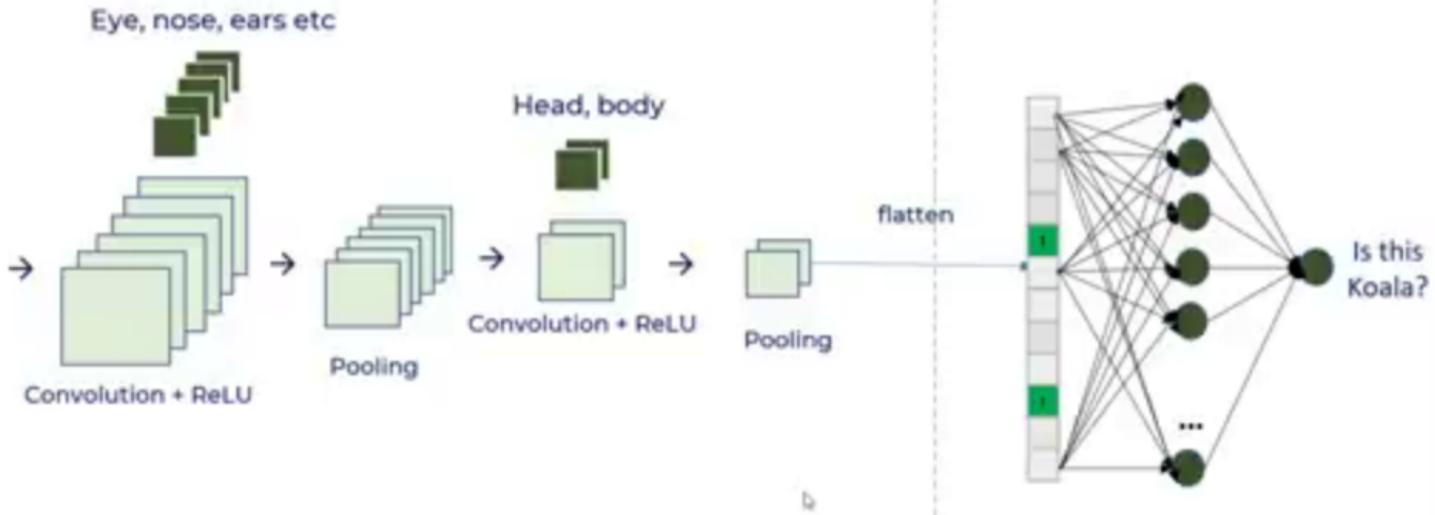
Reduce overfitting as there are less parameters

