ESCOLA NAVAL

talantivs bifaire

Luís Pedro Lages Vale Gonçalves

Automatic detection of castaways in SAR missions using UAVs

A deep neural network approach

A dissertation submitted in partial fulfillment of the requirements for the degree of Marine Military Sciences, specialisation of Engenharia Naval Ramo de Armas e Eletrónica

Alfeite

2021

Luís Pedro Lages Vale Gonçalves

Automatic detection of castaways in SAR missions using UAVs
A deep neural network approach

A dissertation submitted in partial fulfillment of the requirements for the degree of Marine Military Sciences, specialisation of Engenharia Naval Ramo de Armas e Eletrónica

Orientation of: Bruno Duarte Damas
Co-orientation of: Victor José de Almeida e Sousa Lobo

The Master Student, *The Supervisor,*

Luís Gonçalves Bruno Damas

Alfeite
2021

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

- Alan Turing

This work is dedicated to my mother, who was always there for me when I needed most and is dearly missed.

Acknowledgement

Making a thesis may be a solitary work but it cannot be done alone. I would like to thank all the people whose help, whether direct or indirect, lead to the conclusion of this work.

First and foremost, to my family, that has been watching over me for all these years and gave me all the support needed to keep me in the right path.

To all my friends, that were always there when I needed to talk and helped me overcome the challenges I faced.

To all of the Portuguese Naval School, especially to CFR Luís Pedro Dantas Pereira de Castro and 1TEN Manuel Rui Veloso Domingues, without whom it would not have been possible to capture many of the images that made this work possible.

And last but not least, to my tutor Prof. Bruno Duarte Damas that was always present when help was needed and presented new solutions to my problems when I had none.

Abstract

Unmanned Aerial Vehicle (UAV) images can be an important resource when performing Search And Rescue (SAR) operations at sea. With the improving technology UAVs are becoming an accessible and fairly inexpensive resource for many applications such as SAR. In order to maximize the usefulness of these UAV, we propose a method which utilizes a state-of-the-art Object Detection network to perform real-time detection on-board the UAV. In this thesis we have selected the YOLOv4-tiny Object Detection network and trained it to detect castaways at sea. The main goal is to obtained fully trained weights that can be utilised with the YOLOv4-tiny and applied to use in SAR with several UAVs working in parallel that report back to a human operator upon detection of a possible castaway. The proposed approach has been validated on a test dataset obtained for the purpose of this thesis and the final result shows that it has good capabilities that can be further developed.

Keywords: UAV Images, Search and Rescue, Deep Learning, Deep Convolutional Neural Networks

Resumo

Imagens de veículos aéreos não tripulados (UAV) podem ser um recurso importante para a realização de operações de busca e salvamento (SAR) no mar. Com os avanços da tecnologia, os UAVs têm-se tornado um recurso acessível e razoavelmente barato para muitas aplicações como SAR. A fim de maximizar a utilidade destes UAV, propôs-se um método que utiliza uma rede de detecção de objetos de última geração para realizar a detecção em tempo real a bordo do UAV. Nesta tese, selecionou-se a rede de detecção de objetos YOLOv4-tiny e efetuou-se o seu treino para detectar náufragos no mar. O principal objetivo é obter pesos totalmente treinados que possam ser utilizados com o YOLOv4-tiny e aplicados para uso em SAR com vários UAVs a operar em paralelo que reportam a um operador humano aquando a detecção de um possível náufrago. A abordagem proposta foi validada com um conjunto de dados de teste obtido para o propósito desta tese, e o resultado final mostrou boas capacidades que podem ser ainda mais desenvolvidas.

Palavras-chave: Imagens UAV, Busca e Salvamento, Aprendizagem Profunda, Redes Neuronais Convolucionais Profundas

Contents

List of Figures

List of Tables

List of Equations

Glossary

AI	Artificial Intelligence
AP	Average Precision
CIoU	Complete Intersection over Union
CmBN	Cross mini-Batch Normalization
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CSP	Cross-Stage-Partial-connections
FN	False Negative
FP	False Positive
fps	frames per second
GAN	Generative Adversarial Networks
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
ICAO	International Civil Aviation Organization
ICARUS	Integrated Components for Assisted Rescue and Unmanned Search operations
IMO	International Maritime Organization
IoU	Intersection over Union
IR	Infrared
LWIR	Long-Wave Infrared
mAP	mean Average Precision
PAN	Path Aggregation Network
PCB	Partial in Computational Block

PR Precision-Recall

R-CNN Region-based Convolutional Neural Network
RPN Region Proposal Network

SAM Spatial Attention Module
SAR Search And Rescue
SAT Self-Adversarial-Training
SIFT Scale-invariant Feature Transform
SPP Spatial Pyramid Pooling
SSD Single Shot Detector
SVM Support Vector Machine

TN True Negative
TP True Positive

UAV Unmanned Aerial Vehicle
USV Unmanned Surface Vessel
UUV Unmanned Underwater Vessel

YOLO You Only Look Once

Chapter 1

Introduction

Travel through sea has long been an important part of the world economy and as a result of that there is a significant number of ships at sea at any given time. With these many ships it is not surprising that the occurrence of castaways is somewhat common and has, therefore, lead to the need for an ever evolving response capability in terms of SAR at sea. The International Maritime Organization (IMO) and the International Civil Aviation Organization (ICAO) have jointly published the IAMSAR Manual [1] which provides guidelines for the organization of SAR response.

Nowadays most SAR operations count with the support of aircrafts and ships in order to find and retrieve castaways but this involves a lot of resources and manpower. With the evolving technology in the field of UAV there can be new ways to improve our response capabilities with fewer resources and manpower. With the use of UAVs we can greatly extend the area covered and even automate some processes previously done by human operators. The use of UAVs can be done on three levels, starting with the simplest method which is the use of a UAV with an operator performing both the control and detection, followed by a more optimized method where the UAV follows an automated flight plan and the detection is performed by the operator which allows a single operator to work with several UAVs simultaneously but requires a robust datalink for video transmission over long distances. The third and most advanced method is one where the UAV follows an automated flight plan and performs on-board detection allowing the operator to only analyse the situations considered relevant. By combining UAVs with automatic object detection algorithms we can create a system with the capability to find castaways and report the location and visuals back to the operator for confirmation.

The scientific field of machine learning, often associated with the term Artificial Intelligence (AI), has been around for decades but has garnered increasingly more attention in the last few years. It is a field which has several areas of

study, ranging from predictions based on historical data to computer vision, with every possible use in between [2]. Companies such as Tesla have brought the subject to the public eye with the application of Deep Learning on cars and other everyday items, especially through the use of computer vision, prediction and path planning in their products [3].

In this thesis we will be focusing on computer vision, a field which has a dual goal. "From the biological science point of view, computer vision aims to come up with computational models of the human visual system. From the engineering point of view, computer vision aims to build autonomous systems which could perform some of the tasks which the human visual system can perform (and even surpass it in many cases)." [4]. Within this field, our main interest will be the capabilities to perform automatic object detection.

The objective of this thesis is to be able to perform an automatic detection of castaways from images acquired from UAVs and in order to perform that task we will be using recently developed techniques based on Deep Neural Networks. There will be a need to choose an appropriate network architecture that is capable of performing real-time detection while running on an embedded system on-board the UAV and proceed with the training and evaluation of the performance of the chosen architecture using an annotated dataset of castaway at sea situations. In the end we want to achieve an automatic detection that requires relatively low processing power and only reports back upon a positive result enabling the simultaneous use of several UAVs and reducing the needed manpower and resources.

The structure of this thesis follows the work developed. After the introduction, we will perform a review of the literature that pertains to the themes of our work in Chapter 2. We will study the works that provide a background on Object Detection and Data Augmentation as well as some works that are related to ours. With the Literature Review done we proceed to Chapter 3 where we will explain the processes utilized to perform our work. In Chapter 4 we present the results obtained on the several experiments performed and a small analysis of results. We end with our conclusion and suggestions of future works and implementations.

Chapter 2

Literature Review

The need to detect and classify objects has been around for a long time and the recent improvements in technology have allowed for fast and accurate automatic object detection and classification. In this chapter we will be reviewing some of the background work that lead to our thesis as well as the related works which will serve as a reference and starting point.

2.1 Background

Object detection is a problem addressed by Computer Vision which deals with identifying and locating objects of specific classes in an image. Object detection was studied before the application of Convolutional Neural Networks (CNN) in Computer Vision. It is important to take a look at the conventional methods in order to understand how it all began.

After the introduction of CNNs to Computer Vision there was a big evolution, especially in the last decade, and we have seen the development of several applications that make use of CNNs to perform ever faster and more accurate detection and classification of objects. We will be analysing the work done in this field and see the evolution of the capabilities and subsequent improvements.

In order to perform the training of our selected network we will need a method of enlarging our initial dataset and that method is data augmentation. With data augmentation we can apply several different augmentations that will create altered images with sufficiently different features so that they can be used to extend our dataset.

2.1.1 Object Detection

There are several applications of Computer Vision and there is the need to begin by making an important distinction between Image Classification, Object Detection and Image Segmentation. Image Classification is used when we need to classify an image into a category while Object Detection and Image Segmentation aim to locate the objects in the image. For a better understanding of this difference refer to Figure 2.1.

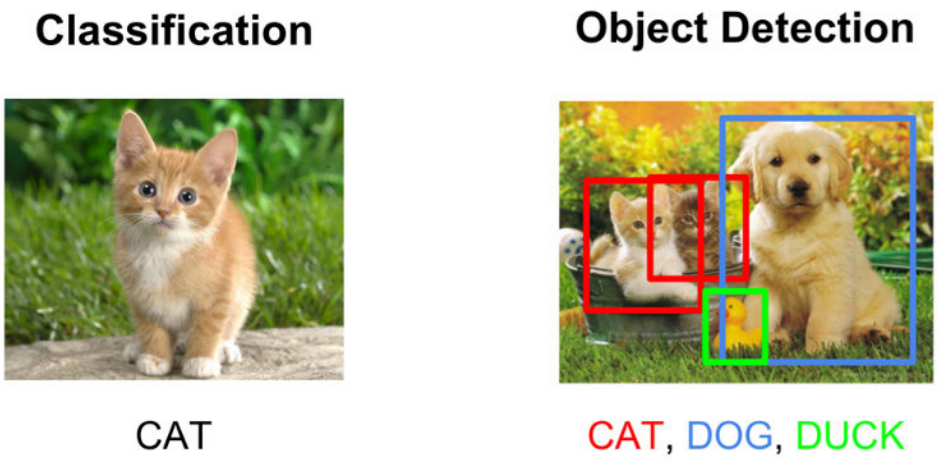


FIGURE 2.1: Difference between classification and object detection
Source: obtained from [5]

With Object Detection we are able to build a bounding box, which is a square or rectangular box that contains our object, but it will not give us any information about the shape of the the objects. With Image Segmentation on the other hand we are able to obtain more precise pixel-wise masks for our objects, giving us a better understanding of their shapes. The distinction between Object Detection and Image segmentation can be easily observed in Figure 2.2.

Now that we have a better understanding of the differences between Image Classification, Object Detection and Image Segmentation we need to understand the metrics used to evaluate the performance of our detections.

There are many metrics that apply to this kind of detection so we'll start with the most basic: True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). A TP is a result that correctly indicates the presence of an object while a FP is a result that indicates the presence of an object when it is

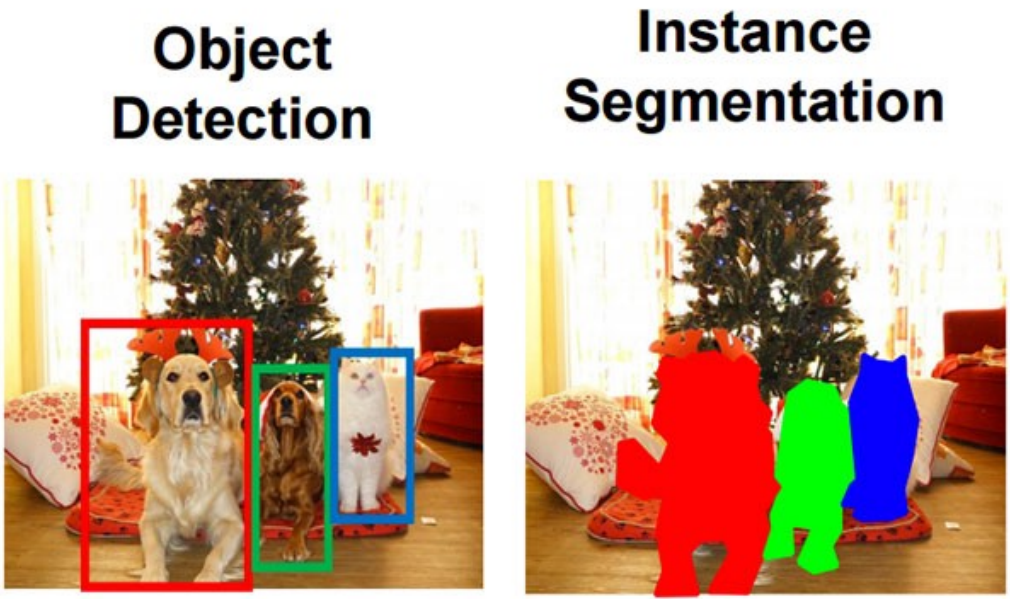


FIGURE 2.2: Object Detection vs Image Segmentation
Source: obtained from [6]

not actually present. A TN is a result that correctly indicates that there isn't an object present while a FN is a result that doesn't indicate the presence of an object when it is in fact present.

Using these basic metrics we can calculate Precision and Recall. Precision measures how accurate the predictions are. It measures how many of the predictions are actually correct using the formula displayed in Equation 2.1. Recall measures how well we find all the positives. It measures the ability to detect all objects present in the image using the formula displayed in Equation 2.2.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.2}$$

Another metric used for Object Detection is Intersection over Union (IoU) which divides the Area of Overlap of the 2 bounding boxes involved with the Area of their Union as we can observe in Figure 2.3. With IoU we obtain a score between 0 and 1 which represents the quality of the overlap between the ground truth, which is the bounding box that contains the true position of the object, and the generated

bounding box. This metric is necessary for the calculation of the precision and recall as it is based on an IoU threshold that we determine if a detection is considered a TP or a FP.

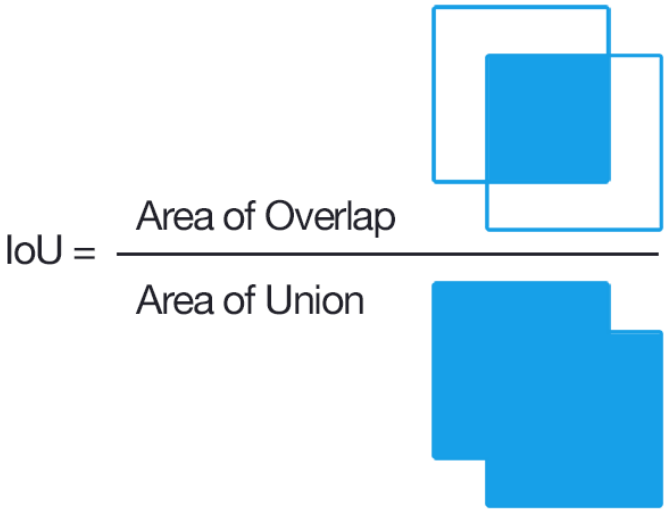


FIGURE 2.3: Intersection over Union
Source: obtained from [7]

The most common evaluation metric is mean Average Precision (mAP) which is a number between 0 and 100. Despite it's name the mAP is not just the average of precisions. Average Precision (AP) is the area under the Precision-Recall (PR) curve and is computed for each class.

