University of Padova

Department of mathematics "Tullio Levi-Civita"

Master's Degree in Computer Science

Rocks detection from satellite images

Master's degree thesis

Thesis supervisor

Prof. Lamberto Ballan

Thesis co-supervisor

Dr. Alessio Aboudan

Student

Giovanni Righi

Academic Year 2021-2022

Giovanni Righi: *Rocks detection from satellite images,* Master's degree thesis, © Febbraio 2022.

Premise

This thesis contains the work done in the last six months of my computer science master's degree studies at the University of Padova.
I worked on my thesis project at the "Giuseppe Colombo" University Center for Space Studies and Activities where i worked with Dr. Alessio Aboudan, a postdoctoral researcher at the University of Padova, while Prof. Lamberto Ballan was my thesis supervisor.
In this period I studied Object Detection, a computer vision technique that allows to identify and locate objects in images and videos. In particular I focused on the YOLO (You Only Look Once) algorithm and I used it to perform object detection on satellite images of Enceladus, the sixth-largest moon of Saturn, in order to detect ice rocks.

Abstract

The modern digital computer has been fundamental to the space exploration program. Before computers, exploring space was quite difficult. Almost everything in space exploration is done with computers. Computers have also led to many major breakthroughs in space research. From designing spaceships to space photography, almost everything is done with computers.
There are thousand of human devices such as telescopes, satellites, landers and rovers that are gathering data from Earth, space and even other planets. Each of these device produces a large amount of data.
The main problem is that the data need to be processed and although computers help speed up this process, there is still a need for a human to process that data, this requires a lot of time to be done so only a small part of the data is actually used and processed.
In recent years with the advent of Artificial Intelligence, in particular Machine Learning, scientists have begun to use this technology to improve space exploration.
AI has proven its great potential and is a game-changer in space exploration such as charting unmarked galaxies, stars, black holes, and studying cosmic events, as well as communication, autonomous starcraft navigation, monitoring and system control. The most recent use case of AI is found in the endeavors to create AI-powered, empathetic robotic assistants to help astronomers in their long space travel by understanding and predicting the crew's needs and comprehending astronauts' emotions.
By using AI we can use the small amount of data processed by scientists to train algorithms which can process enormous amount of data faster than a person.
One of the fields of Machine Learning used for space exploration is Computer Vision which allows computers and systems to derive meaningful information from digital images, videos, and other visual inputs - and take action or report on that information. If AI allows computers to think, computer vision allows them to see, observe and understand.
The exact Computer Vision technique that I studied is called Object Detection which allows to identify and locate objects in images and videos.
The aim is to find an Object Detection Algorithm able to learn to detect complex objects such as rocks from satellite images, in our case the images were taken from a satellite on Enceladus, the sixth-largest moon of Saturn.
But, why is it useful to detect the position of rocks in planets or moons?
For a Lander, knowing the position of rocks helps indicate the best place where to land, while for a rover it can indicate the best route to proceed along.
To a geologist, rocks provide clues for the paleohistory of the planet. Depending on the type and distribution of rocks (e.g. igneous, sedimentary or metamorphic) found, a scientist can deduce what the area was like the time the rocks were being formed and deposited.

Contents

List of Figures

Chapter 1

Introduction

1.1 Planetary Image Processing

In recent years with the advent of space exploration, numerous satellites have been launched to explore and study the planets and moons that make up the solar system. These satellites during their mission have collected an enormous amount of data, such as atmospheric composition, temperatures, humidity, images and videos.
Let's focus on the aim of this thesis, images taken from satellites, first of all, how do satellites take pictures? Satellites take pictures in different ways depending on the analysis that needs to be done.
Satellites can be active or passive. Active satellites use remotes sensors for detecting reflected responses from objects that are irradiated with artificially generated energy sources. For example active sensors such as radars emit radio waves, lasers sensors emit light waves and sonar sensors emit sound waves. In all these cases is the sensor that emits the signal and then calculates the time it takes for the signal to "bounce" back from some remote object. Knowing the speed of the emitted signal, the time delay from the original emission to the return can be used to calculate the distance to the feature.
Passive satellites, alternatively, make use of sensors that detect the reflected or emitted electromagnetic radiation from natural sources. This natural source is typically the energy from the sun, but other sources can be imaged as well, such as magnetism and geothermal activity. Using an example we've all experienced, taking a picture with a flash-enabled camera would be active remote sensing while using a camera without a flash would be passive remote sensing. There are five types of resolution when discussing satellite imagery in remote sensing: spatial, spectral, temporal, radiometric and geometric. These resolutions are defined as:

∗ **spatial resolution** is defined as the pixel size of an image representing the size of the surface area (i.e. m2) being measured on the ground, determined by the sensors' instantaneous field of view (IFOV);

∗ **spectral resolution** is defined by the wavelength interval size (discrete segment of the Electromagnetic Spectrum) and number of intervals that the sensor is measuring;

∗ **temporal resolution** is defined by the amount of time (e.g. days) that passes between imagery collection periods for a given surface location;

∗ **Radiometric resolution** is defined as the ability of an imaging system to record many levels of brightness (contrast for example) and to the effective bit-depth of the sensor (number of grayscale levels) and is typically expressed as 8-bit (0–255), 11-bit (0–2047), 12-bit (0–4095) or 16-bit (0–65,535).

∗ **Geometric resolution** refers to the satellite sensor's ability to effectively image a portion of the Earth's surface in a single pixel and is typically expressed in terms of Ground sample distance, or GSD. GSD is a term containing the overall optical and systemic noise sources and is useful for comparing how well one sensor can "see" an object on the ground within a single pixel.

Remote sensing images contain a large amount of information and if the image quality is not good or if the image analysis doesn't use an optimal feature set, then the impact of remote sensing application for which the technique is used may not be fully utilized. The visual interpretation of remote sensing images utilizes different elements of interpretation such as shape, hue, tone, texture and so forth. Manually interpretation is limited to analyzing only a single image at a time due to the difficulty in multiple image interpretation. Manual interpretation is subjective and time for visual classification depends mainly on the image quality. This has paved way for automatic processing over visual analysis and is useful for simultaneous analysis of many spectral bands and can process large data sets much faster than humans.
While a vast amount of digital satellite and aerial imagery is available, the real challenge is in the analysis of raw images, extraction of useful informations with high accuracy and applying it to real-world decision-making and applications. Even though there are various image processing techniques available, conventional methods need to be critically reviewed and image understanding with respect to features of interest is important. With the advances in artificial intelligence and machine learning, there is an improvement in the extraction of detailed features from high-resolution aerial imagery. The fin different layers are utilized for developing various image processing techniques.

1.1.1 Image Processing in Remote Sensing

A lot of algorithms and methods are available for satellite image processing.
Here's a brief description of the main ones:

Image Enhancement

The satellite images usually have a low brightness level due to the condition in which the pictures are taken. Because of this it's important to enhance while preserving the important details without loss of information.
One important parameter used to s subjectively evaluate an image in terms of its quality is the contrast. From human perception, contrast is what differentiates object-to-object with the background. In other words, it is the color and brightness difference between the background and the object.
Many algorithms have been designed and developed to accomplish contrast enhancement and solve various brightness related problems in image processing. Also, it is one of the primary steps followed before proceeding with any of the other image processing techniques such as segmentation, object identification and so forth.

Feature Extraction

Feature extraction involves generating features for selection and classification. These are differential features and forwarded towards the classification phase. Feature extraction plays a very critical role in deciding the efficiency of the image classification process. The various features can be categorized as—general and domain-specific ones. General features are those which include color, shape and texture while domain-specific features are those pertaining to specific applications such as conceptual features.

Segmentation

With image segmentation images are divided into different segments in order to locate objects and boundaries.

Image Fusion

Image fusion is used to combine two or more images to form a new image. With the advancement of multi-resolution analysis, it is possible to extract high-frequency components and then inject them into a multi-spectral image.

Change Detection

Change detection is done to understand the changes in a particular area given the satellite data over that area at different timestamps.

Image Compression

The hardware resources available onboard in a satellite are limited and the satellite images are very large, which necessitates the need for satellite image compression. This reduces onboard data storage and the bandwidth required to transmit the image from the satellite to the ground station as there is only a limited time during which the satellite passes over the ground station. Hence, it is an important technique that will reduce the dependence on the onboard resources without compromising the image quality and the information content which is required at the ground station for analysis. Image quality is a matter of concern since the compressed images need to be reconstructed at the ground station without missing relevant information

Image Classification

Efficient image classification methods are necessary for categorizing satellite images. Image classification is a method of pattern recognition in which images/pixels are classified based on some similarity measures.

Image Feature Detection

There has been an increase in satellite images which helps to understand and analyze various applications. Feature recognition is gaining importance with the advancement in deep learning. The ability to differentiate features such as buildings, roads, vegetation and so forth is important in areas such as environmental analysis, urban monitoring and finding the disaster-affected areas and so on. In order to find correspondences among a set of images, where feature correspondences between two or more images are needed, it is necessary to identify a set of salient points in each image.

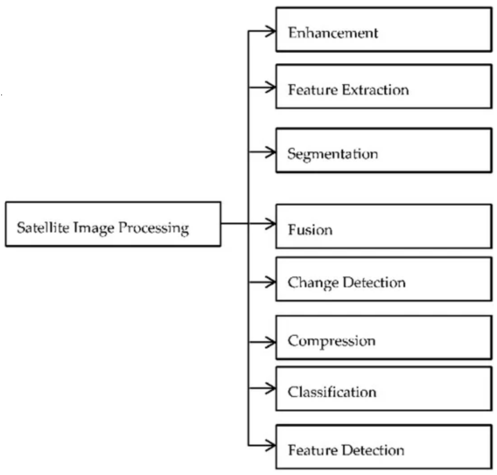


Figure 1.1: Image processing framework.

