DM

Classification and Processing of Marine Images

MASTER DISSERTATION

José Filipe Góis Velosa
MASTER IN COMPUTER ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

September | 2019

Classification and Processing of Marine Images

MASTER DISSERTATION

José Filipe Góis Velosa
MASTER IN COMPUTER ENGINEERING

SUPERVISOR
Diogo Nuno Crespo Ribeiro Cabral

UNIVERSIDADE da MADEIRA

Classification and Processing of Marine Images

José Filipe Góis Velosa

Constituição do júri de provas públicas:

Karolina Baras, (Professora Auxiliar da Universidade da Madeira), Presidente

Pedro Filipe Pereira Campos, (Professor Associado com Agregação da Universidade da Madeira), Vogal

Diogo Nuno Crespo Ribeiro Cabral, (Investigador Auxiliar do Instituto Superior Técnico), Vogal

Novembro 2019

Funchal – Portugal

Resumo

A classificação de imagens possibilita o processamento de um grande número de imagens e identificar objetos específicos. Este trabalho usa esta classificação para identificar animais marinhos como um caso de estudo. Foram treinados e utilizados dois modelos diferentes um deles o YOLOv3 e o outro Keras Retina Net. Ambos foram treinados com o mesmo dataset de imagens, de forma a que os resultados pudessem ser comparados. Este dataset é constituído por um total de 1329 imagens que foram divididas em duas classes, a classe baleia e a classe golfinho. Todas estas imagens necessitaram de ser marcadas conforme a classe a que pertencem. Os resultados demonstram que YOLOv3 obteve uma maior precisão que rondava os 89% tendo obtido com o Keras Retina Net resultados a rondar os 84%. Tendo também o YOLOv3 obtido um tempo de inferência inferior ao obtido pelo Keras Retina Net. Face aos resultados obtidos é possível demonstrar que a utilização de *machine learning* é útil para a classificação e identificação de elementos específicos no meio marinho. Adicionalmente, foi desenvolvida uma aplicação em Python, que permite classificar uma imagem consoante um modelo previamente treinado e que exemplifica como se pode integrar um algoritmo de classificação numa aplicação destinada ao utilizador final. De forma a melhorar a utilização da aplicação desenvolvida efetuaram-se testes de usabilidade informais. Os modelos de classificação de imagens podem ser utilizados em diversos tipos de aplicações. Este trabalho demonstra a sua utilização para ajudar a estudar ecossistemas marinhos, possibilitando uma maior facilidade e rapidez de identificação de elementos em meio aquático.

Palavras-Chave

Imagens Marinas; Classificação de Imagens; Aprendizagem automática; YOLOv3; Keras RetinaNet; Visão Computacional

Abstract

Image classification allows to process a great number of images and identify specific objects within these images. This work uses this classification to identify marine animals, as a case study. Two different models were trained and used, one of them being YOLOv3 and the other Keras RetinaNet. Both were trained with the same image dataset, ensuring that the results could be compared. This dataset is made up by a total of 1329 images and divided into two classes: a whale and a dolphin class. All of the images needed to be labelled accordingly to the class contained in the image. Results showed that YOLOv3 had a higher precision of about 89%, while the results for Keras RetinaNet were around 84%. YOLOv3 also has a lower inference time compared to the one obtained with Keras RetinaNet. The results obtained showed that the use of machine learning can help with the identification of specific marine elements. In addition, a Python application to classify images following a trained model was developed, exemplifying how an image classification algorithm can be integrated into an end-user application. To improve the usability of this program informal usability tests were conducted.

Image classification models can be used on a diverse type of applications. This work shows how they can be used to help in the study of marine ecosystems, allowing easier and faster identification of aquatic elements.

Keywords

Marine Images; Image Classification; Image Processing; YOLOv3; Keras RetinaNet; Computer Vision

Contents

List of Figures

Acronyms

AI *Artificial Intelligence*
API *Application Program Interface*
BSD *Berkeley Software Distribution*
CIS *Carcinoma In Situ*
CNN *Convolutional Neural Networks*
COCO *Common Objects in Context*
CPU *Central Processing Unit*
DOTA *Dataset for Object Detection in Aerial Images*
DSSD *Deconvolutional Single Shot Detector*
FCN *Fully Convolutional Network*
FPN *Feature Pyramid Network*
GB *Gigabyte*
GPU *Graphics Processing Unit*
GUI *Graphical User Interface*
IC *Invasive Caner*
IOU *Intersection Over Union*
LSTM *Long Short Term Memory*
NMS *Non-Maximum Suppression*
OS *Operating System*
PC *Personal Computer*
PIL *Python Imaging Library*
PNG *Portable Network Graphics*
RAM *Random Acess Memory*
R-CNN *Region-Convolutional Neural Network*
RGB *Red Green and Blue*
SORT *Simple Online Real-time Tracking*
SSD *Single Shot Detector*
TIP *Threat Image Projection*
UAV *Unmanned Aerial Vehicle*
VRAM *Video Random Acess Memory*
YOLO *You Only Look Once*

1 Introduction

Covering most of the planet, oceans are among the earth's most valuable natural resources. However, they are the endpoint for much of the pollution produced by humans on land, destroying them and changing the behavior of all the species that depend on it (Nagelkerken and Munday, 2016). From dangerous carbon emissions to marine animals choking on plastic to leaking oil to constant noise, the types of ocean pollution human beings generate, in the ocean are vast. As a collective result, human impact on the oceans is degrading the marine ecosystems at an alarming rate (McCauley et al., 2015). Therefore, it is crucial to study such effects. Having a way to track and identify different species in different situations would help with this as seen on the work of (Beijbom, 2015) with the automation of coral identification. Besides marine's species getting tangled in possible debris, be it from fishing nets or smaller plastics leading to death or injury, or in case they do not get tangled many animals mistake plastic debris for food and eat it, filling their stomachs with plastic and other marine debris they cannot digest. In addition, debris can, in some cases, damage important habitats like coral reefs, which are the basis of the marine ecosystem. Being able to track and identify marine elements, like animals, also helps us study the impact that humans have on their habits and how they react and try to adapt to these changes.

Marine ecosystems are at risk, as such, it is crucial to find ways and techniques that can help to study them. This work aims to help in this issue by working on processing and classification of marine elements, as a case study of sorts about how artificial intelligence can be applied to study marine ecosystems. This can be done by using machine learning and teaching it to identify specific objects on the ocean, in this case, the images were taken from the Open Image Dataset, images of whales and dolphin, which were then processed and used to train a model to identify these two classes, this was done by using YOLOv3 and Keras Retina Net. These methods can also be applied to anything that needs an image classified or identified, a flexible topic that can be applied in multiple situations, for example in health where it is most commonly used for cancer identification, in a way to further assist medical professionals. The application was written using Python due to it being a flexible language and one of the most used in the machine learning field.

1.1 Contribution

In this work, two image classification models were trained, YOLOv3 and Keras RetinaNet, showing what can be done with machine learning and how it can be used, for example in marine preservation. The models were then compared showing that they perform in different situations. The models were trained to identify whales and dolphins using image detection, while a different approach called image prediction was used to distinguish between images of whales and dolphins. The main focus was placed on detection because that is the one that is most useful in almost every situation. In addition, this work contributes with a Python application to classify images, demonstrating the workflow needed to integrate an image classification algorithm in an end-user application.

1.2 Thesis Outline

In chapter 2, the related work is presented looking into previous works that relate to this topic, works that helped in the creation of this work and the development of the prototype.

Chapter 3 describes the developed prototype, its architecture, the frameworks and APIs used as well as the workflow to integrate image classification algorithms in an end-user application.

In chapter 4, the evaluation of the results achieved by the YOLOv3 and Keras RetinanNet are shown and compared, ending with the discussion of these results.

Chapter 5 presents the main conclusions of this work as well as some of the limitations of the development of the application, and the future work that could be done to improve it.

2 Related Work

This section will serve to look at other papers that also use image classification, Keras, TensorFlow, multiple uses for image classification will be shown here, this is a wide topic with many fields that use it for multiple different objectives.

2.1 Image classification

Starting off with a paper that uses deep learning to the classifications of images from colposcopy in the work of (Sato et al., 2018), the objective of the study was to investigate if deep learning could be successfully applied to the classification of images from colposcopy. A total of 158 patients were enrolled, their medical records and data from the gynaecological oncology databases were reviewed. Keras neural networks and TensorFlow were the libraries used for deep learning. Preoperative images from colposcopy were used as the input data, using deep learning the patients were classified into three groups. A total of 485 images were used, divided across the three groups, of which 142 images were of severe dysplasia, 257 were of carcinoma in situ (CIS) and 86 of invasive cancer (IC). The end accuracy of the validation dataset was around 50%. The images were stored in PNG format with a resolution of 640x480. The raw images had to be trimmed, due to some having unwanted information, they were trimmed down to 300x300 pixels. To be able to trim the images Photoshop was used. The images were also re-trimmed to 150x150 and then used as input this size was considered suitable for learning in terms of learning efficacy and time. The validation dataset contained 25 randomly selected images for each of the three diagnoses. Still, overfitting occurred in this study most likely due to the small number of images included. Normally, 500-1000 images are prepared for each of the classes, they still, they managed to demonstrate that gynaecologists, who are not specialists in AI or machine learning can use deep learning in clinical practice.

Another work (Xia et al., 2017) also used TensorFlow but for facial expression recognition. An effective facial expression recognition model based on Inception-V3 model under TensorFlow is created. The transfer learning technique is used to retrain the Inception-v3, this is done to reduce the training time as much as possible. Image preprocessing is done,

in order to improve the efficiency of image classification, this preprocessing included, image format conversion and image clipping. For format conversion, the PIL library from Python was used. To verify the effectiveness of the proposed method it was compared to another method achieving the higher performance of 97%, the number of layers is also reduced compared to the other ones, reducing training time.

In the work of (Sharif Razavian et al., 2014) a look is taken into some features that sometimes might be overlooked when dealing with Convolutional Neural Networks (CNN), the off-the-shelf features. These more generic descriptors can be very powerful as recent results indicate. This is where (Sharif Razavian et al., 2014) comes in by adding to this mounting evidence. Reports on multiple experiments conducted for different recognition tasks, all using publicly available code and model, in this case, the OverFeat network, trained to perform object classification. The features were from OverFeat, as generic image representation, with the objective to deal with a diverse issue in object image classification, scene recognition, fine-grained recognition, attribute detection and image retrieval. These tasks and dataset were selected as they gradually swayed further away from the original objective and data the network was trained to solve. Consistent superior results were reported when compared to more tuned systems in all the visual classification task on multiple datasets. These results show that features gotten from deep learning with CNN's should be considered if not the primary candidate in most visual recognition tasks.

The work of (Hoff et al., 2018) uses where image classification is used for nutrition, dietary tracking making it easier to track what is eaten during the day by using pictures taken with a smartphone instead of having to manually introduce the data each time saving a lot of time and also making it easier for the end-user. The application proposed here would be able to relay basic dietary information about food to the user by using the camera that is integrated into their phones alongside image recognition software. By the user providing a picture of their meal to the application, it would then identify the food on their plate and provide basic dietary information about the food. The system also can identify the approximate serving size of each item on the plate, giving more information about the meal, emphasizing on the convenience, speed and being lightweight not to slow down the user's phone. The system uses a client-server architecture to separate the functionalities of the client program and the

server database. On the client-side, we have the user taking the pictures and sending them to the server for processing. The server examines these pictures using the image recognition model and returns the information for the food that was identified on the picture displaying it back to the user. When looking at the technologies they used, Python is one of them due to its extensive library's options for machine learning and image recognition, TensorFlow is one of the libraries. Image gathering and preprocessing played an important part here as it did in the works that were looked at before this one. Multiple images of a specific food from different angles to establish a solid dataset. The result was an application that could identify a few different foods as a proof of concept.

Another common use of image classification and one of the ones we hear the most about is using it to detect cancer, as we can see in the work of (Agarap and M., 2018). The work presents a comparison of six different machine learning algorithms by measuring the different results for the classification test accuracy, and the sensitivity and specificity values. The dataset consisted of features computed from a digitized image of a fine needle aspirate of a breast mass. These features describe the characteristics of the cell nuclei found in said image. This paper shows us the power of being machine learning and image classification when used for health, achieving very good results. With all exceeding 90% test accuracy on the classification task. With one even going higher with 99.04% test accuracy. Although this was used for breast cancer detection because it is using image classification this can also be applied for marine images, a new model with new images would have to be trained for this purpose.

In the work of (Leira et al., 2015) a paper about the creation of an unmanned aerial vehicle (UAV) used for automatic detection, classification and tracking of objects on the surface of the ocean. These UAVs can operate autonomously in dynamic and varied environments. And with focus on smaller and more powerful hardware capable of being embedded into different objects, making the UAVs onboard data processing more viable. Making it possible to process video feed onboard in real-time while at the same time capturing more video that will then be processed. In this case, algorithms for object identification and tracking. The paper discusses the development and implementation of the machine vision system for a low-cost fixed-wing UAV. The UAV has embedded thermal imaging cameras

for image capture, and onboard processing power for it to perform real-time object detection, classification and tracking of said object at the ocean surface. Results look promising with the test flight results showing it was able to detect 99.6% of objects of interest, capable of classifying 93.3% of the object, while also being capable of tracking 85% of the object types it was searching for.

The work of (Fefilatyev, 2008) presents a new technique for automatic detection of marine vehicles in images and video of the open sea. The source for the images and video is digital cameras or camcorders placed on buoys or stationery mounted in the harbour facilities. The system is intended to work autonomously, taking images or recording surrounding ocean surface and analyzing them for the presence of possible marine vehicles. With the goal to detect approximate window around the ship. Multiple datasets of still images were used for testing the algorithm developed here. A separate dataset of 30 videos with 10 seconds each was used for performance testing.

In the work of (Tariq et al., 2018) a neural network-based classifier to detect fake human faces created by both machines and humans. An ensemble of methods to detect fake images and employ pre-processing techniques to improve fake face image detection created by humans. This approach focuses on image contents for the classification and does not use the meta-data of the images.

In the work of (Beijbom, 2015) in which the objective is the automated annotation of coral reefs and how machine learning and computer vision can be used to help scientists with the bottleneck of making annotations by hand. Several aspects of data collection and annotations work were investigated. Methods for automated annotation are developed and compared to human annotators. These developed methods are implemented on the web at CoralNet(coralnet.ucsd.edu) and made publicly available.

The work of (Yasmin et al., 2014) makes it clear just how important it is to have the right images when they are used for expressing, sharing and interpreting information. With images normally, it is needed a specific image for a specific situation based on the contents of the image. The main goal of their work was to highlight what has been done and to provide a concept for image retrieval techniques, by looking for thousands of images, it becomes more and more complex as the number of classes increases, the number of images required

for each class increases. Better ways of searching are required making it possible to have the right visual content in a reasonable amount of time.

2.2 Object Detection Systems

The version of Retina Net that was looked for this thesis was the one that used Keras for its implementation (Lin et al., 2017b) looking into why one-stage detectors while having the potential to be faster, the accuracy has always trailed that of two-stage detectors. The authors discovered that the central cause for this was the class imbalances during the training of dense detectors. They proposed to address this by reshaping the cross-entropy loss in such a way that it down-weights the loss assigned to examples that are well classified. Focal loss changes the way training is done by training on a set of hard examples and preventing most of the easy negatives from devastating the detector during training. For evaluation, the known Retina Net was created. Results show that Retina Net can match the speeds of other one-stage detectors and at the same time is faster than the existing state of the art two-stage detectors of the time this paper was written.

One of the main things in RetinaNet is the Feature Pyramids on the work of (Lin et al., 2017a) focusses mainly on this aspect. Looking into an inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids. A top-down architecture with lateral connections was developed to build a high-level semantic feature map at all scales. This is called Feature Pyramid Network (FPN) it shows significant improvements as a generic feature extractor.

Looking at Retina Net, the work of (Li et al., 2018) focuses on image detection on video it is still worth mentioning. The same methods used in images can lead to blurring and low-resolution images. Here Recurrent Retina Net is proposed, as a flexible way for object detection in videos. In this paper, a backbone network is used for the feature mapping. The detection boxes are created from the information acquired from the mapping of features. Two subnets are used, with convolutional layers and convolutional LSTM (long short-term memory these are used for box regression and classification. Experiments were done here show that this approach improves detection accuracy while avoiding loss while keeping the model complexity good enough for real-time applications.

With the work of (Cui and Oztan, 2019) where RetinaNet is used for automated detection of firearms in cargo X-ray images. The training data used for this model was generated with threat image projection (TIP) this to make class imbalances less problematic, these are inherent to the x-ray security inspection, this eliminates the costly and tedious staged data collection. This model was tested on unseen weapons into unseen cargo images using TIP. The Retina Net detection model matched the detection accuracies of more traditional convolutional neural net firearm detectors, but also offering a more precise object detector and with faster detection speed.

In the work of (Mulfari et al., 2017) it is possible to see different ways of using TensorFlow, here the aim is to support tourism in an art city, by giving them a mobile application so they can better explore the artistic heritage within an urban setting by using the smartphone's camera. By retraining the preexisting model with new images specific for their problem and using the TensorFlow API for Android, a mobile application was implemented with the goal of processing stills captured by the device and recognize the artwork in real-time. TensorFlow computes the name of the dominant entity within a single photo and scoring it with a confidence level between 0 and 100. The system was able to recognize correctly 47 objects over 50 pictures, with the correct description. This is a good approach since we carry our smartphones everywhere we go. Giving us an extra way to interact with the world around us.

In the work of (Jo and Yoon, 2018), TensorFlow is used in a smart home setting, for energy efficiency purposes. It is divided into three models, on the first model which is designed to transmit data from the sensors to the server with minimal activity, it becomes aware of the situation before-hand and gathers the data. The second model processes the data and learns from it serving as a server. Receives the filtered and user-desired data sent as input. It learns by comparing the current status with the desired status, adjusting accordingly. TensorFlow is used in this model to increase the accuracy of the prediction value, allowing it to create an optimal status through predictions. With the final model being the one responsible for the readjustments dictated by the server. While some limitations still apply, limited by the scarcity of operating systems to integrate the various devices being the main one. A different brand with multiple systems all this leads to inefficient operating systems, excessive network

traffic and energy waste. To overcome this an integrated management system that connects IoT devices to each other needs to be built.

