POLITECNICO DI TORINO

Master's Degree Program
in Aerospace Engineering

MSc thesis

Image processing machine learning algorithms for interplanetary small-sats images to support Martian rovers navigation

Supervisor
prof. Fabrizio Stesina

Candidate
Elena Maria Zandri

Company Tutor
dott.ssa Chiara Leuzzi

Academic year 2021-2022

A mia madre Cristina e
mia nonna Luciana

Summary

Interplanetary exploration and, in particular, Mars exploration have, nowadays, gained interest in more and more technical and scientific research fields, from aerospace engineering to computer science and data science ones. Indeed, the latter can improve and complement the first, speeding up and making processes more efficient. This is now our science frontier and there is an evident need for this scientific and technical field to penetrate each other and work together. This Master Thesis work fits perfectly into this context, as a matter of fact our aim is to enhance Mars exploration and navigation, processing satellite data and images with Deep Learning algorithms with an ultimate goal of tenfold the Martian rovers traveling speed. This happens in the context of the SINAV project, required by ASI and led by ALTEC S.p.A., with the participation of Politecnico di Torino and other partners, leaders in the Italian space economy scenario.

The processing of satellite images is then approached with semantic segmentation methods. Starting from the supervised approach, going through the challenging labeled dataset issue, we are approaching the unsupervised one. This work presents both the approaches with their strengths and issues, underlining the reasons that led us to the choice of the Unsupervised Semantic Segmentation.

Neural Networks are the most suitable tool to approach this kind of processing. Within this work we present the use of a CNN (Convolutional Neural Network) firstly, and the use of a Visual Transformer with the aid of a "head" secondly, to distinguish different types of terrain and help the definition of the path for martian exploration by rover. Both the networks need to be tweaked to adapt to peculiarities of Mars terrain images, rarely approached with these kinds of algorithms.

The dataset is provided by HiRise images (High Resolution Imaging Science Experiment) catalog, in particular the images are mostly from the Jezero crater and Gale crater area, and preprocessed to best fit our neural network and GPUs constraints.

Acknowledgements

Contents

List of Tables 8

List of Figures 9

Acronyms 11

1 Problem presentation 15

1.1 Sinav project 15

1.2 Approach and methodologies 17

1.2.1 Semantic segmentation 17

1.2.2 Artificial Intelligence and Deep Learning 22

2 State of the art 23

2.1 Segmentation Convolutional Neural Networks for Automatic Crater Detection on Mars 23

2.2 Automatic Detection and Segmentation of Barchan Dunes on Mars and Earth Using a Convolutional Neural Network 26

2.3 Benchmark Analysis of Semantic Segmentation Algorithms for Safe Planetary Landing Site Selection 28

2.4 Mars with less labels 33

2.5 STEGO 34

3 Deep Learning for image segmentation 39

3.1 Convolutional Neural Networks 39

3.2 Specific application: ConvDeconv Architecture 43

3.3 Vision Transformer - ViT 45

3.4 Specific application: Dino + Stego 46

4 Code implementation and Dataset preparation 49
4.1 Dataset 49
4.1.1 HiRise DTM 50
4.1.2 HiRise realeses 51
4.2 Method and code implementation 53
4.2.1 ConDeconv adaptation 54
4.2.2 STEGO adaptation 58

5 Results 67
5.1 STEGO validation with Syntethic dataset 67
5.2 Results and prediction on HiRISE 71

6 Conclusions and further steps 73

List of Tables

2.1 Comparison of unsupervised segmentation architectures on 27 class CocoStuff validation set 37
4.1 Confusion matrix example 59
5.1 Synthetic Dataset metrics and configurations 70

List of Figures

1.1 Drawing of the SINAV project acrchitecture. 16
1.2 Operational diagram of a possible future exploration mission equipped with wide-ranging fast navigation systems. 17
1.3 Classification vs Object detection 18
1.4 Image from HiRise database, Jazero2020 20
1.5 3 clusters segmentation 20
1.6 4 clusters segmentation 20
1.7 5 clusters segmentation 21
1.8 Semantic segmentation vs Instance segmentation 21
2.1 Default Crater U-Net structure 24
2.2 Mars in the mid-latitudes, original THEMIS dataset. The top row consists of 0° to 30° latitude, and the bottom row consists of –30° to 0° latitude. The left edge is 0° longitude. Each tile is 30° by 30° (7680 by 7680 px) with a resolution of 256 px per degree (231.55 m/px). 24
2.3 Comparison of solid and edge targets. Craters of interest are highlighted in red boxes. 25
2.4 Mask R-CNN schematic 26
2.5 The global number density of barchan dunes on Mars 27
2.6 Dune identification samples 28
2.7 SegNet Network architecture 29
2.8 Fully Convolutional Network 29
2.9 Image Cascade Network 30
2.10 U-Net 30
2.11 s the accuracy of each model with respect to its computational complexity on the X-axis, and its model complexity (number of parameters) as the size of the ball 31

2.12 The accuracy density by the number of parameters to achieve that result . 32
2.13 Inference time vs accuracy 32
2.14 Loss Function, mean Intersection over Union, Pixel Accuracy 33
2.15 Loss Function, mean Intersection over Union, Pixel Accuracy 34
2.16 Some predictions 35
2.17 Comparison of ground truth labels (middle row) and cluster probe predictions for STEGO (bottom row) for images from the Cityscapes dataset. . . 37
2.18 Qualitative comparison of STEGO segmentation results on the Potsdam-3 segmentation challenge. 38
3.1 Input - Filter - Output. 40
3.2 40
3.3 42
3.4 ConvDeconv architecture 44
3.5 “An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale“ 46
3.6 “Emerging Properties in Self-Supervised Vision Transformers“ 47
3.7 “Emerging Properties in Self-Supervised Vision Transformers“ 47
3.8 “STEGO architeture“ 48
4.1 “HiRISE DTMs interface and products“ 51
4.2 DTMs extra products 52
4.4 Images pre-processing 61
4.5 “End-to-end schematic process“ 64
4.3 “HiRISE RGB - NOMAP product“ 65
5.1 Classes mean pixel value 69
5.2 Respective prediction for validation images n° 119 and n° 120 71
5.3 Source images and predicted masks 72

Acronyms

AI Artificial Intelligence.

ALTEC Aerospace Logistics Technology Engineering Company.

ANN Artificial Neural Network.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

CRF Conditional Random Field.

DEM Digital Elevation Model.

DINO Distillation with no labels.

DL Deep Learning.

DNN Deep Neural Network.

DSM Digital Surface Model.

DTM Digital Terrain Model.

EDR Experiment Data Records.

EMA Exponential Moving Average.

FCN Fully Convolutional Network.

FN False negative.

FP False positive.

GAP Global Average Pooling.

GB Gigabyte.

GPU Graphical Processing Unit.

GT Ground Truth.

HiRISE High Resolution Imaging Science Experiment.

IoU Intersection over Union.

IR Infrared.

KNN K-nearest Neighbors.

LR Learning Rate.

mAP mean Average Precision.

mIoU mean Intersection over Union.

MLP Multilayer Perceptron.

MSP Multilayer Self-Attention Perceptron.

NN Neural Network.

PDS Planetary Data System.

px pixel.

RDR Reduced Data Records.

RGB Red-Green-Blue.

ROI Region of Interest.

RPN Region Proposal Network.

STEGO Self-supervised Transformer with Energy-based Graph Optimization.

TN True negative.

TOF Time-of-flight.

TP True positive.

ViT Vision Transformer.

Poca osservazione e molto ragionamento conducono all'errore.
Molta osservazione e poco ragionamento conducono alla verità.

[A. Carrel]

Chapter 1

Problem presentation

1.1 Sinav project

This Master Thesis work is born inside the Italian project SINAV, financed by ASI (Agenzia Spaziale Italiana). It is led by Altec s.p.a. with the participation of Politecnico di Torino and two other partners, leaders in the Italian Space Companies scenario. ALTEC – Aerospace Logistics Technology Engineering Company – is the Italian center of excellence for the provision of engineering and logistics services to support operations and utilization of the International Space Station and the development and implementation of planetary exploration missions, in which I had the chance to develop this work.

SINAV project has four main targets to be pursued:

- A tenfold increase in the travel speed of current space rovers from 10-20 m/hour up to 100-200 m/hour.
- Proposal of an autonomous system not dependent on lighting conditions and therefore capable of operating at night through the use of active sensors, currently being space qualification, for autonomous navigation (i.e. TOF camera or laser strips).
- Investigating collaborative exploration scenarios between rover and satellite/drone.
- Processing of images acquired by the rover and aerial/satellite systems in line with the most recent approaches proposed in the field of Artificial Intelligence based on the semantic analysis of images, in this process Deep Learning is an essential element.

Nowadays scientists are very interested in exploring steep ravines or caves in search of possible traces of current or past life. Furthermore, the use of satellite data is critical in

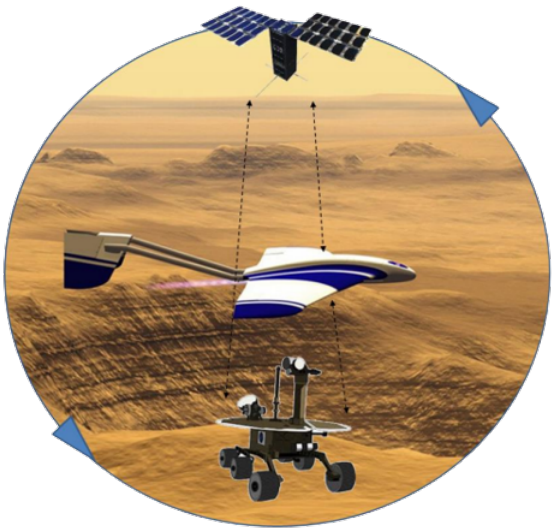


Figure 1.1: Drawing of the SINAV project acrchitecture.

route planning and target selection during the execution and planning of robotic missions. Although the complexity of management precludes near-real-time execution, some collaborative processing procedures between scientific data and satellite and rover operational teams have already been proposed and studied.

Therefore, the study of specific algorithms and data processing is proposed to increase the extension of the area explored within the same mission duration. This also means a greater speed of movement of the rover and a fully autonomous path planning. In addition it also aims to achieve more autonomy in the search for interesting microhabitats and targets and in determining soils or rocks that might be of interest for the mission.

Although the scenario envisaged for this study is Mars, the results obtained could be used for other future interplanetary missions such as Lunar exploration, nowadays center of interest in this field.

The studied scenario involves a satellite or aerial platform that will acquire images of the rover and the surrounding terrain, which will be pre-processed by the platform itself and/or a lander that, possibly after releasing the rover, will perform the task of ground station for data-relay and 'calculation node'. The aim is, again, to demonstrate that by preprocessing satellite images, it is possible to determine inaccessible and dangerous areas or evening interesting ones, before the rover itself could have seen them.

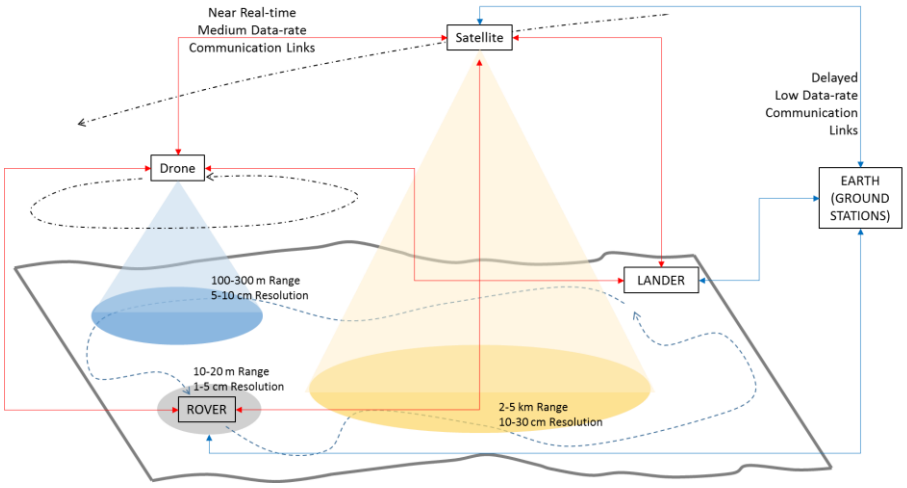


Figure 1.2: Operational diagram of a possible future exploration mission equipped with wide-ranging fast navigation systems.

1.2 Approach and methodologies

Hence inside this Master Thesis work we want to develop and deepen a neural network to perform image semantic segmentation on satellite captures from Mars surface.

