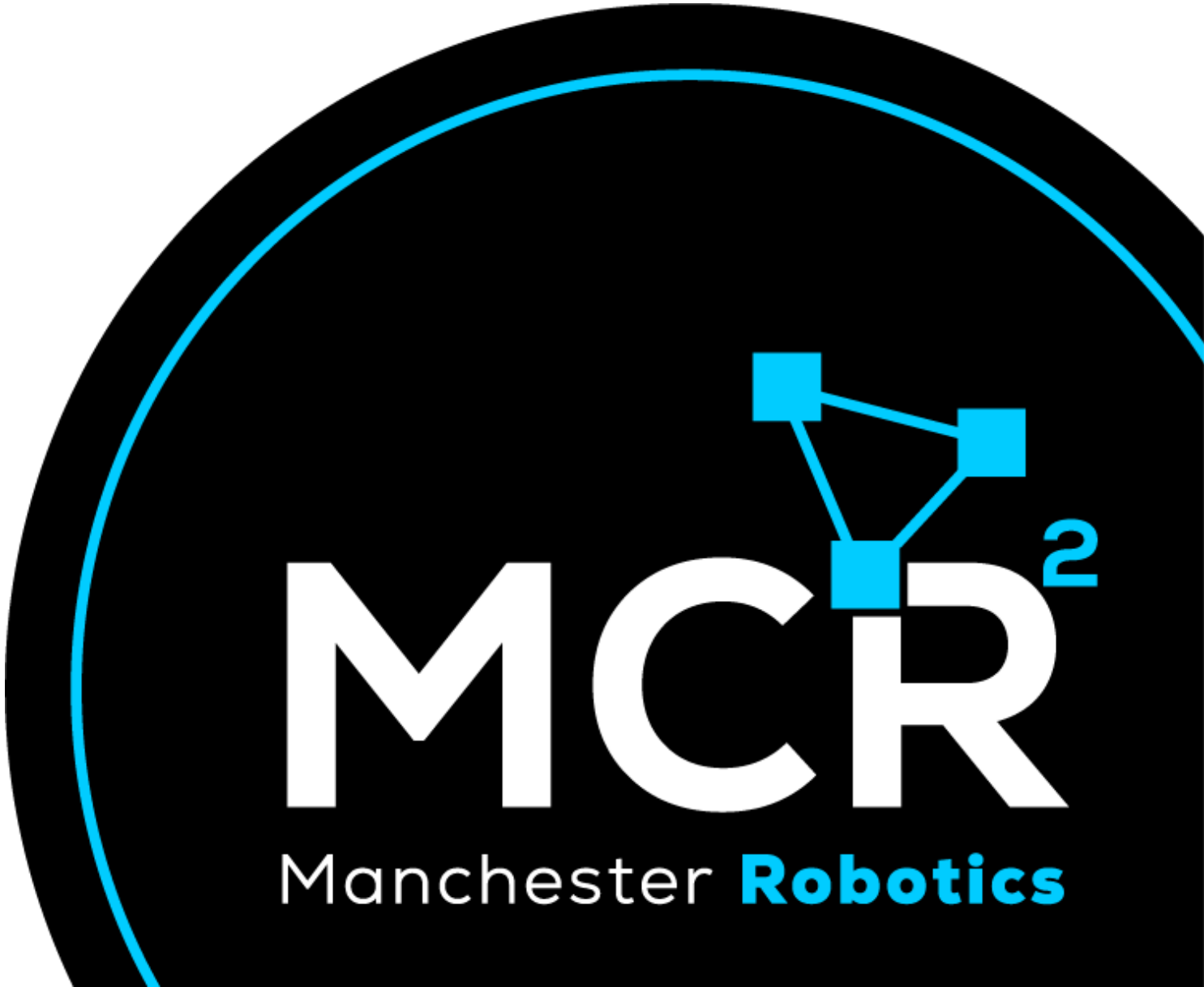
{Learn, Create, Innovate};

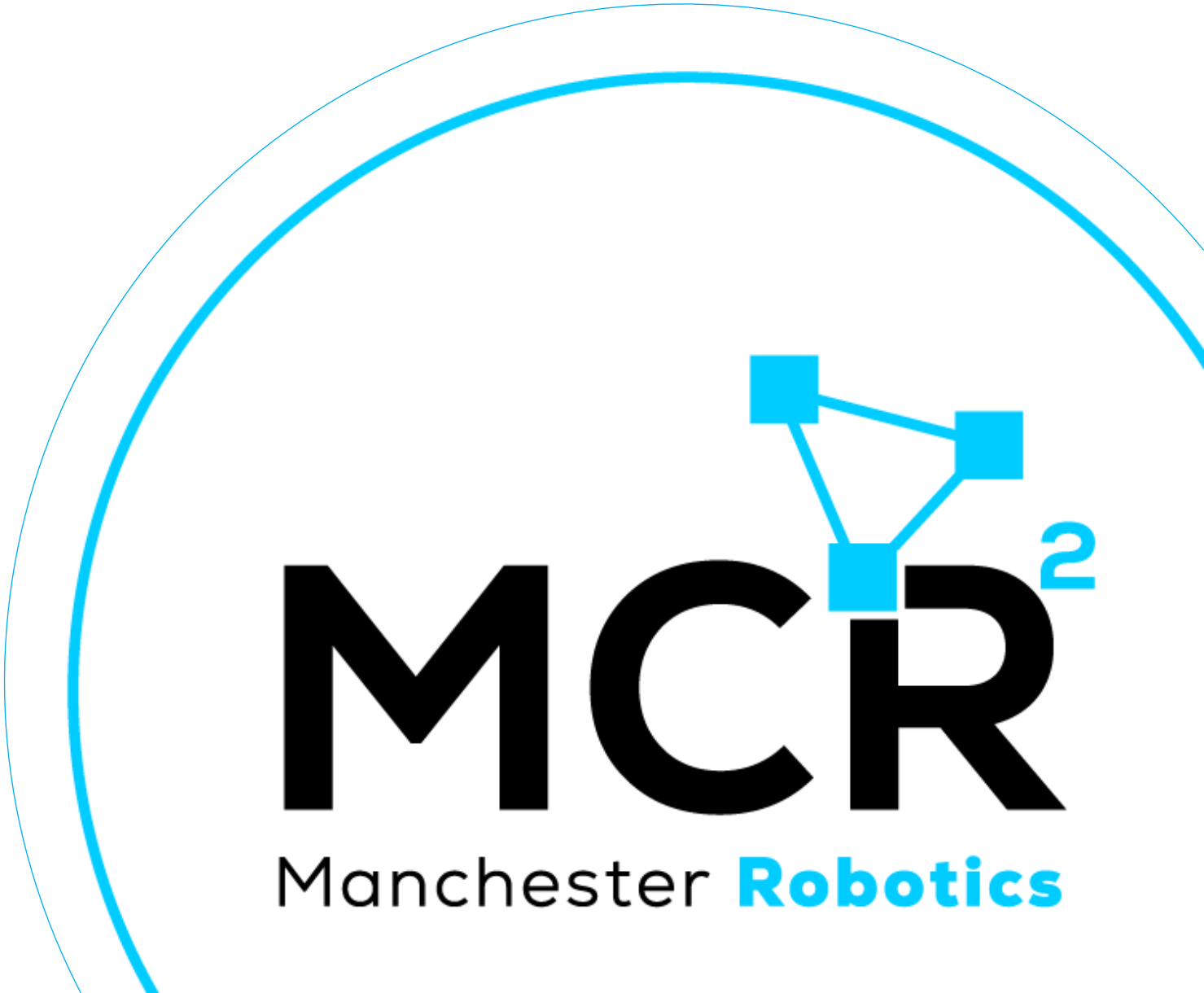
Robot Operating System - ROS

AI in ROS



{Learn, Create, Innovate};