The work of (Abadi et al., 2016) where an explanation about TensorFlow of what it is, the capabilities of the system, also explaining the capabilities of the systems as well as the hardware it has to offer for the developers, the flexibility while also showing several real-world applications.

In the work of (Chakraborty et al., 2018), a semi-supervised technique is used to detect traffic incident trajectories from the cameras. Vehicle trajectories are identified from the camera using YOLOv3 classifier and Simple Online Real-time Tracking (SORT) are used for vehicle tracking. This classification is based on semi-supervised parameter estimation using maximum likelihood, this attempts to identify incident trajectories from the normal trajectories. Incident detection from the videos is divided into four tasks: Vehicle detection, vehicle tracking, trajectory formation and trajectory classification.

The work by (Jensen et al., 2017) where YOLO is used on the public LISA traffic light dataset to train a new model for traffic light detection for the future use in driving assistance systems, this dataset has a high number of annotated traffic lights, in various light and weather conditions. This model serves to better identify the current state of the traffic light to help prevent accidents. All three versions of YOLO were used and compared here, with YOLOv3 reaching the best results due to the improvements is as gotten over time handling low light situations better for example.

On the work of (Luo et al., 2019) the authors created a new training method by using as the base YOLOV3, one of the differences is that it preprocesses the data before training. The images are divided into equal parts using k-fold cross-validation. This makes it so we can train using all the data, by swapping the training and test images. This new model performed better than YOLOv3 in cases with class-imbalance. A new network structure is also proposed which was called Dense-YOLOv3. This new method alleviates overfitting. YOLOv3 was chosen due to how it performs in small object detection and the speed at which it runs, which is of 30 frames per second. In traffic, situation improvements were achieved about 4.1% of mAP with k-fold and 3% with Dense-YOLOv3.

The work of (Tumas and Serackis, 2018) another way in which to use YOLO, using it to create annotations for pedestrians using infrared images. The usage of the deep neural network-based approach has higher accuracy and speed when compared to traditional methods. YOLOv3 was integrated into the labelling software. Verification of pre-labelled results was around eleven times faster than manual labelling of every frame. The detector was able to detect 50 out of the 114 pedestrians that were manually annotated by the user in the test set. This was especially better when dealing with low-resolution images.

Another paper in which it is possible to see the flexibility of YOLO even though the version used was v2, is in the work of (Liu et al., 2019), a new network to deal with optical remote sensing images as its main focus is created based on version 2 of YOLO, which worked well for smaller objects but in larger objects like a football field, it does not work so well, this model also had multiple classes to be identified. The new network was compared to YOLO v2 and v3 and it performed better for the purpose of larger-scale object identification while keeping the performance of small-scale object detection. When compared to YOLOv2 the base for this new network on the DOTA dataset an improvement of 4.4% of the mean average precision was achieved without adding any new parameters.

The work of (Redmon and Farhadi, 2018) describes the improvements that the third version of Yolo has received while also comparing it to other object detection solutions. This helped with the decision making about which solution to use. The main improvement was the speed of YoloV3 while at the same time keeping the mean average precision within the same results as other slower solutions.

This section was used to look at different works that also used image classification, this gives an idea just how versatile image classification can be. Although some do not relate directly to this work, this is only because the models were trained with different purposes in mind.

2.3 Image Processing

Looking into OpenCV[1], starting with (Marengoni and Stringhini, 2011) it is possible to see what can be achieved by using this framework. This article works more like a tutorial, going through how OpenCV can be used for topics such as pattern recognition, computer vision, and parallel computer vision. The goal here is to teach how to design these types of applications using OpenCV as well as how to make use of the graphical processing unit available in most laptops and desktop PC. Things like Thresholds, Histograms, Filtering, Fourier Transform, Finding Edges on an image and basic segmentation, these are some of the basic functions available, as well as more advanced things and the topic of more importance to this project, such as pattern recognition using machine learning techniques like, K-means, Decision Trees, k nearest neighbour, boosting, principal component analysis and neural networks. It also goes into tracking and motion and parallels computer vision using CUDA and OpenCV.

Another paper about OpenCV but this one is a more traditional use case for this framework is (Chandan et al., 2018) in which a real-time object detection and tracking using deep learning and OpenCV are used both used. They go into some detail about the algorithms used for detection and tracking. Single Shot Detector (SSD) and MobileNets Algorithm, as well as the methods in which implementation occurred. Objects were detected using SSD in real-time scenarios, the model showed good results and could be utilized in specific scenarios to detect, track and respond to the targeted objects in a given video.

This section investigated what tools can be used for image processing, things like drawing the bounding boxes around resulting image detection image or removing colour ranges from images. Ways to manipulate an image using code and not external tools like photoshop.

2.4 External Processing Tools

An analysis of Google Collab[2] was done on the work of (Carneiro et al., 2018), as a tool for accelerating machine learning applications, they go into a detailed analysis of the

[1] https://opencv.org/
[2] https://colab.research.google.com

hardware resources, their performance and their limitations. Collab is used here for accelerating deep learning for computer vision and other GPU-centric applications. Three test-cases were created a tree-based combinatorial search and two computer vision cases, object detection and object localization. The hardware used in Google's platform is also compared with more mainstream workstations and a Linux server. The results showed that performance reached by Collab was equivalent to the performance reached by the testbeds, with similar resources. Making these cloud options effective to accelerate, not only deep learning but also other applications that are GPU-centric. Some of the limitations are the lack of CPU cores, making it far from enough to solve more demanding real-world problems, these tools are also not scalable. Other strengths and limitations are discussed in this paper.

Another tool that is an alternative to Google Colab is Azure[3] Machine Learning from Microsoft (Barga et al., 2015). In the work of (Pohekari, Rajshri, Vaibhav Patel, 2017), it is possible to see how Azure can be used, even though it was for cyber-attack detection and classification and not images. One of the main points of the paper is the use of cloud-based machine learning and how this can be used to detect and classify attack into a cloud-based machine learning platform. The results obtained here were promising showing how the cloud can be used to assist with machine learning.

Here the focus was to find tools to accelerate and facilitate the machine learning process, and using the cloud is a very good option for maintaining a fully local machine with all the hardware and software required.

2.5 User interaction & Machine Learning

On the work of (Amershi et al., 2014), the authors look at the users of the machine learning systems, while most progress in this was fueled mostly by advances in machine learning. They promote the user-first approach and demonstrate how it results in a better experience for the user and make learning these systems more effective. Several case studies that look at the impact of this user interactivity, ways in which existing systems fail accounting for the user, exploring ways for learning systems to interact with the user. The

[3] https://azure.microsoft.com

design process for more interactive systems should involve users. A look at some of the progress that was made in the field is given, and challenges faced in this field while moving forward.

In the work (Cai et al., 2019b) a somewhat new approach to machine learning, that uses the more traditional method of machine learning in which the machine does indeed to all the calculations that are expected of it, but the user has more control. Making it a more precise way for more specific cases, the professional can adapt the model for a more specific case, this did not reduce the diagnostic accuracy, it increased the trust in the system making the end-user more likely to use the system again because it did not take away their decision-making ability. New strategies were also noted when using the refinement tools, re-purposing these to test and understand the algorithm beneath, and separating machine learning errors from their own errors. This approach shows us how other machine learning-based systems can work together with the user and not replace them.

A work that involves the creation of a user interface for machine learning is that of (Cai et al., 2019a) where two methods are proposed and evaluated of example-based explanations in the visual domain, with normative explanations and comparative explanations. The effects were investigated by deploying them to 1150 users on QuickDraw when it failed to identify the drawing, the users received a normative explanation, they felt like they had a better understanding of the system and perceived the system to have a higher capacity. The comparative explanations did not always improve the perceptions the users had of the system, because they sometimes expose the algorithm limitations and may lead to a surprise. They found that examples explaining the behaviour of the algorithm, pointing to relative advantages and disadvantages of using different examples.

A work of (Hoff and Bashir, 2015) with the objective of systematically reviewing empirical research on factors that influence the trust in automation to present a three-layered trust model. The existing research guide human-automation interaction, this is centered around trust, this trust determines how willing the human operators are to rely on automation. This systematic review of the research on trust in automation the paper was eligible only if they reported the results of the experiments, in which humans interacted with an automated system. Their analysis showed the three layers of trust, dispositional trust, situational trust

and learned trust. Design recommendations were given to increase the trust in automation and conditions were identified that can influence the relationship between trust and reliance. This model's new way of conceptualizing trust in automation. This structure can be applied to guide future research and develop training interventions and procedures that encourage trust.

The importance of trust in machine learning is again explored in the work of (Ribeiro et al., 2016), understanding why the machine predicted what it did is, quite important when assessing trust, which in turn is important if action will be taken, based on that prediction or should a new model be used, or not. Understanding this provides a way to understand the model, that can transform a bad model or prediction into a more trustworthy one. Here an explanation technique is used that explains the predictions in a way that is easier to interpret, this is done by learning an interpretable model, locally, around the prediction. The way of explaining the models is also presented here by using representative individual predictions and their explanations, framing the task as a submodular optimization problem. The flexibility of these models is also demonstrated by using different models like text and image classification. And the utility of these explanations on various scenarios that require trust when deciding.

The current way of building machine learning systems requires users with great knowledge of machine learning, limiting the number of systems that can be created. Leading to issues with demand and the ability to build these systems. With the work of (Simard et al., 2017), fundamental machine teaching principles are given, they describe that by decoupling knowledge about algorithms used in machine learning from the process of teaching it is possible to accelerate innovation and empower new uses for models in machine learning. They seek to apply three ideas with this, the first one concerns decomposition and modularity so it can scale with complexity. The second one, a standardization of programming languages, while not proposing a specific language but listing important knowledge channels available to the teacher. With the final one being processing discipline, including separating concerns and building of standard tools and libraries.

While not directly related to machine learning the work of (Vaccaro et al., 2018) looks at how the users interact with something that they might not have complete control over.

Thing like control settings which are one of the ways that users use to adjust how things like news feeds get organized. They studied how users engage with difficult-to-validate controls, using two studies using an experimental system, and discovered that these controls setting can work as placebos. Users felt more satisfied when controls were present on their feed, working or not. How people engage in sensemaking around these controls settings was looked at and found that users often take responsibility for broken expectations, in both real and controls that function in a random way. And finally, how users would use their feeds in the wild. The use of existing controls had little impact on how stratified the users were with their feed, turning often to improvised solutions, things like quickly scrolling through the feed to see what they want.

A work that looks into the interface aspect of machine learning is that of (Fails and Olsen, 2003) here a way of creating a more interactive interface is looked at. An interactive machine learning model which gives users the ability to train, classify/view and correct classifications. The concept and its implementation are discussed here. As well as compared with more standard other machine learning models. They also describe a tool/technique called image processing with crayons for creating new interfaces based on cameras using a painting metaphor. This technique uses the same notions of interactive machine learning.

Another work that also focuses on the interaction of the user with a machine learning system is that of (Amershi et al., 2012). Here a specific system is presented, called ReGroup an interactive machine learning system that helps people create custom groups in online social networks. As people are added it iteratively learns a probabilistic model of membership specific to the group. It then suggests more members and group characteristics for filtering. Applying this interactive machine learning system in a social network environment, introduced several challenges for designing effective was of interactions between the user and machine learning. These challenges are identified and discussed as well as how they were addressed in ReGroup

A paper that also looks into the trust aspect of machine learning systems is (Kizilcec and F., 2016). The focus here is to discover how a transparent design of algorithmic interfaces can promote awareness and increase trust. The way in which they investigated this was using a two-stage process of how transparency can affect trust was created based on theories of

information processing and procedural justice. In an online field experiment, a system with three levels of transparency was tested in a peer assessment context. Users that had their expectations violated lost trust in the system unless the grading algorithm was explained to them making it more transparent how the system works. But providing too much information would decrease this trust. The attitudes of the ones whose expectations were met did vary with transparency. They go into more detail of the results gathered but the main point shown by the paper is that designing a system for trust requires balanced interface transparency, not too much and not too little.

2.6 Human influence on the oceans

Starting with the work of (Nagelkerken and Munday, 2016) in which the authors look into the complex interactions between organisms and their environment and interactions with other species. And how humans are changing these marine environments through pollution. The author's review provides a synthesis of the consequences that changes in behaviour could have for these marine ecosystems. And without considering how climate change affects the behaviour if species we are limiting our ability to see the impacts of changes in the ocean.

The work of (McCauley et al., 2015) on marine defaunation provides us with yet another view of how we are affecting our oceans and its wildlife, altering the functioning and provisioning of services in every ocean. Current trends suggest that marine defaunation rates will only increase as humans use of the oceans industrializes. Habitat degradation is likely to increase. Proactive intervention is needed to avert a marine defaunation disaster.

3 Prototype

This chapter describes the developed prototype, its architecture, the frameworks and APIs used as well as the workflow to integrate image classification algorithms in an end-user application. Two different models were trained using YOLOv3 and Keras RetinaNet. These models where used in image detection to identify whales and dolphins in images. Image prediction using YOLOv3 was also trained, where it gives us the probability of having a whale or a turtle in an image. The difference between image detection and prediction is that while detection searches for a specific object in an image and marks that object this also work for multiple objects, in image prediction it takes the whole image and give us the probability of an object being contained in that image but not marking it. These differences are described in the section below.

3.1 System Architecture

Here the system architecture of the prototype designed will be explained, first showing a diagram then explaining the said diagram.

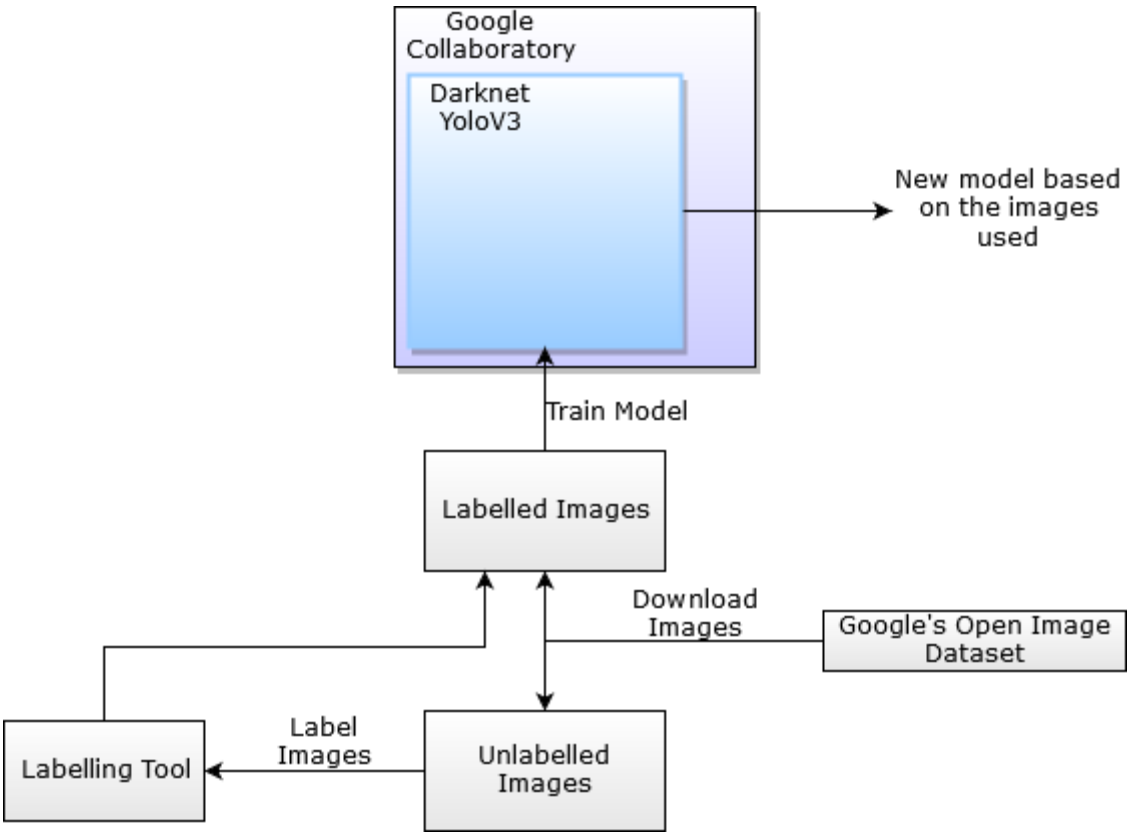


Figure 1: Model Training

In Figure 1, it is shown what must be done to train the model that will be used in the application for the image detection part. Firstly, the images must be gathered, in this case, the Open Image Dataset was used to download the images. These images can be downloaded with the labels or without. If they are downloaded without the label these images must be labelled before being used. After having the labelled images, it is possible to train the image detection model using Darknet YoloV3 or Keras Retina Net. This model is then used in the application to identify the object in the given image.

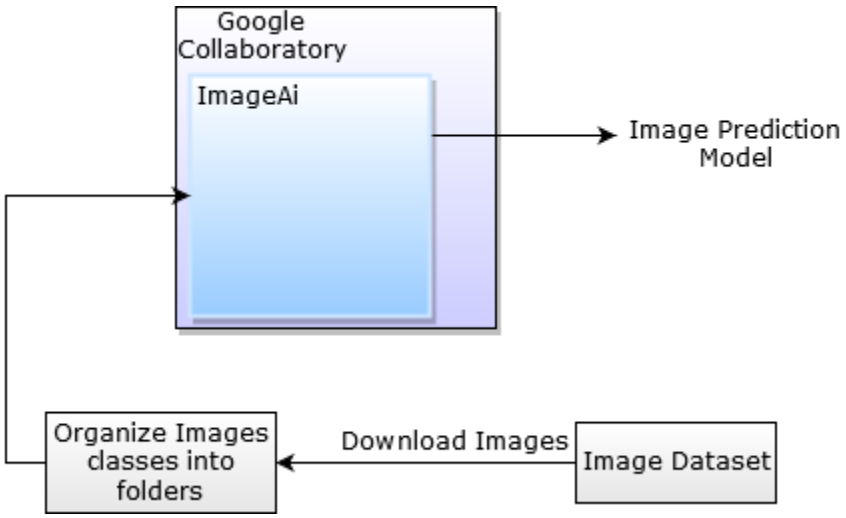


Figure 2: Image Prediction Diagram

For the image prediction model, there is no need to label images individually but they must be separated by folder, what this means is that an image from class A, must be in a different folder than an image of class B. ImageAi uses folder hierarchy to correctly identify different classes and then uses the images in these distinct folders to train the model. This model just like the previous one is then used in the application in the image detection option to classify the images that are provided by the user.

