Maltese Flora Classifier

Isaac Tanti
Institute of Information & Communication Technology
Malta College or Arts, Science & Technology
Corradino Hill
Paola PLA 9032
isaac.tanti.a104765@mcast.edu.mt

***Abstract*—With the help of computer vision as a deep learning method, various types of objects in graphical visuals can be classified. The main target of this research is to deliver a tool that will enable users to identify local flora found on the Maltese islands. In this study, we propose the use of image processing techniques, namely Convolutional Neural Networks (CNN) as the model which will be trained using YOLOv3 as a single-stage object detection algorithm with a custom-built dataset containing images of local Maltese flora such as Opuntia Ficus Indica and Cheirolophus Crassifolius. Google Colabs will be used as a platform to build the tool using Python as the main language, OpenCV library, YOLOv3 with darknet backbone and other tools such as Github and and Google Drive as repositories. The proposed tool managed to detect targets with a Mean Average Precision of up to 76.13% and we identify situations where this is challenging and recommend future directions.**

***Index Terms*—MCAST, IICT, CNN, YOLOv3, F R-CNN, SSD, Dataset**

I. Introduction

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 **dataset** 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 research paper, a detailed explanation will be given on which are the ideal algorithms for this project to train the model and how can we identify the most suitable one, and how the data-sets were obtained and constructed. 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.

II. Literature Review

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 (Deformable Part Model) [1] that makes use of HOG (Histogram of Oriented Gradients) [2] 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 (Region-based Convolutional Neural Networks) [3] 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. [4] proposed YOLO (You Only Look Once) 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 [5], 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.

Soviany and Ionescu had explained in their research paper "*Optimizing the Trade-off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction*" [6] 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 or Mask R-CNN [7] 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 and SSD (Single Shot MultiBox Detector) [8] 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. In Soviany and Ionescu research paper [6], they had tackled this issue using image difficulty predictions and concluded that although both types of detectors have their strengths and weaknesses, their use is recommended for different scenarios, thus even though two-stage detectors have the most accurate rate of predictions, single-stage detectors would still provide a very reasonable accuracy rate whilst being much faster than a two-stage detector.

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.

In Joseph Redmon and Ali Farhadi's paper [5], they compared their brand new algorithm which is YOLOv3 in three different variations with other detectors such as different variants of SSD, FPN and RetinaNet.

TABLE I
ALGORITHM COMPARISON.

Algorithm	Inference time (ms)	mAP
SSD321	61	28.0
DSSD321	85	28.0
SSD513	125	31.2
DSSD513	156	33.2
FPN FRCN	172	36.2
RetinaNet-50-500	90	32.5
RetinaNet-101-500	90	34.4
RetinaNet-101-800	198	**37.8**
YOLOv3-320	**22**	28.2
YOLOv3-416	29	31.0
YOLOv3-608	51	33.0

Through these findings, Redmon and Farhadi have concluded that although all YOLOv3 variants did not have the best mAP (mean average precision), it is still an exceptionally good accuracy rate with YOLOv3 having the best speed from all the other algorithms.

1) **TP = True Positive** - An outcome produced by a model which correctly predicted the positive class.
2) **TN = True Negative** - An outcome produced by a model which correctly predicted the negative class.
3) **FP = False Positive** - An outcome produced by a model which incorrectly predicted the positive class.
4) **FN = False Negative** - An outcome produced by a model which incorrectly predicted the negative class.

- **Accuracy (ACC)** - This shows the measure of effectiveness of the machine learning model.

$$Accuracy(ACC) = \frac{TP + TN}{P + N} \tag{1}$$

- **Precision (PPV)** - Precision is the ratio of correctly predicted positive values to the total predicted positive values. This metric highlights the correct positive predictions out of all the positive predictions. High precision indicates low false positive rate.

$$Precision(PPV) = \frac{TP}{TP + FP} \tag{2}$$

- **Recall (TPR)** - The recall is the ratio of correctly predicted positive values to the actual postive values. Recall highlights the sensitivity of the algorithm i.e. out of all the actual positives how many were caught by the program.

$$Recall(TPR) = \frac{TP}{P} \tag{3}$$

III. RESEARCH METHODOLOGY

Problem & Hypothesis

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 data sets.

