FACULDADE DE CIÊNCIAS E TECNOLOGIA

SAMSUNG Innovation Campus Smart Things Edition

Robot Object Detection

Group Members:
César Martins
Gustavo Sousa
Rodrigo Veríssimo

Abstract

In this study, we developed an object detection model for the Sample Fetch Rover in the Mars Sample Return Campaign, aiming to identify and collect sample tubes left on Mars by the Perseverance rover. We created two distinct models: one for detecting the sample's presence in an image and another for precise tube coordinate determination. Employing Convolutional Neural Networks (CNNs) on a dataset comprising real and synthetic images, we encountered several performance challenges. This paper primarily focuses on an in-depth analysis of those challenges, investigating potential impacts from factors such as model architecture, loss functions, layers and other relevant aspects contributing to the models' underperformance, offering insights for future planetary object detection endeavors.

1. Introduction

The project was born as a suggestion given by the teachers as a theme for the final project of this year's Samsung Innovation Campus edition.

With the constant technological evolution, there is a need to increasingly focus on space exploration in order to seek new perspectives and find new solutions, consequently, the need to optimize those explorations and reduce its costs. Nowadays, rovers, like the Perseverance Rover, are sent to Mars in order to extract samples from the surface that are stored in metal tubes and left behind for an eventual future campaign to recover those samples. Our project was developed in order to support the future missions and help rovers, like the Sample Fetch Rover, to detect the samples left on Mars' surface. The main objective was for this model to have a really low processing cost and to be compatible with weaker machines. Deep Neural Networks are also responsible for a high consumption of energy, which limits the rovers in terms of power supply and for that reason we intended to implement a solution that could reduce the electrical power consumption.

Real-Time Object Detection is a computer vision task that consists in identifying and locating objects in real-time video sequences, while maintaining a base level of accuracy. There are plenty of models that can achieve that, but the one we studied better for the main comparison to our model was YOLO.

Our objective was essentially to create an efficient and compact architecture and model that could achieve relatively good results and could be fast enough while being simple. We weren't able to achieve the results we wanted, but in this report we explain our thoughts during the process and why we think the solution we propose doesn't work as well as why what we did was actually a possible way to go.

The dataset used consists of real images taken at the Planetary Utilisation Testbed by the ExoTeR's [1] cameras (LocCam and NavCam). In those images, there is a metal tube located in different places, positions and orientations. The volume of real images was insufficient, since it is very difficult to have access to Mars' surface, a practically unreachable environment, hence it needed a synthetic solution. The solution was to generate synthetic images on a photorealistic simulator. To complement these images, we increased the dataset using Data Augmentation.

To make our model have a practical application that anyone can have access and fun with, we created a simple space on Hugging Face (a machine learning plataform) with Gradio (a Python library used to build machine learning web applications). This way, we do not need to wait for a rover to show how our model works. The link is just as follows: https://huggingface.co/spaces/gu5ousa/mars-soil-sample-localization

2. Literature Review

If we take a step back and look at the bigger picture, which is the domain of Object Detection, there are several ways to tackle this problem. As we began our research, we stumbled across already existing algorithms, which can be divided into two distinct classes: “One-stage” and “Two-stage” object detectors.

Inside the One-stage detectors’ class, we found very well-known and well-established models such as YOLO (You Only Look Once), SSD (Single Shot Detector) and RetinaNET. These One-stage detectors have the ability to produce faster outputs than Two-stage detectors. In this project, YOLO was the model we used to make comparisons with.

Among the Two-stage detectors, we have the R-CNN family with R-CNN, Fast R-CNN, Faster R-CNN, etc. Even though they are slower than the One-stage methods, they reach better detection accuracy.

In the area of object detection, the advance of studies in the area is immense. For example, Residual Networks (ResNet), developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, which won the ImageNet 2015 competition [15], is basically an architecture where the weight layers learn with the residual functions of the input of the respective layer. This architecture is extremely used for image detection and location[3].

For our work, we took some information and initial guidance from the article “Hardware-accelerated Mars Sample Localization via deep transfer learning from photorealistic simulations”[2], which is mentioned in the website from where we extracted the dataset. For our project, we only considered relevant the “detection” part of the “Sample Localization” chapter.[2] With the original dataset we used, they showed really good results by making use of the tiny version of YOLOv3 to support the project.

YOLO is an object detection algorithm that is known for its capacity to detect bounding boxes and classify objects in one single sweep, as opposed to traditional CNN algorithms, as well as being able to detect more than one label. For what we know, it is the most recurrent object detector, even surpassing ResNet .

There are many versions of the model, where the versions YOLOv1 to YOLOv4 have been developed by the original authors (Joseph Redmon) and, from versions YOLOv5 to YOLOv8 [8], by a company called Ultralytics LLC. Each version has its own characteristics. However, we opted to use the latest version of the model, with that being YOLOv8, taking also in account how YOLOv3 was used and implemented in the main article we took reference.[6]

YOLOv3, which is from 2018, has a more deepened and complex neural network architecture based in CNN, known as Darknet-53, which includes multiple resource layers. It also introduced the use of learning techniques to better precision detection[7].

