# Image Recognition

Dr. Phil. Dativa Tibyampansha

# Overview

- What is image recognition?
- What AI tools can be applied to achieve image recognition
  - Introduction to Convolutional Neural Networks (CNN)
- CNN Hands-on exercise

# What is Image Recognition?

- Ability to see an image of an object and knowing what it is.
- We essentially classify everything that we see into categories based on their attributes
- This gives us ability to categorize items we have never seen before.
- Common AI applications

classification

Cat

object detection

Dog, Dog, Cat

segmentation

Dog, Dog, Cat

video processing, natural language processing

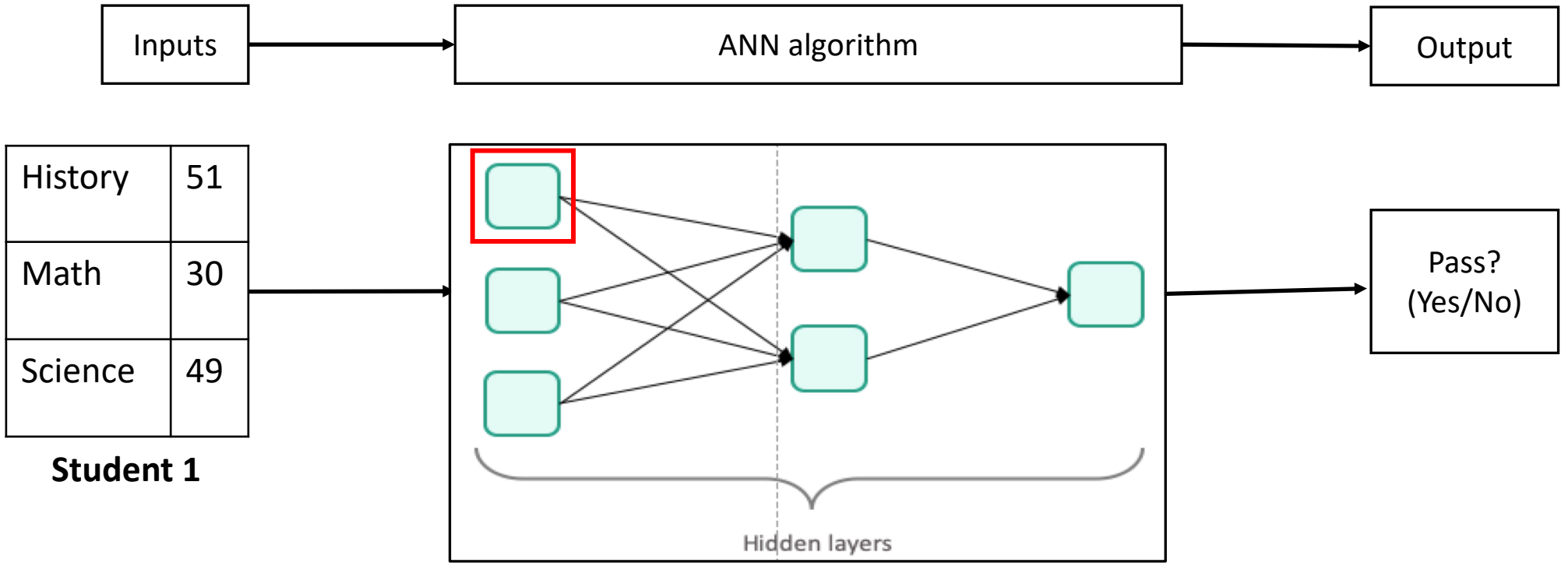
# What AI for image recognition?

- Convolutional Neural Networks (CNN)
- Class of Artificial Neural Networks (ANN)

# Artificial Neural Networks (ANN)- Intro

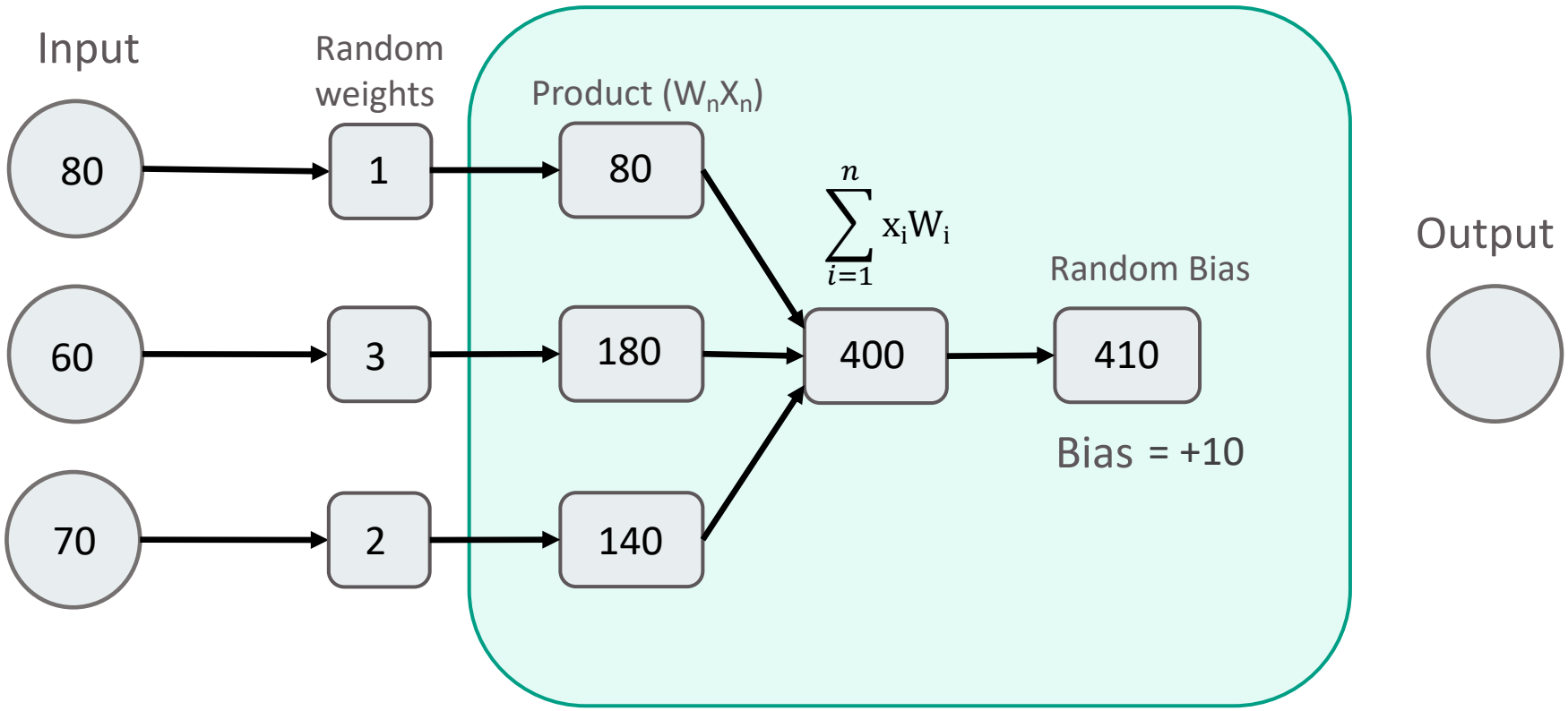
**Case scenario:** Predict if students will pass a final exam based on grades from previous students

**Application**: Early motivational guidance for students at risk of failing the final exam (“work harder”)



# Neuron – info processing

| Student/ Subject | S3 | S2 | S1 |
|---|---|---|---|
| History (H) | 60 | 50 | 80 |
| Math (M) | 70 | 45 | 60 |
| Science (S) | 50 | 51 | 70 |

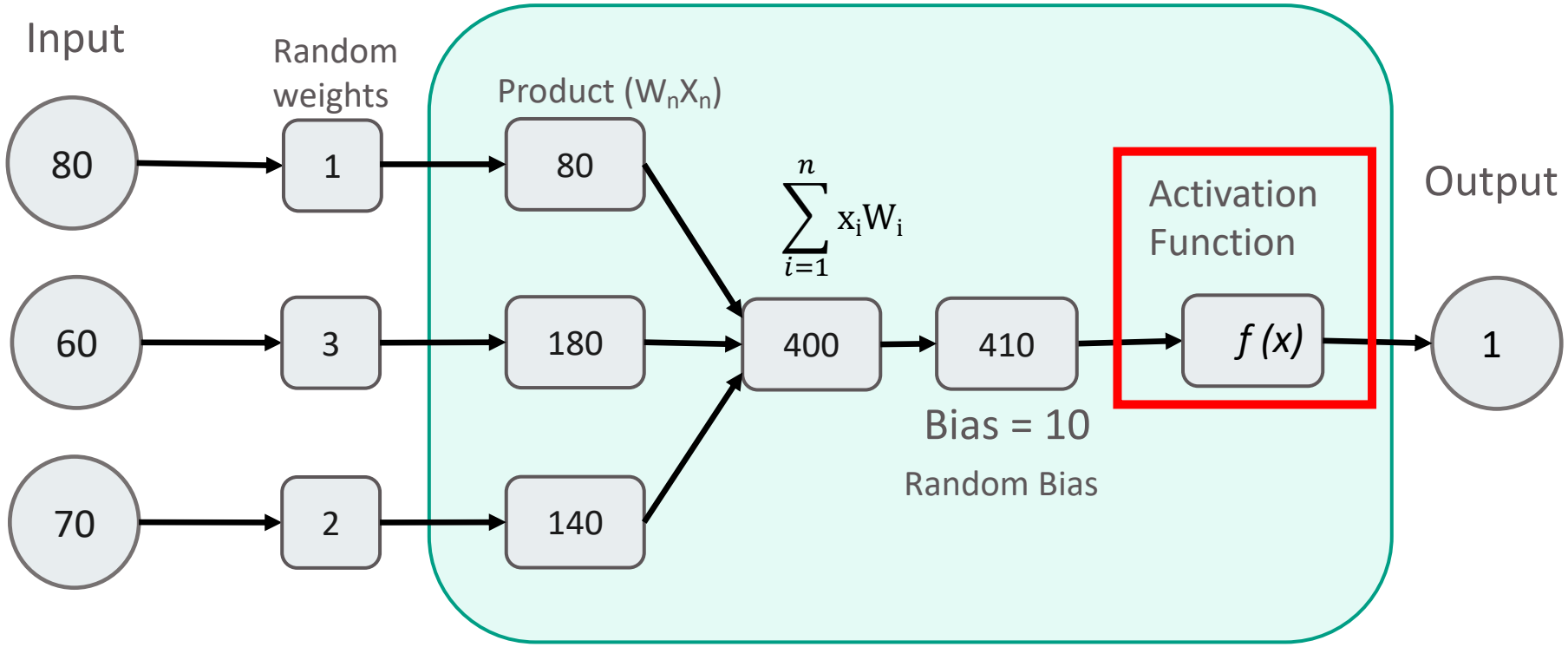



| | S3 | S2 | S1 |
|---|---|---|---|
| Grades | 1 | 0 | 1 |

# Neuron - Activation Function

| Student/ Subject | S3 | S2 | S1 |
|---|---|---|---|
| History (H) | 60 | 50 | 80 |
| Math (M) | 70 | 45 | 60 |
| Science (S) | 50 | 51 | 70 |

