## YOLOv3 architecture

## How does YOLO work?

Prior detection systems use localizers or classifiers to carry out the detection process. Then the model is applied to an image at different scales and locations. The regions of the image with High scoring are considered for detections.

YOLO algorithm uses a completely different approach. The algorithm applies a single neural network to the entire full image. Then this network divides that image into regions which provides the bounding boxes and also predicts probabilities for each region. These generated bounding boxes are weighted by the predicted probabilities.



## Architecture

## 

YOLO is a convolution neural network. It consists of a total of 24 convolutional layers and followed  
by 2 fully connected layers. Each layer has its own importance and the layers are separated by their  
functionality.

— The First 20 convolutional layers followed by an average pooling layer and a fully connected layer is pre-trained on the ImageNet dataset which is a 1000-class classification dataset.

— The pretraining for classification is performed on the dataset with the image resolution of 224 x 224×3.

— The layers comprise 3×3 convolutional layers and1x1 reduction layers.

— For object detection, in the end, the last 4 convolutional layers followed by 2 fully connected layers are added to train the network.

— Object detection requires more precise detail hence the resolution of the dataset is increased to 448 x 448

— Then the final layer predicts the class probabilities and bounding boxes.

— All the other convolutional layers use leaky ReLU activation whereas the final layer uses a linear activation.

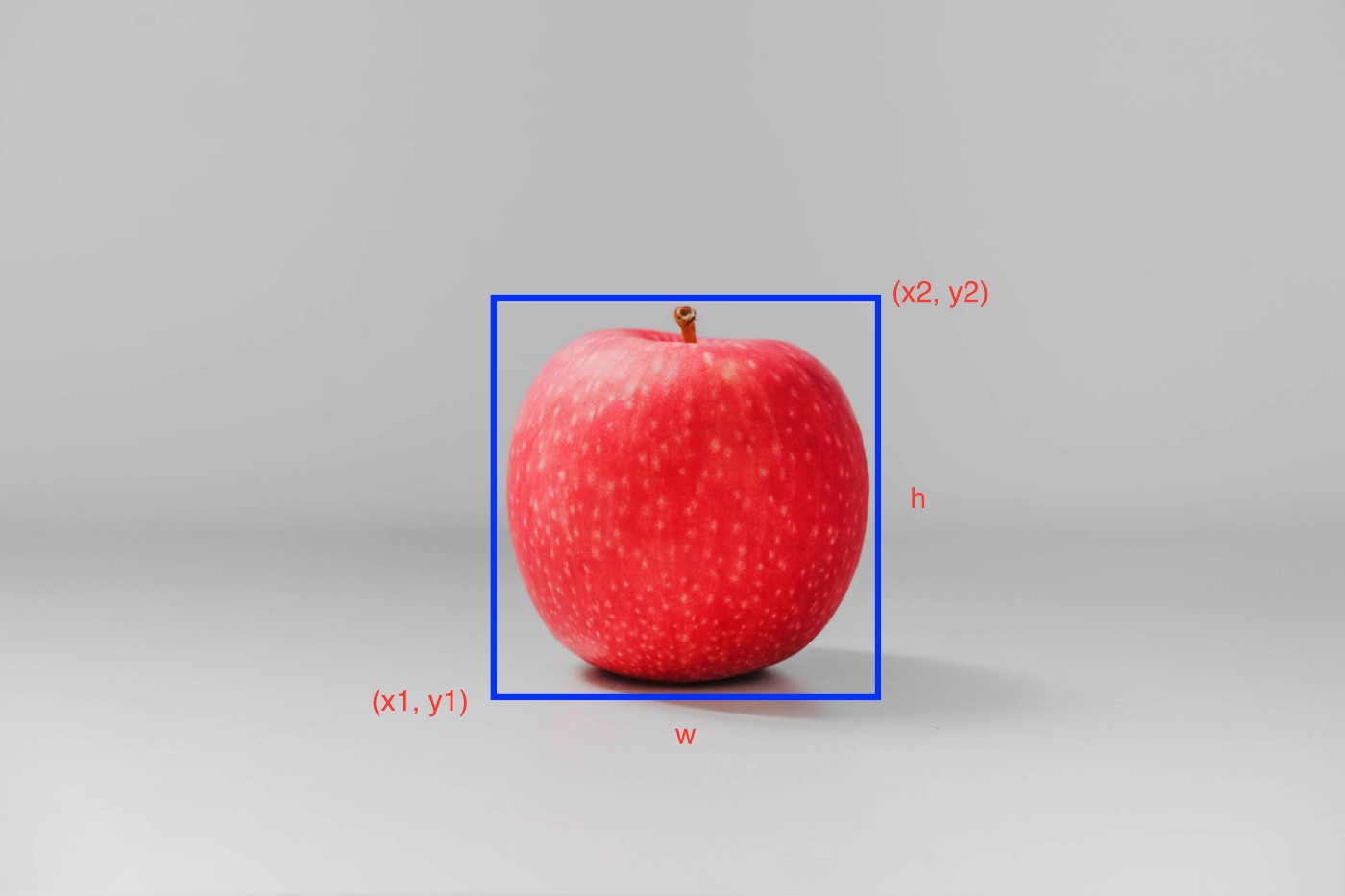
— The input is of 448 x 448 image and the output is the class prediction of the detected object enclosed in the bounding box.