YOLOv8, which is from 2020, is the most recent version and has an augmented precision and velocity. The model has the addition of the upsampling and downsampling layers, which betters the object detection.[7]

3. Data Collection and Preprocessing

The Planetary Utilisation Testbed (PUT) at the ESTEC Robotics Laboratories has been constructed to help testing robotic devices for planetary surface operations. The testbed is composed of a square 8x8 m terrain area filled with various sizes of sand, gravel and rock trying to replicate, in this case, Mars [1]. In the present day, it is near impossible to have access to Mars' surface, so, to recreate a similar environment of the planet, photos were taken by the ExoTeR's [1] cameras (LocCam and NavCam) in the respective laboratory with a metal tube placed somewhere in the local, with different positions and orientations. However, to train the model well, it was necessary to have a bigger dataset. There are many solutions for that problem, but the one chosen by the people who created the dataset was to use a photorealistic simulator. Because of the games market's frenetic evolution, there are currently really powerful engines that can recreate different environments (like in this case, with Mars), which allowed to generate synthetic images that, although not perfect, allowed to increase the dataset, give other scenarios and consequently generalize more the model who would learn from it.

For the sake of comparison and seeking some model improvement, we took 102 different photos of a speaker in different positions and locations, scattered around the house. In order to train the model, we used 'https://www.makesense.ai' to draw by hand the bounding boxes surrounding the speaker and get its coordinates. After that, we would train the model with the same architecture (to have a common ground) and compare it to our model trained with the Mars dataset.

In an attempt to further improve the model, we also created functions that could generate new images, using an image of the sample and other of a landscape to use as its background. We were able to create a set of images with different Mars landscapes with sample tubes in different positions and locations. Unfortunately, the images weren't used since the results weren't the best and the model could be learning patterns in the imperfections of the images, such as the lack of shadows, which made the generated images not that organic.

The Sample Fetch Rover, which is the rover we intend to "teach", takes photos in black and white. To be coherent, the model should learn from grayscale images, so we transformed all the dataset into grayscale. This was also a necessary transformation, considering the limitations in terms of RAM and the objective of reducing the consumption of energy. Even though it was necessary, some color features that the model could learn were lost for this transformation

We also applied resize, so we could reduce the amount of information, improve the speed and reduce the level of the models' processing, but the majority of the tubes in the images were already too small and the resize could make the tubes imperceptible and hard to detect. However, as said before, we had a lot of limitations with RAM, so it was a necessary transformation to perform.

The input images were normalized [12], more precisely its pixels, setting their values from -1 to 1 (it showed better results than from 0 to 1). The normalization has advantages, like the first layer being faster, avoiding the network to focus on the dimensions with larger input and more importantly being able to use a larger learning rate.

We also used the library Albumentations.ai and applied transformations in all of the images of the dataset, such as Random Crops, Zoom, Random Brightness, Solarize, Superpixels.

To train the classification model, it was necessary to generate images that didn't have the sample in it. Because the train dataset had the sample in every single image, it used an algorithm that could replace the pixels inside the respective bounding boxes (image areas referenced by a mask) according to the weighted sum of neighboring pixels. Some images had really good results but, in other cases, some blur is noticeable in the region where the sample was and we think that should be a relevant factor to have in mind for the lack of good results.

Throughout our project, we had a lot of RAM constraints, because of all the amount of data and processes. To overcome that, we decided to split our code in some files, more precisely a file for the pre-processing of images, one for the training of each model, a file with the final model using the two trained models and finally one used to make experiments with the dataset of images of the sound column. This decision of splitting the code came after lots of crashes while running the models..

Depending on the model or dataset in question, there are several ways to represent coordinates of bounding boxes and along the project we came across some of the formats. From our dataset, it was possible to extract bounding box coordinates in pascal_voc, i.e the output of the model is the coordinates of the top left corner and right down corner (x minimum and maximum and y minimum and maximum). To use the albumentations library, the format is the same as pascal_voc with the exception that the coordinates are normalized according to the height and weight of the images. However, for YOLO, the system of coordinates used as an output was the x and y of the centroid, the height and weight of the bounding boxes, all normalized values and those were the ones we ended up using to train our model.

4. Methodology

When tackling the making of both these models, we thought of what would be the best way to reach the desired objective based on the knowledge we obtained during the classes. The main things we believed would be important as a base line was which type of architecture

should be used for the models, in which we decided to use Convolutional Neural Networks (CNN).

Deep Learning and Convolution Neural Networks are ideal to process data in a format of matrices, grids with values like black and white images where each value represents a pixel and its intensity/tone. CNNs are made with the intuition to replicate the human brain and with that comes the possibility to process and extract features like living beings with their vision, i.e processing the images by CNNs allows to detect, find and estimate object positions (for example, tube samples).