1.2 Object Detection

Object detection is a computer vision technique that allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labeling them.
Object detection is usually confused with image classification, so before proceeding, we need to understand what is the difference between these two techniques.
Image classification is another computer vision technique that allows machines to interpret and categorize what they "see" in images or videos.
Let's take as an example an image that contains a dog. Image classification allows us to classify the types of things found in the image. Object detection, on the other

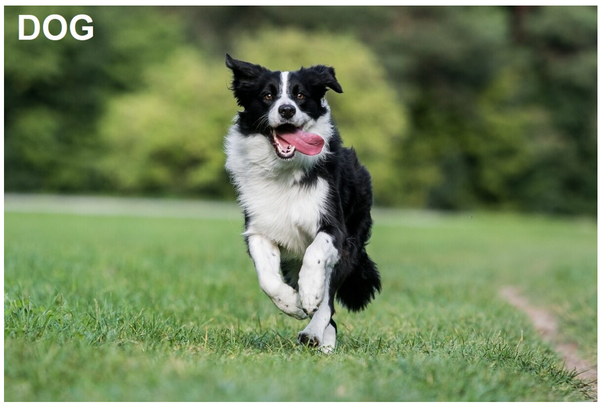


Figure 1.2: Image classification example

hand, draws a box around the dog and labels the box "dog". The model predicts where each object is and what label should be applied. In that way, object detection provides more information about an image than classification.

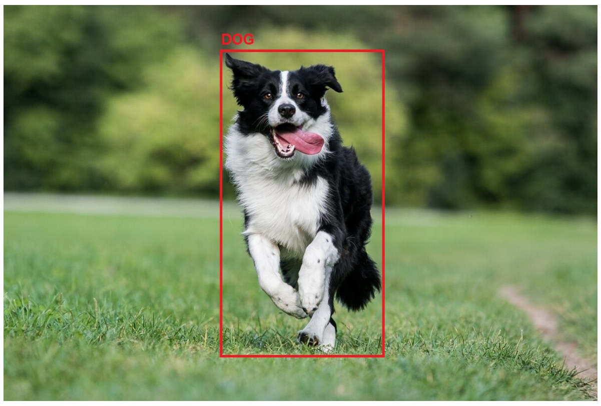


Figure 1.3: Object detection example

1.2.1 Methods of object detection

There are two approaches that can be used for object detection, machine learning-based approaches and deep learning-based approaches.
In more traditional ML-based approaches, computer vision techniques are used to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label.
On the other hand, deep learning-based approaches employ convolutional neural networks (CNNs) to perform end-to-end, unsupervised object detection, in which features don't need to be defined and extracted separately.
On my thesis project, I worked with deep learning methods, which have become state-of-the-art approaches to object detection.

1.2.2 Convolutional neural networks

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in input an image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.
The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

1.2.3 How does object detection work?

In this section, we'll look at several deep learning-based approaches to object detection and assess their advantages and limitations.
Deep learning-based object detection models typically are divided into two parts. An encoder that takes an image as input and runs it through a series of blocks and layers in order to learn to extract statistical features used to locate and label objects. Outputs

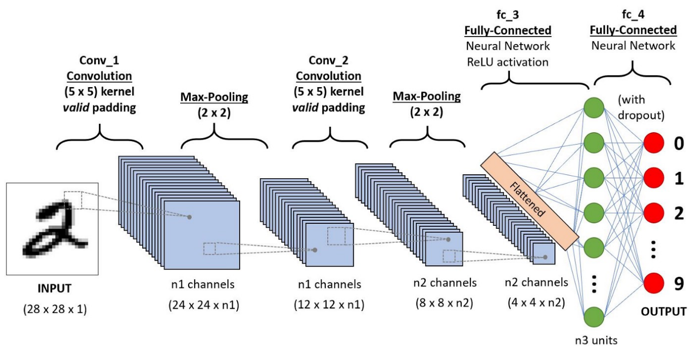


Figure 1.4: A CNN sequence to classify handwritten digits

from the encoder are then passed to a decoder, which predicts bounding boxes and labels for each object.

Bounding Box

In object detection, a bounding box is used to describe the spatial location of an object. The bounding box is rectangular, which is determined by the x and y coordinates of the upper-left corner of the rectangle and the coordinates of the lower-right corner. Another commonly used bounding box representation is the (x,y)-axis coordinates of the bounding box center, and the width and height of the box.
The simplest decoder is a pure regressor. The regressor is connected to the output of the encoder and directly predicts the location and size of each bounding box. The output of the model is the X, Y coordinate pair for the object and its extent in the image. Despite being simple, this type of model is limited. You need to specify the number of boxes ahead of time. If your image has two dogs, but your model was only designed to detect a single object, one will go unlabeled. However, if you know the number of objects you need to predict in each image ahead of time, pure regressor-based models may be a good option.
An extension of the regressor approach is a region proposal network. In this decoder, the model proposes regions of an image where it believes an object might reside. The pixels belonging to these regions are then fed into a classification subnetwork to determine a label (or reject the proposal). It then runs the pixels containing those regions through a classification network. The benefit of this method is a more accurate, flexible model that can propose arbitrary numbers of regions that may contain a bounding box. The added accuracy, though, comes at the cost of computational efficiency.

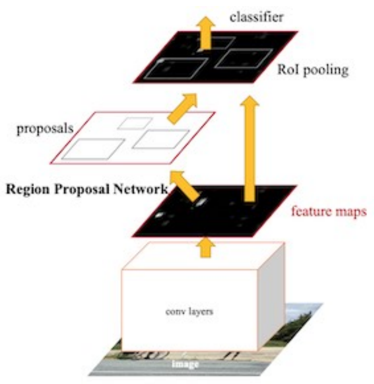


Figure 1.5: Object detection example

Single shot detectors (SSDs) seek a middle ground. Rather than using a subnetwork to propose regions, SSDs rely on a set of predetermined regions. A grid of anchor points is laid over the input image, and at each anchor point, boxes of multiple shapes and sizes serve as regions. For each box at each anchor point, the model outputs a prediction of whether or not an object exists within the region and modifications to the box's location and size to make it fit the object more closely. Because there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. Post-processing must be applied to SSD outputs in order to prune away most of these predictions and pick the best one. The most popular post-processing technique is known as non-maximum suppression.

Non-maximum suppression

Non Maximum Suppression (NMS) is a technique used in numerous computer vision tasks. It is a class of algorithms to select one entity (e.g., bounding boxes) out of many overlapping entities. We can choose the selection criteria to arrive at the desired results. The criteria are most commonly some form of probability number and some form of overlap measure (e.g. Intersection over Union). Object detectors output the

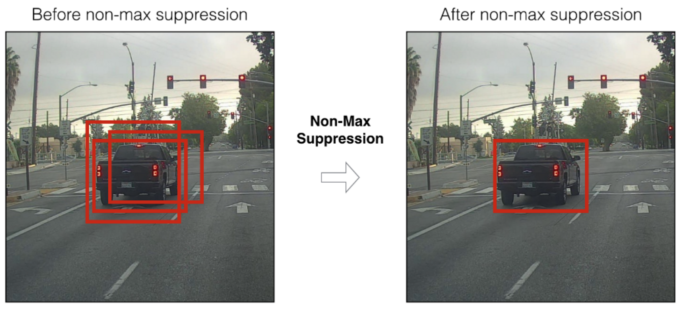


Figure 1.6: Non-maximum suppression example

location and label for each object, but how do we know how well the model is doing? For an object's location, the most commonly-used metric is intersection-over-union (IOU). Given two bounding boxes, we compute the area of the intersection and divide it by the area of the union. This value ranges from 0 (no interaction) to 1 (perfectly overlapping). For labels, a simple "percent correct" can be used.

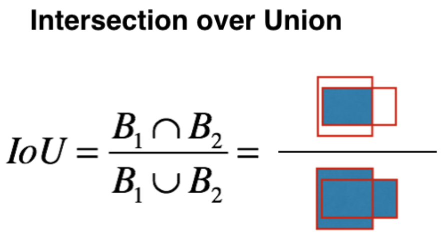


Figure 1.7: Intersection over union

Chapter 2

YOLO

YOLO (You Only Look Once) is a deep learning based object detection algorithm released in 2016 by Joseph Redmon and Ali Farhadi from the University of Washington. Before YOLO there were two major object detection frameworks, DPM(Deformable parts model) and R-CNN both region-based classifiers where, as a first step they would find regions and for the second step, pass those regions to a more powerful classifier to get them classified. This approach involved looking at images thousands of times to perform detection. YOLO started as a project to optimize this approach by building a single neural network that takes a single image and gives back the detections and class in a single pass. That's why the pun "You Only Look Once."

2.1 YOLO v1

The main idea of YOLO is to divide an image in an SxS grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.
Each grid cell predicts B bounding boxes and the confidence scores for those boxes. These scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box that it predicts is.
If no object exists in that cell, the confidence scores should be zero. Otherwise, we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.
Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box.
Each grid cell also predicts C conditional class probabilities, $Pr(Class_i|Object)$. These probabilities are conditioned on the grid cell containing an object. The algorithm only predicts one set of class probabilities per grid cell, regardless of the number of boxes B.
At test time the conditional class probabilities are multiplied with the individual box confidence predictions, which gives the class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

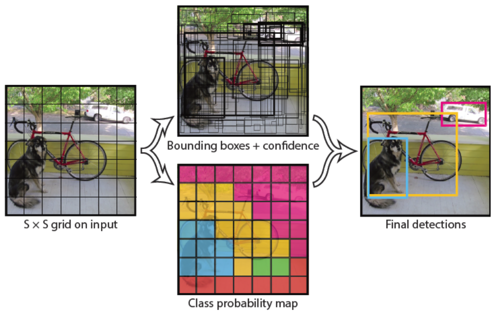


Figure 2.1: Object detection example

2.1.1 Network Design

The model is implemented as a CNN and is evaluated on the PASCAL VOC detection dataset.
The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. The network has 24 convolutional layers followed by 2 fully connected layers. It also simply uses 1 × 1 reduction layers followed by 3 × 3 convolutional layers. The final output of our network is the 7 × 7 × 30 tensor of predictions.