The PR curve is represented with the x-axis being recall and the y-axis being precision. In order to obtain multiple Precision-Recall value pairs we change the score cutoff. The score cutoff defines the level of confidence that is necessary for a detection to be considered valid meaning that detections with a confidence value below the score cutoff are not considered as detections and, as we adjust the score cutoff, we can calculate the precision and recall for each score cutoff and obtain our PR curve [8]. We can see an example of a PR curve in Figure 2.4.

The mAP is the average of the AP over the different classes. In order to determine if a detection is a TP the IoU of that detection must be above a certain threshold. So, in order to be more clear about the IoU threshold we have used, we can add it to the mAP, thus obtaining something more specific like mAP@0.5 which is the mAP with an IoU threshold of 0.5.

Object Detection was used even before the advent of CNNs in this area of study. It was a several step process, which started with edge detection and feature extraction, performed by using techniques like Scale-invariant Feature Transform

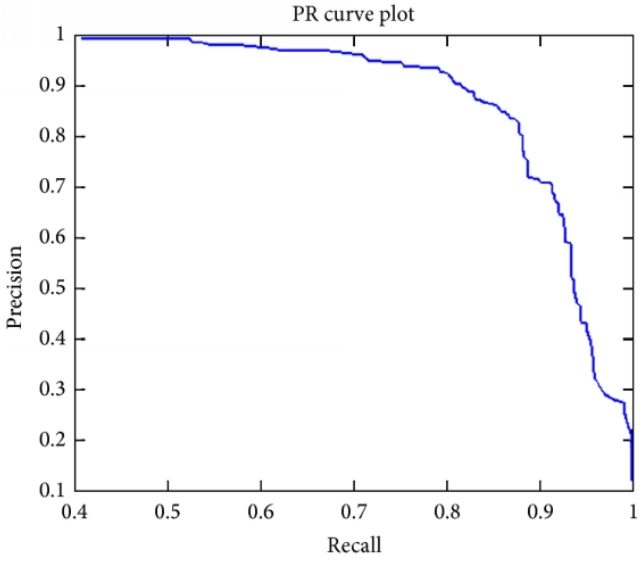


FIGURE 2.4: Precision-Recall curve
Source: obtained from [8]

(SIFT) and Histogram of Oriented Gradients (HOG). After this first step the images were then compared with already existing Support Vector Machine (SVM) templates of objects, usually at multi scale levels, in order to detect and localize the objects present in the image [7]. The comparison process, on different scales is very well demonstrated in Figure 2.5.

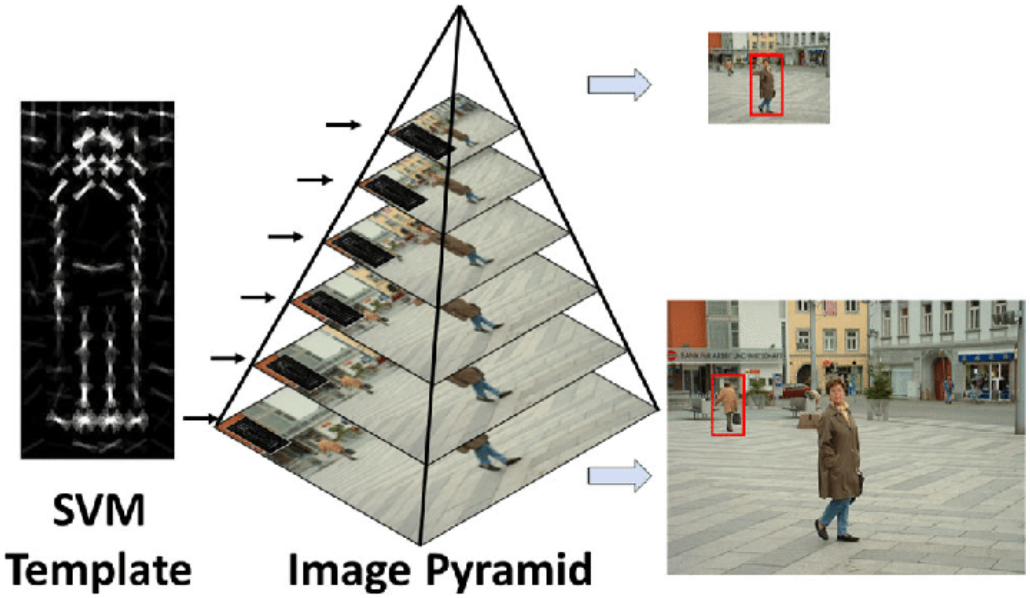


FIGURE 2.5: Object Detection process before Deep Learning
Source: obtained from [7]

CNNs were first introduced in the 1980s by Yann LeCun. The early version

of CNNs, named LeNet after it's creator, was able to recognize handwritten digits. These CNNs were mostly adopted by the banking and postal services, where they were used to read zip codes on envelopes and digits on checks.

A CNN performs convolution operations which slide a predefined kernel (usually called filter) over the input feature map, multiplying and adding the values of the filter and the input features in order to produce an output. The output values form an output matrix [9]. We can observe this process in Figure 2.6. By adjusting the values of our filter we can give different weights to different features, filtering the features we are interested in.

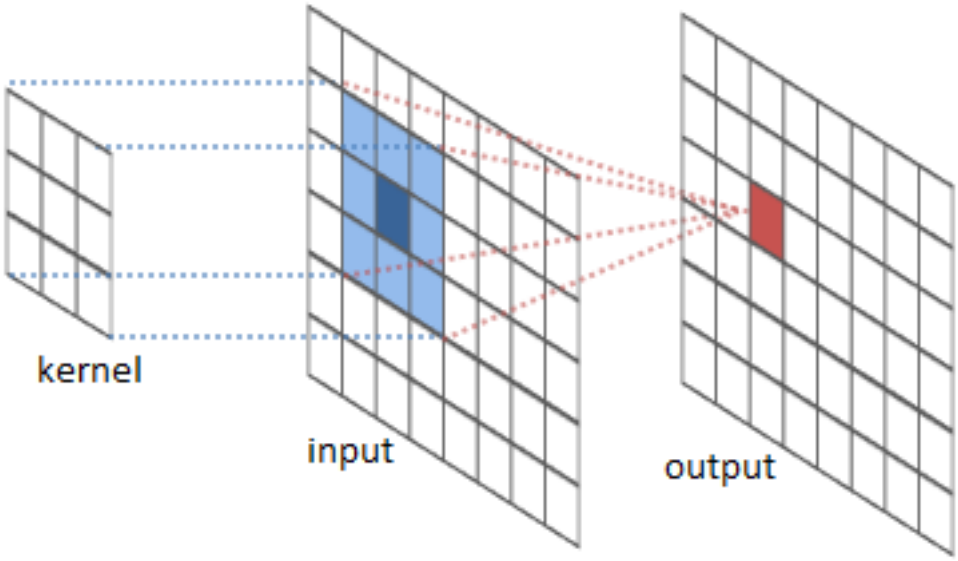


FIGURE 2.6: Illustration of the convolution operation
Source: obtained from [10]

Even though these CNNs were ingenious, they faced a serious roadblock: they could not scale, and because of that they remained on the sidelines of Computer Vision and Artificial Intelligence. CNNs required a lot of data and computing power to work efficiently on large images and, due to the limitations of the time, they were only applicable to images with low resolutions.

It was only in 2012 that AlexNet [11] showed us that perhaps the time had come to revisit CNNs. With the availability of large datasets and the increasing computing power available, researchers were now able to create complex CNNs capable of tackling Computer Vision problems that were previously impossible to overcome [12].

Now that the available CNNs were capable of performing bigger tasks we could apply them to Object Detection, but first we need to understand how they work.

Convolutional Neural Networks are composed of multiple layers of artificial neurons, which are mathematical functions that calculate the weighted sum of multiple inputs and give us an activation value, as we can observe in Figure 2.7. The behaviour of these artificial neurons is defined by their weights. They receive the pixel values and pick out various visual features [12].

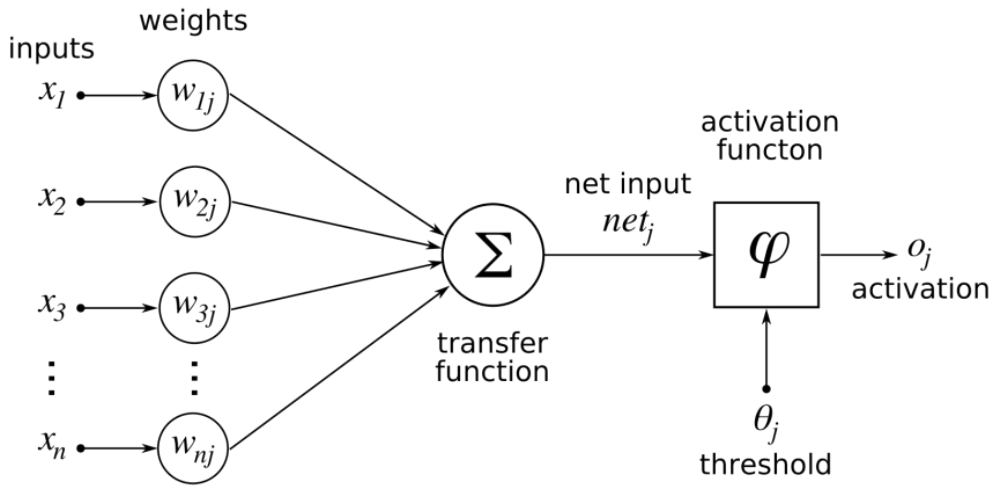


FIGURE 2.7: The structure of an artificial neuron
Source: obtained from [12]

When we input an image into a CNN, each of its layers creates several activation maps. These activation maps highlight the relevant features of the image. The first layer of the CNN detects basic features such as horizontal, vertical and diagonal edges. The output of this layer is then passed as input to the second layer, which extracts more complex features, like corners and combinations of edges. This process of passing the output of a layer to the following layer as input continues and, as we move deeper into the network, the layers begin detecting higher-level features such as objects and faces. Figure 2.8 gives us a good visualization of the functioning of these layers.

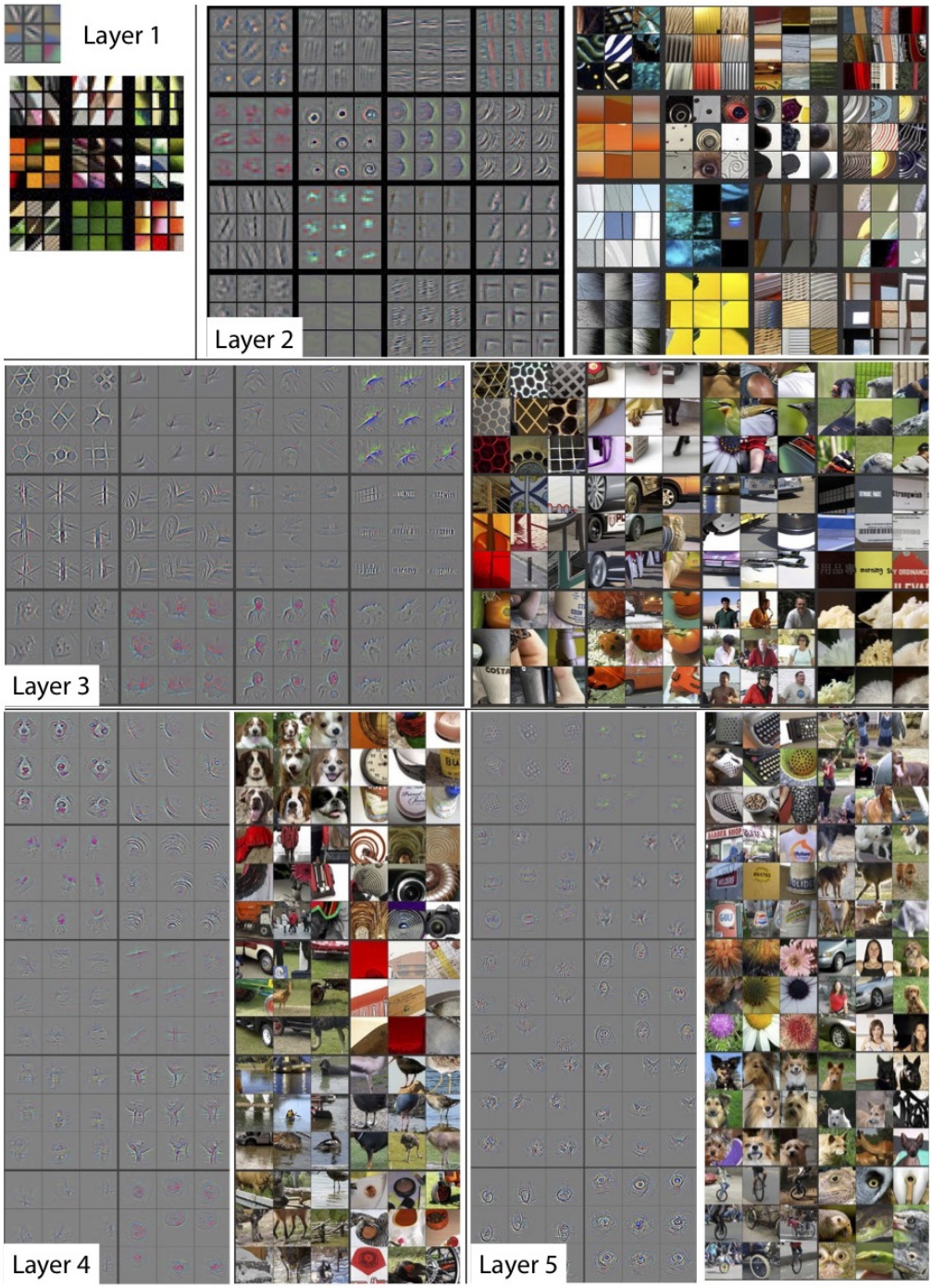


FIGURE 2.8: Neural Network layers visualization
Source: obtained from [12]

Using the activation map of the final layer, the classification layer outputs confidence scores that specify the likelihood of the image belonging to a class. These confidence scores have a value between 0 and 1 and the classification layer outputs one for each class that the CNN is prepared to detect. A simplification of the whole

detection process performed by a CNN can be seen in Figure 2.9.

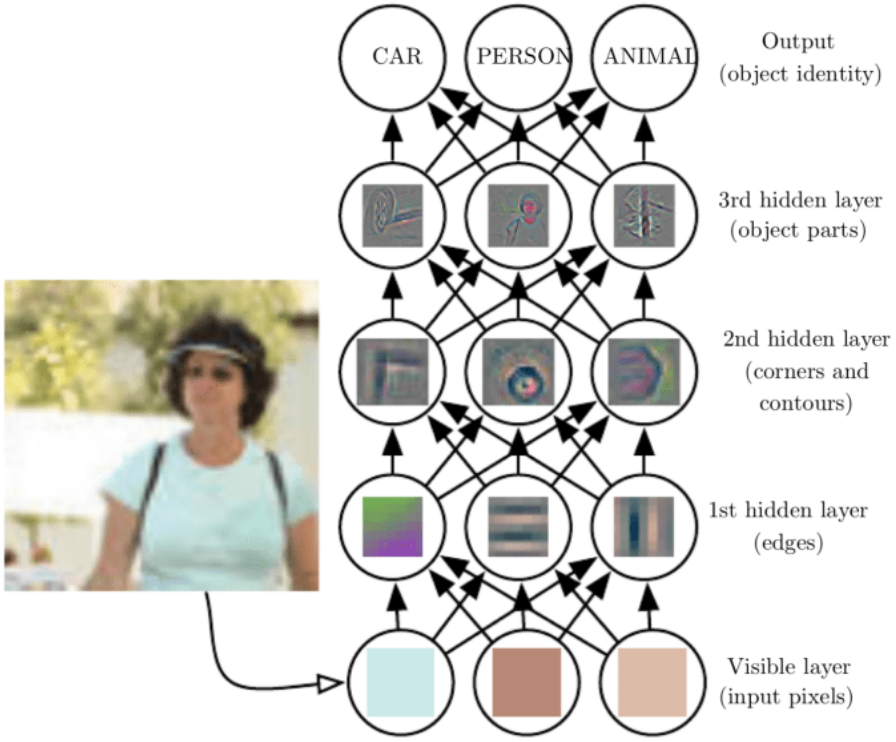


FIGURE 2.9: Simplification of the process performed by a CNN
Source: obtained from [12]

After the introduction of the revisited CNN, Two-Step Object Detection architectures began being developed. Two-Step Object Detection involves algorithms that perform the identification of bounding boxes that may contain objects and then proceed to classify each bounding box separately.