Model is tolerant towards variations, distortions



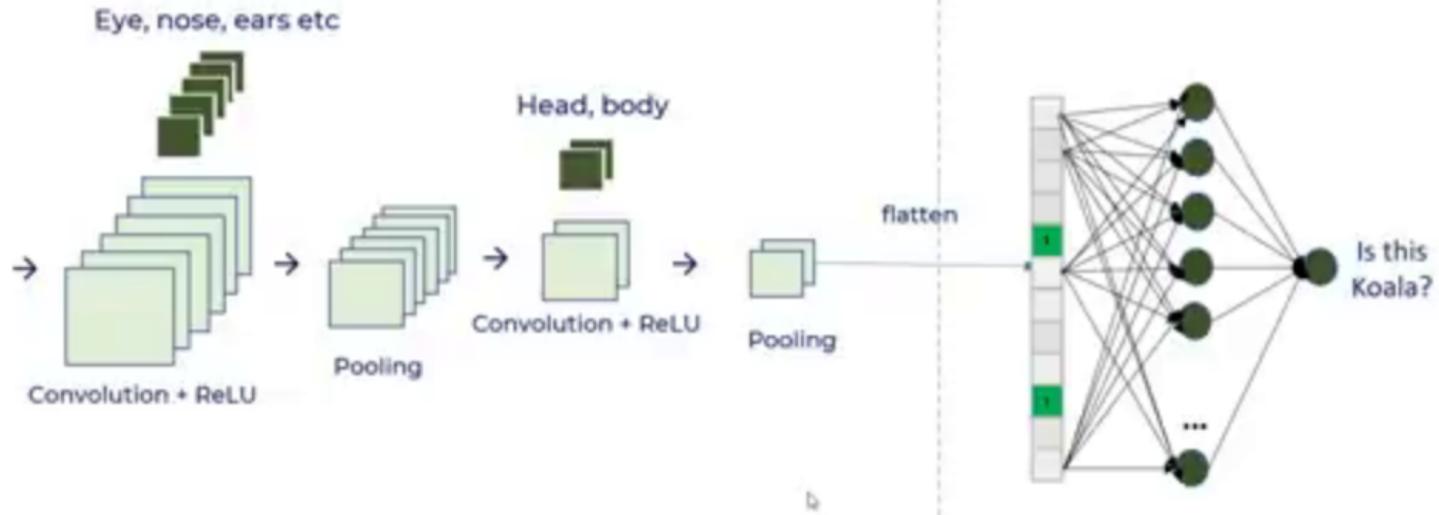
Feature Extraction

Classification



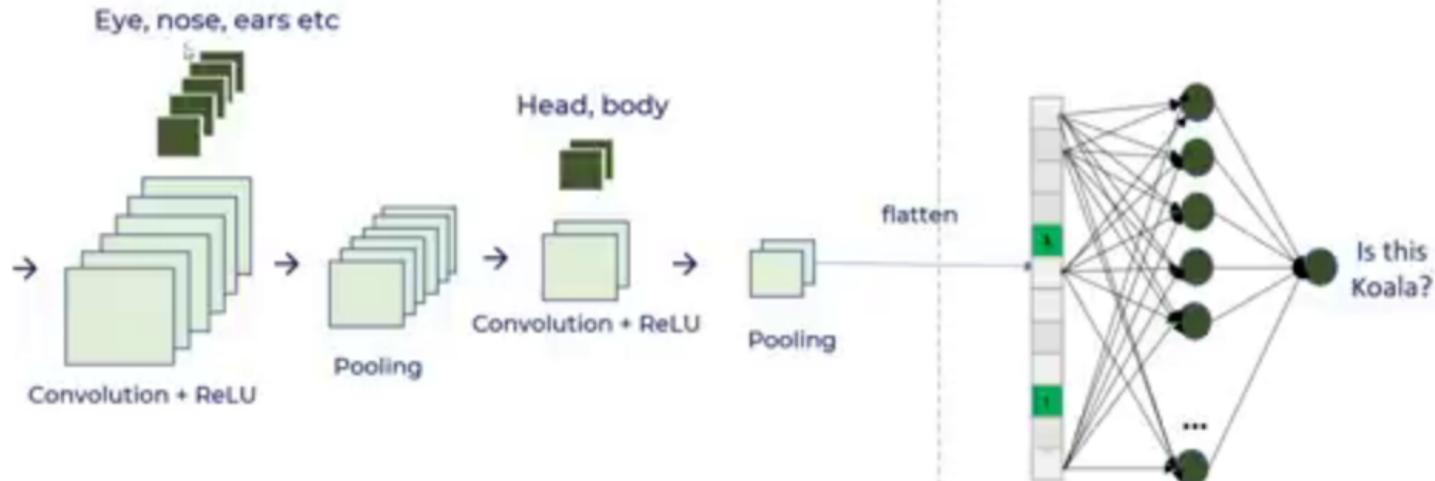
Feature Extraction

Classification



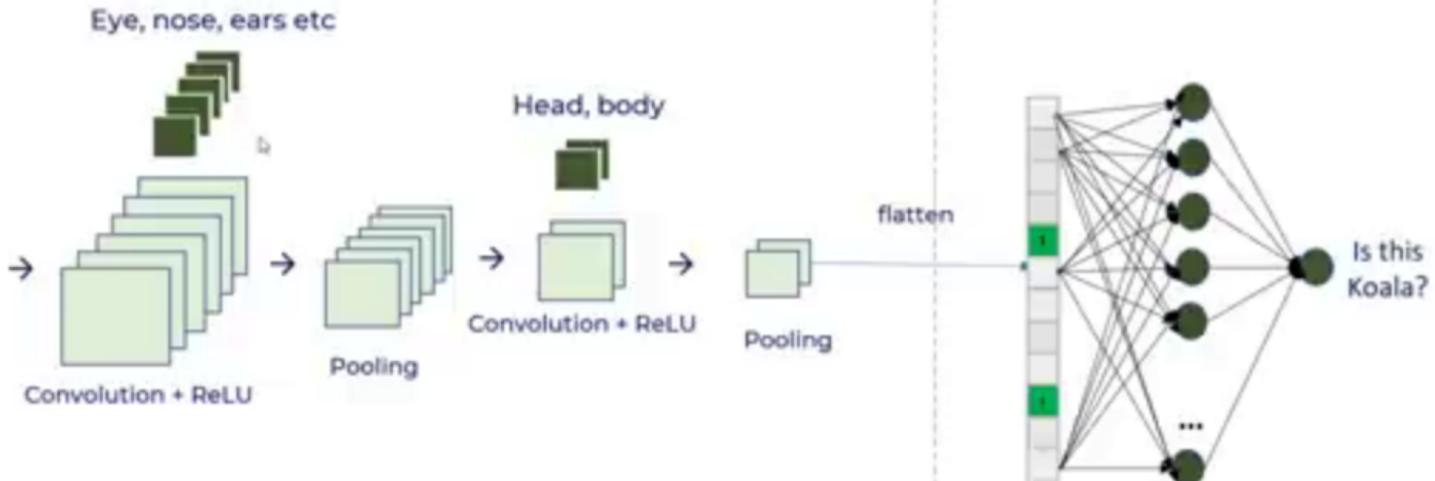
Feature Extraction

Classification



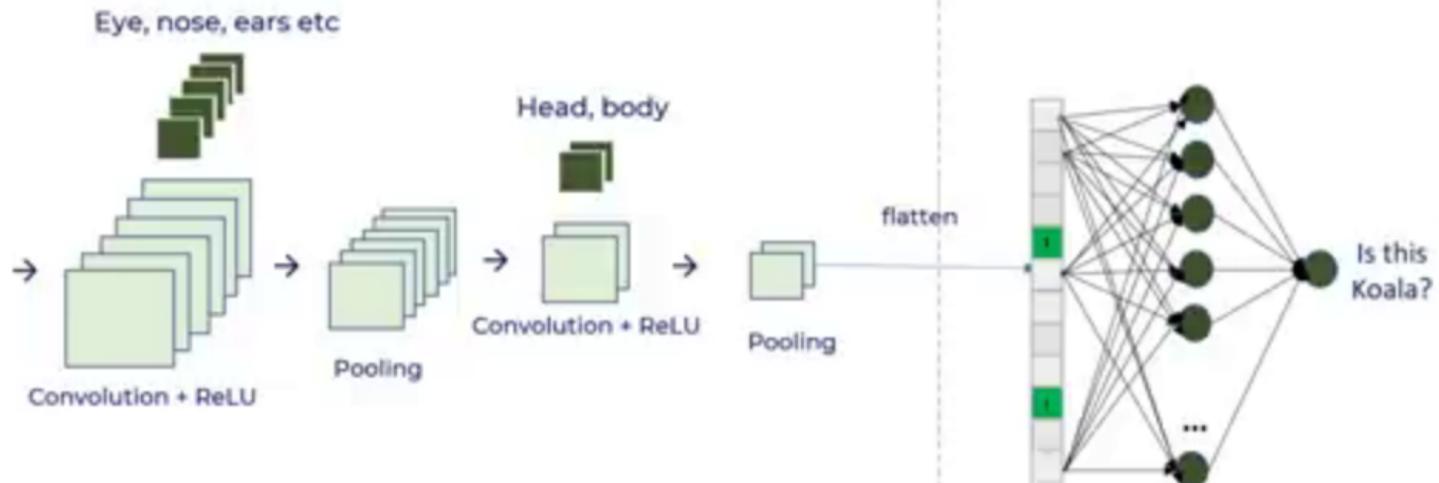
Feature Extraction

Classification



Feature Extraction

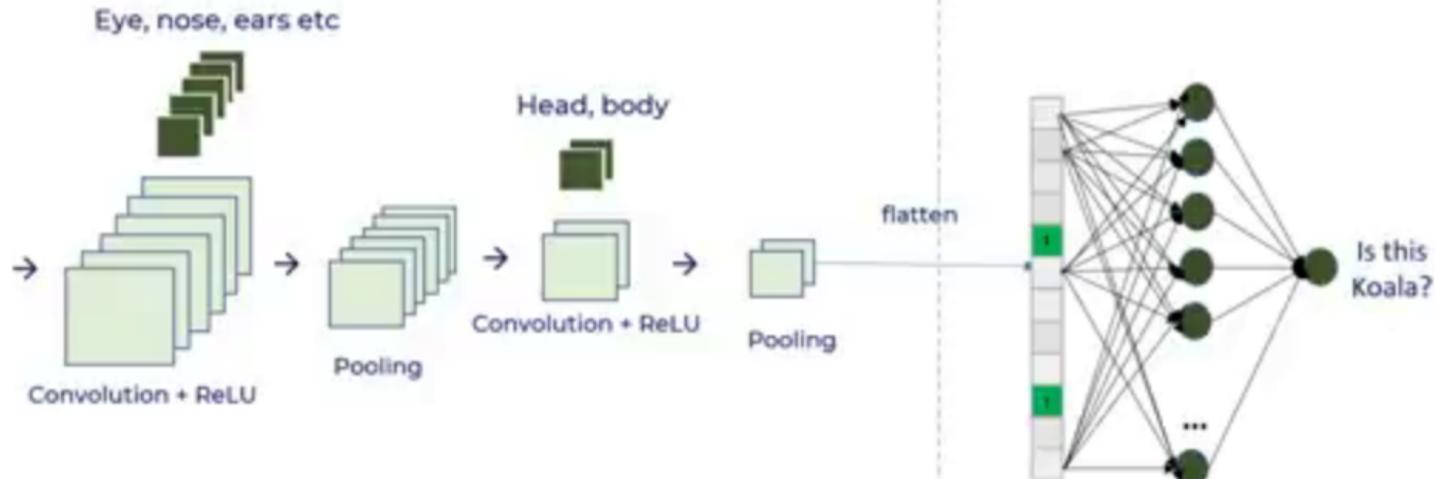
Classification



b

Feature Extraction

Classification



Feature Extraction

Classification

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

Convolution

ReLU

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

- Introduces nonlinearity
- Speeds up training, faster to compute

Convolution

ReLU

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

- Introduces nonlinearity
- Speeds up training, faster to compute

Convolution

- Connections sparsity reduces overfitting
- Conv + Pooling gives location invariant feature detection
- Parameter sharing

ReLU

- Introduces nonlinearity