Libraries such as Tensorflow, Keras and OpenCV were used to structure the dataset as well as build the model.

As an alternative to the model created, in order to reach better results and understand what we could improve, there is YOLO, which is pre-trained and was initially trained with a vast amount of data and capable of detecting with good precision a variety of objects. Additionally, this model is able to give the coordinates and draw boxes around the objects properly identified.

5. Model Development

We built and trained the two models separately. The classification model receives an image with or without the tube as input and, if it predicts that there is a sample in the image, the image in question will pass through and serve as an input to the regression model. The regression model will now try to predict the coordinates of the bounding box where the sample is inserted.

We built two CNN [9], one in each model. Starting with the convolution (conv2D) layers, which are the ones that require most of the network's computational load and adding to the fact that our personal computers training the model couldn't process heavy algorithms, to use a minimum number of convolution layers is a must. For our case, three layers showed the best results / efficiency while filtering tube details. The number of Kernels presented along the convolution layers is descendent for the detection model and ascendent for the regression one, being 64, 32 and 16 respectively in each layer for both models. Although intuitively adding more kernels would lead to better results, the efficiency decreases a lot and the model tends to overfit due to the amount of data and complexity.

The kernel size [13] used was set to be odd because, if it was not the case, the output would not have a central pixel and that lack of symmetry would cause distortion along the layers and more unnecessary complexity [4]. So, the choice is 3x3 in order to avoid loss of information. 1x1 filter size does not extract relevant features. The stride use is 1x1 to promote the minimum loss of information from the dataset.

This saving of information has the objective to compensate for the existent need we had to resize images and like this keep the maximum details extracted from the images. The dropout was a way we found to reduce the bias towards the training data and generalize the model more.

Max Pooling extracts the features that stand out by selecting the maximum values from the output matrix of the convolution layers. However, the rover's cameras take black and white photographs and, although this helps to reduce the computational cost and increase the processing speed of all these images, the tubes become imperceptible, since the tone in relation to the ground and the surrounding rocks is very similar. Despite all the problems with Max Pooling, it turns out to be better than Average Pooling. Most of the images had the samples far from the edges of the images, so the padding used was "valid", which causes loss of information essentially at the edges. Another option was the padding being equal to "same", combating this problem by giving an extra layer of zeros at the edges of the input of the convolution operations, but this also implies a higher cost, so a padding equal to "valid" was chosen.

Dense layers only receive one-dimensional matrices, so the Flatten function reduces the multidimensional matrices of the previous layers to one dimension. Dense layers are fully connected layers, which means that each neuron in the layer is connected to all the neurons in the previous layer. This layer has the function of learning features by combining the features of the previous layers while converging the number of neurons throughout the remaining layers to the number of outputs. Accordingly, we decided to use only 2 dense layers for both models to save on consumption, with the first having a size of 128 and the last obviously equal to the number of outputs, 2 for the detection model and 4 for the localization model.

The outputs we are trying to predict are non-linear, so the solution is to use an activation function that introduces this desired non-linearity into the model, helping with complexity and avoiding overfitting. In the last layer, for the detection model, despite the poor results, we found that the best activation function [10] was the Sigmoid, which returns values between 0 and 1 and supposedly the values of the coordinates of the normalized bounding boxes. Therefore, for the classification model, Softmax was the best, as it is better at generalization than Sigmoid. Basically, Softmax returns a probability distribution, i.e. the sum of the results is 1.

The architecture has been built, so all that remains is to train the model and adjust some other parameters.

After some training, it was clear that Adam was the main choice as an optimizer. We also tried Stochastic Gradient Descent, RMS Prop and others, but Adam was the fastest and allowed us to obtain the highest classification accuracy and precise detection in the classification model. Adam outperforms SGD by additionally and dynamically adjusting the learning rate based on the weights.

We opted for the Binary Cross Entropy loss function in the detection model. Our choice was influenced by the accuracy it offered during the training phase, even though it didn't consistently produce the best overall results. Mean squared error (MSE) was the loss used for the localization model. Another prominent loss function we explored for the regression model was the Generalized Intersection over Union (GIoU), which, in theory, is considered the most suitable choice for object detection models. Surprisingly, although this loss function led to better results, it often resulted in larger bounding boxes than necessary, occasionally covering

unrelated objects. We suspect that this result may be attributable to the nature of GIoU loss, which emphasizes the intersection of bounding boxes and may favor larger predictions.

To improve the training of our model, we used a programmed learning rate [11], which started with an initial value of 0.001 and decreased by 96% over an interval of 1000 steps and, on other occasions, a learning rate also with an initial value of 0.001 and decreased by a fifth every 5 epochs in which the validation loss did not improve. This dynamic adjustment introduced variability in the learning rate over the epochs, helping to improve training. However, a limitation of this approach is its fixed nature; it does not adapt even if changing the learning rate affects training performance negatively. With this in mind, we experimented with adaptive learning rates that adjust based on the model's response, but unfortunately this produced inferior results or overloaded our computational resources, leading to model failures and overfitting.