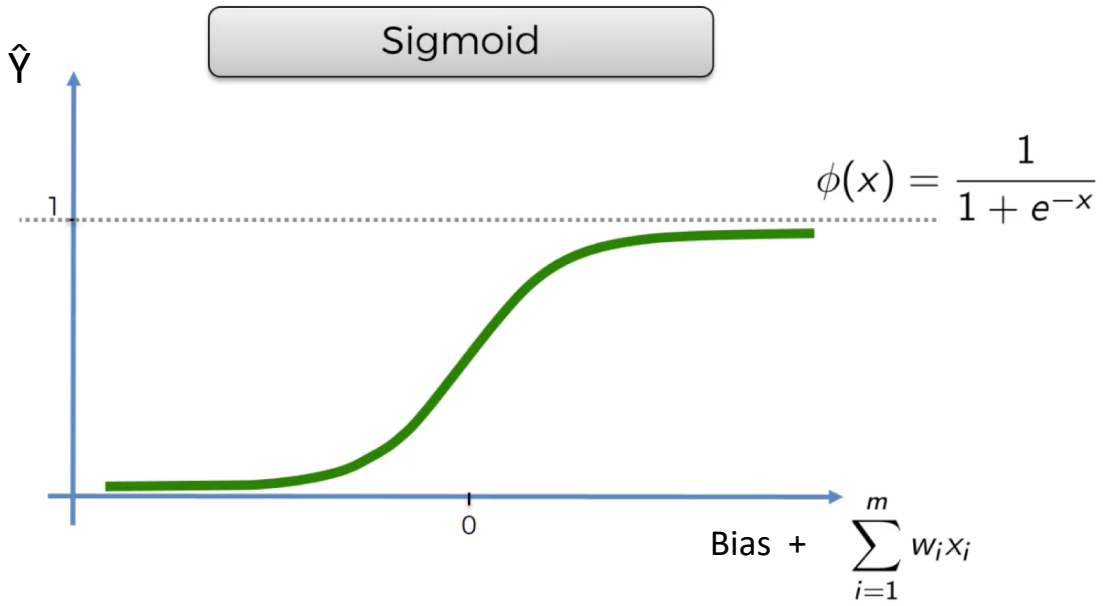
| | S3 | S2 | S1 |
|---|---|---|---|
| Grade | 1 | 0 | 1 |

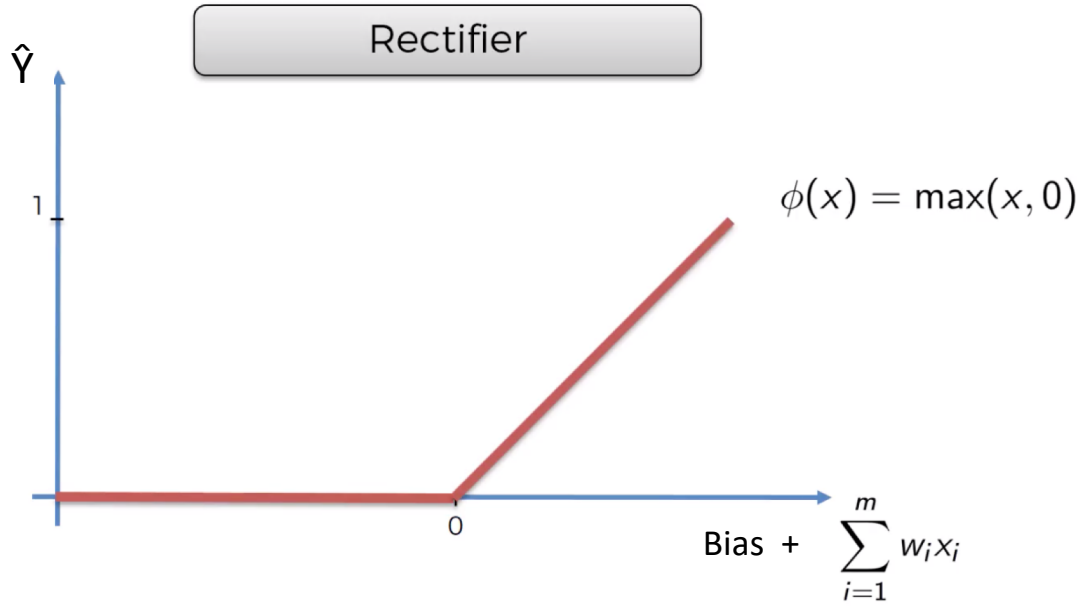



Activation Function: $f(x) = \begin{cases} 1 \text{ if } x \geq 200 \\ 0 \text{ if } x < 200 \end{cases}$