- Speeds up training, faster to compute

Pooling

- Reduces dimensions and computation
- Reduces overfitting
- Makes the model tolerant towards small distortion and variations

Rotation



Thickness



CNN by itself doesn't take care of rotation and scale

- You need to have rotated, scaled samples in training dataset
- If you don't have such samples than use **data augmentation** methods to generate new rotated/scaled samples from existing training samples

CNN by itself doesn't take care of rotation and scale

- You need to have rotated, scaled samples in training dataset
- If you don't have such samples than use **data augmentation** methods to generate new rotated/scaled samples from existing training samples

CNN by itself doesn't take care of rotation and scale

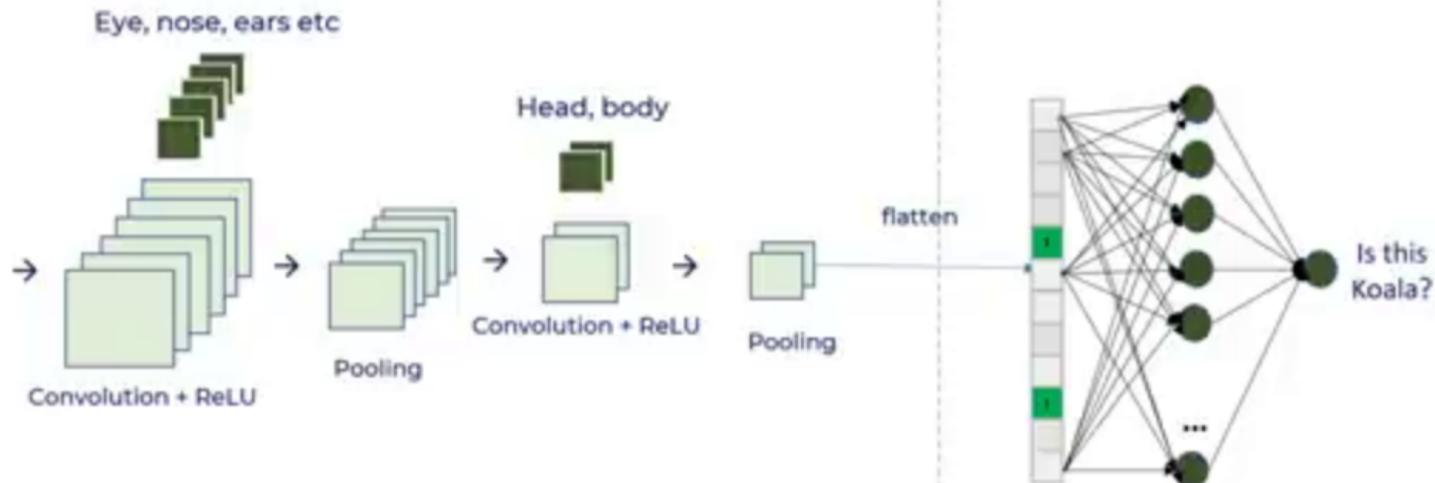
- You need to have rotated, scaled samples in training dataset
- If you don't have such samples than use **data augmentation** methods to generate new rotated/scaled samples from existing training samples