The first step utilizes a Region Proposal Network (RPN) which provides a number of regions that are then passed to common Deep Learning based classification architectures as shown in Figure 2.10.

These Two-Step architectures started with the development of the Region-based Convolutional Neural Network (R-CNN). R-CNN combined region proposal with CNNs, creating regions with CNN features and producing an increase of mAP by 30% relative to the previous best result and achieving a mAP of 53.3% [14].

The evolution continued as Fast R-CNN and Faster R-CNN, newer and improved versions of R-CNN, were published. Fast R-CNN improved both time, as it was able to train the deep detection network 9 times faster than it's predecessor, and precision, reaching a mAP of 66% [15]. Faster R-CNN further improved on the

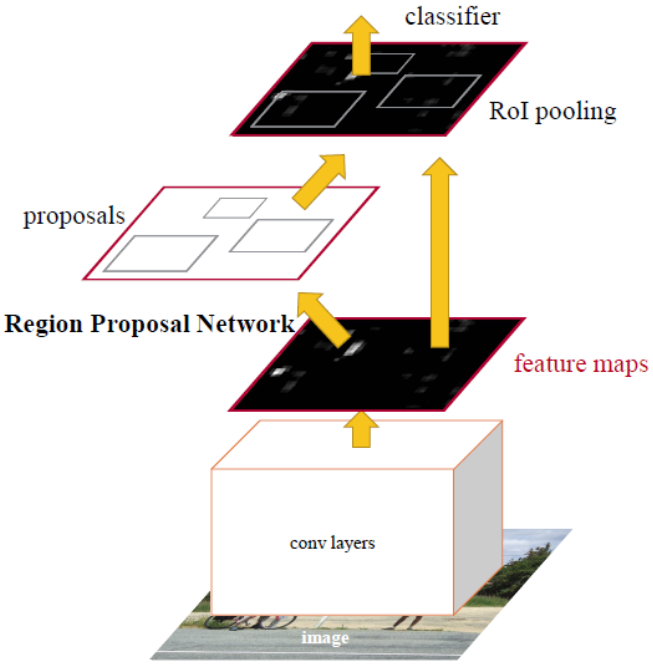


FIGURE 2.10: Two-Step Object Detection
Source: obtained from [13]

processing speed, making the process almost real-time, by introducing "RPN that share convolutional layers with state-of-the-art object detection networks" [16].

Two-Step Object Detection algorithms are known to perform better than their One-Step counterparts but that comes at the cost of time. As Two-Step Object Detection couldn't handle the need for real time Object Detection, many One-Step Object Detection architectures were proposed, such as YOLO [17], YOLOv2 [18], YOLOv3 [19], Single Shot Detector (SSD) [20], RetinaNet [21] and YOLOv4 [22], which tried to combine the detection and classification steps.

YOLO performs detection as a straightforward regression dilemma, taking an input image and learning the class possibilities with bounding box coordinates. YOLO divides the image into a grid of S x S and every grid predicts N bounding boxes and their respective confidence. The end result is a total of SxSxN forecasted bounding boxes, a large number which can be narrowed down by setting a threshold to exclude the bounding boxes with lower confidence scores.

SSD runs a CNN on the input image only once and computes a feature map. A small 3x3 convolutional kernel is then run on this feature map to predict the bounding boxes and their categorization probability.

RetinaNet was introduced to fill in some of the imbalances and inconsistencies of other Single-Step detectors like YOLO and SSD when dealing with extreme foreground-background classes. RetinaNet accommodates Focal Loss, which is a method used to prevent negatives from clouding the detector.

At the time of writing of this thesis YOLO is the Object Detection Network that shows better results in detection speed while maintaining a good mAP. YOLOv3 showed very good speed with a mAP that is comparable to other methods as we can observe in Figure 2.11. YOLOv4 further improved on YOLOv3 and displayed the best results in terms of speed while maintaining a performance comparable with other state-of-the-art Object Detectors such as EfficientDet. In Figure 2.12 we can see the results of YOLOv4 and other Object Detectors on the MS COCO dataset.

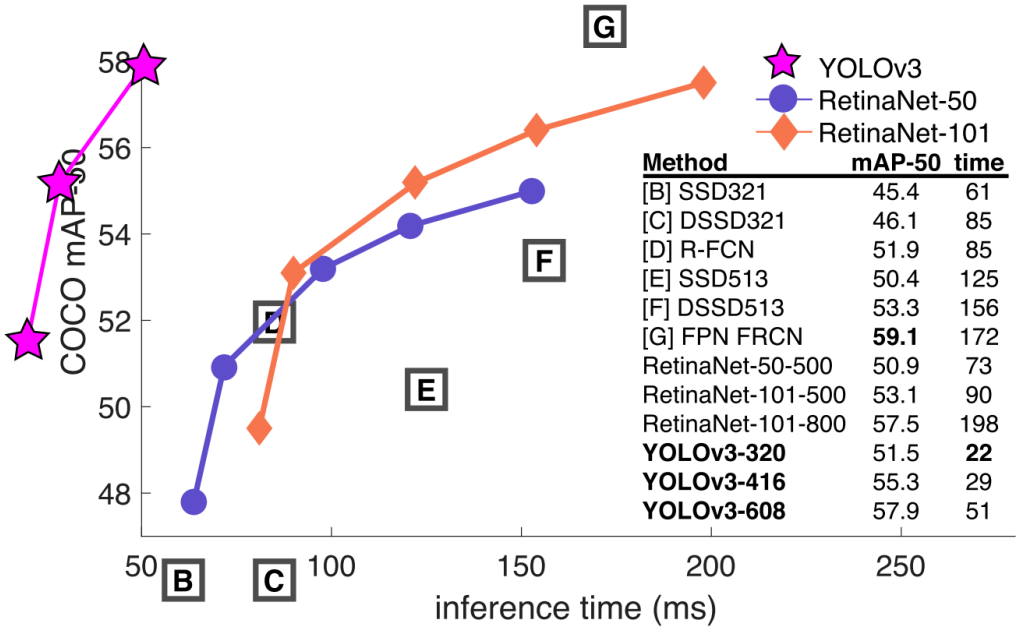


FIGURE 2.11: Speed/Accuracy comparison of YOLOv3 with other state-of-the-art Object Detectors
Source: obtained from [19]

For our work we decided to utilize the YOLO object detection network which has shown very good results and has had several improvements in the past years. We will be using YOLOv4-tiny [23], a scaled version of the YOLOv4 that can operate on low end Graphics Processing Unit (GPU) devices and can therefore perform real-time detection on-board the UAV.

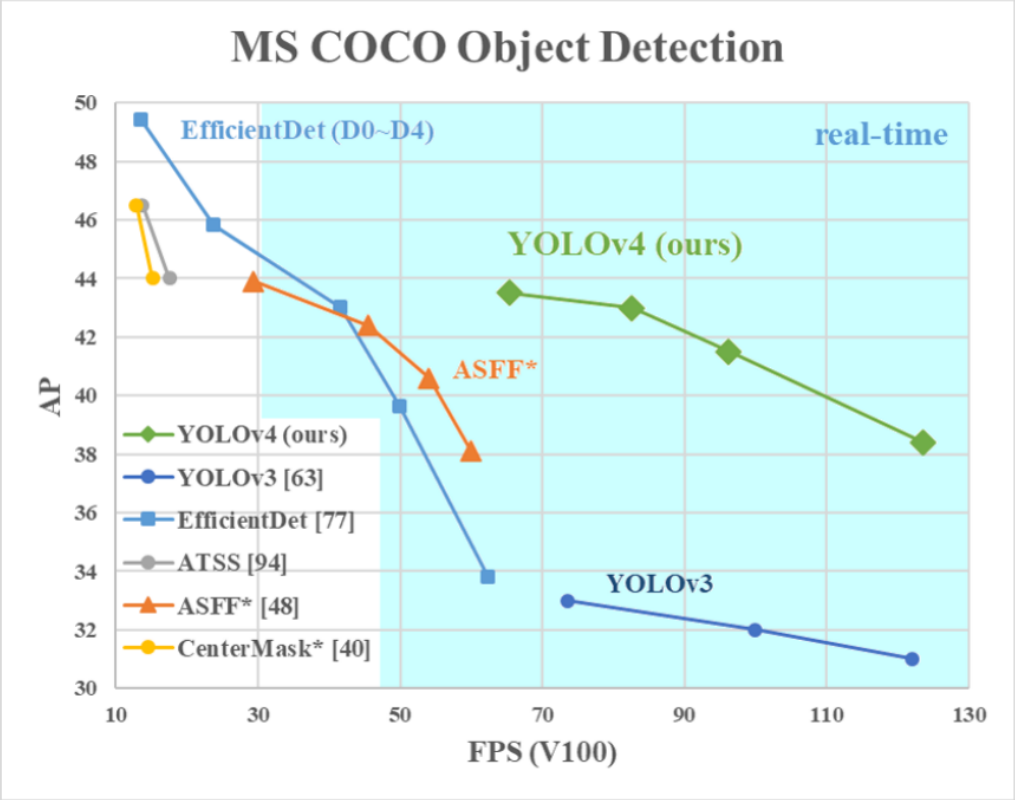


FIGURE 2.12: Comparison of YOLOv4 with other state-of-the-art Object Detectors
Source: obtained from [22]

2.1.2 You Only Look Once

The YOLO object detection network was introduced in the 2016 paper by Redmon et al, and brought a new approach to object detection and classification by performing both tasks in a single neural network. "A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation". This architecture is able to perform extremely fast processing, reaching real-time at 45 frames per second (fps) and even reaching 155 fps with it's smaller version, Fast YOLO at a slight cost in accuracy [17]. YOLO continued to improve with the appearance of new versions with YOLOv4 being introduced in 2020 by Bochkovskiy et al [22].

YOLOv4 consist of a backbone, a neck and a head. The backbone is a deep neural network composed mainly of convolutional layers and its main objective is to extract essential features. For the backbone they have used the CSPDarknet53, an application of Cross-Stage-Partial-connections (CSP) to the Darknet53 that was previously used in YOLOv3. CSP is derived from the DenseNet architecture and results in different dense layers repeatedly learning copied gradient information. By

applying CSP it is possible to "greatly reduce the amount of computation, and improve inference speed as well as accuracy" [24].

The neck has the essential role of collecting feature maps from different stages. For the neck, they have used Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN). SPP generates a "fixed-length representation regardless of image size/scale" which eliminates the requirement for fixed-size input images [25]. PAN aims at "boosting information flow in proposal-based instance segmentation framework" and is applied in order to help include features from different layers [26].

The head performs dense prediction, the final prediction which is composed of a vector containing the coordinates of the predicted bounding box (center, height, width), the confidence score of the prediction and the label. For the head, they chose to useYOLOv3. They use Complete Intersection over Union (CIoU)-loss, Cross mini-Batch Normalization (CmBN), DropBlock regularization, Self-Adversarial-Training (SAT), Mish activation, Spatial Attention Module (SAM) and a few more tools in order to obtain the final prediction. With this detector they were able to obtain faster and more accurate results than all available alternative detectors at the time of publishing.

In order to adapt to the computing power available there is a need to adjust the scale of our YOLOv4 network. With Scaled-YOLOv4, Wang, Bochkovskiy and Liao were able to develop YOLOv4-large designed for cloud GPU and YOLOv4-tiny designed for low-end GPU devices. "The YOLOv4-tiny model achieves 22.0% AP (42.0% AP50) at a speed of 443 fps on RTX2080Ti, while by using TensorRT, batch size = 4 and FP16-precision the YOLOv4-tiny achieves 1774 fps". YOLOv4-tiny uses CSPOSANet with Partial in Computational Block (PCB) architecture to form the backbone of YOLOv4. [23].

2.1.3 Data Augmentation

In order to perform an adequate training that results in as little overfitting as possible we need to have a large and varied dataset. "Data Augmentation approaches overfitting from the root of the problem, the training dataset.This is done under the assumption that more information can be extracted from the original dataset through augmentations.". When working with limited datasets Data Augmentation can help "artificially inflate the training dataset size by either data warping or oversampling". With data warping we make use of augmentations such as "geometric and color transformations, random erasing, adversarial training, and neural style

transfer" to transform existing images while keeping their labels intact while with oversampling the augmentations "create synthetic instances (...) [which] includes mixing images, feature space augmentations, and Generative Adversarial Networks (GAN)." [27].

Data Augmentation can be performed using several different augmentation techniques. We will take a look at the more popular ones such as Flips, Rotations, Scaling, Cropping, Translations and Gaussian noise.

A flip consists of mirroring the image either horizontally or vertically as displayed in Figure 2.13.

FIGURE 2.13: From the left, we have the original image, followed by the image flipped horizontally, and then the image flipped vertically
Source: obtained from [28]

A rotation consists of, as the name implies, rotating the image by any number of degrees. The example on Figure 2.14 applies 90 degrees rotations.

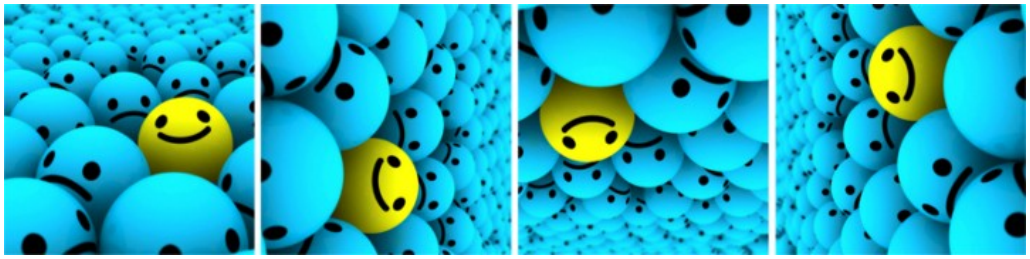


FIGURE 2.14: The images are rotated by 90 degrees clockwise with respect to the previous one, as we move from left to right
Source: obtained from [28]

Scaling allows us to scale the image outward or inward. Most image frameworks used maintain the image's original size. We can see this scaling in Figure 2.15.

Cropping is a process similar to scaling but we randomly sample a section from the original image and resize it to the original image's size. We can notice the

FIGURE 2.15: From the left, we have the original image, the image scaled outward by 10%, and the image scaled outward by 20%
Source: obtained from [28]

difference between this method and scaling by comparing Figure 2.15 and Figure 2.16.

FIGURE 2.16: From the left, we have the original image, a square section cropped from the top-left, and then a square section cropped from the bottom-right. The cropped sections were resized to the original image size
Source: obtained from [28]

Translation involves moving the image so that the position of the objects changes. As we are working with CNNs the position of the object has little difference and makes it so that this augmentation is of little use for our work. In Figure 2.17 we can see the results obtained from applying this technique.

Gaussian noise can be applied to the image in order to simulate some distortion. We can apply Gaussian noise or a toned down version called salt and pepper noise, which presents itself as random black and white pixels randomly spread through the image. Both of these techniques can be seen in effect in Figure 2.18.

