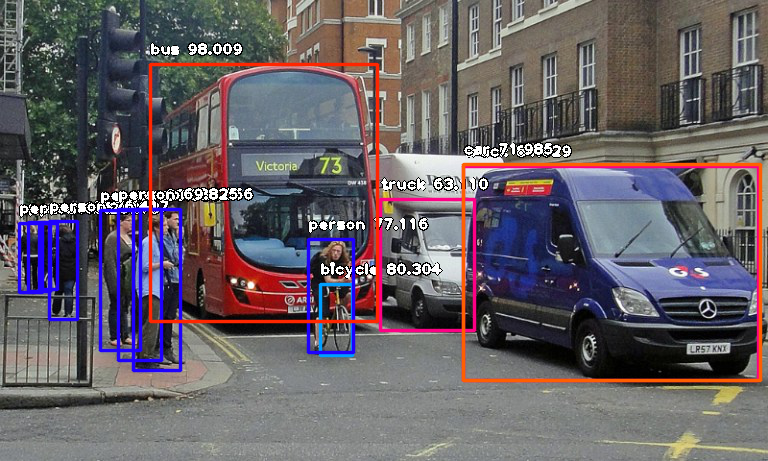
**Detection Classes**



**ImageAI** provides very powerful yet easy to use classes and functions to perform **Image Object Detection and Extraction**.

**ImageAI** allows you to perform all of these with state-of-the-art deep learning algorithms like **RetinaNet**, **YOLOv3** and **TinyYOLOv3**. With **ImageAI** you can run detection tasks and analyse images.

Find below the classes and their respective functions available for you to use. These classes can be integrated into any traditional python program you are developing, be it a website, Windows/Linux/MacOS application or a system that supports or part of a Local-Area-Network.

**======= imageai.Detection.ObjectDetection =======**

This **ObjectDetection** class provides you function to perform object detection on any image or set of images, using **pre-trained** models that was trained on the **COCO** dataset. The models supported are **RetinaNet**, **YOLOv3** and **TinyYOLOv3**. This means you can detect and recognize 80 different kind of common everyday objects. To get started, download any of the **pre-trained model** that you want to use via the links below.

[Download RetinaNet Model - resnet50\_coco\_best\_v2.0.1.h5](https://github.com/OlafenwaMoses/ImageAI/releases/tag/1.0/)

[Download YOLOv3 Model - yolo.h5](https://github.com/OlafenwaMoses/ImageAI/releases/tag/1.0/)

[Download TinyYOLOv3 Model - yolo-tiny.h5](https://github.com/OlafenwaMoses/ImageAI/releases/tag/1.0/)

Once you have downloaded the model of your choice, you should create a new instance of the **ObjectDetection** class as seen in the sample below:

**from** **imageai.Detection** **import** ObjectDetection

detector = ObjectDetection()

Once you have created an instance of the class, you can use the functions below to set your instance property and start detecting objects in images.

* **.setModelTypeAsRetinaNet()** , This function sets the model type of the object detection instance you created to the **RetinaNet** model, which means you will be performing your **object detection tasks using the pre-trained “RetinaNet” model you downloaded from the links above**. Find example code below:
* detector.setModelTypeAsRetinaNet()
* **.setModelTypeAsYOLOv3()** , This function sets the model type of the object detection instance you created to the **YOLOv3** model, which means you will be performing your **object detection tasks using the pre-trained “YOLOv3” model you downloaded from the links above**. Find example code below:
* detector.setModelTypeAsYOLOv3()
* **.setModelTypeAsTinyYOLOv3()** , This function sets the model type of the object detection instance you created to the **TinyYOLOv3** model, which means you will be performing your **object detection tasks using the pre-trained “TinyYOLOv3” model you downloaded from the links above**. Find example code below:
* detector.setModelTypeAsTinyYOLOv3()
* **.setModelPath()** , This function **accepts a string which must be the path to the model file you downloaded and must corresponds to the model type you set for your object detection** instance. Find example code,and parameters of the function below:
* detector.setModelPath("yolo.h5")

– *parameter* **model\_path** (required) : This is the **path to your downloaded model file**.

* **.loadModel()** , This function loads the model from the path you specified in the function call above into your object detection instance. Find example code below:
* detector.loadModel()

– *parameter* **detection\_speed** (optional) : This parameter **allows you to reduce the time it takes to detect objects in an image by up to 80% which leads to slight reduction in accuracy**. This parameter accepts string values. The available values are “normal”, “fast”, “faster”, “fastest” and “flash”. **The default values is “normal”**

* **.detectObjectsFromImage()** , This is the function that **performs object detection task after the model as loaded. It can be called many times to detect objects in any number of images**. Find example code below:
* detections = detector.detectObjectsFromImage(input\_image="image.jpg", output\_image\_path="imagenew.jpg", minimum\_percentage\_probability=30)

– *parameter* **input\_image** (required) : This refers to the **path to image file which you want to detect**. You can set this parameter to the Numpy array of File stream of any image if you set the paramter **input\_type** to “array” or “stream”

—*parameter* **output\_image\_path** (required only if **input\_type** = “file” ) : This **refers to the file path to which the detected image will be saved**. It is required only if **input\_type** = “file”

– *parameter* **minimum\_percentage\_probability** (optional ) : This parameter is used to **determine the integrity of the detection results. Lowering the value shows more objects while increasing the value ensures objects with the highest accuracy are detected. The default value is 50**.