6. Results

To evaluate our final model, we needed a dataset that was not seen by the models or had any influence in the hyperparameters search like the validation sets did. For that, we used the entire 'field_test_images' folder of the dataset, which had 2884 unseen images. From those 2884 images, 2060 had the sample and the rest did not have the sample in them.

To have a clear understanding of the predictions the model made with the test images and make a good comparison with the results we got from YOLOv8, we needed certain specific metrics that are usually used to evaluate object detection models (and are the ones YOLO also uses): Intersection over Union (IoU), Confusion Matrix, which then gives us Precision, Recall, F1-score and Average Precision (AP).

IoU[6][14] is a metric that focuses on how much area the ground truth bounding box and the predicted bounding box share with each other. Mathematically, it is given by the intersection of the bounding boxes' areas divided by the union of their areas, which means the values of this metric are limited by 0 and 1, with 0 meaning no intersection (considering as well the cases were there are not ground truth boxes because there is no sample in the image) and 1 meaningful intersection (the boxes have the same dimensions and match perfectly on top of one another).

The Confusion Matrix, because we are discussing a single-class object detection, is much simpler then when it is a multi-class problem. It can give us 4 different variables on how our model classified and labeled the test data. In this case, the two labels were "Sample" and "No Sample" which means, respectively, "the sample is in this image" and "the sample is not in this image". True Positives (TP) is when the true label is "Sample" and the prediction was "Sample"; True Negatives (TN) shows us the number of times we predicted "No Sample" and there was not a sample in the image; False Positives (FP) gives us how many times the model predicted "Sample" and it was actually "No Sample"; False Negatives (FN) is when the true label is "Sample" and the predicted label was "No Sample".

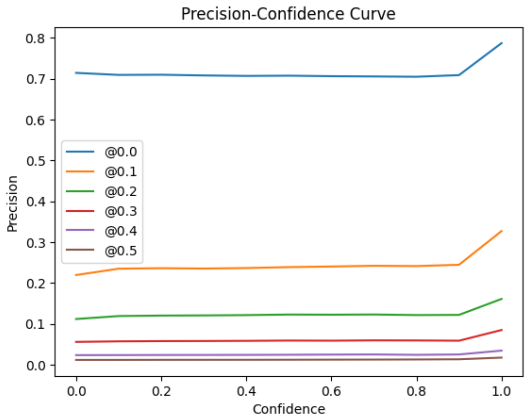
From the Confusion Matrix and its variables, we can calculate the Prediction. Prediction focuses directly on all the labels we considered "Sample", whether they were TP or FP. It indicates how much we should trust our model to catch true positives in between all the images we considered to have the sample. Mathematically, it is calculated with the following equation: "TP / (TP + FP)"

We can also figure out the Recall from the Confusion Matrix. Its mathematical equation is "TP / (TP + FN)" and it portrays how well is the model catching all the images with the sample in it.

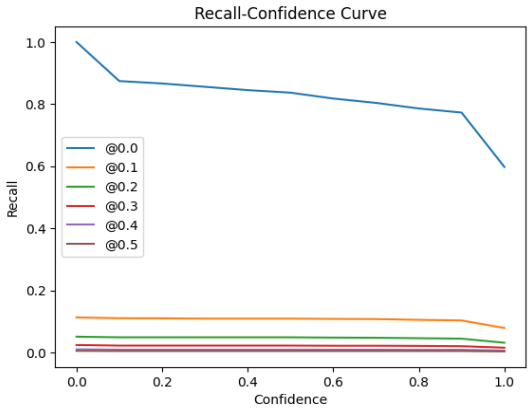
Then we have the F1-score, which is a metric that comes from the need for those who want to find the best balance between false negatives and false positives (precision and recall) in their model. As expected, its equation is influenced by false positives and false negatives: "TP / (TP + 0.5(FP + FN))"

Finally, we have the Average Precision metric. This one is the numerical value of the area under the Precision-Recall curve, where each point of the graph is given by the different thresholds we set for the confidence of the detection model (11, evenly spaced by 0.1, from 0 to 1). For example, given a confidence threshold of 0.3, every prediction of the detection model that predicted the label as "Sample" with a confidence lower than 0.3 was not considered. Every prediction above this threshold was considered as a positive label (the sample is in the image). Since we are dealing with a single-class object detection problem, there is no need to use the usual Mean Average Precision (mAP) that YOLO also brings up in its graphics (the mAP is basically just the mean of every class' AP).

