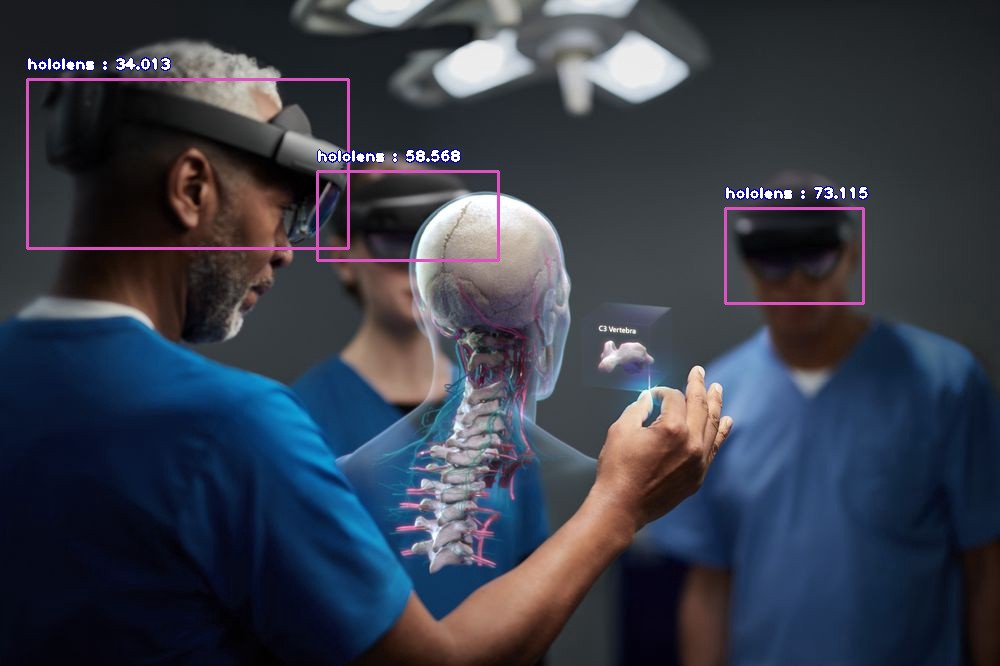
**Train Object Detection AI with 6 lines of code**

*Step-by-step tutorial on training object detection models on your custom dataset*



**Object detection** is one of the most profound aspects of **computer vision** as it allows you to **locate, identify, count and track any object-of-interest in images and videos**. Object detection is used extensively in many interesting areas of work and study such as:

* autonomous vehicles
* security
* pedestrian/crowd detection
* plate number and vehicle detection
* industrial automation (E.g item picking and sorting)
* robotics and more.

However, the challenge with using these public datasets and pre-trained models is that they do not provide a convenient way for you to easily train new object detection models to detect and identify your desired object(s) of interest.   
I am most glad to announce that with the latest release of **ImageAI**v2.1.0, support for training your custom **YOLOv3** models to detect literally any kind and number of objects is now fully supported, and that is what we will guide you to do in this tutorial. Let’s get started.

For the purpose of this tutorial, we will be using **Google Colab**to train on a sample dataset we have provided. Follow the steps below.

**Step 1 — Preparing your dataset**

For your custom detection training, you have to provide **sample images** (your image dataset) for training your model and validating the model after training for accuracy.  For the purpose of this tutorial, we have provided a sample dataset for the **Hololens Mixed Reality**headset, on which we will train a model that can detect and identify the **Hololens**in pictures and videos. You can download the sample dataset via the link below.

<https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/headset.zip>

If you want to train on your own images for your custom object(s), follow the instructions below.