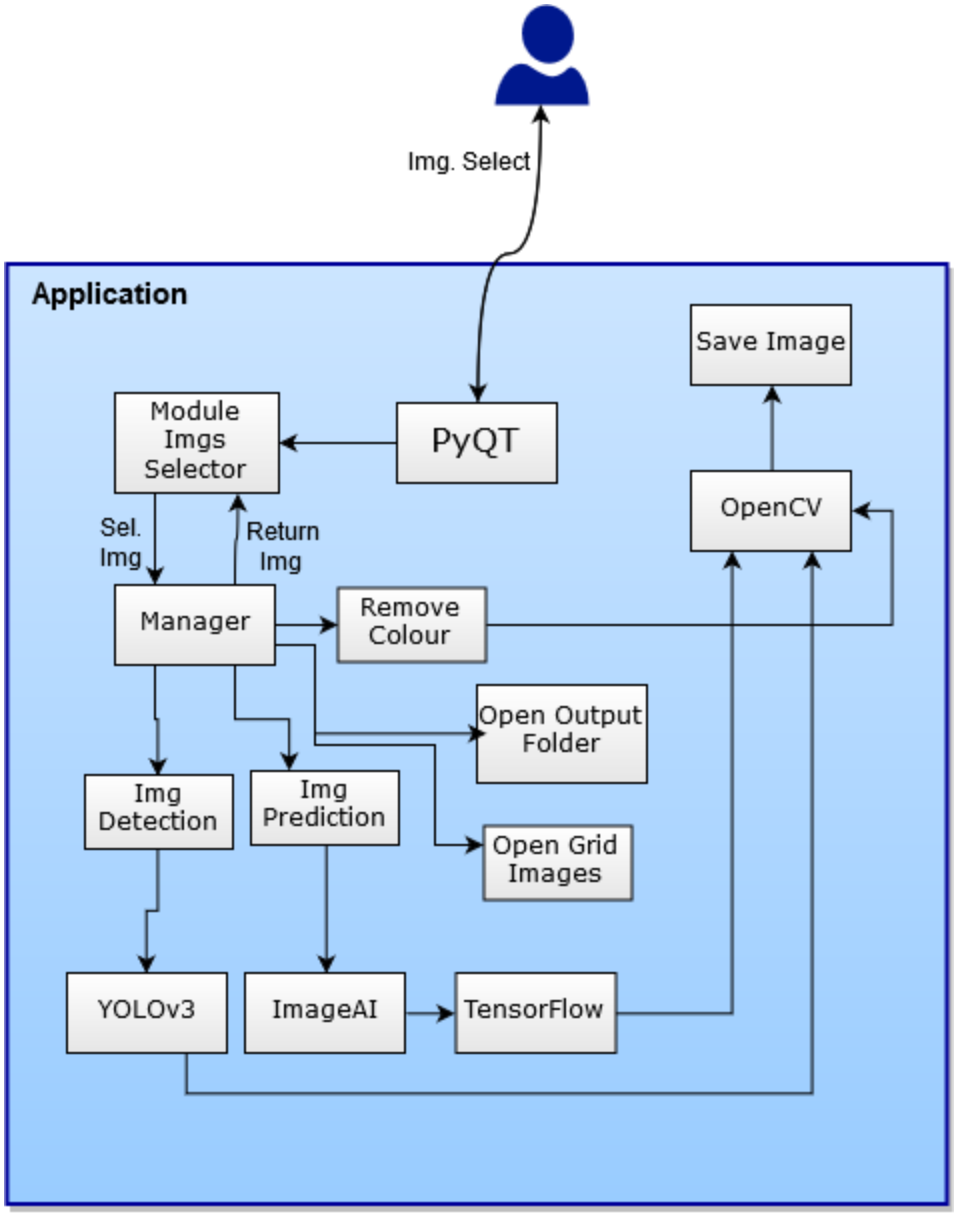


Figure 3: System Architecture

In the GUI, the user must choose what action she wants to take, be it image prediction, detection, remove the Green and Blue color from the image. After selecting the course of action, the user then must select the folder that contains the images that will be processed. The manager then takes these images and saves their location. After having the locations of the images that are going be used, it then either calls the ImageAi API which contains TensorFlow to predict what the image is based on the trained model to identify either whales or turtles. It also can use the newly trained YOLOv3 model to detect custom objects on the images. After identifying the objects OpenCV is used to draw the bounding boxes before

saving the new images. Finally, it can also use OpenCV to remove green and blue from the stored images, this color removal could in some cases improve with detection by highlighting some objects over others, for example removing the blue range from an aerial image of the ocean, highlighting objects on the ocean. After any of these three actions, the location of where the images were saved too is sent back into the GUI letting the user know where the newly edited images are saved too. The user can also choose to open the output folder using the button assigned for this task, it then uses the file explorer of the operating system to open the output folder or the user can also open a grid that contains the images that have been processed.

3.2 ImageAi

A framework designed and maintained by DeepQuest AI, a python library that was built to empower developers to independently build applications and systems with a self-contained Computer Vision capability. ImageAI supports various Machine learning algorithms for image recognition, object detection, custom object detection, video object detection, video object tracking, custom image recognition training and custom prediction. The focus of this thesis will be on custom prediction and object detection. This framework provides an API to train a new image recognition model on a new image dataset for custom use cases, it provides an implementation to integrate and deploy the custom image recognition model. It also provides us with an API to detect, locate and identify 80 of the most common objects in everyday life in each picture using pre-trained models that were trained on the COCO Dataset. The implementations for the models provided include RetinaNet, YOLOv3 and TinyYOLOv3.

3.3 TensorFlow

TensorFlow is an open-source library for numerical computation and machine learning. It bundles together many machine learning and deep learning models and algorithms. It uses python for the front-end API for building applications with the framework, while it executes those applications in high-performance C++. TensorFlow can train and run

deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing. Interaction with TensorFlow is all being made through the ImageAI framework, making it easier and faster to implement the solution presented in this thesis.

3.4 PyQt

PyQt is a set of Python v2 and v3 bindings for The Qt Company's Qt applications framework. Qt is mostly known as a graphical user interface toolkit, but it is capable of much more but in this prototype, PyQt was used to give us the ability to use Qt with Python instead of the normal C++. This was done to keep the language the same because of the use of ImageAI, a framework also written in Python. Keeping the project with a single language. The use of drag and drop with the GUI makes for quick designing instead of having to manually code the position of each element, it is possible to just drag and drop where you need the element to be. Qt is also a reliable platform with constant updates, while the ease for cross-platform deployment being able to deploy one app with just one version on multiple platforms is always a good option to have while this might not be used having it has an option is always good.

3.5 OpenCV

Open Source Computer Vision Library released in a BSD license making it free for both academic and commercial use. It uses C++, Python, and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and a strong focus on real-time applications. The library can take advantage of multi-core processing. Enabled with OpenCL, it takes advantage of the hardware acceleration of the underlying heterogeneous computing platform. OpenCV was chosen over other solutions like say Matlab, as it was more specific it was made for image processing while Matlab is more generic and I did not work with Matlab before, so it made the coding approach that OpencCV provided a simpler one due to my background. It is also faster, and it does not have anything standing in the way of running the code making it faster at processing this

same code. Another point in favour of OpenCV is the system requirements, again no specific GUI, like in the case of Matlab making it more efficient memory-wise.

3.6 Google Collaboratory

It is a tool for machine learning used in education and research, a Jupyter notebook environment which requires no setup to use, giving us access to greater computing power, it is completely free to use. It supports Python 2.7 and Python 3.6. The code executed is these notebooks run in a virtual machine dedicated to the user's Google account. These virtual machines are recycled when idle and have a maximum lifetime enforced by the system of 12 hours. The main reason for the use of Google Collaboratory is access to the hardware it provides:

- Tesla K80 GPU with the 2496 CUDA and the 12GB GDDR5 VRAM
- CPU – 1xsingle core hyperthreaded Xeon Processors @2.3Ghz
- RAM – Around 12.6 GB
- Disk – 320 GB

This hardware, especially the GPU that it provides really speeds up the training process.

3.7 Google's Open Images Dataset V4

All the images used here were taken from Open Images Dataset V4, this is an image database created and initially released by Google in 2016. It is mostly used for building Data Science and Artificial Intelligence algorithms. With the key use being Machine Learning. Open Images V4 contains 15.4 million bounding boxes for 600 categories on 1.9 million images, this makes it the largest existing dataset with the annotations for the object location. These images and their bounding box information can be used to train smaller datasets based on the needs of the said dataset without the need to download every single image and the accompanying bounding box information. This is where the images were downloaded, images related to whales and turtles to create a prediction model, this model does not use the bounding box. After training the computer generates a probability of the image either being an image of a whale or an image of a turtle. The images for the detection case were also

downloaded from here, these images already came marked with the bounding box information required to train a new YOLOv3 model that would identify Dolphins or Whales.

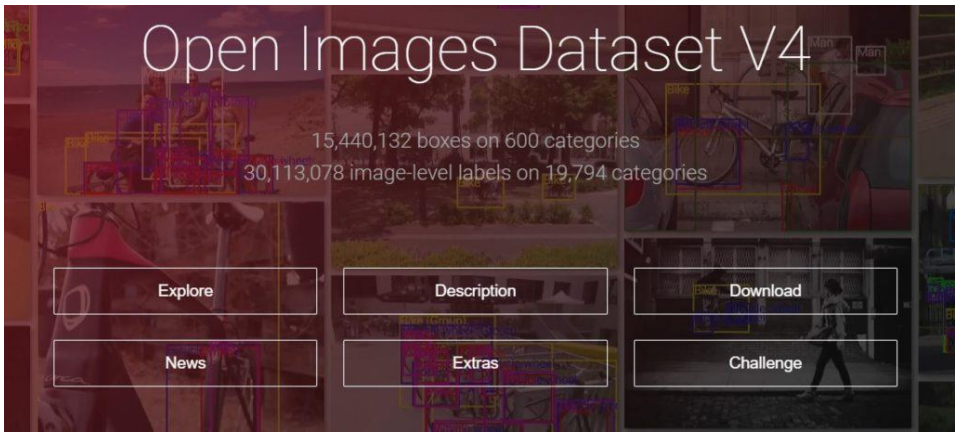


Figure 4: Home page of Open Images Dataset V4

3.8 YOLOv3

YOLO (You Only Look Once), a network for object detection. This task consists of locating in an image where certain objects are and classifying these same objects. Previous methods for this same task, like R-CNN (Region- Convolutional Neural Network) and variations, use pipelines to perform this task in multiple steps. This can be slow to run and hard to optimize, each individual component must be trained separately. While YOLO does it all with a single neural network. It is trained with the COCO dataset. The dataset contains photos of 91 object types easily recognizable by anyone, a total of 2.5 million labelled instances in 328k images. (Lin et al., 2014). The input image is divided into a $S\ x\ S$ grix of cells. And for each object that is present in this image, one of the grids is responsible for predicting it. The cell where the center of the object falls into. Each of these grids also predicts B bounding boxes and C class probabilities. Firstly, the bounding box has 5 components: $x, y, w, h, confidence$. The (x, y) coordinates represent the center of the box, relative to the grid cells. These values are normalized to fall between 0 and 1. The (w, h) which are the box dimensions are also normalized to $[0,1]$ relative to the image.

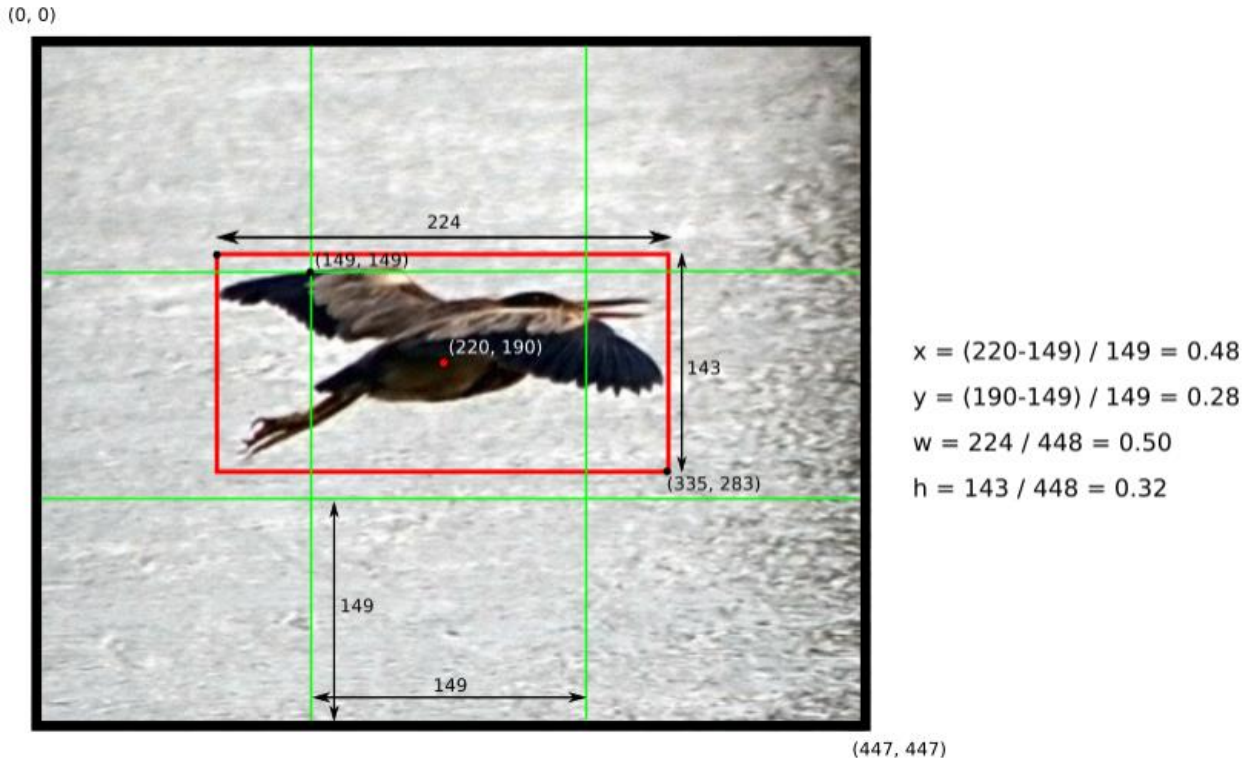


Figure 5: Image example of YOLO

The last component in the bounding box prediction is the confidence score, which reflects the presence or absence of an object of any class. The class probabilities also need to be predicted, $\Pr(Class(i)|\, Object)$. The probability is conditioned on the grid that contains one object. What this means is if no object is present on the grid, the loss function will not penalize the grid for a wrong class prediction. Only one set of class probabilities per cell are predicted, regardless of the number of boxes B. Making $SxSxC$ class probabilities in total.

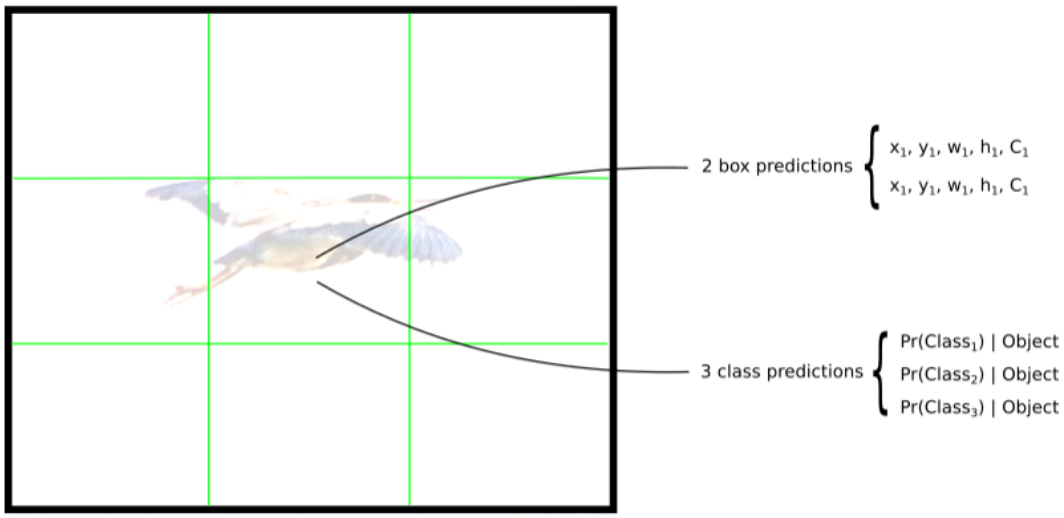


Figure 6: YOLO grid image

Each grid cell makes B bounding box predictions and C class predictions (S = 3, B = 2, C = 3), as shown in the example of Figure 6. The network structure is a convolutional and max-pooling layer, followed by two fully connected layers in the end. The last part of YOLO is the loss function, which can be divided into three parts. The first part of the loss function computes the loss that is related to the bounding box that predicts the position (x, y). It Computes a sum over each box predictor of each grid cell. With each object being defined as:

- 1, if there is an object present in grid cell i and the jth bounding box predictor is responsible for that prediction
- 0, otherwise

YOLO predicts multiple boxes per cell. But during training, only one bounding box predictor is responsible for each object. One predictor is assigned as "responsible" for predicting an object based on what prediction has the highest current IOU (Intersection over Union, an evaluation metric used to measure the accuracy of an object detector on a dataset) with the ground truth. Other terms like (x, y) are the box position and $(\hat{x}, \hat{y})$ are the actual position from the training data, both these terms are used in the equation in the background. The second part of the loss function, it is related to the width and height. Small deviations in large boxes matter less than in small boxes. To address this a square root is used to predict the height and width instead to the direct values. The third part is where it computes the loss that is associated with the confidence score for each of the box predictor and the intersection over

union of the predicted box with the ground truth. Just like before, object equals one when an object is present in the cell, 0 if there is no object. Weights part of the loss functions are necessary to increase model stability. The highest penalty is used to the coordinate predictions and lowest for confidence predictions when no objects are present. Now for the last part of the loss function. In the final part a normal sum-squared error for classification, except for the object term. This is used to avoid penalizing the classification error when no object is present on the cell.