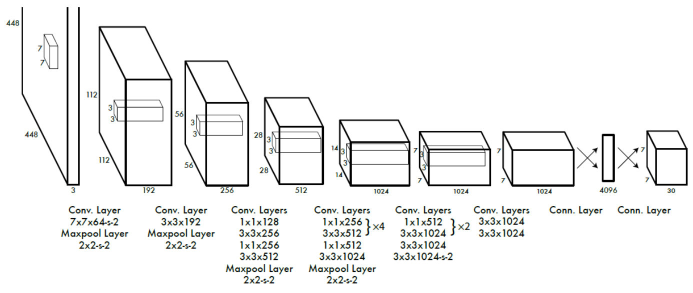


Figure 2.2: Yolo v1 network design

2.2 YOLO v2

An error analysis of the first version of YOLO compared to Fast R-CNN pointed out that YOLO made a significant number of localization errors. Furthermore, YOLO had relatively low recall compared to region proposal-based methods.
To overcome these shortcomings and improve the performance of the algorithm, a series of improvements have been made so in 2017 a new version of Yolo was released.

Batch normalization

Deep neural networks with a lot of layers are challenging to be trained because they can be sensitive to the configuration of the algorithm and the initial random weights. Batch Normalization is a technique used to standardize the inputs to a layer for each mini-batch when training very deep neural networks. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. With batch normalization added on all the convolution layers in YOLO, the mean average precision (mAP) got improved by more than 2%. Also, it helped to regularize the model allowing the removal of dropout from the model without overfitting.

Anchor boxes

Anchor boxes are a priori defined bounding boxes that are placed throughout the image with different sizes and ratios that are going to be used for reference when first predicting object locations.
By using anchor boxes on YOLO two issues were encountered.
The first issue was about the box dimensions which were initially handpicked, this led to having not very good detections of the objects. In order to automatically find good anchor boxes and improve the detection, the K-means algorithm was used on the training set bounding boxes.
The second issue was model instability, especially during the first iterations. Most of the instability came from predicting the (x, y) locations for the box.
In order to overcome this issue, instead of predicting offsets, the new version of YOLO predicts location coordinates relative to the location of the grid cell and uses logistic activation to constrain the network's predictions to fall between 0 and 1.
Predicting offsets instead of coordinates simplifies the problem and makes easier the learning for the network. The network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box, t_x, t_y, t_w, t_h and t_o. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:
$b_x = \sigma(t_x) + c_x$
$b_y = \sigma(t_y) + c_y$
$b_w = p_w e^{t_w}$
$b_h = p_h e^{t_h}$
$Pr(object) * IOU(b, object) = \sigma(t_o)$

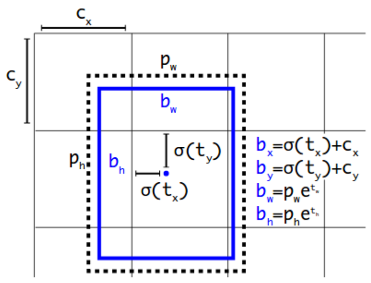
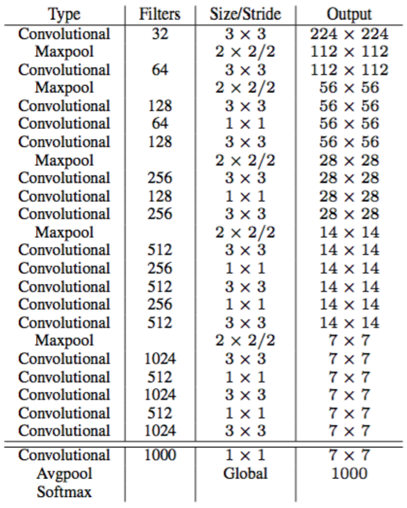


Figure 2.3: Bounding boxes with dimension priors and location prediction

Since the location prediction is constrained, the parametrization is easier to learn, making the network more stable. Using dimension clusters along with directly predicting the bounding box center location improves YOLO by almost 5% over the version with anchor boxes.

Darknet-19

To improve the speed of detection, YOLO v2 comes with a new classification network called Darknet-19 which has 19 convolutional layers and 5 maxpooling layers.
This new part of the network replaces the custom network based on the Googlenet



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 2.4: Darknet19

architecture of Yolo v1 which was faster than the standard classification network (ex. VGG-16) but its accuracy was slightly worse. So with darknet19 YOLO improved not only in speed of detection but also in accuracy, in fact, Darknet-19 only requires 5.58 billion operations to process an image yet achieves 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet.

2.3 YOLO v3

Yolo v3 was released in 2018. In this version minor changes has been done in order to improve accuracy and efficiency on the network, in particular prediction at different scales and a new version of darknet were implemented.

Prediction at different scales

In YOLO v3 the prediction of the boxes is done using 3 different scales, in this way the largest scale is used to detect smaller objects while the smaller one identifies larger objects.
In order to implement the prediction at different scale, several convolutional layers were added to the base feature extractor, the last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions.
Next, the feature map from 2 layers previous is taken and upsample by 2x, it also takes a feature map from earlier in the network and merge it with the upsampled features using concatenation. In this way, the network gets more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. Then few more convolutional layers are added to process this combined

feature map, and eventually predict a similar tensor, although now twice the size of the previous one. This design is repeated one more time to predict boxes for the final scale so the predictions for the 3rd scale benefit from all the prior computation as well as the fine-grained features from early on in the network.

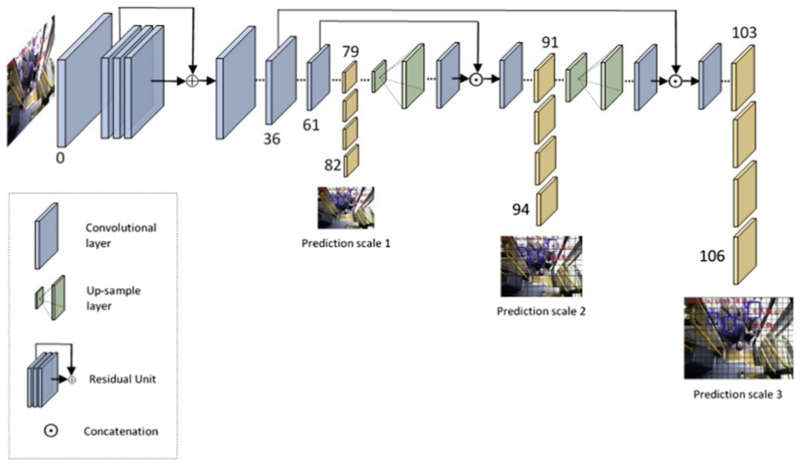
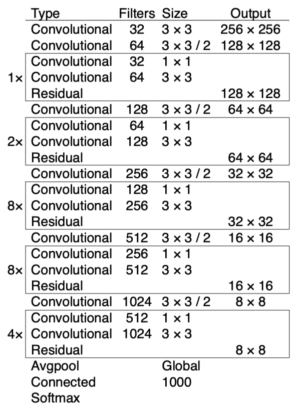


Figure 2.5: Prediction at different scales

Darknet-53

A new version of the darknet network is used for performing feature extraction. This new network uses successive 3 × 3 and 1 × 1convolutional layers but now has some shortcut connections as well and is significantly larger.
It has 53 convolutional layers so it's called Darknet-53. This new network is much more powerful than Darknet19 but still more efficient than ResNet-101 or ResNet-152.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.6: Darknet53

YOLO v3 was the last official release of this algorithm in fact in February 2020, Joseph Redmon, the creator of YOLO announced that he has stopped his research in computer vision! He additionally stated that it was due to several concerns regarding the potential negative impact of his work. However other developers took the work of Redmon and keep working on it with new unofficial versions.

2.4 YOLO v4

In Spril 2020 the 4th generation of YOLO was released by Alexey Bochkovskiy, he continued the work of Redmon in the fork of the main repository. The YOLO v4 has been considered the fastest and most accurate real-time model for object detection. Here's a description of the changes made in YOLO v4.

Bag of Freebies

YOLOv4 employs a "Bag of Freebies" so termed because they improve the performances of the network without adding inference time in production. Most of the Bag of Freebies have to do with data augmentation.
Image augmentation creates new training examples out of existing training data. It's impossible to truly capture an image for every real-world scenario our model may be tasked to see in inference. Thus, adjusting existing training data to generalize to other situations allows the model to learn from a wider array of situations.
The new contribution is mosaic data augmentation which tiles four images together, teaching the model to find smaller objects and pay less attention to surrounding scenes that are not immediately around the object.
Another unique contribution the authors make in data augmentation is Self-Adversarial Training (SAT). SAT aims to find the portion of the image that the network most relies on during training, then it edits the image to obscure this reliance, forcing the network to generalize to new features that can help it with detection.

Bag of Specials

Bag of special methods are the set of methods which increase inference cost by a small amount but can significantly improve the accuracy of object detection. The authors experiment with various activation functions. Activation functions transform features as they flow through the network. With traditional activation functions like ReLU, it can be difficult to get the network to push feature creations towards their optimal point. So the research has been done to produce functions that marginally improve this process. Mish is an activation function designed to push signals to the left and right.
Mish is bounded below and unbounded above with a range of $[\approx -0.31, \infty[$. Due to the preservation of a small amount of negative information, Mish eliminated by design the preconditions necessary for the Dying ReLU phenomenon. A large negative bias can cause saturation of the ReLu function and causes the weights not to be updated during the backpropagation phase making the neurons inoperative for prediction.
Mish properties help in better expressivity and information flow. Being unbounded above, Mish avoids saturation, which generally causes training to slow down due to near-zero gradients drastically. Being bounded below is also advantageous since it results in strong regularization effects.

