# Contents

1 Installing ImageAI
   1.1 Prediction Classes ................................................................. 7
   1.2 Detection Classes ................................................................. 10
   1.3 Video and Live-Feed Detection and Analysis ................................. 15
   1.4 Custom Training: Prediction .................................................... 25
   1.5 Custom Object Detection: Training and Inference .......................... 33

2 Indices and tables ........................................................................ 47
ImageAI is a python library built to empower developers, researchers, and students to build applications and systems with self-contained Deep Learning and Computer Vision capabilities using simple and few lines of code. This documentation provides detailed insight into all the classes and functions available in ImageAI, coupled with a number of code examples. ImageAI is a project developed by Moses Olafenwa and John Olafenwa, the DeepQuest AI team.

The Official GitHub Repository of ImageAI is https://github.com/OlafenwaMoses/ImageAI
Installing ImageAI

**ImageAI** requires that you have Python 3.6.x installed as well as some other Python libraries and frameworks. Before you install **ImageAI**, you must install the following dependencies.

- **Python** 3.5.1 or higher, [Download Python](https://www.python.org/downloads/)
- **pip3** , [Download PyPi](https://pypi.org/
- **Tensorflow** 1.13.1 or higher
  ```
  pip3 install tensorflow==1.13.1
  ```
- **OpenCV**
  ```
  pip3 install opencv-python
  ```
- **Keras**
  ```
  pip3 install keras==2.2.4
  ```
- **Numpy**
  ```
  pip3 install numpy==1.16.1
  ```
- **ImageAI**
  ```
  pip3 install imageai --upgrade
  ```

Once **ImageAI** is installed, you can start running very few lines of code to perform very powerful computer visions tasks as seen below.

**Image Recognition**

*Recognize 1000 different objects in images*

- convertible : 52.459555864334106
- sports_car : 37.61284649372101
• pickup : 3.1751200556755066
• car_wheel : 1.817505806684494
• minivan : 1.7487050965428352

Visit Documentation

Image Object Detection

*Detect 80 most common everyday objects in images.*

Visit Documentation

Video Object Detection

*Detect 80 most common everyday objects in videos.*

Visit Documentation

Video Detection Analysis

*Generate time based analysis of objects detected in videos.*

Visit Documentation

Custom Image Recognition Training and Inference

*Train new image new deep learning models on recognize custom objects*

Visit Documentation

Custom Objects Detection Training and Inference

*Train new YOLOv3 models to detect custom objects*
When a Software Engineer get married to another Software Engineer then this happens.
ImageAI provides very powerful yet easy to use classes to perform Image Recognition tasks. You can perform all of these state-of-the-art computer vision tasks with python code that ranges between just 5 lines to 12 lines. Once you have Python, other dependencies and ImageAI installed on your computer system, there is no limit to the incredible applications you can create. 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.Prediction.ImagePrediction

The ImagePrediction class provides you the functions to use state-of-the-art image recognition models like SqueezeNet, ResNet, InceptionV3 and DenseNet that were pre-trained on the the ImageNet-1000 dataset. This means you can use this class to predict/recognize 1000 different objects in any image or number of images. To initiate the class in your code, you will create a new instance of the class in your code as seen below

```python
from imageai.Prediction import ImagePrediction
prediction = ImagePrediction()
```

We have provided pre-trained SqueezeNet, ResNet, InceptionV3 and DenseNet image recognition models which you use with your ImagePrediction class to recognize images. Find below the link to download the pre-trained models. You can download the model you want to use.
After creating a new instance of the `ImagePrediction` class, you can use the functions below to set your instance property and start recognizing objects in images.

- `.setModelTypeAsSqueezeNet()`, This function sets the model type of the image recognition instance you created to the `SqueezeNet` model, which means you will be performing your image prediction tasks using the pre-trained “SqueezeNet” model you downloaded from the links above. Find example code below

```python
prediction.setModelTypeAsSqueezeNet()
```

- `.setModelTypeAsResNet()`, This function sets the model type of the image recognition instance you created to the `ResNet` model, which means you will be performing your image prediction tasks using the pre-trained “ResNet” model you downloaded from the links above. Find example code below

```python
prediction.setModelTypeAsResNet()
```

- `.setModelTypeAsInceptionV3()`, This function sets the model type of the image recognition instance you created to the `InceptionV3` model, which means you will be performing your image prediction tasks using the pre-trained “InceptionV3” model you downloaded from the links above. Find example code below

```python
prediction.setModelTypeAsInceptionV3()
```

- `.setModelTypeAsDenseNet()`, This function sets the model type of the image recognition instance you created to the `DenseNet` model, which means you will be performing your image prediction tasks using the pre-trained “DenseNet” model you downloaded from the links above. Find example code below

```python
prediction.setModelTypeAsDenseNet()
```

- `.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 image prediction instance. Find example code, and parameters of the function below

```python
prediction.setModelPath("resnet50_weights_tf_dim_ordering_tf_kernels.h5")
```

- `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 image prediction instance. Find example code below

```python
prediction.loadModel()
```

- `prediction_speed` (optional): This parameter allows you to reduce the time it takes to predict 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” and “fastest”. The default values is “normal”

- `.predictImage()`, This is the function that performs actual prediction of an image. It can be called many times on many images once the model as been loaded into your prediction instance. Find example code, parameters of the function and returned values below

```python
predictions, probabilities = prediction.predictImage("image1.jpg", result_count=10)
```
-- parameter **image_input** (required) : This refers to the path to your image file, Numpy array of your image or image file stream of your image, depending on the input type you specified.

-- *parameter result_count* (optional) : This refers to the number of possible predictions that should be returned. The parameter is set to 5 by default.

-- *parameter input_type* (optional) : This refers to the type of input you are parse into the **image_input** parameter. It is “file” by default and it accepts “array” and “stream” as well.

returns **prediction_results** (a python list) : The first value returned by the **predictImage** function is a list that contains all the possible prediction results. The results are arranged in descending order of the percentage probability.

returns **prediction_probabilities** (a python list) : The second value returned by the **predictImage** function is a list that contains the corresponding percentage probability of all the possible predictions in the **prediction_results**.

- **predictMultipleImages()** : This function can be used to perform prediction on 2 or more images at once. Find example code, parameters of the function and returned values below

```
results_array = multiple_prediction.predictMultipleImages(all_images_array,
                                               result_count_per_image=5)

for each_result in results_array:
    predictions, percentage_probabilities = each_result["predictions"], each_
                                        result["percentage_probabilities"]
    for index in range(len(predictions)):
        print(predictions[index], " : ", percentage_probabilities[index])
    print("-----------------------")
```

- **sent_images_array** (required) : This refers to a list that contains the path to your image files, Numpy array of your images or image file stream of your images, depending on the input type you specified.

- **result_count_per_image** (optional) : This refers to the number of possible predictions that should be returned for each of the images. The parameter is set to 2 by default.

- **input_type** (optional) : This refers to the format in which your images are in the list you parsed into the **sent_images_array** parameter. It is “file” by default and it accepts “array” and “stream” as well.