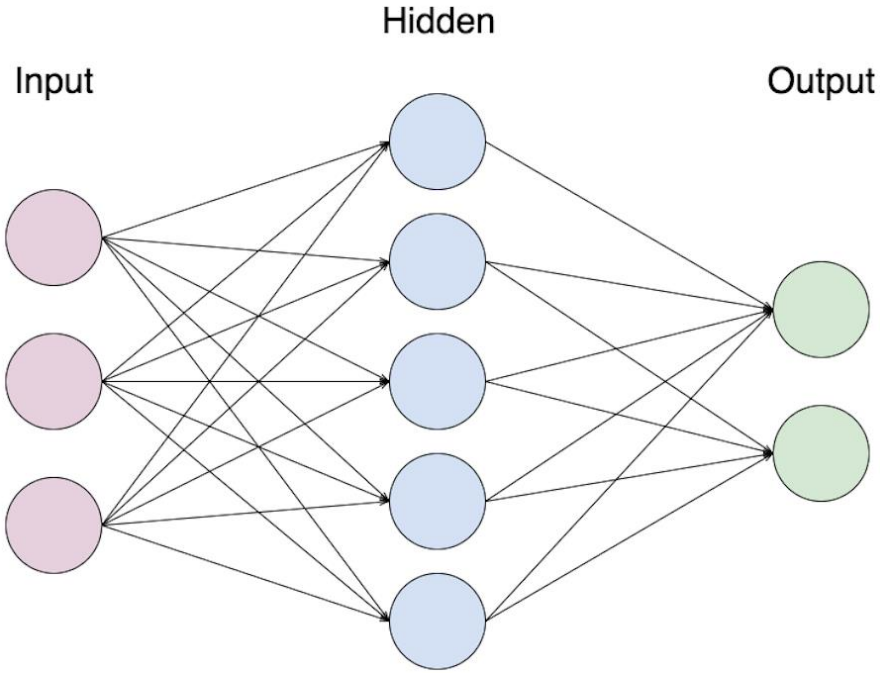
Introduction to neural networks



Introduction to neural networks

What is a neural network?

Set of nodes (neurons) that take the information the user sends through the Input layer, apply some mathematical operations and send them back through the output layer.



Input layer

Neurons directly connected to the user input

Hidden layer

Neurons only connected to other neurons

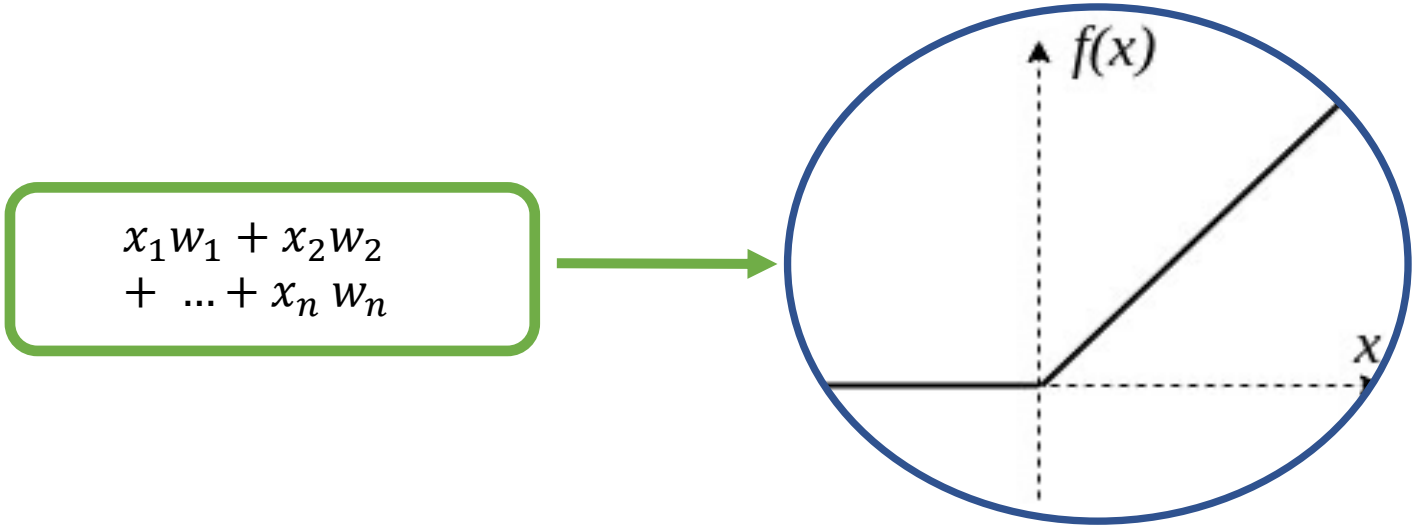
Output layer

Neurons delivering the result

Introduction to neural networks

What is a neuron?

Minimal element of a neural network, it takes as an input a weighted combination of numbers and outputs the result of applying a given function to it, known as activation function.

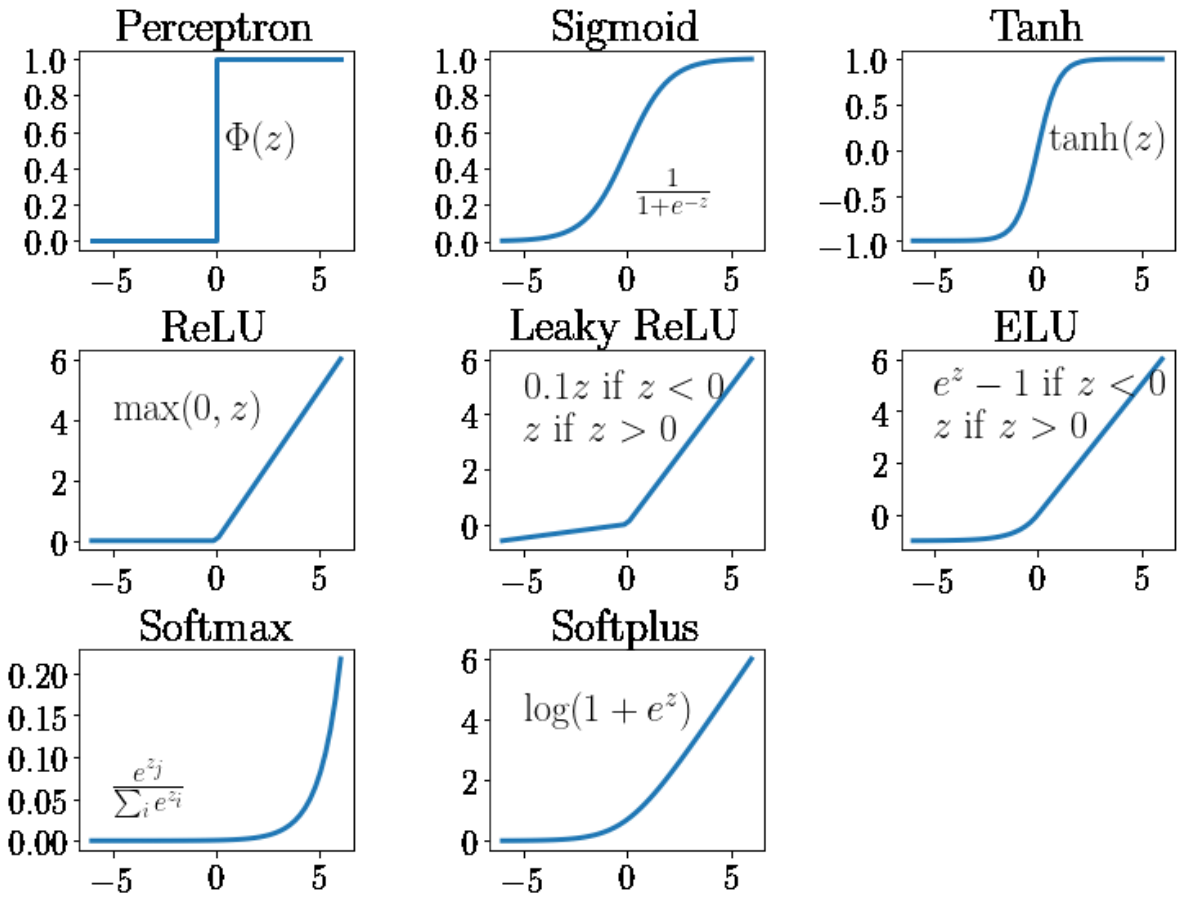


Neuron with a sigmoid activation function

Activation functions

- Output of each neuron.
- They usually range on a -1 to 1 interval
- Linear and non-linear
- Linear
  - Do not deal correctly with high-complexity datasets
- Non-linear
  - Sigmoid – recommended when probability is the output
  - Hyperbolic tangent – classification between two classes
  - Rectified Linear Unit (ReLU) – Most used function