1.2.1 Semantic segmentation

Image segmentation is a computer vision task that aims at an understanding of images, it involves various levels of granularity and the coarsest is the image classification.

With image *classification* we ask the computer to return a discrete label that identifies the principal object in the image. In order to classify an image, we presume that it contains just one object, not several.

If we ask for a *localization* too, we expect the algorithm to return some parameters to identify a bounding box. Even in this case we presume the image contains only one object.

Moving to a finest problem we find *object detection*: object detection takes localization to a higher level by allowing the image to contain multiple objects rather than just one. All of the objects in the image must be classified and located. Again, the concept of bounding boxes is used for localization.

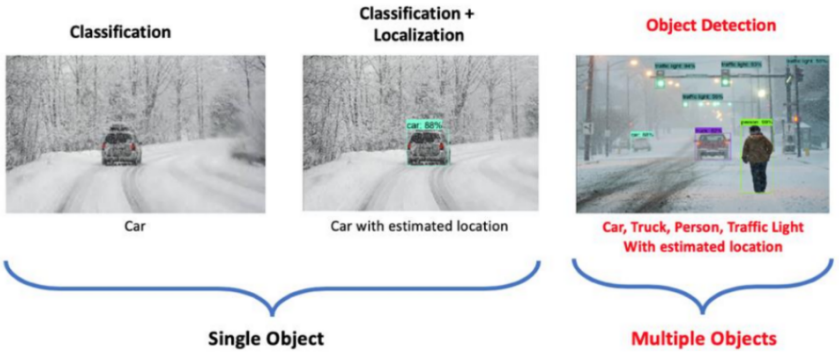


Figure 1.3: Classification vs Object detection

Image segmentation is then the next step and the finest one. The final result is a mask or a matrix with different elements indicating the object class or instance that each pixel belongs to.

I now wish to specifically deepen two approaches to image segmentation, one without the usage of a neural network and the other with it, and we'll also look into some particular methods that are relevant for space applications.

Image segmentation without the use of Neural Networks

Threshold segmentation The simplest technique for segmenting images is called threshold segmentation. It is a typical segmentation algorithm that splits the processing of image gray scale information depending on themultiple targets varying gray values. There are two types of threshold segmentation: local threshold method and global threshold method. With the global threshold method the target and background are separated, to get two parts of the image, by a single threshold. On the other hand, the local threshold approach separates the image into numerous target regions and backgrounds by multiple thresholds, and requires the user to pick multiple segmentation thresholds.

The threshold technique has the advantages of being faster and having a simpler calculation with respect to other techniques.The segmentation can be well produced, in particular, when the target and the background have significant contrast. The drawback is that it is challenging to produce reliable findings for images where there is little to no grayscale difference or high grayscale overlap in the image. Due to its sensitivity to noise and uneven grayscale, which results from simply taking into account the image's gray information and ignoring its spatial information, it is frequently supplemented with other

approaches.

Otsu N. [1979] is an interclass variance method and the most used threshold segmentation algorithm. The algorithm will be able to recognize each of the pixels in the image as a single entity using this method if their values are lower or higher than the threshold value that is set on the image histogram. The image returned in this instance is a binary image.

In this case it aims to find the threshold value that maximizes the variance between the two classes:

$$\sigma_B^2(k^*) = max\sigma_B^2(k)$$

considering:

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2$$

This method can be extended to multi thresholding situations, but as the number of classes to be distinguished rises, the selected thresholds tend to lose some of their validity.

Kmeans K-means clustering is a machine learning method but it does not employ neural networks. It is also an unsupervised algorithm, therefore it doesn't require a ground truth to use. A cluster is an aggregation of data points with related characteristics.

First of all, k-means needs the definition of the number of clusters we want and consequently the number of centroids, i.e. the locations of the clusters center. Each datapoint is then allocated to the closest cluster. At the beginning, the centroids are randomly selected, then, in each iteration they are defined with calculations to optimize the position. Indeed, finding the centroid is what "means" in the K-means algorithm indicates: averaging the data each iteration.

This type of segmentation is fast, simple, high efficient and scalable on large dataset but it has also some drawbacks, for example the number k no explicit selection criteria and difficult estimation, very expensive time of the algorithm and finally the fact that it is applicable only to convex dataset.

Here are some examples of K-means clustering segmentation I produce during my first approach to image segmentation. The image to be segmented is part of the HiRise product, in particular it represents the Jezero crater.

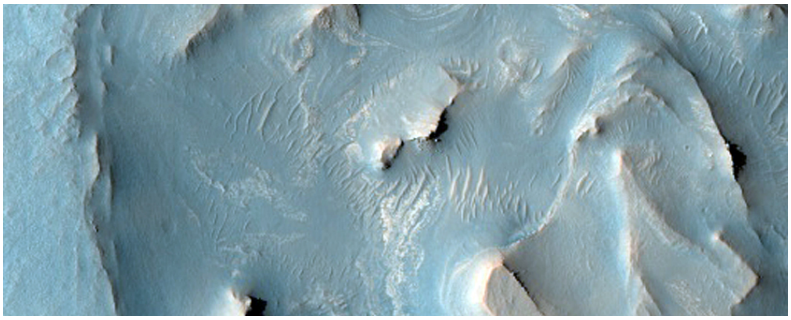


Figure 1.4: Image from HiRise database, Jazero2020

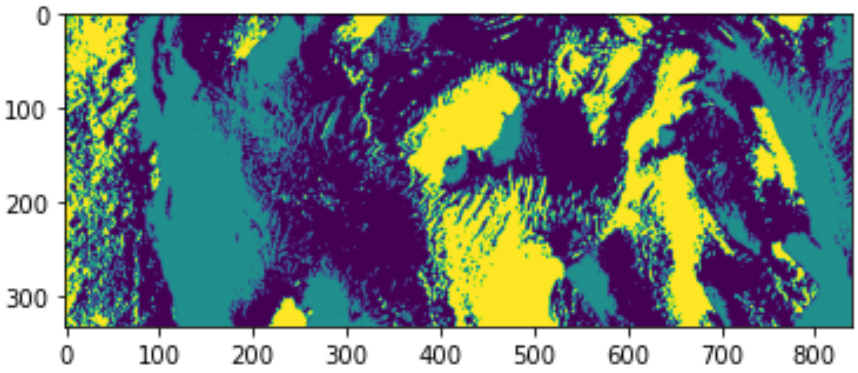


Figure 1.5: 3 clusters segmentation

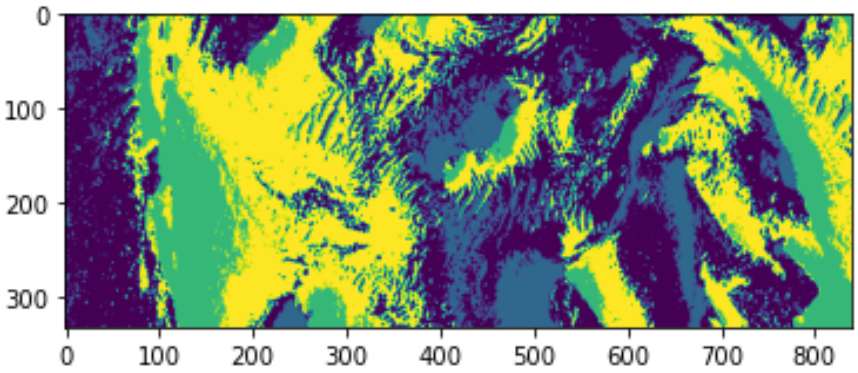


Figure 1.6: 4 clusters segmentation

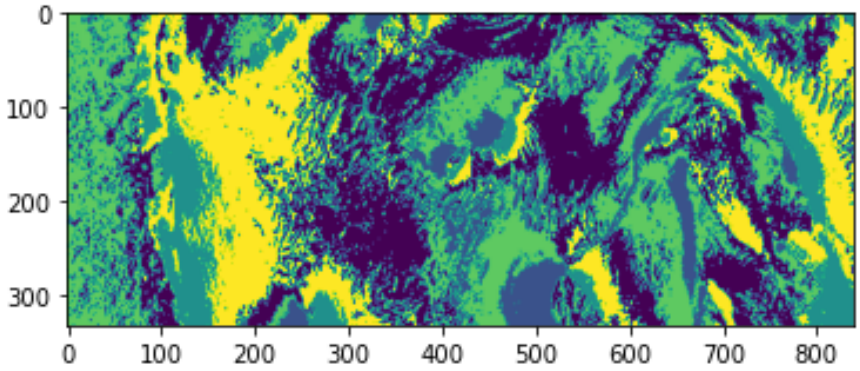


Figure 1.7: 5 clusters segmentation

Image segmentation with the use of neural networks

Neural network image segmentation can be divided into 2 different types: semantic segmentation and instance segmentation.

With the use of semantic segmentation, we aim to label each pixel of an image with a class that defines which one that pixel is representing. This means that the boundary of my object is no more a box, but a clear and defined contour. The output of this task is an image with the same high resolution of the input image, that is different from the only labels and bounding boxes output seen in the previous tasks. We can define this as a pixel level image classification.

On the other hand, instance segmentation makes a further distinction, identifying each example object separately. More easily, we define each single instance rather than the generic class of semantic segmentation.

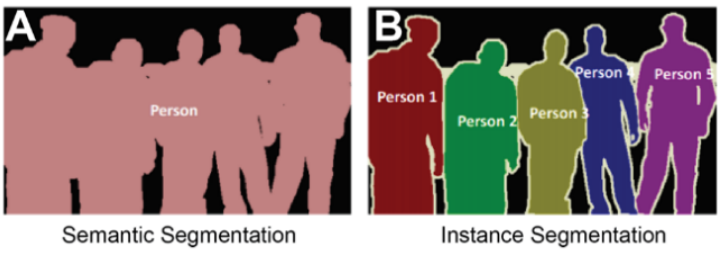


Figure 1.8: Semantic segmentation vs Instance segmentation

We can leave the manual approach to image segmentation thanks to neural networks architectures.

1.2.2 Artificial Intelligence and Deep Learning

In terms of techniques and approaches that can be used, the field of Artificial Intelligence is extremely broad. Machine Learning is a particularly active field and its name refers to a rather diverse set of algorithms, including kNN, K-Means, SVM, Kalman filter, and ANN. These enable the classification or processing of input data in probabilistic terms. The SINAV project and this thesis focus in particular on the use of ANN (Artificial Neural Networks) and more precisely on DNN (Deep Neural Networks).

A neural network is a series of algorithms that want to replicate the operations of a human brain. It consists of layers of interconnected artificial neurons and as in traditional machine algorithms, neural networks can learn in the training phase certain values.

The first Deep Learning (DL) approach was presented in 2010 and then in 2013 as a participant to the ImageNet competition. It was the winner of the competition with a 15% percentage of enhancement compared to all the other algorithms. It is able to categorize, for example, a dataset of 1.2 million of images with a 96% accuracy. These percentages are even higher than the human ones.

Such algorithms are extremely efficient at performing tasks and can greatly help in the optimization of space mission processes, from the initial operation to the final results. However, no space missions with this type of application, such as the use of DL to help path planning and merge the global information to the local one, are currently scheduled. The information we expect from this kind of DNN (Deep Neural Networks) is qualitative, and it is similar to the information we humans have of our distant surroundings when we move.

The approach we take begins with a review of the state of the art, which has relevant research studies that helped us and guided us in the selection of two NN to adapt to our case study, one supervised and the other not, culminating in some result comparison and visualization.

Chapter 2

State of the art

In this chapter we want to present the state of the art for what concerns the use of DNN in particular for Mars terrain segmentation studies. It was not easy to find examples in the field of interplanetar exploration due to the fact that most of the applications of Deep Neural network architectures are in the fields of biomedical images analysis and autonomous driving on Earth, for which it was born. We choose the following papers as the most relevant in terms of approach they used and as the ones with the most notable results. They are going to be presented with a lunge regarding neural network architecture, training and validation dataset, preprocessing on the images, metrics and results.

This State-of-the-Art research paved the way for the definition of the approach to follow to reach our objectives, we are going to explain in the next chapters.

2.1 Segmentation Convolutional Neural Networks for Automatic Crater Detection on Mars

This study, by DeLatte et al. [2019], takes into account the problem of optimization with the aid of DL for semantic segmentation for the crater detection and identification on the surface of a planetary body, and finally the measurement of its size. Craters can be caused by impacts or erosion and their identification and location could help for age dating, positioning, and hazard avoidance. Dataset of large dimension for crater counting have been published in the last years and are available to be used with machine learning techniques and to automate the recognition process. The best technique is the one that uses neural networks, in fact these kind of architecture are able to do complex pattern

recognition on their own. In particular, they chose the U-net architecture, a modified CNN, whose structure you can see in the figure 2.1 and we are going to discuss in next chapters.