$$X = \text{bias} + \sum_{i=1}^{n} x_i W_i$$

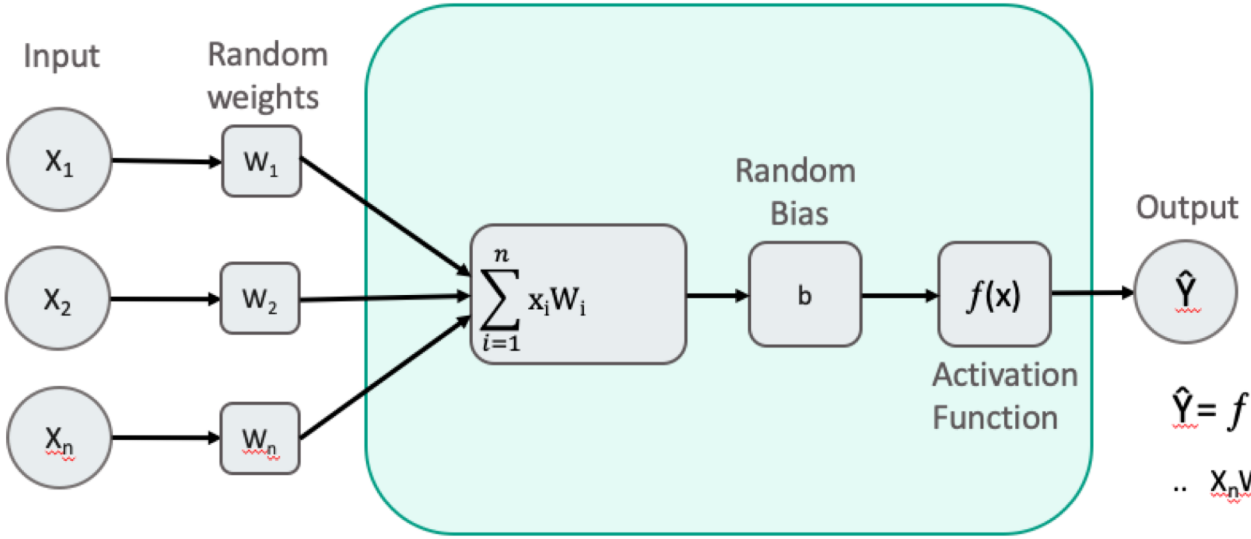
# Examples of Activation Functions



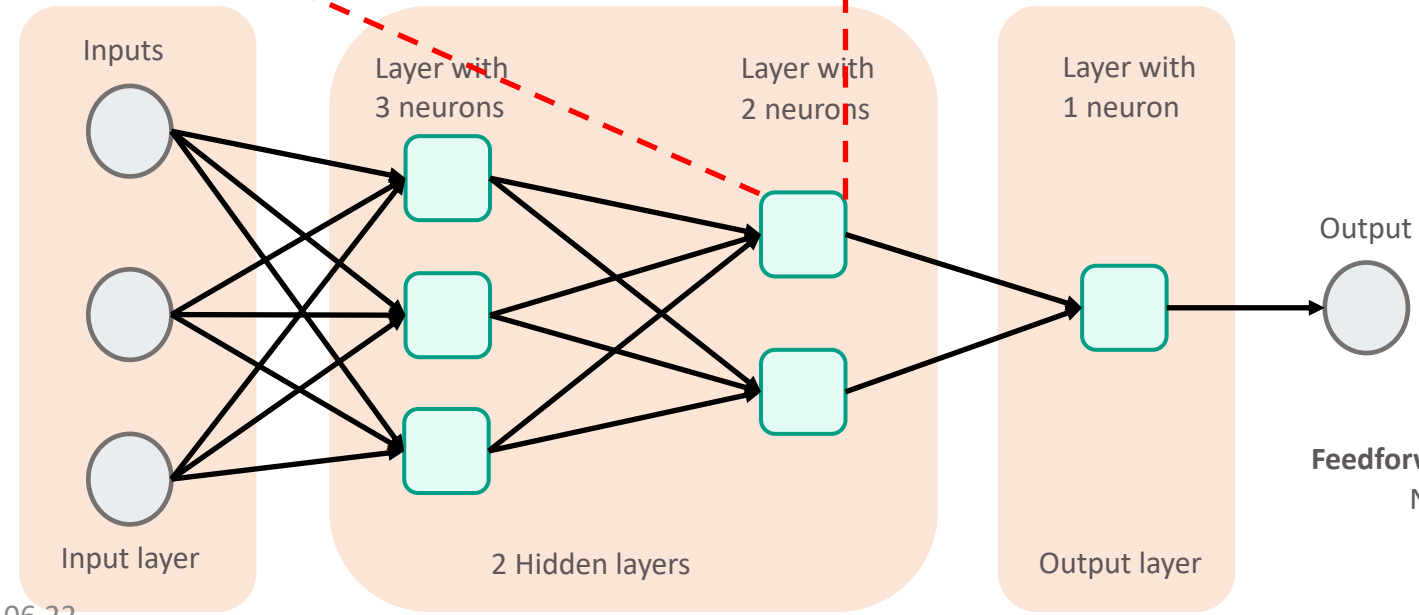




Ŷ = predicted output

# Layers – Higher-order ANN elements



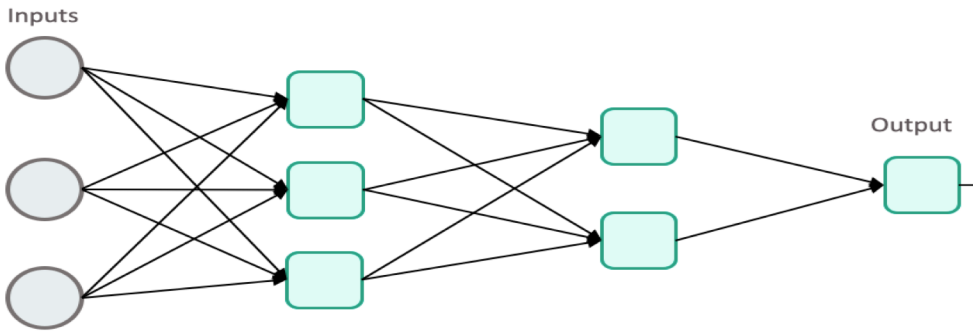

$$\hat{Y} = f\left( (X_1W_1 + X_2W_2 + .. X_nW_n) + b \right)$$




**Feedforward /Fully Connected** Neural Network

# How does an ANN learn?

| | S3 | S2 | S1 |
|---|---|---|---|
| H | 60 | 50 | 80 |
| M | 70 | 45 | 60 |
| S | 50 | 51 | 70 |




| | S1 | S2 | S3 |
|---|---|---|---|
| Y | 1 | 0 | 1 |

| | S1 | S2 | S3 |
|---|---|---|---|
| $\hat{Y}$ | 1 | 1 | 0 |

**Cost function**

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

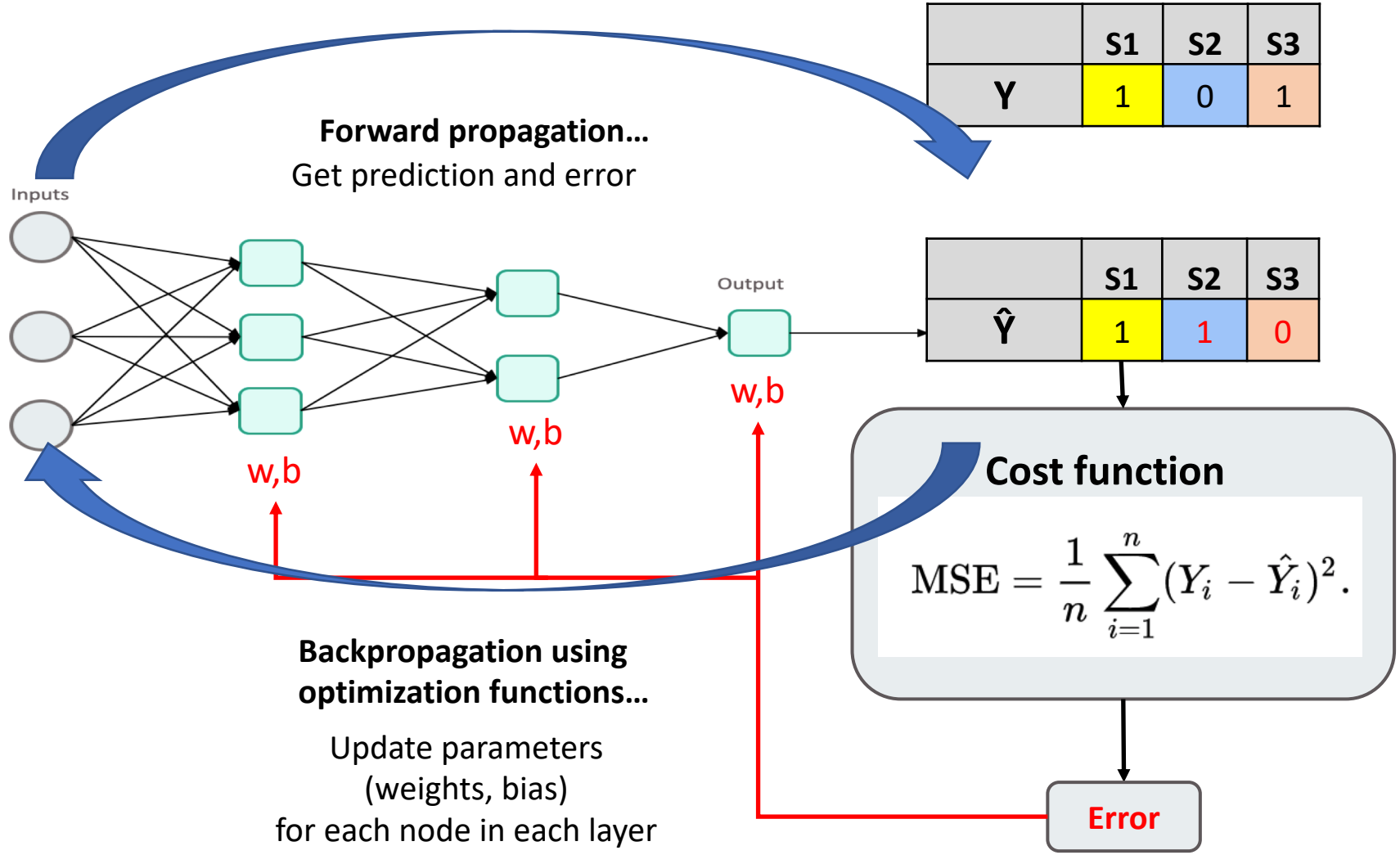
**Error**

- **Cost function:** measure of error between its estimated value **($\hat{Y}$)** and true value **(Y)**
- Application and examples of cost functions
  - Regression: Mean Squared Error Loss,
  - Binary Classification: Binary Cross-Entropy, Hinge Loss
  - Multi-class Classification: Multi-class Cross Entropy Loss

https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

# ANN – optimizing …

| | S3 | S2 | S1 |
|---|---|---|---|
| H | 60 | 50 | 80 |
| M | 70 | 45 | 60 |
| S | 50 | 51 | 70 |

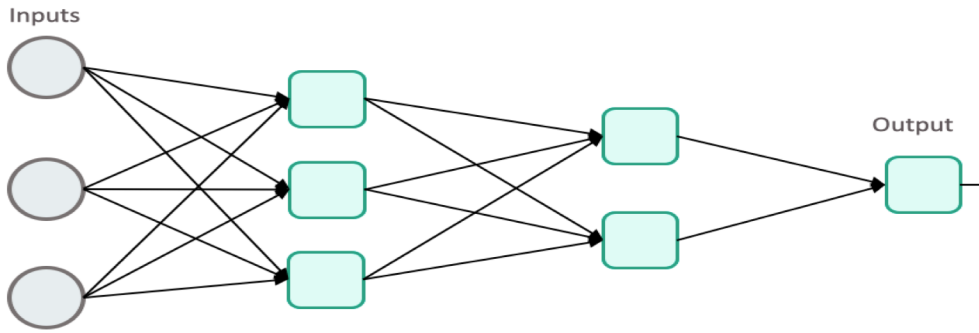



| | S1 | S2 | S3 |
|---|---|---|---|
| Y | 1 | 0 | 1 |

| | S1 | S2 | S3 |
|---|---|---|---|
| $\hat{Y}$ | 1 | 1 | 0 |

# Validation

| | S6 | S5 | S4 |
|---|---|---|---|
| H | 65 | 64 | 88 |
| M | 32 | 57 | 87 |
| S | 46 | 78 | 65 |

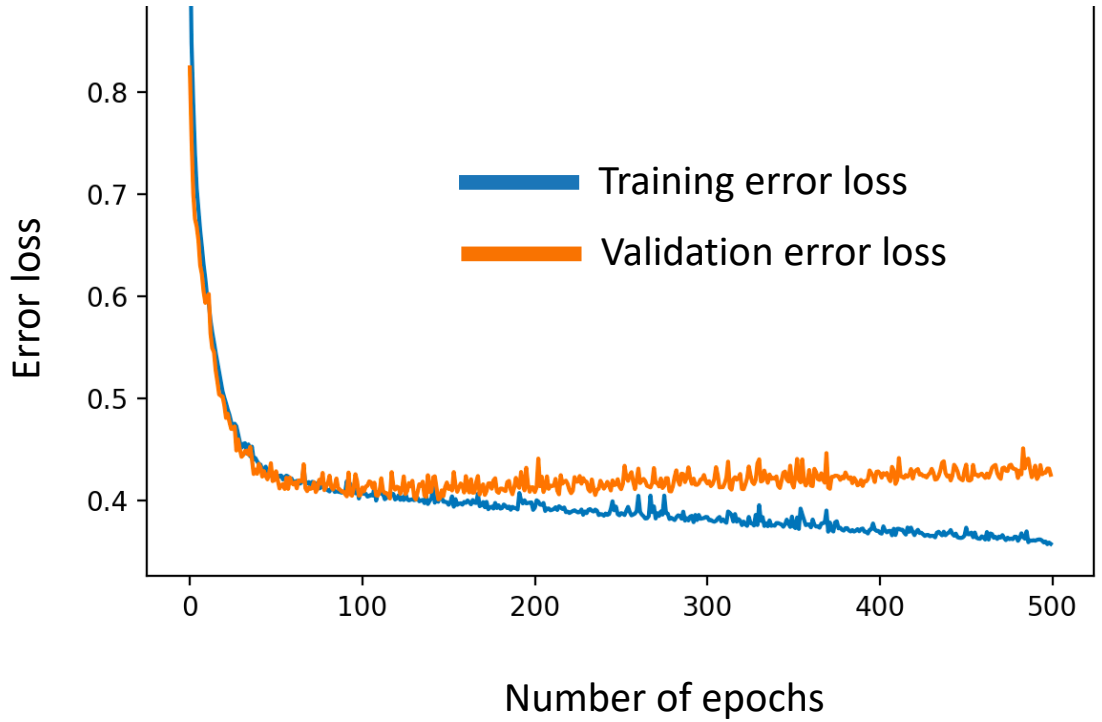



| | S6 | S5 | S4 |
|---|---|---|---|
| Y | 1 | 0 | 1 |

| | S4 | S5 | S6 |
|---|---|---|---|
| $\hat{Y}$ | 1 | 1 | 1 |

**Cost function**

**Error**



# Application of ANN

RNA Transcription Data (x)

| | ENSG00000227232.4 | ENSG00000233750.3 | ENSG00000237683.5 |
|---|---|---|---|
| 0 | 10.338030 | 0.362602 | 5.832498 |
| 1 | 7.609902 | 0.361494 | 7.007413 |
| 2 | 7.146308 | 0.194017 | 0.121261 |
| 3 | 5.991603 | 0.207083 | 5.584339 |
| 4 | 5.382497 | 0.256680 | 6.525889 |
| 5 | 4.962599 | 0.069896 | 0.271817 |
| 6 | 5.379276 | 0.273397 | 0.399012 |
| 7 | 5.716319 | 0.103211 | 0.103211 |
| 8 | 5.469149 | 0.106916 | 0.271401 |
| 9 | 5.740473 | 0.364957 | 5.383119 |
| 10 | 5.037652 | 0.456240 | 4.819037 |
| 11 | 8.415640 | 0.117598 | 0.176398 |
| 12 | 6.828088 | 0.228110 | 0.258524 |
| 13 | 5.239712 | 0.243512 | 0.067176 |
| 14 | 4.428489 | 0.119689 | 1.735489 |
| 15 | 6.401946 | 0.235315 | 0.283762 |
| 16 | 6.601959 | 0.153911 | 0.550838 |
| 17 | 18.234970 | 0.227369 | 5.237059 |

Labels(y)

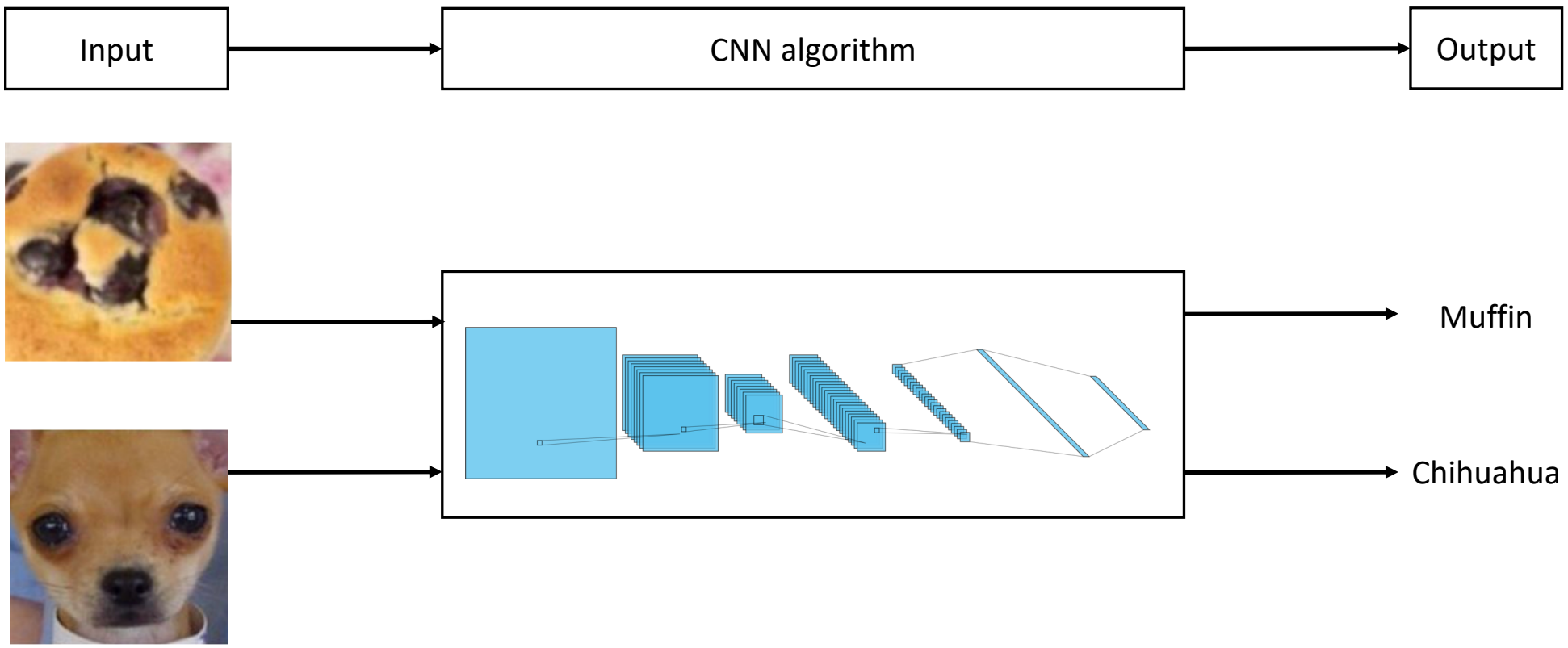
| Doxorubicin |
|---|
| 0.106768 |
| 0.026413 |
| 0.160035 |
| 0.056119 |
| 0.035957 |
| 0.043444 |
| 0.017766 |
| 0.045722 |
| 0.026402 |
| 0.268859 |
| 0.020532 |
| 0.026486 |
| 0.045192 |
| 0.061365 |
| 0.010237 |
| 0.020460 |
| 0.034832 |
| 0.038035 |

