The Maltese islands are filled with life and vegetation and it is home to some very rare and unique plants. Unfortunately, a few numbers of people are educated about the subject and this might bring risks that one day these plants are either forgotten or simply becomes extinct. With the help of computer vision as a deep learning method, an image classifier tool can be implemented using an object detection algorithm such as YoloV3 which is used to train a model with a custom built data-set that contains images of Malta’s unique flora. Once that the prediction’s rate of accuracy is high enough, the tool can be later installed on a server and used in conjunction with a number of online services such as an API which can be used in web applications to identify our local flora. In the following sections of this paper, a detailed explanation will be given on how the data-sets were obtained and constructed, which are the ideal algorithms for this project to train the model and how can we identify the most ideal one. Finally, the results obtained during this research will be shown followed by a brief break down of these results to determine if they were accurate and efficient.

In recent years, deep learning has led to substantial improvements in today’s technology, more specifically computer vision tasks such as image classification, object detection, and object segmentation. A deep learning-based model depends on on feature extraction from deep learning networks, contrasting to the traditional machine learning base such as DPM that makes use of HOG as the main feature of learning. A deep learning object detection model is usually designed as a two-stage detector to highlight precision. A two-stage detector type model such as Faster R-CNN has substantial improvements compared to the traditional method. Though, a two-stage detector still has one main flaw, that is in order to operate, it requires a huge number of resources such as computational power and time. As a solution to this problem, Redmond, et al. proposed YOLO as a one-stage deep learning-based object detection model that focuses on speed whilst maintaining accuracy. The main variance from previous works is that YOLO is a one-stage deep learning-based object detection. This model treats the object detection problem as a regression problem which makes YOLO faster by design. The recent release, YOLOv3, holds many improvements over previous versions. YOLO can be considered as a huge milestone that makes deep learning-based object detection model get closer to real-time tasks.

In this paper, an image classification tool is proposed which uses an object detection algorithm. The algorithm has to be quick, efficient and most importantly accurate. Currently, there are primarily two types of state-of-the art object detectors which are two-stage and single-stage detectors. A two-stage detector, such as Faster R-CNN (Region-based Convolutional Neural Networks) or Mask R-CNN uses a Region Proposal Network (RPN) to create regions of interest during the first stage and later sends this region through the pipeline for object classification and bounding-box regression. These models are able to reach the highest accuracy rates but are generally slower. On the other hand, single-stage detectors such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) considers object detection as a simple regression problem by accepting an image as input to understand and learn the class probabilities and bound-box coordinates. Single-stage detectors are very quick and efficient, but they tend to be less accurate than a two-stage detector. With that being said, finding a model that delivers the most ideal trade-off between accuracy and speed is not an easy task.

Evaluation Measure

The performance of object detectors is typically evaluated using the mean Average Precision (mAP) over classes, which is based on the ranking of detection scores for each class. For each object class, the Average Precision is given by the area under the precision-recall (PR) curve for the detected objects. The PR curve is constructed by first mapping each detected bounding box to the most-overlapping ground-truth bounding box, according to the Intersection over Union (IoU) measure, but only if the IoU is higher than 50%. Then, the detections are sorted in decreasing order of their scores. Precision and recall values are computed each time a new positive sample is recalled. The PR curve is given by plotting the precision and recall pairs as lower scored detections are progressively included.

Mean Average Precision (mAP) is typically used to evaluate the performance of object detectors, which is based on the accuracy ranking of detection rates for each class. ‘Precision’ calculates the accuracy rate of a prediction whilst ‘Recall’ calculates the rate of true positives in a number of predictions.  
  
  
  
 In order to obtain the Average Precision for each object class, the area under the precision-recall (PR) curve for the detected objects must be recorded. The PR curve is created by first mapping each detected bounding box to the most overlapping ground-truth bounding box, with regards to the IoU (Intersection over Union) measure, but only if the IoU is higher than 50%. Furthermore, the detections are organized in decreasing order of their scores. Precision and recall values are calculated each time a new positive sample is recalled. The PR curve is given by plotting the precision and recall pairs as lower scored detections are progressively included.

|  |
| --- |
| With the help of computer vision as a deep learning method, I believe that various types of local flora can be classified by training a model using an algorithm and datasets. With this research I intend to deliver a tool which will allow users to identify local flora found in the Maltese islands. With that being said, my objectives are: -To identify whether such data sets exist or how they can be constructed -Analysis existing algorithms and propose a suitable model -Examine the proposed model using real data to determine its effectiveness |

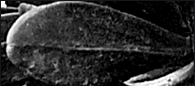
Research questions:

1. How can the required datasets be obtained or created?
2. Which are the ideal algorithms for this project and how can we identify the most ideal one?
3. Are the results accurate and efficient?

Requirements:

Software:  
  
-This project was built on a GPU environment provided by Google Colabs. Google Colabs was used as it out performs any of my computers and it is also a free to use online cloud service.  
  
-Google Drive was used as the main storage location for the project. It was chosen as not only it is free but being a cloud service provided by Google, it is highly secure thus making it safe from any attacks or data corruption. Finally, both the environment and the storage are provided by the same vendor which is Google, making it slightly more user friendly.  
  