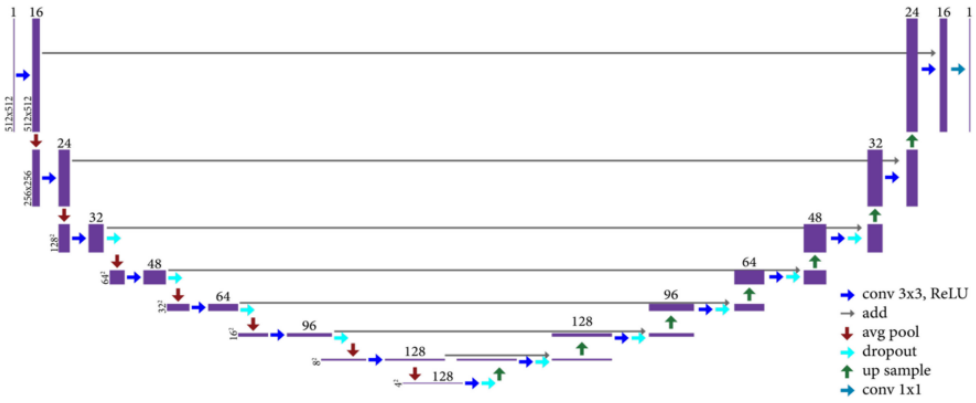


Figure 2.1: Default Crater U-Net structure

U-net is a supervised architecture so it needs a labeled dataset. The choice fell on the one created by Robbins and Hynek [2012a] that identifies craters down to 1 kilometer. They use circular representation of craters to create all annotations and targets, without taking care of crater degradation level. In particular, those images were chosen for the high resolution they have and they only used a ±30° latitude range to simplify processing, indeed in that latitudes the projection of craters remains circular and doesn't stretch into ellipses. Moreover, using only a specific dataset, the network will be able to find craters only in that specific style. The dataset contains 24 tiles of 30° by 30°, each is 7680x7680 px then split into 512x512 px sub-images which will be stitched together again after the prediction.

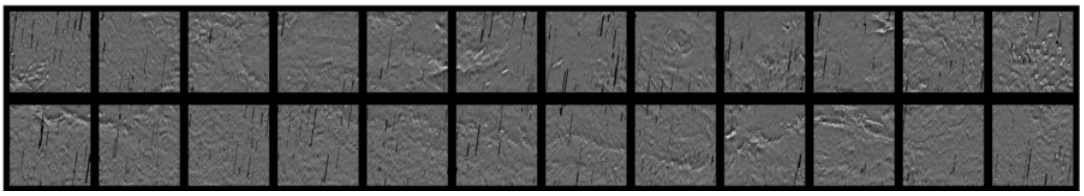


Figure 2.2: Mars in the mid-latitudes, original THEMIS dataset. The top row consists of 0° to 30° latitude, and the bottom row consists of –30° to 0° latitude. The left edge is 0° longitude. Each tile is 30° by 30° (7680 by 7680 px) with a resolution of 256 px per degree (231.55 m/px).

For this type of application the image segmentation to be used is the instance segmentation. Indeed, it is the most appropriate approach for counting crater in an automatic way, as told in the previous chapter it identifies each example object separately. Crater U-Net is the network used in this study and differs from the classic U-Net only in:

- the output image dimension, that is equal to the input one,
- the use of average pooling instead of max pooling,
- the presence of a pooling layer right after each convolutional layer, instead of having two in a row.

In order to evaluate the predictions of the neural networks, the output images are compared to the human annotation/labeling. Some metrics should then be defined as, for example, precision, recall and F1, for which we need the definition of TP, true positive, FN, false negative, and FP, false positive. They test the network with two types of target and the results are made up comparing loss and accuracy. The first targets are the solid circles and the second targets are the edge targets that allow craters within craters to be detected.

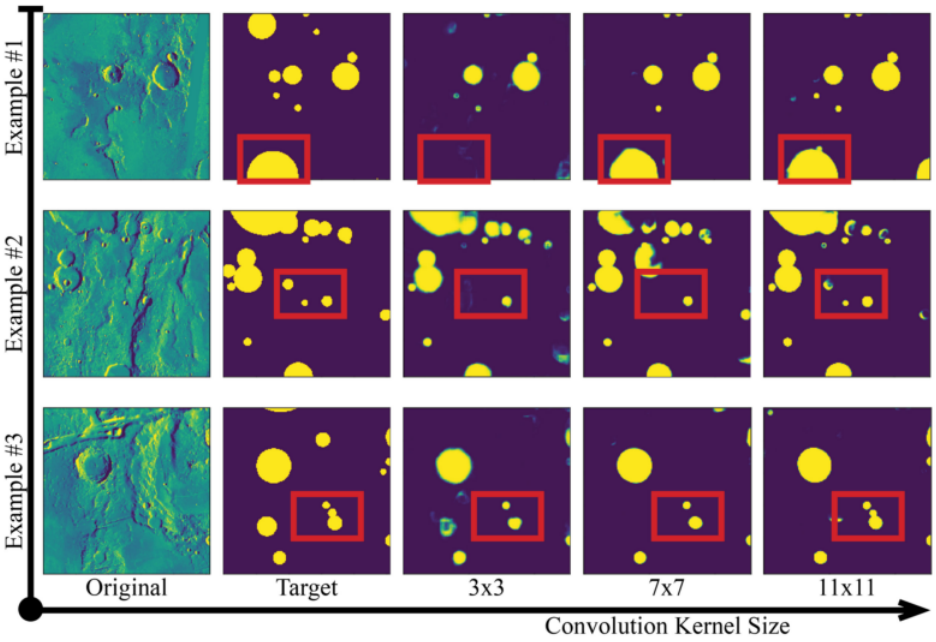


Figure 2.3: Comparison of solid and edge targets. Craters of interest are highlighted in red boxes.

2.2 Automatic Detection and Segmentation of Barchan Dunes on Mars and Earth Using a Convolutional Neural Network

The characteristics of a barchan dunes are, first of all, the horn shape pointing downwind and secondly a steep angle of repose slipface in their lee, while the orientation is normally in the dominant wind direction. Before this study was published, dunes have been vastly analyzed using aerial and satellite images, but only with a manual approach to the problem, due to the fact that automatic detection techniques were not accurate enough. Indeed, the difficulties were separating the features of interest from the background. Digital Elevation Model (DEM) could be useful in this task but they are often with a lower resolution than the required.

To demonstrate the efficacy of Deep Learning in solving this task, Rubanenko et al. [2021] present the use of a Mask R-CNN, an instance segmentation neural network whose schematic is shown in figure 2.4.

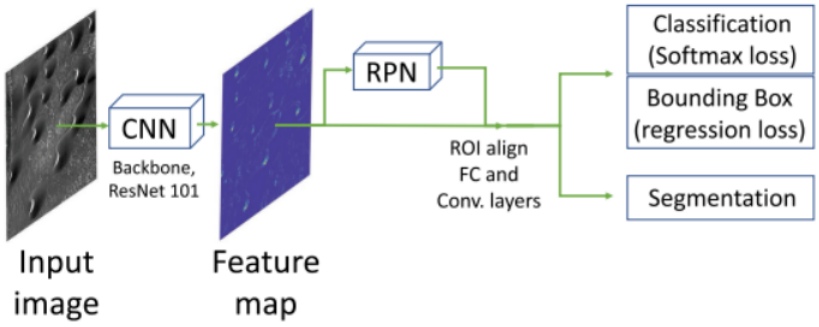


Figure 2.4: Mask R-CNN schematic

Inside a Mask R-CNN we have three stages. The first is the backbone, a CNN that extracts from the input image a future map, usually a well-studied architecture is chosen. The second stage involves the RPN, a region proposal network, that proposes positive and negative anchors after scanning the feature map, where with anchors we intend parts of the image that can contain objects or background. In particular, the positive anchors bounds were defined with the ratio between the intersection and the union of the predicted bounding box with the manually labeled one. After that, they join the positive anchors, filtering them too, and they create regions of interest (ROIs). In the last stage a pyramid

network examines the ROIs and determines their classification and their masks, thanks to the use of several loss functions.

To evaluate the model, the used metric is mAP, the mean Average Precision, where true positive (TP), false positive (FP) and false negative (FN) are considered.

Then, two options were taken into account to initialize the model weights, one with random values and the other one with values already optimized for a different dataset, the second process is best known as transfer learning. For the optimization of the hyper-parameters, in contrast to model weights that are optimized during training,they are only prescribed at one set. By hyperparameters we mean for example, the number of layers of the backbone (its depth) or parameters that regard the training itself (learning rate, convergence rate). Above all the detections, all the images containing less than two dunes were discarded because barchan dunes usually appear in fields of isolated dunes and rarely as solitary landforms; they did so to avoid the inclusion of spurious detections.

The resulting distribution of barchan dunes on Mars is shown in the map (a) that displays the number density of detected objects per 100 km. With white contours dune fields of all types are indentified and we can notice that in the northern polar region most of them are barchan dunes, in marked contrast with the southern polar region. We can also see in figure (c), (d) and (e) some examples of what objects the neural network recognized to be a barchan dune, in particular a true positive detection in (c), a false positive in (d) and a true negative of South Crater dunes in (e). At the end they also tested this network, trained with martian satellite images only, with terrestrial barchan dunes images reaching an accuracy of 50 - 70%.

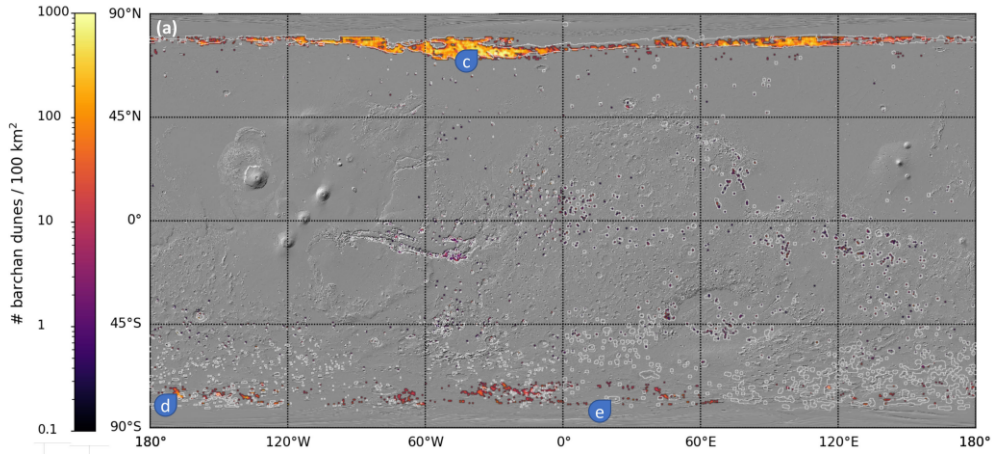


Figure 2.5: The global number density of barchan dunes on Mars

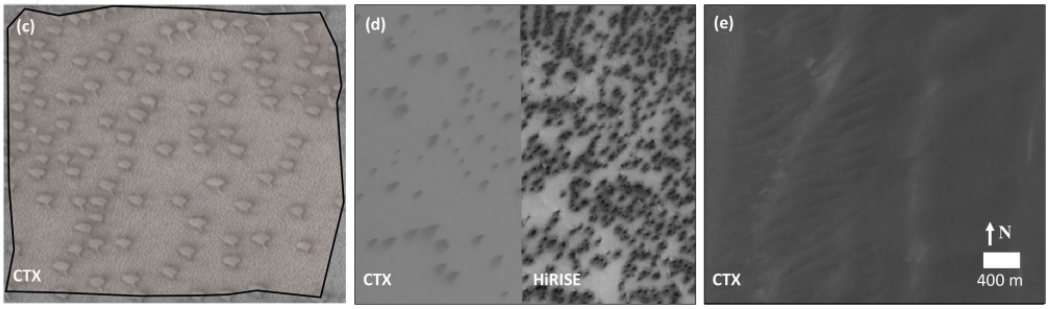


Figure 2.6: Dune identification samples

2.3 Benchmark Analysis of Semantic Segmentation Algorithms for Safe Planetary Landing Site Selection

Nowadays, safe landing is still the most critical part of every space mission that wants to reach the ground to conduct experiments, even if it has already switched to fully autonomous thanks to the exploration of Mars. The problem is also that the outcome can only be seen with a delay due to communications and in the case of Mars this means having news only after the landing has already occurred. Moreover, the Martian surface, and particularly the surface of regions of scientific interest, is plenty of cliffs, craters, cracks, jagged boulders that are for all intents and purposes landing obstacles. For these reasons high requirements on the safety level prediction accuracy are needed.