- returns **output_array** (a python list) : The value returned by the **predictMultipleImages** function is a list that contains dictionaries. Each dictionary corresponds the images contained in the array you parsed into the **sent_images_array**. Each dictionary has “prediction_results” property which is a list of the prediction result for the image in that index as well as the “prediction_probabilities” which is a list of the corresponding percentage probability for each result.

**Sample Codes**

Find below sample code for predicting one image

```
from imageai.Prediction import ImagePrediction
import os

execution_path = os.getcwd()
prediction = ImagePrediction()
prediction.setModelTypeAsResNet()
prediction.setModelPath(os.path.join(execution_path, "resnet50_weights_tf_dim__ordering_tf_kernels.h5"))
prediction.loadModel()  # (continues on next page)
```
predictions, probabilities = prediction.predictImage(os.path.join(execution_path, "image1.jpg"), result_count=10)
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction, " : ", eachProbability)

Find below sample code for prediction multiple images

```python
from imageai.Prediction import ImagePrediction
import os

execution_path = os.getcwd()

multiple_prediction = ImagePrediction()
multiple_prediction.setModelTypeAsResNet()
multiple_prediction.setModelPath(os.path.join(execution_path, "resnet50_weights_tf_dim_ordering_tf_kernels.h5"))
multiple_prediction.loadModel()

all_images_array = []
all_files = os.listdir(execution_path)
for each_file in all_files:
    if each_file.endswith(".jpg") or each_file.endswith(".png"):  
        all_images_array.append(each_file)

results_array = multiple_prediction.predictMultipleImages(all_images_array, result_count_per_image=5)
for each_result in results_array:
    predictions, percentage_probabilities = each_result["predictions"], each_result["percentage_probabilities"]
    for index in range(len(predictions)):
        print(predictions[index], " : ", percentage_probabilities[index])
    print("-----------------------")
```

1.2 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
Download YOLOv3 Model - yolo.h5
Download TinyYOLOv3 Model - yolo-tiny.h5

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:

```python
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:

  ```python
detector.setModelTypeAsRetinaNet()
  ...
  
  from imageai.Detection import ObjectDetection
  detector = ObjectDetection()
```  

- **.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:

  ```python
detector.setModelTypeAsYOLOv3()
  ...
  
  from imageai.Detection import ObjectDetection
  detector = ObjectDetection()
```  

- **.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:

  ```python
detector.setModelTypeAsTinyYOLOv3()
  ...
  
  from imageai.Detection import ObjectDetection
  detector = ObjectDetection()
```
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:

  ```python
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:

  ```python
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:

  ```python
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 parameter 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 separate 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 the this function with the detectCustomObjectsFromImage() function. Find the details in the comment and code sample below:

```python
***

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, donut, 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.

```python
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:

```python
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:

```python
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
```
1.3 Video and Live-Feed Detection and Analysis

ImageAI provided very powerful yet easy to use classes and functions to perform Video Object Detection and Tracking and Video analysis. 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 videos and live-video feeds from device cameras and IP cameras. 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.VideoObjectDetection ======

This VideoObjectDetection class provides you function to detect objects in videos and live-feed from device cameras and IP cameras, 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 in any video. 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
Download YOLOv3 Model - yolo.h5
Download TinyYOLOv3 Model - yolo-tiny.h5

Once you have downloaded the model you chose to use, create an instance of the VideoObjectDetection as seen below:

```python
from imageai.Detection import VideoObjectDetection
detector = VideoObjectDetection()
```

Once you have created an instance of the class, you can call the functions below to set its properties and detect objects in a video.

- `.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:

  ```python
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")
```

  - `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()
```

  - `detection_speed` (optional) : This parameter allows you to reduce the time it takes to detect objects in a video 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”

- `detectObjectsFromVideo()` . This is the function that performs object detection on a video file or video live-feed after the model has been loaded into the instance you created. Find a full sample code below:

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path, "traffic.mp4"),
                                           output_file_path=os.path.join(execution_path, "traffic_detected"),
                                           frames_per_second=20, log_progress=True)

print(video_path)
```

  - `input_file_path` (required if you did not set `camera_input`) : This refers to the path to the video file you want to detect.

  - `output_file_path` (required if you did not set `save_detected_video` = False) : This refers to the path to which the detected video will be saved. By default, this function saves video .avi format.

  - `frames_per_second` (optional , but recommended) : This parameters allows you to set your desired frames per second for the detected video that will be saved. The default value is 20 but we recommend you set the value that suits your video or camera live-feed.

  - `log_progress` (optional) : Setting this parameter to True shows the progress of the video or live-feed as it is detected in the CLI. It will report every frame detected as it progresses. The default value is False.
— `parameter return_detected_frame` (optional) : This parameter allows you to return the detected frame as a Numpy array at every frame, second and minute of the video detected. The returned Numpy array will be parsed into the respective `per_frame_function`, `per_second_function` and `per_minute_function` (See details below)

— `parameter camera_input` (optional) : This parameter can be set in replacement of the `input_file_path` if you want to detect objects in the live-feed of a camera. All you need is to load the camera with OpenCV’s `VideoCapture()` function and parse the object into this parameter.

See a full code sample below:

```python
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
camera = cv2.VideoCapture(0)
detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(camera_input=camera,
                                          output_file_path=os.path.join(execution_path, "camera_detected_video"),
                                          frames_per_second=20, log_progress=True, minimum_percentage_probability=30)

print(video_path)
```

— `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 display_percentage_probability` (optional) : This parameter can be used to hide the percentage probability of each object detected in the detected video 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 video if set to False. The default values is True.

— `parameter save_detected_video` (optional) : This parameter can be used to or not to save the detected video or not to save it. It is set to True by default.

— `parameter per_frame_function` (optional) : This parameter allows you to parse in the name of a function you define. Then, for every frame of the video that is detected, the function will be parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned can be visualized or saved in a NoSQL database for future processing and visualization.

See the sample code below:

```python
###
This parameter allows you to parse in a function you will want to execute after each frame of the video is detected. If this parameter is set to a function, after every video frame is detected, the function will be executed with the following values parsed into it:
```

(continues on next page)
from imageai.Detection import VideoObjectDetection
import os

def forFrame(frame_number, output_array, output_count):
    print("FOR FRAME " , frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("------------END OF A FRAME --------------")

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel()

video_detector.detectObjectsFromVideo(input_file_path=os.path.
join(execution_path, "traffic.mp4"), output_file_path=os.path.
join(execution_path, "video_frame_analysis"), frames_per_
second=20, per_frame_function=forFrame, minimum_percentage_
probability=30)

In the above example, once every frame in the video is processed and detected, the function
will receive and prints out the analytical data for objects detected in the video frame as you
can see below:

| Output for each object : [{'box_points': (362, 295, 443, 355), 'name' => 'boat', 'percentage_probability': 26.666194200515747}, {'box_points': (319, 245, 386, 296), 'name': 'boat', 'percentage_probability': 30.05296859672546}, {'box_points': (219, 308, 341, 358), 'name': 'boat', 'percentage_probability': 47.46982455253601}, {'box_points': (589, 198, 621, 241), 'name': 'bus', 'percentage_probability': 24.62330162525177}, {'box_points': (519, 181, 583, 263), 'name': 'bus', 'percentage_probability': 27.446213364601135}, {'box_points': (493, 197, 561, 272), 'name': 'bus', 'percentage_probability': 59.81815457344055}, {'box_points': (432, 187, 491, 240), 'name': 'bus', 'percentage_probability': 64.42965269088745}, {'box_points': (157, 225, 220, 255), 'name': 'car', 'percentage_probability': 21.150341629981995}, {'box_points': (324, 249, 377, 293), 'name': 'car', 'percentage_probability': 37.205660343170166}, {'box_points': (152, 275, 260, 327), 'name': 'car', 'percentage_probability': 38.52525353431702}, {'box_points': (3, 296, 134, 359), 'name': 'car', 'percentage_probability': 47.94844686985016}, {'box_points': (481, 266, 546, 314), 'name': 'car', 'percentage_probability': 65.858786198458}, {'box_points': (597, 269, 624, 318), 'name': 'person', 'percentage_probability': 27.125394344329834}] |
Output count for unique objects: {'bus': 4, 'boat': 3, 'person': 1, 'car': 8}

Below is a full code that has a function that tasks the analytical data and visualizes it and the detected frame in real time as the video is processed and detected:

```python
from imageai.Detection import VideoObjectDetection
import os
from matplotlib import pyplot as plt

execution_path = os.getcwd()


resized = False

def forFrame(frame_number, output_array, output_count, returned_frame):
    plt.clf()
    this_colors = []
    labels = []
sizes = []

counter = 0

for eachItem in output_count:
    counter += 1
    labels.append(eachItem + " = " + str(output_count[eachItem]))
    sizes.append(output_count[eachItem])
    this_colors.append(color_index[eachItem])

global resized

if resized == False:
    manager = plt.get_current_fig_manager()
    manager.resize(width=1000, height=500)
    resized = True

plt.subplot(1, 2, 1)
plt.title("Frame : " + str(frame_number))
plt.axis("off")
plt.imshow(returned_frame, interpolation="none")

plt.subplot(1, 2, 2)
plt.title("Analysis: " + str(frame_number))
plt.pie(sizes, labels=labels, colors=this_colors, shadow=True,
       startangle=140, autopct="%.1f%%")

plt.pause(0.01)

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel()

plt.show()

video_detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path, "traffic.mp4"), output_file_path=os.path.join(execution_path, "video_frame_analysis"), frames_per_second=20, per_frame_function=forFrame, minimum_percentage_probability=30, return_detected_frame=True)

---

**parameter per_second_function (optional)**: This parameter allows you to parse in the name of a function you define. Then, for every second of the video that is detected, the function will be parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned can be visualized or saved in a NoSQL database for future processing and visualization.

See the sample code below:
is detected, the function will be executed with the following values parsed into it:

- position number of the second
- an array of dictionaries whose keys are position number of each frame present in the last second, and the value for each key is the array for each frame that contains the dictionaries for each object detected in the frame
- an array of dictionaries, with each dictionary corresponding to each frame in the past second, and the keys of each dictionary are the name of the number of unique objects detected in each frame, and the key values are the number of instances of the objects found in the frame
- a dictionary with its keys being the name of each unique object detected throughout the past second, and the key values are the average number of instances of the object found in all the frames contained in the past second
- If return_detected_frame is set to True, the numpy array of the detected frame will be parsed as the fifth value into the function

```python
from imageai.Detection import VideoObjectDetection
import os

def forSeconds(second_number, output_arrays, count_arrays, average_output_count):
    print("SECOND : ", second_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame: ", count_arrays)
    print("Output average count for unique objects in the last second: ", average_output_count)
    print("-------------END OF A SECOND --------------")

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel()

video_detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path, "traffic.mp4"),
output_file_path=os.path.join(execution_path, "video_second_analysis"),
frames_per_second=20, per_second_function=forSecond, minimum_percentage_probability=30)
```

In the above example, once every second in the video is processed and detected, the function will receive and prints out the analytical data for objects detected in the video as you can see below:

```
Array for the outputs of each frame [[{'box_points': (362, 295, 443, 355), 'name': 'boat', 'percentage_probability': 26.666194200515747}, {'box_points': (319, 245, 386, 296), 'name': 'boat', 'percentage_probability': 30.052968859672546}, {'box_points': (219, 308, 341, 358), 'name': 'boat', 'percentage_probability': 47.46982455253601}, {'box_points': (589, 198, 621, 241), 'name': 'bus', 'percentage_probability': 24.62330162525177}, {'box_points': (519, 181, 583, 263), 'name': 'bus', 'percentage_probability': 27.446213364601135}, {'box_points': (493, 197, 561, 272), 'name': 'bus', 'percentage_probability': 37.205660343170166}, {'box_points': (432, 187, 491, 240), 'name': 'bus', 'percentage_probability': 64.42965269088745}, {'box_points': (157, 225, 220, 255), 'name': 'car', 'percentage_probability': 21.150341629981995}, {'box_points': (324, 249, 377, 293), 'name': 'car', 'percentage_probability': 24.089913070201874}, {'box_points': (152, 275, 260, 327), 'name': 'car', 'percentage_probability': 30.341443419456482}, {'box_points': (433, 198, 485, 244), 'name': 'car', 'percentage_probability': 37.205660343170166}, {'box_points': (184, 226, 233, 260), 'name': 'car', 'percentage_probability': 38.52525353431702}, {'box_points': (3, 296, 134, 359), 'name': 'car', 'percentage_probability': 47.80363142490387}, {'box_points': (357, 302, 439, 359), 'name': 'car', 'percentage_probability': 47.94844686985016}, {'box_points': (481, 266, 546, 314), 'name': 'car', 'percentage_probability': 65.8585786819458}, {'box_points': (597, 269, 624, 318), 'name': 'person', 'percentage_probability': 27.125394344329834}],
```

---

1.3. Video and Live-Feed Detection and Analysis 21
Output average count for unique objects in the last second: {'truck': 0.5, 'bus': 3.7, 'umbrella': 0.8, 'boat': 1.3, 'person': 1.0, 'car': 6.6}

------------END OF A SECOND -------------

Below is a full code that has a function that tasks the analytical data and visualizes it and the detected frame at the end of the second in real time as the video is processed and detected:

```
from imageai.Detection import VideoObjectDetection
import os
from matplotlib import pyplot as plt

execution_path = os.getcwd()


resized = False

def forSecond(frame2_number, output_arrays, count_arrays, average_count, returned_frame):
    plt.clf()
    this_colors = []
    labels = []
    (continues on next page)
sizes = []
counter = 0

for eachItem in average_count:
    counter += 1
    labels.append(eachItem + " = " + str(average_count[eachItem]))
    sizes.append(average_count[eachItem])
    this_colors.append(color_index[eachItem])

global resized

if resized == False:
    manager = plt.get_current_fig_manager()
    manager.resize(width=1000, height=500)
    resized = True

plt.subplot(1, 2, 1)
plt.title("Second : " + str(frame_number))
plt.axis("off")
plt.imshow(returned_frame, interpolation="none")

plt.subplot(1, 2, 2)
plt.title("Analysis: " + str(frame_number))
plt.pie(sizes, labels=labels, colors=this_colors, shadow=True,
       startangle=140, autopct="%.1f%%")
plt.pause(0.01)

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel()

plt.show()

video_detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path, "traffic.mp4"), output_file_path=os.path.join(execution_path, "video_second_analysis"), frames_per_second=20, per_second_function=forSecond, minimum_percentage_probability=30, return_detected_frame=True, log_progress=True)

---

**parameter per_minute_function** (optional) : This parameter allows you to parse in the name of a function you define. Then, for every frame of the video that is detected, the function which was parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned has the same nature as the per_second_function; the difference is that it covers all the frames in the past 1 minute of the video.

See a sample function for this parameter below:

```python
def forMinute(minute_number, output_arrays, count_arrays, average_output_count):
    print("MINUTE : ", minute_number)
    print("Array for the outputs of each frame ", output_arrays)
```
---parameter **video_complete_function** (optional): This parameter allows you to parse in the name of a function you define. Once all the frames in the video is fully detected, the function will be parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned has the same nature as the per_second_function and per_minute_function; the differences are that no index will be returned and it covers all the frames in the entire video.

See a sample function for this parameter below:

```python
def forFull(output_arrays, count_arrays, average_output_count):
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame ": count_arrays)
    print(\"Output average count for unique objects in the entire video: \", average_output_count)
    print("----------END OF THE VIDEO "")
```

---parameter **detection_timeout** (optional): This function allows you to state the number of seconds of a video that should be detected after which the detection function stop processing the video.

See a sample code for this parameter below:

```python
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()

camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, \"resnet50_coco_best_v2.0.1.h5\"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(capture_input=camera, output_file_path=os.path.join(execution_path, \"camera_detected_video\"), frames_per_second=20, log_progress=True, minimum_percentage_probability=40, detection_timeout=120)
```

### 1.4 Custom Training: Prediction

**ImageAI** provides very powerful yet easy to use classes to train state-of-the-art deep learning algorithms like SqueezeNet, ResNet, InceptionV3 and DenseNet on your own image datasets using as few as 5 lines of code to generate your own custom models. Once you have trained your own custom model, you can use the CustomImagePrediction class provided by **ImageAI** to use your own models to recognize/predict any image or set of images.
The `ModelTraining` class allows you to train any of the 4 supported deep learning algorithms (SqueezeNet, ResNet, InceptionV3 and DenseNet) on your own image dataset to generate your own custom models. Your image dataset must contain at least 2 different classes/types of images (e.g. cat and dog) and you must collect at least 500 images for each of the classes to achieve maximum accuracy.

The training process generates a JSON file that maps the objects types in your image dataset and creates lots of models. You will then peak the model with the highest accuracy and perform custom image prediction using the model and the JSON file generated.

Because model training is a compute intensive tasks, we strongly advise you perform this experiment using a computer with a NVIDIA GPU and the GPU version of Tensorflow installed. Performing model training on CPU will my take hours or days. With NVIDIA GPU powered computer system, this will take a few hours. You can use Google Colab for this experiment as it has an NVIDIA K80 GPU available. To train a custom prediction model, you need to prepare the images you want to use to train the model. You will prepare the images as follows:

- Create a dataset folder with the name you will like your dataset to be called (e.g. pets)
  - In the dataset folder, create a folder by the name train
  - In the train folder, create a folder for each object you want to the model to predict and give the folder a name that corresponds to the respective object name (e.g. dog, cat, squirrel, snake)
  - In each folder present in the train folder, put the images of each object in its respective folder. This images are the ones to be used to train the model
  - To produce a model that can perform well in practical applications, I recommend you about 500 or more images per object. 1000 images per object is just great
  - In each folder present in the test folder, put about 100 to 200 images of each object in its respective folder. These images are the ones to be used to test the model as it trains
- Once you have done this, the structure of your image dataset folder should look like below

  pets//train//dog//dog-train-images
  pets//train//cat//cat-train-images
  pets//train//squirrel//squirrel-train-images
  pets//train//snake//snake-train-images
  pets//test//dog//dog-test-images
  pets//test//cat//cat-test-images
  pets//test//squirrel//squirrel-test-images
  pets//test//snake//snake-test-images

Once your dataset is ready, you can proceed to creating an instance of the `ModelTraining` class. Find the example below

```python
from imageai.Prediction.Custom import ModelTraining

model_trainer = ModelTraining()
```

Once you have created an instance above, you can use the functions below to set your instance property and start the training process.
• `setModelTypeAsSqueezeNet()` , This function sets the model type of the training instance you created to the SqueezeNet model, which means the SqueezeNet algorithm will be trained on your dataset. Find example code below

```python
model_trainer.setModelTypeAsSqueezeNet()
```

• `setModelTypeAsResNet()` , This function sets the model type of the training instance you created to the ResNet model, which means the ResNet algorithm will be trained on your dataset. Find example code below

```python
model_trainer.setModelTypeAsResNet()
```

• `setModelTypeAsInceptionV3()` , This function sets the model type of the training instance you created to the InceptionV3 model, which means the InceptionV3 algorithm will be trained on your dataset. Find example code below

```python
model_trainer.setModelTypeAsInceptionV3()
```

• `setModelTypeAsDenseNet()` , This function sets the model type of the training instance you created to the DenseNet model, which means the DenseNet algorithm will be trained on your dataset. Find example code below

```python
model_trainer.setModelTypeAsDenseNet()
```

• `setDataDirectory()` , This function accepts a string which must be the path to the folder that contains the test and train subfolder of your image dataset. Find example code, and parameters of the function below

```python
prediction.setDataDirectory(r"C:/Users/Moses/Documents/Moses/AI/Custom Datasets/pets")
```

- `parameter data_directory (required)`: This is the path to the folder that contains your image dataset.
- `parameter test_subdirectory (optional)`: This is the path to the test folder of your dataset.
- `parameter model_subdirectory (optional)`: This is the path to the folder in which your trained models will be saved.
- `parameter json_subdirectory (optional)`: This is the path to the folder in which the JSON file for your trained models is saved.

• `trainModel()` , This is the function that starts the training process. Once it starts, it will create a JSON file in the dataset/json folder (e.g pets/json) which contains the mapping of the classes of the dataset. The JSON file will be used during custom prediction to produce results. Find example code below