**Dataset**

Dataset contains images with and without mak and animals and cars and a lot of objects. Dataset is already split into 3 sets for convenience. Images are already annotated in yolo format.  
Dataset was gathered using google images, bing and other kaggle datasets,

In YOLO labeling format, a . txt file with the same name is created for each image file in the samedirectory. Each . txt file contains the annotations for the corresponding image file, that is object class, object coordinates, height and width,

Each image we have it’s bounding box, bounding boxes are usually represented by either two coordinates (x1, y1) and (x2, y2) or by one co-ordinate (x1, y1) and width (w) and height (h) of the bounding box. (See image below)



**Implementation details**

Implemented code:

class Yolo\_v3:

"""Yolo v3 model class."""

def \_\_init\_\_(self, n\_classes, model\_size, max\_output\_size, iou\_threshold,

confidence\_threshold, data\_format=None):

"""Creates the model.

Args:

n\_classes: Number of class labels.

model\_size: The input size of the model.

max\_output\_size: Max number of boxes to be selected for each class.

iou\_threshold: Threshold for the IOU.

confidence\_threshold: Threshold for the confidence score.

data\_format: The input format.

Returns:

None.

"""

if not data\_format:

if tf.test.is\_built\_with\_cuda():

data\_format = 'channels\_first'

else:

data\_format = 'channels\_last'

self.n\_classes = n\_classes

self.model\_size = model\_size

self.max\_output\_size = max\_output\_size

self.iou\_threshold = iou\_threshold

self.confidence\_threshold = confidence\_threshold

self.data\_format = data\_format

def \_\_call\_\_(self, inputs, training):

"""Add operations to detect boxes for a batch of input images.

Args:

inputs: A Tensor representing a batch of input images.

training: A boolean, whether to use in training or inference mode.

Returns:

A list containing class-to-boxes dictionaries

for each sample in the batch.

"""

with tf.compat.v1.variable\_scope('yolo\_v3\_model'):

if self.data\_format == 'channels\_first':

inputs = tf.transpose(inputs, [0, 3, 1, 2])

inputs = inputs / 255

route1, route2, inputs = darknet53(inputs, training=training,

data\_format=self.data\_format)

route, inputs = yolo\_convolution\_block(

inputs, filters=512, training=training,

data\_format=self.data\_format)

detect1 = yolo\_layer(inputs, n\_classes=self.n\_classes,

anchors=\_ANCHORS[6:9],

img\_size=self.model\_size,

data\_format=self.data\_format)

inputs = conv2d\_fixed\_padding(route, filters=256, kernel\_size=1,

data\_format=self.data\_format)

inputs = batch\_norm(inputs, training=training,

data\_format=self.data\_format)

inputs = tf.nn.leaky\_relu(inputs, alpha=\_LEAKY\_RELU)

upsample\_size = route2.get\_shape().as\_list()

inputs = upsample(inputs, out\_shape=upsample\_size,

data\_format=self.data\_format)