There are also some more advanced data augmentation techniques such as conditional GAN which can perform more complex augmentations. Conditional GANs can transform an image from one domain to a different one such as taking an

FIGURE 2.17: From the left, we have the original image, the image translated to the right, and the image translated upwards
Source: obtained from [28]

FIGURE 2.18: From the left, we have the original image, image with added Gaussian noise, image with added salt and pepper noise
Source: obtained from [28]

input image taken in summer and outputting that same image in a winter setting or vice-versa. They can also alter the time of day of the image, allowing for a big variety of outputs. The outputs generated by GANs can sometimes appear more artistic than realistic but they can still be useful to enlarge some datasets. We can see some examples of the applications of GANs in Figures 2.19 and 2.20.

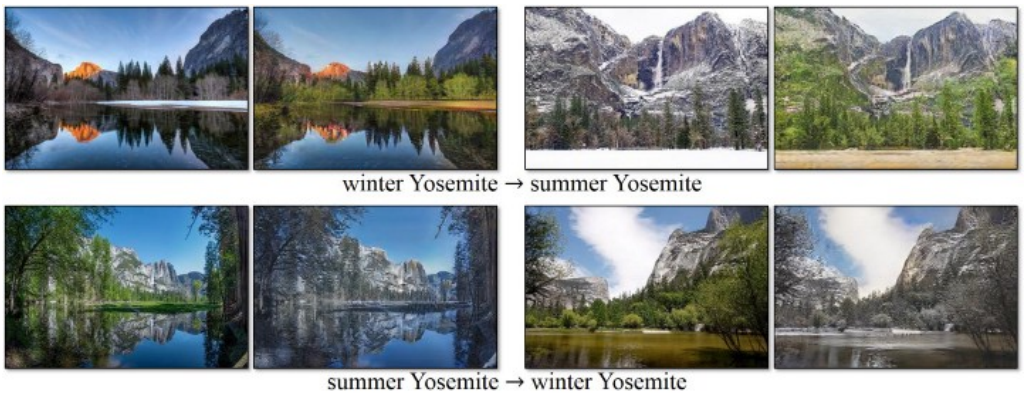


FIGURE 2.19: Changing seasons using a CycleGAN
Source: obtained from https://junyanz.github.io/CycleGAN/

FIGURE 2.20: Deep Photo Style Transfer
Source: obtained from https://arxiv.org/abs/1703.07511

2.2 Related Works

The use of object detection at sea has been studied recently and has shown very positive results, especially regarding the detection and classification of ships at sea. By taking a look at the work done in the detection of ships we can get some valuable information on the methods used and try to used them to improve our detection of castaways at sea.

The detection of people using Object Detection, mostly done in pedestrian detection, is also of interest. We will analyse some of the existing methods used to detect people in order to understand what has been done and what can be applied to the castaway scenario.

2.2.1 Object Detection at sea

The study of the use of machine vision for detection of castaways is not something new. As early as 1994 there have been works done in this area. Sumimoto et al. present a study in 1994 that dealt with the use of image processing techniques for the detection of rescue targets. They proposed a system that utilized color and shape information with aerial footage in order to detect possible rescue targets. The proposed system configuration can be seen in Figure 2.21 [29].

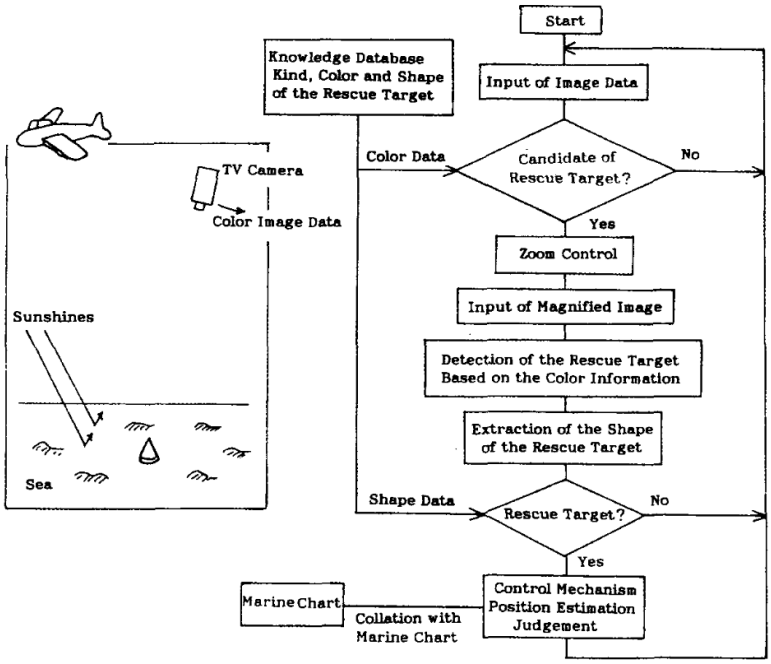


FIGURE 2.21: System proposed by Sumimoto et al.
Source: Obtained from [29]

The work published by Mace in 2011 [30] tackled the issue of detecting marine debris, such as fishing nets, buoys and ropes. This work did not use Object Detection models like the ones we have explained before. They utilized data from satellite observations and aircraft in conjunction with mathematical models to observe debris. Unfortunately, the conclusion of this work was that the detection and removal of these debris remained a serious challenge to current technology at the time of publishing.

The work published by Ramirez et al. in 2011 [31] focused on sea rescue. They proposed a coordinated sea rescue system based on UAV and Unmanned Surface Vessel (USV). In the system presented in this paper "the USV in charge of rescuing the castaways benefits from the information provided by a searching and sensing UAV, which is capable of locating the castaways quicker than the USV. The coordinated use of both vehicles reduces the search time, because the USV can predict the castaway positions better, using the information provided by its remote sensing UAV and its more powerful Central Processing Unit (CPU)". The two subsystems were able to work in real-time with really simple defined behaviours . This system presents a good basis where the improvements of automatic Object Detection can be applied.

"Started in 2012, the ICARUS research project aims to develop robotic

tools which can assist 'human' operator during SAR operations" [32]. The Integrated Components for Assisted Rescue and Unmanned Search operations (ICARUS) project, was a European project that counted with the participation of 24 partners from 9 countries the goal of which was to develop autonomous robots that could help SAR teams both on land and sea. There were various thesis and publications that contributed for the work done on this project, which resulted in the development of several tools that can be used to assist in SAR situations. The project results can be accessed at `http://www.fp7-icarus.eu/project-results`.

Detection at sea can be done using methods which do not use UAVs and camera footage, and one of them is the use of Unmanned Underwater Vessel (UUV). The 2015 thesis by Palma [33], done as part of the ICARUS project, studied the possibility of an underwater approach to castaway detection using a sonar upward looking system . This work showed that, although it is possible to perform detection of castaways using UUVs, it is very limited and would require further improvements to be a feasible option.

In 2017, Hoai and Phuong published a paper that studied the use of anomaly color detection on UAV images for SAR works. They were able to determine that in different SAR situations "the appropriate color space and detection algorithm can be chosen that give best performance of anomaly detection" [34].

In the same year, Dinnbier et al. published a paper that focused on using Gaussian Mixture Model (GMM) and Fourier Transforms for target detection in UAV maritime Search And Rescue. Using GMMs they were able to remove background from images while leaving moving targets intact as can be seen in Figure 2.22. By combining GMMs with Fourier Transforms they were able to obtain improved results as shown in Figure 2.23 [35].

In their 2019 thesis, Laxmi uses, modifies and trains "four of the state-of-art pretrained deep learning network models, namely VGG16, Xception, ResNet50 and InceptionResNet" to identify ships from images captured from UAVs. In this work they are able to achieve high accuracy (99.6 to 99.9% correct classifications) but to do so, require high computing power [36].

On the subject of object detection at sea, the work done by Pires in their 2020 thesis studies the use of a two stages cascade model with a detection part followed by a segmentation stage to perform real-time ship segmentation during maritime surveillance missions using onboard cameras [37].

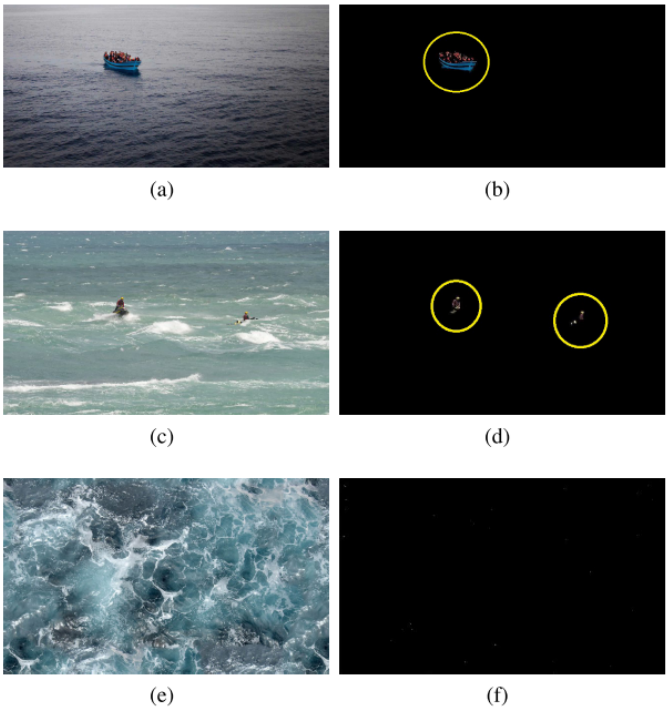


FIGURE 2.22: Original images on the left, with the salient images on the right (the yellow circles have been added for visibility)
Source: obtained from [35]

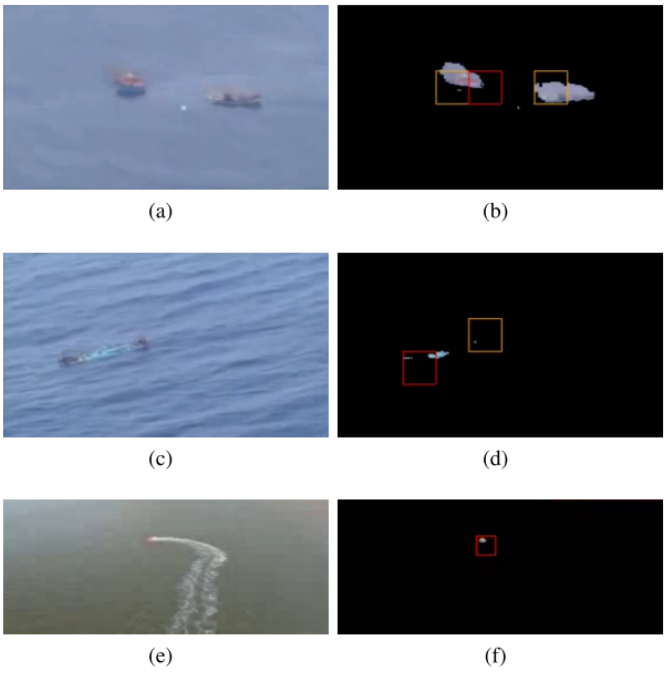


FIGURE 2.23: Original images are on the left, with output images on the right, where salient pixels show GMM detection and colored frames show FFT detection on the combinatorial method
Source: adapted from [35]

2.2.2 Detection of People

Pedestrian Detection, much like other forms of object detection, was originally performed without the use of CNNs. The 2013 paper by Ouyang and Wang studies a joint approach to pedestrian detection, proposing a joint deep learning framework that encompasses four main components: Feature Extraction, Part Deformation Handling, Occlusion Handling and Classification. An overview of these four key components can be seen in Figure 2.24. The proposed network was able to outperform the best performing approaches at the time, reducing the average miss rate by 9% on the Caltech dataset [38].

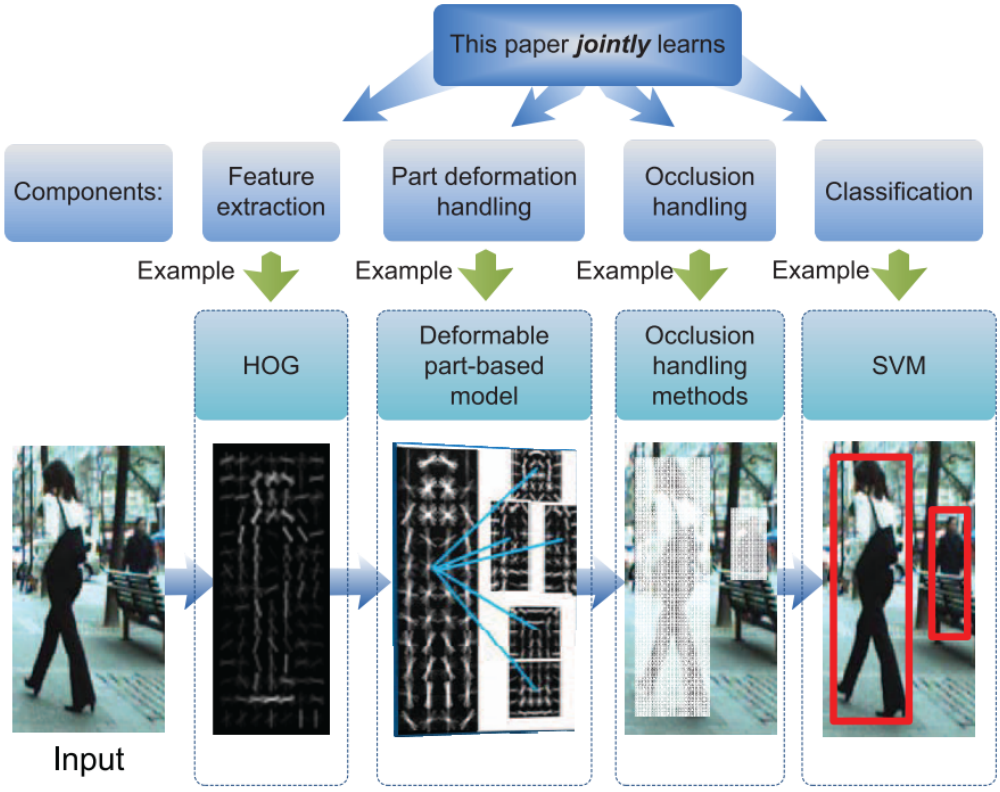


FIGURE 2.24: Four key components in pedestrian detection: feature extraction, deformation handling models, occlusion handling models, and classifiers
Source: obtained from [38]

In 2016, Tomè et al. proposed a pedestrian detection system based on CNNs. The proposed system utilized the scores provided by the region proposal algorithm in parallel with the scores provided by the trained CNN to classify a given region, therefore improving the system accuracy. The proposed system was able to outperform other alternative approaches based on both handcrafted and

learned features with their lightweight version being capable of detecting pedestrians in real-time using modern hardware [39].

The works we have seen focus on the detection of people in the visual spectrum but there are other methods such as the detection of people using thermal Infrared (IR) cameras. The paper published by by Teutsch et al. in 2014 studied the use of a moving thermal IR camera for low resolution person detection. In this paper, they proposed "a two-stage person recognition approach for Long-Wave Infrared (LWIR) images" that was able to perform real-time detection of people in LWIR images acquired by moving cameras. Some of the results of this paper can be seen in Figure 2.25 [40].

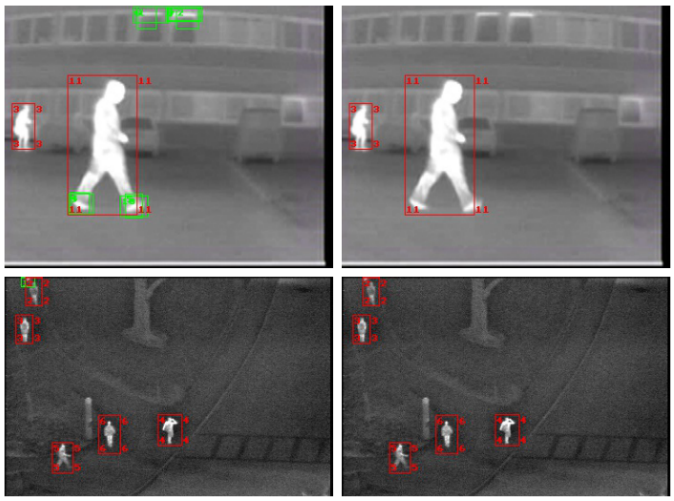


FIGURE 2.25: Results from the Teutsch et al. paper
Source: adapted from [40]

Chapter 3

Methodology

In order to develop our Castaway detector we must start by defining our methodology. In this chapter we will explain the process used to work with the selected Object Detection Network and how the training and testing were performed. We will also elaborate on the contents of each of the datasets and the augmentations applied on each of them to create our augmented datasets.