```python
model_trainer.trainModel(num_objects=4, num_experiments=100, enhance_data=True, batch_size=32, show_network_summary=True)
```

- `parameter num_objects (required)`: This refers to the number of different classes in your image dataset.
- `parameter num_experiments (required)`: This is the number of times the algorithm will be trained on your image dataset. The accuracy of your training does increases as the number of times it trains increases. However, it does peak after a certain number of trainings; and that point depends on the size and nature of the dataset.
- `parameter enhance_data (optional)`: This parameter is used to transform your image dataset in order to generate more sample for training. It is set to False by default. However, it is useful to set it to True if your image dataset contains less than 1000 images per class.
**parameter batch_size** (optional) : During training, the algorithm is trained on a set of images in parallel. Because of this, the default value is set to 32. You can increase or reduce this value if you understand well enough to know the capacity of the system you are using to train. Should you intend to change this value, you should set it to values that are in multiples of 8 to optimize the training process.

**parameter show_network_summary** (optional) : This parameter when set to True displays the structure of the algorithm you are training on your image dataset in the CLI before training starts. It is set to False by default.

**parameter initial_learning_rate** (optional) : This parameter is a highly technical value. It determines and control the behaviour of your training which is critical to the accuracy that can be achieved. Change this parameter’s value only if you understand its function fully.

**parameter training_image_size** (optional) : This is the size at which the images in your image dataset will be trained, irrespective of their original sizes. The default value is 224 and must not be set to less than 100. Increasing this value increases accuracy but increases training time, and vice-versa.

**parameter continue_from_model** (optional) : This is used to set the path to a model file trained on the same dataset. It is primarily for continuos training from a previously saved model.

**parameter transfer_from_model** (optional) : This is used to set the path to a model file trained on another dataset. It is primarily used to perform transfer learning.

**parameter transfer_with_full_training** (optional) : This is used to set the pre-trained model to be re-trained across all the layers or only at the top layers.

**parameter save_full_model** (optional) : This is used to save the trained models with their network types. Any model saved by this specification can be loaded without specifying the network type.

### Sample Code for Custom Model Training

Find below a sample code for training custom models for your image dataset

```python
from imageai.Prediction.Custom import ModelTraining

model_trainer = ModelTraining()
model_trainer.setModelTypeAsResNet()
model_trainer.setDatasetDirectory(r"C:/Users/Moses/Documents/Moses/AI/Custom Datasets/pets")
model_trainer.trainModel(num_objects=10, num_experiments=100, enhance_data=True,
batch_size=32, show_network_summary=True)
```

Below is a sample of the result when the training starts

<table>
<thead>
<tr>
<th>Epoch</th>
<th>ETA</th>
<th>Loss</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/25</td>
<td>52s</td>
<td>2.3026</td>
<td>0.2500</td>
</tr>
<tr>
<td>2/25</td>
<td>41s</td>
<td>2.3027</td>
<td>0.1250</td>
</tr>
<tr>
<td>3/25</td>
<td>37s</td>
<td>2.2961</td>
<td>0.1667</td>
</tr>
<tr>
<td>4/25</td>
<td>36s</td>
<td>2.2980</td>
<td>0.1250</td>
</tr>
<tr>
<td>5/25</td>
<td>33s</td>
<td>2.3178</td>
<td>0.1000</td>
</tr>
<tr>
<td>6/25</td>
<td>31s</td>
<td>2.3214</td>
<td>0.0833</td>
</tr>
<tr>
<td>7/25</td>
<td>30s</td>
<td>2.3202</td>
<td>0.0714</td>
</tr>
<tr>
<td>8/25</td>
<td>29s</td>
<td>2.3207</td>
<td>0.0625</td>
</tr>
<tr>
<td>9/25</td>
<td>27s</td>
<td>2.3191</td>
<td>0.0556</td>
</tr>
<tr>
<td>10/25</td>
<td>25s</td>
<td>2.3167</td>
<td>0.0750</td>
</tr>
<tr>
<td>11/25</td>
<td>23s</td>
<td>2.3162</td>
<td>0.0682</td>
</tr>
<tr>
<td>12/25</td>
<td>21s</td>
<td>2.3143</td>
<td>0.0833</td>
</tr>
<tr>
<td>13/25</td>
<td>20s</td>
<td>2.3135</td>
<td>0.0769</td>
</tr>
<tr>
<td>14/25</td>
<td>18s</td>
<td>2.3132</td>
<td>0.0714</td>
</tr>
<tr>
<td>15/25</td>
<td>16s</td>
<td>2.3128</td>
<td>0.0667</td>
</tr>
</tbody>
</table>

(continues on next page)
Let us explain the details shown above:

1. The line Epoch 1/100 means the network is training the first experiment of the targeted 100
2. The line 1/25 [>. . . . . . . . . . . . . . . . . . . . . . . . . . . ..] - ETA: 52s - loss: 2.3026 - acc: 0.2500 represents the number of batches that has been trained in the present experiment
3. The line Epoch 00000: saving model to C:UsersUserPycharmProjectsImageALTestPetsmodelsmodelex_000acc-0.100000.h5 refers to the model saved after the present experiment. The ex_000 represents the experiment at this stage while the acc0.1000 and valacc: 0.1000 represents the accuracy of the model on the test images after the present experiment (maximum value value of accuracy is 1.0). This result helps to know the best performed model you can use for custom image prediction.

Once you are done training your custom model, you can use the CustomImagePrediction class described below to perform image prediction with your model.

===== imageai.Prediction.Custom.CustomImagePrediction ======

This class can be considered a replica of the imageai.Prediction.ImagePrediction as it has all the same functions, parameters and results. The only differences are that this class works with your own trained model, you will need to specify the path to the JSON file generated during the training and will need to specify the number of classes in your image dataset when loading the model. Below is an example of creating an instance of the class

```python
from imageai.Prediction.Custom import CustomImagePrediction

prediction = CustomImagePrediction()
```

Once you have created the new instance, you can use the functions below to set your instance property and start recognizing objects in images.

- **.setModelTypeAsSqueezeNet()**. This function sets the model type of the image recognition instance you created to the SqueezeNet model, which means you will be performing your image prediction tasks using the “SqueezeNet” model generated during your custom training. Find example code below

```python
prediction.setModelTypeAsSqueezeNet()
```

- **.setModelTypeAsResNet()**. This function sets the model type of the image recognition instance you created to the ResNet model, which means you will be performing your image prediction tasks using the “ResNet” model model generated during your custom training. Find example code below

```python
prediction.setModelTypeAsResNet()
```
• `.setModelTypeAsInceptionV3()` , This function sets the model type of the image recognition instance you created to the *InceptionV3* model, which means you will be performing your image prediction tasks using the “InceptionV3” model generated during your custom training. Find example code below

```python
prediction.setModelTypeAsInceptionV3()
```

• `.setModelTypeAsDenseNet()` , This function sets the model type of the image recognition instance you created to the *DenseNet* model, which means you will be performing your image prediction tasks using the “DenseNet” model generated during your custom training. Find example code below

```python
prediction.setModelTypeAsDenseNet()
```

• `.setModelPath()` , This function accepts a string which must be the path to the model file generated during your custom training and must corresponds to the model type you set for your image prediction instance. Find example code, and parameters of the function below

```python
prediction.setModelPath("resnet_model_ex-020_acc-0.651714.h5")
```

— *parameter model_path (required) : This is the path to your downloaded model file.*

• `.setJsonPath()` , This function accepts a string which must be the path to the JSON file generated during your custom training. Find example code and parameters of the function below

```python
prediction.setJsonPath("model_class.json")
```

— *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 image prediction instance. You will have to set the parameter `num_objects` to the number of classes in your image dataset. Find example code and parameter details below

```python
prediction.loadModel(num_objects=4)
```

— *parameter num_objects (required) : This must be set to the number of classes in your image dataset.*

--- *parameter prediction_speed (optional) : This parameter allows you to reduce the time it takes to predict 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” and “fastest”. The default values is “normal”*

• `.loadFullModel()` , This function is used to load the model structure into the program from the file path defined in the `setModelPath()` function. As opposed to the ‘loadModel()’ function, you don’t need to specify the model type. This means you can load any Keras model trained with or without ImageAI and perform image prediction