CSPDarknet53

The Cross Stage Partial architecture is derived from the DenseNet architecture which uses the previous input and concatenates it with the current input before moving into the dense layer.
Each stage layer of a DenseNet contains a dense block and a transition layer, and each dense block is composed of k dense layers. The output of the ith dense layer will be concatenated with the input of the ith dense layer, and the concatenated outcome will become the input of the (i + 1)th dense layer.
The CSP is based on the same principle except that instead of concatenating the ith output with the ith input, it divided the input ith into two parts x0' and x0", one part will pass through the dense layer x0", the second part x0' will be concatenated at the end with the result at the output of the dense layer of x0".
This will result in different dense layers repeatedly learning copied gradient information.

2.5 YOLO v5

On June 25th, the first version of YOLO v5 was released by Ultralytics. This new version of YOLO doesn't come with a paper but only a Github repo so there are not many informations about his structure and the changes done. YOLOv5 is implemented using PyTorch which training procedures permitted good performance improvement, while the model architecture remains close to YOLOv4. Another main thing about YOLO v5 is that there are 5 different models which can be trained.

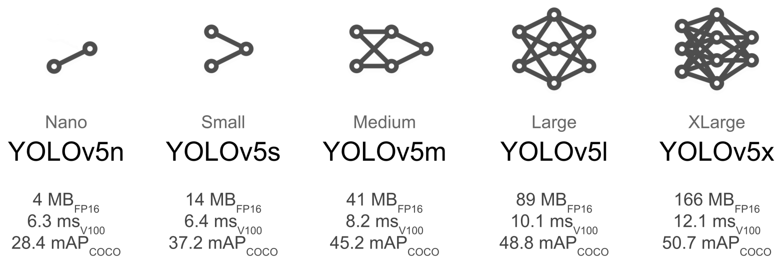


Figure 2.7: YOLO v5 model comparison

Chapter 3

Dataset

The dataset used for the training and testing of the algorithm comes from the Cassini ISS-NAC imagery dataset, the images used are the one of the surface of Enceladus. Enceladus is a 500 km-size icy moon of Saturn, orbiting at a distance of 3.94 Saturn radii from the planet. Its surface is dominated by pure water ice, heavily cratered in the northern latitudes as well as in its trailing and leading mid-latitude hemispheres. In particular the image used was given to us by Maurizio Pajola a Research Scientist at INAF - Istituto Nazionale di Astrofisica. He used the dataset on a paper *Blocks Size Frequency Distribution in the Enceladus Tiger Stripes Area: Implications on Their Formative Processes* where he and his team studied the images of Enceladus in order to identify the icy blocks located in the Enceladus SPT.
Using ArcGIS, the team identified the blocks that are located on, or close to, the "tiger stripes", 100 km-long equally-spaced tension fractures, thanks to the presence of a nearby elongated shadow. As commonly done by similar analyses performed on other bodies of the Solar System, once such features have been manually identified, the team measured their position on the surface, assumed their shapes to be circumcircles (the maximum resolution of the images does not allow to outline them as polygons) and then calculated their diameter. The dataset given by Dr. Maurizio Pajola was

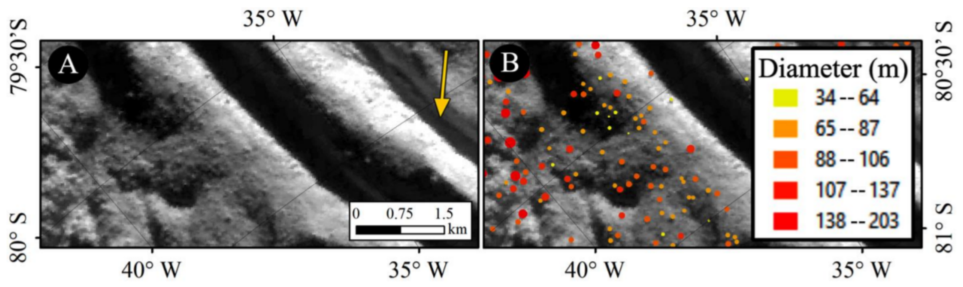


Figure 3.1: Methodology used to identify the blocks located on Enceladus tiger stripes. (A) Subframe of the ISS-NAC N1597182500 image. The yellow arrow shows the direction of the solar irradiation. (B) The identified blocks are grouped in size (m)

composed of different files but only two of them had the information needed for the training.

* a .tif (Tagged Image File) file containing the image of the surface of Enceladus;
* a .shape file containing a list of circle shapes whose coordinates correspond to the coordinates of the icy rocks on the image, the shapes were manually drawn by Dr. Maurizio Pajola.

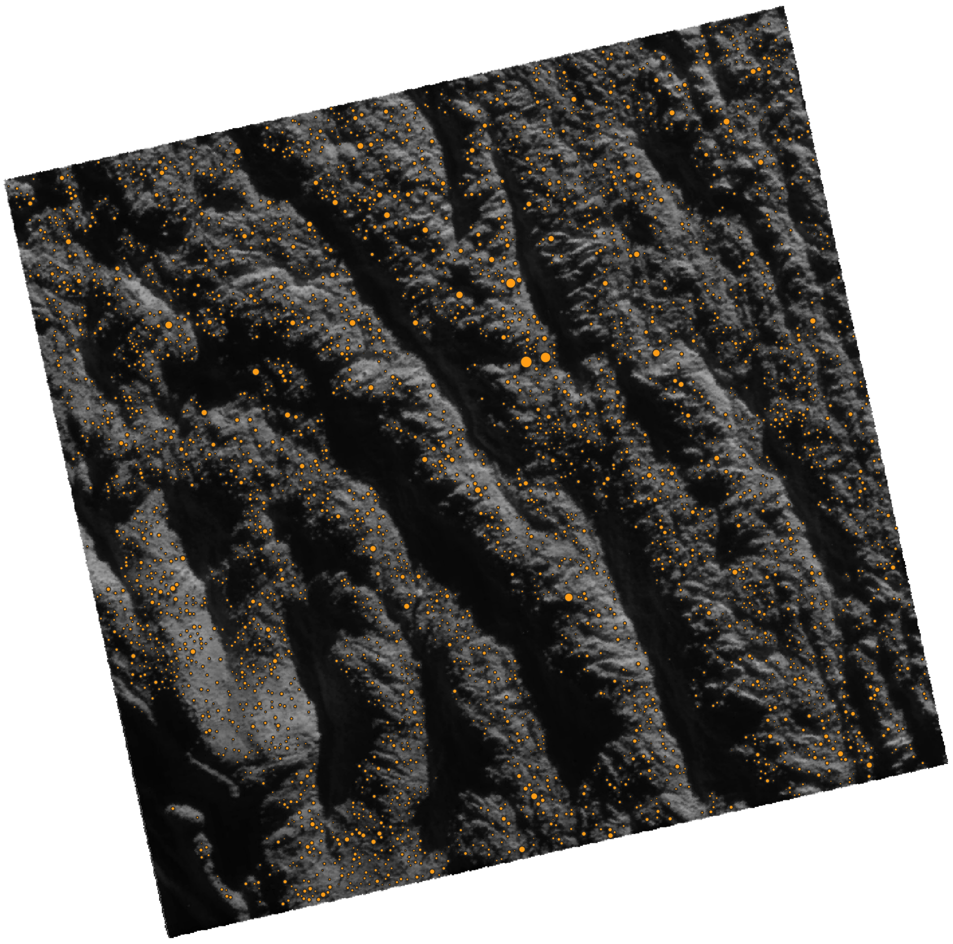


Figure 3.2: Cassini ISS-NAC image of Enceladus Surface with manually identified icy rocks

Chapter 4

Workflow and Tools

Let's recap the aim of this thesis. We wanted to find an object detection algorithm that is able to detect icy rocks from satellite images of the surface of Enceladus.

4.1 First attempts with YOLO v3

I made the first attempts with YOLO v3, but why I decide to use this algorithm? The decision of using YOLO v3 comes first of all because Dr. Alessio Aboudan had already worked with this algorithm so we already had a base where to begin, also YOLO is considered one of the best algorithms for object detection.

4.1.1 Preprocessing

To begin I had to prepare the dataset for the training of YOLO v3. To do this, I found a tutorial for custom training of YOLO v3, I studied it in order to understand how it worked and how the dataset needed to be prepared for performing the training.

Coordinates conversion and bounding box

As all the object detection algorithms YOLO works with bounding boxes to detect the objects, in particular the version of YOLO that we used works with the coordinates in pixel, each bounding is defined as follow: x_min, y_min, x_max, y_max, class_id. The first four parameters are relative to the coordinates of the bounding box while the fifth parameter identifies the class of the object by its id. For the classes, there is a file containing all the possible class one for each line the id of a class correspond to the number of the line in the file.
An example of the classes file is the following:

```
0   Dog
1   Cat
2   Car
3   Telephone
```

Listing 4.1: Example of class listing file

The first problem that I needed to solve was that the .tif and the .shape files were defined using two different coordinate systems but the bounding box needed to be in pixel so in order to convert every coordinate in the shapefile to pixels and ensure

that the coordinates reflected the position of the rocks on the image I had to convert the coordinates of the shape file into the coordinates of the .tif file and then convert everything into pixels.
But how did I do this?
I used QGIS, an open source software that permits to visualize, organize and manipulate geospatial data.
QGIS is able to automatically is capable of automatically superimposing the shapes of the .shape file onto the .tif file despite having two different coordinate systems.
Also in QGIS is possible to create and run Python scripts to manipulate the data loaded in the application.
So I wrote a python script which take the coordinates of the .shape file converts them to the coordinates system in the .tif file and then convert everything in pixels relative to the image, part of this script uses code from another script written by Dr. Aboudan for another project.
In this way, I had the circle shapes coordinates in pixel perfectly matching the pixel of the icy rocks on the image.
The second thing to do was to convert the circle shapes into bounding boxes, so I extended the python script used for the conversion to convert the circle coordinates into box coordinates. Here's the result of the conversion:

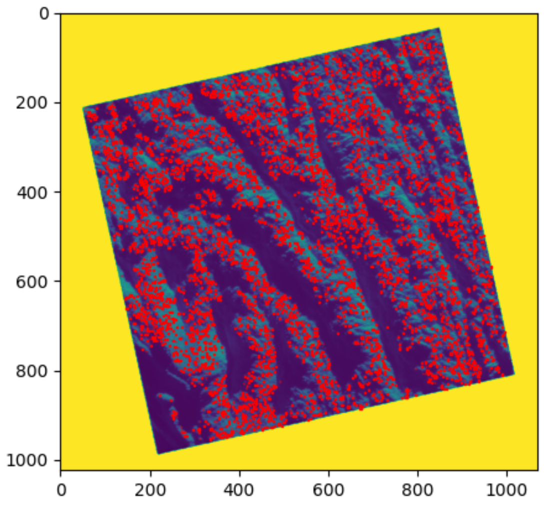


Figure 4.1: .tif file with rectangular bounding boxes

After converting everything I saved all the data on a file in order to use them in the following steps.