ANN definition with Keras in python

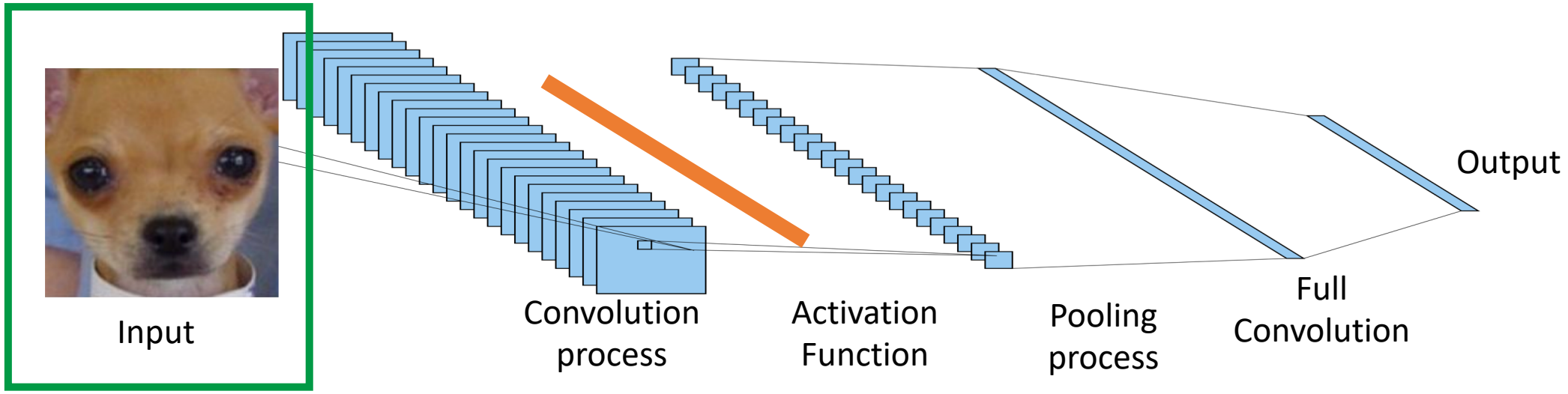
```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('data.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

# Convolutional Neural Networks

Case scenario: Classify between a chihuahua and a muffin

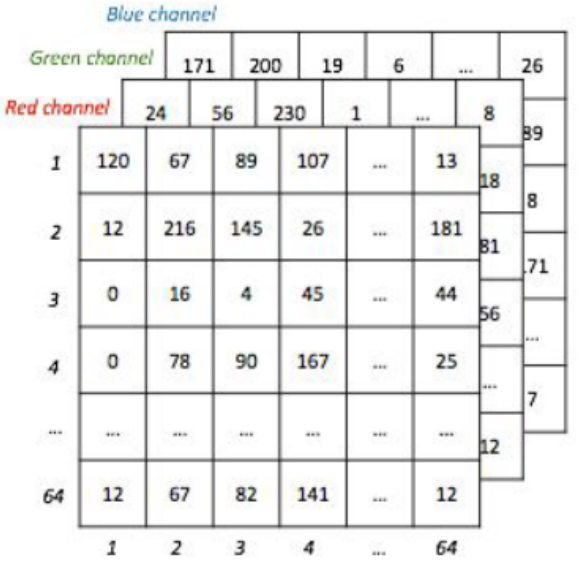


# CNN Steps

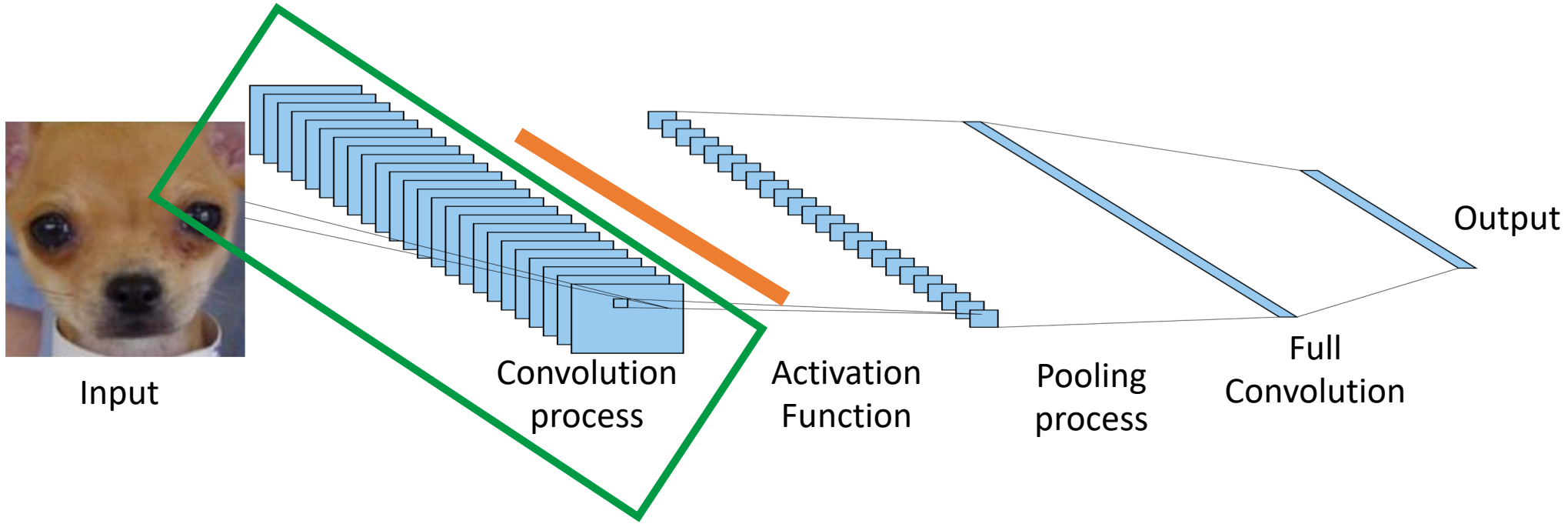

Input
Convolution process
Activation Function
Pooling process
Full Convolution
Output

# Input - images are numbers!

- Consist of pixels, with intensities 0 to 255
- Colors channels:
  - RGB (almost all) vs. BGR (OpenCV)
  - Other color spaces exist.
- Shape:
  - Computer vision: [height, width, channels] (3D)
  - Deep learning: [batch_size, height, width, channels] (4D)

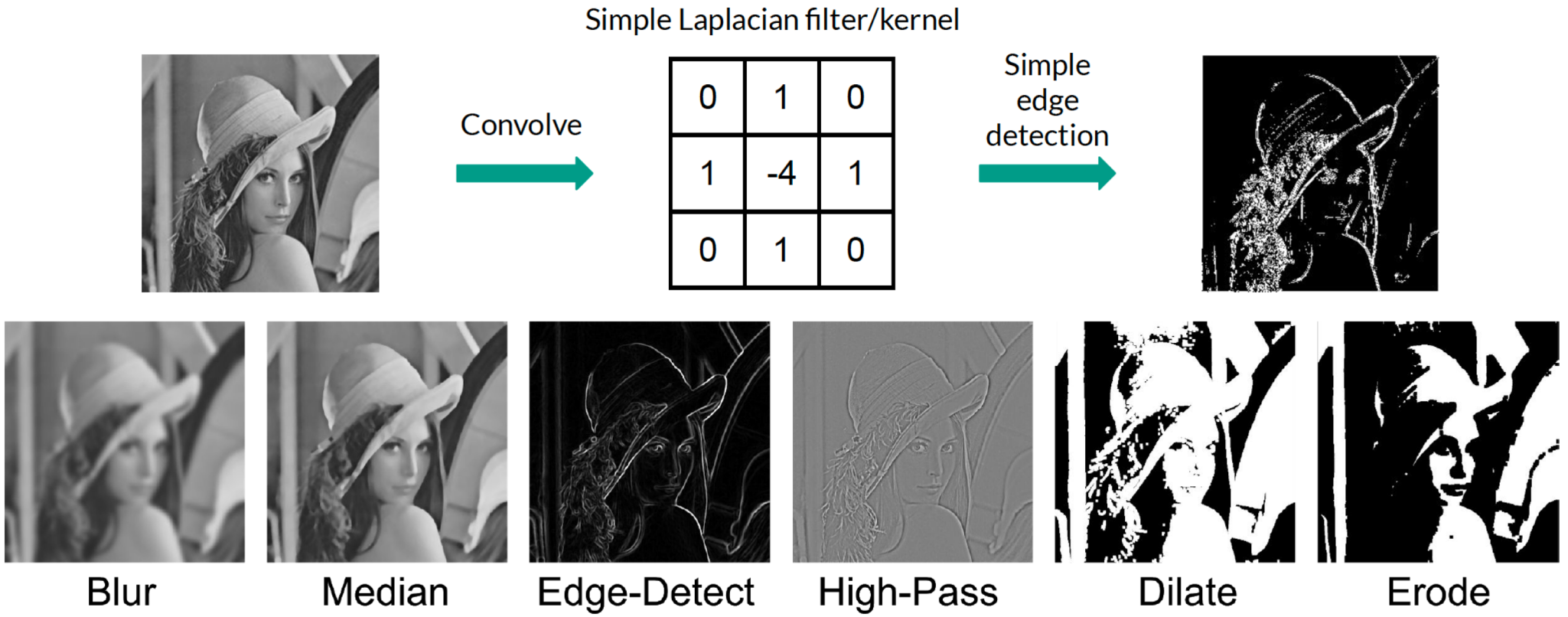


# CNN Steps


Input
Convolution process
Activation Function
Pooling process
Full Convolution
Output

# Convolution

- **matrix filters** extract features from images



# Convolution process

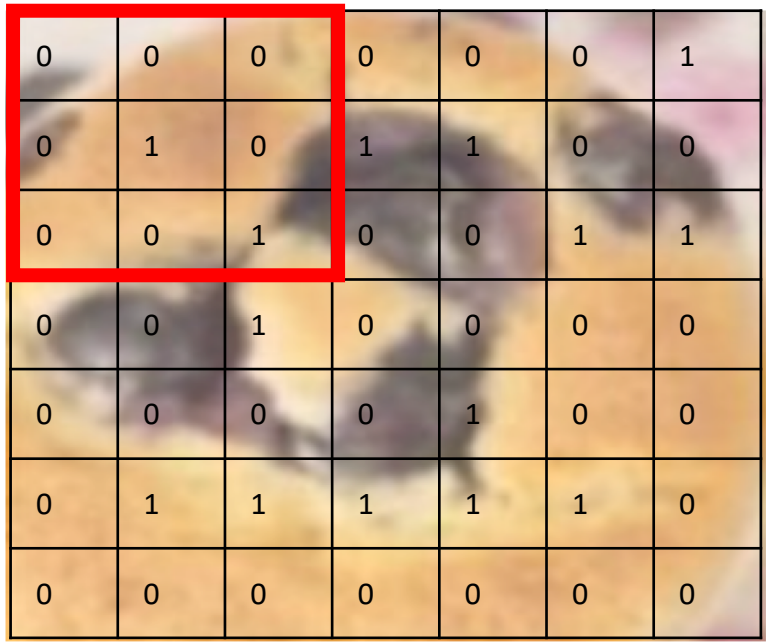
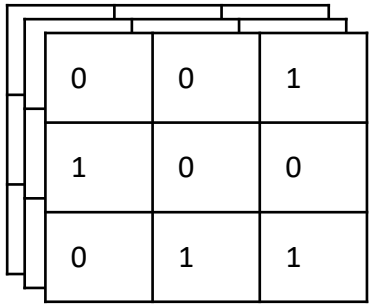



Image input

X

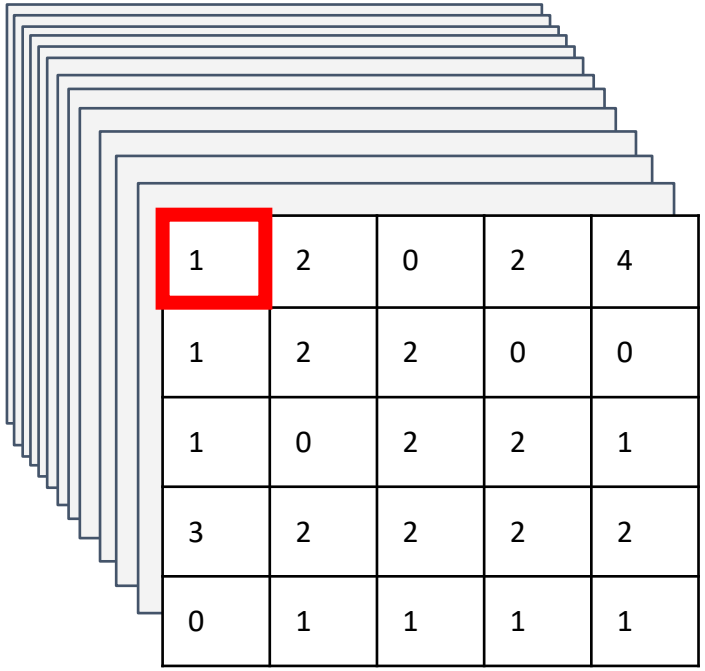
Random learnable **weights**




Kernel (3X3)
Stride (1,1)

=

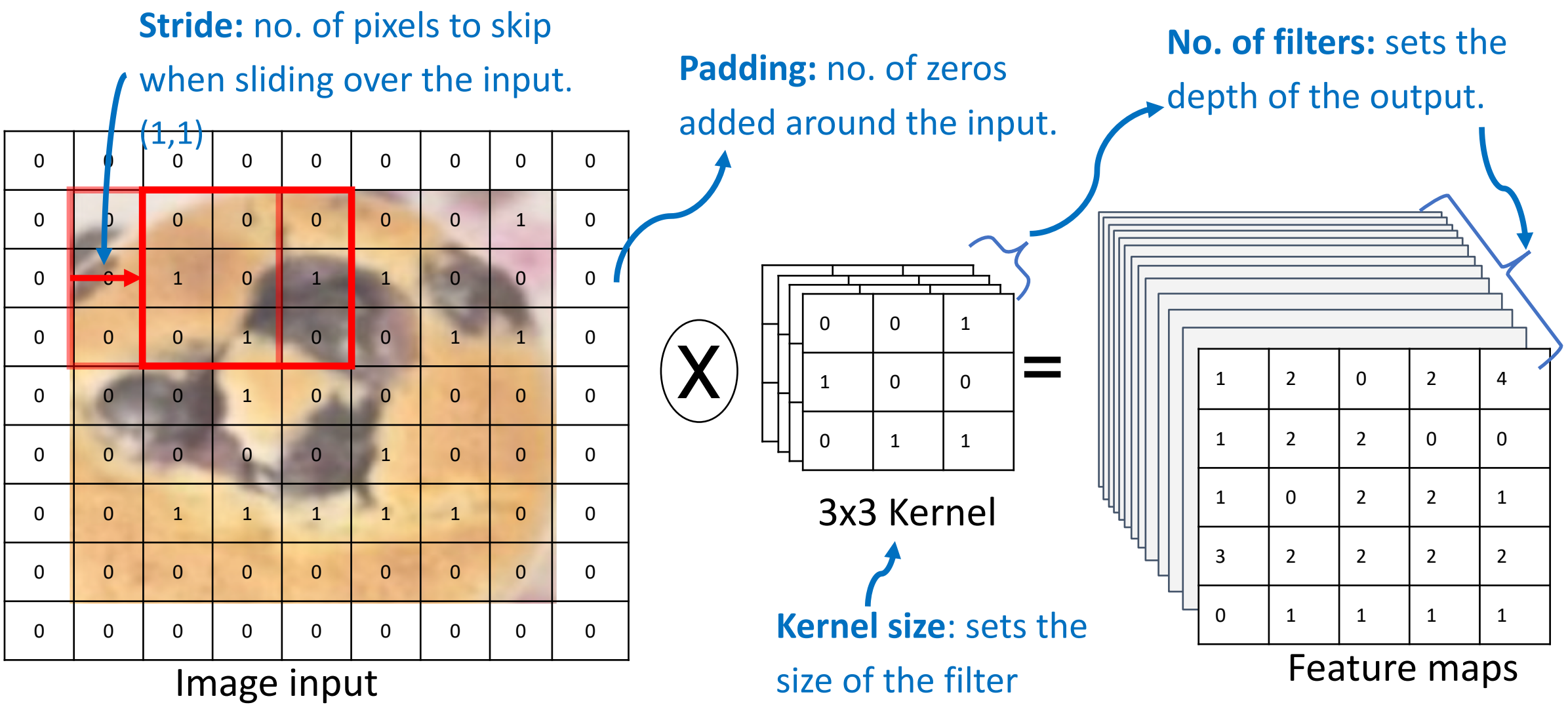
dot-product of image pixels with a kernel weights



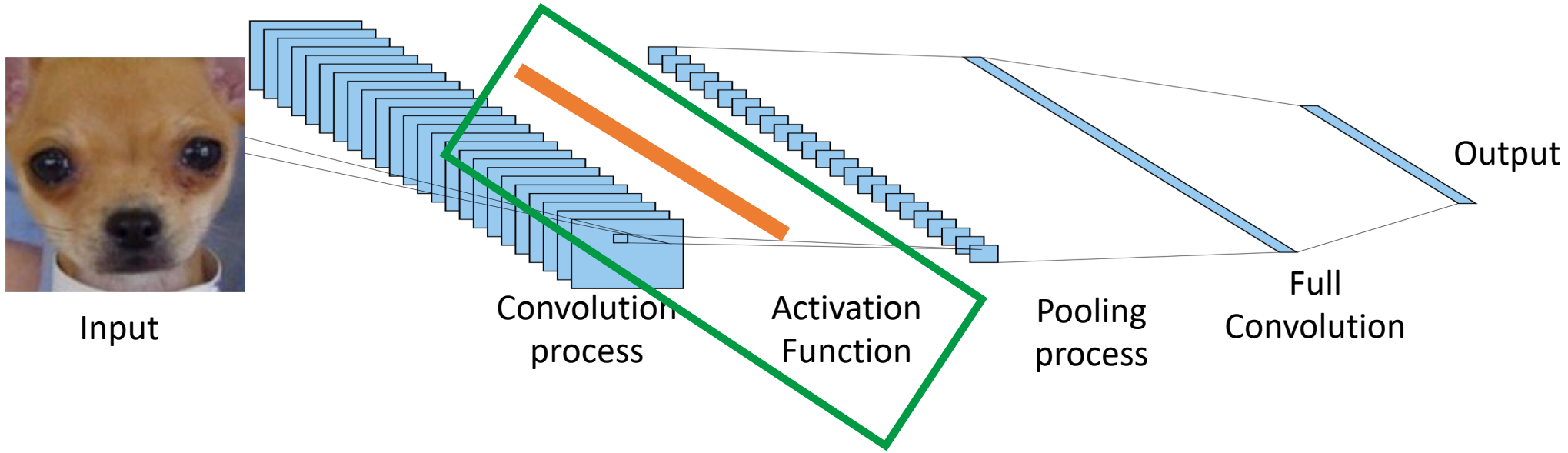