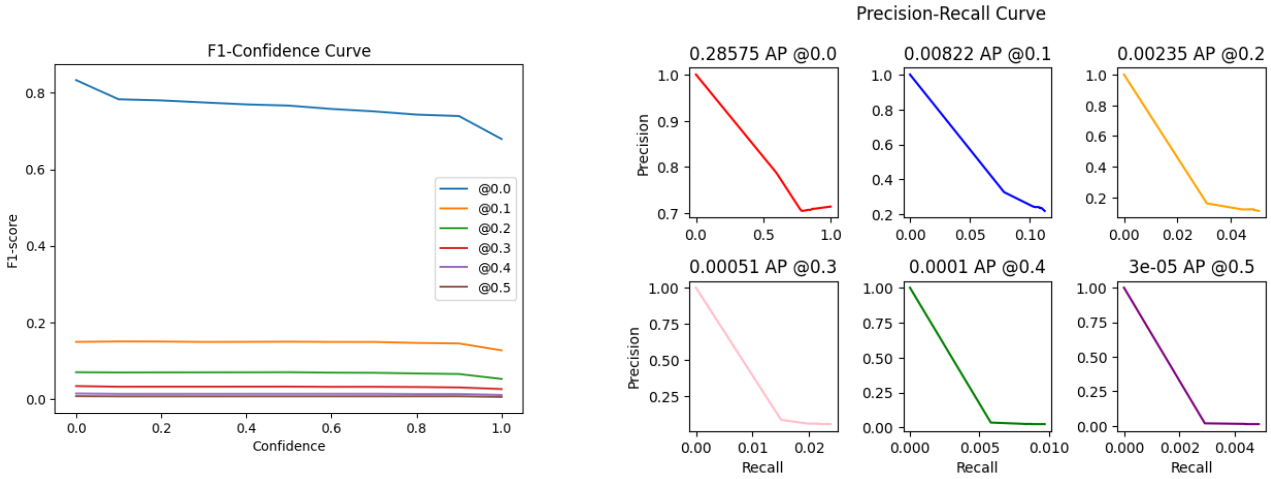
Below there is a graph for each of the four latest mentioned metrics. Each of the graphics has five different lines, with each one of them for a different IoU threshold (represented by the @). The IoU threshold has the same logic that the confidence threshold of the detection model has: for values of IoU below the given threshold, those imagens in question will be labeled as "No Sample" (negatives). Once we saw the results were not changing much after the 0.1 threshold for the IoU, we only got data from 0 to up until the 0.5 mark (six different thresholds). When we are talking about 0.0 threshold for the IoU we are basically evaluating only the detection model since even cases where there is no intersection count as one. [redacted]



(a)



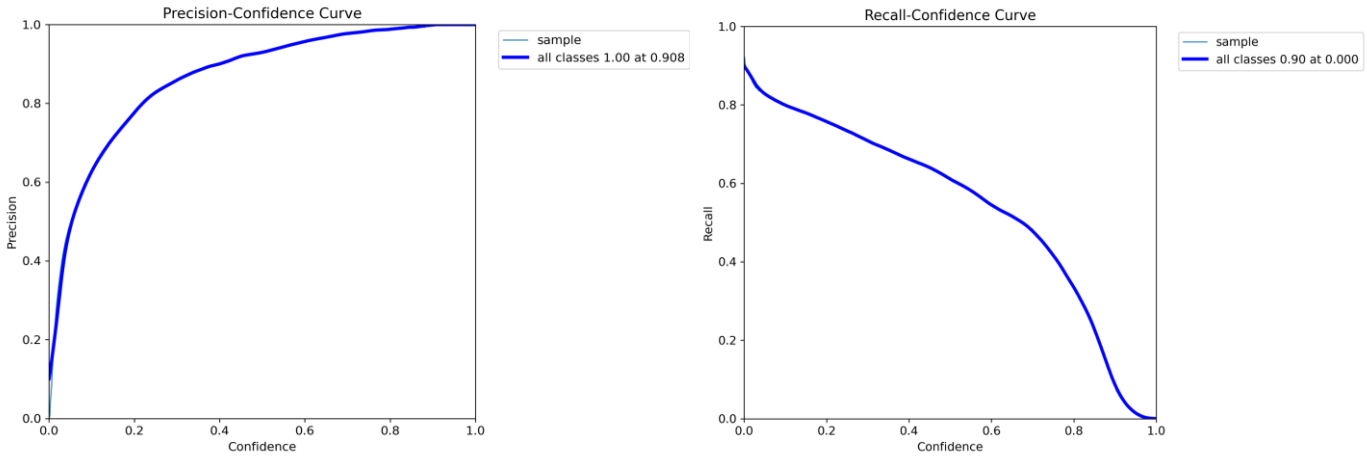
(b)