Aim & Objectives

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 **data sets** be obtained or created?
2) Which are the ideal **algorithms** for the purpose of this project and how can I identify the most ideal one?
3) Are the **results** accurate and efficient?

Chosen algorithm:

With the research done in the previous section, I have concluded that YOLOv3 is the most ideal algorithm to use in this project. The reason for this decision is although YOLOv3 is a single-stage algorithm, it still provides an outstanding accuracy rate with the quickest response time from all the other state-of-the-art algorithms.

Software Requirements:

- **Windows 10** - Windows 10 is the latest version of Windows operating system. It was used as it is very user friendly whilst still providing the key tools such as a terminal.
- **Google Chrome** - Google Chore was used as it is one of the fastest and most optimized internet browsers at this time. Furthermore, it is produced by Google, which makes it safe and reliable. Finally, being a Google product, it goes hand in hand with other Google services such as Goolge Drive and Google Colab.

- **Google Colab** - 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** - 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** - 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** - Github is used as a data repository and also as a way to download libraries for python to install and use.
- **WinZip** - 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** - 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** - 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** - 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 with Darknet backbone** - 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** - This tool 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 Requirements:

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. As an image capturing device, I used my smartphone which is a OnePlus 7 Pro

- Desktop Computer:
  - CPU: Intel Core i5-9400F
  - RAM: 32GB
  - Storage Technology: NVMe SSD
  - GPU: Nvidia GeForce GTX 1070 8GB
- OnePlus 7 Pro
  - Main Camera: 48MP, f/1.6

Research Pipeline

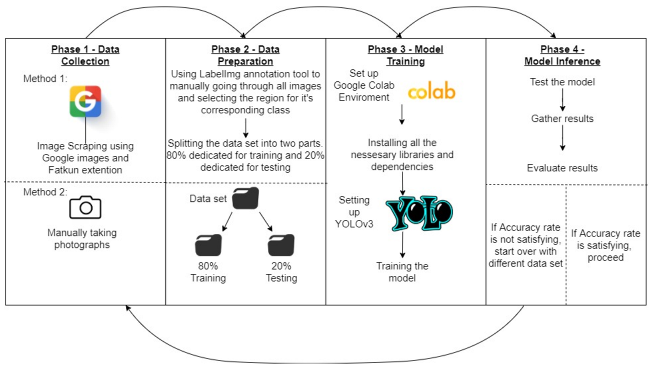

Phase 1 - Data Collection
Method 1:
Image Scraping using Google images and Fatkun extention
Method 2:
Manually taking photographs
Phase 2 - Data Preparation
Using LabelImg annotation tool to manually going through all images and selecting the region for it's corresponding class
Splitting the data set into two parts. 80% dedicated for training and 20% dedicated for testing
Data set
80% Training
20% Testing
Phase 3 - Model Training
Set up Google Colab Enviroment
colab
Installing all the nessesary libraries and dependencies
Setting up YOLOv3
YOLO
Training the model
Phase 4 - Model Inference
Test the model
Gather results
Evaluate results
If Accuracy rate is not satisfying, start over with different data set
If Accuracy rate is satisfying, proceed

Fig. 1. Research Pipeline

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-Bahar'. 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.

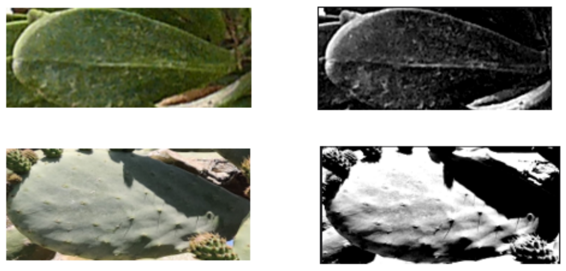


Fig. 2. Leaf comparison

Phase 2: Data Preparation

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 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%.