Generally this study, by Claudet et al. [2022], wants to contribute in 3 different points, but the one that caught our interest was the one that benchmarks some algorithms and neural network architectures using metrics to evaluate their performance.

Seven architectures have been taken into account:

1. SegNet
2. FCN
3. ICNet
4. U-Net and TransUNEt
5. ENet

6. ConvDeconv

7. DeepLabV3 with a ResNet backbone

SegNet, which schematic you can see in figure 2.7, is an encoder-decoder network with a pixel-wise classification layer. It stores the pooling indices and uses them in the upsampling phase.

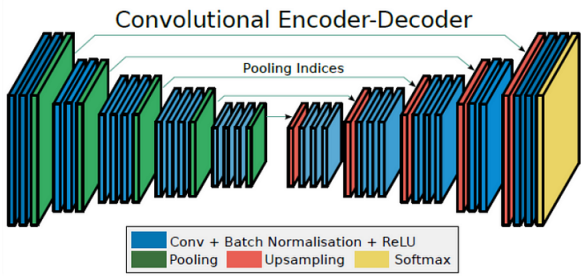


Figure 2.7: SegNet Network architecture

Fully Convolutional Network has the encoding process similar to the one in SegNet but differs in the fact that it doesn't store the pooling indices. Three FCN are taken into account, 32s, 16s, 8s going respectively from the one where the most information is lost to the final where the accuracy is higher as it loses less information. The last one is also the heaviest on a computational level.

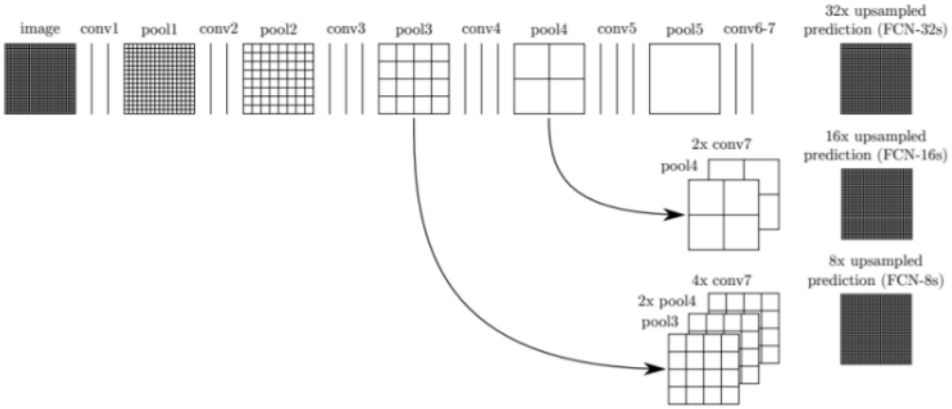


Figure 2.8: Fully Convolutional Network

The Image Cascade Network (ICNet), depicted in 2.9, operates in a series of steps. First, the low-resolution images reach the network's endpoint and a preliminary mapping

is obtained. Furthermore, medium and high resolution features are available. It can be improved with a cascade feature fusion unit and a cascade label guidance strategy.

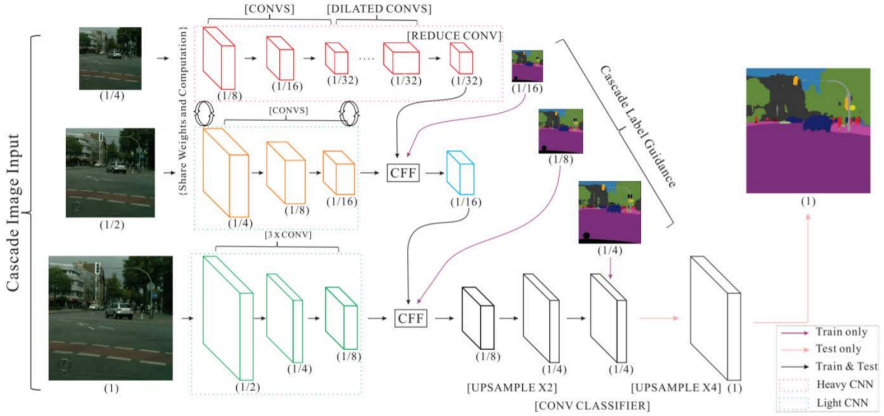


Figure 2.9: Image Cascade Network

U-NET is an upgrade of the FCN providing higher precision with a smaller number of training images. The core is replacing pooling operation with upsampling to reach a better resolution of the output. It also integrates the use of skip connections that allows U-Net to copy the image matrix from the earlier layers and to use it in the later layers. Doing this, the model reduces information loss. It also has many channels inside the decoder to better propagate context information to layers with high resolution, this is what gives U-Net its characteristic shape.

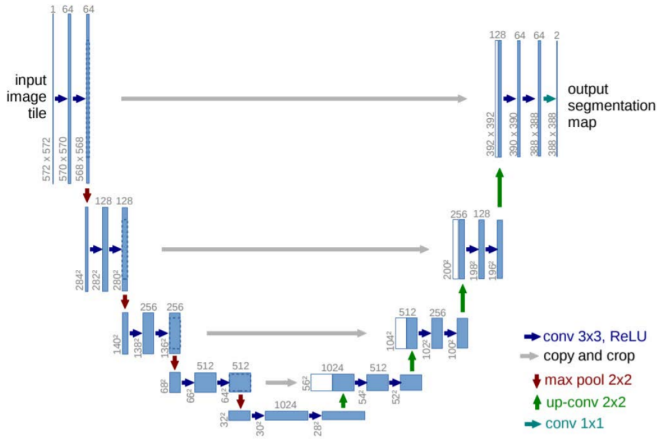


Figure 2.10: U-Net

Efficient Neural Network (ENet) uses an encoder-decoder structure as well but it is

more well fitted with real-time uses. It down-sample in the decoder in the early phase to cut the cost due to large input frames processing. It uses the PReLUs activation function.

ConvDeconv takes its cues from SegNet, using a smaller kernel size and reducing the number of layers. On the other hand, it saves the pooling indices as SegNet does.

DeepLabV3 with a ResNet backbone uses convolution in parallel or sequentially to do segmentation at multiple scales. ResNet is used as a features extractor but it could be substituted with ther different backbones.

The metrics they used to evaluate the performance are recognition accuracy, computational complexity, model complexity and inference time. With regard to the Dataset, data and images from Mars HiRise camera are used, their resolution is 1m/px and they were upsampled to 128x128px. The ground-truth was instead generated from DTMs measuring slope and roughness. The dataset was composed of 1000 normalized DTMs and was then divided into training dataset (800), validation dataset (100) and testing dataset (100). It is also useful to remember the fact that they didn't use any data augmentation. Finally they used the Adam optimizer and set the learning rate to the value of 0.00001.

Training and analysis came up with these results:

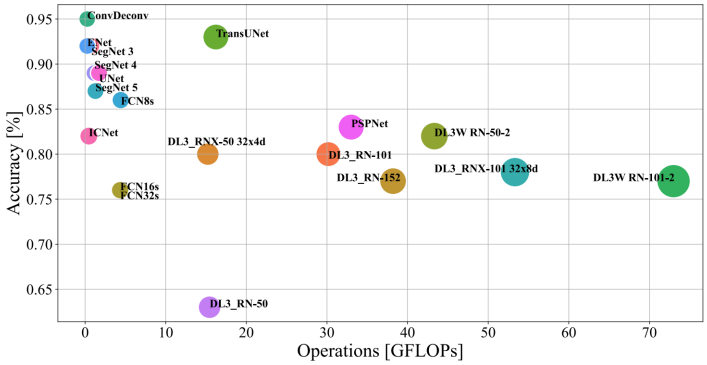


Figure 2.11: s the accuracy of each model with respect to its computational complexity on the X-axis, and its model complexity (number of parameters) as the size of the ball

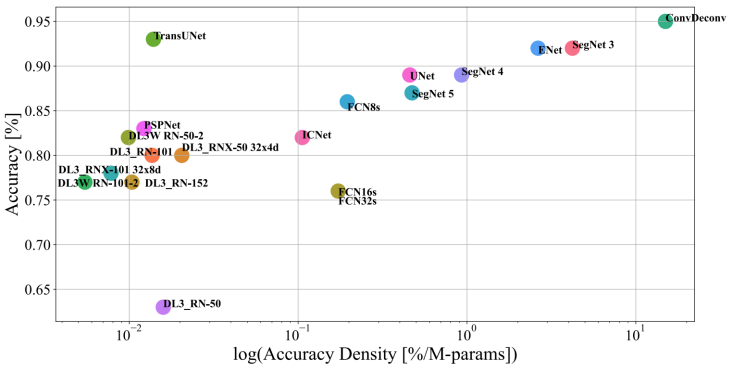


Figure 2.12: The accuracy density by the number of parameters to achieve that result

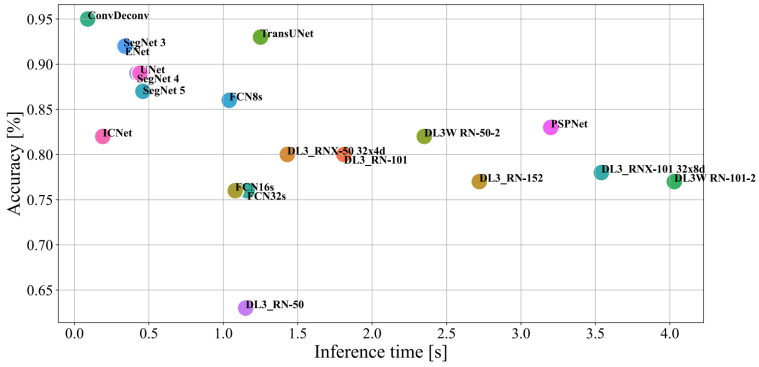


Figure 2.13: Inference time vs accuracy

Looking at the first plot, it is evident that ConvDeconv is the best network, reaching 95% pixel accuracy with the least amount of parameters, indeed to have lots of parameters cannot assure us a better accuracy.

In the second plot instead we consider the accuracy density which is the ratio of the recognition performance by number of parameters. Even in this case ConvDeconv is one of the best, making a better use of the model complexity compared to DeepLabv3s and TransUNet, for example it uses 5 times more efficiently its parameters than SegNet.

In the third plot there is accuracy-rate vs inference time comparison. Again we have that ConDeconv develops the best accuracy with the smallest inference time.

After ascertaining that ConvDeconv gives the best performances we can better visualize its specific training results in terms of loss function, mean intersection over union and pixel accuracy.

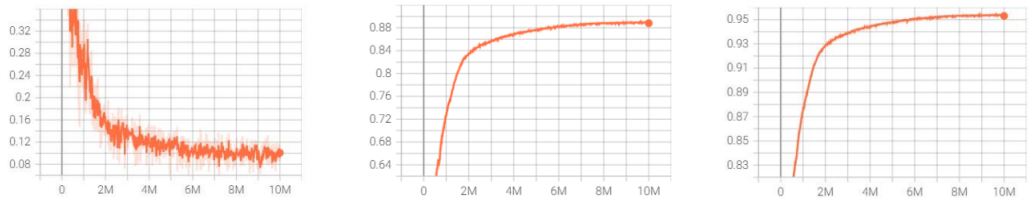


Figure 2.14: Loss Function, mean Intersection over Union, Pixel Accuracy

2.4 Mars with less labels

A semi-supervised learning framework is proposed in this research by Goh et al. [2022] to do segmentation on Mars terrain images captured from rovers. Planetary rovers are nowadays the means by which we as humans are allowed to explore other planets or in general other celestial bodies. As already said, to be able to perform image segmentation is important to improve rovers independence and increase the travel distance and velocity. Indeed, segmentation will help avoid obstacles and estimate traversability. Due to the lack of annotating training data and the fact that the phase of data acquisition is too expansive and consuming, they decided to use and take advantage of transfer learning techniques and a semi-supervised approach.

AI4Mars is the dataset they chose and it is composed of NAVCAM (Curiosity rover's navigation camera) images. The train set is made up of 16064 images, labeled with a modern approach that leverages the work of citizens that volunteer, in particular they identified four classes: soil, big rocks, bed rocks and sand plus a null class, a rover class and a background class. These images and the respective masks have been resized to 512x512 px, while the originals were 1024x1024 px, and they have been also normalized to 0-1 values after being brightened uniformly by 50% (multiplying all pixels by 1,5).