4.1.2 Multi images

Generally, when training a neural network, it is common knowledge that too little training data results in a poor approximation. An over-constrained model will underfit the small training dataset, whereas an under-constrained model, in turn, will likely overfit the training data, both resulting in poor performance. Too little test data will

result in an optimistic and high variance estimation of model performance.
So for training our rock detection version of YOLO v3 I needed a lot of images with different types of rocks but I had only one image with a dimension of 1024x1024 pixel and thousand of rocks on it so here's how I proceeded into creating the dataset.
First of all, I had to split the image into at least hundreds of images in order to do this I wrote a script in PyCharm which randomly take two coordinates of the image and using them as the top-left vertex take a sub-image of dimension 64x64 pixels, I choose 64 as dimension because YOLO works better with multiple 32 size images.
Then for each image, I searched the bounding box that were completely into them and created a file with the coordinates of the bounding boxes converted to fit the new coordinates of the rocks in the sub-image.
So after all this work, I had hundreds of images each with a file with the coordinates of the bounding box of the rocks in the image.
Here's an example of three images used for the training:

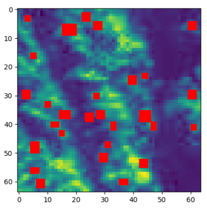


Figure 4.2: tile_1

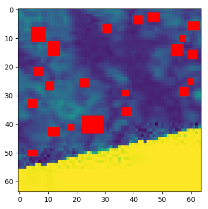


Figure 4.3: tile_2

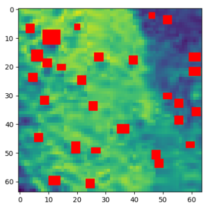


Figure 4.4: tile_3

4.1.3 Training set and Validation set

Another thing that needed to be done when preparing the data to train a neural network is to divide them into a training set and a validation set:

* The Training dataset is the sample of data used to fit the model. The actual dataset that we use to train the model (weights and biases in the case of a Neural Network). The model sees and learns from this data.

* The Validation dataset is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as the skill on the validation dataset is incorporated into the model configuration.

The main thing to which I needed to pay attention when creating a training set and a validation set is that the images in the two sets need to be different. In order to create the two sets, I updated the Python script which creates the sub-images as follows.
First of all, We need to know that each rock in the .tif image has a unique id.
So I prepared the training set by taking 100 random sub-images with their bounding box file keeping track of the id of the rocks present in these images. After this I created the validation set by taking 50 random sub images making sure that they did not contain rocks already present in the training set. So now I had 100 images for the training set and 50 images for the validation set.

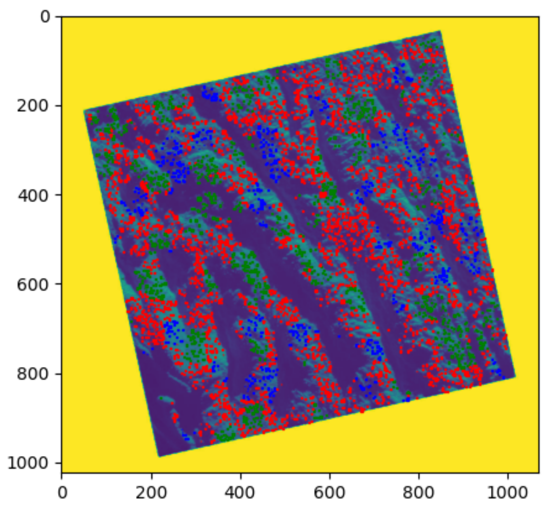


Figure 4.5: full image with rocks divided into training set(red) and validation set(blue)

The last thing to do in order to train our custom version of YOLO was to create an annotation file (.xml) to give to the algorithm here's the structure of the file:

* One row for one image;
* Row format: image_file_path box1 box2 ... boxN;
* Box format: x_min,y_min,x_max,y_max,class_id (no space)..

Here's an example of an annotation file for the training of YOLO v3:

```
path/to/img1.jpg 50,100,150,200,0 30,50,200,120,3
path/to/img2.jpg 120,300,250,600,2
...
```

Listing 4.2: Example of an annotation file

4.1.4 Training and Results

For the training of the network I ran the YOLO v3 training script on Pycharm on my computer which specifications are:

CPU : AMD Ryzen 5 5600X 6-Core Processor 3.70 GHz

RAM : 16GB

GPU : Gainward GTX1060 Phoenix 6GB

After the first runs where I got some runtime error due to some configuration problem I managed to perform the training. The first training I made had 100 epochs, during the training I was able to see the loss and the validation_loss for each epoch were decreasing epoch after epoch so it seemed like the algorithm was actually learning from the data. At the end of these first trainings, I tried to do the detection with the trained network on some spare sub-image that I had but the network wasn't able to detect anything right. In order to figure out what the problem could be I searched on the internet and discussed with Dr. Aboudan and Prof. Ballan, here're some of the things that we thought were affecting the training:

∗ **Not enough training data**: Having only 100 images on the training set and 50 images on the validation set which may have some repetition of the same object may affect the training of the network;

∗ **Color channels**: The network was initially set to work with 3 channels images (RGB) while the images given had just 1 channel;

∗ **Low image definition**: Each image was 64x64 pixel and could contain tens of rocks so each rock was "big" a few pixels, YOLO v3 is not able to detect very small objects.

∗ **Network too big for detecting simple objects as rocks** : The YOLO v3 network is pre-trained on the COCO 80 dataset, this dataset contains 80 object categories of labeled and segmented images such as dogs, cars and persons. These kinds of objects are a bit complex so the network need to be bigger to learn all their features.

For the first problem there wasn't much I could do having just one image which need to be split in order to get a decent amount of training data and the possibility that some object may be used multiple times during the training wasn't very good but this was also a challenging task to take. The only thing that I could do was increasing a bit the number of images on the dataset. The second problem was quite easy to fix, the implementation of YOLO v3 that I used contained a configuration file where some changes could be made to the network, one of these was the number of channels of the images so I decreased it from 3 to 1. For the low image definition problem I used a tool that automatically resizes the images for YOLO. The last problem was about the network YOLO v3 comes with a smaller network called tiny-yolo I tried using this for the training. After having implemented all the changes discussed above I tried again training the network with different settings, the results were better but still the network wasn't able to detect the rocks so I had to think about something else.

4.2 YOLO v5: A new hope

While searching for a solution for the training of YOLO v3 I speculated that a possible problem could be the implementation of the algorithm I was using so I started searching for a better one, in the various sites consulted I kept finding people talking about a new YOLO implementation called YOLO v5. So I decided to take a look at it. As explained in chapter 2, YOLO v5 it's an unofficial release of YOLO made by Ultralytics and released in 2021, the main difference of this version is that it's implemented in PyTorch.
I downloaded the code and following the tutorial on the Github repo of Ultralytics I started prepare everything for the training.
The first thing I had to do was to understand how the annotation files needed to be prepared for the training of the network.

The format of the file was almost the same as YOLO v5 but with some minor differences. Here's the YOLO V5 annotation file format:

∗ One row per object;

∗ Each row is class x_center y_center width height format;

∗ Box coordinates must be in normalized xywh format (from 0 - 1). If your boxes are in pixels, divide x_center and width by image width, and y_center and height by image height;

∗ Class numbers are zero-indexed (start from 0).

Initially, I thought that I needed to change my preprocessing script in order to create the annotation file in the new format but in the tutorial for the custom training of the YOLOv5 network, it suggested using Roboflow.
Roboflow is an AI company that creates software-as-a-service products to enable faster, easier computer vision. It manages your images, annotations and labels, preprocessing, augmentations, and file formats so you can focus on the business logic of your problem.
So I uploaded on Roboflow all the data that I was using for YOLO v3, images, annotation files and the classes list, and immediately the tool managed to merge all them together so I could work on them.
One of the many things that Roboflow offer is the preprocessing on the data, when the images and relative annotations are loaded you can decide how divide the in the training set, validation set, and test set and also resize and reorient the images.
The resizing tool is way better than the one I used for YOLO v3 so I prepared two different datasets one with images of dimension 320x320 and another with dimension 640x640. In this step there is the possibility to apply data augmentation to the dataset but I didn't do it because augmentation is already included in the YOLO v5 algorithm.
Another important this that Roboflow offer is to convert annotation data between the various format used across the various YOLO implementation, so I converted the annotation files from YOLO v3 to YOLO v5 and I was ready for the training.
Because now I had about a thousand of images I decided to use Google Colab to perform the training in order to make it faster.
I trained both datasets on all YOLO v5 networks and the results I got were quite good.