Feature maps

# Hyperparameters

- The size of the feature map can be controlled by the following hyper-parameters:

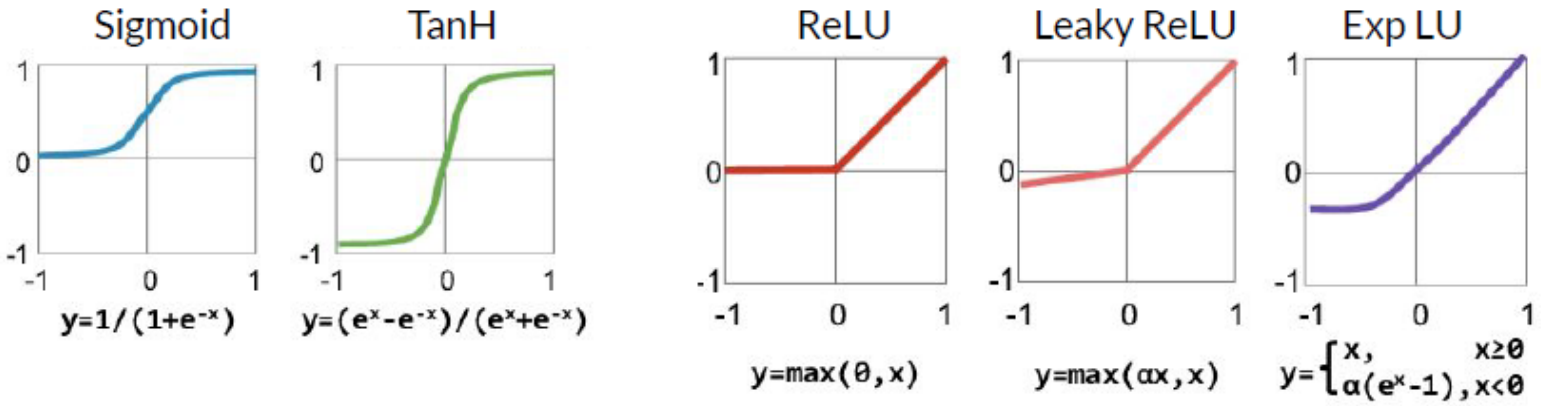


# CNN Steps

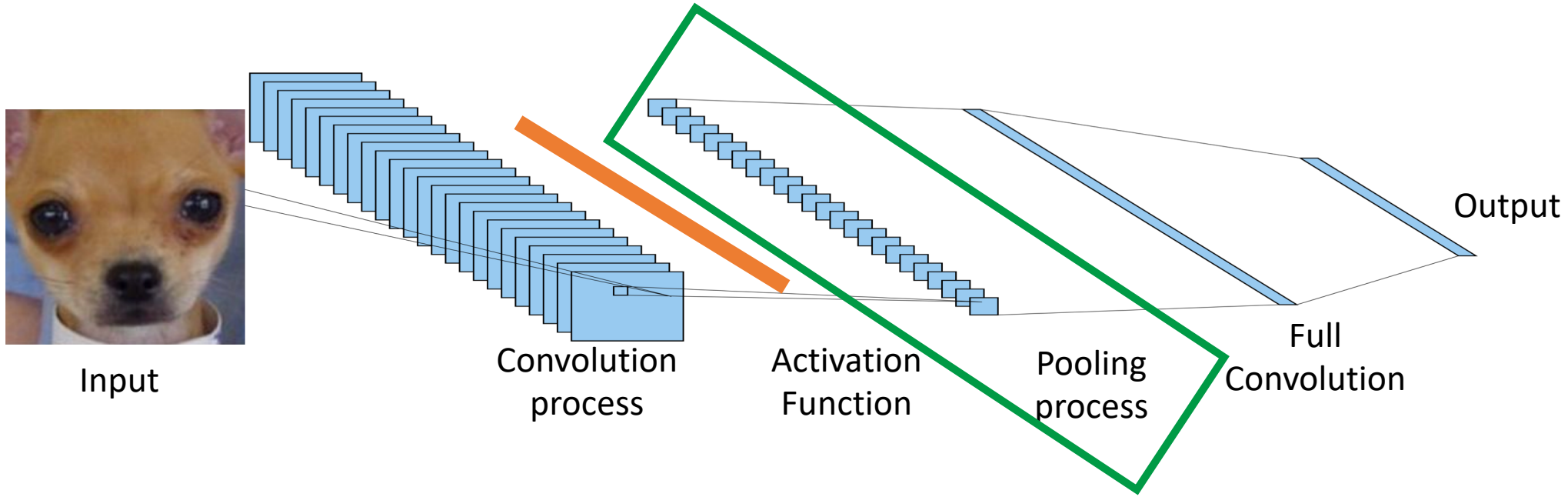

Input
Convolution process
Activation Function
Pooling process
Full Convolution
Output

# Activation Function

- Provide non-linearity to output.
- Applied to every feature map
- Rectified Linear units (**ReLU**) became popular in recent years.
  - They speed up training and help in convergence.
- Usually used directly after convolution.

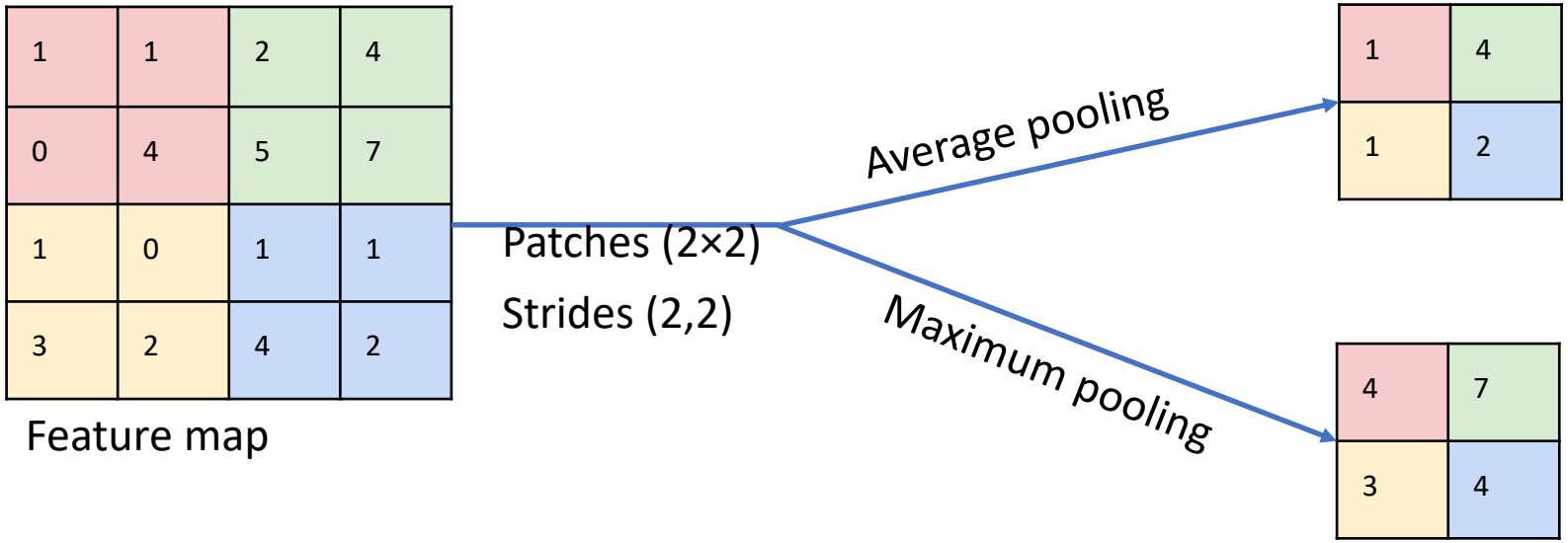


# CNN Steps

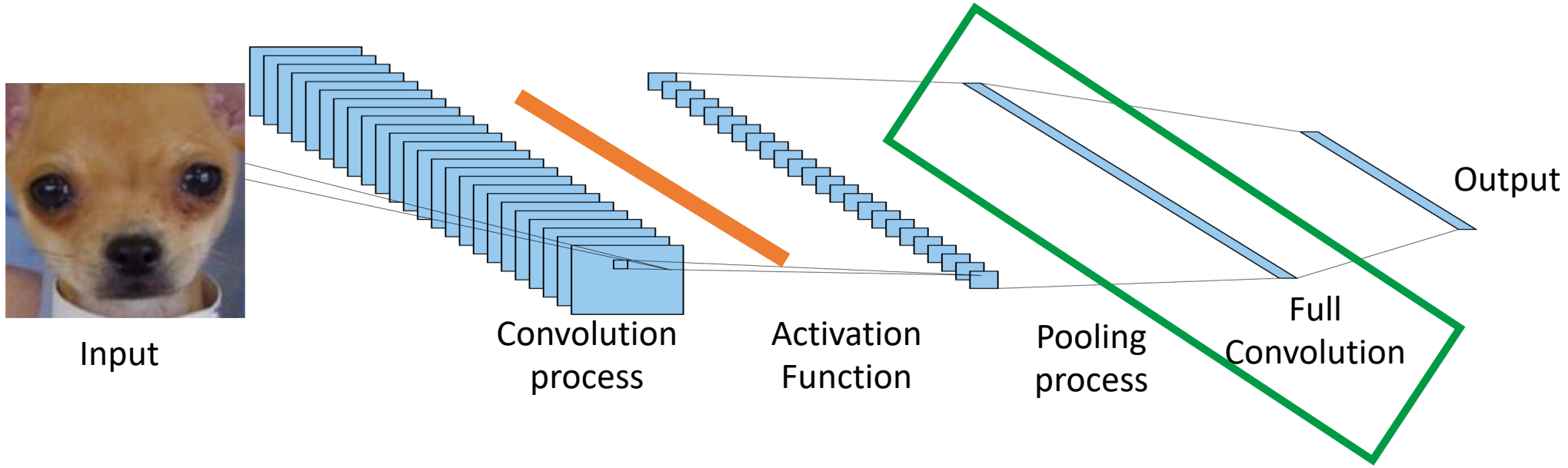

Input
Convolution process
Activation Function
Pooling process
Full Convolution
Output

# Pooling process

- Reduces the dimensionality of feature maps but retains important information.
  - Focuses on the main features
  - Reduce computational costs - gets rid of extra data
  - Reduce overfitting - combines neighboring values into a single one
- Types of pooling
  - **Average Pooling:** average value for each patch on the feature map.
  - **Maximum Pooling:** maximum value for each patch of the feature map.

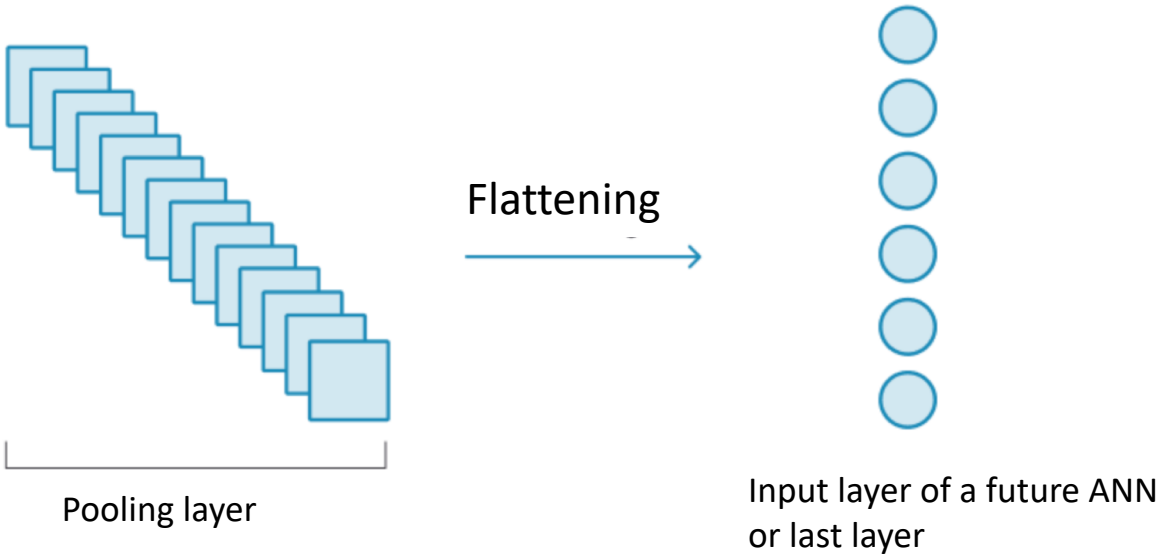


# CNN Steps

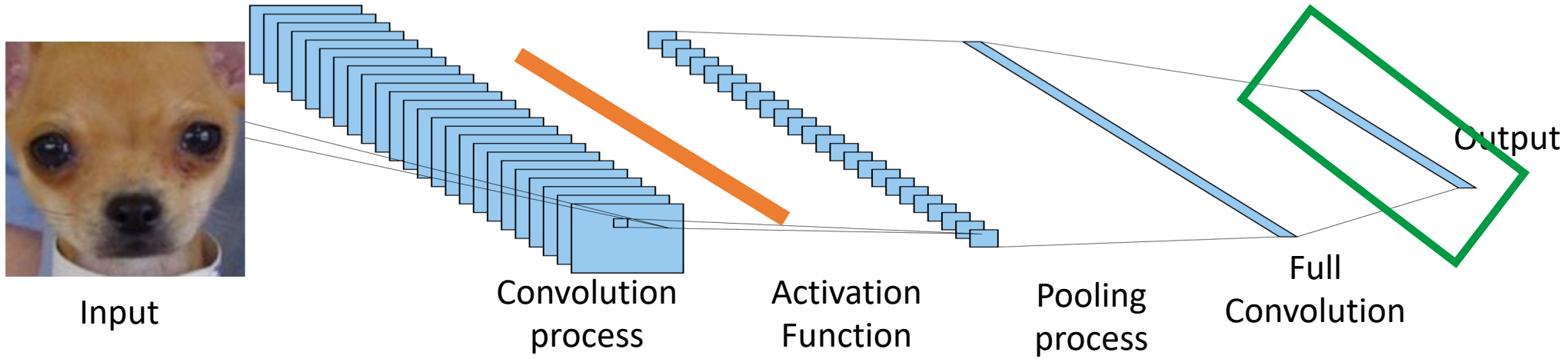

Input
Convolution process
Activation Function
Pooling process