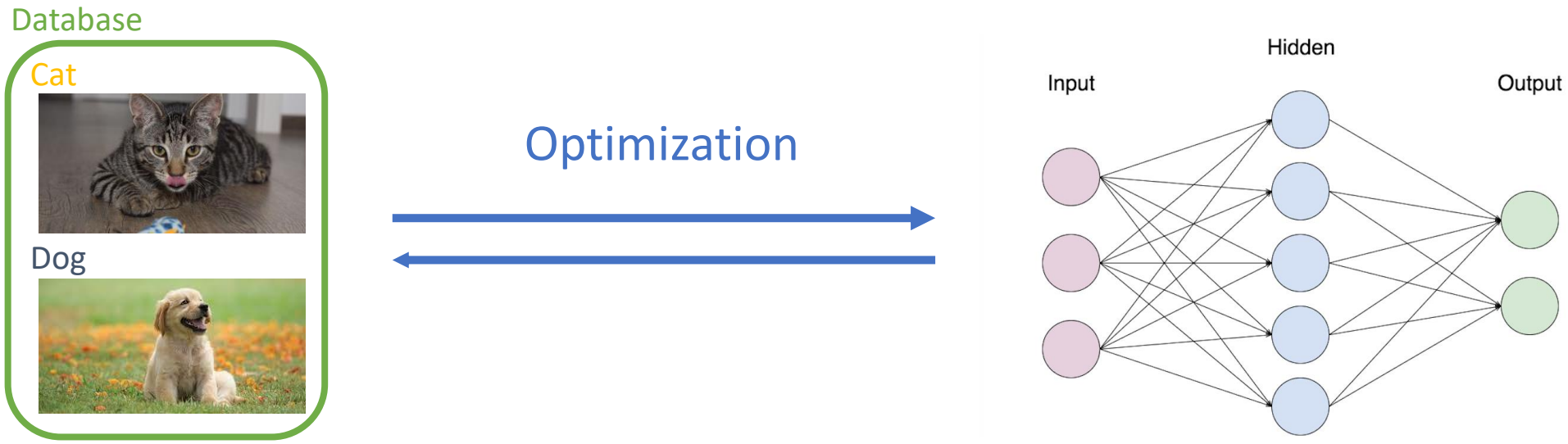
Activation functions



Introduction to neural networks

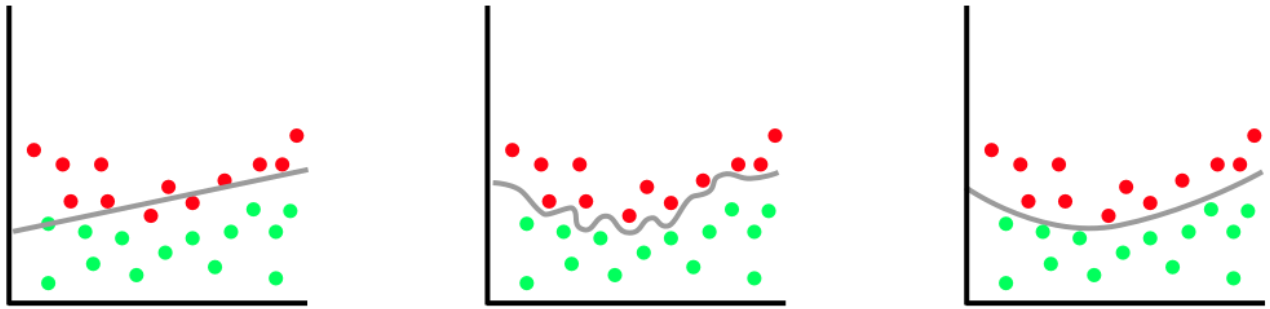
How to set up a network?

- The optimization process involved in training a neural network requires to compute the optimal value for w_i that minimizes some error metric.
- Two sets of data are used to both train and verify that the network works properly.



Introduction to neural networks
Overfitting

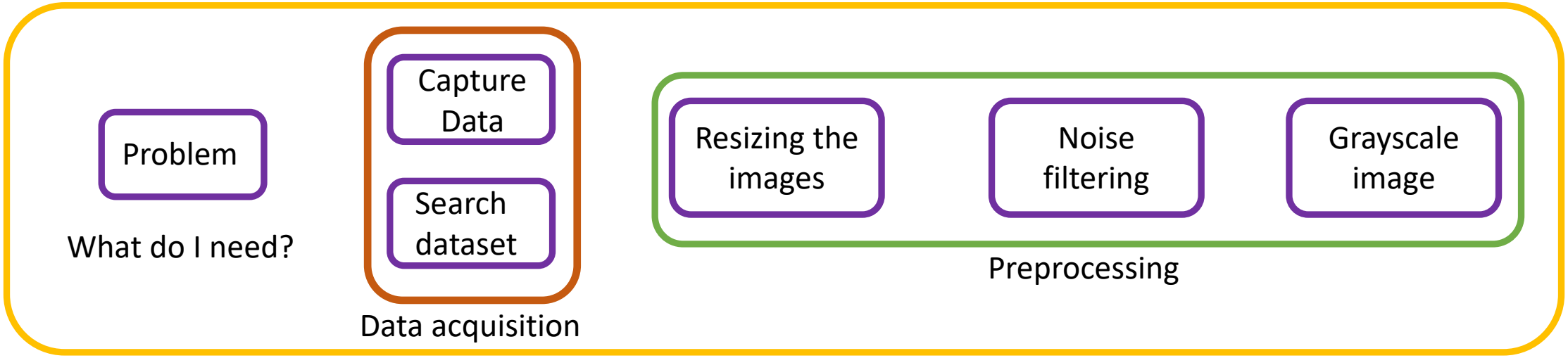
- When the network is too adapted to our data, we say it is *overfitted*.
- This phenomena makes our network too stiff, it's very good working with the data that we used to train it but it's very bad with new samples, making in unusable in application.