Chapter 5

Results

Let's recap how the training of YOLO v5 was performed.
I had two different datasets both containing 500 images, one dataset had images with dimensions 320x320 pixels while the other one 640x640 pixels.
I trained the first dataset on 4 different YOLO v5 models, small, medium, large and extra-large each of the training took 100 epochs.
While for the second dataset I used only the small and medium model also with 100 epochs. I didn't manage to train on the large and extra-large network because it took too much time and the Google Colab execution time wasn't long enough to perform the whole training.
I also did a long training with 1000 epochs on the dataset with smaller images on the small model, I managed to do this test only in this configuration because despite being the fastest one it still took around 5 hours for the training.

5.1 Metrics

Before explaining the results of the training phases there is a need to understand the metrics used to monitor the progress of the training.
While during the training of the YOLO v3 network there was only the monitoring of the training loss and the validation loss, in YOLO v5 many other metrics were monitored during the training.
But first of all we need to understand how the accuracy of the network is calculated. When working with object classification and object detection the terms True positive and True negative are commonly used to understand the accuracy of a network but what are these terms?
I will use as example the task of this thesis so rocks detection:

∗ **True Positive**: A true positive is an object labeled and located correctly, in our case if in the image there is a rock that has been detected and labeled as rock, that's a true positive.

∗ **False Negative** A false negative is an object which hasn't been located or has been labeled wrong, in our case if a rock isn't detected it is a false negative.

∗ **True Negative** A true negative is a portion of the image which has no object and no object is detected from the algorithm. In our case all the background of the image.

∗ **False Positive** A false positive is a portion of the image where an object is detected and labeled but in reality, there is no object. In our case if in the background a rock is detected that's a false positive.

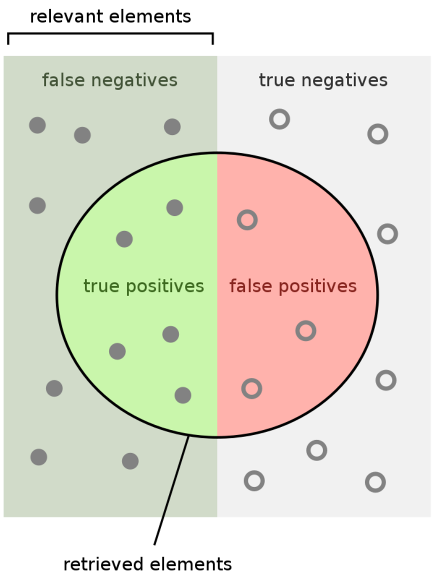


Figure 5.1: Explanation of the True/False Positive/Negative system

But how can we exactly tell define true positive or a False Negative in object detection? Object detection systems make predictions in terms of a bounding box and a class label.
For each bounding box, we measure the overlap between the predicted bounding box and the ground truth bounding box. This is measured by IoU (intersection over union).

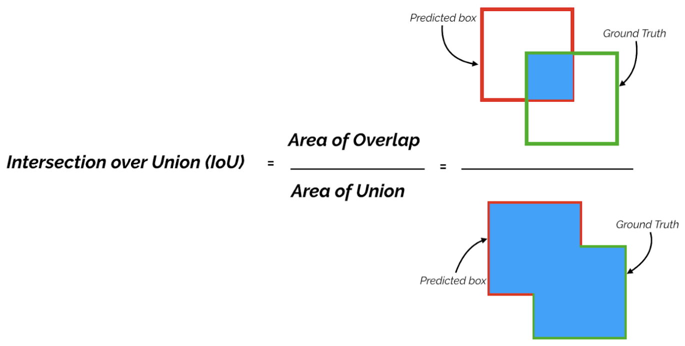


Figure 5.2: Intersection over Union

For object detection tasks, we calculate Precision and Recall using IoU value for a given IoU threshold.
For example, if IoU threshold is 0.5, and the IoU value for a prediction is 0.7, then

we classify the prediction as True Positive (TF). On the other hand, if IoU is 0.3, we classify it as False Positive (FP).

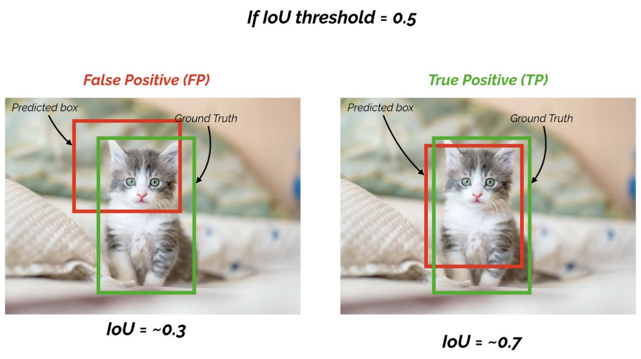


Figure 5.3: Example of IoU threshold

That also means that for a prediction, we may get different binary TRUE or FALSE positives, by changing the IoU threshold.

5.1.1 Precision

Precision is a metric used to measure how accurate the predictions are i.e. the percentage correct predictions. It measures how many of the predictions that your model made were correct.

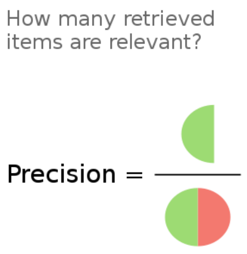


Figure 5.4: precision

5.1.2 Recall

Recall measures how well you find all the positives. For example, we can find 80% of the possible positive cases in our top K predictions.

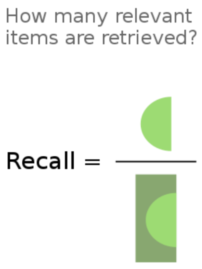


Figure 5.5: recall

5.1.3 Tradeoff between Precision and Recall

Just by looking at the formulas for calculating Precision and Recall, we could suspect that for a given classification model, there lies a trade-off between its precision and recall performance. If we are using a neural network, this trade-off can be adjusted by the model's final layer softmax threshold.
For our precision to be high, we would need to decrease our number of FP, by doing so, it will decrease our recall. Similarly, by decreasing our number of FN would increase our recall and decrease our precision. Very often for information retrieval and object detection cases, we would want our precision to be high (our predicted positives to be TP).

5.1.4 Mean average Precision

The mean average precision or mAP is the main metric used to compare the performance of object detection systems.
Once we have the predicted bounding boxes from all the images of a particular class, we can plot the precision-recall curve for that class. The precision-recall curve shows the tradeoff between precision and recall for different thresholds. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. Average

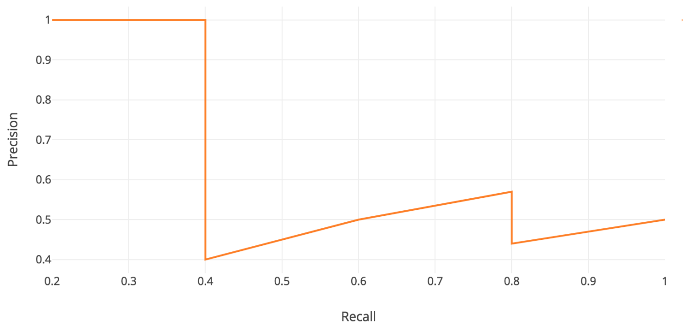


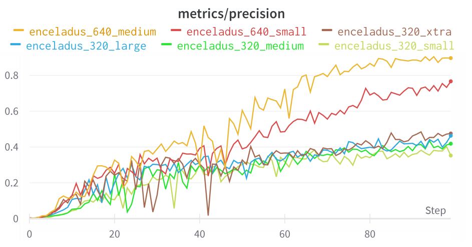
Figure 5.6: Example of precision/recall curve

Precision (AP) is computed by calculating the area under the curve for that particular class. AP for all the classes is averaged to give the MAP of the model. The higher the MAP, the better is the model's performance.

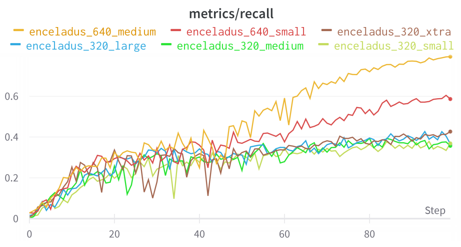
5.2 Results

5.2.1 Metrics results

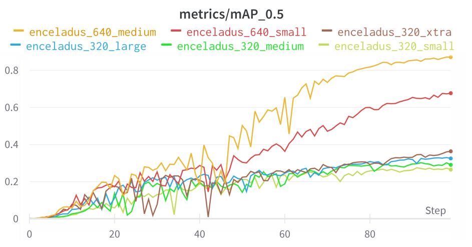
YOLO v5 integrates Weights & Biases (W&B) which is a software for real-time visualization and cloud logging of training runs, so during the execution of the training the data were sent to W&B which automatically creates graphics with them.
Here's the result of the different training I've done:



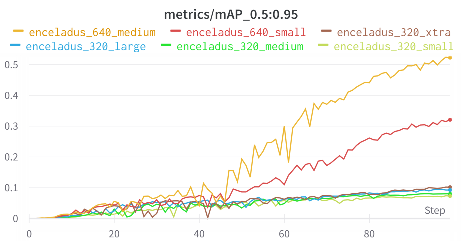
(a) Models Precision



(b) Models Recall



(c) Models mAP_0.5



(d) Models mAP_9.5

Figure 5.7: Comparison between models metrics

The first two graphs show the precision and the recall of the training while the other two graphs show the mAP.
Here's a table that summarizes the results obtained by the different training.