Full Convolution
Output

# Full convolution

- Also known as Dense Layer
- Facilitated by a process called Flattening.
- Flattening transforms the feature maps into **one** vector
- The vector fed into a neural network of ***n*** layers.
- Last layer has ***c*** neurons, representing the **number of classes** to classify.

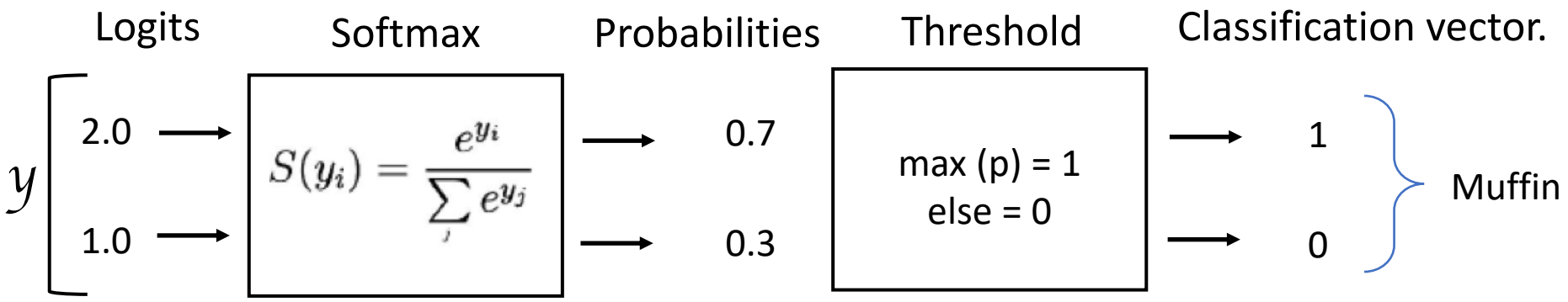


# CNN Steps

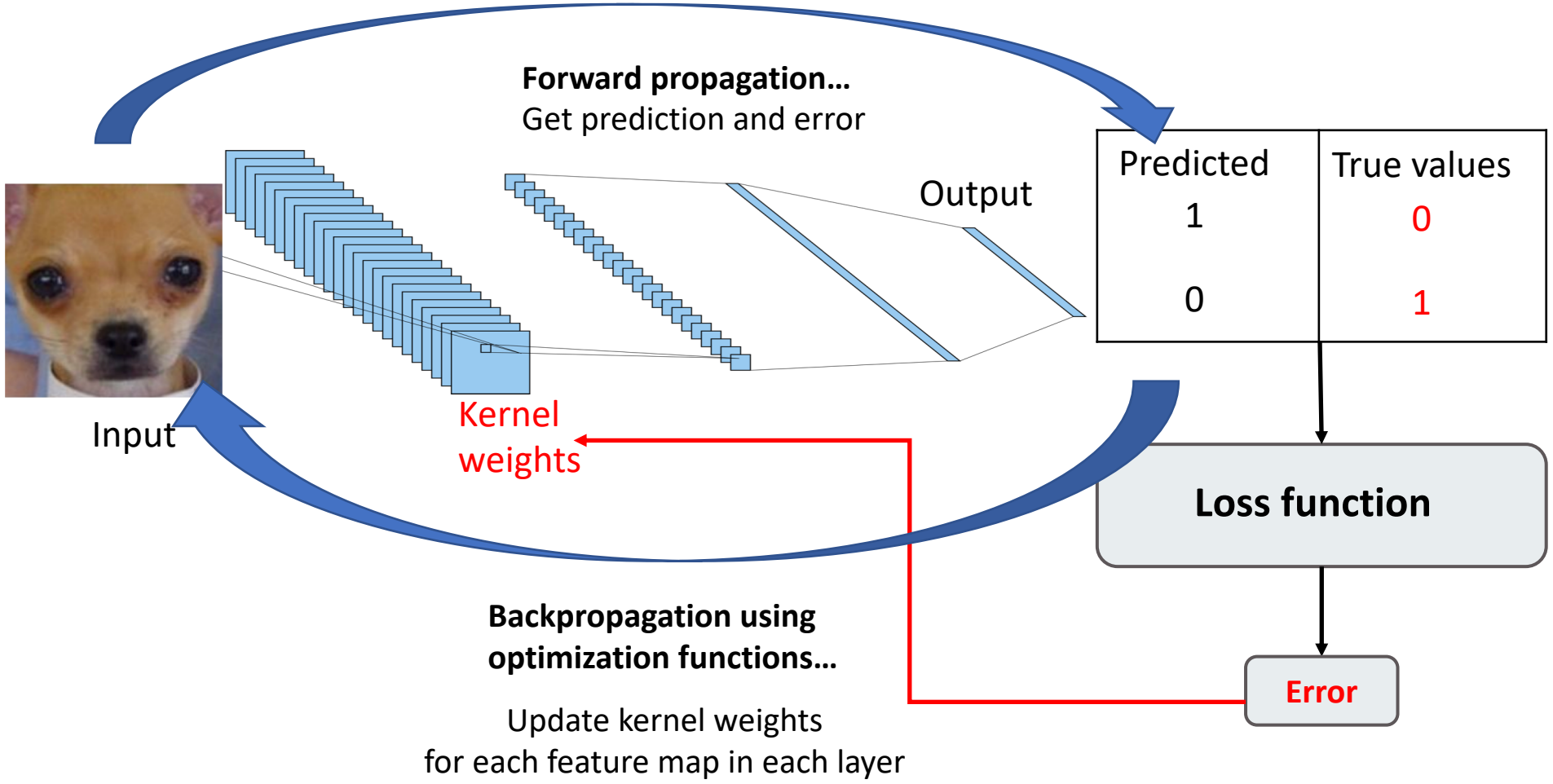

Input
Convolution process
Activation Function
Pooling process
Full Convolution
Output

# Output

- **Problem:** Last layer outputs numbers (a.k.a **logits**) are hard to interpret.
- **Solution:** normalize these numbers with an activation function
  - e.g. Softmax function results in probabilities with sum equal to 1
  - Threshold probabilities to generate a classification vector.

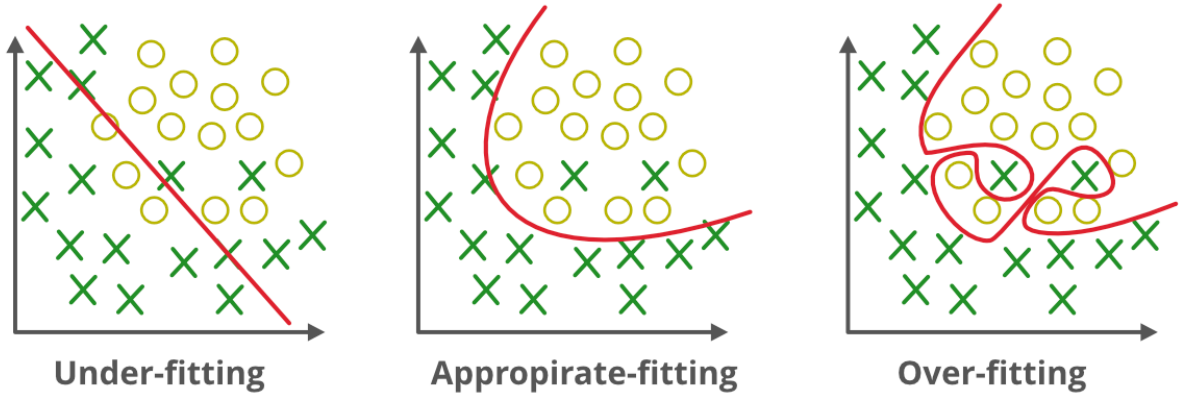


# Training - updating the kernel weights



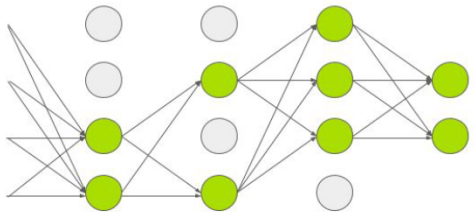
# Overfitting

Algorithm performs well on the training set but performs poorly on the testing set => the algorithm is overfitted on the Training data.

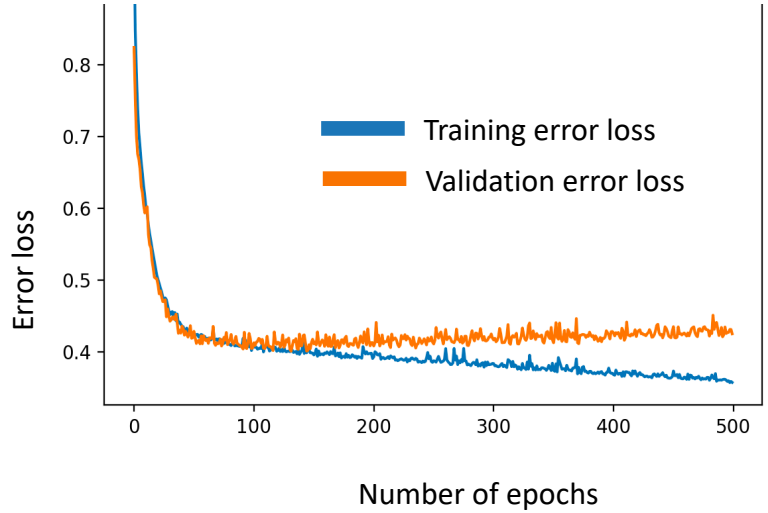



**Remedy: apply Regularization**

1. Adds penalty to weights and biases
   - **L1 :** the penalty is proportional to the absolute value of the weight coefficients.