CNN by itself doesn't take care of rotation and scale

- You need to have rotated, scaled samples in training dataset
- If you don't have such samples than use **data augmentation** methods to generate new rotated/scaled samples from existing training samples

CNN by itself doesn't take care of rotation and scale

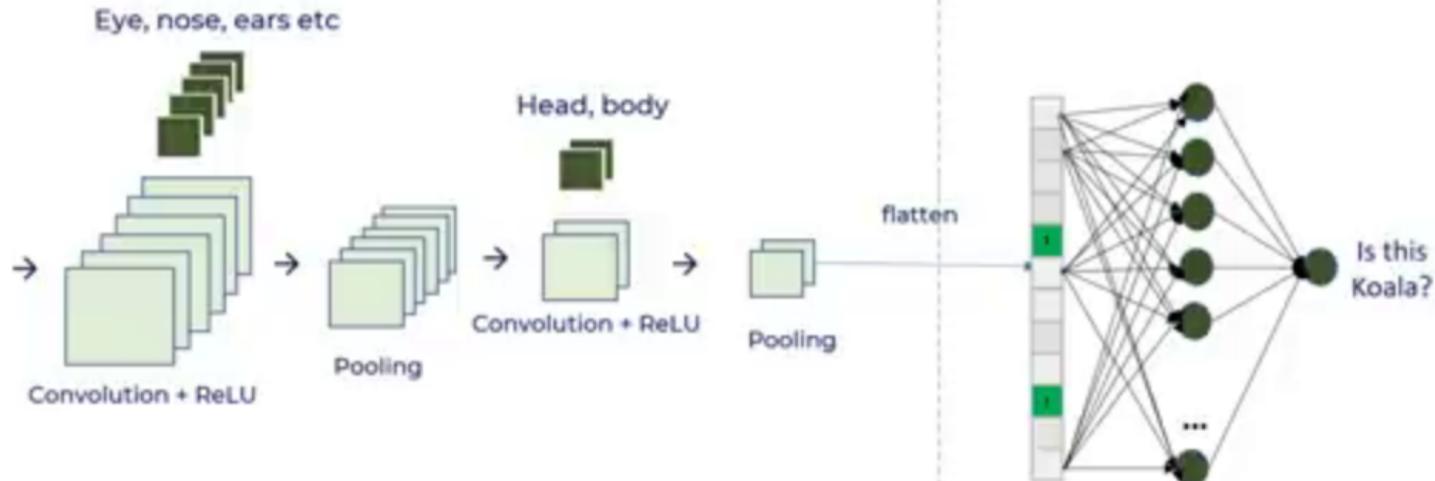
- You need to have rotated, scaled samples in training dataset
- If you don't have such samples than use **data augmentation** methods to generate new rotated/scaled samples from existing training samples