The way YOLO does the training is the following:

1. Pre train the first 20 convolutional layers using the ImageNet 1000 dataset, with inputs with the size of 224x224
2. Increase the input resolution to 448x448
3. Train the full network for around 135 epochs using a batch size of 64, momentum of 0.9 and decay of 0.0005
4. Learning rate, for the first epochs, is slowly raised from 0.001 to 0.01. For about 75 epochs then starts decreasing
5. Use data augmentation with random scaling and translations, also randomly adjusting exposure and saturation.

This procedure, as well as everything else related to YOLO, is described with much greater depth in (Redmon et al., 2015). Now looking at the YOLOv3 performance when compared to Keras Retina Net, YOLOv3 got a comparable mAP@0.5 (mean average precision) with a faster inference time. While overall mAP performance is dropped significantly. While still being on par with SSD (Single Shot Detector) variants. YOLOv3 is better than SSD and has similar performance as DSSD (Deconvolutional Single Shot Detector) as it is possible to see in (Redmon and Farhadi, 2018).

3.9 Keras RetinaNet

This section explaining how RetinaNet works were written, based on (Zenggyu, 2018).

RetinaNet is a composite network composed of three networks:

1. Firstly, a backbone network, the Feature Pyramid, built on top of ResNet this is responsible for computing convolutional feature maps of an image.

2. A subnetwork responsible that performs object classification using the backbone output
3. Another subnetwork that performs bounding box regression using the backbone output

Backbone Network:

This enables the network to take an image and the outputs are proportionally sized feature maps at multiple levels in the feature pyramid. The Feature Pyramid Network (FPN) has two pathways with lateral connections. The Bottom-up pathway is done by choosing the last feature map of each group of consecutive layers that output feature maps of the same scale these feature maps are used as a foundation of the feature pyramid.

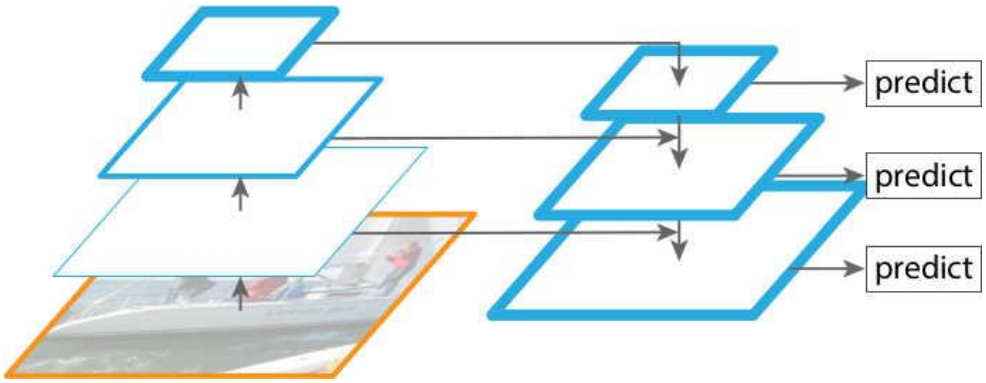


Figure 7: Bottom-up with lateral connections

The Top-down pathway with lateral connections uses the nearest neighbour upsampling, the last feature map is expanded to the same scale as the second to last feature map. These two are then merged to form a new feature map, this is done until each feature map from the bottom-up pathway has a new feature map connected with the lateral connections. This can be seen in Figure 7. The FPN can take an image of arbitrary size and output proportionally sized feature maps at multiple levels. The feature maps at higher levels contain grids, these cover larger regions of the image, making it better for detecting bigger objects, while the grid cells at lower levels are better at dealing with smaller objects. With the top-down pathway and the lateral connections, this makes every level of the feature map both semantically and spatially strong, being able to be used independently to make predictions, making the model scale-invariant and providing better performance.

Classification subnet:

This is a fully convolutional network (FCN) attached to each of the FPN levels. The subnet consists of four three by three convolutional layers with 256 filters. With another three by three convolutional layer with $K \times A$ filters are applied. The output shape of the feature map would be (**W**, **H**, **KA**), with **W** and **H** proportional to the width and height of the input feature map, with **K** and **A** being the number of object class and anchor box.

Regression Subnet:

These are also attached to each feature map of the FPN. The design is the same as the classification subnet with changes to the last three by three convolutional layers with **4A** filters. This would make its shape (**W**, **H**, **4A**)

A closer look at the subnets:

When an image is given the width and height the feature map output by the FPN is three by three, for each of the grid cells RetinaNet defines **A = 9** boxes called anchor boxes covering an area in the image. Each of these anchor boxes detects the existence of objects from the **K** classes in the area it covers, each box corresponds to **K** numbers that indicate the class probabilities. Because there are **A** bounding boxes for each grid, the output feature map of the classification subnet has **KA** channels. These boxes are also responsible for detecting the shape/size of the objects. The regression subnetwork is responsible for this, it outputs 4 numbers for each anchor box, this predicts the relative offset between the anchor and ground-truth box. That is why the output feature map of the regression subnet has **4A** channels.

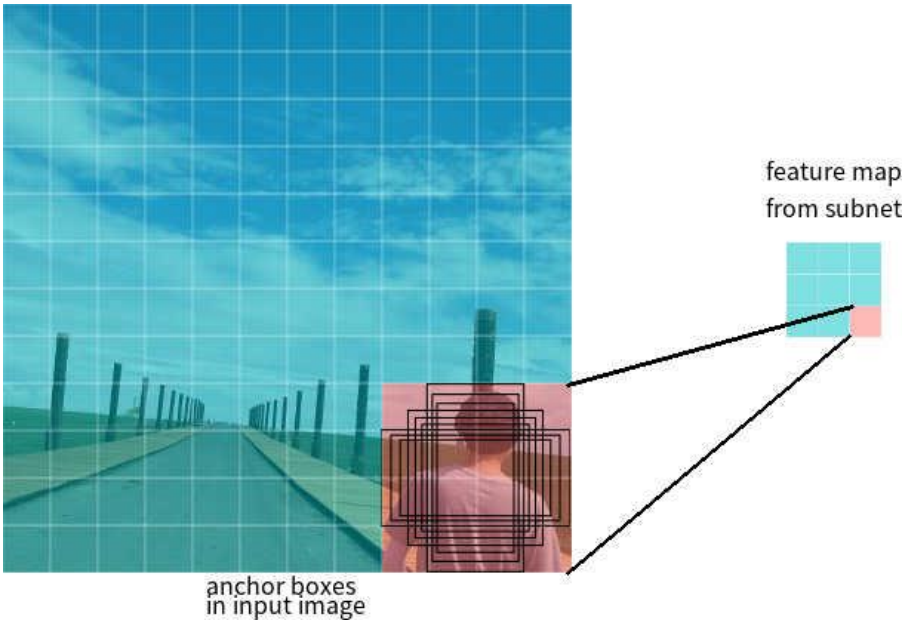


Figure 8: Anchor box & feature map example

Loss Function:

A multi-task loss that contains two terms one for localization (L_{loc}) and another for classification (L_{cls}). This function is written as:

$$L = \lambda L_{loc} + L_{cls}$$

Equation 1: Loss Function

Where λ is the parameter that controls the balance between the two.

Regression Loss:

This is calculated based on the match of the anchor box and ground-truth box, with matching pairs. For each anchor, with a match, the subnet predicts four number. The first two specify the offset between the centre of the anchor and ground- truth, with the last two the offset between width/ height of the anchor and ground-truth.

Classification Loss:

This is a variant of the focal loss, which is the most innovative design of the detector. Now to briefly explain how the function works, it has **K** which is the number of classes, they equal 1 if the ground-truth belongs to the i-th class and 0 otherwise. There is also the predicted

probability for the i-th class; $y \in (0, +\infty)$ is the focusing parameter; $\alpha_i \in [0,1]$ is a weighting parameter for the i-th class.

Prediction:

Now to look at how RetinaNet generates the predictions after it has been trained, for each input image there multiple anchor boxes from all FPN levels, for each of these anchor boxes the classification subnet predicts K numbers each indicating the probability distribution of classes, meanwhile the regression subnet predicts four numbers that indicate the difference between each anchor box and corresponding bounding box.

At most 1k anchor boxes are selected these with the highest confidence score from each FPN level, there is a threshold to limit the amount of anchor set at 0.05. Only these anchors are included in the next steps.

An object in the image can now be identified by multiple anchor boxes. Non-maximum suppression (NMS) is applied to every class independently, which is done to remove redundancies. This is done by choosing an anchor with the highest confidence and removing any overlapping anchors with an intersection-over-union greater than 0.5.

For the last stage, the regression subnet gives offset predictions these can then be used to further refine the anchors which are then used to get the bounding box predictions.

3.10 Training the Models

In this work, the YOLOv3 fork by (AlexeyAB, n.d.) was used. Initially, installing the Windows version of Darknet was tried, following the steps described, problems arose with missing DLL's, after getting these missing files and installing CUDA, which is Nvidia's way to allowing the user their own Graphics Processing Unit (GPU) to train the model. The Windows installation ended up not working, it was then tried using the Linux bash on the windows platform, the installation process becomes a lot easier and straight forward in Linux, but unfortunately, it is not possible to use the GPU with this. A basic dataset for testing purposes was used with aerial drone images of Whales and Dolphin this dataset had 88 images, after marking the images with the required bounding boxes so the computer could identify the objects in the images then proceeded to train this small dataset, this was done to

test how the hardware would fair. The hardware is an older Laptop with an i5-3230M running at 2.60GHZ with 8GBs of ram and a GEFORCE GT 635M with 2GB of dedicated memory. Being limited to the CPU each interaction took around 2 hours each and after 10 interactions the system ran out of memory, this makes training a new model with a new dataset not viable on a local machine due to hardware limitations.

Figure 9: Results of the failed train

In Figure 9 the results of the failed training, which is explained above and we can see this small number of images and interactions does not provide any usable results.

For context, a model with a single class, requires at least 1000 images each with their bounding boxes, with a total of 5200 interactions. The label was created by using the application from (João Cartucho, n.d.) an easy and fast way to go through the images and adding labels to each image, after the labelling process a text file is created with the correct format used by YOLOv3 for Retina Net a different tool was used.

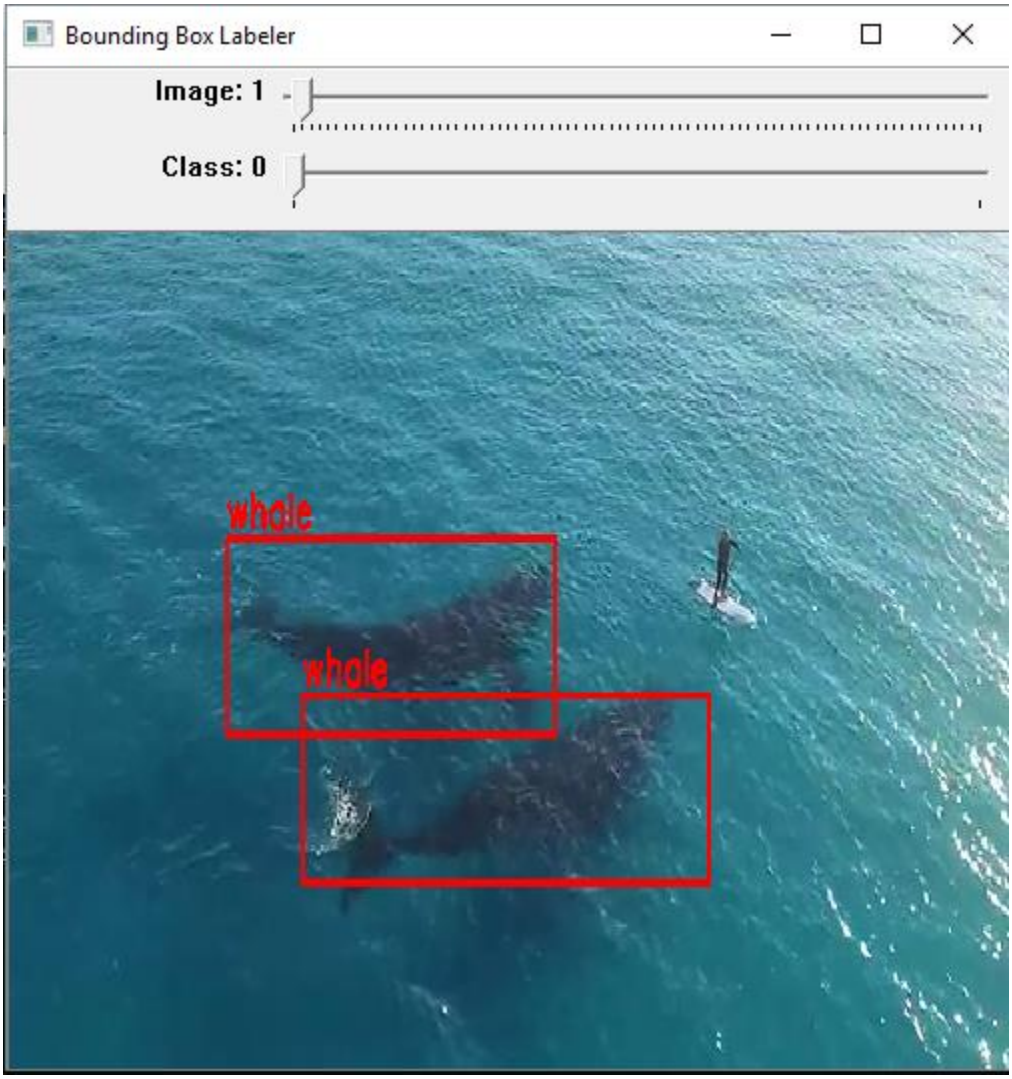


Figure 10: OpenLabeling & Bounding boxes

Training locally was not an option due to the hardware limitations; it was then that the use of Google Collaboratory was considered in part due to the access to the GPU provided on the platform to speed up the training process to an acceptable time frame. After managing to set it up to work with (AlexeyAB, n.d.) and all its requisites. It is then possible to start the training process, this becomes a lot faster, an initial test was done with a training model to identify snowman, the training was interrupted after a few interactions as an experiment. Although 1000 interactions were enough to produce an acceptable model for snowman identification. Bellow we can see in Figure 11 that the model correctly identifies a snowman with a probability of 0.63. Making Google Collab a viable and faster way to train the model when compared to a local solution.

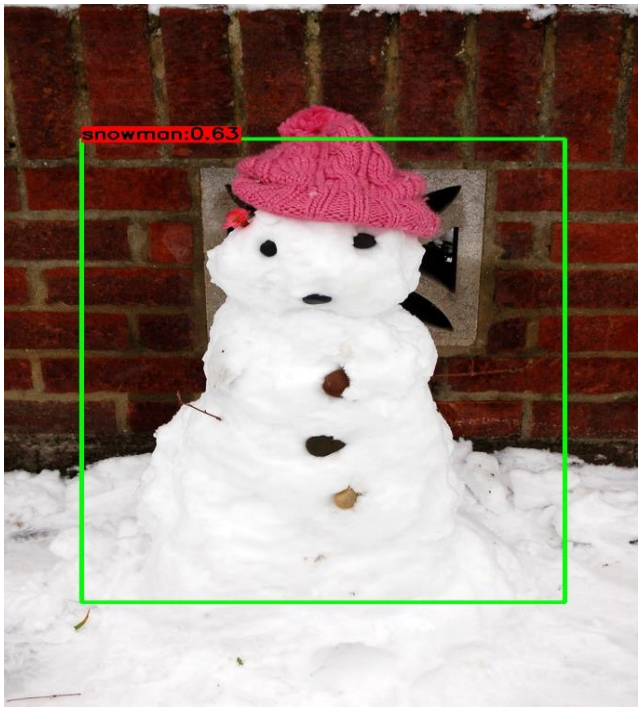


Figure 11: Snowman testing

3.10.1 Image Detection Model – YOLOV3

To train a model for image detection the images are the most important part of the process, the model used to train also has an impact on the result but the number of images, as well as their quality, has a much greater impact on the model's final accuracy. To start this training process first the images must be gathered, in this case, all the images used here are from Googles Open Images Dataset. A small python script was used to download the images and the labels associated with these images, which contain the information of the bounding boxes and the classes on each image, this was only used for YOLOv3 as the format of the annotations used in Retina Net is different. A total of 1329 images were downloaded from Open Images Dataset, split somewhat evenly between the two classes, while this number is not the ideal number of images good results were still achieved, these were the images available on the platform. This image bank was used to help with the process of image gathering and labelling, making it faster labelling just 100 images using Open Labeling tool, for example, can take up to one hour depending on the number of classes. The labels have

the same name as the image file, this is done so that during training it knows what label file belongs to what image. The labels have the following format:

- <Object-Class-Id> <Center - X> <Center - Y> < Width> <Height>

If multiple objects are present in an image, they are separated by each row of the file, each row equals one object. After having the images and the labels, these must split into training and testing groups. The training group contains the images the model will be trained with, while the testing group contains the images that will be used to test the model's accuracy, for the best results the same image must not be in both groups. The splitting was done using a python script that does this split and does it by creating two text files, one called train.txt and another one called test.txt. These files contain the path to the images, in this case, 90% of the images are in the training file with the other 10% on the test file. After splitting, some files must be modified specifically the yolov3.cfg file, the following lines must be modified:

- Line batch changed to $batch = 64$
- Subdivisions changed to 16
- Max_batches to ($nº\ of\ classes * 2000$) – Max batch used 4000
- Steps changed to 80% and 90% of max_batches – Steps used were 3200, 3600
- Classes line must be changed to the number of objects in the three yolo layers
- Change to $filters = (classes + 5) * 3$ in the three convolutional before each yolo layer, this case filters = (2+5) * 3 = 21

After these changes a file must be created for example $obj.names$ with objects names each in a new line. Another file also must be created $obj.data$ containing the following:

```
Classes = (nº of classes)
Train = /path/to/train.txt
Valid = /path/to/test.txt
Names = /path to/obj.names
Backup = /path/where/to/save/weight/files
```

Both these files must be placed into the $build\darknet\x64\data\$ folder, the image files of the objects in a folder also in the $build\darknet\x64\data\obj(for\ example)$ the

labels should be placed in the same folder alongside the images. The pre-trained weights must also be downloaded and placed in the directory $build\darknet\x64$. After these changes it is possible to start the training process by running a Linux command:

$$\$./darknet.exe\ detector\ train\ obj\obj.data\ yolo-obj.cfg\ darknet53.conv.74$$

The paths will need some adjusting, after running this command the training starts. The training was done using Google Collab to speed up the process. These steps can be viewed on the darknet branch on GitHub (AlexeyAB, n.d.).