   - **L2 :** the penalty is proportional to the square of the value of the weight coefficients
2. Drop out: randomly turns off a subset of neurons in a layer

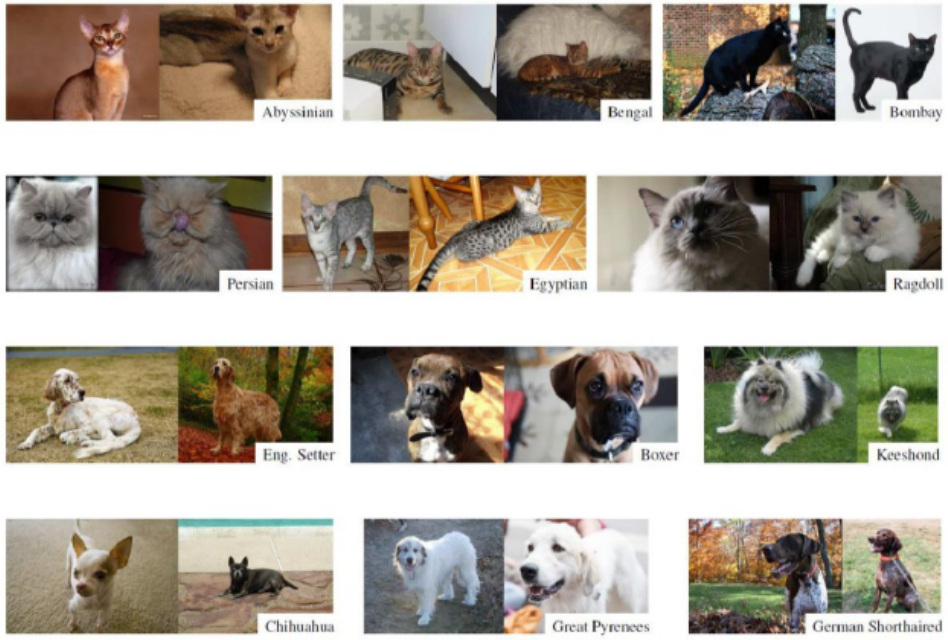


3. Regularization rate: controls the amount of regularization applied



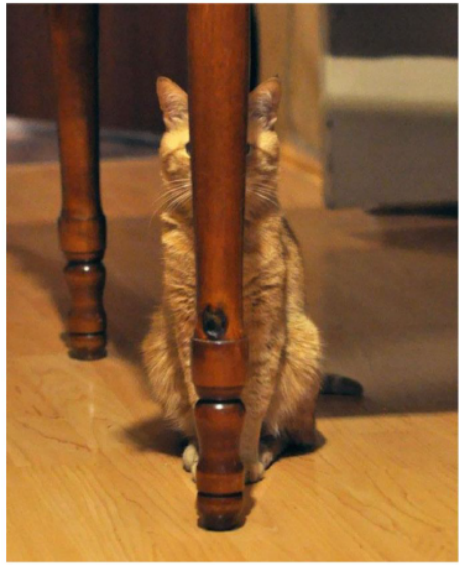
# Computer Vision is Hard

## Intra-Class Variability

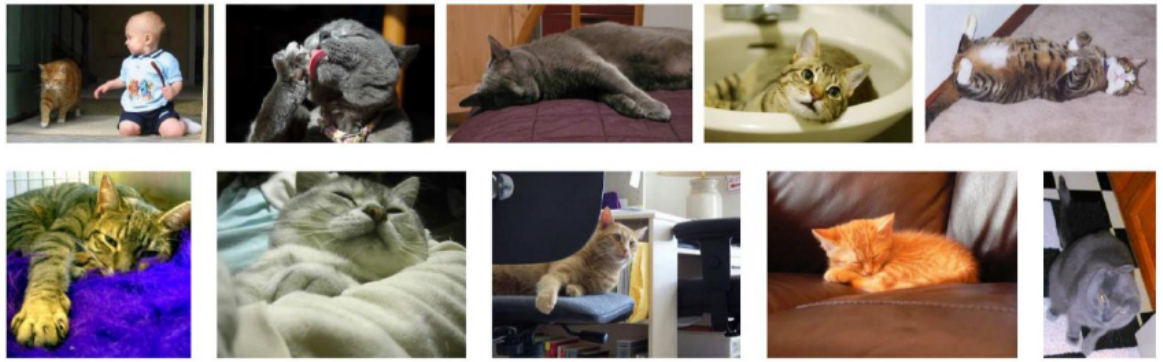



## Illumination Variability

## Occlusion

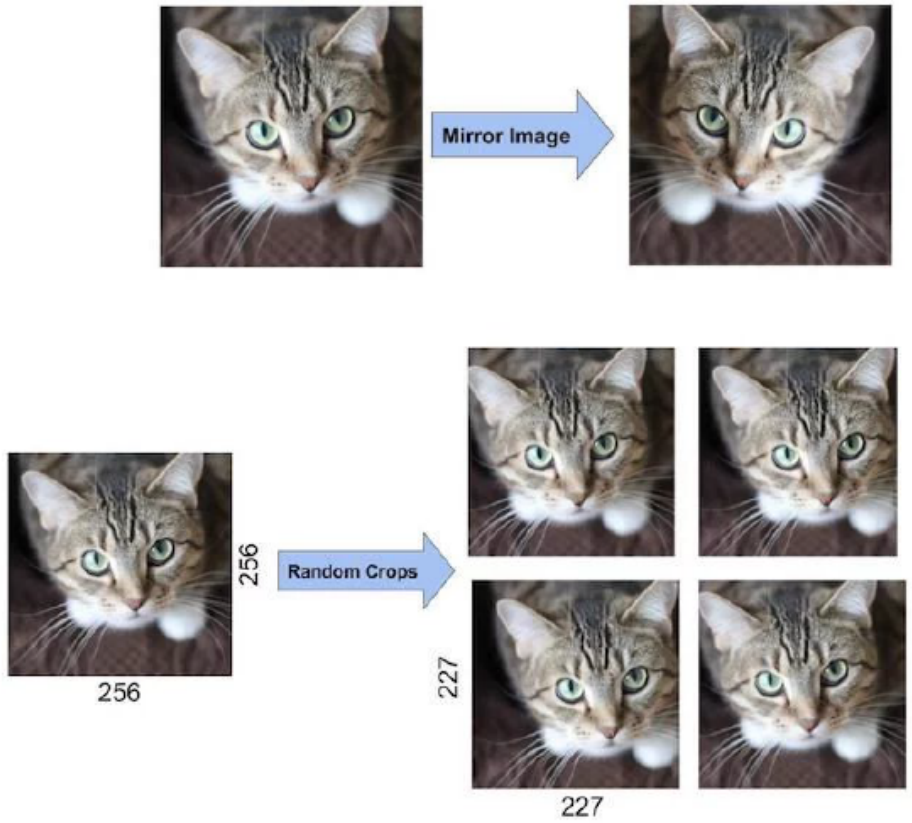


## Pose Variability



# Solution (I): Data augmentation

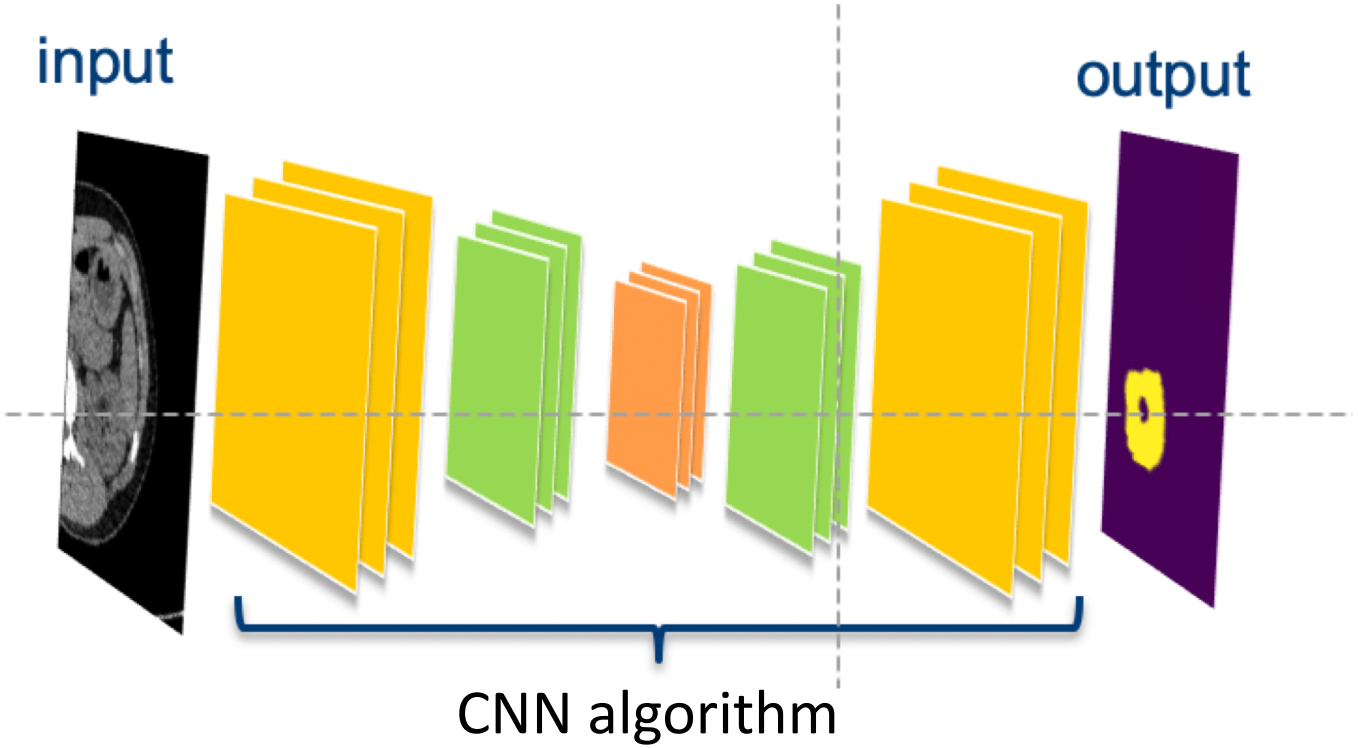
- Data augmentation is used as a way to make models more robust.
- It reduces overfitting by letting models learn harder and more general data representations.
- Most popular:
  - Noise addition
  - Mirroring
  - Rotating
  - Resizing
  - Cropping
  - Occlusion
- Newer methods:
  - Synthetic data

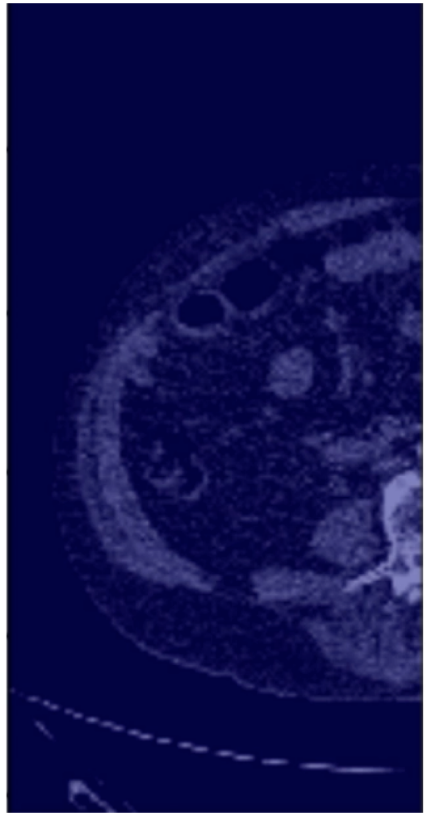


# Solution (II) - Transfer Learning & Fine-Tuning

- Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.
- Start with a model that is pre-trained on a large amount of data for a task.
- Fine-tune some parts of the model to converge faster on a similar/relevant task where data isn’t as abundant.
- Most popular is fine-tuning the feature extractor and adding a new classifier on top.

# Application example – kidney segmentation



Predictions