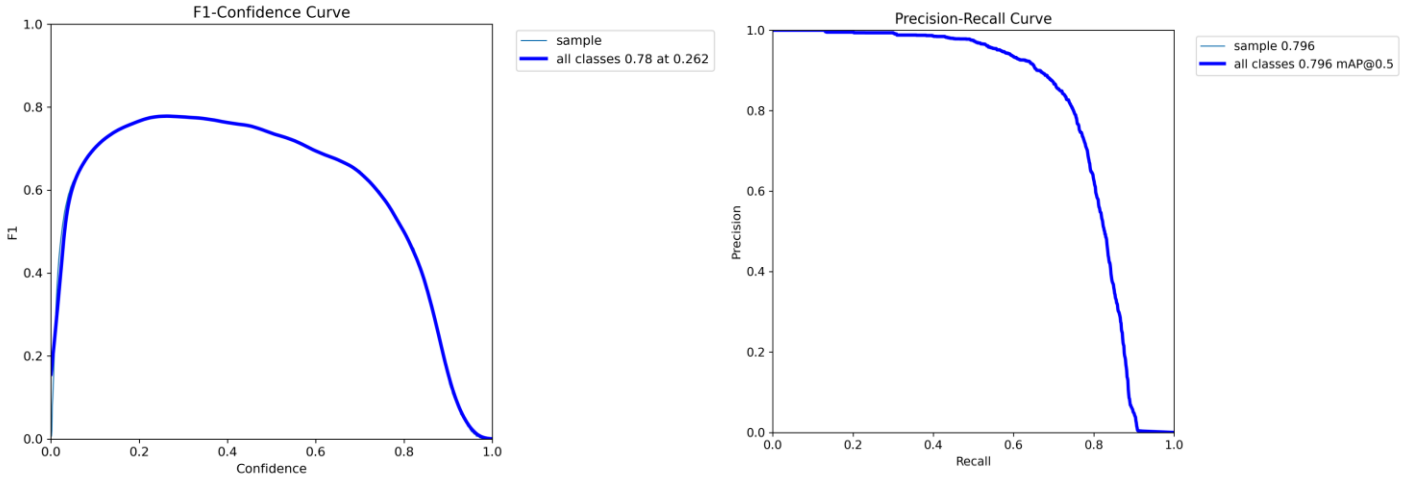
(c) (d)

Fig. 1. (a) Prediction-Confidence Curve of Model (b) Recall-Confidence Curve of Model(c) F1-Confidence Curve of Model (d) Prediction-Recall Curve of Model: all with 5 different results for each IoU threshold

If we compare our results to YOLO's, we can easily understand that YOLO showed way better results. If we look directly at the average precision: with a threshold of 0.5 for the IoU, YOLO got an average precision of 0.796; on the other hand, for the same threshold, our model got 0.00003. Even without a threshold, our model still got 0.28575 for this metric. As soon as we use a IoU threshold (which the first threshold is 0.1), the average precision values drop from 0.28575 to 0.00822, which is practically a 97% drop from the initial value. With these results, it shows how much the regression model is failing a lot of times, even though the detection model is making some good predictions.