* Decide the type of object(s) you want to detect and collect about **200 (minimum recommendation) or more pictures** of each of the object(s)
* Once you have collected the images, you need to annotate the object(s) in the images. ImageAI uses the Pascal VOC format for image annotation. You can generate this annotation for your images by following a step-by-step tutorial we have provided via the link below.
* 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 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 the sample below:

>> train >> images >> img\_1.jpg  
 >> images >> img\_2.jpg  
 >> images >> img\_3.jpg  
 >> annotations >> img\_1.xml  
 >> annotations >> img\_2.xml  
 >> annotations >> img\_3.xml  
  
  
>> validation >> images >> img\_151.jpg  
 >> images >> img\_152.jpg  
 >> images >> img\_153.jpg  
 >> annotations >> img\_151.xml  
 >> annotations >> img\_152.xml  
 >> annotations >> img\_153.xml

**Step 2 — Installing ImageAI and Dependencies**

Go to [https://colab.research.google.com](https://colab.research.google.com/) and create a new [**Python3.6**](https://www.python.org/downloads/release/python-360/)notebook. Ensure you change the**runtime** for your new notebook to a **GPU.**Then:

* Run the command below to download the sample hololens dataset

!wget <https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/hololens.zip>

* Unzip the zip file for the hololens dataset

!unzip hololens.zip

* For the purpose of this training, install **Tensorflow GPU** version 1.13.1. This is because the default **Tensorflow GPU**version installed on your notebook does generate a *‘\_TfDeviceCaptureOp’ object has no attribute ‘\_set\_device\_from\_string’*error when training your model. If you receive a prompt that your notebook requires a restart after the installation, do restart your notebook after the installation.

!pip3 install tensorflow-gpu==1.13.1

* Install Keras

!pip3 install keras==2.2.4

* Install Numpy

!pip3 install numpy==1.16.1

* Then install the latest version of **ImageAI** from PIP

**Step 3 — Initiate your detection model training**

To ensure that our trained custom models have better detection accuracy, we will be using transfer learning from a **pre-trained YOLOv3**model in the training. **ImageAI** provides the option to train with and without transfer learning. I will strongly recommend you use **transfer learning** except you have thousands of object samples in your dataset.

* Download the pre-trained YOLOv3 model

!wget <https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/pretrained-yolov3.h5>

* Then run the training code as seen below

|  |
| --- |
| 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=100, train\_from\_pretrained\_model="pretrained-yolov3.h5") | |
| trainer.trainModel() |

SIMPLE! The above **6-lines**of code is all you need to initiate the training on your custom dataset. Now let’s break down the code to its part:

* In the first line, we import the “**DetectionModelTrainer”**class from ImageAI
* In the 2nd and 3rd lines, we created an instance of the class and set our model type to **YOLOv3**
* In the 4th line, we set the **path to our custom dataset**
* In the 5th line, we specified the following **parameters**:

— **object\_names\_array:** This is an **array of the names of all the objects in your dataset**. Please note that if your custom dataset annotation has more than one object, simple set the values as shown in the example below

object\_names\_array = ["hololens", "google-glass", "oculus"]

— **batch\_size:**This is the batch size for the training. Kindly note that **the larger the batch size**, the**better**the **detection accuracy**of the saved models. However, due to memory limits on the **Nvidia K80 GPU**available on Colab, we have to keep this value as 4. The batch size can be values of 8, 16 and so on.

— **num\_experiments:**This is the number of **times we want the training code to iterate** on our custom dataset.

— **train\_from\_pretrained\_model:**This is used to leverage transfer learning using the pretrained **YOLOv3**model we downloaded earlier.