3.1 Detection Framework

To perform automatic detection we needed to implement the YOLOv4-tiny Object Detection network we have previously mentioned in Chapter 2.

We started with an implementation in Tensorflow 2.0 by Viet Hung [41]. With this implementation we were able to use YOLOv4 and YOLOv4-tiny on our local computer. We then proceeded to perform some experimentation on a small sample of footage in order to verify its capability to detect the already existing person class in different environments.

For the training of our network used the Darknet implementation by Bochkovskiy [42], which is the usual method used when performing training of YOLO networks.

The YOLOv4-tiny network consists of 38 layers and its structure can be seen in Figure 3.1. The structure presented in Figure 3.1 can be better seen in Annex I. For the initial tests and as a point of comparison we utilized the YOLOv4-tiny pre-trained weights, trained with the COCO dataset with 80 different object classes.

The resulting weights of our work can be later applied on a lightweight computer for embedded applications like a Jetson Nano. The process of utilizing

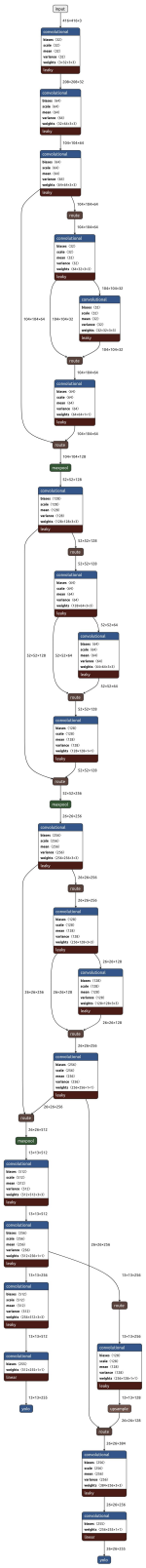


FIGURE 3.1: YOLOv4-tiny network structure

YOLOv4-tiny on a Jetson Nano is quite simple and a quick guide on how to prepare the Jetson Nano is available at: `https://tinyurl.com/JetsonNanoTutorial`.

3.2 Datasets

In order to perform an adequate training we must first prepare a suitable dataset that is able to represent the castaway at sea. To generate the bounding boxes we used Make Sense, a free to use online tool for labelling images. With this tool we were able to obtain our bounding boxes in the YOLO format [43].

For the purpose of this thesis we have prepared several datasets. In our datasets surfers, kayakers and swimmers were considered as part of the castaway class.

Our first dataset consists of gathered UAV footage of people swimming and surfing at different beaches at different times of day obtained from Youtube. This dataset consists of 550 images and has an average of 13 objects per image with the number of objects on a single image varying between 1 and 76. Some examples of the images that make up this dataset can be seen in Figure 3.2.

FIGURE 3.2: Examples of images contained in Dataset 1

For our second dataset we used a DJI Mini 2 to collect footage of a castaway and people kayaking near the Oeiras Marina in the morning with a clear sky and fair sea conditions. The DJI Mini 2 camera specifications can be seen in Table 3.1. We were able to obtain 67 images with an average of 2 objects per image with most

DJI Mini 2	
Sensor	1/2.3" CMOS
Aperture	f/2.8
Focus range	1m to ∞
ISO range	100-3200
Video resolution	4K: 3840×2160 @ 24/25/30fps

TABLE 3.1: Camera Specifications of the DJI Mini 2

Parrot Anafi	
Sensor	1/2.4" CMOS
Aperture	f/2.4
Focus range	1.5m to ∞
ISO range	100-3200
Video resolution	4K: 3840×2160 @ 24/25/30fps

TABLE 3.2: Camera Specifications of the Parrot Anafi

images containing 2 objects. Some examples of the images captured to create this dataset can be observed in Figure 3.3.

FIGURE 3.3: Examples of images contained in Dataset 2

From the 67 images captured at the Oeiras Marina only 53 were used to create Dataset 2 because 20% of the images were separated to add to the Test Dataset.

For our third dataset we used a Parrot Anafi to collect footage of cadets from the Portuguese Naval School on a field exercise. The footage in this dataset consists of people swimming across a small canal in the afternoon. The Parrot Anafi was able to capture video at a resolution of 3840×2160 at 30 fps. We were able to obtain 292 images with an average of 5 objects per image with the number of objects varying between 0 and 7. Some examples of the images captured to create this dataset are displayed in Figure 3.4.

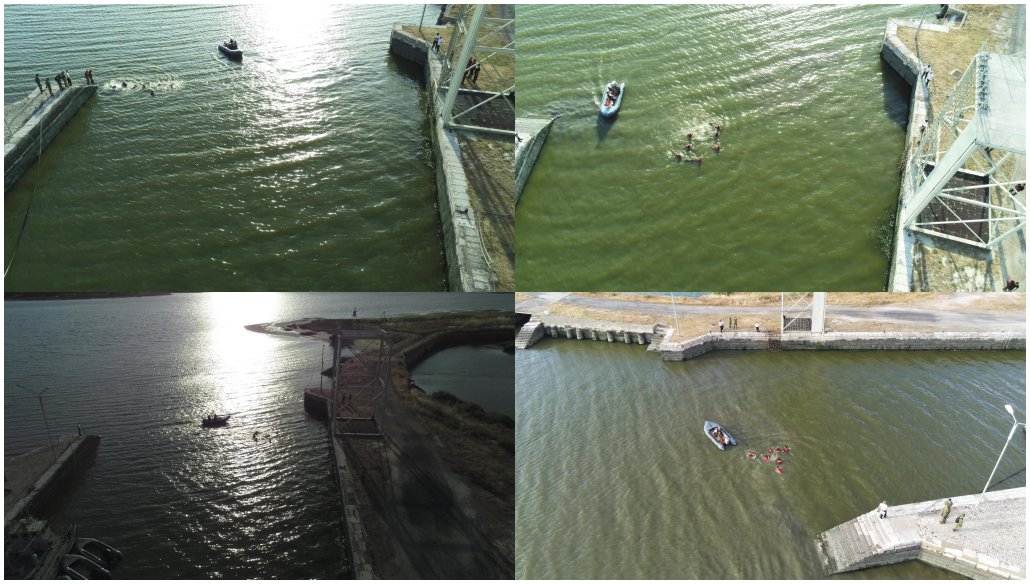


FIGURE 3.4: Examples of images contained in Dataset 3

Dataset	Images	Average number of objects
Dataset 1	550	13
Dataset 2	53	2
Dataset 3	233	5
Dataset 123	836	10
Test Dataset	73	4

TABLE 3.3: Datasets created before augmentations

From the 292 images captured at the TROIA 21 Exercise only 233 were used to create Dataset 3 because 20% of the images were separated to add to the Test Dataset.

For training purposes we created the Dataset 123 which consists of all images from Datasets 1, 2 and 3.

For the testing of our detector, we created a Test Dataset composed of 20% of the images from Datasets 2 and 3, which will not be used in the training process and will be kept separate until the test phase. This dataset is composed of 73 images. The information of the datasets mentioned above can be seen in Table 3.3.

With our datasets annotated we can then proceed with the necessary data augmentation in order to obtain a sufficiently large and varied dataset.

We used Roboflow, an online tool that "organizes, prepares, and improves your image and annotation training data" [44], to perform our augmentations. We used this tool to perform the rotations, HUE alterations and Flips on the images

from Dataset 1 and utilized the images obtained as our first augmented dataset for training.

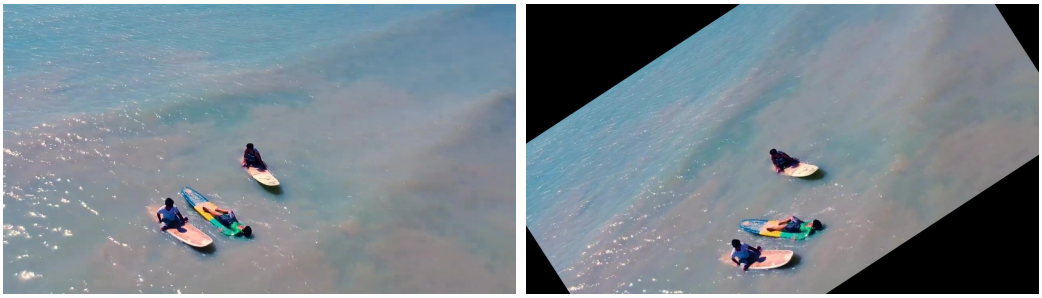


FIGURE 3.5: Random Rotation

FIGURE 3.6: HUE alteration -50º (left) and +50º (right)

FIGURE 3.7: Horizontal Flip (left) and Vertical Flip (right)

After the augmentations we obtained the Augmented Dataset 1, with 9137 images.

For the creation of the Augmented Dataset 123, which will contain all images from Dataset 123, we used Roboflow to perform the rotations, HUE alterations and Flips. After the creation of Augmented Dataset 1 we came to the conclusion that there was an excess of rotations applied, which led to an unnecessarily big dataset, and reduced the number of rotations when creating the Augmented Dataset, as a result we obtained the Augmented Dataset with 4964 images.

The information of the Augmented Datasets mentioned above can be seen in Table 3.4.

Dataset	Original Images	Images after augmentation
Augmented Dataset 1	550	9137
Augmented Dataset 123	836	4964

TABLE 3.4: Datasets created after augmentations

3.3 Training

In order for our detection to work properly we began by training YOLOv4-tiny using the first augmented dataset to create a whole new Castaway class. This allowed us to train our own class from the start. Our training begins with the pre-trained YOLOv4-tiny weights we mentioned at the beginning of this chapter and obtains weights for a single new class: castaway.

In order to have a more user friendly environment we used Google Colaboratory, a tool from Google which "allows you to write and execute Python in your browser, with zero configuration required, free access to GPUs, easy sharing" [45]. The training was performed on a Google Colaboratory notebook, a slightly modified version of the notebook from the tutorial on training a custom YOLOv4-tiny object detector [46]. Our notebook is publicly available[1]. With this notebook we were able to use the Darknet implementation by Bochkovskiy [42] to train our YOLOv4-tiny with our custom class.

The initial training was performed with the configurations suggested in [46]. The training was performed using a batch size of 64 with 16 subdivisions, a network size of 416 x 416, maximum number of batches set to 10000 and the steps adjusted accordingly. We also changed the number of filters on the convolutional layers before each YOLO layer to 18, and the number of classes on each of the YOLO layers to 1, as that is our number of classes. The number of filters in the convolutional layers is related to the number of classes using the following formula obtained from [46].

$$\text{Filters} = (\text{classes} + 5) \times 3. \tag{3.1}$$

It is important to note that YOLOv4-tiny already performs image augmentations in its training process. The augmentations applied by YOLOv4-tiny are saturation, exposure and hue and their default values are set to 1.5, 1.5 and 0.1, respectively.

[1]Notebook available at `https://tinyurl.com/YOLOv4TinyTrainingNotebook`

Test	Description
Test 1	Evaluation of the results obtained after training with Augmented Dataset 1
Test 2	Comparison between retraining with Weights 1 and training from scratch
Test 3	Evaluation of the influence of the use of augmented images for training
Test 4	Evaluation of the influence of changing the meta-parameters
Test 5	Ablation study

Table 3.5: Experiments performed

3.4 Testing

In order to obtain the best possible results we tested the results of training with different Datasets and configurations, such as varying the batch size, learning rate and other meta-parameters.

We began by performing the first training with the Augmented Dataset 1 to obtain our first weights which we named Weights 1 for future reference.

We proceeded with a comparison of the results obtained from retraining the Weights 1 with Dataset 123 and from training from scratch with Dataset 123. This allowed us to understand if there is any benefit from performing a pre-training with our Augmented Dataset 1.

After understanding the influence of the pre-training we needed to understand the influence of data augmentations on the performance of our network. We performed the training with the Augmented Dataset and compared the results with the ones obtained from training with Dataset 123.

It was then necessary to test the influence of changing some of our meta-parameters. We evaluated the performance using different batch and subdivision sizes, training augmentations and learning rates, in order to determine the configuration that provides us the best detection.

At this point it was necessary to perform an ablation study. Ablation is the removal of a component of an AI system, and an ablation study helps us understand the contribution of the different components, in our case, the different datasets, on the overall system. In this section we verified the importance of each dataset and tried to understand how much their absence could influence our results.

The experiments mentioned above are summarized in Table 3.5.

For the purpose of measuring our results in most of our tests we used the Test Dataset. After each training was complete, we received the mAP of those

weights in reference to this Test Dataset. The results obtained are presented in Chapter 4 and the resources used such as datasets and weights obtained are publicly available[2].

[2]Resources available at `https://tinyurl.com/TrainAndTestResources`

Chapter 4

Results

In this chapter we will be presenting the results obtained from our work.

It is important to note that the average time of most trainings was 12 hours but this time sometimes varied due to the fact that the resources provided by Google Colaboratory, such as GPUs, varied over time and there was no way to choose what type of GPU we were connected to.

4.1 Obtaining the first weights

After the 1st training, which was performed with the Augmented Dataset 1, we obtained a low mAP on the Test Dataset. The weights obtained were saved as yolov4-tiny-custom-best-1stTraining. The result was a 7.10% mAP which can be easily explained by two factors, the fact that the images in Dataset 1 did not contain many examples with intense light reflection on the water and that there are a lot of false positives on the images obtained from Dataset 3 due to the number of people outside of the water that are not considered castaways. In figure 4.1 we can see an example of an image where there are several of the support staff present at the exercise that were outside of the water and can confuse the detector.

We can see that there is an improvement in the detection of castaways in good visibility conditions when compared to the Original weights in the images in Figure 4.2. But we can also see that the current results are still lacking when it comes to bad visibility conditions, as displayed in Figure 4.3.