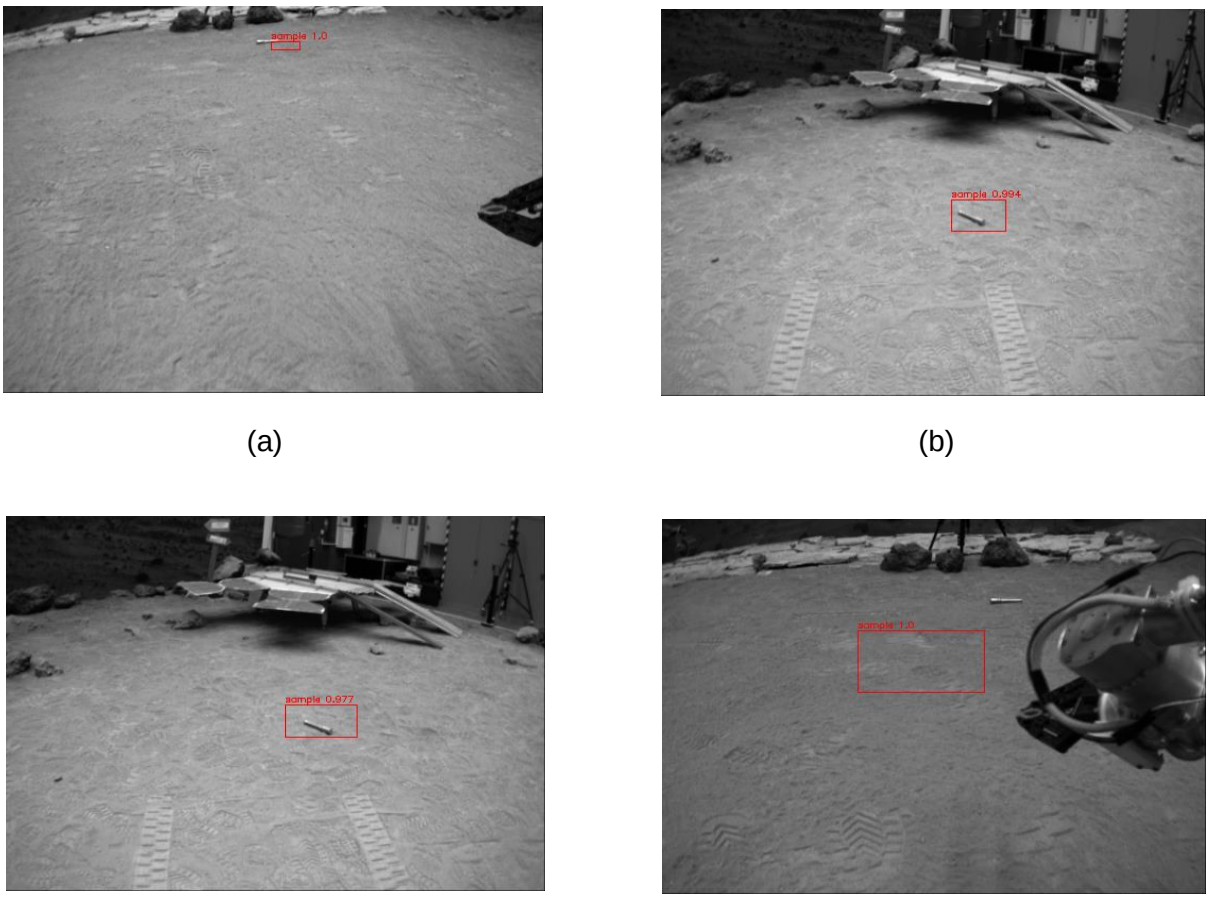


(a) (b)



(c) (d)

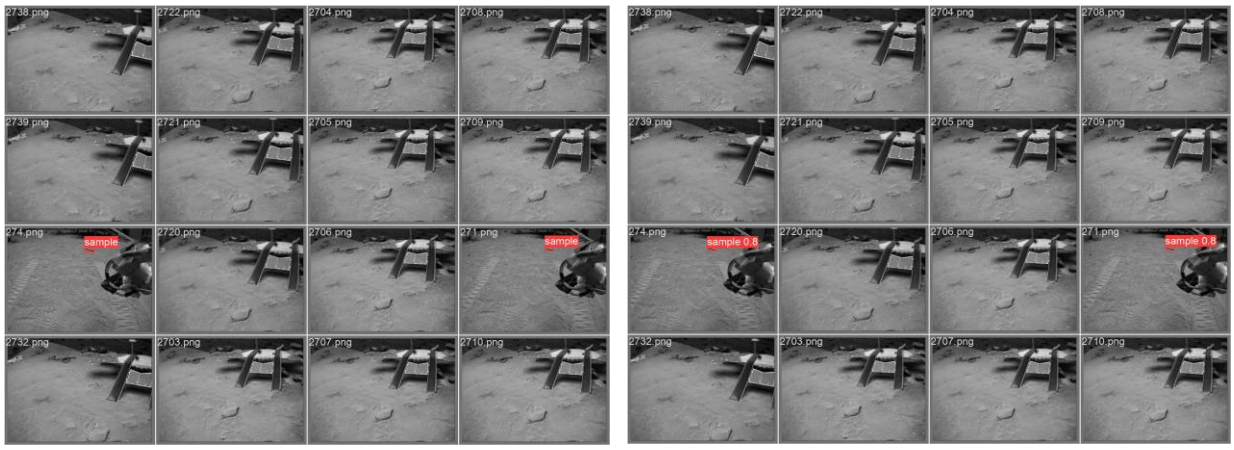
Fig. 2. (a) Prediction-Confidence Curve of YOLO (b) Recall-Confidence Curve of YOLO (c) F1-Confidence Curve of YOLO (d) Prediction-Recall Curve of YOLO



(a) (b)

(c) (d)

Fig. 3 (a,b,c,d) Our Model Prediction Results with the Test set



(a) (b)

Fig.4 (a) and (b) YOLO Prediction Results: in the left the images with the ground truth bounding boxes and, on the right, the predictions made by YOLO

To have a global understanding on how the regression model was learning, we created a function called "iou_info" that would give us useful information about how well the predicted boxes were corresponding to the ground truth bounding boxes of the entire set. The outputs of our model on this metric are as follows:

```
- for y_test_detection and y_pred_detection:
mean % of the area intersected in all intersections: 14.93403693931398
number of intersections: 379
percentage of intersections: 13.14147018030513
```

Fig.5 mean of area, number and percentage of intersections of predicted Bounding Boxes with ground truth boxes

Besides the YOLO, we also tried to train our model with the sound column dataset for further comparison and maybe improvement of our own model. However, the model just failed to learn from this dataset and it would stagnate the loss and accuracy right after the first epoch. Once we could not change the architecture of the model to make it learn (because we would lose the common ground of comparison), we were not able to get any results from that dataset.