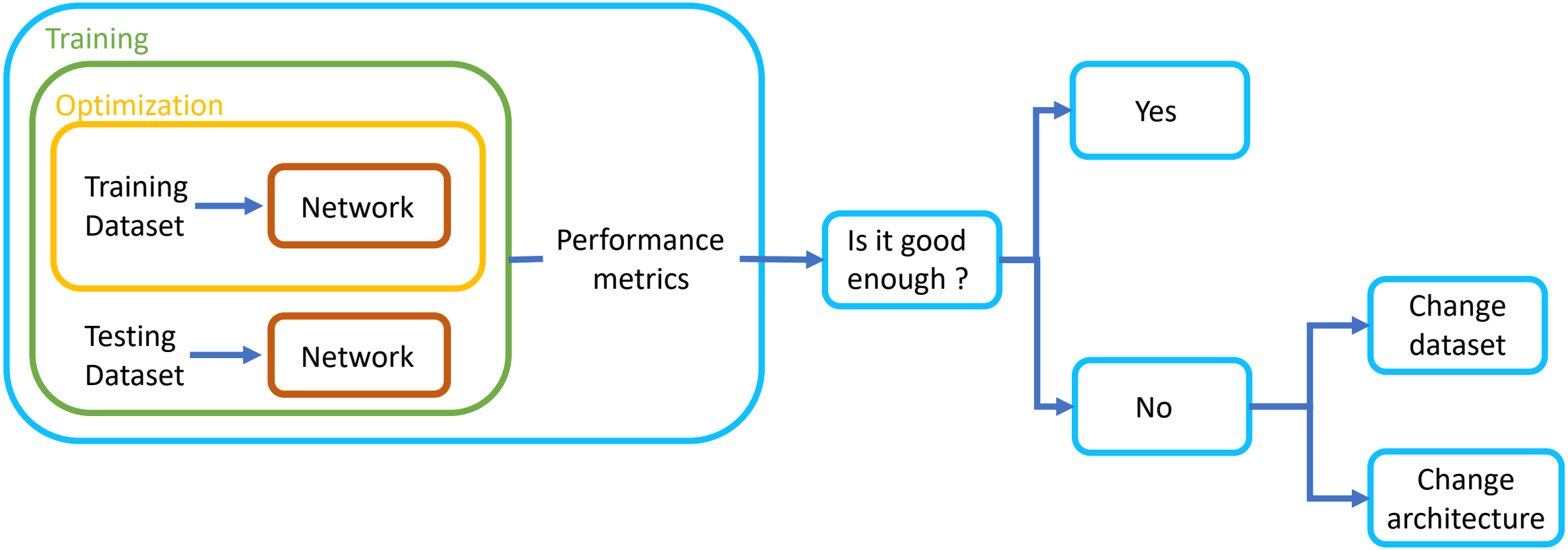
Building an AI solution
Generating a dataset

- When picking a dataset all classes, for example cats and dogs when classifying animals, need to be equally represented to ensure that the network is not biased.
- At the same time, we need to be sure that there are all types of dogs within the dataset to make the solution as general as possible.
- Finally, the data should be normalized to reduce the influence of noise or artifacts in the data used in our application

Building a dataset pipeline



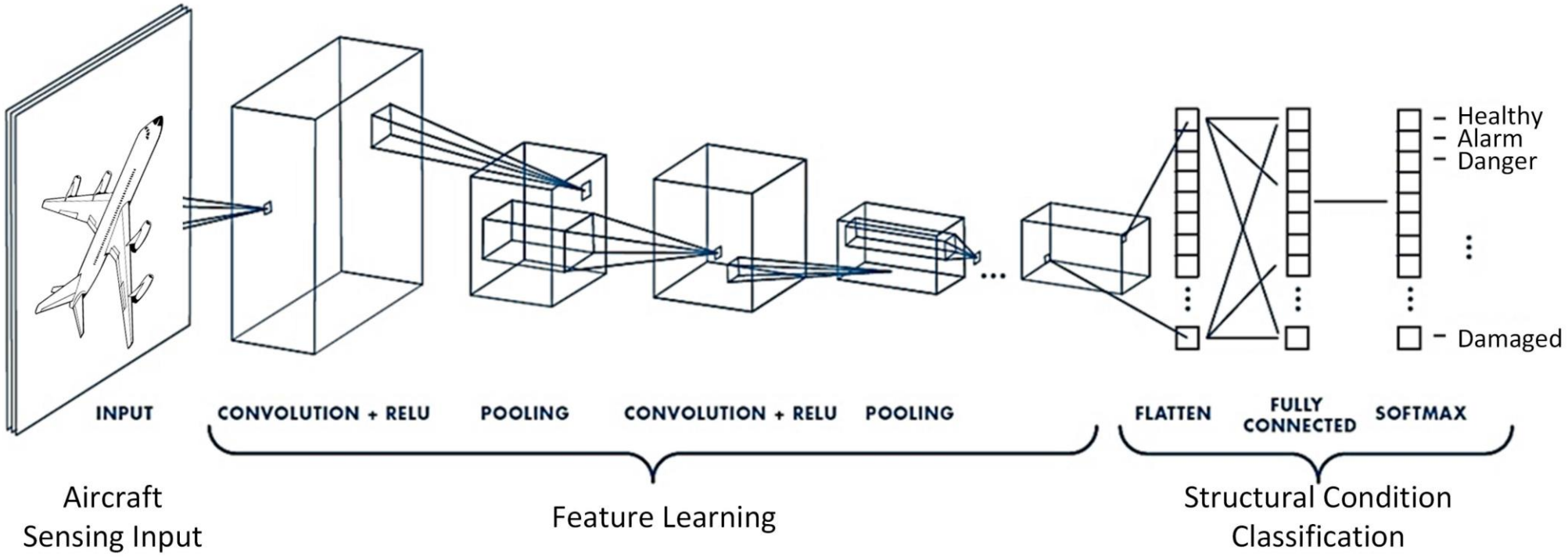
Building an AI solution
Training a neural network



Shape Classification with AI

Problem statement and proposed solution

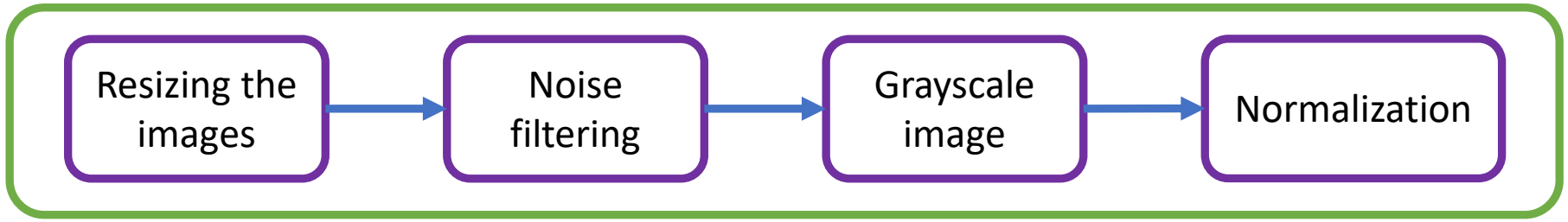
- The most common type of neural networks used with images are known as convolutional neural networks.
- An image is going to be provided and the the extract some features and classify it.



Shape recognition with AI
Preprocessing and regularization

In our example the original images are colored a and have different shapes; in order to process them we need to apply the following procedure.

1. Resizing the images to a common shape that matches the input of the network.
2. Apply histogram equalization to remove artifacts and reduce noise
3. Convert the data to grayscale
4. Normalize the data between 0 and 1



Preprocessing

Shape recognition with AI
Feature extraction

As we don't want to extract any feature manually, we use a convolutional neural network that will learn what are the important parts of the image, leading to a unidimensional vector used to classify the sample using regular neurons.

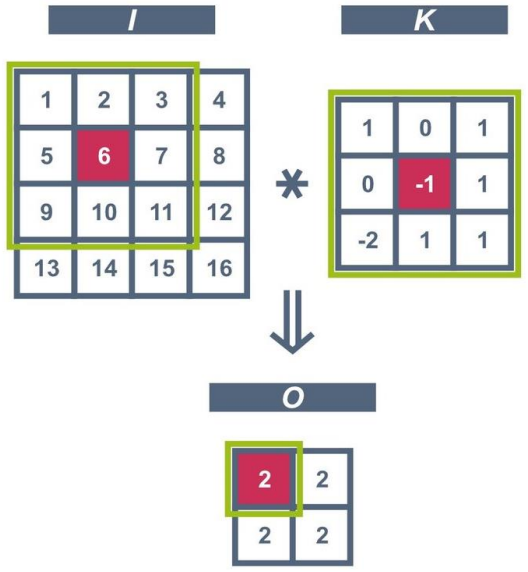
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 60)	1560
conv2d (Conv2D)	(None, 24, 24, 60)	90060
max_pooling2d (MaxPooling2D)	(None, 12, 12, 60)	0
conv2d(Conv2D)	(None, 10, 10, 30)	16230
conv2d (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d (MaxPooling2D)	(None, 4, 4, 30)	0

Convolutional layers used to extract features from the by convoluting patches of pixels together, leading to a more compact representation of the information.