4

Feature Extraction

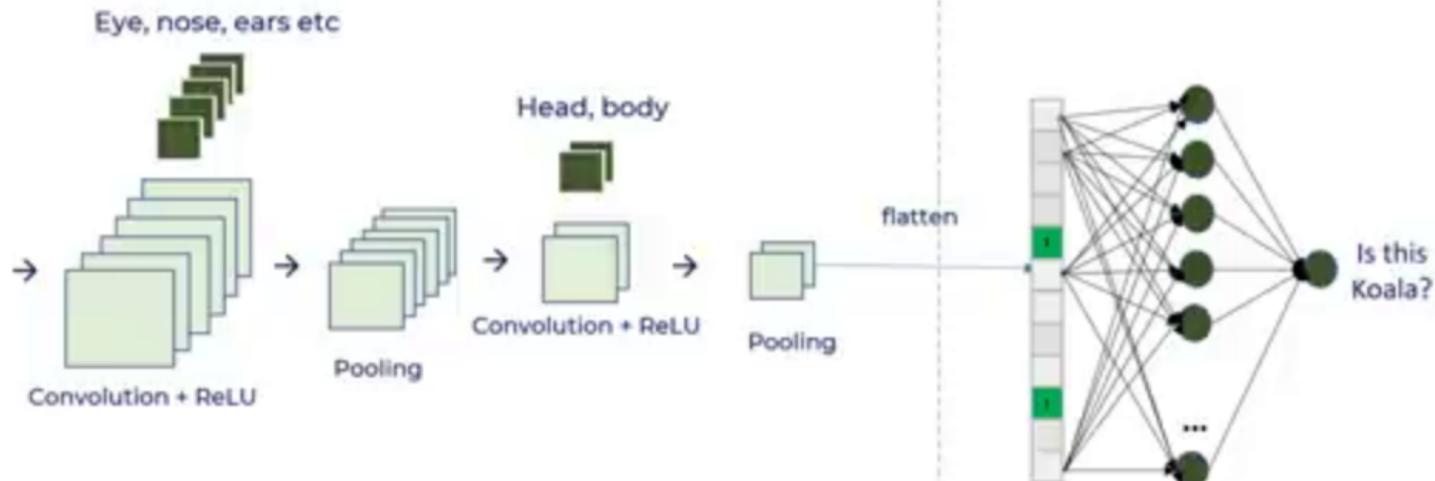
Classification



4

Feature Extraction

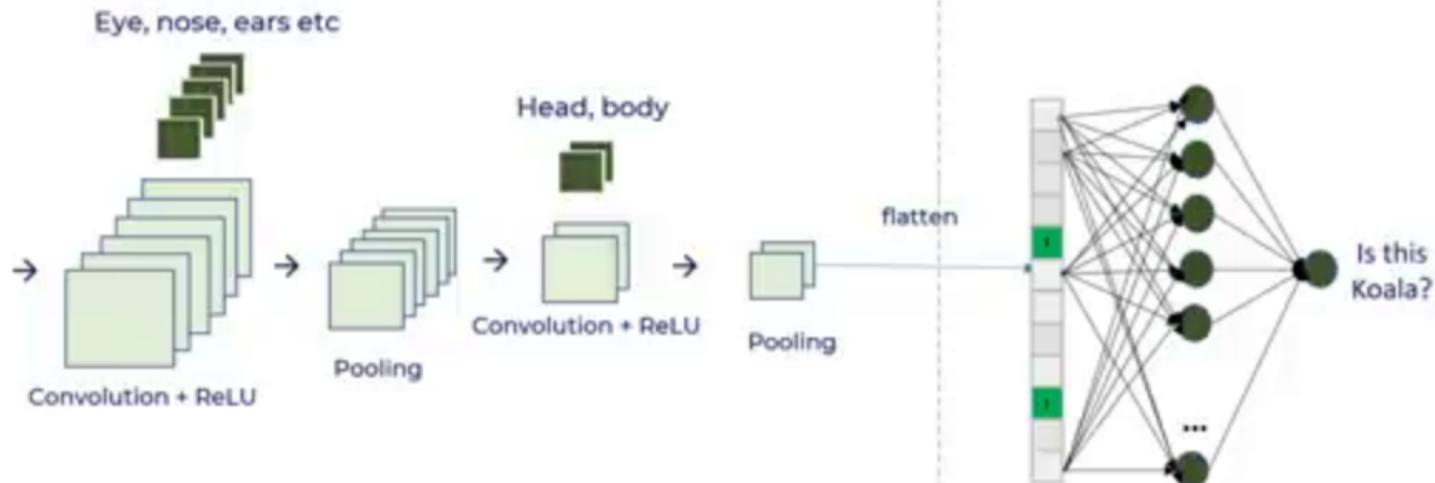
Classification



4

Feature Extraction

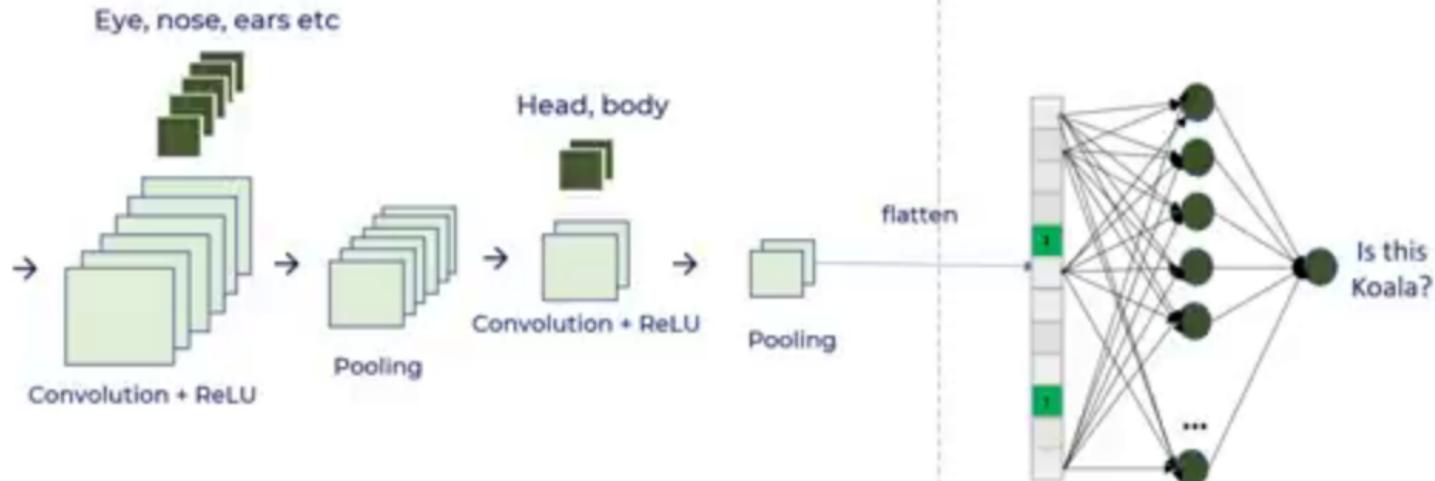
Classification



4

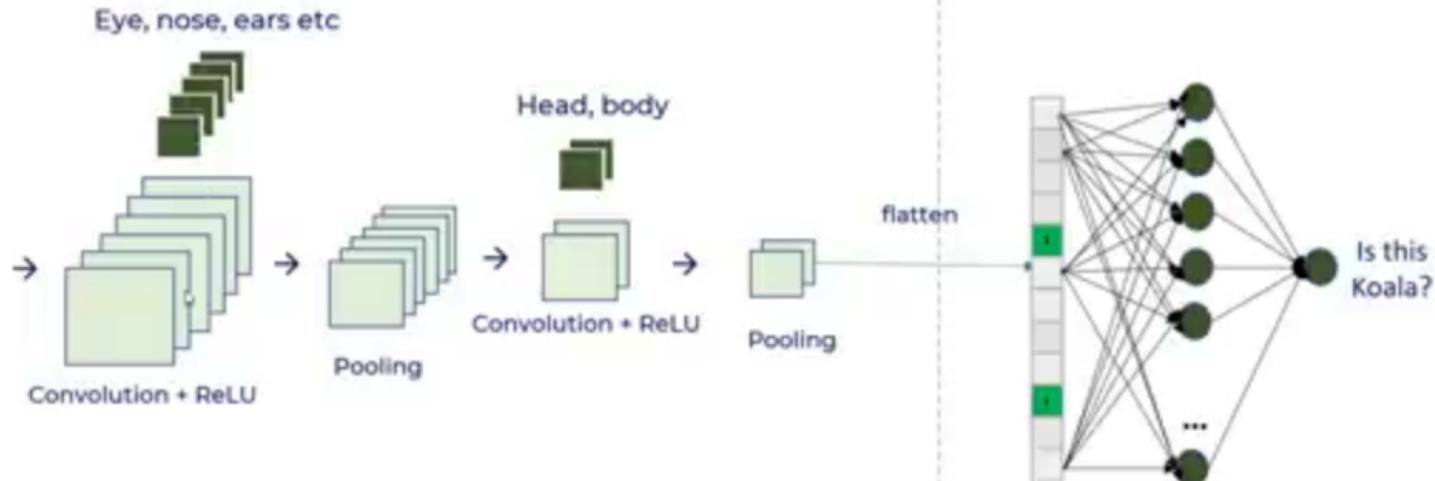
Feature Extraction

Classification



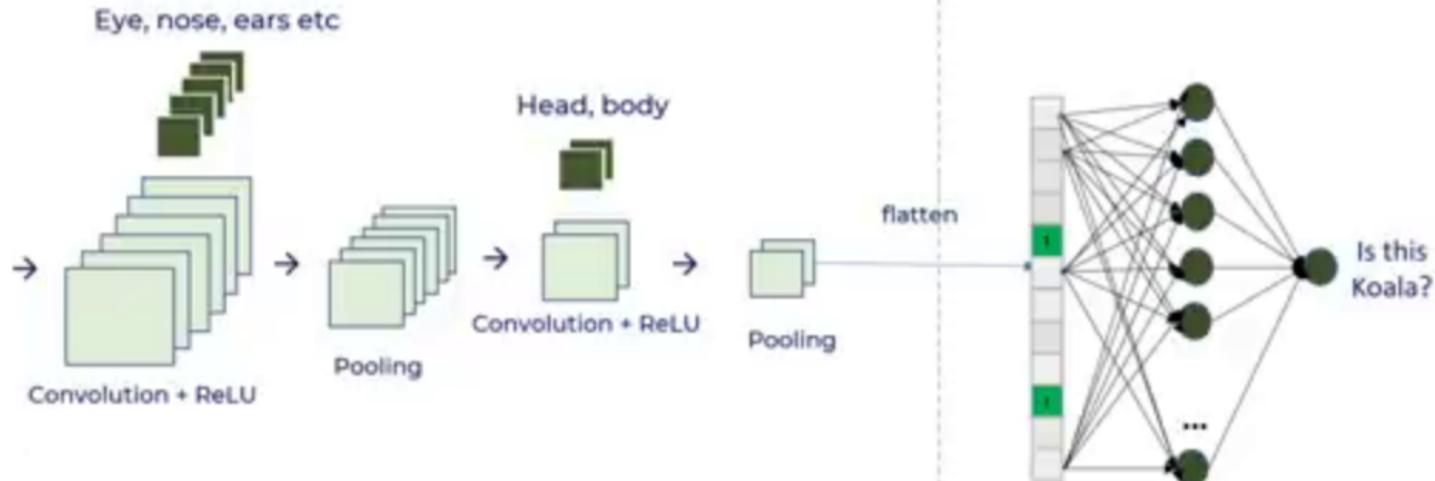
Feature Extraction

Classification



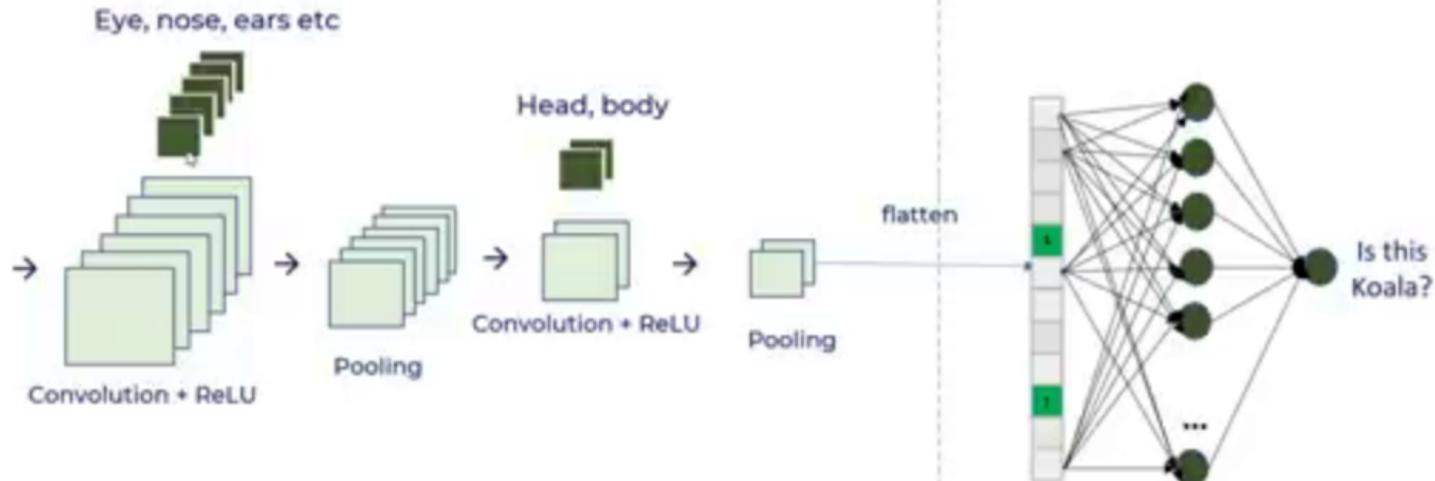
Feature Extraction

Classification



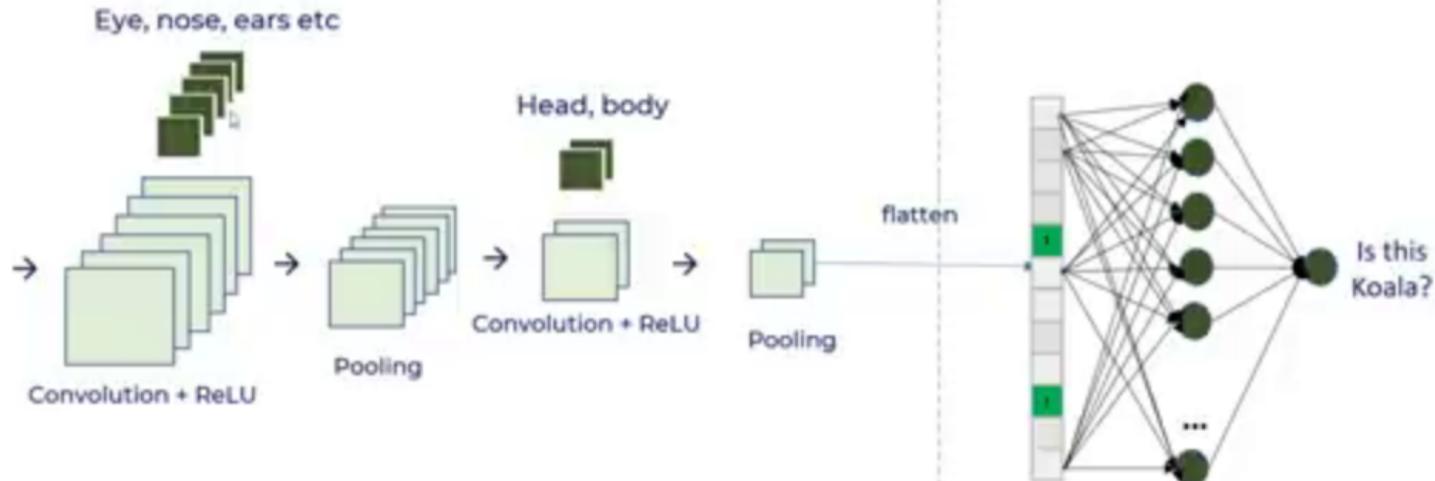
Feature Extraction

Classification



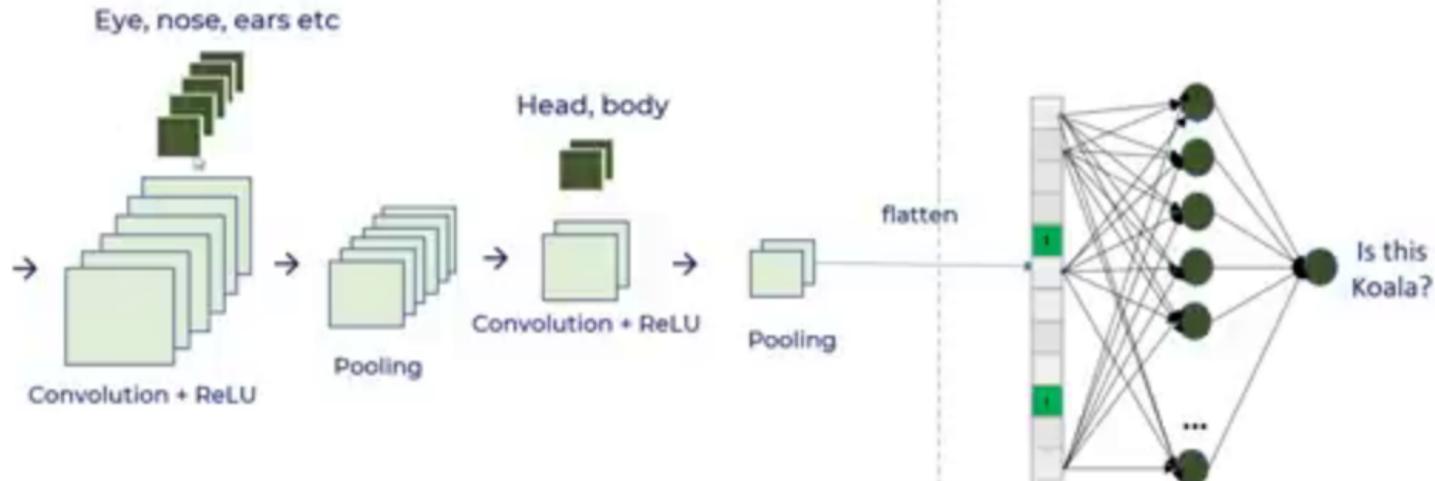
Feature Extraction

Classification



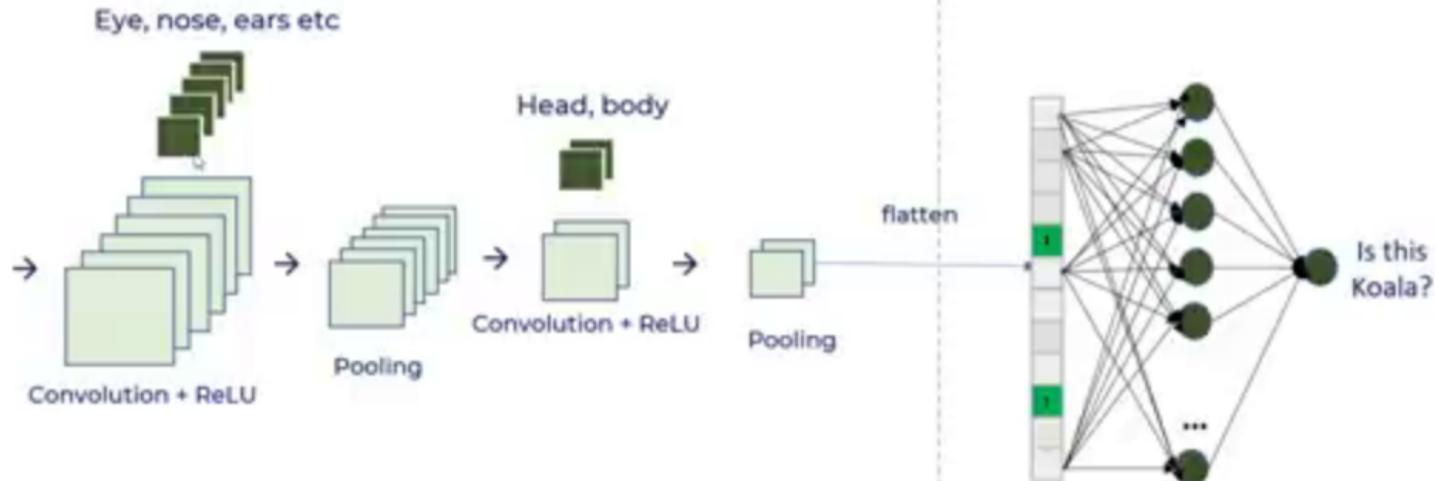
Feature Extraction

Classification



Feature Extraction

Classification



Feature Extraction

Classification







- Home
- Trending
- Subscriptions
- Library
- History
- Your videos
- Watch later
- Motivation, Technolo...
- Show more



PLAY ALL

Machine Learning Tutorial Python | Machine Learning For Beginners