Some other lines can be changed in the .cfg file to further customize the training. It is possible to specify the input image size and the number of channels, by changing the width, height and channel values. For example, increasing image values to 608x608 might improve results but this would increase the training time, the default channel of three indicates the processing is going to use the 3-channel RGB (Red Green and Blue). Other parameters like the momentum which is used to penalize large weight changes between iterations, this is done to try and avoid overfitting. The decay parameter controls the penalty term. The learning rate parameter controls how the machine learns based on the current batch of data. When starting at zero information, the learning rate needs to be high, but as training goes on the neural network sees a lot of data and the weight need to be changed less aggressively. Learning rate needs to be decreased over time this was done by the steps parameter which is equal to policy. For example, if it starts from 0.001 it can remain constant for several iterations, and then it will multiply by the scale parameter to get the new learning rate. The parameter that controls the number of iterations is the max batches parameter, for n-classes, it recommended to update this value to $2000 * n\ batches$.

3.10.2 Image Detection Model – Keras RetinaNet

In order to train a Keras RetinaNet model, a different tool to label images was used. This model uses a different format compared to YOLOv3, the tool used for this was (tzutalin, n.d.) this tool is more complete than the one used for YOLOv3. The images used for this model were the same ones used on the YOLOv3 model, the previously stated 1329 images split between the two classes. After having the images labelled a script needs to be run to convert the annotations into the format used by Keras RetinaNet. Just like with YOLO Google

Collaboratory was used for training the model. A pre-trained model was used to help with the training, this is better than training from scratch.

3.10.3 Image Prediction Model

An image prediction model was trained using images from Open Images Dataset V4 a total of 1828 images divided into two categories these categories being:

- Whale with a total of 768 images, 640 for training and 128 for testing
- Turtle with a total of 1060 images, 858 for training and 202 for testing

Figure 12: Example image of the whale category

Figure 13: Image example of the Turtle category

Figure 12 and Figure 13 are examples of the images used in the training and testing of this image prediction model.

Training this model with two categories was done using the ImageAI framework, this does not single out an object in the image, it takes the whole image and then gives us a probability of it being one of the categories in this case either a whale or a turtle. The accuracy achieved with the testing set of images was 0.930 (values range between 0 and 1). The model took about five hours to train using Google Collab, this was done to speed up the training process due to the Tesla K80 GPU from Nvidia. Some of the limitations of this prediction model are that if we give it an image without a turtle of whale it will still try and give us the probability of there being either one in the image as we can see on below on Figure 14.

Figure 14: Image Prediction without one of the classes.

Even though there is neither a whale nor a turtle in Figure 14 it still is calculating the probabilities of having either one of the classes as we can see in the upper left corner of the image. Since the training images for the whale class contained more images that contain the ocean it gives us a 72% probability of this image containing a whale and just 28% of it containing a turtle.

3.11 Prototype

A Python prototype for image classification was developed, integrating a trained YOLO model for image detection and the ImageAI for image prediction. It was iterated following informal user feedback.

Version 0:

The current prototype works by using PyQt for the GUI and the ImageAI framework.

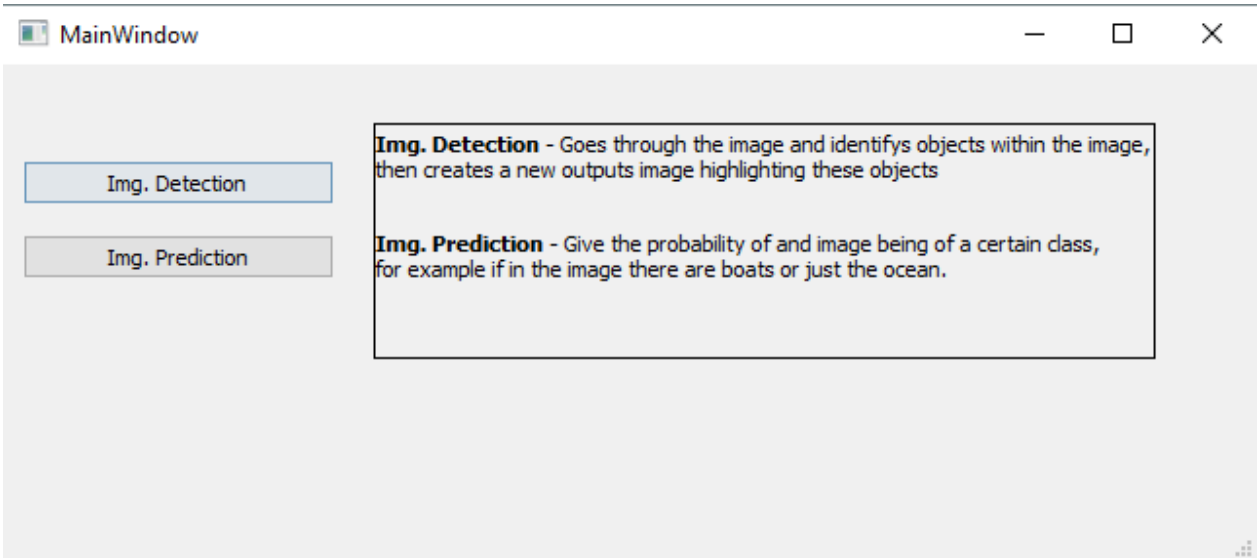


Figure 15: Prototype Home

The users select which action they wish to execute, either an image detection or custom image prediction with a specifically trained model. In the case of image detection, after the user presses the button, then is asked to select the folder using the default file explorer browser window after selecting the folder the images can then be processed. The model used for both image detection and prediction here is the pre-trained one from the ImageAi framework. After being processed these image new ones are created and stored in the Output folder in the home folder of the program's executable. The new images come with rectangles of various colors identifying the objects that were detected in the image as well as their identification probabilities.

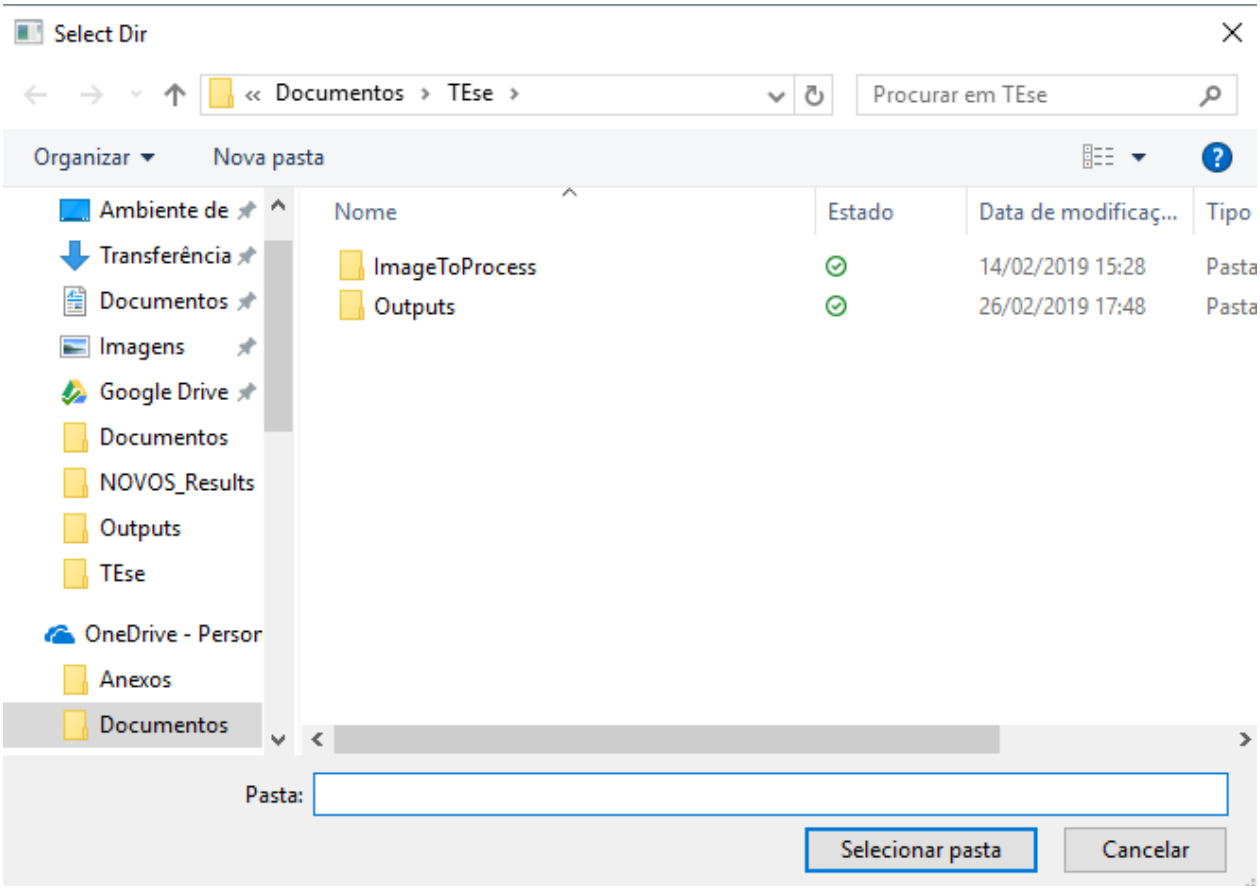


Figure 16: Select folder using the windows browse the default

In the case of image prediction, the process is the same but with a different output since we are predicting images and not identifying objects in a given image, the outputs have the probabilities of each image prediction on the upper left corner as seen in Figure 14 with the following format:

- <Class Name> < % Probability>

These predictions and probabilities come from a custom trained model for the purpose of this project with multiple images of whales and turtles trained on Google Collab using ImageAI, Collab was used for this training because it was faster than training the model locally.

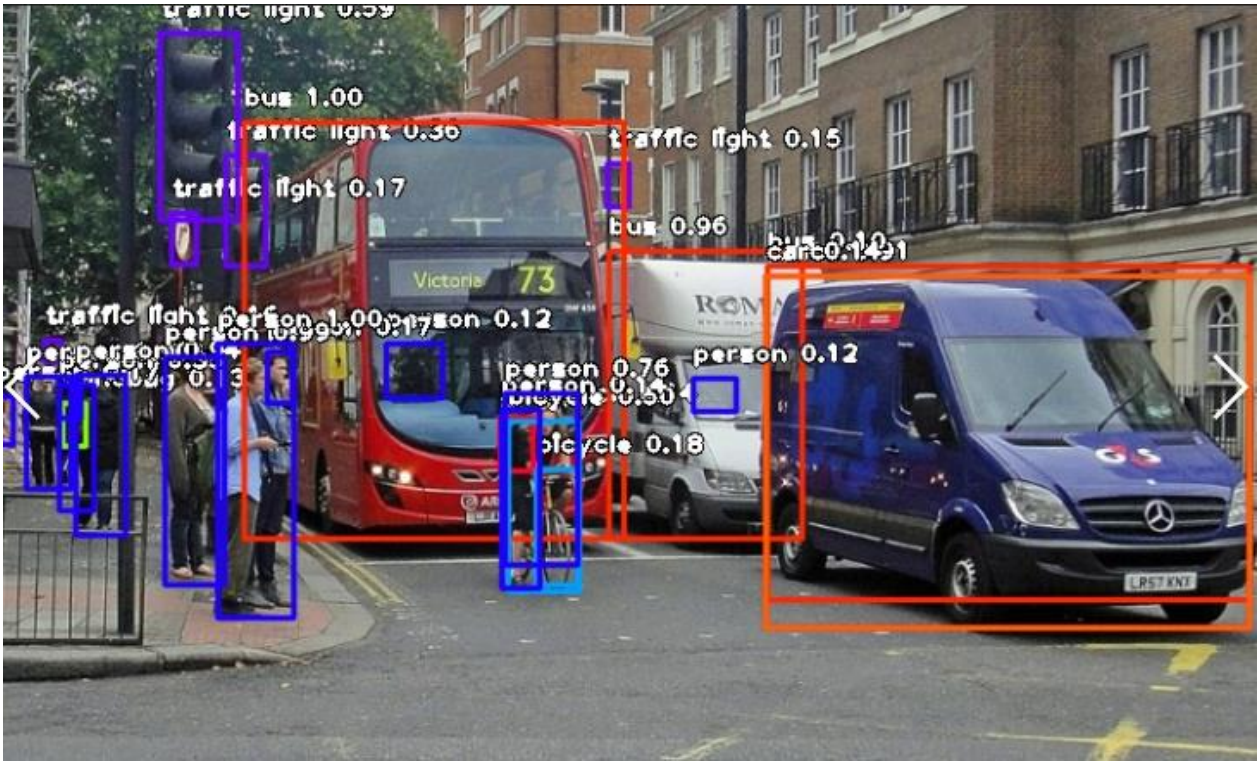


Figure 17: Image detection results

Version 0 – Usability Testing

Some basic tests were done for the initial graphical user interface (GUI), gathering user feedback. This was done with three people they were asked to do the following actions:

- Run the classification operation
- Run the prediction operation
- Open one or more of the images that were run through the classification process
- Open one or more of the images that were run through the prediction process

The first and the second action require the user to click the button related to each operation and then select a folder with images, which will be used for the selected action. These images are then stored in an output folder located in the applications home folder. The third and fourth require the user to find the output image with the images and open the image related to the task.

These initial tests showed that the initial design was missing visual feedback of what is happening. When the classification or prediction operation is underway the application will lock giving the impression that it has crashed, while it is running the algorithm in the

background, the user is locked out of the application during this time. A loading progress bar or visual queue was introduced to give the user feedback that the application is working and has not crashed as all three of the users pointed out.

Version 1:

In this version, a new feature was added: the ability to remove the green and blue from the given images, the image detection was changed from using ImageAI to using a newly trained model that uses YOLOv3, and using OpenCV to draw the bounding boxes around the objects it detects and save this new image, a loading bar was added to give the user more feedback on if the applications are working or not as well as the progress when doing any of the three actions.

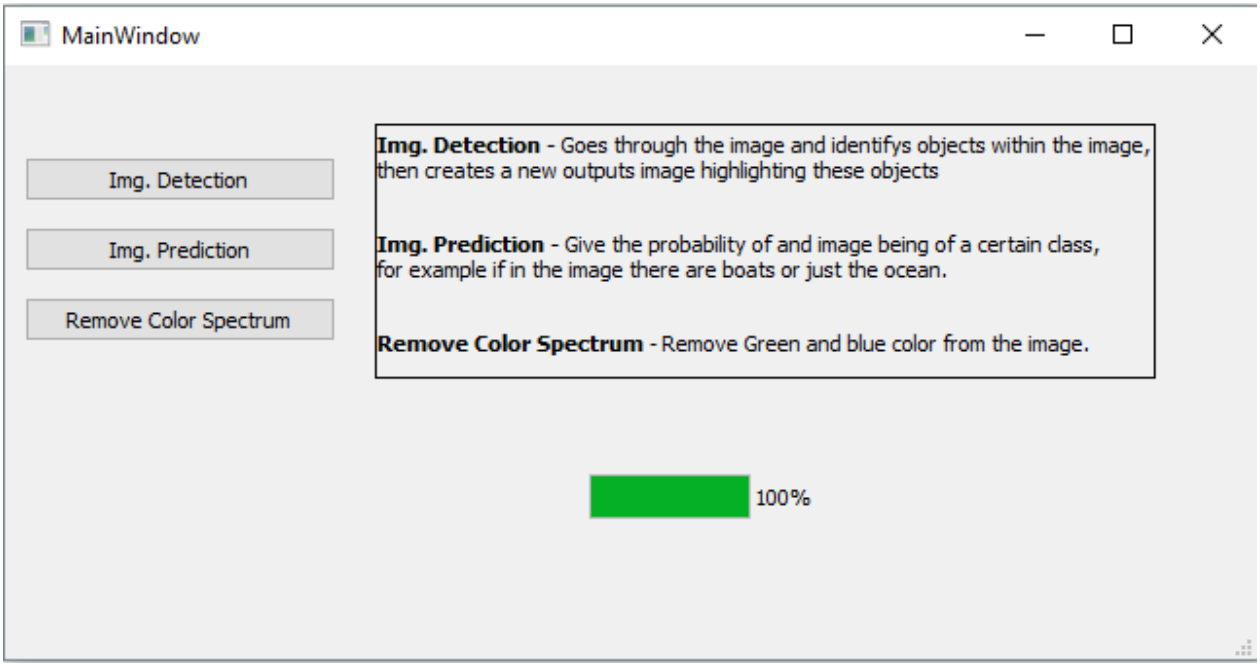


Figure 18: GUI version 1

The use of PyQt makes it a lot easier to design and implement these features, it gives all the options with a GUI with a drag and drop environment to create the interface. But because QT does not fully work out of the box with Python an extra step has to be taken, that is running a Python script that converts the GUI file created in Qt to a python file which can then be run and edit using a code editor.

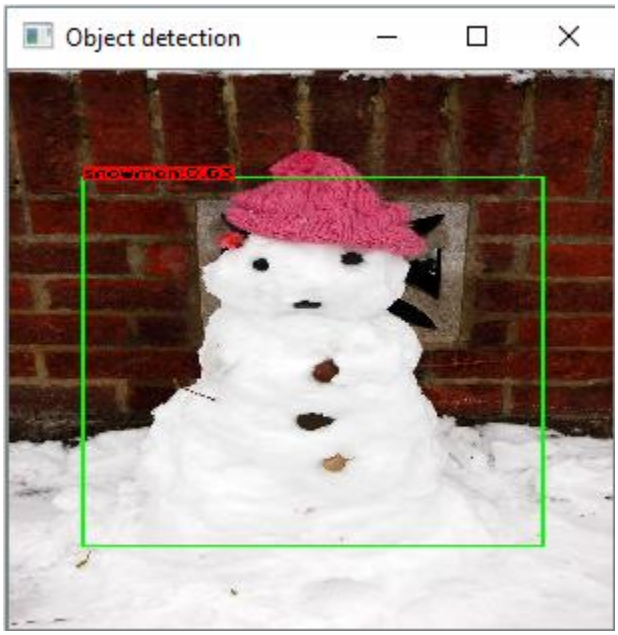


Figure 19: Initial test Model, snowman

Initially using the ImageAi framework the image was searched for objects trained with the COCO dataset, this dataset contained mostly everyday object things like bicycles. Backpacks, cars, buses, people among other things. In the case of Figure 19, the new model was trained to identify just snowman. This model was trained using YOLOv3, the training was not completed due to it being just a test, still, it identified in Figure 19 the snowman with 0.63% chance of it being a snowman. But as with any machine learning sometimes, we get mistakes. As previously stated, all of the image manipulations done, like the bounding box and the percentages were drawn using OpenCV.