Layer used to downscale the information extracted by the convolutional layers through pooling the maximum value per patch.

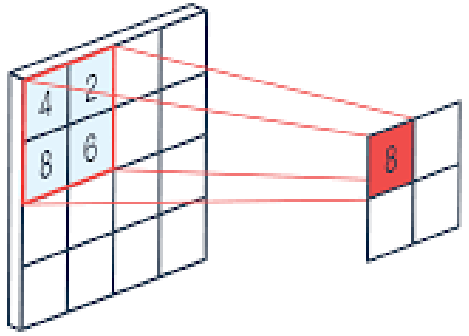
Shape recognition with AI
Convolution and MaxPooling

2D Convolution



The network will learn which convolutional weights need to be applied to extract features.

Max Pooling



A max pooling layer does not have any trainable weights, it acts as an hyperparameter used to reduce the dimensionality after convolutional layers.

Shape recognition with AI
Classification and Output Layers

In our application, we used fully connected layers with a ReLu activation function, and an output layer called Softmax, which computes the degree of certainty for a sample to belong to a certain class.

Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 4, 4, 30)	0
flatten (Flatten)	(None, 480)	0
dense (Dense)	(None, 500)	240500
dropout (Dropout).	(None, 500)	0
dense (Dense).	(None, 43)	21543

This layer sets some random nodes to 0 to prevent overfitting.

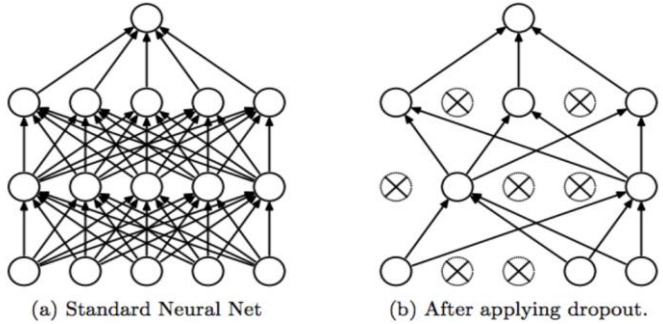
Used to reshape the output of previous layers, taking a unidimensional shape.

Layer of 500 neurons fully connected between each other.

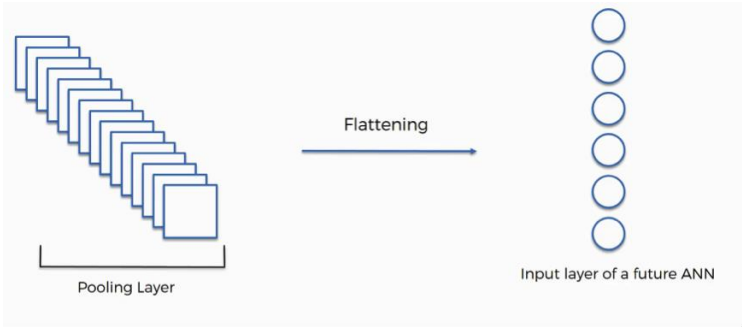
Fully connected layer that computes the likelyhood of the estimation (Softmax).

Shape recognition with AI
Classification and Output Layers

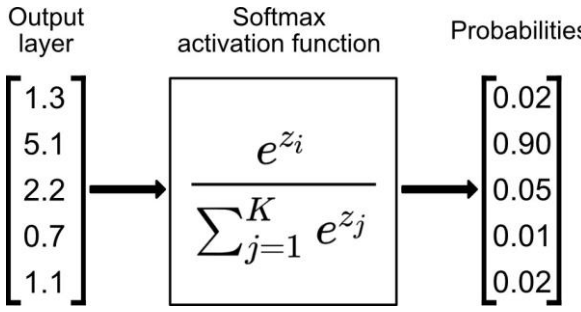
Dropout



Flatten



SoftMax activation function



Shape recognition with AI
Problems and solutions in ML

Computational power and code complexity

Keras and TensorFlow allow to easily create and train your networks.

Google provide powerful computers to scientists so that they can train faster the networks.

Richness of the database

It is very common for machine learning researchers to use public databases available online as a starting point. Furthermore, in the case of having a poor database there are *data augmentation* techniques to manipulate it and generate a bigger ones.

Another alternative is to reuse models from other researchers, as you can continue training them with your own data to make the process faster.

Implementation
Pseudocode

Training

1. Initialization and hyperparameter definition.
    1. Duration of the training
    2. Input and output shape
2. Data regularization
    1. Preprocess the database if needed.
    2. Split the data into training (60%), testing (20%) and validation (20%)
3. Define the structure of the model that you want to train
    1. Add input layer
    2. Add convolution layers and MaxPooling.
    3. Add dense layers and dropout
    4. Add an output layer.
4. Train the model with training and testing sets.
5. Evaluate your model with the validation data

Testing

1. Initialization.
    1. Import model
    2. Import any dependencies needed
2. Configure a data input stream.
3. Preprocess raw data
4. Send the data to your previously trained network.
5. Display the resulting prediction.

Dataset: GTSRB - German Traffic Sign Recognition Benchmark

What does the dataset contain?

- Images cropped to a single traffic sign
- Labels file that maps from integer class labels (0,1,2 …) to strings containing their actual meaning (stop, velocity limit …)
- All images are grouped depending on their class

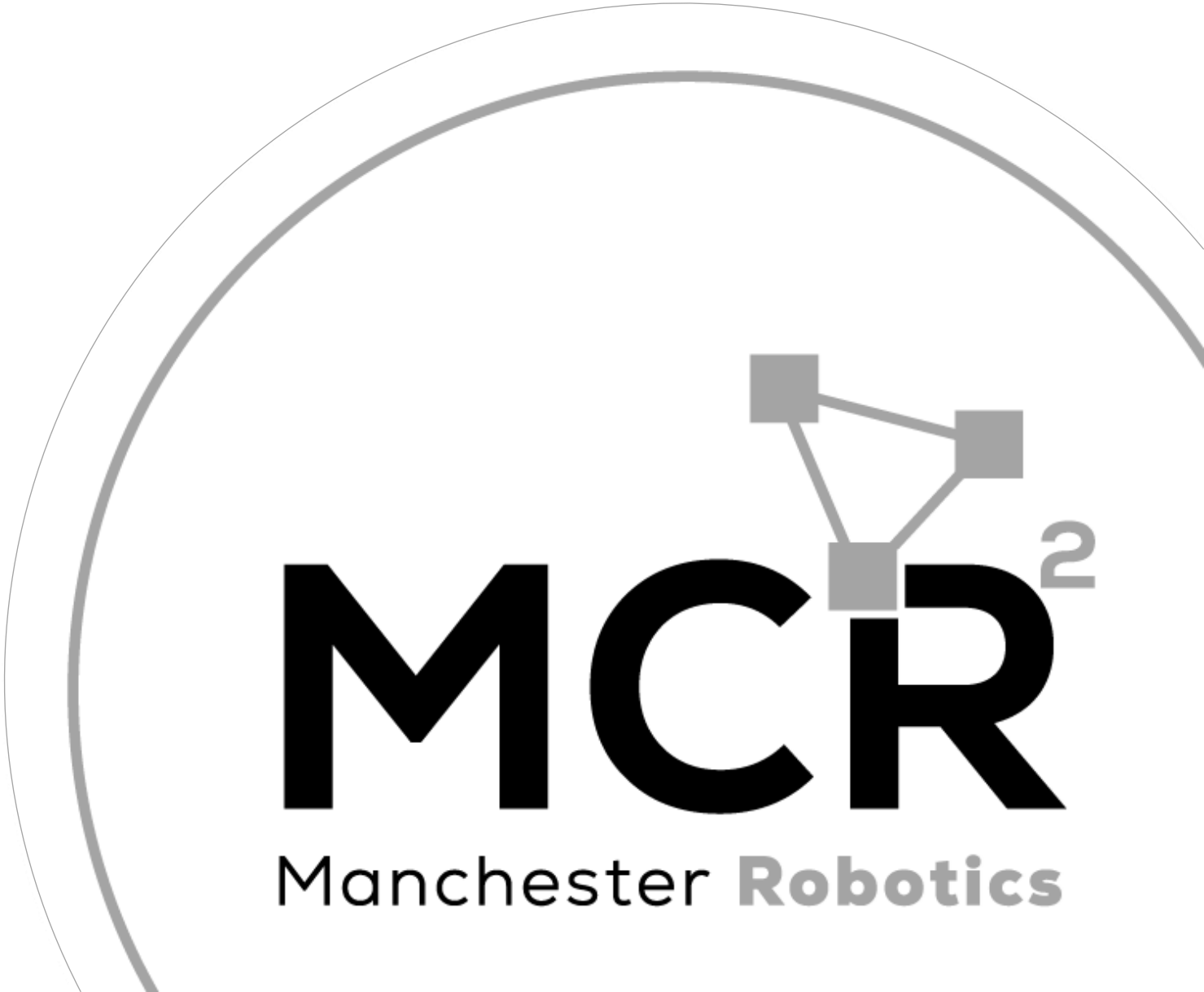
How is it structured?

```
Project
├── MyData
│   ├── 0
│   │   ├── 00000_00000.ppm
│   │   └── GT-00000.csv
│   ├── 41
│   └── 42
└── labels.csv
```