# Deep Learning  News and new use cases

- OpenAI: https://openai.com/
- Deepmind: https://deepmind.com/blog
- Nvidia: https://blogs.nvidia.com/blog/category/deep-learning
- Towardsdatascience: https://towardsdatascience.com

# Emerging Tech & Research

Reddit News:

- https://www.reddit.com/r/deeplearning/
- https://www.reddit.com/r/MachineLearning/

Research:

- https://distill.pub
- http://www.arxiv-sanity.com/
- https://paperswithcode.com/

# Very good Tutorials

Computer Vision: https://www.pyimagesearch.com/

Mixed: https://machinelearningmastery.com/blog/

# Pretrained Tensorflow models

Tensorflow.keras.applications:

- http://keras.io/applications/
- https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/applications

Tensorflow Hub:

- https://tfhub.dev/
- https://www.tensorflow.org/hub

Tensorflow Model Collection:

- https://github.com/tensorflow/models

There are a lot more repositories

# Hands on exercise

1. Open the following link
   https://drive.google.com/file/d/1TH-gRPnWKFsTi32uTh0JjeOp5iOohXfZ/view?usp=sharing
2. Click "open with Google Colaboratory" , highlighted in red in the image below


3. Go to "File" at the left of the top menu, and click "Save a copy in Drive"
4. Click the  icon at the top left menu
5. Open your copied file "Copy of CNN-intro.ipynb"