Weights	mAP on Test Dataset
yolov4-tiny-custom-best-1stTraining	7.10%

Table 4.1: Weights obtained from Test 1

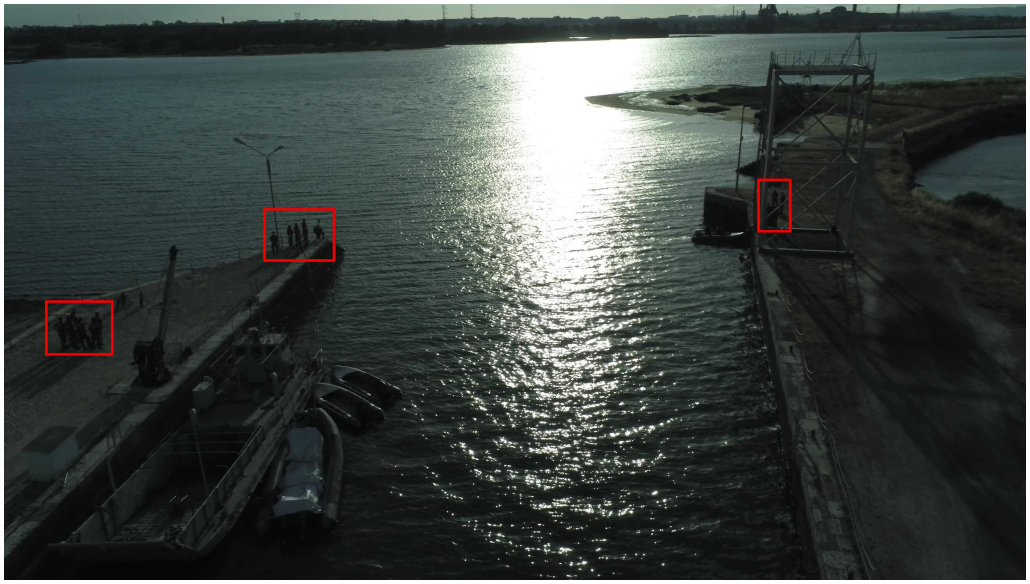


FIGURE 4.1: Example of support staff outside of the water

FIGURE 4.2: Comparison between Detections on test images from Oeiras Marina with Original YOLOv4-tiny weights (left) and Detections with our first trained weights (right)

4.2 Pre-training vs Training

We began by performing the training with Dataset 123 which gave us a mAP of 38.73%. The weights obtained were saved as yolov4-tiny-custom-best-All-Datasets. We then performed the retraining and obtained a mAP of 39.65%. The weights obtained were saved as yolov4-tiny-custom-best-Retrain. The increase in the mAP, compared to the one obtained after the initial training, occurs in both

FIGURE 4.3: Comparison between Detections on test images from TROIA 21 Exercise with Original YOLOv4-tiny weights (left) and Detections with our first trained weights (right)

Weights	mAP on Test Dataset
yolov4-tiny-custom-best-All-Datasets	38.73%
yolov4-tiny-custom-best-Retraining	39.65%

TABLE 4.2: Weights obtained from Test 2

cases and seems to be a result of the inclusion of the images from Dataset 3 that allow our detector to better understand the difference between people outside of the water and castaways.

We can also see that there is little difference between the retraining and training and so we will be performing our trainings from scratch instead of using our first weights.

4.3 Influence of Data Augmentation

After performing the training with Augmented Dataset 123 we were able to obtain a mAP of 35.78% which is slightly lower than the one obtained with the original images without augmentation. The weights obtained were saved as yolov4-tiny-custom-best-Augmentations. This result leads us to conclude that, in this particular case, the use of augmentations does not improve our detection and may even cause a slight decrease. It is important to remember that the training

Weights	mAP on Test Dataset
yolov4-tiny-custom-best-Augmentations	35.78%

Table 4.3: Weights obtained from Test 3

process of YOLOv4-tiny already applies some augmentations and that can be the reason the impact of previous augmentations is so small.

4.4 Influence of meta-parameters

All training in this section will be performed with Dataset 123.

For the first training in this section we tried changing our batch size to 32 and our subdivisions to 8 in order to verify the impact on the performance. The weights obtained were saved as yolov4-tiny-custom-best-Batch32-Sub8. With these meta-parameters we were able to obtain a mAP of 37.00% which is slightly lower than the results obtained with the original parameters.

We proceeded by training with batch size set to 128 and subdivisions set to 32 and we were able to achieve a mAP of 42.51%. This mAP is the highest that we were able to achieve so far but it comes at the cost of a higher training time which is a little more than 2 times longer than with the original parameters. The weights obtained were saved as yolov4-tiny-custom-best-Batch128-Sub32.

The next training we performed had the objective of understanding the impact of the augmentations within YOLO itself. We set the hue meta-parameter to 0.5 while maintaining the saturation and exposure meta-parameters with their original value of 1.5. The reason we maintained the saturation and exposure values was due to the fact that the default values were already at a good value while the default hue value was set to a considerably low value. We were able to obtain a mAP of 40.46%, a slight improvement over the original meta-parameters. The weights obtained were saved as yolov4-tiny-custom-best-Hue05.

We decided that it was time to try varying the learning rate. We continued training with the configuration of values used on the previous training and the learning rate set to 0.01 to understand if it could influence our results. We were able to obtain a mAP of 42.29%, a slight improvement on our previous result while maintaining a training time similar to the average of the trainings. The weights obtained were saved as yolov4-tiny-custom-best-LRate01.

Weights	mAP on Test Dataset
yolov4-tiny-custom-best-Batch32-Sub8	37.00%
yolov4-tiny-custom-best-Batch128-Sub32	42.51%
yolov4-tiny-custom-best-Hue05	40.46%
yolov4-tiny-custom-best-LRate01	42.29%

Table 4.4: Weights obtained from Test 4

From the results obtained we made the decision to maintain the original settings except for the Hue meta-parameter which was set to 0.5 and the Learning rate which was set to 0.01. We did not use the increased batch and subdivision sizes because the increase of training time was too big when compared to the increase of mAP.

4.5 Ablation Study

Now that we had made slight adjustments to our training process and meta-parameters we proceeded with our tests to verify the influence of different train and test datasets.

Our first tests were simple, we tested the yolov4-tiny-custom-best-LRate01 weights on Dataset 1, Dataset 2 and Dataset 3 separately in order to see the performance on each individual dataset.

With the test set to Dataset 1 we were able to obtain a mAP of 28.32%. With test set to Dataset 2 we obtained a mAP of 81.43%. With test set to Dataset 3 we obtained a mAP of 38.58%. This was an interesting result as the Dataset with which we achieved the best mAP was the one that is less present in the training dataset. This seems to be due to the fact that the castaways in Dataset 2 are more clearly defined and there are less objects present that can hinder the detection. We will take special attention to the results of the next set of tests to see if this trend continues.

The next test we performed was training with Datasets 1, 2 and 3 separately and testing the performance on the datasets not used on each of the trainings as well as the Test Dataset.

We performed training with Dataset 1 and tested the performance on the Test Dataset, Dataset 2 and Dataset 3, obtaining a mAP of 10.45%, 21.19% and 9.48%, respectively. The weights obtained were saved as yolov4-tiny-custom-best-Dataset1.

Weights	mAP Test	mAP Dataset 1	mAP Dataset 2	mAP Dataset 3
LRate01	42.29%	33.70%	80.95%	45.42%
Dataset1	10.45%	16.89%	21.19%	9.48%
Dataset2	8.39%	0.35%	100%	0.11%
Dataset3	35.29%	4.11%	4.32%	49.65%

TABLE 4.5: Results from the ablation study

Training with Dataset 2 and testing the performance on the Test Dataset, Dataset 1 and Dataset 3, we obtained a mAP of 8.39%, 0.35% and 0.11%, respectively. The weights obtained were saved as yolov4-tiny-custom-best-Dataset2.

Performing the training with Dataset 3 and testing the performance on the Test Dataset, Dataset 1 and Dataset 2, we were able to obtain a mAP of 35.29%, 4.11% and 4.32%, respectively. The weights obtained were saved as yolov4-tiny-custom-best-Dataset3.

We also tested the weights obtained on their respective datasets and, as expected, they showed improved results. Dataset 1 had a mAP of 16.89%, while Dataset 2 had a mAP of 100% and Dataset 3 had a mAP of 49.65%. These results can be mostly attributed to overfitting due to testing on the images that were used for training. The effect of the overfitting is clearly visible on Dataset 2, which had less variety than the other datasets and a lower number of images and led to a great mAP when testing on Dataset 2 but the lowest when testing on Datasets 1 and 3. Although the yolov4-tiny-custom-best-Dataset2 weights show a low performance on all the other datasets, it is interesting to note that Dataset 2 is the dataset that best represents castaway situations in clear conditions. This could mean that these weights could be used in clear conditions to obtain a good performance while the yolov4-tiny-custom-best-Hue05 weights give us a more robust performance overall.

These tests provide us some interesting results. None of the weights obtained using individual Datasets was able to outperform the weights obtained using all the Datasets, with all of these weights obtaining relatively low mAP on most Datasets. This is likely caused by the lack of variety of each individual dataset. It is interesting to note that the variety of Dataset 1, which is definitely higher than the other datasets, allowed for a relatively good mAP when testing on Dataset 2.

4.6 Final Weights

With all the tests concluded we have selected the weights yolov4-tiny-custom-best-LRate01 as our final weights. These weights showed the best results overall, with good mAP on all the Datasets which means that it has a good detection ability on a wide range of conditions.

In Figure 4.4 we can observe the precision-recall curve obtained with detection threshold values between 0.1 and 0.9. The precision only varies slightly while the recall decreases significantly with each increase of the detection threshold. From this graph we are able to determine that a threshold of 0.4 should be used in order to achieve a good balance between precision and recall, achieving a precision of 68% with a recall of 55%.

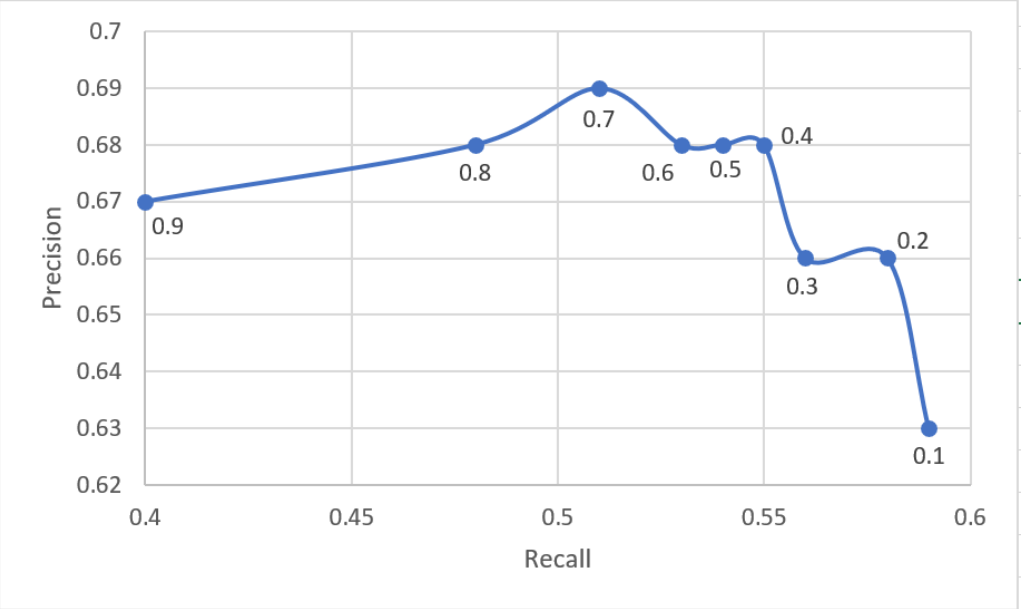


FIGURE 4.4: Precisio-Recall curve for our final weights

The results from the tests performed are summarized in Table 4.6.

A comparison between the results of the first training and the results of the final weights can be seen in Figure 4.5 and Figure 4.6. Figure 4.6 shows the most difference, with castaways being better detected and the people on the shore no longer mislabelled as castaways.

Weights	mAP on Test Dataset
yolov4-tiny-custom-best-1stTraining	7.10%
yolov4-tiny-custom-best-All-Datasets	38.73%
yolov4-tiny-custom-best-Retraining	39.65%
yolov4-tiny-custom-best-Augmentations	35.78%
yolov4-tiny-custom-best-Batch32-Sub8	37.00%
yolov4-tiny-custom-best-Batch128-Sub32	42.51%
yolov4-tiny-custom-best-Hue05	40.46%
yolov4-tiny-custom-best-LRate01	**42.29%**
yolov4-tiny-custom-best-Dataset1	10.45%
yolov4-tiny-custom-best-Dataset2	8.39%
yolov4-tiny-custom-best-Dataset3	35.29%

TABLE 4.6: Results from all of the Tests

FIGURE 4.5: Comparison between the results of our first training and our final on images from Oeiras Marina

FIGURE 4.6: Comparison between the results of our first training and our final on images from TROIA 21 Exercise

Conclusion

During the course of this thesis we studied the works that stand as a basis for modern Object Detection Networks and the applications of these networks. We took a look at the studies related to SAR operations, especially the ones related to the use of UAVs. After having a better understanding of the existing works we were able to select an Object Detection network that fit our needs. We selected the YOLOv4-tiny for its ability to perform real-time detection while maintaining a good performance.

We proceeded by testing the effects of different training conditions on our selected network in order to improve its ability to detect a castaway, experimenting with different configurations to obtain the best results. The initial results seemed promising and as we proceeded with the tests we were able to obtain increasingly better results, with the final chosen weights showing a mAP of 42.29%.

The results obtained, although very positive, can still be further improved with the addition of more datasets which could provide even more varied examples of castaways that improve the capability of detection on different conditions.

The work done in this thesis serves as a proof of concept, showing the possible application of the YOLO detection network, more specifically the YOLOv4-tiny detection network, in SAR operations using UAVs. The introduction of a system of UAVs with Object Detection capabilities could be an important addition to the already existing SAR resources, providing a way to extend the area covered.

In this thesis we prepared the weights for the Object Detection network that can be applied on a UAV, but we could not perform the practical application on-board. There is still much that can be done in future works and we suggest the development and implementation of the on-board components for use with a UAV.

Bibliography

[1] IMO and ICAO. *IAMSAR Manual*. Vol. III. 1998. URL: https://www.imo.org/en/OurWork/Safety/Pages/IAMSARManual.aspx.

[2] Serokell. *Top Areas for Machine Learning in 2020 | by Serokell | Better Programming | Medium*. 2020. URL: https://medium.com/better-programming/the-top-areas-for-machine-learning-in-2020-4c880bf5e288 (visited on 01/29/2021).

[3] Yarrow Eady. *Tesla's Deep Learning at Scale: Using Billions of Miles to Train Neural Networks | by Yarrow Eady | Towards Data Science*. 2019. URL: https://towardsdatascience.com/teslas-deep-learning-at-scale-7eed85b235d3 (visited on 01/29/2021).

[4] T S Huang. "Computer Vision: Evolution and Promise". In: *Report* (1997).

[5] Lars Hulstaert. *A Beginner's Guide to Object Detection - DataCamp*. URL: https://www.datacamp.com/community/tutorials/object-detection-guide (visited on 06/01/2021).

[6] Pulkit Sharma. *Image Classification vs. Object Detection vs. Image Segmentation | by Pulkit Sharma | Analytics Vidhya | Medium*. URL: https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81 (visited on 06/02/2021).

[7] Prakhar Ganesh. *Object Detection : Simplified. Take a peek into the world of one of... | by Prakhar Ganesh | Towards Data Science*. URL: https://towardsdatascience.com/object-detection-simplified-e07aa3830954 (visited on 06/01/2021).

[8] Yangfen Liu. *The Confusing Metrics of AP and mAP for Object Detection / Instance Segmentation | by Yanfeng Liu | Medium*. URL: https://yanfengliux.medium.com/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef (visited on 06/21/2021).

[9] Lilian Weng. *Object Detection for Dummies Part 2: CNN, DPM and Overfeat*. URL: https://lilianweng.github.io/lil-log/2017/12/15/object-recognition-for-dummies-part-2.html (visited on 06/21/2021).

[10] Intel Labs. *River Trail*. URL: http://intellabs.github.io/RiverTrail/tutorial/ (visited on 06/21/2021).

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017). ISSN: 15577317. DOI: 10.1145/3065386.

[12] Ben Dickson. *What are convolutional neural networks (CNN)? – TechTalks*. URL: https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/ (visited on 06/08/2021).

[13] Shivy Yohanandan. *mAP (mean Average Precision) might confuse you! | by Shivy Yohanandan | Towards Data Science*. URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2 (visited on 06/07/2021).

[14] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2014), pp. 580–587. ISSN: 10636919. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524.

[15] Ross Girshick. "Fast R-CNN". In: *Proceedings of the IEEE International Conference on Computer Vision* 2015 Inter (2015), pp. 1440–1448. ISSN: 15505499. DOI: 10.1109/ICCV.2015.169. arXiv: 1504.08083.