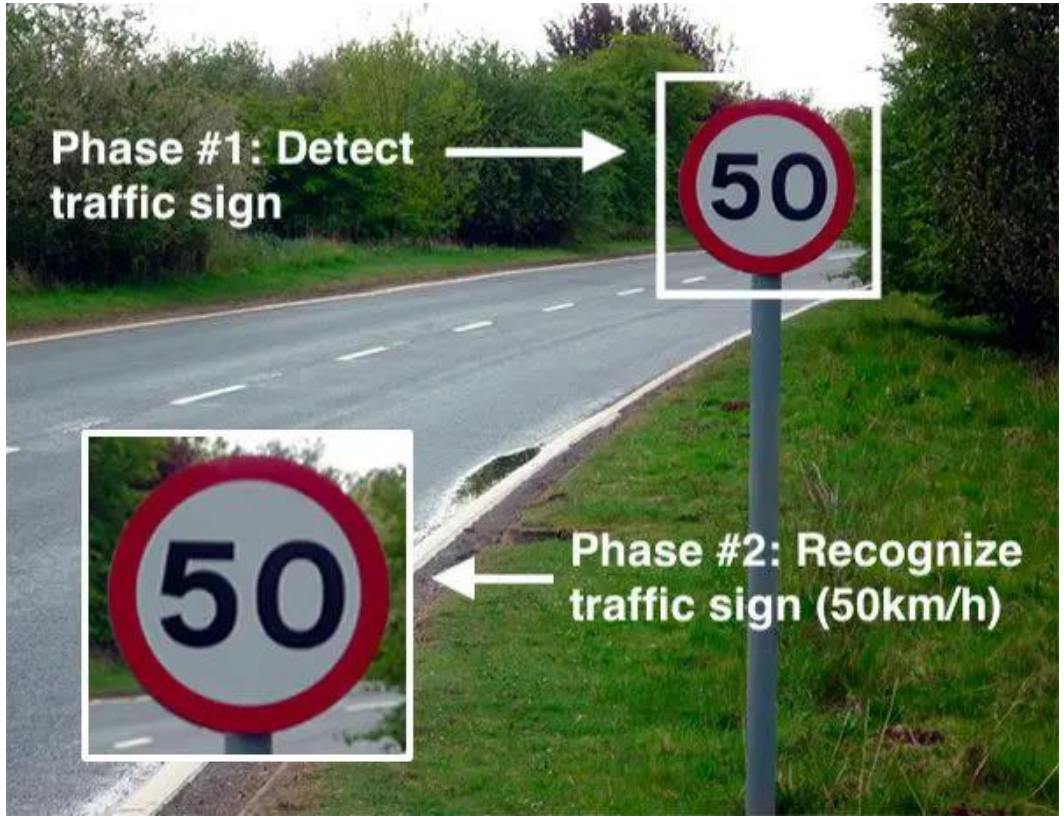
Robot Operating System - ROS

Recognition vs detection

{Learn, Create, Innovate};



Traffic Sign Detection



- So far, we can detect which type. However, in a real-life scenario, we will have a more complex problem due to background noise, and more than one traffic sign in each image. Which requires a higher code complexity

Design Tips

When using the jetson nano we will need to consider the following limitations related to the capabilities of the device.

1. We shouldn't train out networks in the Jetson, as it will take a really LONG time or it might even fail due to the lack of memory.
2. We have to keep the models as small as possible to have a reasonable inference time, also the image size needs to be reduced.
3. We must use CUDA hardware acceleration as it is the main strength of Jetson devices compared with other microcomputers.

Model formatting

Pytorch/Tensorflow

Native formats of the libraries used to train and deploy AI solutions.

TensorFlow

PYTORCH

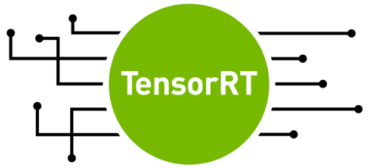
ONNX

Open format built to represent machine learning models used to share networks between devices and framework

ONNX

TensorRT

Nvidia's GPU-only format that optimises the networks for the hardware where it is being deployed



What If the network doesn't suit our needs?

- We have several networks that cover some of the most common problems but are not practical (for example too many classes, not the database we expect to use etc)

Off the shelf

Network	Classes
SSD-Mobilenet-v2	91 (COCO classes)
Detect Net-COCO-Airplane	Airplanes

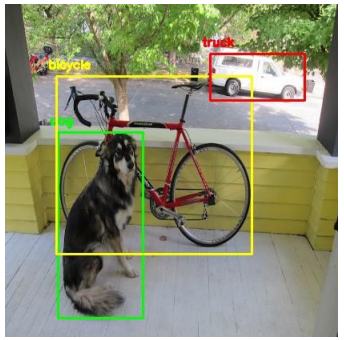
Our case

Up to 39 classes of traffic signs

Transfer learning

”In transfer learning, we first train a base network on a base dataset and task, then repurpose the learned features or transfer them to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, meaning suitable to both base and target tasks, instead of specific to the base task.”