Run	**Precision**	**Recall**	**mAP_0.5**
320_small	0.3524	0.3569	0.2646
320_medium	0.4183	0.3659	0.2903
320_large	0.4630	0.3687	0.3248
320_xlarge	0.4751	0.4268	0.3632
320_small_1000	0.8270	0.6991	0.7669
640_small	0.7667	0.5863	0.6765
640_medium	0.8965	0.7941	0.8708

Table 5.1: Table of the metrics result of the different training run

What we can notice from the graphs is that both precision and recall follow a positive trend during training this can give us a first indication that the training is working, also the mAP follows a positive trend which means that there is a good trade of between precision and recall.
Now let's take a detailed look into the values of the metrics in the different training.
First the dataset with the images of dimension 320x320, this dataset has been trained in 4 different models, the results reached from all the models are almost the same with $Precision \approx 0.4$, $Recall \approx 0.4$ and $mAP \approx 0.3$.
As we can see all the metrics are quite low all below 0.5 this means that the models aren't very good and they may need more epochs or different parameters to perform better.
This is also confirmed by the only training done with 1000 epochs on this dataset with the small model.

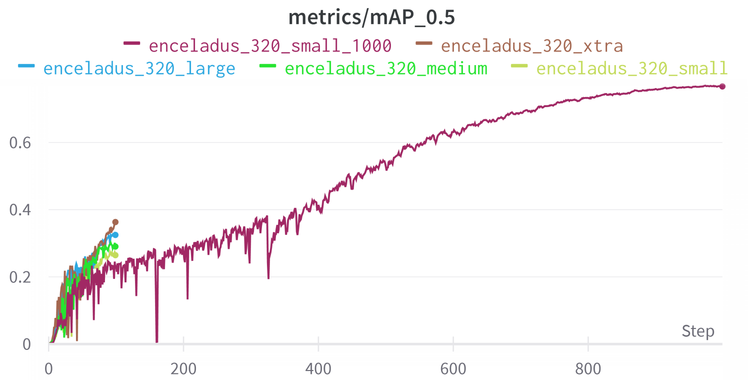


Figure 5.8: Map comparison between 100 epochs and 1000 epochs training

As we can see from the table the $mAP = 0.7669$ which is quite a good result this means that with more epochs the training would have achieved better results.
Unfortunately for time constrain I wasn't able to perform long training on the other models.
The dataset with images of dimension 640x640 performed way better the other dataset with both models.
The training on the small model reached a $Precision = 0.7667$, $Recall = 0.5863$ and $mAP = 0.6765$, while the training on the medium model reached a $Precision = 0.8965$, $Recall = 0.7941$ and $mAP = 0.8708$ which is a very good result and let us think that the model has learned how to detect rocks.
This suggests us something that I already pointed out when explaining the YOLO v3 training that is, having images with a higher resolution leads to better network performances.
One last graph that I want to show about the training phase is the following which shows how much time each training run took.

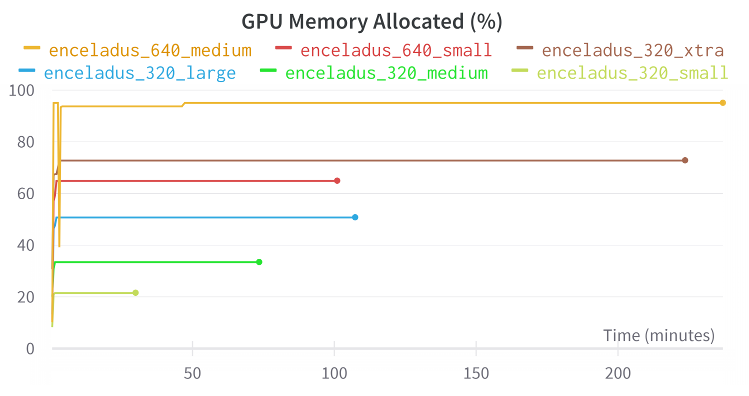


Figure 5.9: Time of GPU allocated

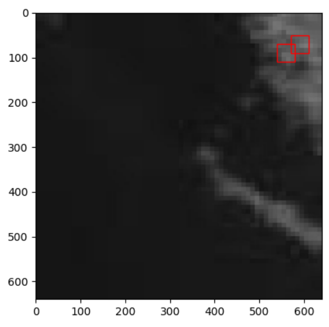
The first thing that we can see from the graph is that training the smaller network required less time than training the bigger one, this is quite obvious since it is well known that the bigger a network is, the longer it takes to train it.
Another interesting thing is that on the same network training the dataset with images of dimension 320x320 require much less time than training the dataset with images of dimension 640x640 this is due to the fact that the network require more time to elaborate the image.

5.2.2 Detection results

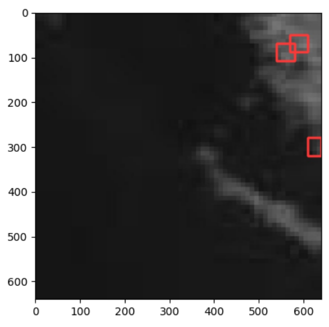
In order to understand if the trained network was actually working and was able to detect rocks I did some detection tests on the data used during the training and also data that the algorithm had never seen before.
All the test are done with the network trained on the dataset with dimension 640x640 using the medium model of YOLO V5 which got the best training results.

Training images

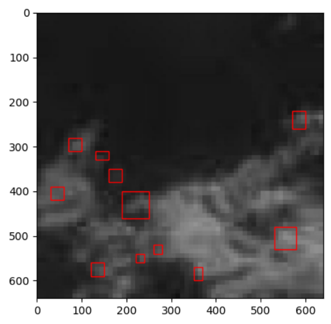
Here's a comparison between some images of the training set with the manually taken bounding box and the same images with the detected bounding box.



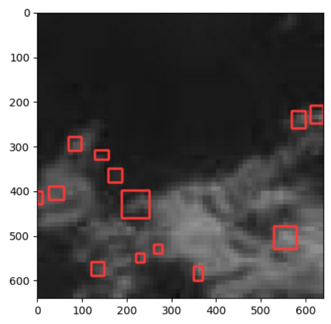
(a) Tile0 original



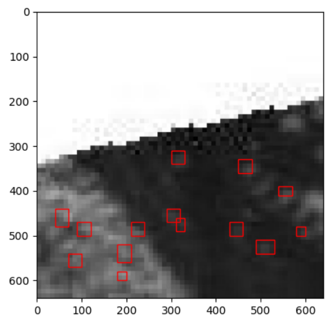
(b) Tile0 detected



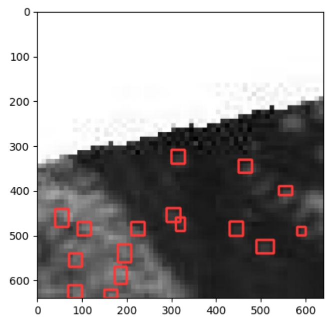
(c) Tile2 original



(d) Tile2 detected



(e) Tile3 original



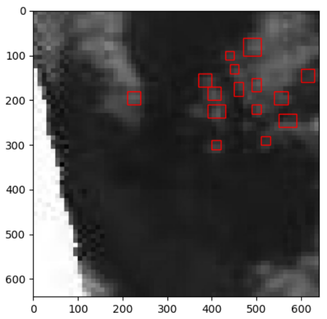
(f) Tile3 detected

Figure 5.10: Comparison between manual detection and model detection on the training set

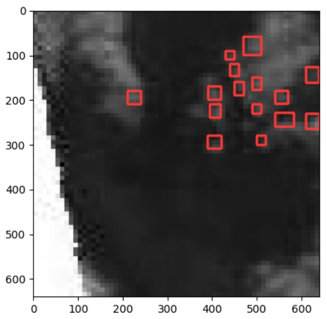
As we can see from the images above the algorithm is actually working and is able to detect the rocks in the correct position, obviously being this the training set this tells us only that the algorithm is able to do detection only in the images he had already seen but we still don't know how it behaves with new images it may be just overfitting the data given.
But there is still something interesting that we can see from this comparison in all the three images with the rocks detected by the algorithm we can see on the border some detected rocks which seem to be cut, these rocks are the rocks not included in the set of rocks of the training tile because they weren't fully inside the image. So the algorithm is able to detect rocks that are not entirely in the image.

Validation images

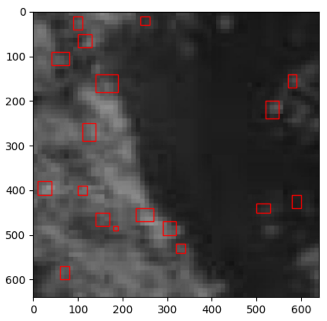
Here's a comparison between some images of the validation set with the manually taken bounding box and the same images with the detected bounding box.



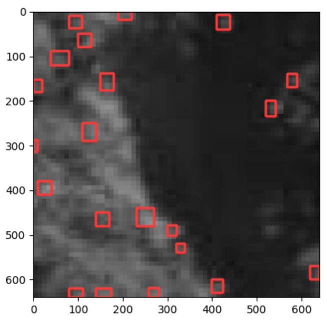
(a) Tile1 original



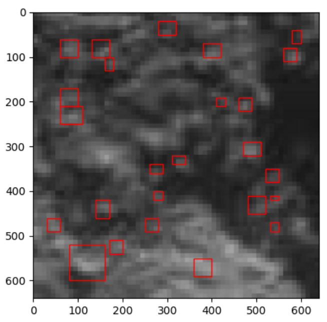
(b) Tile1 detected



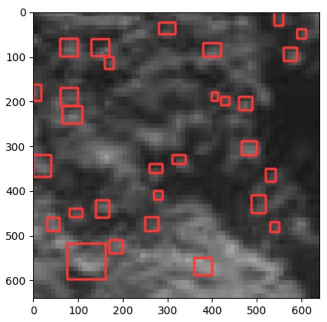
(c) Tile15 original



(d) Tile15 detected



(e) Tile18 original



(f) Tile18 detected

Figure 5.11: Comparison between manual detection and model detection on validation set

Doing the detection on the tiles used in the validation set has given the same result like the one performed on the training set.