```python
prediction.loadFullModel(num_objects=4)
```

— *parameter prediction_speed (optional) : Acceptable values are “normal”, “fast”, “faster” and “fastest”.*

— *parameter num_objects (required) : The number of objects the model is trained to recognize.*

• `.save_model_to_tensorflow()` , This function allows you to save your loaded Keras (.h5) model and save it to the Tensorflow (.pb) model format

```python
save_model_to_tensorflow(new_model_folder= os.path.join(execution_path, "tensorflow_model"), new_model_name="idenprof_resnet_tensorflow.pb")
```
- **parameter** new_model_folder (required) : The path to the folder you want the converted Tensorflow model to be saved

- **parameter** new_model_name (required): The desired filename for your converted Tensorflow model e.g 'my_new_model.pb'

- .save_model_for_deepstack() , This function allows you to save your loaded Keras (.h5) model and save it to the deployment format of DeepStack custom API. This function will save the model and the JSON file you need for the deployment

```python
save_model_for_deepstack(new_model_folder= os.path.join(execution_path, "deepstack_model"), new_model_name="idenprof_resnet_deepstack.h5")
```

- **parameter** new_model_folder (required) : The path to the folder you want the converted Tensorflow model to be saved

- **parameter** new_model_name (required): The desired filename for your converted Tensorflow model e.g 'my_new_model.pb'

- **parameter** new_model_folder (required): The path to the folder you want the model to be saved – **parameter** new_model_name (required): The desired filename for your model e.g 'my_new_model.h5'

- .predictImage() , This is the function that performs actual prediction of an image. It can be called many times on many images once the model as been loaded into your prediction instance. Find example code,parameters of the function and returned values below

```python
predictions, probabilities = prediction.predictImage("image1.jpg", result_count=2)
```

- **parameter** image_input (required) : This refers to the path to your image file, Numpy array of your image or image file stream of your image, depending on the input type you specified.

- **parameter** result_count (optional) : This refers to the number of possible predictions that should be returned. The parameter is set to 5 by default.

- **parameter** input_type (optional) : This refers to the type of input you are parse into the image_input parameter. It is “file” by default and it accepts “array” and “stream” as well.

- **parameter** thread_safe (optional) : This ensures the loaded detection model works across all threads if set to true.

- **returns** prediction_results (a python list) : The first value returned by the predictImage function is a list that contains all the possible prediction results. The results are arranged in descending order of the percentage probability.

- **returns** prediction_probabilities (a python list) : The second value returned by the predictImage function is a list that contains the corresponding percentage probability of all the possible predictions in the prediction_results.

- .predictMultipleImages() , This function can be used to perform prediction on 2 or more images at once. Find example code, parameters of the function and returned values below

```python
results_array = multiple_prediction.predictMultipleImages(all_images_array, result_count_per_image=2)
for each_result in results_array:
    predictions, percentage_probabilities = each_result["predictions"], each_result["percentage_probabilities"]
    for index in range(len(predictions)):
        print(predictions[index] , " : " , percentage_probabilities[index])
    print("-----------------------")
```
– **parameter** sent_images_array (required) : This refers to a list that contains the path to your image files, Numpy array of your images or image file stream of your images, depending on the input type you specified.

– **parameter** result_count_per_image (optional) : This refers to the number of possible predictions that should be returned for each of the images. The parameter is set to 2 by default.

– **parameter** input_type (optional) : This refers to the format in which your images are in the list you parsed into the sent_images_array parameter. It is “file” by default and it accepts “array” and “stream” as well.

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

– **returns** output_array (a python list) : The value returned by the predictMultipleImages function is a list that contains dictionaries. Each dictionary corresponds to the images contained in the array you parsed into the sent_images_array. Each dictionary has “prediction_results” property which is a list of the prediction result for the image in that index as well as the “prediction_probabilities” which is a list of the corresponding percentage probability for each result.

Sample Codes

Find below sample code for custom prediction

```
from imageai.Prediction.Custom import CustomImagePrediction
import os

execution_path = os.getcwd()

prediction = CustomImagePrediction()
prediction.setModelTypeAsResNet()
prediction.setModelPath(os.path.join(execution_path, "resnet_model_ex-020_acc-0.651714.h5"))
prediction.setJsonPath(os.path.join(execution_path, "model_class.json"))
prediction.loadModel(num_objects=4)

predictions, probabilities = prediction.predictImage(os.path.join(execution_path, "4.jpg"), result_count=5)

for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction, " : ", eachProbability)
```

1.5 Custom Object Detection: Training and Inference

ImageAI provides the simple and powerful approach to training custom object detection models using the YOLOv3 architecture. This allows you to train your own model on any set of images that corresponds to any type of object of interest.

You can use your trained detection models to detect objects in images, videos and perform video analysis.

```
====== imageai.Detection.Custom.DetectionModelTrainer ======
```

This is the Detection Model training class, which allows you to train object detection models on image datasets that are in Pascal VOC annotation format, using the YOLOv3. The training process generates a JSON file that maps the objects names in your image dataset and the detection anchors, as well as creates lots of models.

To get started, you need prepare your dataset in the Pascal VOC Format and organize it as detailed below:

– Decide the type of object(s) you want to detect and collect about 200 (minimum recommendation) or more picture of each of the object(s)
– Once you have collected the images, you need to annotate the object(s) in the images. You can use a tool like LabelIMG to generate the annotations for your images.

– Once you have the annotations for all your images, create a folder for your dataset (E.g. headsets) and in this parent folder, create child folders **train** and **validation**.

– In the **train** folder, create **images** and **annotations** sub-folders. Put about 70-80% of your dataset of each object’s images in the **images** folder and put the corresponding annotations for these images in the **annotations** folder.

– In the **validation** folder, create **images** and **annotations** sub-folders. Put the rest of your dataset images in the **images** folder and put the corresponding annotations for these images in the **annotations** folder.

– Once you have done this, the structure of your image dataset folder should look like below:

<table>
<thead>
<tr>
<th>train</th>
<th>images</th>
<th>&gt;&gt; img_1.jpg</th>
<th>(shows Object_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>images</td>
<td>&gt;&gt; img_2.jpg</td>
<td>(shows Object_2)</td>
</tr>
<tr>
<td></td>
<td>images</td>
<td>&gt;&gt; img_3.jpg</td>
<td>(shows Object_1, Object_3 and Object_n)</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td>&gt;&gt; img_1.xml</td>
<td>(describes Object_1)</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td>&gt;&gt; img_2.xml</td>
<td>(describes Object_2)</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td>&gt;&gt; img_3.xml</td>
<td>(describes Object_1, Object_3 and Object_n)</td>
</tr>
<tr>
<td>validation</td>
<td>images</td>
<td>&gt;&gt; img_151.jpg</td>
<td>(shows Object_1, Object_3 and Object_n)</td>
</tr>
<tr>
<td></td>
<td>images</td>
<td>&gt;&gt; img_152.jpg</td>
<td>(shows Object_2)</td>
</tr>
<tr>
<td></td>
<td>images</td>
<td>&gt;&gt; img_153.jpg</td>
<td>(shows Object_1)</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td>&gt;&gt; img_151.xml</td>
<td>(describes Object_1, Object_3 and Object_n)</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td>&gt;&gt; img_152.xml</td>
<td>(describes Object_2)</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td>&gt;&gt; img_153.xml</td>
<td>(describes Object_1)</td>
</tr>
</tbody>
</table>

• You can train your custom detection model completely from scratch or use transfer learning (recommended for better accuracy) from a pre-trained YOLOv3 model. Also, we have provided a sample annotated Hololens and Headsets (Hololens and Oculus) dataset for you to train with. Download the pre-trained YOLOv3 model and the sample datasets in the link below.
Sample dataset and pre-trained YOLOv3

- For the purpose of training your detection model, we advice that you have the **Tensorflow-GPU v1.13.1** installed to avoid errors:

```
pip3 install tensorflow-gpu==1.13.1
```

Below is the code to train new detection models on your dataset:

```python
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="hololens")
trainer.setTrainConfig(object_names_array=["hololens"], batch_size=4, num_experiments=200, train_from_pretrained_model="pretrained-yolov3.h5")
trainer.trainModel()
```

In the first 2 lines, we imported the **DetectionModelTrainer** class and created an instance of it

```python
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
```

Then we called the following functions

- **.setModelTypeAsYOLOv3()**, This function sets the model type of the object detection training instance to the YOLOv3 model:

```
trainer.setModelTypeAsYOLOv3()
```

- **.setDataDirectory()**, This function is sets the path to your dataset’s folder:

```
trainer.setDataDirectory()
```

- **parameter data_directory** (required) : This is the path to your dataset folder.

- **parameter object_names_array** (required) : This is a list of the names of all the different objects in your dataset.

- **parameter batch_size** (optional) : This is the batch size for the training instance.

- **parameter num_experiments** (required) : Also known as epochs, it is the number of times the network will train on all the training.

- **parameter train_from_pretrained_model** (optional) : This is used to perform transfer learning by specifying the path to a pre-trained YOLOv3 model

When you run the training code, **ImageAI** will perform the following actions:

- generate a **detection_config.json** in the **dataset_folder/json** folder. Please note that the JSON file generated in a training session can only be used with the **detection models** saved in the training session.

- saves the **Tensorboard** report for the training in the **dataset_folder/logs** folder.

- saves new models n the **dataset_folder/models** folder as the training loss reduces.

As the training progresses, the information displayed in the terminal will look similar to the sample below:
Using TensorFlow backend.

Generating anchor boxes for training images and annotation...
Average IOU for 9 anchors: 0.78
Anchor Boxes generated.
Detection configuration saved in hololens/json/detection_config.json
Training on: ['hololens']
Training with Batch Size: 4
Number of Experiments: 200

Epoch 1/200
Epoch 2/200
Epoch 3/200
Epoch 4/200
480/480 [==============================] - 297s 618ms/step - loss: 5.5802 - yolo_layer_1_loss: 0.9742 - yolo_layer_2_loss: 1.8916 - yolo_layer_3_loss: 2.7144 - val_layer_1_loss: 6.4275 - val_yolo_layer_1_loss: 1.6153 - val_yolo_layer_2_loss: 2.1203 - val_yolo_layer_3_loss: 2.6919
Epoch 5/200
Epoch 6/200
480/480 [==============================] - 300s 624ms/step - loss: 4.7989 - yolo_layer_1_loss: 0.8708 - yolo_layer_2_loss: 1.6683 - yolo_layer_3_loss: 2.2598 - val_layer_1_loss: 5.8672 - val_yolo_layer_1_loss: 1.2349 - val_yolo_layer_2_loss: 2.0504 - val_yolo_layer_3_loss: 2.5820
Epoch 7/200

After training is completed, you can evaluate the mAP score of your saved models in order to pick the one with the most accurate results.

To do this, simply run the code below:

```python
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="hololens")
metrics = trainer.evaluateModel(model_path="hololens/models", json_path="hololens/json/detection_config.json", iou_threshold=0.5, object_threshold=0.3, nms_threshold=0.5)
print(metrics)
```

The above code is similar to our training code, except for the line where we called the `evaluateModel()` function. See details on the function below.
• **.trainer.evaluateModel()**, This function allows you to compute and obtain the **mAP** of your saved model(s) based on criterias such as **IoU** and **confidence score**

trainer.setTrainConfig()

- **parameter model_path** (required): This can be the path to a single model or the folder containing your saved models.
- **parameter json_path** (required): This is the detection_config.json generated during the training that saved the models.
- **parameter iou_threshold** (optional): This is used to set the desired **Intersection over Union** for the **mAP** evaluation.
- **parameter object_threshold** (optional): This is used to set the minimum **confidence score** for the **mAP** evaluation.
- **parameter nms_threshold** (optional): This is used to set the minimum **Non-maximum Suppression** value for the **mAP** evaluation.

When you run the above code, you get a result similar to the one below:

```json
[
    {
        'average_precision': {'hololens': 0.92313437735249},
        'map': 0.92313437735249,
        'model_file': 'hololens/models/detection_model-ex-07--loss-4.42.h5',
        'using_iou': 0.5,
        'using_non_maximum_suppression': 0.5,
        'using_object_threshold': 0.3
    },
    {
        'average_precision': {'hololens': 0.97253437735249},
        'map': 0.97253437735249,
        'model_file': 'hololens/models/detection_model-ex-10--loss-3.95.h5',
        'using_iou': 0.5,
        'using_non_maximum_suppression': 0.5,
        'using_object_threshold': 0.3
    },
    {
        'average_precision': {'hololens': 0.920413437735249},
        'map': 0.920413437735249,
        'model_file': 'hololens/models/detection_model-ex-05--loss-5.26.h5',
        'using_iou': 0.5,
        'using_non_maximum_suppression': 0.5,
        'using_object_threshold': 0.3
    },
    {
        'average_precision': {'hololens': 0.812013437735249},
        'map': 0.812013437735249,
        'model_file': 'hololens/models/detection_model-ex-03--loss-6.44.h5',
        'using_iou': 0.5,
        'using_non_maximum_suppression': 0.5,
        'using_object_threshold': 0.3
    },
    {
        'average_precision': {'hololens': 0.943113437735249},
        'map': 0.943113437735249,
        'model_file': 'hololens/models/detection_model-ex-18--loss-2.96.h5',
        'using_iou': 0.5,
        'using_non_maximum_suppression': 0.5,
    }
]
```

(continues on next page)
'using-object-threshold': 0.3
},
{
'avg-average_precision': {'hololens': 0.94041334437735249},
'map': 0.94041334437735249,
'model_file': 'hololens/models/detection_model-ex-17--loss-3.10.h5',
'using_iou': 0.5,
'using_non_maximum_suppression': 0.5,
'using-object-threshold': 0.3
},
{
'avg-average_precision': {'hololens': 0.97251334437735249},
'map': 0.97251334437735249,
'model_file': 'hololens/models/detection_model-ex-08--loss-4.16.h5',
'using_iou': 0.5,
'using_non_maximum_suppression': 0.5,
'using-object-threshold': 0.3
}]

======= imageai.Detection.Custom.CustomObjectDetection ======

CustomObjectDetection class provides very convenient and powerful methods to perform object detection on images and extract each object from the image using your own custom YOLOv3 model and the corresponding detection_config.json generated during the training.

To test the custom object detection, you can download a sample custom model we have trained to detect the Hololens headset and its detection_config.json file via the links below:

Hololens Detection Model
detection_config.json

- Sample Image

Once you download the custom object detection model file, you should copy the model file to the your project folder where your .py files will be. Then create a python file and give it a name; an example is FirstCustomDetection.py. Then write the code below into the python file:

```python
from imageai.Detection.Custom import CustomObjectDetection
detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("hololens-ex-60--loss-2.76.h5")
detector.setJsonPath("detection_config.json")
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image="holol1.jpg", output_image_path="holo1-detected.jpg")
for detection in detections:
    print(detection["name"], " : ", detection["percentage_probability"], " : ", detection["box_points"])
```

When you run the code, it will produce a result similar to the one below:

```
hololens : 39.69653248786926 : [611, 74, 751, 154]
hololens : 87.6643180847168 : [23, 46, 90, 79]
hololens : 89.25175070762634 : [191, 66, 243, 95]
hololens : 64.49641585350037 : [437, 81, 514, 133]
hololens : 91.78624749183655 : [380, 113, 423, 138]
```
See more details below:

- `.setTypeAsYOLOv3()` , This specifies that you are using a trained YOLOv3 model

```python
detector.setTypeAsYOLOv3()
```

- `.setModelPath()` , This is used to set the file path to your trained model

```python
detector.setModelPath()
```

---

- parameter `detection_model_path` (required) : This is path to your model file

- parameter `configuration_json` (required) : This is path to `detection_json` file

- `.loadModel()` , This is load the detection model:

```python
detector.loadModel()
```

- `.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:

```python
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 parameter `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:

```python
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 separate 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 (list of x1,y1,x2 and y2 coordinates)

```python
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)

```python
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

```python
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 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

```python
returned_image, detections, extracted_objects = detector.detectObjectsFromImage(input_image="image.jpg", output_type="array", extract_detected_objects=True, minimum_percentage_probability=30)
```
== imageai.Detection.Custom.CustomVideoObjectDetection ==

CustomVideoObjectDetection class provides very convenient and powerful methods to perform object detection on videos and obtain analytical from the video, using your own custom YOLOv3 model and the corresponding detection_config.json generated during the training.

To test the custom object detection, you can download a sample custom model we have trained to detect the Hololens headset and its detection_config.json file via the links below:

Hololens Detection Model
detection_config.json

Download a sample video of the Hololens in the link below.
Sample Hololens Video

Then run the code below in the video:

```python
from imageai.Detection.Custom import CustomVideoObjectDetection
import os

execution_path = os.getcwd()

video_detector = CustomVideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath("hololens-ex-60--loss-2.76.h5")
video_detector.setJsonPath("detection_config.json")
video_detector.loadModel()

video_detector.detectObjectsFromVideo(input_file_path="holol.mp4",
output_file_path=os.path.join(execution_path, 
"holol-detected"),
frames_per_second=30,
minimum_percentage_probability=40,
log_progress=True)
```

See details on the available functions below

- **.setModelTypeAsYOLOv3()**, This specifies that you are using a trained YOLOv3 model

- **.setModelPath()**, This is used to set the file path to your trained model

- **.setJsonPath()**, This is used to set the file path to your configuration json file

- **.loadModel()**, This is load the detection model:

- **.detectObjectsFromVideo()**, This is the function that performs object detection on a video file or video live-feed after the model has been loaded into the instance you created. Find a full sample code below:
– parameter input_file_path (required if you did not set camera_input) : This refers to the path to the video file you want to detect.

– parameter output_file_path (required if you did not set save_detected_video = False) : This refers to the path to which the detected video will be saved. By default, this function saves video .avi format.

– parameter frames_per_second (optional, but recommended) : This parameter allows you to set your desired frames per second for the detected video that will be saved. The default value is 20 but we recommend you set the value that suits your video or camera live-feed.

– parameter log_progress (optional) : Setting this parameter to True shows the progress of the video or live-feed as it is detected in the CLI. It will report every frame detected as it progresses. The default value is False.

– parameter return_detected_frame (optional) : This parameter allows you to return the detected frame as a Numpy array at every frame, second and minute of the video detected. The returned Numpy array will be parsed into the respective per_frame_function, per_second_function and per_minute_function (See details below)

– parameter camera_input (optional) : This parameter can be set in replacement of the input_file_path if you want to detect objects in the live-feed of a camera. All you need is to load the camera with OpenCV’s VideoCapture() function and parse the object into this parameter.

– 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 display_percentage_probability (optional) : This parameter can be used to hide the percentage probability of each object detected in the detected video if set to False. The default value is True.

– parameter display_object_name (optional) : This parameter can be used to hide the name of each object detected in the detected video if set to False. The default value is True.

– parameter save_detected_video (optional) : This parameter can be used to save or not to save the detected video or not to save it. It is set to True by default.

– parameter per_frame_function (optional) : This parameter allows you to parse in the name of a function you define. Then, for every frame of the video that is detected, the function will be parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned can be visualized or saved in a NoSQL database for future processing and visualization.

See a sample function for this parameter below::

```python
def forFrame(frame_number, output_array, output_count):
    print("FOR FRAME ", frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("———END OF A FRAME ————")
```

1.5. Custom Object Detection: Training and Inference
—parameter per_second_function (optional) : This parameter allows you to parse in the name of a function you define. Then, for every second of the video that is detected, the function will be parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned can be visualized or saved in a NoSQL database for future processing and visualization.

See a sample function for this parameter below:

```python
def forSeconds(second_number, output_arrays, count_arrays, average_output_count):
    print("SECOND : ", second_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
    print("Output average count for unique objects in the last second: ", average_output_count)
    print("————END OF A SECOND ————–")
```

—parameter per_minute_function (optional) : This parameter allows you to parse in the name of a function you define. Then, for every frame of the video that is detected, the function which was parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned has the same nature as the per_second_function; the difference is that it covers all the frames in the past 1 minute of the video.

See a sample function for this parameter below:

```python
def forMinute(minute_number, output_arrays, count_arrays, average_output_count):
    print("MINUTE : ", minute_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
    print("Output average count for unique objects in the last minute: ", average_output_count)
    print("------------END OF A MINUTE --------------")
```

—parameter video_complete_function (optional) : This parameter allows you to parse in the name of a function you define. Once all the frames in the video is fully detected, the function will was parsed into the parameter will be executed and analytical data of the video will be parsed into the function. The data returned has the same nature as the per_second_function and per_minute_function; the differences are that no index will be returned and it covers all the frames in the entire video.

See a sample funtion for this parameter below:
def forFull(output_arrays, count_arrays, average_output_count):
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame"): count_arrays
    print("Output average count for unique objects in the entire video: ", average_output_count)
    print("--------------END OF THE VIDEO --------------")

—parameter detection_timeout (optional) : This function allows you to state the number of seconds of a video that should be detected after which the detection function stop processing the video.
CHAPTER 2

Indices and tables

- genindex
- modindex
- search