Original training



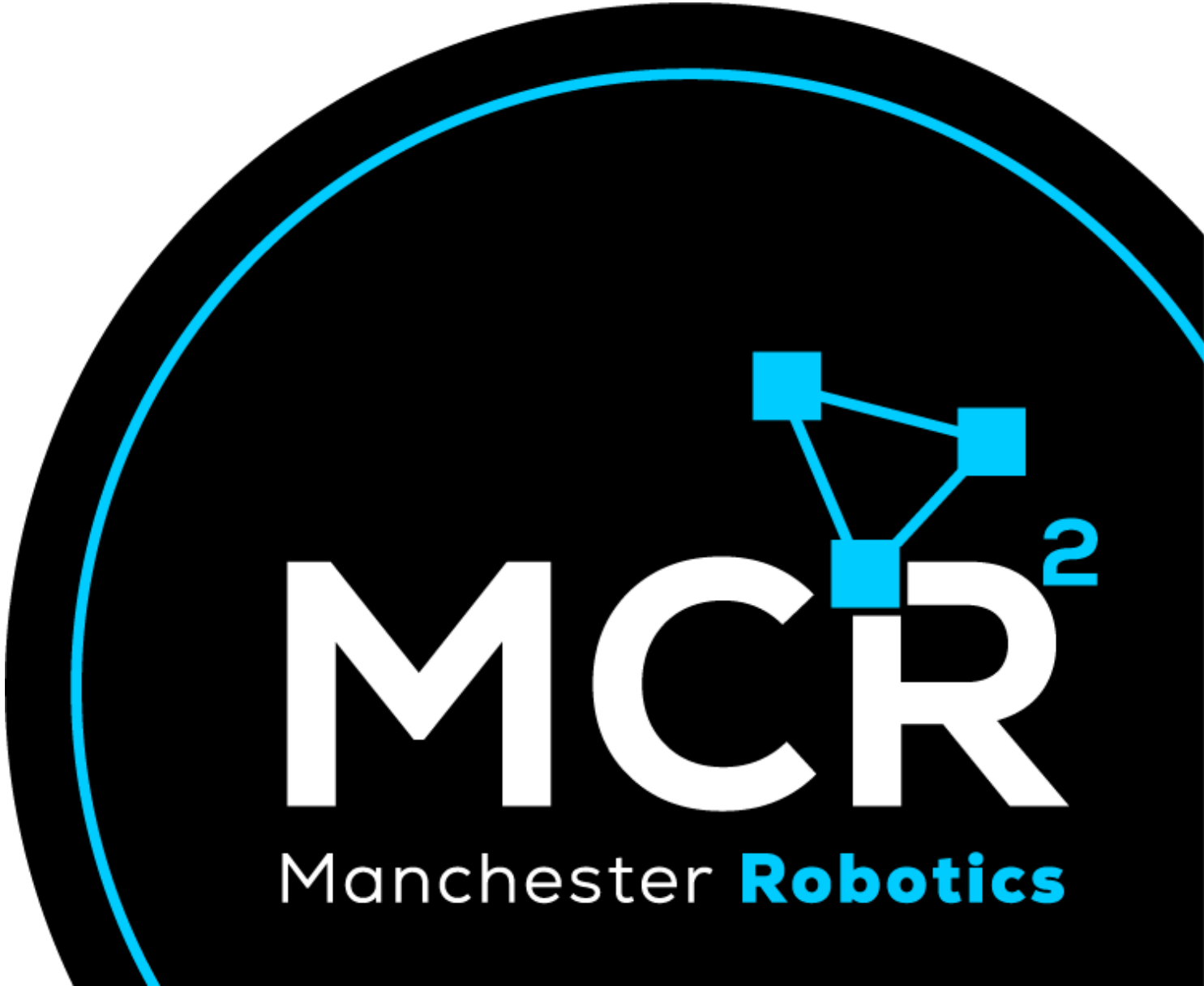
Custom Database

Adapted network

{Learn, Create, Innovate};

YOLO in ROS

Node integration



Object detection networks: YOLO

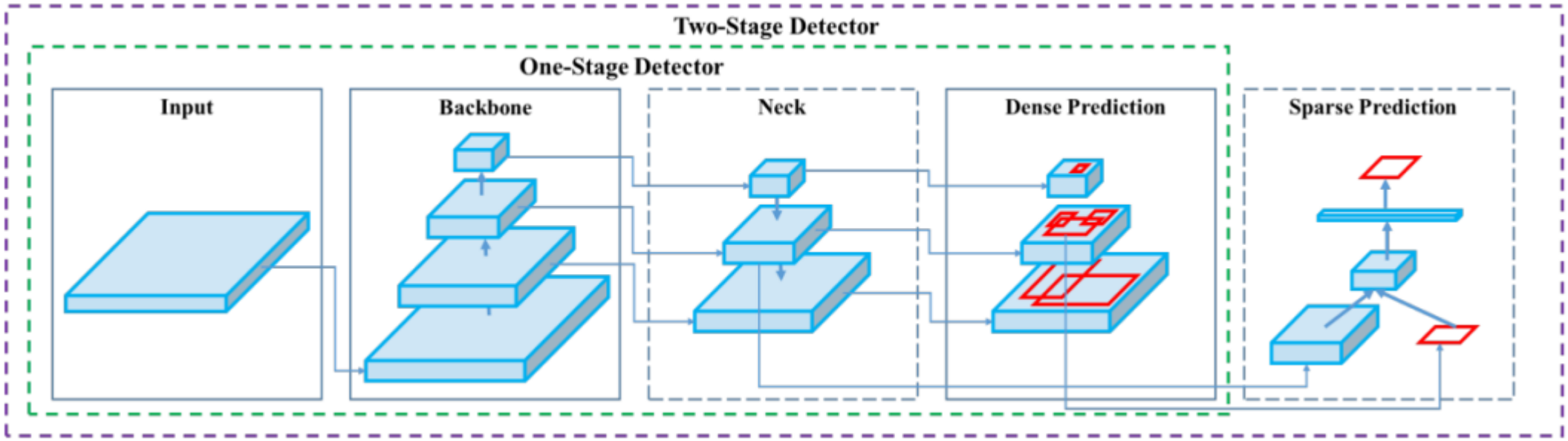
- YOLO (You only look once) is a CNN-based object detector, very popular in the literature, you can find the original paper here.
- The main issue regarding model selection is that we need a very lightweight implementation, which comes as a tradeoff with smaller accuracy in the prediction.
- Extensive documentation and tools to implement your project with YOLO can be found in the ultralitics git hub repo, which has been used as part of the development of this course.
- Guidance and good practices when training your yolo instances can be found here

YOLOv8

- Launched in 2015 by Joseph Redmon and Ali Farhadi of the University of Washington.
- Anchor-free object detection and image segmentation model.
- Single-shot algorithm that classifies an object in a single pass with one NN predicting bounding boxes and class probabilities.

YOLOv8

- Input – Image
- Backbone – Pretrained CNN
- Neck – merge feature maps
- Head – one-stage or dense prediction models



Install YOLO8

- Install the dependencies

```
pip3 install ultralytics
```

- Download the yolobot_recognition and the yolov8_msgs packages.
- Examine the message structures

InferenceResult

```
string class_name
int64 top
int64 left
int64 bottom
int64 right
```

Yolov8Inference

```
std_msgs/Header header
InferenceResult[] yolov8_inference
```

Code

```
from ultralytics import YOLO
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

from yolov8_msgs.msg import InferenceResult
from yolov8_msgs.msg import Yolov8Inference

bridge = CvBridge()

class Camera_subscriber(Node):

    def __init__(self):
        super().__init__('camera_subscriber')

        self.model = YOLO('~/ros2_ws/src/yolobot_recognition/scripts/yolov8n.pt')

        self.yolov8_inference = Yolov8Inference()

        self.subscription = self.create_subscription(
            Image,
            '/image_raw',
            self.camera_callback,
            10)
        self.subscription

        self.yolov8_pub = self.create_publisher(Yolov8Inference, "/Yolov8_Inference", 1)
        self.img_pub = self.create_publisher(Image, "/inference_result", 1)
```

Code

```
    def camera_callback(self, data):
        img = bridge.imgmsg_to_cv2(data, "bgr8")
        results = self.model(img)
        self.yolov8_inference.header.frame_id = "inference"
        self.yolov8_inference.header.stamp = camera_subscriber.get_clock().now().to_msg()

        for r in results:
            boxes = r.boxes
            for box in boxes:
                self.inference_result = InferenceResult()
                b = box.xyxy[0].to('cpu').detach().numpy().copy()  # get box coordinates in (top, left, bottom, right) format
                c = box.cls
                self.inference_result.class_name = self.model.names[int(c)]
                self.inference_result.top = int(b[0])
                self.inference_result.left = int(b[1])
                self.inference_result.bottom = int(b[2])
                self.inference_result.right = int(b[3])
                self.yolov8_inference.yolov8_inference.append(self.inference_result)

        annotated_frame = results[0].plot()
        img_msg = bridge.cv2_to_imgmsg(annotated_frame, encoding='bgr8')

        self.img_pub.publish(img_msg)
        self.yolov8_pub.publish(self.yolov8_inference)
        self.yolov8_inference.yolov8_inference.clear()

if __name__ == '__main__':
    rclpy.init(args=None)
    camera_subscriber = Camera_subscriber()
    rclpy.spin(camera_subscriber)
    rclpy.shutdown()
```