axis = 1 if self.data\_format == 'channels\_first' else 3

inputs = tf.concat([inputs, route2], axis=axis)

route, inputs = yolo\_convolution\_block(

inputs, filters=256, training=training,

data\_format=self.data\_format)

detect2 = yolo\_layer(inputs, n\_classes=self.n\_classes,

anchors=\_ANCHORS[3:6],

img\_size=self.model\_size,

data\_format=self.data\_format)

inputs = conv2d\_fixed\_padding(route, filters=128, kernel\_size=1,

data\_format=self.data\_format)

inputs = batch\_norm(inputs, training=training,

data\_format=self.data\_format)

inputs = tf.nn.leaky\_relu(inputs, alpha=\_LEAKY\_RELU)

upsample\_size = route1.get\_shape().as\_list()

inputs = upsample(inputs, out\_shape=upsample\_size,

data\_format=self.data\_format)

inputs = tf.concat([inputs, route1], axis=axis)

route, inputs = yolo\_convolution\_block(

inputs, filters=128, training=training,

data\_format=self.data\_format)

detect3 = yolo\_layer(inputs, n\_classes=self.n\_classes,

anchors=\_ANCHORS[0:3],

img\_size=self.model\_size,

data\_format=self.data\_format)

inputs = tf.concat([detect1, detect2, detect3], axis=1)

inputs = build\_boxes(inputs)

boxes\_dicts = non\_max\_suppression(

inputs, n\_classes=self.n\_classes,

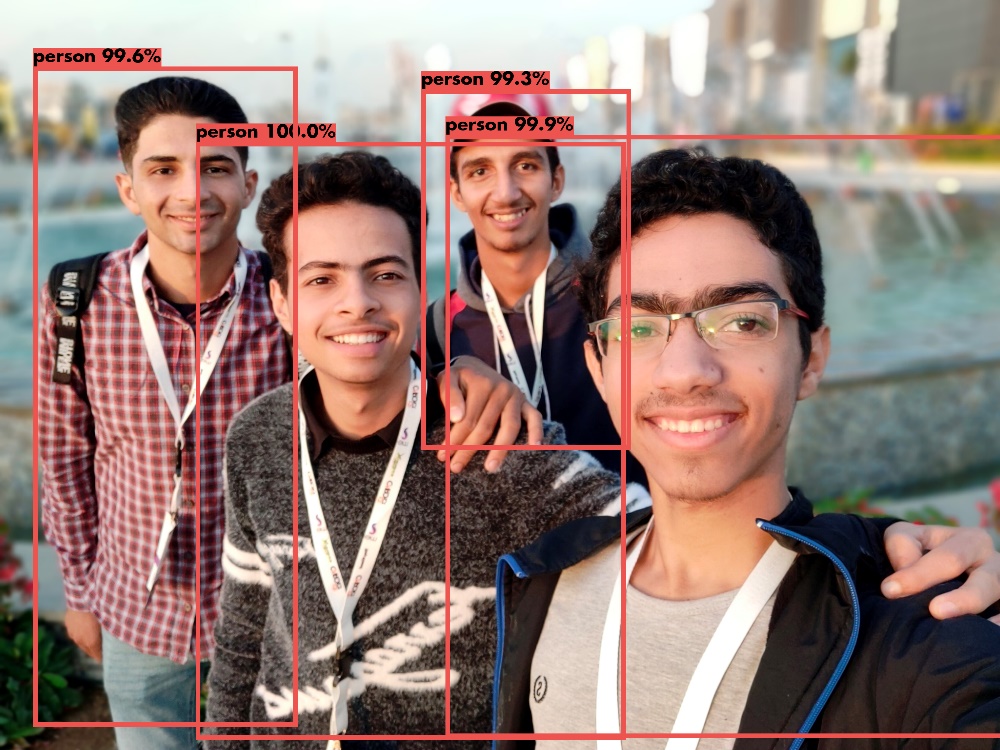
max\_output\_size=self.max\_output\_size,

iou\_threshold=self.iou\_threshold,

confidence\_threshold=self.confidence\_threshold)

return boxes\_dicts

**Results and visualizations**

****