[16] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[17] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016), pp. 779–788. ISSN: 10636919. DOI: 10.1109/CVPR.2016.91. arXiv: arXiv:1506.02640v5.

[18] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, faster, stronger". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua (2017), pp. 6517–6525. DOI: 10.1109/CVPR.2017.690. arXiv: 1612.08242.

[19] Joseph Redmon and Ali Farhadi. "YOLOv3: An incremental improvement". In: *arXiv* (2018). ISSN: 23318422. arXiv: 1804.02767.

[20] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: (). DOI: 10.1007/978-3-319-46448-0. URL: https://github.com/weiliu89/caffe/.

[21] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2017. DOI: 10.1109/ICCV.2017.324. arXiv: 1708.02002v2. URL: https://github.com/facebookresearch/Detectron..

[22] Alexey Bochkovskiy, Chien Yao Wang, and Hong Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *arXiv* (2020). ISSN: 23318422. arXiv: 2004.10934.

[23] Chien Yao Wang, Alexey Bochkovskiy, and Hong Yuan Mark Liao. "Scaled-YOLOv4: Scaling cross stage partial network". In: *arXiv* (2020). ISSN: 23318422. arXiv: 2011.08036.

[24] Chien Yao Wang et al. "CSPNet: A new backbone that can enhance learning capability of CNN". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2020-June (2020), pp. 1571–1580. ISSN: 21607516. DOI: 10.1109/CVPRW50498.2020.00203. arXiv: 1911.11929.

[25] Kaiming He et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8691 LNCS.PART 3 (2014), pp. 346–361. ISSN: 16113349. DOI: 10.1007/978-3-319-10578-9_23. arXiv: 1406.4729.

[26] Shu Liu et al. "Path Aggregation Network for Instance Segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018), pp. 8759–8768. ISSN: 10636919. DOI: 10.1109/CVPR.2018.00913. arXiv: 1803.01534.

[27] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (2019). ISSN: 21961115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[28] Arun Gandhi. *Data Augmentation | How to use Deep Learning when you have Limited Data*. URL: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ (visited on 06/11/2021).

[29] Tetsuhiro Sumimoto et al. "Machine Vision for Detection of the Rescue Target in the Marine Casualty". In: *Proceedings of IECON'94 - 20th Annual Conference of IEEE Industrial Electronics* 3 (1994), pp. 723–726.

[30] Thomas H. Mace. "At-sea detection of marine debris: Overview of technologies, processes, issues, and options". In: *Marine Pollution Bulletin* 65.1-3 (2012), pp. 23–27. ISSN: 0025326X. DOI: 10.1016/j.marpolbul.2011.08.042. URL: http://dx.doi.org/10.1016/j.marpolbul.2011.08.042.

[31] Francisco Fernández Ramírez et al. "Coordinated sea rescue system based on unmanned air vehicles and surface vessels". In: *OCEANS 2011 IEEE - Spain* (2011). DOI: 10.1109/Oceans-Spain.2011.6003509.

[32] ICARUS. *FP7-Icarus*. URL: http://www.fp7-icarus.eu/ (visited on 06/21/2021).

[33] Tiago Filipe Ramião Ramos da Palma. "ICARUS - Busca e Salvamento Utilizando Deteção Sonar". PhD thesis. Escola Naval, 2015. URL: http://comum.rcaap.pt/handle/10400.26/11241.

[34] Dao Khanh Hoai and Nguyen Van Phuong. "Anomaly color detection on UAV images for search and rescue works". In: *Proceedings - 2017 9th International Conference on Knowledge and Systems Engineering, KSE 2017* 2017-Janua.October 2017 (2017), pp. 287–291. DOI: 10.1109/KSE.2017.8119473.

[35] Nuria Martinez Dinnbier et al. "Target detection using Gaussian mixture models and fourier transforms for UAV maritime search and rescue". In: *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017* (2017), pp. 1418–1424. DOI: 10.1109/ICUAS.2017.7991312.

[36] Laxmi Thapa. "Ship recognition on the sea surface using aerial images taken by Uav: a deep learning approach". PhD thesis. Universidade Nova de Lisboa IMS, 2019, pp. 1–83. URL: http://hdl.handle.net/10362/63805.

[37] Carlos David Coito Pires. "Ship Segmentation in Aerial Images for Maritime Surveillance". PhD thesis. Instituto Superior Técnico, 2020.

[38] Wanli Ouyang and Xiaogang Wang. "Joint deep learning for pedestrian detection". In: *Proceedings of the IEEE International Conference on Computer Vision* (2013), pp. 2056–2063. DOI: 10.1109/ICCV.2013.257.

[39] D. Tomè et al. "Deep Convolutional Neural Networks for pedestrian detection". In: *Signal Processing: Image Communication* 47 (2016), pp. 482–489. ISSN: 09235965. DOI: 10.1016/j.image.2016.05.007. arXiv: 1510.03608.

[40] Michael Teutsch et al. "Low resolution person detection with a moving thermal infrared camera by hot spot classification". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (2014), pp. 209–216. ISSN: 21607516. DOI: 10.1109/CVPRW.2014.40.

[41] Vit Hùng. *GitHub - hunglc007/tensorflow-yolov4-tflite: YOLOv4, YOLOv4-tiny, YOLOv3, YOLOv3-tiny Implemented in Tensorflow 2.0, Android. Convert YOLO v4 .weights tensorflow, tensorrt and tflite*. URL: https://github.com/hunglc007/tensorflow-yolov4-tflite (visited on 03/26/2021).

[42] Alexey Bochkovskiy. *GitHub - AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)*. URL: https://github.com/AlexeyAB/darknet (visited on 06/22/2021).

[43] Piotr Skalski. *Make Sense*. URL: https://www.makesense.ai/ (visited on 08/23/2021).

[44] Joseph Nelson, Matt Brems, and Brad Dwyer. *Overview - Roboflow*. URL: https://docs.roboflow.com/ (visited on 03/26/2021).

[45] Google. *Welcome to Colaboratory - Colaboratory*. URL: https://colab.research.google.com/notebooks/intro.ipynb (visited on 03/30/2021).

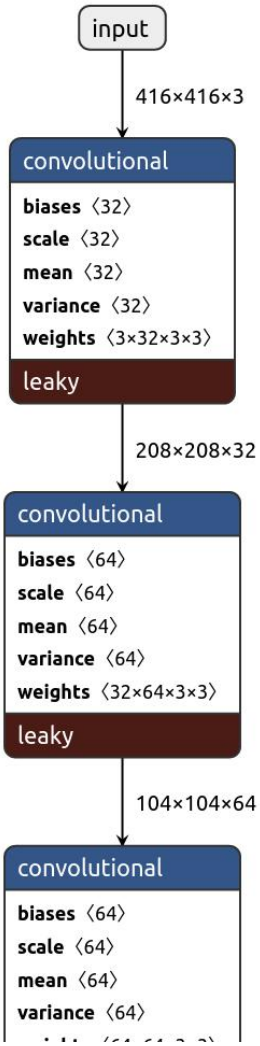
[46] Techzizou. *TRAIN A CUSTOM YOLOv4-tiny OBJECT DETECTOR USING GOOGLE COLAB | by Techzizou | Analytics Vidhya | Medium*. URL: https://medium.com/analytics-vidhya/train-a-custom-yolov4-tiny-object-detector-using-google-colab-b58be08c9593%7B%5C#%7Da70f (visited on 06/18/2021).

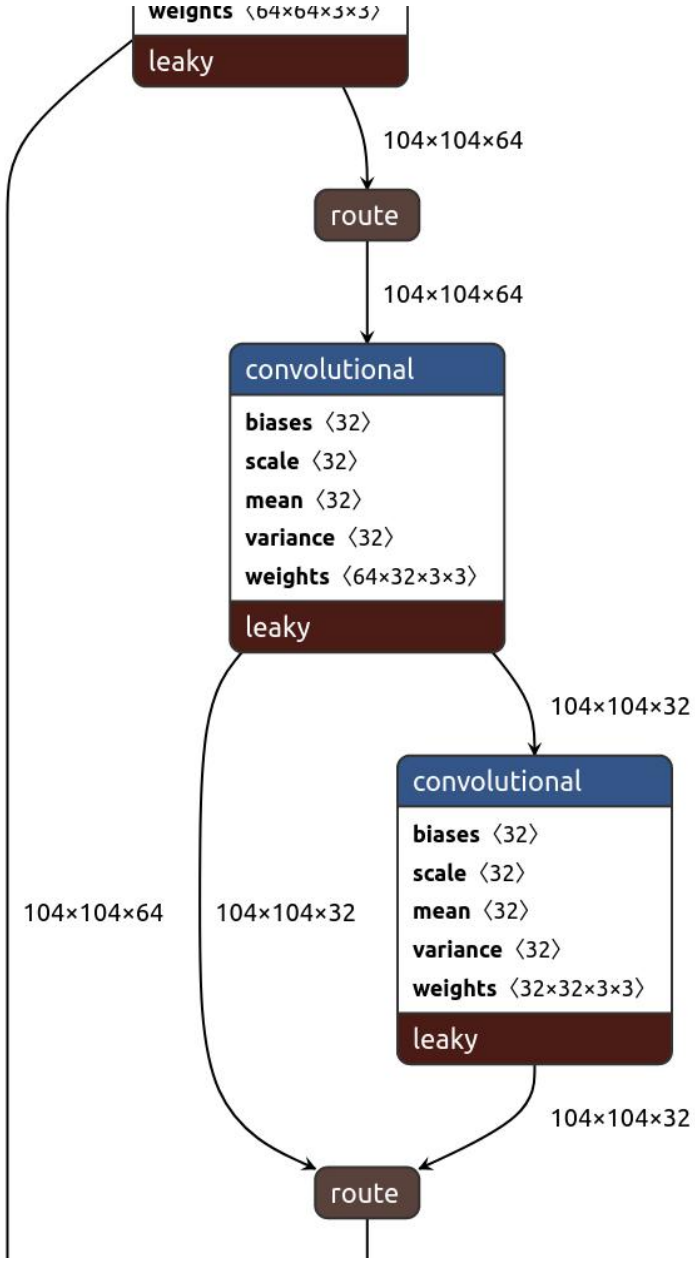
[47] *Visualize yolov4-tiny and yolov4 network structure diagram - Programmer Sought*. URL: https://www.programmersought.com/article/20944423185/ (visited on 08/23/2021).

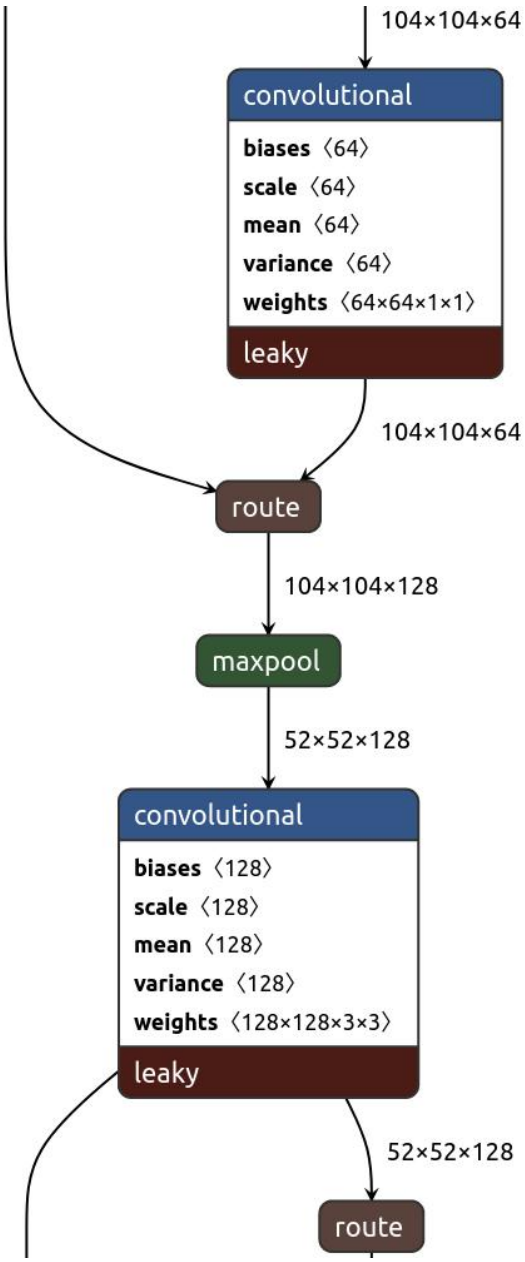
Annex I

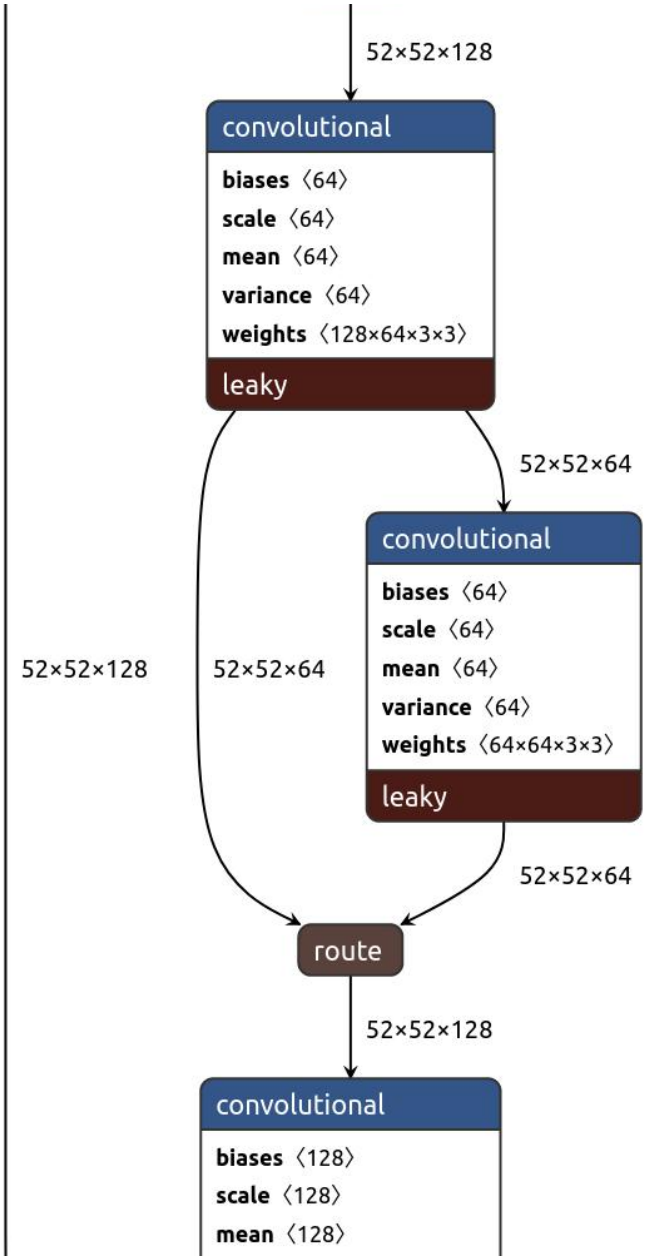
YOLOv4-tiny Network Structure

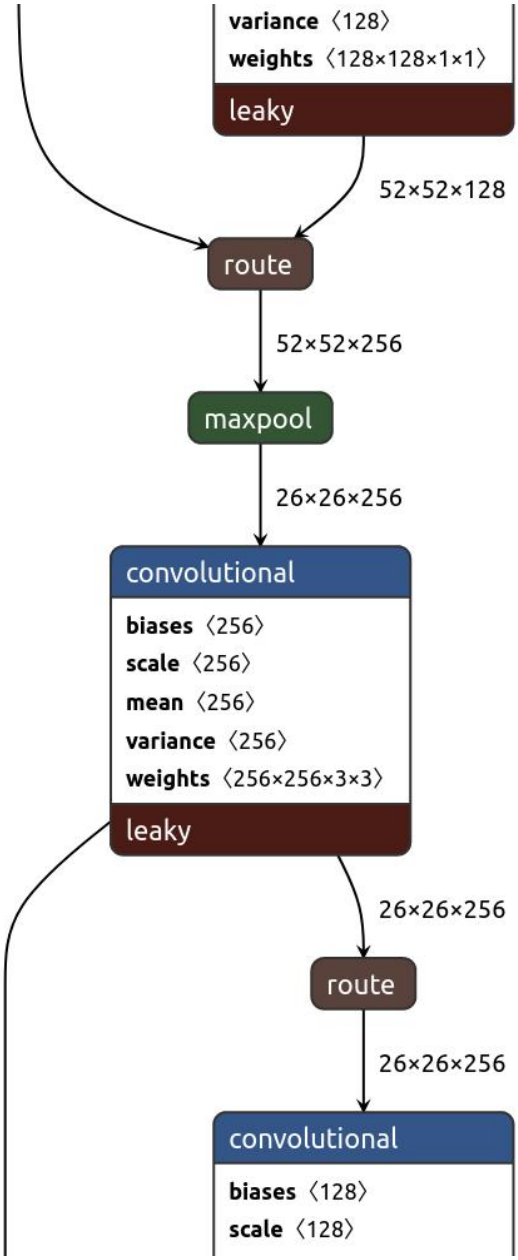
In this annex we show a representation of the network structure of YOLOv4-tiny which was obtained from [47].