Once the training starts,

* ImageAI will generate **detection\_config.json**file in the **hololens/json**folder. This **JSON file will be used during detection of objects in images and videos**
* ImageAI will create **hololens/models**folder which is where all generated models will be saved
* You will see at the log like the sample details 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  
 - 733s - loss: 34.8253 - yolo\_layer\_1\_loss: 6.0920 - yolo\_layer\_2\_loss: 11.1064 - yolo\_layer\_3\_loss: 17.6269 - val\_loss: 20.5028 - val\_yolo\_layer\_1\_loss: 4.0171 - val\_yolo\_layer\_2\_loss: 7.5175 - val\_yolo\_layer\_3\_loss: 8.9683  
Epoch 2/200  
 - 648s - loss: 11.1396 - yolo\_layer\_1\_loss: 2.1209 - yolo\_layer\_2\_loss: 4.0063 - yolo\_layer\_3\_loss: 5.0124 - val\_loss: 7.6188 - val\_yolo\_layer\_1\_loss: 1.8513 - val\_yolo\_layer\_2\_loss: 2.2446 - val\_yolo\_layer\_3\_loss: 3.5229  
Epoch 3/200  
 - 674s - loss: 6.4360 - yolo\_layer\_1\_loss: 1.3500 - yolo\_layer\_2\_loss: 2.2343 - yolo\_layer\_3\_loss: 2.8518 - val\_loss: 7.2326 - val\_yolo\_layer\_1\_loss: 1.8762 - val\_yolo\_layer\_2\_loss: 2.3802 - val\_yolo\_layer\_3\_loss: 2.9762  
Epoch 4/200  
 - 634s - loss: 5.3801 - yolo\_layer\_1\_loss: 1.0323 - yolo\_layer\_2\_loss: 1.7854 - yolo\_layer\_3\_loss: 2.5624 - val\_loss: 6.3730 - val\_yolo\_layer\_1\_loss: 1.4272 - val\_yolo\_layer\_2\_loss: 2.0534 - val\_yolo\_layer\_3\_loss: 2.8924  
Epoch 5/200  
 - 645s - loss: 5.2569 - yolo\_layer\_1\_loss: 0.9953 - yolo\_layer\_2\_loss: 1.8611 - yolo\_layer\_3\_loss: 2.4005 - val\_loss: 6.0458 - val\_yolo\_layer\_1\_loss: 1.7037 - val\_yolo\_layer\_2\_loss: 1.9754 - val\_yolo\_layer\_3\_loss: 2.3667  
Epoch 6/200  
 - 655s - loss: 4.7582 - yolo\_layer\_1\_loss: 0.9959 - yolo\_layer\_2\_loss: 1.5986 - yolo\_layer\_3\_loss: 2.1637 - val\_loss: 5.8313 - val\_yolo\_layer\_1\_loss: 1.1880 - val\_yolo\_layer\_2\_loss: 1.9962 - val\_yolo\_layer\_3\_loss: 2.6471  
Epoch 7/200

**Step 4 — Evaluate your models**

In the sample log shown above, **new models are saved based on the decrease in the validation loss** (E.g — loss: 5.2569) . In most cases, **the lower the loss, the more accurate the model will be detecting objects in images and videos**. However, some models may experience overfitting and still have lower losses. To ensure that you pick the best model for your custom detection, **ImageAI**allows you to evaluate the mAP (mean Average Precision, r*ead more about it*[*here*](https://medium.com/@yanfengliux/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef)) of all the trained models saved in the **hololens/models**folder.

**The higher the mAP, the better the detection accuracy of the model**.

Simple run the code below **on the models saved** during the training.

from imageai.Detection.Custom import DetectionModelTrainer   
trainer=DetectionModelTrainer()  
trainer.setModelTypeAsYOLOv3()  
trainer.setDataDirectory(data\_directory="hololens")  
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)

When you run the above code, you get a result like the example below.

Model File: hololens/models/detection\_model-ex-07--loss-4.42.h5  
Using IoU : 0.5  
Using Object Threshold : 0.3  
Using Non-Maximum Suppression : 0.5  
hololens: 0.9231  
mAP: 0.9231  
===============================  
Model File: hololens/models/detection\_model-ex-10--loss-3.95.h5  
Using IoU : 0.5  
Using Object Threshold : 0.3  
Using Non-Maximum Suppression : 0.5  
hololens: 0.9725  
mAP: 0.9725  
===============================  
Model File: hololens/models/detection\_model-ex-05--loss-5.26.h5  
Using IoU : 0.5  
Using Object Threshold : 0.3  
Using Non-Maximum Suppression : 0.5  
hololens: 0.9204  
mAP: 0.9204  
===============================

Let’s breakdown the evaluation code:

* In the first 4 lines, we import the same training class, created the class instance, set the detection model type and set the path to our dataset’s directory.
* In the 5th line, we called the **.evaluateModel**function and specified the parameters below

—**model\_path:** This is the **path to the folder containing our models**. It can also be the filepath to a specific model.