Despite numerous attempts and modifications to both models, the results obtained were not good. One of the reasons for this may have been due to the proportion of synthetic images in relation to the real ones, creating a bias in relation to these generated images.

Although the synthetic images presented were of good quality, they had imperfections, such as the lack of shading or even the inorganic insertion of the tube in that environment, so when training the model, it could be essentially learned according to the imperfections of the synthetic images.

Another possible reason for the poorer results would be the use of black and white images, which makes the sample less highlighted. In fact, this conversion, especially for the synthetic ones, was even more impactful, since the lack of shading and the tonality of the background used made the pipe even more imperceptible.

To add to the equation, the resizing of the images further contributed to the lack of perceptibility of the tube and once again especially brought a greater problem to the synthetic images since the tube inserted there was smaller in size overall than the real ones.

7. Discussions

The results obtained, taking into account the problem statement (which is to develop a low-cost object detection model), are very plausible, since the models are basically three CNNs each, their main differences being in the hyperparameters used in their layers. Both have a basic layout for a deep neural network, which is widely used for many other problems than just object detection.

After obtaining the results, we realized that the model didn't match the objectives we wanted. This mainly told us that we should have taken a closer look at how we approached this specific problem of developing a model from scratch (as opposed to working directly with a pre-trained model and fine-tuning it) and studying the importance and impact of certain hyperparameters in the context of object detection.

A significant challenge we faced throughout our project was the limited computing resources. Certain architectural experiments we wanted to explore were often hampered by training processes that frequently failed shortly after starting. This limitation emerged as a major obstacle during our project, potentially influencing the results we achieved. Given the resource-intensive nature of image-based models, we strongly recommend using high-performance machines to develop or improve an object detection model.

8. Conclusion

During the development of this project, it was possible to come to various conclusions, especially regarding the hyperparameters to use in object detection problems.

It was important to have this kind of information in mind so that we could assess which approach would perform best with the tools we had available to work with, as well as with the problem we had in hand to solve.

One thing we noticed in both models was the need for a lot of computing power and memory. That said, for anyone who wants to create a good object detection model on their own, using only their personal computers is a tricky challenge. That's exactly what we tried to overcome with this project. Unfortunately, we realized that in order to be able to work in such a restricted environment, the performance of the model has to be sacrificed.

9. Future Work

A significant challenge we faced throughout our project was limited computing resources. Certain architectural experiments we wanted to explore were often hampered by training processes that frequently failed shortly after starting. This limitation emerged as a major obstacle during our project, potentially influencing the results we achieved. Given the resource-intensive nature of image-based models, we strongly recommend using high-performance machines when developing or improving an object detection model.

In the future and with more power, it would be interesting to return to this project and try to improve our model using other techniques or different types of models. An interesting thing to do would be to estimate the position of the object so that the robot can collect the sample more accurately. Depending on the future, work focusing on satellite images specifically for Mars would also be an idea and would somehow give the rovers even more support in extracting useful features such as the depth of surface areas.

Speaking of some not-so-similar projects, one thing we wanted to explore after this project could be, instead of working with images, exploring motion recognition and working with videos or even some work around sound, perhaps slang recognition with the recording of real people's voices.

References

[1] M. Azkarate *et al.*, “Design, Testing, and Evolution of Mars Rover Testbeds: European Space Agency Planetary Exploration,” *IEEE Robot Autom Mag*, vol. 29, no. 3, pp. 10–23, Sep. 2022, doi: 10.1109/MRA.2021.3134875.

[2] R. Castilla-Arquillo, C. Perez-Del-Pulgar, G. J. Paz-Delgado, and L. Gerdes, “Hardware-Accelerated Mars Sample Localization Via Deep Transfer Learning From Photorealistic Simulations,” *IEEE Robot Autom Lett*, vol. 7, no. 4, pp. 12555–12561, Oct. 2022, doi: 10.1109/LRA.2022.3219306.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015, [Online]. Available: http://arxiv.org/abs/1512.03385

[4] “Deciding optimal kernel size for CNN | by Sabyasachi Sahoo | Towards Data Science.” https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363 (accessed Sep. 09, 2023).

[5] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” Aug. 2016, [Online]. Available: http://arxiv.org/abs/1608.06993

[6] J. He, S. Erfani, X. Ma, J. Bailey, Y. Chi, and X.-S. Hua, “Alpha-IoU: A Family of Power Intersection over Union Losses for Bounding Box Regression.”

[7] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.02767

[8] H. Lou *et al.*, “DC-YOLOv8: Small-Size Object Detection Algorithm Based on Camera Sensor,” *Electronics (Switzerland)*, vol. 12, no. 10, May 2023, doi: 10.3390/electronics12102323..

[9] https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[10] https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

[11]https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

[12]https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

[13] https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

[14] https://sh-tsang.medium.com/review-giou-generalized-intersection-over-union-b4dd1ab89493

[15] "ILSVRC2015 Results". image-net.org.