Figure 20: Miss detection

As it can be seen in Figure 20, the model does have some issues, this might be due to it not finishing the training or just the similar colour of the flower.

Version 1 – User Testing:

Just like with version 0 a few user tests were done using the same task as in the testing of version 0, the feedback issue now positive with the loading bar but some issues with opening the saved images. The location where the images are being saved is not very clear for the users, a label or textbox must be added in which it is shown where the images are being saved.

Version 2:

The textbox showing the location of the newly created images with the bounding boxes was added, giving the user feedback as to where these images are being saved.

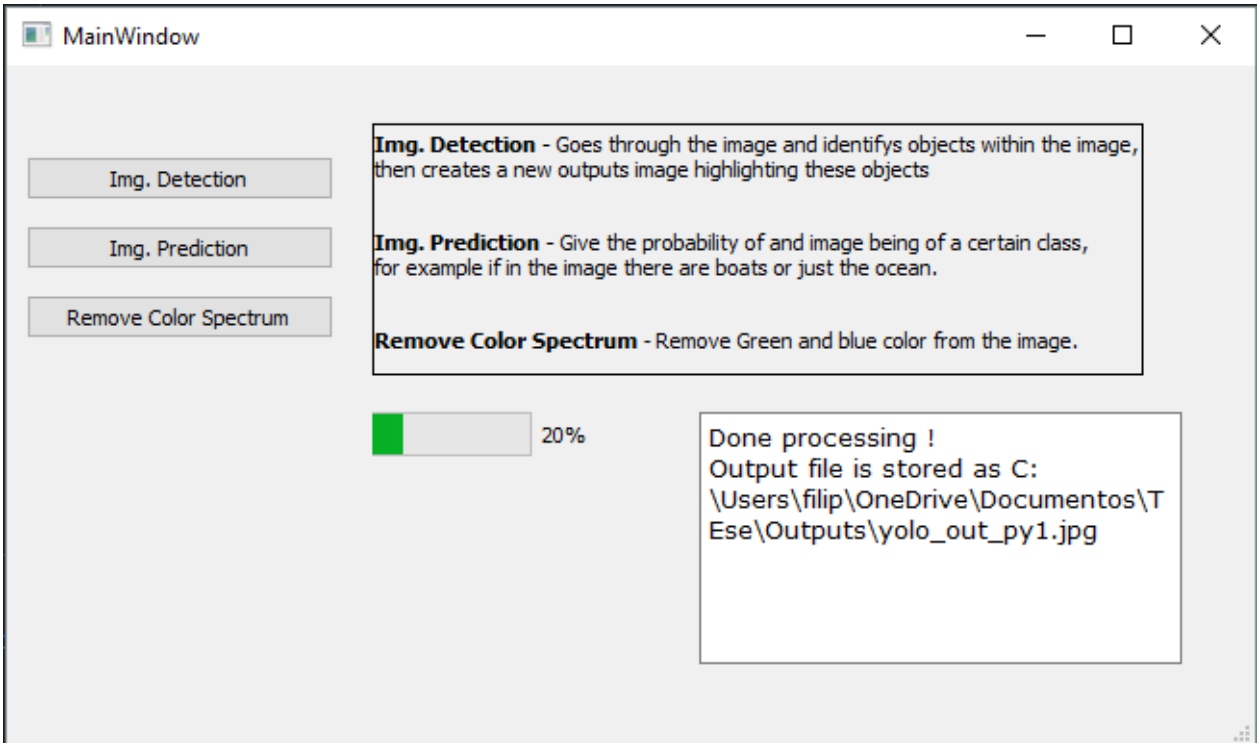


Figure 21: Version two GUI

In Figure 21 the textbox was added to the lower right corner this will display information when an image is done processing followed by the path to where the image is being saved. This will also display some error messages like is the user tries to select a folder which does not contain images. This feedback was required as seen in the user testing of the previous version.

The image detection was changed yet again, a new model was trained this one to identify whales and dolphins as previously explained in the training section of image detection. This model was also created used in Google Collab just like the snowman model. As shown in Figure 22 with the dolphin, and in Figure 23 with a whale.

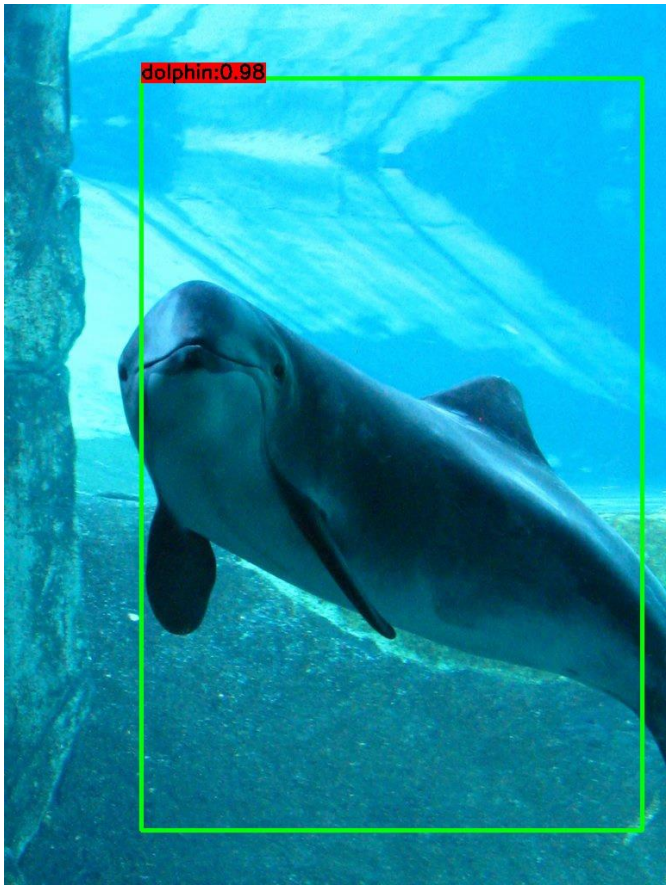


Figure 22: Dolphin image detection

The detection for the dolphin class (Figure 22) compared to the snowman, in Figure 11, presents better results, in part because of the number of interactions done in this second model were a lot higher than the ones in the snowman model, which was not concluded being interrupted after approximately 1000 iterations. The dolphin/whale model had around 3000 iterations. This led to better results as shown in Figure 22 and Figure 23 with a percentage of 0.98% and 0.94% respectively.

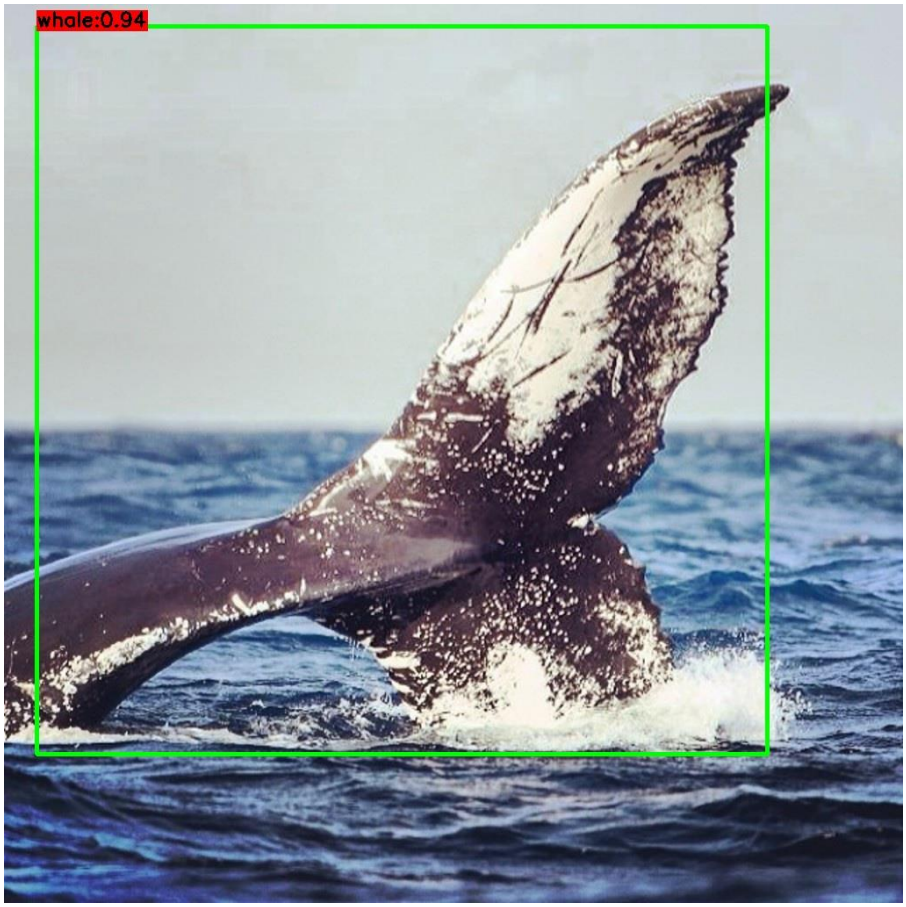


Figure 23: Whale image detection

Version 2 – Usability Testing:

The testing of version 2 was done by using the same tasks in the previous two testing phases. The issues raised by previous tasked were fixed. The added textbox showing the path where the image is being saved. One of the issues raised in this third version were ones related to image prediction due to it freezing the whole applications when it is being used making it look like the application crashed. Another one was that some of the users had some difficulties accessing the folder that contained the output images created in either detection or prediction.

Version 3:

This version checks if the output folder exists if it does not exist, it creates this folder to be used for storing the output images, two buttons were also added to the GUI window.

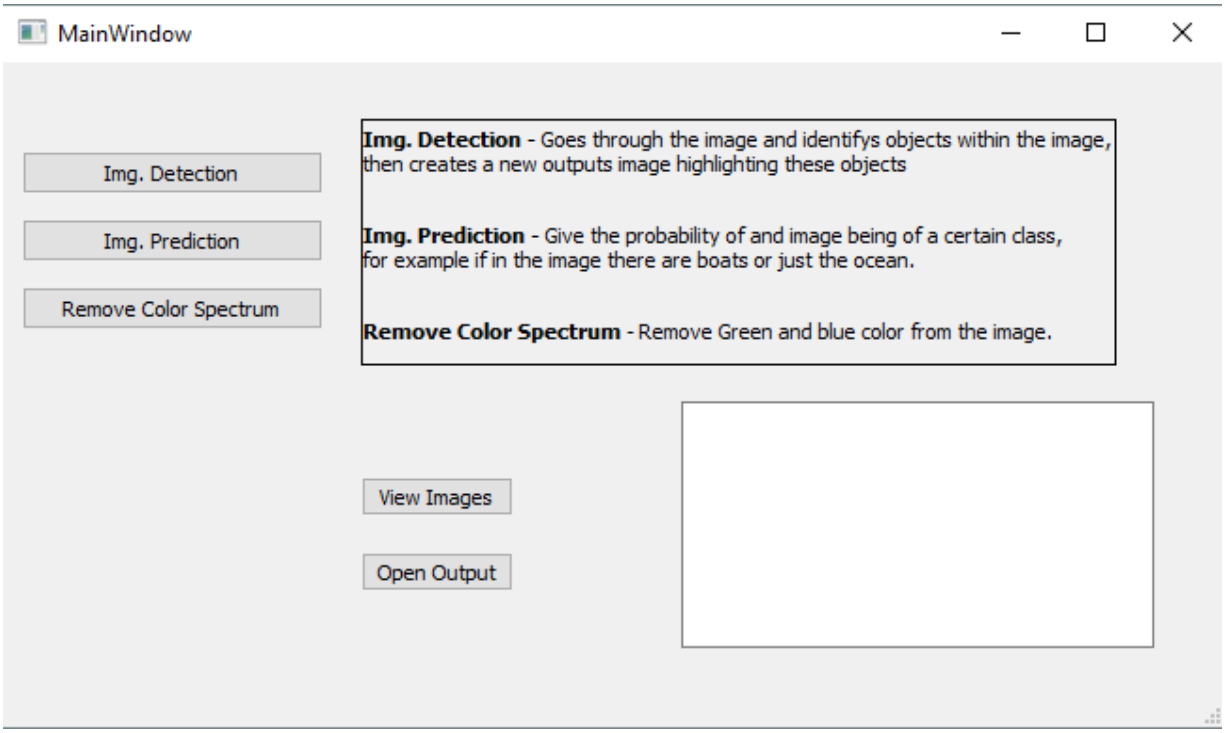


Figure 24: Version 3 GUI

These two buttons are **View Images** and **Open Output**, these two buttons were added because some of the less tech-savvy users were having a difficult time finding the output folder. The first one View Images open a grid that contains all the images inside the output folder as we can see in Figure 25.

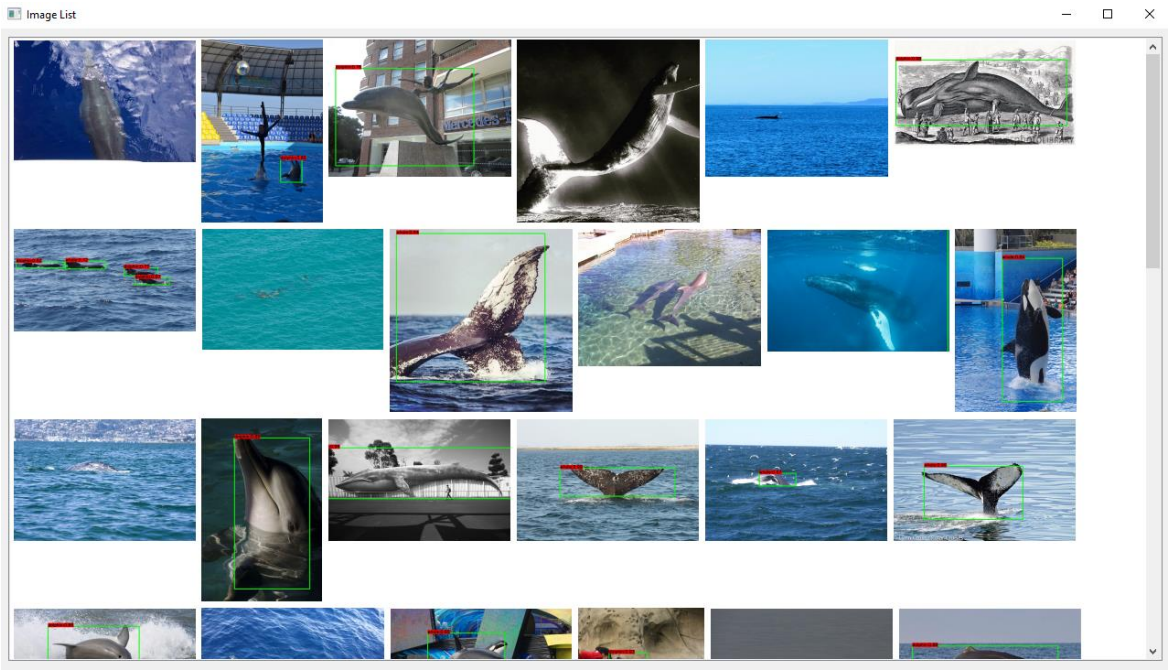


Figure 25: Grid Image Viewer

The other button, which is named **Open Output**, opens the outputs folder using the operating systems folder explorer, this way the user does not have to go looking for the folder instead just needs to press the button and the system does the rest.

Version 3 – Usability Testing:

As expected, the users who are less familiar with technology have found it a lot easier to open the output folder with the use of the button eliminating this issue. A new task was also created to make use of the grid and output folder this task is:

- Using the grid or output folder write down the percentages of two images.

The grid, as pointed out by users, makes looking at the images and even reading the percentages somewhat hard depending on the image, some users had difficulty with using just the grid initially, the majority of the users just used the output folder and then opened two different images and write down the two percentages. The reason is that the grid is sometimes harder to read, and users prefer to use their default image viewer. A small part of the users also pointed out that the name used for the output folder might not be as clear as it should be.

Left blank on purpose.

4 Evaluation

In this chapter, the results that were achieved with YOLOv3 and Keras RetinaNet will be shown and compared. A comparison between the two models is also given to justify why one was used over the other. Ending with the final discussion of about the results achieved by both models.

Going first into the model and some of the issues with it, starting off with false positives. The image detection model was trained using images of whales and dolphins and while it can correctly identify these two classes as shown in previous images, there are issues with other marine animals that look similar.

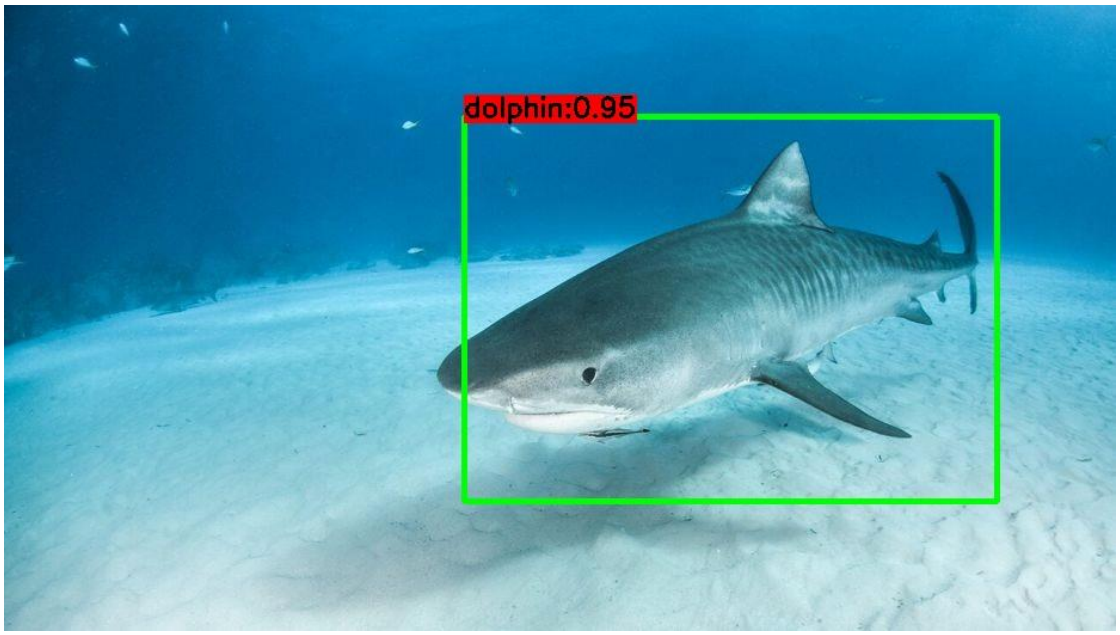


Figure 26: Shark false positive

In Figure 26 as we can see it is a false positive, the machine in this case because it did not have a class for shark specifically and a shark does indeed look like a dolphin if we do not know what a shark is. Imagine the model is like a child which does not know that sharks exist and it only know about whales and dolphins, in this case, the child would make the same assumptions until it was corrected and had been given an explanation of what a shark is, only then would he or her be able to tell the difference between these species.