— **json\_file:** This is the path to the **detection\_config.json** file saved during the training.

—**iou\_threshold:** This is our desired minimum **Intersection over Union**value for the mAP computation. It can be set to values between 0.0 to 1.0

— **object\_threshold:** This is our desired **minimum class score** for the mAP computation. It can be set to values between 0.0 to 1.0.

— **nms\_threshold:** This is our desired **Non-maximum suppression** for the mAP computation.

**Step 5 — Detecting our custom Object in an image**

Now that we have trained our custom model to detect the Hololens headset, we will **use the best model saved** as well as the **detection\_config.json**file generated to detect the object in an image.



* Let’s take the above sample image to test our trained custom hololens detection model.
* We have provided an already trained Hololens detection model for you to test. Download the model and the corresponding **detection\_config.json**file via the links below.

<https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/hololens-ex-60--loss-2.76.h5>

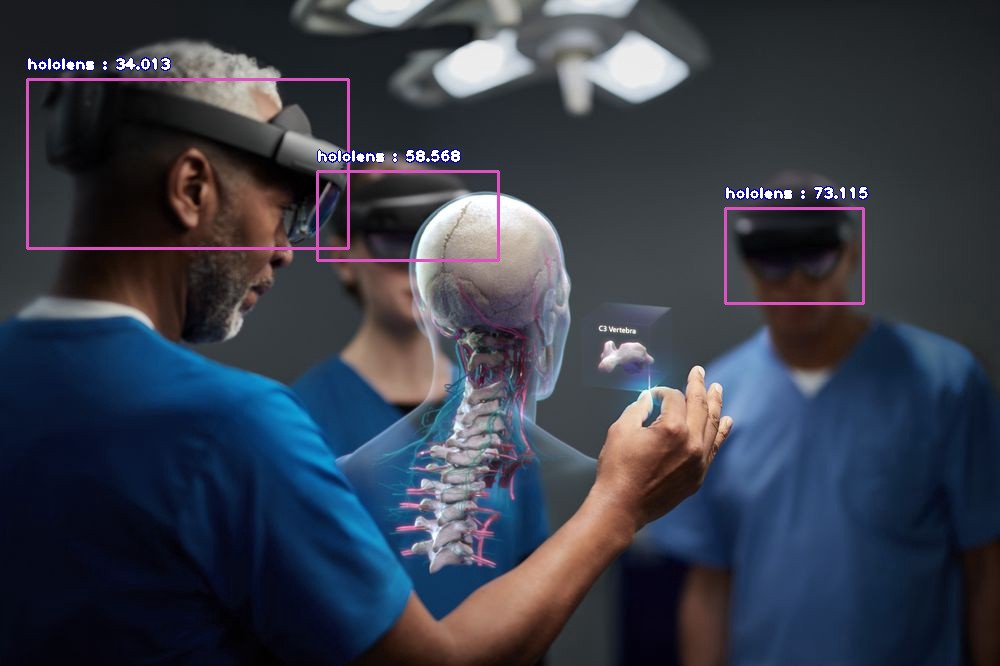
<https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/detection_config.json>

* Now let’s run our custom hololens detection code as seen below.

|  |
| --- |
| 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="holo3.jpg", output\_image\_path="holo3-detected.jpg") |
| for detection in detections: |
| print(detection["name"], " : ", detection["percentage\_probability"], " : ", detection["box\_points"]) |

When we run the above code, we get the result below.

**— RESULT —**



hololens : 34.01297628879547 : [27, 79, 348, 248]  
hololens : 58.56814980506897 : [317, 171, 498, 260]  
hololens : 73.11487197875977 : [725, 208, 863, 303]

**VOILA!**Now we have been able to successfully train a new detection model to detect the **Hololens Mixed Reality**headset.

On a final note, **ImageAI**also allows you to use your custom detection model to detect objects in videos and perform video analysis as well.