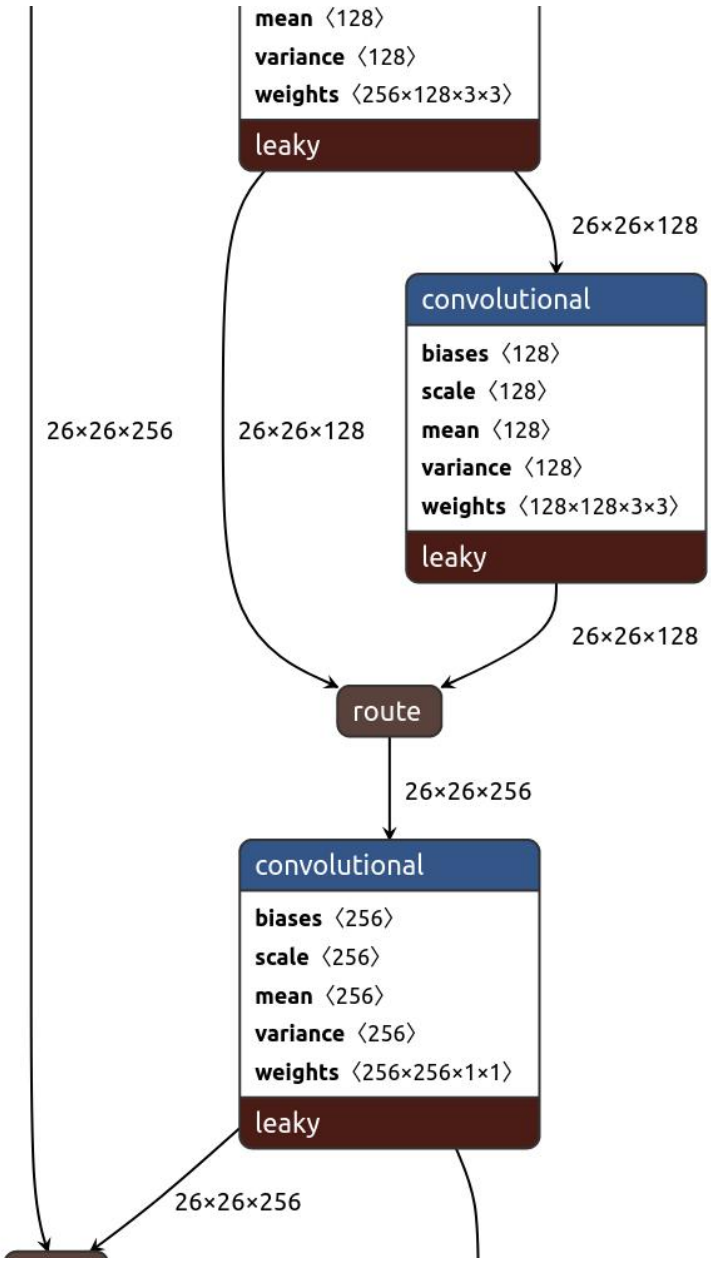


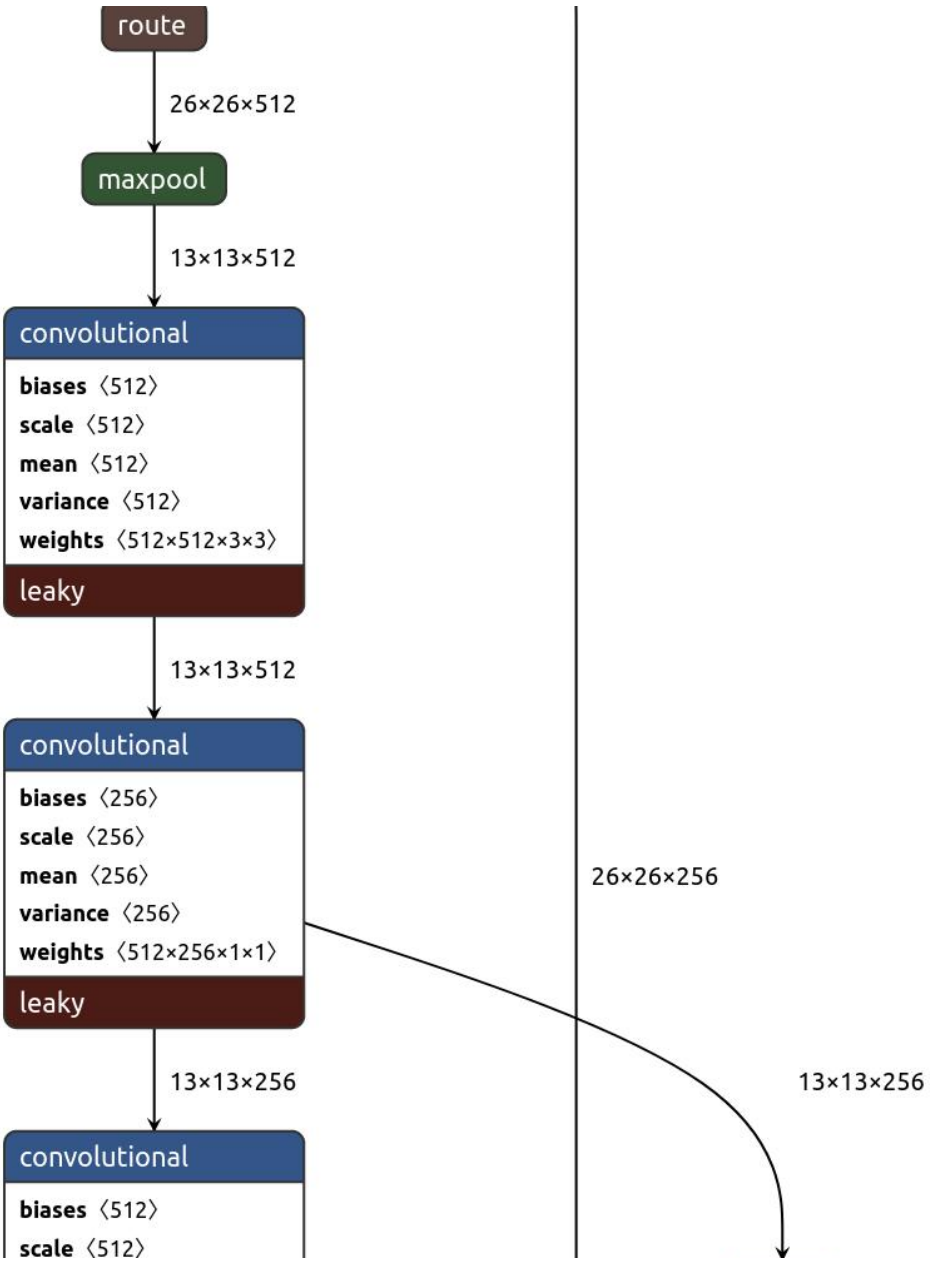

weights ⟨64×64×3×3⟩
leaky
104×104×64
route
104×104×64
convolutional
biases ⟨32⟩
scale ⟨32⟩
mean ⟨32⟩
variance ⟨32⟩
weights ⟨64×32×3×3⟩
leaky
104×104×32
convolutional
biases ⟨32⟩
scale ⟨32⟩
mean ⟨32⟩
variance ⟨32⟩
weights ⟨32×32×3×3⟩
leaky
104×104×64
104×104×32
104×104×32
route

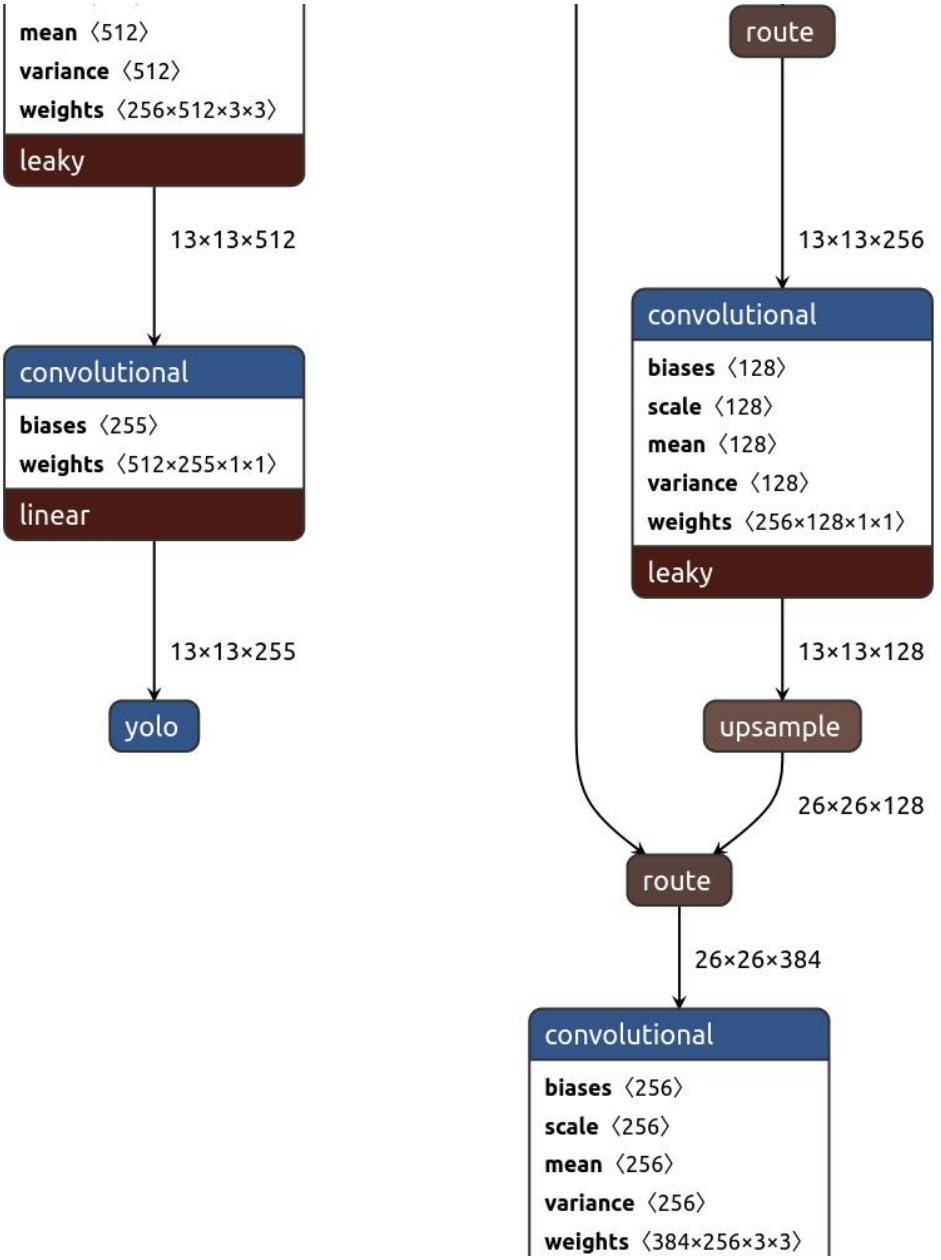

104×104×64
convolutional
biases ⟨64⟩
scale ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
weights ⟨64×64×1×1⟩
leaky
104×104×64
route
104×104×128
maxpool
52×52×128
convolutional
biases ⟨128⟩
scale ⟨128⟩
mean ⟨128⟩
variance ⟨128⟩
weights ⟨128×128×3×3⟩
leaky
52×52×128
route

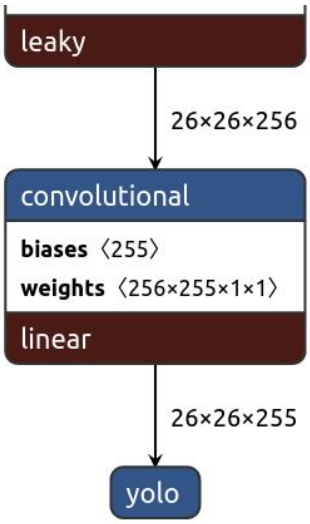

52×52×128
convolutional
biases ⟨64⟩
scale ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
weights ⟨128×64×3×3⟩
leaky
52×52×64
convolutional
biases ⟨64⟩
scale ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
weights ⟨64×64×3×3⟩
leaky
52×52×128
52×52×64
52×52×64
route
52×52×128
convolutional
biases ⟨128⟩
scale ⟨128⟩
mean ⟨128⟩


variance ⟨128⟩
weights ⟨128×128×1×1⟩
leaky
52×52×128
route
52×52×256
maxpool
26×26×256
convolutional
biases ⟨256⟩
scale ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩
weights ⟨256×256×3×3⟩
leaky
26×26×256
route
26×26×256
convolutional
biases ⟨128⟩
scale ⟨128⟩


mean 〈128〉
variance 〈128〉
weights 〈256×128×3×3〉
leaky
26×26×128
convolutional
biases 〈128〉
scale 〈128〉
mean 〈128〉
variance 〈128〉
weights 〈128×128×3×3〉
leaky
26×26×256
26×26×128
26×26×128
route
26×26×256
convolutional
biases 〈256〉
scale 〈256〉
mean 〈256〉
variance 〈256〉
weights 〈256×256×1×1〉
leaky
26×26×256


route
26×26×512
maxpool
13×13×512
convolutional
biases ⟨512⟩
scale ⟨512⟩
mean ⟨512⟩
variance ⟨512⟩
weights ⟨512×512×3×3⟩
leaky
13×13×512
convolutional
biases ⟨256⟩
scale ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩
weights ⟨512×256×1×1⟩
leaky
26×26×256
13×13×256
13×13×256
convolutional
biases ⟨512⟩
scale ⟨512⟩


mean 〈512〉
variance 〈512〉
weights 〈256×512×3×3〉
leaky
13×13×512
convolutional
biases 〈255〉
weights 〈512×255×1×1〉
linear
13×13×255
yolo
route
13×13×256
convolutional
biases 〈128〉
scale 〈128〉
mean 〈128〉
variance 〈128〉
weights 〈256×128×1×1〉
leaky
13×13×128
upsample
26×26×128
route
26×26×384
convolutional
biases 〈256〉
scale 〈256〉
mean 〈256〉
variance 〈256〉
weights 〈384×256×3×3〉


leaky
26×26×256
convolutional
biases ⟨255⟩
weights ⟨256×255×1×1⟩
linear
26×26×255
yolo