—*parameter* **output\_type** (optional ) : This parameter is used to set the format in which the detected image will be produced. **The available values are “file” and “array”. The default value is “file”.** If this parameter is set to “array”, the function will return a Numpy array of the detected image. See sample below:

returned\_image, detections = detector.detectObjectsFromImage(input\_image=”image.jpg”, output\_type=”array”, minimum\_percentage\_probability=30)

—*parameter* **display\_percentage\_probability** (optional ) : This parameter can be used to **hide the percentage probability of each object detected in the detected image if set to False. The default values is True.**

– *parameter* **display\_object\_name** (optional ) : This parameter can be used to **hide the name of each object detected in the detected image if set to False. The default values is True.**

—*parameter* **extract\_detected\_objects** (optional ) : This parameter can be used to **extract and save/return each object detected in an image as a seperate image. The default values is False.**

– *parameter* **thread\_safe** (optional) : This **ensures the loaded detection model works across all threads if set to true.**

—*returns* : The returned values will depend on the parameters parsed into the **detectObjectsFromImage()** function. See the comments and code below

**“”“**

If all required parameters are set and ‘output\_image\_path’ is set to a file path you want the detected image to be saved, the function will return:

1. **an array of dictionaries, with each dictionary corresponding to the objects**

**detected in the image. Each dictionary contains the following property:**

* + name (string)
  + percentage\_probability (float)
  + box\_points (tuple of x1,y1,x2 and y2 coordinates)

“”” detections = detector.detectObjectsFromImage(input\_image=”image.jpg”, output\_image\_path=”imagenew.jpg”, minimum\_percentage\_probability=30)

**“”“**

If all required parameters are set and output\_type = ‘array’ ,the function will return

1. a numpy array of the detected image
2. **an array of dictionaries, with each dictionary corresponding to the objects**

**detected in the image. Each dictionary contains the following property:**

* + name (string)
  + percentage\_probability (float)
  + box\_points (list of x1,y1,x2 and y2 coordinates)

“”” returned\_image, detections = detector.detectObjectsFromImage(input\_image=”image.jpg”, output\_type=”array”, minimum\_percentage\_probability=30)

**“”“**

**If extract\_detected\_objects = True and ‘output\_image\_path’ is set to a file path you want**

the detected image to be saved, the function will return: 1. an array of dictionaries, with each dictionary corresponding to the objects detected in the image. Each dictionary contains the following property: \* name (string) \* percentage\_probability (float) \* box\_points (list of x1,y1,x2 and y2 coordinates). 2. an array of string paths to the image of each object extracted from the image

“”” detections, extracted\_objects = detector.detectObjectsFromImage(input\_image=”image.jpg”, output\_image\_path=”imagenew.jpg”, extract\_detected\_objects=True, minimum\_percentage\_probability=30)

**“”“**

**If extract\_detected\_objects = True and output\_type = ‘array’, the the function will return:**

1. a numpy array of the detected image
2. **an array of dictionaries, with each dictionary corresponding to the objects detected in the image**. Each dictionary contains the following property: \* name (string) \* percentage\_probability (float) \* box\_points (list of x1,y1,x2 and y2 coordinates)
3. an array of numpy arrays of each object detected in the image

“”” returned\_image, detections, extracted\_objects = detector.detectObjectsFromImage(input\_image=”image.jpg”, output\_type=”array”, extract\_detected\_objects=True, minimum\_percentage\_probability=30)

* **.CustomObjects()** , This function is used **when you want to detect only a selected number of objects. It returns a dictionary of objects and their True or False values. To detect selected objects in an image, you will have to use the dictionary returned by this function** with the **detectCustomObjectsFromImage()** function. Find the details in the comment and code sample below:
* *"""*
* *There are 80 possible objects that you can detect with the*
* *ObjectDetection class, and they are as seen below.*
* *person, bicycle, car, motorcycle, airplane,*
* *bus, train, truck, boat, traffic light, fire hydrant, stop\_sign,*
* *parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra,*
* *giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard,*
* *sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket,*
* *bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange,*
* *broccoli, carrot, hot dog, pizza, donot, cake, chair, couch, potted plant, bed,*
* *dining table, toilet, tv, laptop, mouse, remote, keyboard, cell phone, microwave,*
* *oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair dryer,*
* *toothbrush.*
* *To detect only some of the objects above, you will need to call the CustomObjects function and set the name of the* *object(s) you want to detect to through. The rest are False by default. In below example, we detected only chose detect only person and dog.*
* *"""*
* custom = detector.CustomObjects(person=**True**, dog=**True**)
* **.detectCustomObjectsFromImage()**, This function have all the parameters and returns all the values the **detectObjectsFromImage()** functions does but a slight difference. **This function let detect only selected objects in an image**. Unlike the normal **detectObjectsFromImage()** function, this **needs an extra parameter which is “custom\_object”** which accepts the dictionary returned by the **CustomObjects()** function. In the sample below, we set the detection funtion to report only detections on persons and dogs:
* custom = detector.CustomObjects(person=**True**, dog=**True**)
* detections = detector.detectCustomObjectsFromImage( custom\_objects=custom, input\_image=os.path.join(execution\_path , "image3.jpg"), output\_image\_path=os.path.join(execution\_path , "image3new-custom.jpg"), minimum\_percentage\_probability=30)

**Sample Image Object Detection code**

Find below a code sample for detecting objects in an image:

**from** **imageai.Detection** **import** ObjectDetection

**import** **os**

execution\_path = os.getcwd()

detector = ObjectDetection()

detector.setModelTypeAsYOLOv3()

detector.setModelPath( os.path.join(execution\_path , "yolo.h5"))

detector.loadModel()

detections = detector.detectObjectsFromImage(input\_image=os.path.join(execution\_path , "image.jpg"), output\_image\_path=os.path.join(execution\_path , "imagenew.jpg"), minimum\_percentage\_probability=30)

**for** eachObject **in** detections:

print(eachObject["name"] , " : ", eachObject["percentage\_probability"], " : ", eachObject["box\_points"] )

print("--------------------------------")