Test images

In order to understand if the algorithm was actually working and just not doing overfitting on the training data, I had to find another satellite image of Enceladus and perform detection on it.
Because the image used in the training was published on the Cassini dataset which is accessible by anyone I searched for other images and I found another image taken during the same Enceladus fly-by as the one used during the training.
Before showing the results it must be noted that the image has a different exposure to light than the training image, this can affect the detection as the algorithm is more adapted to the conditions of the images used for training.
Here's how detection is performed on some tiles of that image.

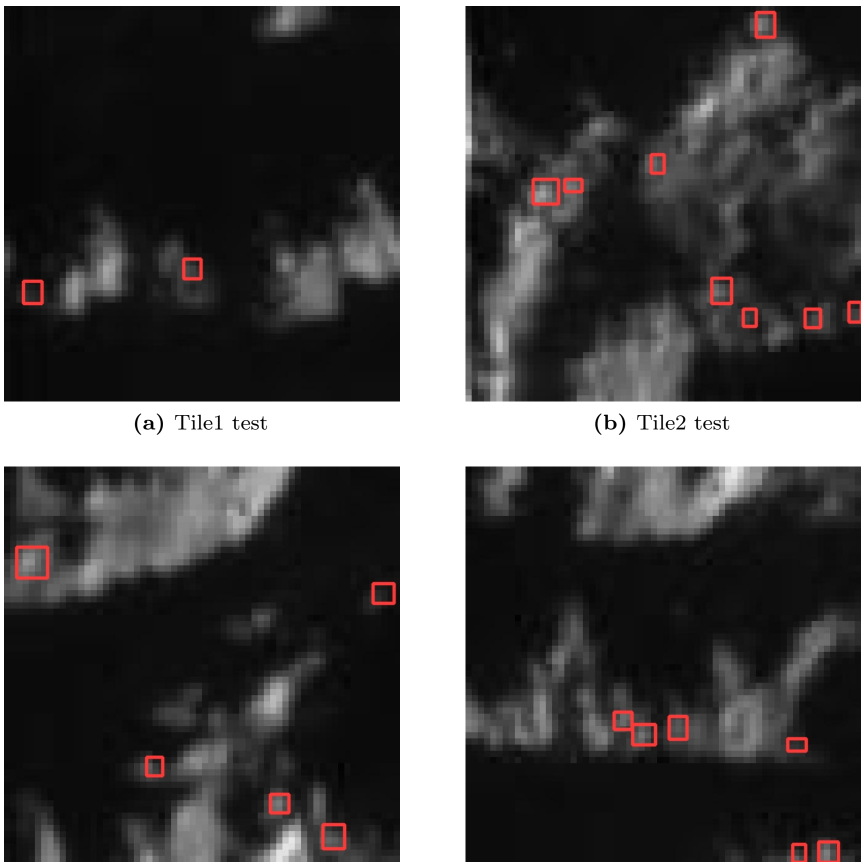

(a) Tile1 test
(b) Tile2 test
(c) Tile3 test
(d) Tile4 test

Figure 5.12: Detection on test images

As we can see the algorithm seems to be able to detect rocks on these images, in order to be sure that the object identified are actually rocks the images must be studied by a geologist who have studied the Enceladus surface.
But the fact that the algorithm is actually giving some output is actually a good result.

One last test I did in order to understand the performance of the trained network was to try detection on the full images used during the training and testing phase instead of using the tiles.
In order to have the same image resolution of the training images I had to increase the image dimension from 1024x1024 pixels to 10240x10240 pixels.
Here's the result of detection on the full training image.

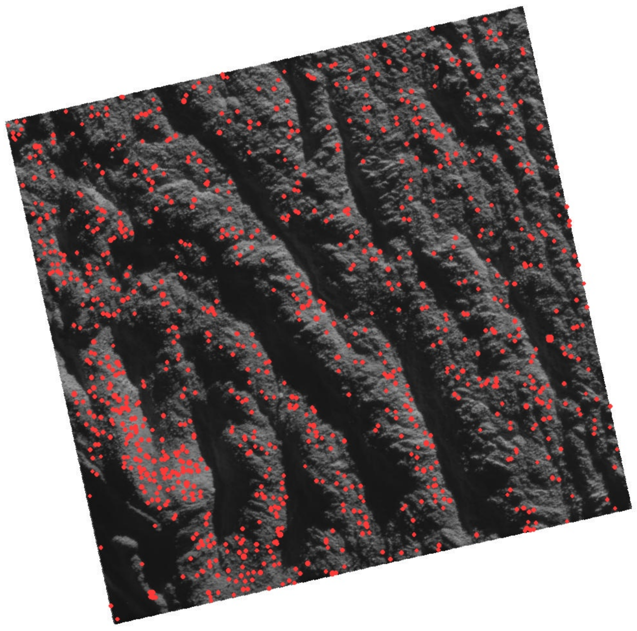


Figure 5.13: Detection on full training image

As we can see from the image the algorithm seems to be working as is giving good visual results.
By comparing the rocks detected by the algorithm with the ones manually taken we can see that way less rocks are detected so the algorithm is not performing very well in this case.
These results were quite reliable as the algorithm was not trained with images of this size but it was interesting to carry out this test to see the behaviour of the algorithm.

The last test done regards the detection on the full test image. Here's the result.

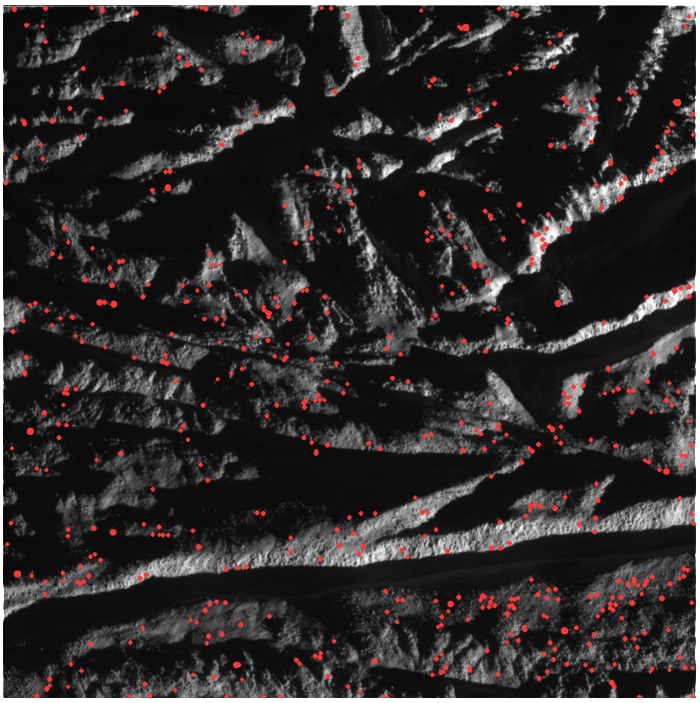


Figure 5.14: Detection on full test image

As we can see there are not many rocks identified, this may be due to both the size of the image and the exposure to light.

In order to confirm the results obtained by the training of the algorithm the images were shown to Dr. Maurizio Pajola which confirmed the goodness of the results. Despite this, a more in-depth analysis of the data must be carried out, in particular a size frequency analysis between the real rocks and the detect rocks will be done to confirm the results.

Chapter 6

Conclusions and Future Works

In this thesis, we studied the possibility to train an object detection algorithm capable of detecting rocks from satellite images of Enceladus.
The main difficulty of this task was in the dataset which was composed of a unique image with thousand of rocks in very low definition, a very common thing in satellite images since the distance from the surface doesn't permit to have high definition images.
We decided to use an already existing algorithm because I still don't have the knowledge and the experience to create one on my own which would have also required too much time for the development.
The algorithm chosen was YOLO which was already used by Dr. Aboduan, in particular, the version 3 of YOLO was the first to be tested, but given the poor results obtained we decided to test the new YOLO v5 algorithm.
The training of YOLO v5 gave good results related to precision, recall and mAP, and also the detection on the training set and validation set was good.
The analysis performed by Dr. Maurizio Pajola on the images with rocks detected by the algorithm confirmed the goodness of the results.

Future works

There are still improvements and other studies that can be done. Here's a list of some things that should be done in order to improve the result of this work:

* Train the algorithm with a different dataset such as satellite images from other planets and moons with a different types of rocks;
* Train another object detection algorithm on the same dataset and compare it with YOLO.
* Write a custom object detection algorithm appropriate to the Enceladus dataset.

Bibliography

References

Asokan, Anju. *Image Processing Techniques for Analysis of Satellite Images for Historical Maps Classification—An Overview.* 2020.

Joseph Redmon, Ali Farhadi. *YOLO9000: Better, Faster, Stronger.* University of Washington, Allen Institute for AI, Facebook AI Research, 2017.

— *YOLOv3: An Incremental Improvement.* University of Washington, Allen Institute for AI, Facebook AI Research, 2018.

— *You Only Look Once: Unified, Real-Time Object Detection.* University of Washington, Allen Institute for AI, Facebook AI Research, 2016.

Maurizio Pajola Alice Lucchetti, Lara Senter and Gabriele Cremonese. *Blocks Size Frequency Distribution in the Enceladus Tiger Stripes Area: Implications on Their Formative Processes.* INAF-Astronomical Observatory of Padova, Vicolo dell'Osservatorio 5, 35122 Padova, Italy, Department of Physics and Astronomy "G. Galilei", Università di Padova, 35122 Padova, Italy, 2018.

Object detection guide. URL: https://www.fritz.ai/object-detection/.

Saha, Sumit. *A comprehensive guide to Convolutional Neural Networks the eli5 way.* 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Solawetz, Jacob. *Breaking down Yolov4.* 2021. URL: https://blog.roboflow.com/a-thorough-breakdown-of-yolov4.

Yohanandan, Shivy. *Map (mean average precision) might confuse you!* 2020. URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2.