The architecture consists of three components. The first one is a backbone/encoder that is pre-trained with a self supervised approach, more precisely the network used is the ResNet one. The second is a MLP projection head that maps the latent embedding space to the one in which contrastive loss is used during self-supervised pre-training. Finally the third one is another head that can be attached to the encoder or to a middle layer of the second component, and it does supervised fine tuning.

In the self supervised pre training, the network is trained to learn that two augmented version of the same image have similar embeddings and this is done by minimizing this contrastive loss, where sim is the cosine similarity between two vectors, τ is a temperature

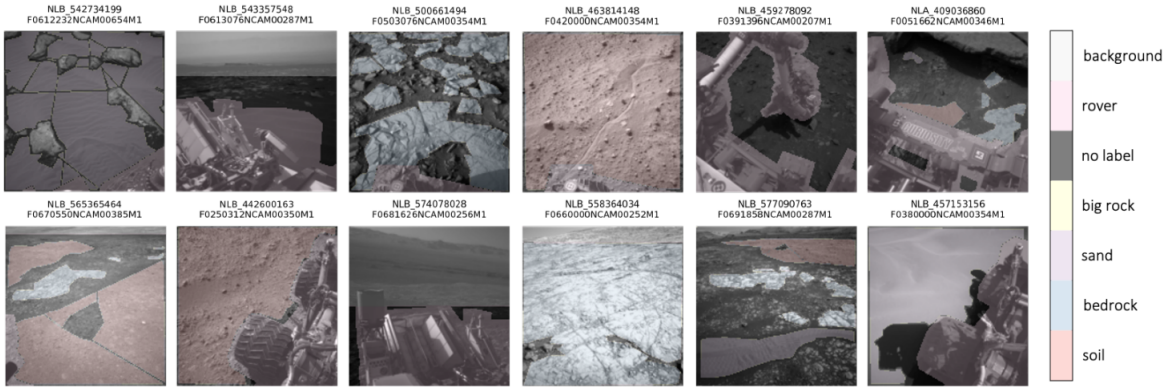


Figure 2.15: Loss Function, mean Intersection over Union, Pixel Accuracy

scalar, and N is the batch number of images (for contrastive learning is preferred to be big). As the network learns to generate similar embeddings from augmented views of the same image and distant embeddings from augmented views of different images, loss decreases. To train on unlabeled data and to use a large batch size means, however, that a large computational capacity is needed.

In the Supervised Finetuning, that happens in the segmentation head attached to the encoder, convolutions with 256 filters are applied. These filters are then passed through a 1x1 convolutional layer that gives us the final logits, but only after being concatenated and resized to the original size.

In this work a pixel-wise cross entropy loss function is also used to ignore unlabeled pixels and the number of epochs for each experiment is 50. It is worth noting that each run needs 4 Nvidia V100 GPUs.

Some visual results are plotted in figure 2.16.

These predicted masks are here compared to the plain supervised model that uses only a very small percentage, 1%, of the training data.

2.5 STEGO

As said before, one of the major issues in semantic segmentation is the difficulties in labeling data, and recently many works introduced the learning from weaker labels such as bounding boxes or classes for example. This work by Hamilton et al. [2022] wants to introduce, instead, the semantic segmentation without any cue or form of supervision.

This is possible using pre-trained features learning frameworks and with the aim of

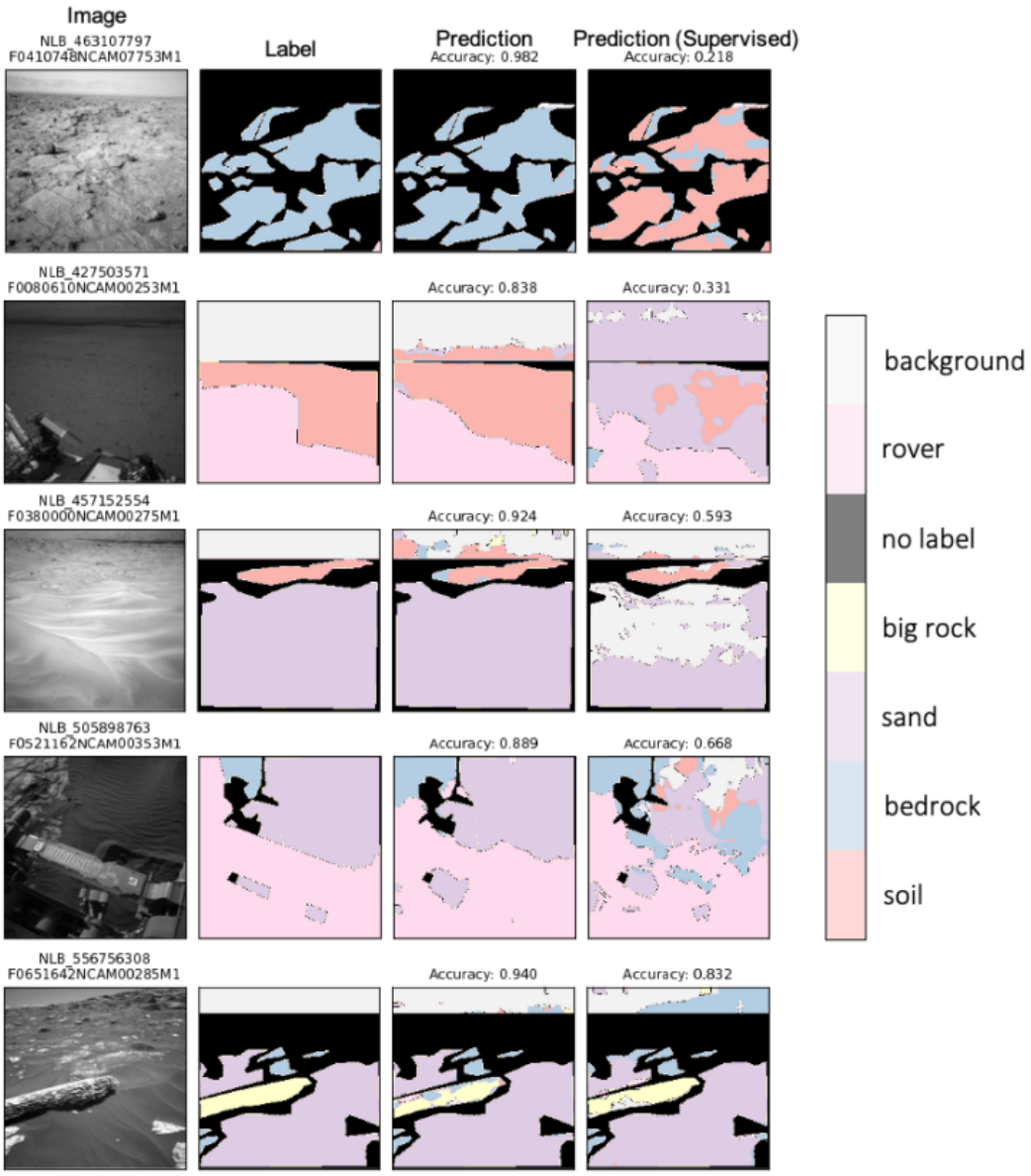


Figure 2.16: Some predictions

distilling them into discrete structures remembering their relations across the image. The name of the proposed head is STEGO and stands for Self-supervised Transformer with Energy-based Graph Optimization.

Moreover, a pre-trained backbone is taken into account, whose name is DINO, to refine its features with STEGO, but it could work with any other deep network architecture. DINO does classification with a Visual Transformer (ViT) but its features, that detects

salient objects, are also meaningful for the extraction of correspondences between images.

To let feature correspondences predict class co-occurrence we focus on intermediate dense features that we know are relevant semantically speaking. Feature correspondence tensor is then created as follow:

$$F_{uwij} = \sum_{n=1} \frac{f_{chw} * g_{cij}}{|f_{hw} * g_{ij}|}.$$

The elements inside the formula are cosine similarities between the feature at spatial position *(h,w)* of tensor f and position *(i,j)* of tensor g. If the similarity is computed between two regions of the same image, there will be $f = g$. In this way, it is possible to see how 2 images are correlated as shown in the example below where there is also a comparison with the K-nearest neighbors (KNN). KNN is an algorithm that determines the classification of a point combining classes of the K-nearest points, it computes the likelihood of that point to belong to a class and sent it to the most common class between its neighbors.

This feature correspondence tensor is also linked to the true label co-occurence tensor

$$L_{hwij} = \begin{cases} 1, & \text{if } l_{hw} = k_{ij} \\ 0, & \text{if } l_{hw} \neq k_{ij} \end{cases}.$$

where k and l are ground-truth semantic segmentation labels, this means that I can compute L only if I have labels available. It came up that DINO made already very good predictions of label co-occurrence even if it never saw the labels.

Then a loss function needs to be built, its aim is pushing the items of s and t together if there is a pairing between two f and g corresponding items. Where we consider f and g two feature tensors form a pair of images and s and t are their corresponding segmentation features. Unfortunately balancing small object learning signals is not easy, for this reason a Spatial Centering operation and zero clamping are added.

In the end the loss function is as follows:

$$\mathcal{L}_{corr}(x, y, b) = -\sum_{hwij} (F^{SC}_{hwij} - b) max(S_{hwij},0).$$

STEGO is evaluated on CocoStuff and Cityscapes datasets, both containing 27 classes. The images were resized to 320x320 px doing a simple center crop and the metrics used were mIoU and Accuracy. At last a comparison based on the Potsdam-3 dataset was added.

MODEL	UNSUP. ACCURACY	UNSUP. MIOU
ResNet50	24.6	8.9
MoCoV2	25.2	10.4
DINO	30.5	9.6
DeepCLuster	19.9	-
SIFT	20.2	-
Doersch et al.	23.1	-
Isola et al.	24.3	-
AC	30.8	-
InMARS	31.0	-
IIC	21.8	6.7
MDC	32.2	9.8
PiCIE	48.1	13.8
PiCIE+H	50.0	14.4
STEGO	56.9	28.2

Table 2.1: Comparison of unsupervised segmentation architectures on 27 class CocoStuff validation set

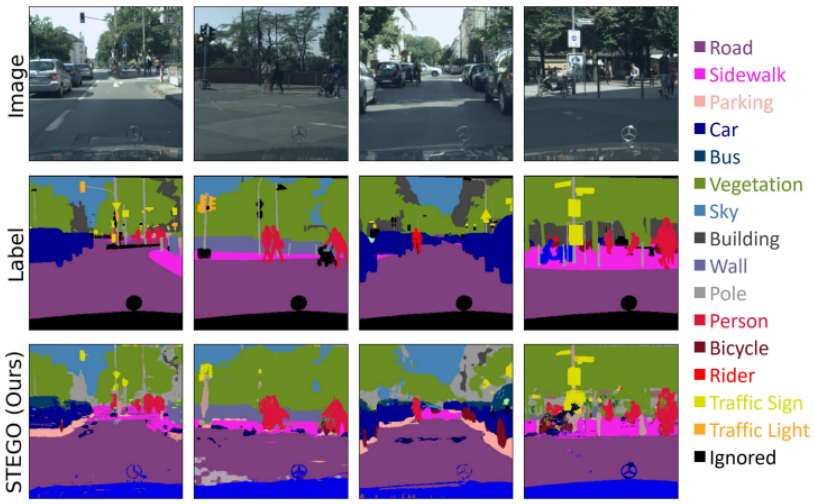


Figure 2.17: Comparison of ground truth labels (middle row) and cluster probe predictions for STEGO (bottom row) for images from the Cityscapes dataset.

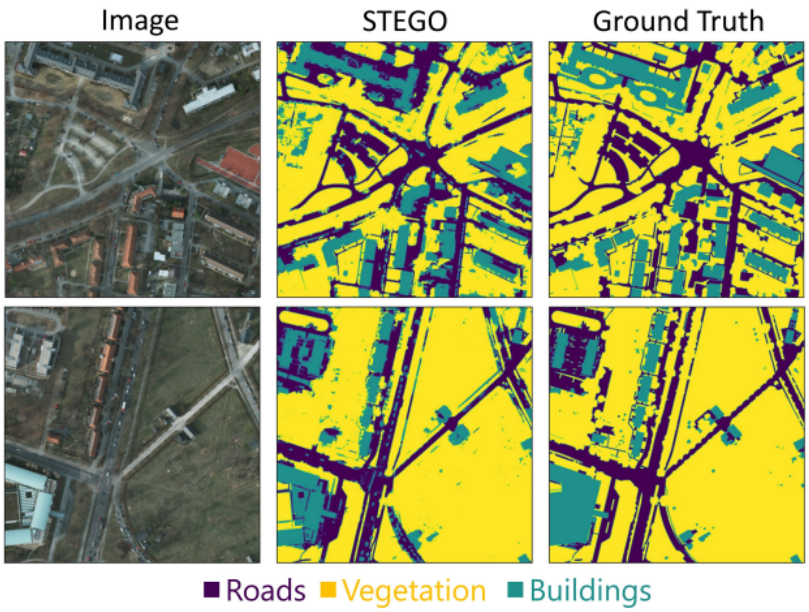


Figure 2.18: Qualitative comparison of STEGO segmentation results on the Potsdam-3 segmentation challenge.