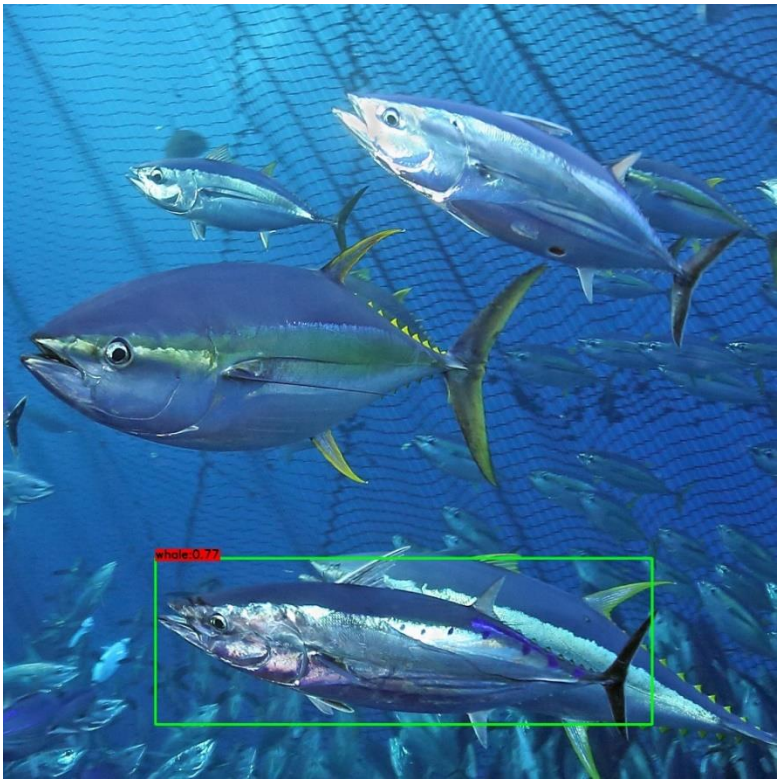


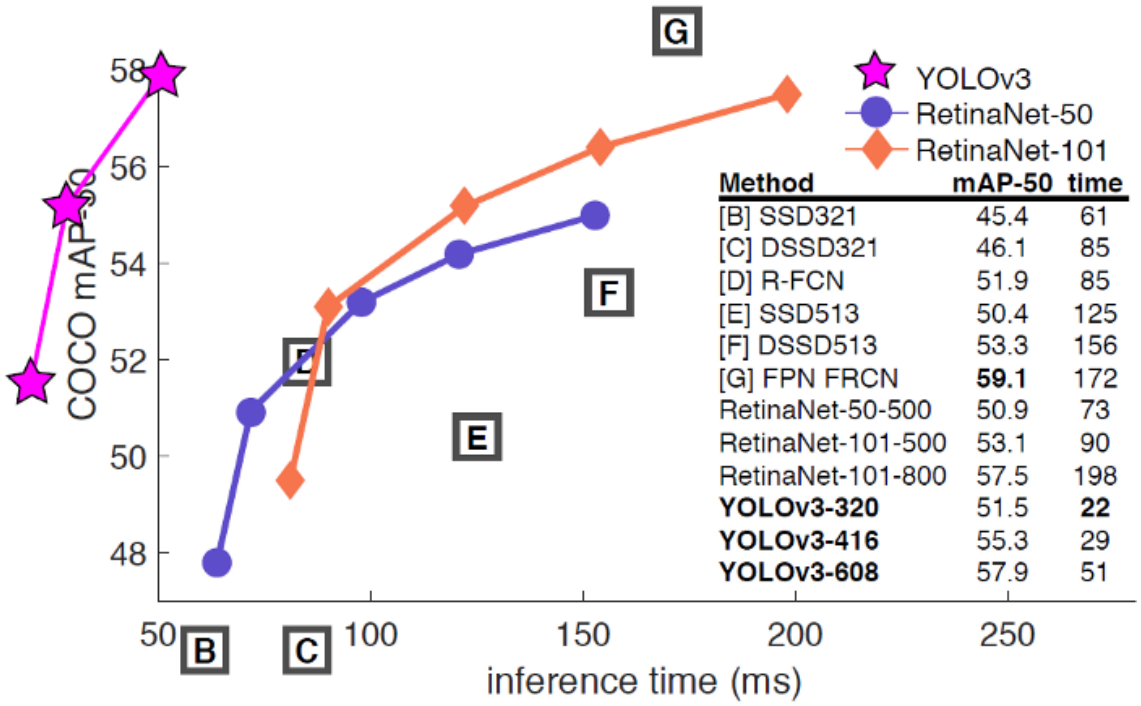
Figure 27: Whale False Positive

The same can be said in Figure 27 but in this case, the comparison would be a tuna and a whale. The detector using the test image that contained images of both classes these being whales and dolphins, a total of 133 images were used for testing out of the 1198 training image. An average precision of 0.63 was achieved for this whale and dolphin model.

Moving on to image prediction this ended not being the main part of the application this being the case it did not receive so much attention. This method to classify images showed a lot of issues and is evident in Figure 14. For this method of classification to work it needs to be done with a very specific type of images in a more controlled environment and still it might not prove the better option, more on this in the discussion section. The color spectrum option uses OpenCV to remove this might differ using other options or other image editing programs.

4.1 Choosing between Keras RetinaNet and YOLOv3

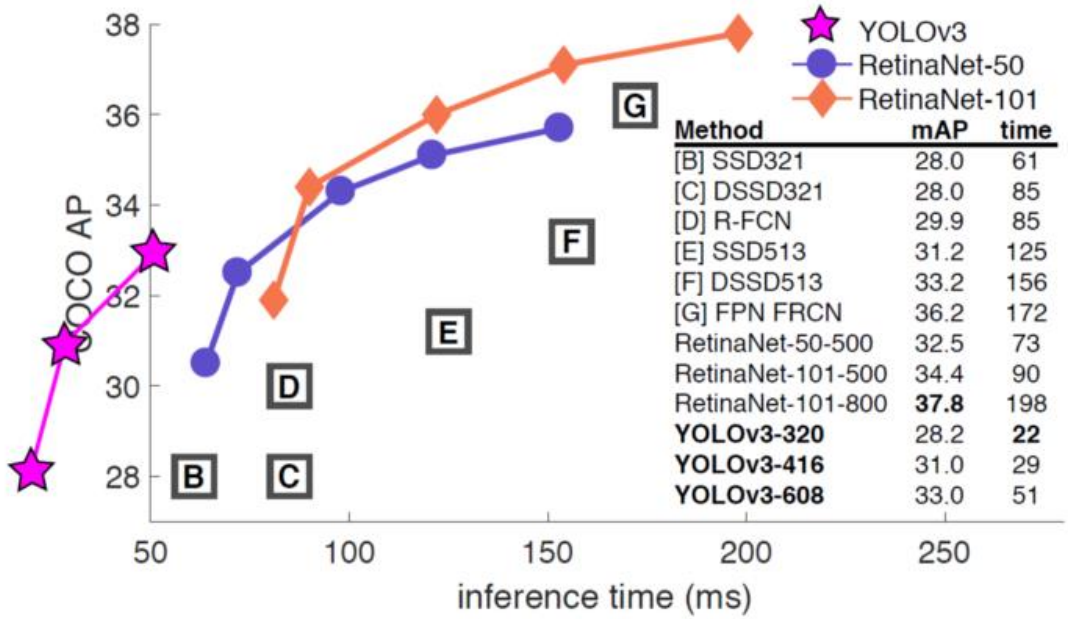
The choice of using YOLOv3 or RetinaNet will be explained here. First up a comparison between the two and other networks using COCO with mAP (mean average precision) @0.5.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Graphic 1: COCO mAP@0.5 (Redmon and Farhadi, 2018)

Just looking at the RetinaNet and YOLOv3, presented in Graphic 1, they have comparable results with mAP@0.5, but YOLOv3 has a much better inference time. As an example, YOLOv3-608 got 57.9% mAP in 51ms and RetinaNet101-800 got 57.5% mAP in 198ms, which is 3.8 times faster.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Graphic 2: Overall mAP in COCO (Redmon and Farhadi, 2018)

Graphic 2 shows the overall results of YOLOv3 and RetinaNet among others, the performance dropped significantly but still, it is possible to see YOLOv3-603 got a 33.0% mAP with 51ms inference time, RetinaNet-50-500 got 32.5% mAP with 73ms inference time. YOLOv3 is on par with other single-shot detectors (SSD) variants but 3 times faster.

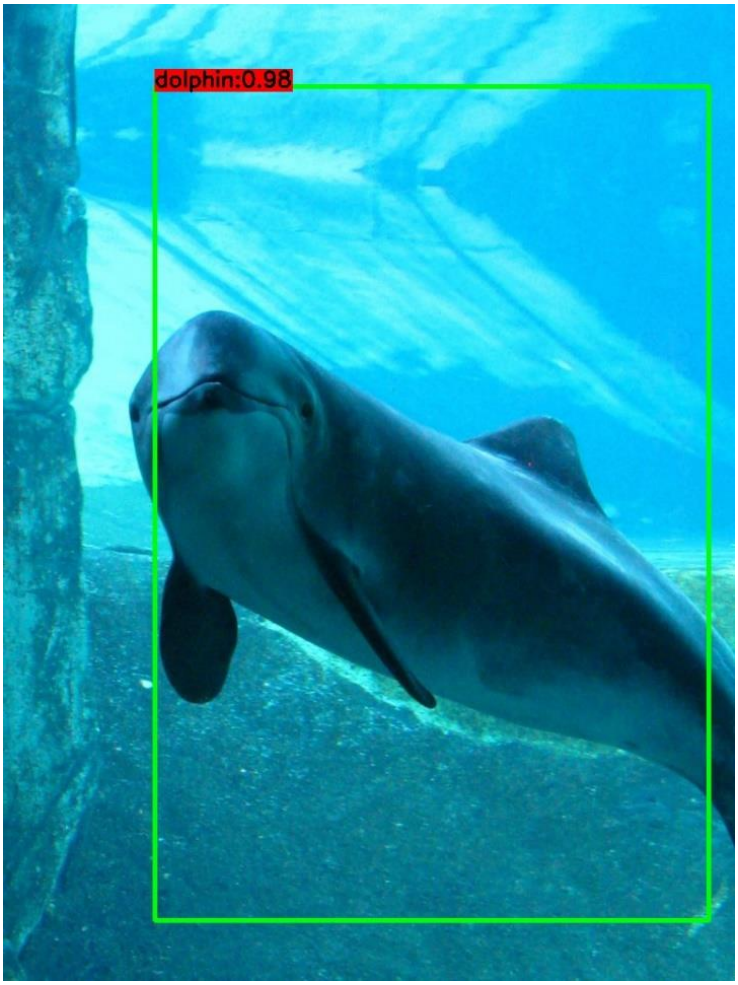


Figure 29: YOLOv3Dolphin result

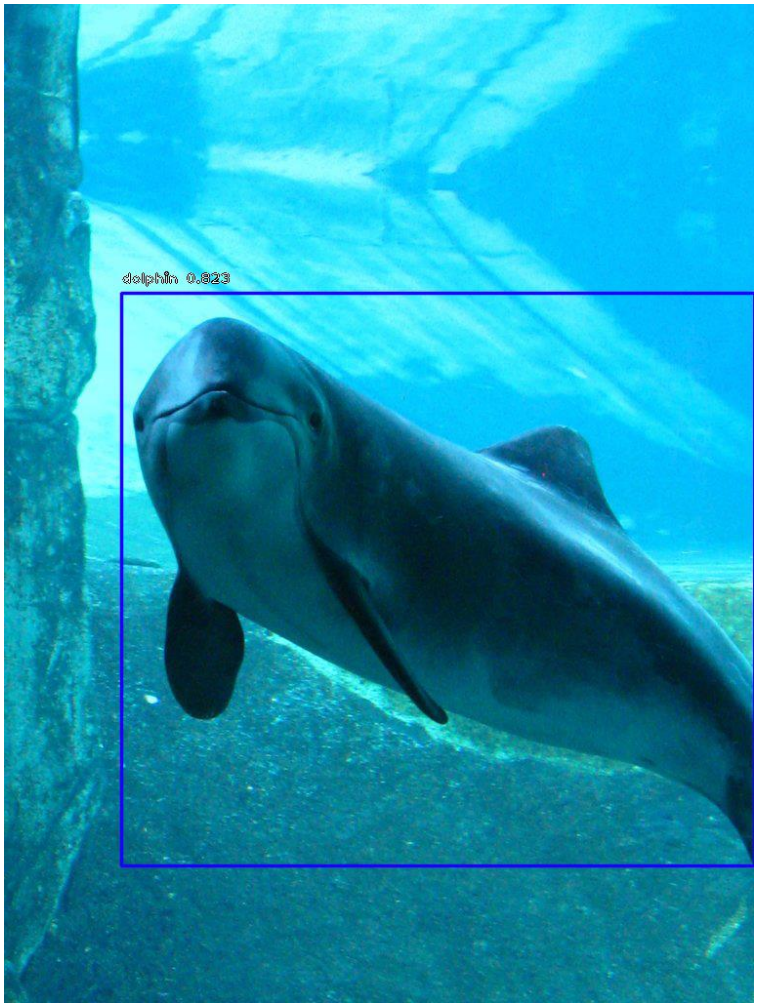


Figure 28:RetinaNet Dolphin Result

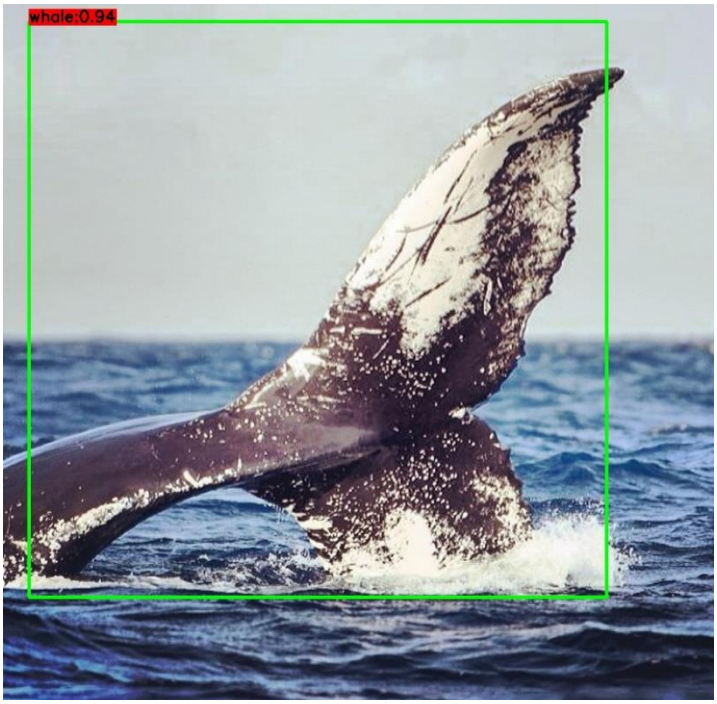


Figure 30: YOLOv3 Whale Identification

Figure 31: RetinaNet Whale Identification

Another comparison can be made with this image of the same whale, while YOLOv3 correctly identified a whale with 0.94%, RetinaNet identified a dolphin with 0.82%. This shows that the Keras RetinaNet model had more difficulty in learning from the images, but there are issues in both models as can be seen in the next comparisions.

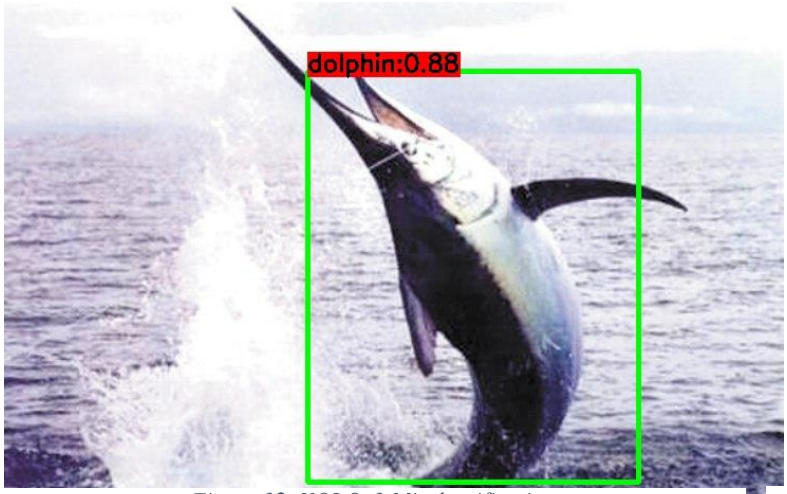


Figure 32: YOLOv3 Misclassification

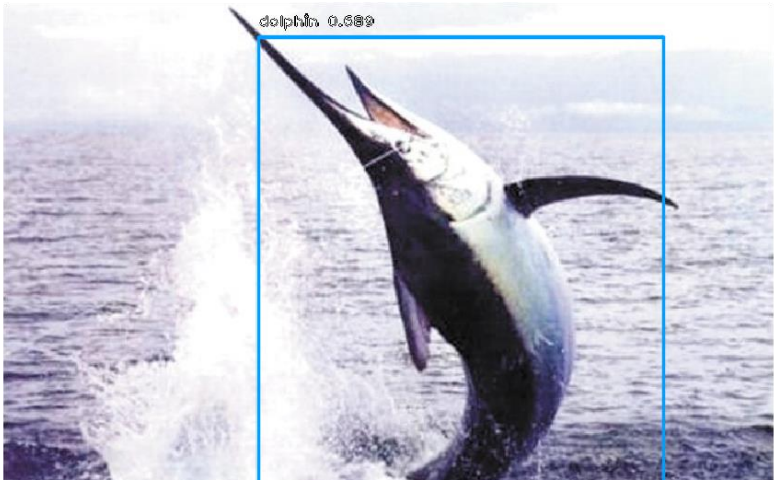


Figure 33: RetinaNet Misclassification

The two images Figure 32 and Figure 33 show just how important good images are to the end model no matter how good the model can be if the classes are not all accounted for when creating the model there will always be misclassification, in this case, identifying a swordfish as a dolphin, YOLOv3 with 0.88% and Keras RetinaNet with 0.69%.