- Python 3 is one of the most popular scripting languages at this time and it has a very low learning curve yet still very powerful. Google Colabs is a Linux based environment thus it goes hand in hand with Python 3.  
  
-Github is used as a data repository and also as a way to download libraries for python to install and use.  
  
-WinZip is a file compression tool which is used to compress files such as the images for the datasets. The reason for the usage of this software is because Google drive has a limited amount of storage, datasets were compressed before being uploaded so that they take the least amount of space possible.  
  
-LabelImg Annotation tool is an open source python annotation tool developed by a Github user ‘Tzutalin’. It was used to create the annotations for the images in the datasets by manually selecting the region for it’s corresponding class.  
  
-OpenCV library was installed on the environment as the algorithm used makes heavy usage of it. OpenCV is a highly optimized library with focus on real-time applications.   
  
-Nvidia Cuda ToolKit is used to fully optimize the GPU so it can perform in the best possible way to achieve the best possible results.  
  
-YOLOv3 algorithm was used by installing it’s backbone which is Darknet. YOLOv3 is a real-time, state-of-the-art object detection system.   
  
-Fatkun Batch Downloader is a google extention which was used to scrape images from Google search and finally mass download them. This tool was used to gather images to construct the datasets.   
  
Hardware:   
Image classification algorithms tend to require a lot of resources such as computational and graphical power, but since I used Google Colabs and Google Drive which both are cloud based services, I did not require a very powerful computer to implement this project.   
  
Desktop Computer:  
CPU: Intel Core i5-9400F  
RAM: 32Gb  
Storage technology: NVMe SSD  
GPU: Nvidia Gefore GTX 1070 8GB  
  
As an image capturing device, I used my smartphone which is a OnePlus 7 Pro  
Camera: 48MP, f/1.6

Phase 1: Data Collection  
  
When it comes to data sets which includes images of flora which is mostly found on the Maltese islands, its quite impossible to find a ready built one publicly on the internet. So, I had to rely on the traditional methods of scraping the images from a search engine such as Google Images and driving around rural areas around Malta to take photos of these indigenes’ plants. For the purpose of this project, I selected two kinds of plants that are ‘Opuntia Ficus Indica’ which is more commonly known as ‘Bajtar’ and ‘Cheirolophus Crassifolius’ which is more commonly known as ‘Widnet il-Baħar’. There are two reasons why I have selected only two kinds of plants from the wide variety of flora available, and this is because training the model takes a lot of time and as I add more images and classes, the training duration drastically increases. The second reason is that although the leaf of ‘Opuntia Ficus Indica’ is much larger than that of ‘Cheirolophus Crassifolius’, they share a very similar shape, thus taking this opportunity to witness the accuracy rate of such scenario.







  
Once I had about one hundred of each type of plant, I started working on the annotations. LabelImg was used using python as a compiler to manually label the each and every image to create annotations by selecting a region on the image and linking it with its corresponding class such as ‘Cheirolophus Crassifolius flower’ or ‘Cheirolophus Crassifolius bulb’ .

Once this is done, a text file with the same name as corresponding image gets created which contains the pre-selected region’s x axis and y axis positions and the corresponding class. Once all of the images have been processed, they are divided into two folders; one for training which contains 80% of the total images, and one for testing which contains the other 20%.

Phase 3 - Model Training

Once the data set has been obtained, I commenced to start training the model. I started by creating an environment on Google Colab which uses GPU as the main runtime type. After the environment was configured, I connected it to my Google Drive storage and authenticated it for further use. Using a Git command, darknet was downloaded into the environment solution and then I ran a command to alter the darknet make file, so it enables and uses GPU and OPENCV libraries. Once that is done, Darknet was compiled using the ‘!make’ command whilst the dataset was compressed and uploaded from my local machine to Google Drive, and finally copied and extracted into the Darknet data folder in Google Colab’s solution. An additional backup folder was created in Google Drive which will serve as a location which YOLOv3 will use to save the trained model in later on. Two specific python functions [upload() and download()]were included into the environment which helped me to upload and download files directly from and to Colab’s solution, and an additional function [imShow(path)] was added which enabled me to display images in Colab’s terminal. Now that the algorithm was installed and configured, I uploaded four very important files. The first one is the configuration file for the YOLOv3 algorithm, the second one included all the classes names and the third file contained the dataset details and configuration file path, images directory path, image annotations directory path and the backup folder path which was created earlier. Finally, the fourth file was uploaded which contained a python script that used the first three files to generate a text document containing all the paths of each and every image with their corresponding annotatio. Once that was done, a pretrained convolutional layer model which was provided by Darknet developers, was downloaded using Git. The purpose of this pretrained model is so that it would serve as a base to the model which I was about to train. Finally, one small but crucial command was executed in the browser console which stopped Google Colab from timing out after a couple of minutes with no user interactivity, thus stopping the training process. This allowed me to train the model for countless hours with no interruption whatsoever. Once everything was in order, I executed the train command to commence the model training.