Chapter 3

Deep Learning for image segmentation

To better understand how the previous studies reached those results and which are the tools we chose for our work, we are going to deepen some Deep Learning techniques and Neural Networks architectures. We are starting again from the supervised architectures going then through the unsupervised one, trying to explain with details each of the methods adopted. The next sections we are now going through are the following:

1. Convolutional Neural Networks
2. Specific case study: ConvDeconv architecture
3. Visual Transformers (ViT)
4. Specific case study: DINO+STEGO

3.1 Convolutional Neural Networks

Convolutional Neural Networks base their actions in the simple process of convolutions. A convolution is how a filter modifies an input, moving across it from left to right and from up to bottom. Taking into account images, both 2D (gray scale) and 3D (RGB), that are actually matrices in which each element is a pixel with values from 0 to 255, we can show some examples of how a convolution works.

In a 2D case, the filter slides along the input and each region it is passing through will be multiplied to the filter itself, the result of this operation is then saved in the output, like this image is showing here below. It is to notice the fact that the output is typically smaller in size than the input.

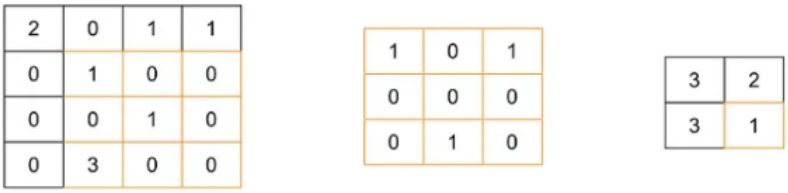
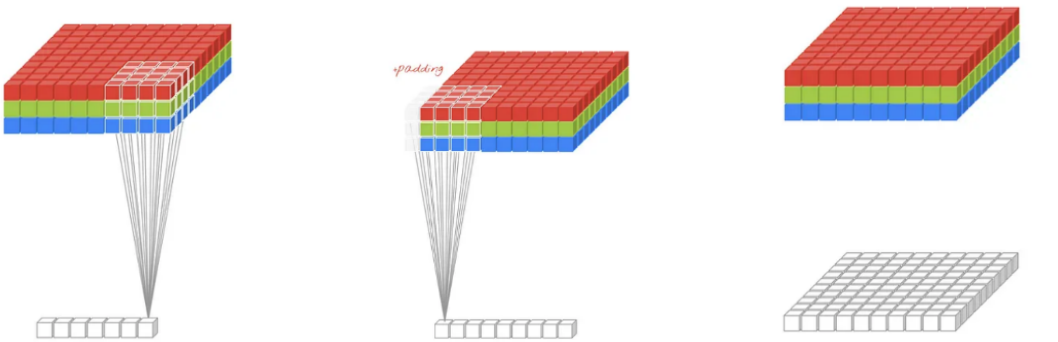


Figure 3.1: Input - Filter - Output.

After that, we can focus on the 3D case study, in which the procedure is the same but with an input and a filter with more than one channel. Even in this case the output is normally smaller than the input.

If we don't want the output to be smaller compared to the input, what we should do is introduce padding. Padding is an external frame of zeros pixel we can add to our input to increase the size of our output. Moreover, if we use more than one filter we will have more than one channel in the output, and exactly one per every single filter we are using. Each of these output channels are the features map, i.e. an evidence of the presence or absence of the specific feature we are investigating. Each convolution can be considered as a layer of the network.



(a) 3D convolution (b) Padding (c) 3D convolution output

Figure 3.2

The pooling is another step in a common CNN architecture. Indeed, it is used to

decrease the width and the height. Max pooling for example reduces the size of a region by taking only the maximum value in that region. The average-pooling layer is instead a variation of the max-pooling layer; it differs in the fact that it takes the average of the values in that region rather than the maximum value.

The flatten layer and the dense layer are other types of layers commonly used in building a CNN. The former, as already told by its name, is used to transform our data into a 1-dimensional array whilst the latter is a layer in which each neuron of the previous layers gives input to neurons of that layer. The aim of the dense layer is to classify the image based on convolutions output. Additionally the dense layer can only receive 1d array as input and that is why it is always coupled with the flatten one.

We can then divide the layers of a network in three types: the input layer, the hidden layers and finally the output layer. The input layer takes raw data directly from the domain, without doing any kind of computation but only passing on data; the output layer is the one that actually makes predictions and the hidden layers are all the layers between input and output.

Neural Networks require activation functions, that are the ones that define the output of a neuron given one or more inputs and most importantly, it adds non linearity to the model. We want non linearity particularly to be able to manage and execute complex tasks. All the hidden layers usually operate with the same activation function that should be different from the one used by the output layer. Many types of activation functions exist and they are often differentiated according to the type of layer they are working with. There exist three types of activation functions: binary, linear and nonlinear. Binary is the simplest, indeed the activation function is compared to a threshold and then if the input overcomes the threshold the neuron will activate, otherwise it will stay deactivated. On the other hand we have the linear one which is also known as the no-activation, in fact it is proportional to the input. The drawback is that all the layers will break down into only one because the last layer will still be linear compared to the first one. With respect to nonlinear activation functions, the most known are: Sigmoid, Softmax, Tanh e ReLu.

Sigmoid takes as input any real value and then gives as output a value in the range (0,1). The more positive my input is, the closer the output will be to 1 and on the contrary, the more negative the input, the closer to 0 will be the output. Mathematically speaking we can write it like this:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Thanks to its output range from 0 to 1, it perfectly fits the needs of a model that predicts probabilities. The drawbacks are that it makes the training of the neural networks unstable and difficult and it has a valuable gradient only between -3 and 3 values.

Softmax is equivalent to the union of more sigmoids, it also returns probability for each class. It is mostly used for output layers and multi-class classification.

Tanh activation function is similar to sigmoid but with output values between -1 and 1. It also has similar problems for vanishing gradients. The mathematical formulation could be

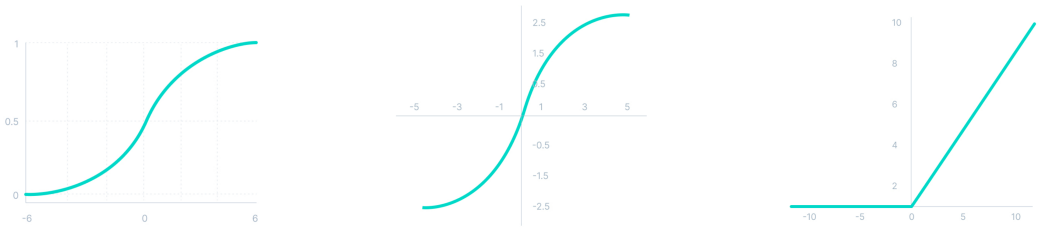
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

We can take advantage of Tanh activation function thanks to the fact that it is Zero-Centered and this helps mapping the values as strongly negatives, neutrals or strongly positive (this was not possible with sigmoid whose output values were only in the range (0,1)).

ReLU activation function has the peculiarity of not activating all the neurons at the same time. ReLU means Rectified Linear Unit and it activates the neurons only if the output value is greater than 0. The mathematical expression of ReLU is

$$f(x) = max(0, x).$$

The point that ReLU only switches on some neurons, led it to be the most computationally efficient among the other activation functions here represented. On the other hand the drawback is due to the zero gradient in the case of outputs with value less than 0. Thereby the process of backpropagation, i.e. when weights and parameters of the neural networks are updated, is impossible with the risk of having neurons never activated too. ReLU is also the most used activation function for hidden layers of a Convolutional Neural Network.



(a) Sigmoid (b) Tanh (c) ReLU

Figure 3.3

Generally the neural networks are trained thanks to an optimizer, the most common

is the stochastic gradient descent algorithm. It estimates the error gradient and then updates all the weights using back-propagation of the error algorithm, better known as back-propagation. But to know how much these weights are updated we need to look for the value of the learning rate. Learning rate is usually defined in the range (0,1) and actually measures how quickly our model adapts to the problem. So the bigger the learning rate, the more rapid the changes will be. However a learning rate too large can cause some drawbacks, e.g. suboptimal solutions, and on the other hand, a learning rate too small may result in the process being stuck.

This is why learning rate is one of the most important hyperparameters to tune.

Finally we define what is the meaning of batch, number of iterations and epochs, which are parameters that define how the training is performed. An epoch is a back and forth passage on the entire dataset, this is not possible considering at a single time in only one step all the images, indeed they are fed to the network in smaller groups, whose size is defined by the batch size, in this way iterations are the number of batch needed to complete a single epoch.

3.2 Specific application: ConvDeconv Architecture

ConvDeconv is the architecture that won each benchmark analysis within the case study of section 2.3. As already said it was developed in 2017 to be applied in image processing tasks, more specifically for semantic segmentation tasks and it was written using PyTorch library. This model is a simplified version of SegNet and it is actually composed of a symmetric structure made up of two neural networks that we can call as an encoder and a decoder, as many other known network architectures do. At the same time though it is much more simple than SegNet.

The two neural networks usually use the same structure, but the first works in the conventional way whilst the second in a reverse way. In the encoder of ConvDeconv the input passes through, in order, a convolution layer with three channels in input and sixteen in output, a pooling layer with a kernel equal to two, a second convolutional layer with sixteen channel in input and thirtytwo in output, a second pooling layer similar to the previous one and then again a convolutional layer with sixtyfour channels in output. Following the symmetry, in the decoder we have three deconvolutional layers alternating with two max unpooling layers. The activation function is the ReLu function we just explained in the previous chapter, and it is always the same, even in the last layer.

Moreover, the Adam algorithm is the optimizer chosen for the network. The optimizer is the algorithm that decides in which way to update the weights of the network during the training, based on its data, as we already saw previously. In the classic stochastic algorithm the learning rate has always a constant value between all the weights, and doesn't change during the training, while Adam optimizer uses the squared gradients to scale the learning rate. The initial learning rate was 0,001.

Finally the batch size is equal to ten, the fixed number of iteration is 10000 and the number of epochs is calculated as follow:

$$n_{epochs} = \frac{n_{iter}}{\frac{len(traindataset)}{batchsize}}.$$

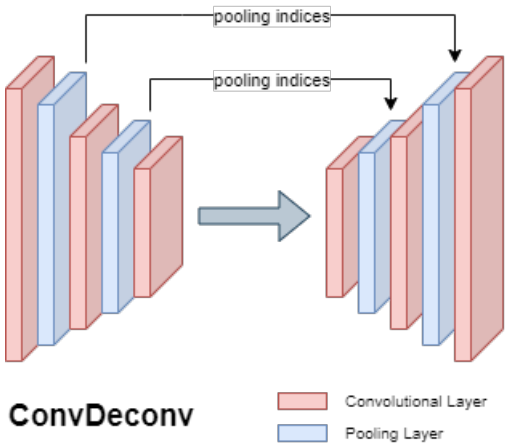


Figure 3.4: ConvDeconv architecture

3.3 Vision Transformer - ViT

Vision Transformer (ViT) architecture comes from the concept of Transformer that was actually born for Natural Language Processing (NLP) to comprehend the content of the text and do meaningful deductions from it. Transformers for NLP firstly split the text in words or groups of words called tokens and then convert these tokens into encoded vectors to feed them into an encoder. In the encoder each token receives by the *Multi-Layer Self-Attention Network (MSP)* a weight that corresponds to the importance they have in the sentence, that is in fact how they relate with other tokens. After that, another network, the *Multi-Layer Perceptron*, encodes the output from the previous one. It is possible to have more blocks of MSP and MLP and then finally an MLP head layer is added outside the encoder. This is the layer that provides the final logits that according to need might be converted to probabilities applying an activation function, e.g. softmax.

It should be noted that this network is essentially a generic network, in fact the output of the encoder was not developed for a specific task, but at the same time, applying after it a specific head it could adapt to many different applications. This characteristic lets Transformers be really useful in the task of transfer learning, by which we mean the reuse of an already trained model, developed for a specific task as the starting point for a second model whose objective is another different task.