Run the code

- Run the camera node

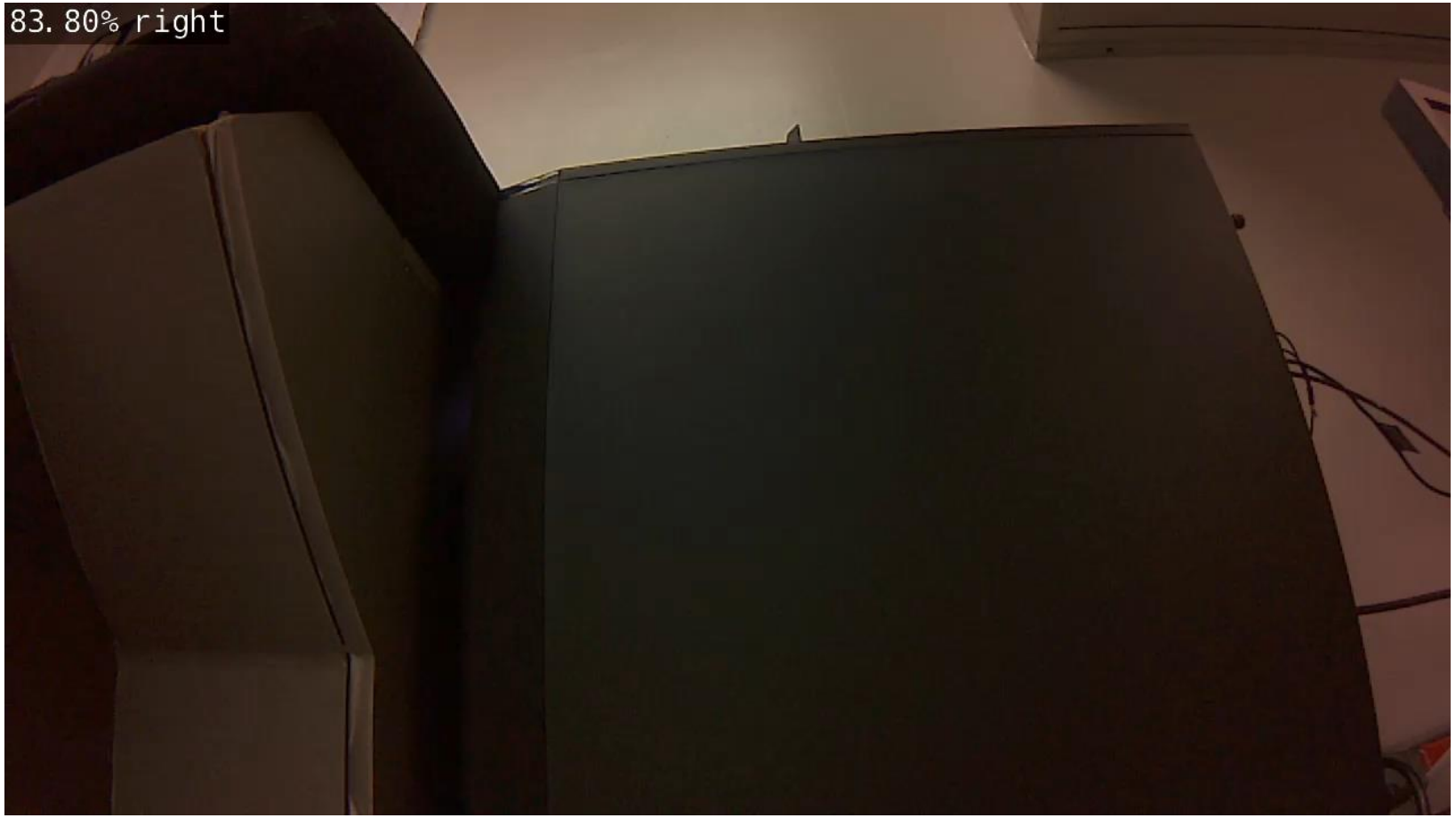
```
$ ros2 run usb_cam usb_cam_node_exe
```

- Build the workspace, source it, and launch the file

```
cd ~/ros2_ws/
colcon build
source install/setup.bash
$ ros2 launch yolobot_recognition launch_yolov8.launch.py
```

- Open the rqt_image_view application and verify your work

```
$ ros2 run rqt_image_view rqt_image_view
```


83.80% right

Thank You

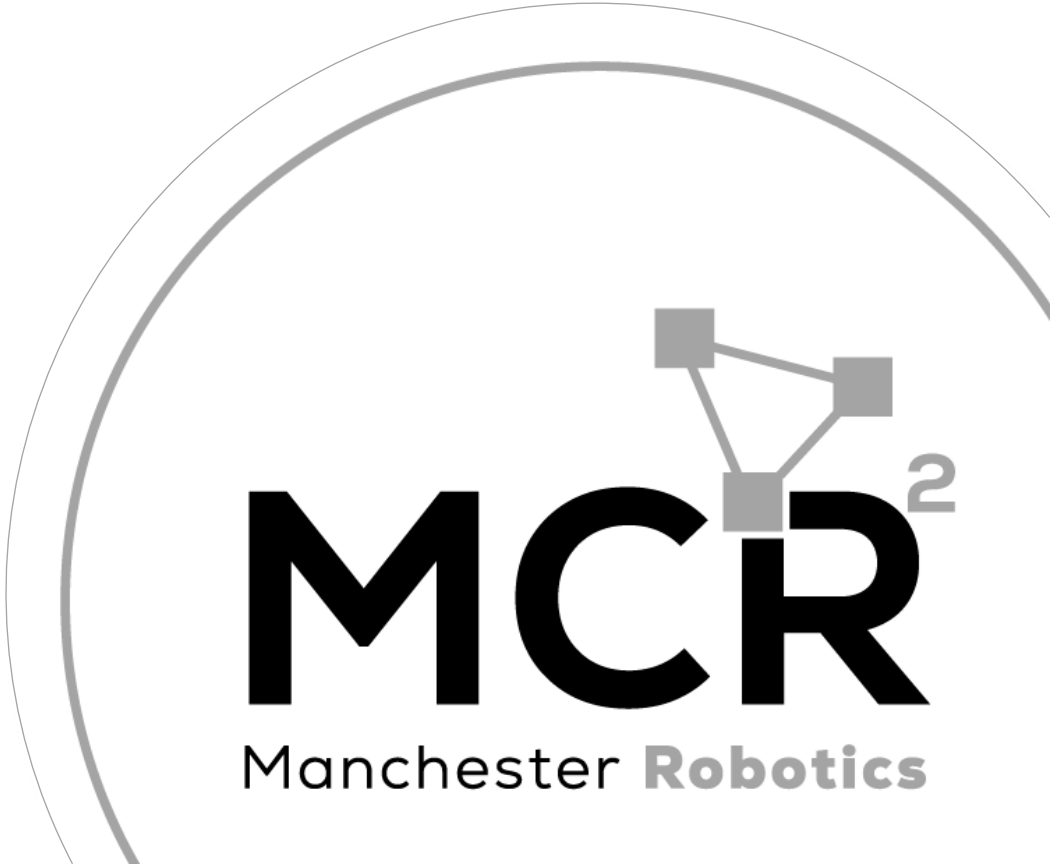
Robotics For Everyone

{Learn, Create, Innovate};

T&C

Terms and conditions

{Learn, Create, Innovate};



Terms and conditions

- *THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.*
- *THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.*
- *WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.*