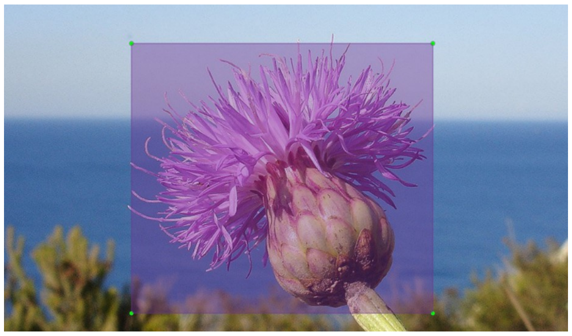


Fig. 3. Image annotation tool

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 were included into the environment which helped me to upload and download files directly from and to Colab’s solution, and an additional function 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. The time it takes to train depends on the size of the data set, and I ended up waiting for around 17 hours. During that time, I could monitor the progress by displaying a graph (Fig. 4) of loss over iterations.

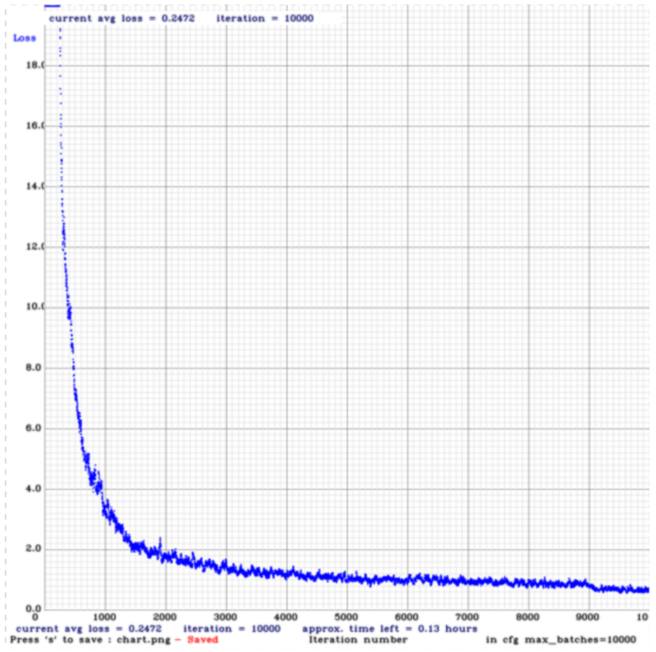
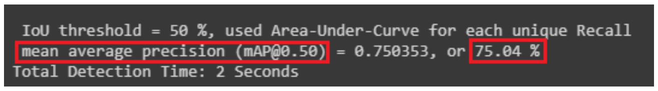


Fig. 4. Graph of Loss over Iteration

Phase 4 - Model Inference

Once that the training has been complete, a command was executed to start the testing to obtain the mAP and IoU percentages. A total of 10 YOLOv3 weights has been produces, and each and every weight has different percentage ratings. After observing the results, I have determined the mAP and average mAP of all the weights. After the results were observed, I had noticed that they they aren’t satisfying enough for image classification, therefore I had to restart the whole process and recreated an updated dataset with more images which has better and clearer graphics. After the second model was trained, the results were observed and I noticed that the mAP has drastically increased, with the highest percentage exceeding my expectations, therefore concluding my training there.



```
 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.750353, or 75.04 %
Total Detection Time: 2 Seconds
```

Fig. 5. YOLOv3 Output of mAP calculation

IV. Findings & Discussion of Results

In the previous sections of this research paper, we have identified the requirements needed to construct the proposed solution. The steps which show how it was implemented were also documented, with the respective results recorded in the tables below.

TABLE II
First data set Weights mAP comparison.

Weight	mAP %
yolov3_custom_weight_1000	51.00%
yolov3_custom_weight_2000	53.75%
yolov3_custom_weight_3000	53.45%
yolov3_custom_weight_4000	**56.13%**
yolov3_custom_weight_5000	52.78%
yolov3_custom_weight_6000	54.51%
yolov3_custom_weight_7000	53.52%
yolov3_custom_weight_8000	52.31%
yolov3_custom_weight_9000	52.53%
yolov3_custom_weight_10000	55.04%
Average	53.502%

In table II, the results of the first model are shown and one can clearly observe that they aren't satisfying enough for image classification. With an average mAP of 53.502% and the highest mAP rate of 56.13%, they weren't high enough to deem the tool accurate and efficient. After the whole process was done again with an updated dataset, the following results shown in table III were obtained.

TABLE III
Second data set Weights mAP comparison.