59 videos • 580,262 views • Updated 6 days ago

Public



Machine learning tutorial playlist that is best suitable for a total beginner. It will start with basics of machine learning, cover various ML algorithms for regression and classification, feature engineering and also includes some real life end to end projects. In terms of technology I use these tools: sklearn, python, pandas, numpy, jupyter notebook, excel, tensorflow.

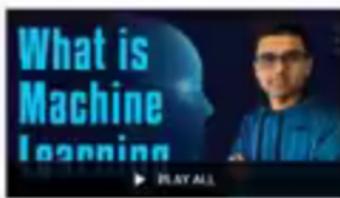


codebasics

- Machine Learning Tutorial Python - 9 Decision Tree
WATCHED 1445
- Machine Learning Tutorial Python - 10 Scaler/Normalizer (Z-Score)
WATCHED 1322
- Machine Learning Tutorial Python - 11 Support Vector
WATCHED 1249
- E Field Gross
Machine Learning Tutorial Python - 12 K-Means Clustering
WATCHED 1199
- E. Bozzo
Machine Learning Tutorial Python - 13 K-Means Clustering
WATCHED 1119
- Machine Learning Tutorial Python - 14 Naive Bayes Part 1
WATCHED 1094
- Machine Learning Tutorial Python - 15 Naive Bayes Part 2