Going back to Vision Transformer, they fundamentally use the same approach with some cautions and adjustments due to the different nature of the input, that in this case is an image with at least two-dimensions and not a sentence. First of all it is useful to make the image shape resemble as closely as possible the shape of the phrase. To do that the image is divided in smaller patches in a quantity equal to $N = \frac{H*W}{P^2}$ where H and W are the height and the width of the initial image and (P,P) is the resolution of one of the final patches. After that and before entering the decoder each patch is also flattened to a vector of length $P^2 * C$ where C is the number of channels of the image (i.e. three if it is an RGB image, one if it is a grayscale image etc.). Then using a trainable linear projection, the flattened patches are mapped and before them a Learnable Class Embedding is prepended, to predict the input image class. Finally a positional embedding is added to patch embeddings to add positional information to the input. From this moment on the procedure beginning with the entrance in the Transformer encoder is the same as for NLP.

Usually ViTs are pre-trained on larger and well known dataset, for example one could be ImageNet, and only after that they will be fine-tuned on smaller dataset; this is exactly

the meaning of *transfer learning* we name previously.

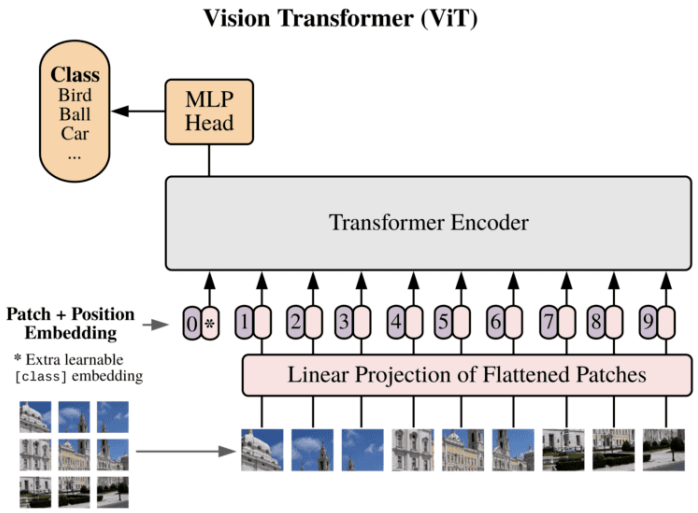


Figure 3.5: "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale"

3.4 Specific application: Dino + Stego

Dino is a self-supervised system, developed by Meta AI, which makes use of self-distillation, that is also from where it took its name: self-DIstillation with NO labels. Self-distillation is a method that creates two networks, we are going to call them student network and teacher network, but they both share the same architecture.

The image we are giving the network as input is divided in patches, one patch is fed to the student and another one to the teacher. The difference between the two stays in the outputs, indeed teacher network output receives a centering operation on it that student network output doesn't have. After that both of the outputs are normalized with the use of softmax function and consequently are fed into the cross-entropy loss function. At first only student network weights are updated so that teacher weights can be updated using exponential moving average (EMA) on students weights. Where with EMA, we mean an average whose weights and significance are greater if the data points are the most recent ones.

The architecture used by the student and the teacher network is not fixed, it could be either a ViT or ConvNet, but it is useful to underline that DINO reaches the best results with the application of ViT and particularly of ViT-B/8. It also works really well for

classification, reaching high accuracy for self supervised architectures, with values that reach in the best case 80.1%.

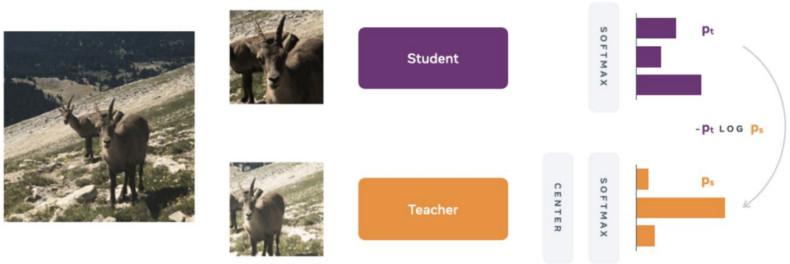


Figure 3.6: "Emerging Properties in Self-Supervised Vision Transformers"

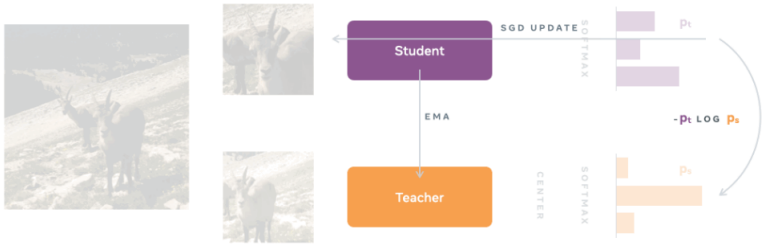


Figure 3.7: "Emerging Properties in Self-Supervised Vision Transformers"

To improve the classification results obtained by DINO and trying to move the task to image segmentation, a STEGO head can be added.

Going into detail, STEGO objective is to distill feature relationships between the input image and its K-nearest neighbors, other images and even itself. It can do this instantiating for times the contrastive loss function described in section 2.5 and shown here:

$$\mathcal{L}_{corr}(x, y, b) = -\sum_{hwij}(F^{SC}_{hwij} - b)max(S_{hwij},0).$$

Stego uses a backbone "frozen", i.e. without any type of fine tuning, and this is the input of the segmentation head with which it predicts distilled features. First of all some global image features are extracted using global average pooling (GAP). Then each batch of the training is made up of random images and random nearest neighbors checking, however that no image matches itself. The output are feature correspondences but they are actually handled like they were probabilities. Doing so is useful to verify the efficaciousness of features and at the same time to compare it with other kinds of networks that use probability logits.

After the segmentation head, a cluster and a CRF are applied too. CRF is a model that fits perfectly with applications in which predictions are influenced by information and state of the neighbors.

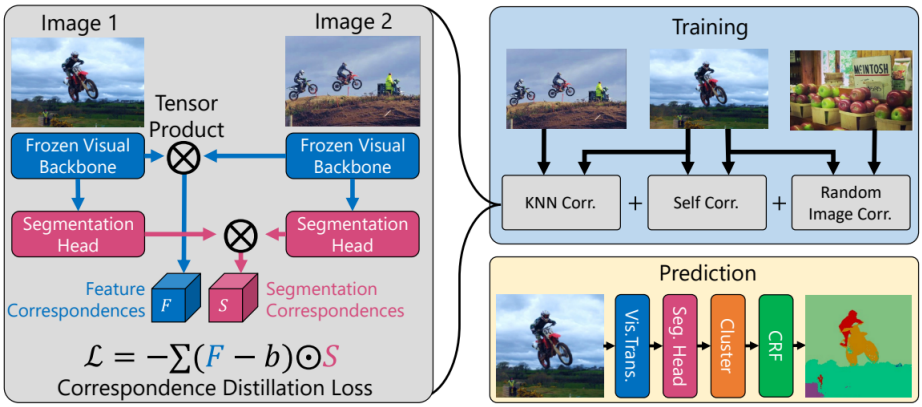


Figure 3.8: "STEGO architeture"

Chapter 4

Code implementation and Dataset preparation

In this chapter we are going to present the choices made in terms of dataset and coding, which are the steps that led us in the selection of a type of data instead of others and the reasons that guide me in correcting and adapting to our needs some neural networks architectures.

4.1 Dataset

Even if in these last decades the number of experiments and missions on Mars increased, there is an evident problem in the collections of useful data for artificial neural networks purposes, particularly for the case of satellite images. The only free source available is, indeed, the one supplied by High Resolution Imaging Science Experiment (HiRise) that is a camera on board of the Mars Reconnaissance Orbiter that can also reach 30 centimeter per pixel resolution. Its raw data, the Experiment Data Records (EDRs), are in 28 channels for each image, but only users with a high level of specialization will have the need to access them. On the other hand we have RDRs, Reduced Data Records, that are radiometrically calibrated, geometrically mapped images saved in a .JP2 extension not to lose any data.

4.1.1 HiRise DTM

At the initial stages of this thesis study, during which we were investigating the hypothesis of developing a supervised neural network model, we took into consideration the use of DTM products. In fact these are one of the most used products from HiRise thanks to the fact that they contain inside themselves also information about altitude. But starting from the beginning we firstly want to clarify what a DTM is. DTM stands for Digital Terrain Models and together with Digital Elevation Models (DEM) and Digital Surface Models, it is a way to model elevation.

A Digital Elevation Model referred to Earth is a raster grid of our planet surface with respect to a *surface datum*, i.e. a surface commonly considered of zero elevation from the scientific and academic world. Moreover, a DEM data file's information becomes more detailed the smaller the grid cells are. In general, this type of data is generated thanks to acquisitions by satellites, drones etc.

A DEM model can be divided and segmented into DSM and DTM. The former captures both natural and human-made structures, for example trees and buildings, while the latter represents the evolution of the geodesic surface.

If we can already confuse DTMs with DEMs in the Earth study thanks to its definition, this is even more justified taking into consideration Mars terrain studies, where vegetation and human buildings are maybe no more, or not yet present. This is also why from now on we will use indistinctly DTM and DEM to talk about HiRISE DTM products.

HiRISE DTMs are created using two images taken from different angulations of the same area on mars surface, these images are called stereo-pairs, and passed through a long and difficult process to create the elevation model. The resolution of DTMs is around 1-2 meters per pixel. After this process it is also possible to develop products such as orthoimages, which are images in which pixels have been projected in such a way that it seems like you are looking directly down at the terrain.

DTM product respect the standard PDS (Planetary Data System) image object format .IMG and contains also some metadata with informations about:

- File format and length
- Identification Information
- Valid minimum and valid maximum
- Map projection information to associate pixel to latitude and longitude

To navigate inside these files a specific library is needed: Rasterio, it indeed allows us to convert spatial location to pixel location and easily browse inside this PDS image that is actually a matrix in which each element/pixel contains the altitude data with respect to a zero level reference surface.

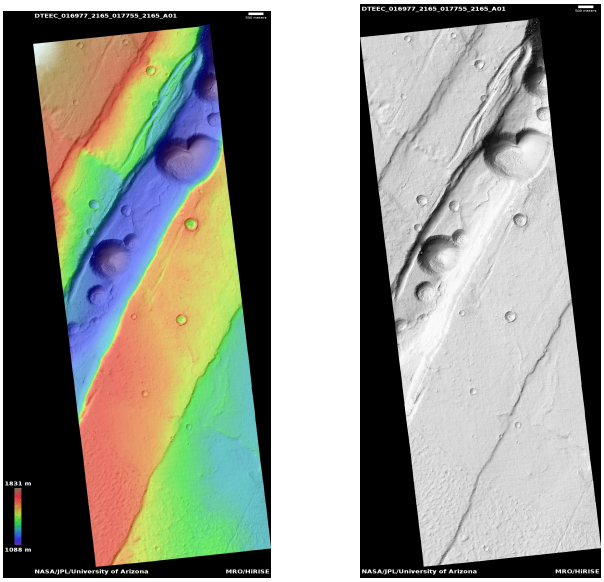
In addition to the effective DTM, we have many other products associated to it, starting with many version of the orthoimages (same resolution of DTM in JP2 format, same resolution of original image in JP2 format, both with version in 1 RED band and 3 IR, RED, BG band), up to some DTM extras, .jpg browsable. These .jpg products are grayscale images, lower resolutions orthoimages, colorized altimetry pictures and a shaded relief version of the image, i.e. a map where region with no or very few features are smooth and on the other hand represents slopes and mountain as they were defined by a light source chosen by the user.

These products could be used for this study as a trainable dataset and as the ground truth to which the network should refer during the training process. Adapting the product to be used as a ground truth was however time consuming and long processing and this was one of the reasons that brought us to the choice of switching to an unsupervised architecture as I am going to explain later, in next chapters.

Figure 4.1: “HiRISE DTMs interface and products“

4.1.2 HiRise realeses

In addition to the DTM product, HiRISE offers simple RDRs and its correlated extra products. The extension of the RDR file is .JP2 in this case too and similar to DTMs it



(a) Colorized altimetry (b) Shaded Relief

Figure 4.2: DTMs extra products

has a label attached with information regarding map projections and viewing information. These files are generally very large in dimensions, with a size close to 1 GB. That is why some extra products are generated with reduced scale and a more compressed format, to be more easily controllable and usable.