Weight	mAP %
yolov3_custom_weight_1000	71.00%
yolov3_custom_weight_2000	73.75%
yolov3_custom_weight_3000	75.15%
yolov3_custom_weight_4000	**76.13%**
yolov3_custom_weight_5000	71.78%
yolov3_custom_weight_6000	74.91%
yolov3_custom_weight_7000	73.52%
yolov3_custom_weight_8000	72.33%
yolov3_custom_weight_9000	74.93%
yolov3_custom_weight_10000	75.04%
Average	73.854%

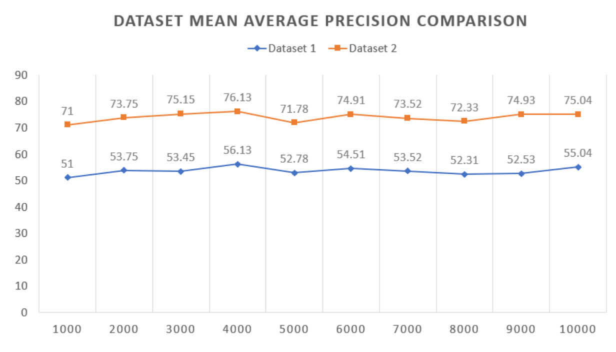


Fig. 6. Dataset comparison

The above graph (Fig. 6) shows that both datasets were compared and a drastic increase in accuracy can be clearly noticed between the two. This does not only show that the updated dataset had effected the accuracy of the model, but also it has proven that the larger the dataset, the more accurate the model is.

From the results in the above tables, one can also notice the similarities in both datasets as the 4th weight from both datasets had the highest mAP percentage, therefore concluding that although there were more latest variants of weights, it does not necessarily mean that they will have a better mAP. In figure 5 one can also notice the time taken to finalize the testing which only took 2 seconds for the whole test. This is clear proof that YOLOv3 is indeed a very fast algorithm.

V. Conclusion

Finally, with regards to the findings and results in the previous sections, we can conclude by answering the initial research questions.

Research Questions covered:

1: How can the required datasets be obtained or created?

The internet is the largest source of information which one may use in order to gather any type of data. Having said that, pre-constructed datasets are widely available to download for free, but unfortunately, the specific type of dataset that I was looking for was not yet available. Therefore, I constructed my own dataset by scraping images from search engines and manually taking photos of the subject class, which later each image would be manually analyzed and the region of interest which corresponds to a class would be recorded.

2: Which are the ideal algorithms for the purpose of this project and how can I identify the most ideal one?

The two main types of image classification algorithms are single-stage and two-stage state-of-the-art algorithms where single-stage algorithms provide a fast result with an moderate rate of accuracy whilst two-stage algorithms provide the best accuracy rates with a much slower result. For the sake of this research paper and the tool implementation, a single-stage algorithm, more specifically YOLOv3, was deemed to be the most ideal to use as it is the fastest from all the algorithms researched in the section above and it still provided a decent accuracy rate.

3: Are the results accurate and efficient?

Taking into consideration the quality of datasets used, an average mAP rate of 73.854% produced by the trained model was obtained. With regards to the algorithm's documentation, the achieved rate exceeded the documented ideal rate for the best results. Finally, by using the fastest single-stage algorithm at this time, we can conclude that the results are both accurate and efficient.

VI. Future Directions

This tool has unlimited potential and can be used in several productive ways. We plan to keep working on new datasets which will include more varieties of local flora. Another future option could be; adding local fauna such as indigenes insects, birds, lizards, mammals and fish. The use of an additional database containing flora and fauna information, combined with this tool and an Augmented Reality approach could result in a fun and educational smartphone application for schools to use to educate their students.

Acknowledgement

The author is truly grateful and would like to thank the following personal for assisting in the proof reading of this research paper: Michaela Micallef, Michela Cutajar. The author would also like to thank Christian Cassar, Keane Schembri, Matthew Vella and any other personal who contributed in finding necessary research for this paper.

References

[1] P. F. Felzenszwalb, R. B. Girshick, D. Mcallester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, p. 1627–1645, 2010.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR05)*.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, p. 1137–1149, Jan 2017.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5]

[6] P. Soviany and R. T. Ionescu, "Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction," *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2018.

[7]

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Computer Vision – ECCV 2016 Lecture Notes in Computer Science*, p. 21–37, 2016.