WATCHED 1035
- Machine Learning Tutorial Python - 16 Hyper Parameter Tuning (GridSearchCV)
WATCHED 1030
- Machine Learning & Data Science Project - 1. Interactive (Real Estate Price Prediction Project)
WATCHED 934
- Machine Learning & Data Science Project - 2. Data Cleaning (Real Estate Price Prediction Project)
WATCHED 923

- More from YouTube
- YouTube Premium
- Movies & Shows
- Gaming
- Live
- Fashion & Beauty

- Home
- Frending
- Subscriptions
- Library
- History
- Your videos
- Watch later
- Motivation, Technology...
- Show more



PLAY ALL

Machine Learning Tutorial Python | Machine Learning For Beginners

50 videos • 580,282 views • Updated 6 days ago

Public



Machine learning tutorial playlist that is best suitable for a total beginner. It will start with basics of machine learning, cover various ML algorithms for regression and classification, feature engineering and also includes some real life end-to-end projects. In terms of technology I use these tools: sklearn, python, pandas, numpy, jupyter notebook, excel, tensorflow.



codebasics

- Data Science & Machine Learning Project - Part 7 Build Website | Image Classification
- Data Science & Machine Learning Project - Part 8 Deployment & Exercise | Image Classification
- What is feature engineering | Feature Engineering Tutorial Python # 1
- Outlier detection and removal using percentile | Feature engineering tutorial python # 2
- Outlier detection and removal: z score, standard deviation | Feature engineering tutorial python # 3
- Outlier detection and removal using IQR | Feature engineering tutorial python # 4
- Introduction | Deep Learning Tutorial 1 (Tensorflow2.0, Keras & Python)
- Why deep learning is becoming so popular? | Deep Learning Tutorial 2 (Tensorflow2.0, Keras & Python)
- What is a neuron? | Deep Learning Tutorial 3 (Tensorflow2.0, Keras & Python)
- Very Simple Explanation Of Neural Network | Deep Learning Tutorial 4 (Tensorflow2.0, Keras & Python)