The results as seen in the previous two graphics showed that results between YOLOv3 and RetinaNet do not vary that much. In Figure 28 and Figure 29 the results showed a bigger difference between the two: YOLOv3 has a result of 0.98% and and the RetinaNet has a results, with the same image, of 0.823%. Further training and testing would be required to completely understand why there is such a great difference between the two models. This is the reason why in the application the model that is being used is YOLOv3 and not RetinaNet even though RetinaNet can be more accurate as seen in the graphics above.

4.2 Discussion

Firstly, the differences between the models, YOLOv3 and Retina Net, it is shown in Graphic 2 comparing the overall scores of both and others. It can be seen while in most cases Keras RetinaNet is more accurate than YOLOv3, it is also slower on the inference side. While YOLOv3 can have a higher mAP is some cases, this can be seen when the bounding does not need to be so accurate, the way that it is set over the object in the image. While the current research shows that Keras Retina Net has a greater accuracy this was not seen in this test, this might be because of some difference in the steps used, or the level of detail of the images. These differences have an impact on the final model. Still, the difference between the two is not that great when looking at accuracy, while the inference time is greatly different, with YOLOv3 being almost four times faster than Keras Retina Net. Both offer a good option for image detection the purpose of the application is going to be what determines which model is going to be used. If inference time is not important and greater accuracy is required then Keras Retina Net is the better option, but if inference time is an important factor then YOLOv3 is the better option with good accuracy levels but lower than Keras Retina Net.

The false positives in image detection were expected to happen and will continue to happen if the model is not taught the difference between these similar species. To minimize these false positives the model would have to be trained with a greater number of classes,

each representing different species of a marine animal, to make a complete and more competent model. The complete elimination of false positives is something that is impossible to achieve because a certain degree of uncertainty will always exist, and factors like lighting, image quality, the distance among others will always play an important part in image detection algorithms. The average precision result was good considering the time limit of 12 hours to train the model that Google Collab has on the virtual machine, finishing the training was not possible due to this limitation. The number of images gathered for each class was also not the right amount, again this was due to the use of Open Images Dataset for the image gathering process, being limited to the number of images contained in this platform. These were the two main limitations, which can easily be avoided by using a local machine with enough requirements to train the model in an acceptable amount of time, a GeForce 1080, for example, would not be as fast as the GPU provided by Google Collaboratory but it is enough to train the model in an acceptable time frame. To deal with the other limitation the images would have to be gathered manually to serve the model that is going to be trained, this is something that consumes a lot of time and why Open Images Dataset was used. Even with these two limitations the model still achieved an acceptable level of precision, with some issues of misclassification, swapping a whale for a dolphin and vice versa, due to them being similar in some images.

In image predictions, as was mentioned in the results, this is best used in a more controlled environment like, for example, to identify mechanics.

Figure 34: Image of a car

Figure 35: Image of a Mechanic

Good examples of such contexts are shown in Figure 35 and Figure 34, where the images do not contain many objects like these two and where the machine does not get

overloaded with many possible classes. These two examples are both used in the ImageAI. The training for this model also requires about 1000 images per class to correctly identify. This type of model is not as flexible as one that uses bounding boxes to specify objects. To apply it for example for the classification of thrash on the ocean, the output images are not as clear, due to the types of images these being aerial images taken from a drone, using a more traditional image detection makes it easier to distinguish between the trash elements, if that is one of the requirements of the application. As using this prediction type does not distinguish between the objects, it just gives you the probability of the image containing a specific class without marking the said class.

The use of Python made it simpler to bring the various parts of the project together, Python is one of the main languages used for these types of problem. This made choosing PyQt an obvious decision for quick prototyping as well as making a fully functional GUI faster and more visually appealing.

The overall applicability of the applications is only limited by the models being used. And the models are just limited by the images being used. If the model for image detection would be trained to the initial objective of this thesis, the only two requirements would be the images of the trash and every single one of these images would have to be identified by using a tool like OpenLabeling to classify the objects in the image that the computer would have to identify. This would require a powerful computer to make the process of training the model viable and not have the limitations on using Google Collaboratory, like the lack of a CPU and RAM limits, but mainly the time limit of the Jupiter notebook.

5 Conclusion

The work shows that it is possible to train a model with good accuracy using either YOLOv3 or Keras Retina Net and some of the differences between the two models. Going into how to train these models and how to apply them. This work shows how the models can identify marine species, in this case, it was whales and dolphins, even though there are some issues just like with any machine learning models, also went into some ways these downsides can be minimized. A model was created that could correctly identify whales and dolphins, this can be easily applied to marine debris like plastics and other items that can be found on the surface of the ocean, by retraining the model. This can be used to further help the cleaning efforts to identify areas that require cleaning as that was the initial objective of this thesis. However, it can also be applied to other areas, needing the correct images for the desired objective and labels for each of the images for the object that are going to be identified. Image classification is a technology that can be applied in various areas, making it flexible and a powerful tool to go through a large number of images and identify specific items in each one of these images.

5.1 Limitations

One of the main limitations of this work was the lack of a computer capable of training the models, this led to another limitation. The 12 hours limit per virtual machine imposed by Google Collaboratory this made it harder to train the models and finish the training of both models.

The initial goal of this thesis was the recognition of marine litter in areal images. However, two major limitations were observed: the lack of existing models for such recognition and the lack of images for training models.

Although public image datasets provide an interesting number of object categories (e.g., COCO provides 91 object categories), most of these categories identify objects used in daily life. Therefore, the recognition of elements that are outside of the scope of daily life objects, like marine litter or animals, require the training of the existing models to recognize such elements. In addition, it was not possible to obtain images of marine litter in quantity and

quality enough for model training. It was observed that public datasets do not include such images and the areal images provided by biologists lack the quality to be recognized by machine learning models. In order to overcome this issue, it was decided to train models to recognize marine animals, applying automatic image classification and recognition to marine ecosystems and providing an initial workflow to integrate image classification algorithms into an end-user application.

5.2 Future Work

This section will serve to go through what would be required to do as well as some future improvements that could be done in the application.

Firstly, addressing the model and how some of the issues talked about in the Results and Discussion section could be resolved. A new model would have to be trained but before even beginning the training sequence a new list of requirements would have to be taken, these requirements would be related to the purpose of the model. To make it simpler to explain this will be done using ocean thrash classification as the purpose. A list of possible objects would have to be created, this list would contain possible thrash items that can be found floating in the ocean after this list is completed, it would need to be reduced to eliminate possible redundant or similar objects. Then some sample photos would have to be used to see if it is possible to distinguish between the objects in the picture or not, the class list would then have to be updated again. The type of images that are going to be used would also play an important part, will the images be with the top-down view taken from a drone or at the sea level taken from a boat. After deciding this the sample images would be collected. After having these initial requirements down the image collection process can then begin, this would require something between 800 to 1500 images for each of the classes previously decided upon. Each of these images would then be required to be individually looked at, each class correctly identified by using Open Labeling for example only then would it be possible to train the model.

Further training to make the correct choice between the model to use would also have to be done, and a choice would then have to be made between the two models, would a faster

inference time that YOLOv3 offers be better over the higher accuracy of Keras Retina Net, or would losing some of the accuracies be acceptable.

Now looking more into possible feature improvements for the application itself, adding the capability for it to do image classification by using video, not only images. This video option could then be explored into a real-time video classification, making it possible to use a drone to fly over an area and classify the object in the area, while at the same time saving the video. The GUI can always be improved further, better placements of button, descriptions, more user testing overall.

Another improvement would be making it possible to train the model in the application without having to train it externally, this would be the hardest part to develop and implement. Using Darknet version of YoloV3 for example and try to find a way to import it into the applications making it possible to use it inside the application itself.

6 References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. TensorFlow: A System for Large-Scale Machine Learning 2016:265–83.
2. Agarap AFM, M. AF. On breast cancer detection. Proc. 2nd Int. Conf. Mach. Learn. Soft Comput. - ICMLSC '18, New York, New York, USA: ACM Press; 2018, p. 5–9. doi:10.1145/3184066.3184080.
3. AlexeyAB. AlexeyAB/darknet: Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection (Tensor Cores are used) n.d. https://github.com/AlexeyAB/darknet (accessed April 8, 2019).
4. Amershi S, Cakmak M, Knox WB, Kulesza T. Power to the People: The Role of Humans in Interactive Machine Learning. AI Mag 2014;35:105. doi:10.1609/aimag.v35i4.2513.
5. Amershi S, Fogarty J, Weld D. Regroup: interactive machine learning for on-demand group creation in social networks. Proc. 2012 ACM Annu. Conf. Hum. Factors Comput. Syst. - CHI '12, New York, New York, USA: ACM Press; 2012, p. 21. doi:10.1145/2207676.2207680.
6. Barga R, Fontama V, Tok WH. Introducing Microsoft Azure Machine Learning. Predict. Anal. with Microsoft Azur. Mach. Learn., Berkeley, CA: Apress; 2015, p. 21–43. doi:10.1007/978-1-4842-1200-4_2.
7. Beijbom O. Automated Annotation of Coral Reef Survey Images. University of California, 2015.
8. Cai CJ, Jongejan J, Holbrook J. The effects of example-based explanations in a machine learning interface. Proc. 24th Int. Conf. Intell. User Interfaces - IUI '19, New York, New York, USA: ACM Press; 2019a, p. 258–62. doi:10.1145/3301275.3302289.
9. Cai CJ, Reif E, Hegde N, Hipp J, Kim B, Smilkov D, et al. Human-Centered Tools for Coping with Imperfect Algorithms during Medical Decision-Making 2019b.
10. Carneiro T, Medeiros Da Nobrega RV, Nepomuceno T, Bian G-B, De Albuquerque VHC, Filho PPR. Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. IEEE Access 2018;6:61677–85.

doi:10.1109/ACCESS.2018.2874767.

11. Chakraborty P, Sharma A, Hegde C. Freeway Traffic Incident Detection from Cameras: A Semi-Supervised Learning Approach. 2018 21st Int. Conf. Intell. Transp. Syst., IEEE; 2018, p. 1840–5. doi:10.1109/ITSC.2018.8569426.
12. Chandan G, Jain A, Jain H, Mohana. Real Time Object Detection and Tracking Using Deep Learning and OpenCV. 2018 Int. Conf. Inven. Res. Comput. Appl., IEEE; 2018, p. 1305–8. doi:10.1109/ICIRCA.2018.8597266.
13. Cui Y, Oztan B. Automated firearms detection in cargo x-ray images using RetinaNet. In: Ashok A, Gehm ME, Greenberg JA, editors. Anom. Detect. Imaging with X-Rays IV, vol. 10999, SPIE; 2019, p. 24. doi:10.1117/12.2517817.
14. Fails JA, Olsen DR. Interactive machine learning. Proc. 8th Int. Conf. Intell. user interfaces - IUI '03, New York, New York, USA: ACM Press; 2003, p. 39. doi:10.1145/604045.604056.
15. Fefilatyev S. Detection of marine vehicles in images and video of open sea. Grad Theses Diss 2008.
16. Hoff KA, Bashir M. Trust in Automation. Hum Factors J Hum Factors Ergon Soc 2015;57:407–34. doi:10.1177/0018720814547570.
17. Hoff S, Jaffurs P, Enriquez M, Wilde Q. Snap-n-Snack: a Food Image Recognition Application. Comput Eng Sr Theses 2018.
18. Jensen MB, Nasrollahi K, Moeslund TB. Evaluating State-Of-The-Art Object Detector on Challenging Traffic Light Data 2017:9–15.
19. Jo H, Yoon YI. Intelligent smart home energy efficiency model using artificial TensorFlow engine. Human-Centric Comput Inf Sci 2018;8:9. doi:10.1186/s13673-018-0132-y.
20. João Cartucho. Cartucho/OpenLabeling: Label images and video for computer vision applications n.d. https://github.com/Cartucho/OpenLabeling (accessed April 8, 2019).
21. Kizilcec RF, F. R. How Much Information?: Effects of Transparency on Trust in an Algorithmic Interface. Proc. 2016 CHI Conf. Hum. Factors Comput. Syst. - CHI '16, New York, New York, USA: ACM Press; 2016, p. 2390–5. doi:10.1145/2858036.2858402.

22. Leira FS, Johansen TA, Fossen TI. Automatic detection, classification and tracking of objects in the ocean surface from UAVs using a thermal camera. 2015 IEEE Aerosp. Conf., IEEE; 2015, p. 1–10. doi:10.1109/AERO.2015.7119238.
23. Li X, Zhao H, Zhang L. Recurrent RetinaNet: A Video Object Detection Model Based on Focal Loss, Springer, Cham; 2018, p. 499–508. doi:10.1007/978-3-030-04212-7_44.
24. Lin T-Y, Dollar P, Girshick R, He K, Hariharan B, Belongie S. Feature Pyramid Networks for Object Detection. 2017 IEEE Conf. Comput. Vis. Pattern Recognit., IEEE; 2017a, p. 936–44. doi:10.1109/CVPR.2017.106.
25. Lin T-Y, Goyal P, Girshick R, He K, Dollár P. Focal Loss for Dense Object Detection 2017b.
26. Lin T-Y, Maire M, Belongie S, Hays J, Perona P, Ramanan D, et al. Microsoft COCO: Common Objects in Context, Springer, Cham; 2014, p. 740–55. doi:10.1007/978-3-319-10602-1_48.
27. Liu W, Ma L, Wang J, Chen H. Detection of Multiclass Objects in Optical Remote Sensing Images. IEEE Geosci Remote Sens Lett 2019;16:791–5. doi:10.1109/LGRS.2018.2882778.
28. Luo S, Xu C, Li H. An Application of Object Detection Based on YOLOv3 in Traffic. Proc. 2019 Int. Conf. Image, Video Signal Process. - IVSP 2019, New York, New York, USA: ACM Press; 2019, p. 68–72. doi:10.1145/3317640.3317657.
29. Marengoni M, Stringhini D. High Level Computer Vision Using OpenCV. 2011 24th SIBGRAPI Conf. Graph. Patterns, Images Tutorials, IEEE; 2011, p. 11–24. doi:10.1109/SIBGRAPI-T.2011.11.
30. McCauley DJ, Pinsky ML, Palumbi SR, Estes JA, Joyce FH, Warner RR. Marine defaunation: Animal loss in the global ocean. Science (80-) 2015;347:1255641–1255641. doi:10.1126/science.1255641.
31. Mulfari D, Longo Minnolo A, Puliafito A. Building TensorFlow Applications in Smart City Scenarios. 2017 IEEE Int. Conf. Smart Comput., IEEE; 2017, p. 1–5. doi:10.1109/SMARTCOMP.2017.7946991.
32. Nagelkerken I, Munday PL. Animal behaviour shapes the ecological effects of ocean acidification and warming: moving from individual to community-level responses.

Glob Chang Biol 2016;22:974–89. doi:10.1111/gcb.13167.

33. Pohekari, Rajshri, Vaibhav Patel and AS. Cyber Attack Detection and Classification Using Mahcine Learning Technique Using Microsoft Azure Cloud. Int Res J Eng Appl Sci 2017.
34. Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection 2015.
35. Redmon J, Farhadi A. YOLOv3: An Incremental Improvement 2018.
36. Ribeiro MT, Singh S, Guestrin C. "Why Should I Trust You?" Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD '16, New York, New York, USA: ACM Press; 2016, p. 1135–44. doi:10.1145/2939672.2939778.
37. Sato M, Horie K, Hara A, Miyamoto Y, Kurihara K, Tomio K, et al. Application of deep learning to the classification of images from colposcopy. Oncol Lett 2018;15:3518–23. doi:10.3892/ol.2018.7762.
38. Sharif Razavian A, Azizpour H, Sullivan J, Carlsson S. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition 2014:806–13.
39. Simard PY, Amershi S, Chickering DM, Pelton AE, Ghorashi S, Meek C, et al. Machine Teaching: A New Paradigm for Building Machine Learning Systems 2017.
40. Tariq S, Lee S, Kim H, Shin Y, Woo SS. Detecting Both Machine and Human Created Fake Face Images In the Wild. Proc. 2nd Int. Work. Multimed. Priv. Secur. - MPS '18, New York, New York, USA: ACM Press; 2018, p. 81–7. doi:10.1145/3267357.3267367.
41. Tumas P, Serackis A. Automated Image Annotation based on YOLOv3. 2018 IEEE 6th Work. Adv. Information, Electron. Electr. Eng., IEEE; 2018, p. 1–3. doi:10.1109/AIEEE.2018.8592167.
42. tzutalin. GitHub - tzutalin/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images n.d. https://github.com/tzutalin/labelImg (accessed July 26, 2019).
43. Vaccaro K, Huang D, Eslami M, Sandvig C, Hamilton K, Karahalios K. The Illusion of Control. Proc. 2018 CHI Conf. Hum. Factors Comput. Syst. - CHI '18, New York, New York, USA: ACM Press; 2018, p. 1–13. doi:10.1145/3173574.3173590.

44. Xia X-L, Xu C, Nan B. Facial Expression Recognition Based on TensorFlow Platform. ITM Web Conf 2017;12:01005. doi:10.1051/itmconf/20171201005.
45. Yasmin M, Mohsin S, Sharif M. Intelligent Image Retrieval Techniques: A Survey. J Appl Res Technol 2014;12:87–103. doi:10.1016/S1665-6423(14)71609-8.
46. Zenggyu. RetinaNet Explained and Demystified 2018. https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/.