Extra products are available in .JP2 if the image is at its full spatial scale and in .jpeg if we are dealing with a browser image, that is a reduced scale one. In addition to this we can also find a map-projected and a nomap-projected which are recognizable looking at the presence or absence of "NOMAP" in the name. The no-map products have the same geometry as raw images and due to the characteristics of the Mars Reconnaissance Orbiter orbit, the majority of the images have the north pointing down.

To better examine and take into consideration those images it is useful to better look at the colors represented in these. Even if the color band is adaptable to each image to better underline the contrast, some general consideration can be made. For example it is to note that dust, that is the reddest material, looks like red in RGB but yellow in IRB. Rocks and coarse material are typically blue, and at the same time ice and frost are blue too, but with a bright nuance. Even if sometimes it could be difficult to read and interpret the colors, these images are pretty helpful to avoid ambiguities present in the grayscale images, indeed in color images different materials should also have different colors.

Once this dataset with all its products has been analyzed and the DTM product was already discarded, we focused our attention on the extra products. In particular, as first step we decided not to consider map-projections images that however are important for future Mars referentiations, but at that moment were unnecessary considering our goals and the necessity of having images without any form of added black frame. In addition to this, the .jpeg is a file format much more easier than .JP2 to manipulate and finally RGB was the color model we had already the chance to work with. All these reasons led us to the choice of a product of the type of figure 4.3.

4.2 Method and code implementation

The first stage was focused on finding the best supervised architecture with the following characteristics, looking at the scholarly literature at our disposal: it should reach good results in an interplanetary environment, it should have applications on satellite images and finally it should perform tasks with semantic segmentation. For these reasons all the instance segmentation and classification applications were excluded. Going back to the benchmark analysis faced on chapter 2.3, it came up that ConvDeconv network met all of our requirements. Therefore in the next section we are going to frame all the parameters and fundamental characteristics we change to adapt the network to this thesis work.

During this first code development we carried on the study of the Dataset already explained in previous chapters. First of all, DTM needed many preprocessing steps to be used as a ground truth or just as a training set. Indeed to use it as a valuable data it should represent not the altitude itself, which alone brings little useful information for the rover movement and path, but the slope or the roughness of the terrain that represent much better its traversability. Thus, many more computations were needed.

For the reasons explained above the DTMs were discarded in favor of HiRISE extra releases in NOMAP-RGB.jpg format, that were reliable as well and at the same time more accessible in terms of manipulation.

This decision involved also an in-depth analysis of some labeling tools available on the web. Summarizing their main features:

- *Labelbox*: fully manual labeling, no tools that follow boundaries (e.g. Lazo in Adobe); the export file could be an annotation object or a Json file.
- *Studiolabel*: fully manual labeling but also a polygonal border tool and a magic wand tool; the file can be exported with many different file formats, among them Json,

Json-min, CSV, TSV, and, with a premium version, png image and numpy array too.

Unfortunately, even with studio label, which we chose as the tool that best fitted our objectives, the time needed just to label just one NOMAP-RGB.jpg image was about 1 hour. This time was not reasonable compared to our timeline. In addition to this it was pretty difficult to reach the adequate level of detail, even with the aid of these tools, and it is evident that for this kind of works an expert in this field, planetary terrain morphology for example, is strongly recommended. To distinguish only by eye, and in particular unexeperties eyes, would have drove us to mistakes and inaccuracy. This awarenesses led us to change roadmap and head to an unsupervised network.

Once we studied the state-of-the-art for selfsupervised and unsupervised networks, we focused on two architectures we found were relevant for this field and which studies we already presented in chapter 2, section 2.4 and 2.5. The strength of the former was in its native interplanetary application, and more specifically its Mars terrain application, whilst the latter with its achievements was actually the state-of-the-art for total unsupervised segmentation. At the same time the images segmented within the first work were only rover camera images from AI4MARS dataset, which are slightly different from our satellite images, and the computational power required was equivalent to the use of four GPUs. On the other hand STEGO training lasted only two hours on a single GPU and even if it has never been trained on Mars images, it was tested on Earth satellite images from the Potsdam dataset.

Having established this and taking into account that our availability in terms of computational power were restricted on the GPUs provided by Google Colab and eventually on a local 16 Gb GPU, the choice fell on STEGO architecture backed by DINO architecture. Moreover STEGO is the state-of-the-art for what concerns total unsupervised segmentation with very good results with dataset such as COCOStuff, Cityscapes and even with Potsdam, a Hearth Satellite Images dataset. This was then an attempt to verify the worthiness of this method on data such as ours.

We are going now into details on how we first tailored the supervised network, ConvDeconv, and then the total unsupervised one, STEGO, to our needs.

4.2.1 ConDeconv adaptation

ConvDeconv source is available and accessible to all users at Garg and Jain [2022] and it is written in Python with the use of the PyTorch library. Due to company and project

needs the first step was to translate the code to a Tensorflow/Keras written code trying to keep the neural network consistent in its evolution.

The structure of the network is defined in a specific class where parameters for convolutions and pooling are specified. The adjustments made were mainly about the last layer, which of course need to be adapted to the task, indeed the number of this layer filters was modified to be equal to the number of classes to look for in the segmentation. Moreover, its activation function was changed moving from ReLU to Softmax, which is perfect to output probabilities and especially for multiclass classification. In this way the output is a matrix with as many channels as the number of classes we want and with a value in each element that represents the probability of that specific pixel to be assigned to the corresponding class of the channel we are looking into.

To define the convolution, deconvolution and pooling layers we relied on functions already existing in Tensorflow and Keras, while for the unpooling a specific function needed to be defined. The code for the Network Class and for the Unpooling Function is here reported.

```
class conv_deconv():
  def __init__(self):
    #ENCODER
    #conv1
    self.conv1=keras.layers.Conv2D(filters=16, kernel_size=4, strides=1,
     activation='relu')
    #conv2
    self.conv2=keras.layers.Conv2D(filters=32, kernel_size=5, strides=1,
     activation='relu')
    #conv3
    self.conv3=keras.layers.Conv2D(filters=64, kernel_size=3, strides=1,
     activation='relu')

    #DECODER
    #deconv1
    self.deconv1=keras.layers.Conv2DTranspose(filters=32, kernel_size=3,
     activation='relu')
    #deconv2
    self.deconv2=keras.layers.Conv2DTranspose(filters=16, kernel_size=5,
     activation='relu')
    #deconv3
    self.deconv3=keras.layers.Conv2DTranspose(filters=24, kernel_size=4,
     activation='softmax')
```

```
18
19  #POOLING AND UNPOOLING LAYERS
20  #maxpool1
21  def maxpool1_indices(self, input_):
22    self.maxpool1_value, self.maxpool1_index=tf.nn.max_pool_with_argmax(
    input_, ksize=2, strides= 2, padding= 'VALID')
23    return self.maxpool1_value
24  #maxpool2
25  def maxpool2_indices(self, input_):
26    self.maxpool2_value, self.maxpool2_index=tf.nn.max_pool_with_argmax(
    input_, ksize=2, strides=2, padding='VALID')
27    return self.maxpool2_value
28  #maxunpool1
29  def maxunpool1_(self, value, index, shape2, input_):
30    self.maxunpool1= up_sampling(value, index, shape2, 10, name= "
    unpool1")#(input_)
31    return self.maxunpool1
32  #maxunpool2
33  def maxunpool2_(self, value,index, shape1, input_):
34    self.maxunpool2= up_sampling(value, index, shape1, 10, name="unpool2
    ")#(input_)
35    return self.maxunpool2
36
37  def call (self,x):
38    out=self.conv1(x)
39    shape1=out.shape
40
41    out=self.maxpool1_indices(out)
42    index1=self.maxpool1_index
43    value1=self.maxpool1_indices(out)
44
45    out=self.conv2(out)
46    shape2=out.shape
47    print(shape2)
48
49    out= self.maxpool2_indices(out)
50    index2=self.maxpool2_index
51    value2= self.maxpool2_indices(out)
52
53    out=self.conv3(out)
54
```

```
    out=self.deconv1(out)

    out=self.maxunpool1_(out,index2,shape2,out)
    out=self.deconv2(out)

    out=self.maxunpool2_(out, index1, shape1, out)
    out=self.deconv3(out)
    model = tf.keras.Model(inputs=x, outputs=out)

    return model
```

Here is instead how the unpooling function is defined.

```
def up_sampling(pool, ind, output_shape, batch_size, name=None):
    """
       Unpooling layer after max_pool_with_argmax.
       Args:
           pool:   max pooled output tensor
           ind:      argmax indices
           ksize:     ksize is the same as for the pool
       Return:
           unpool:    unpooling tensor
           :param batch_size:
    """
    with tf.compat.v1.variable_scope(name):
        pool_ = tf.reshape(pool, [-1])
        batch_range = tf.reshape(tf.range(batch_size, dtype=ind.dtype),
    [tf.shape(pool)[0], 1, 1, 1])
        b = tf.ones_like(ind) * batch_range
        b = tf.reshape(b, [-1, 1])
        ind_ = tf.reshape(ind, [-1, 1])
        ind_ = tf.concat([b, ind_], 1)
        ret = tf.scatter_nd(ind_, pool_, shape=[batch_size, output_shape
    [1] * output_shape[2] * output_shape[3]])
        ret = tf.reshape(ret, [tf.shape(pool)[0], output_shape[1],
    output_shape[2], output_shape[3]])
        print(type(ret))
        return ret
```

For what concerns the training parameters, the chosen batch size is equal to 10 and the total number of iteration is 1000, the number of epochs will then be computed based on them and on the size of the dataset.

Adam was retained as the optimizer but the learning rate was adjusted to be constant at a value of 0.001. To fulfill our objectives the Crossentropy was the chosen loss function and the metrics chosen to be monitored and which it is useful to take track of are Accuracy, Precision, Recall and Intersection over Union (IoU). These metrics can be calculated as follows:

$$Accuracy = \frac{N_{TP} + N_{TN}}{N_{TP} + N_{FP} + N_{TN} + N_{FN}}$$

$$Precision = \frac{N_{TP}}{N_{TP} + N_{FP}}$$

$$Recall = \frac{N_{TP}}{N_{TP} + N_{FN}}$$

$$IoU = \frac{N_{TP}}{N_{FP} + N_{TP} + N_{FN}}.$$

Where TP stands for true positive, that are pixels belonging to a class and predicted in the same class, FP stands for false positive, which are pixels predicted in that class but not actually belonging to it, TN stands for true negative, i.e. pixels not belonging to that class and not predicted in that class and finally FN, false negative, pixels that are not predicted to be in that class but that actually belong to it. All these definitions means that Accuracy is indeed how many predictions are correct above all the predictions, Precision indicates how many positive predictions were correct with respect to all the positive predictions made, Recall represents how many positive predictions were correct compared to all the predictions that should have been positive.

To better visualize these metrics a classification report and a confusion matrix were prepared. The former is a table where it is easier to compare how the metrics values are varying in each class and the latter is useful to evaluate the general performance of the supervised network.

To test this network end-to-end, a terrestrial drone images dataset available on Kaggle.com was used: *Semantic Drone Dataset*. It contains 24 classes for objects, vegetation and people and it allowed monitoring of operation for our updated version of ConvDeconv supervised network.

4.2.2 STEGO adaptation

Once head to an unsupervised model, STEGO in this specific case, it was needed an understanding and mastery of the code. The model is pretty complex and this fact

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

Table 4.1: Confusion matrix example

required a further study of the theories behind it to learn then how to adapt it to our dataset and needs. Firstly, due to the large dimensions of the HiRISE products and not to lose resolution, a crop strategy should be defined. The strategy proposed by STEGO developers was mainly based on the $five - crops$ function from pytorch. In particular from each corner the image is cropped following the dimensions given by the user, for example those could be half of the initial dimensions of the pictures, and then a final crop is applied to the center. This strategy doesn't really fit our study needs, in fact images from HiRISE - NOMAP.RGB.jpg format have 3 channels, a fixed dimension of 512 px and the height variable but at least in the range of 5000-10000 px. For this reason the strategy adopted was to crop images in 512x512 px patches saving information about the number of crops, the column and the row and the source name in a dedicated dictionary, to be able then to reconstruct them. This procedure should be always applied to each image we want to feed to the model, either for the training, the validation and for future predictions. The code of the two functions we developed to crop the images and save the dictionary is reported here.

Crops:

```
def multi_crop2(img, input_dim, patches, filenum, count, filename):
 h,w,c = img.shape
 imgs_crop=[]
 num_row= h//input_dim
 num_col= w//input_dim
```

```
  patches[filenum]={'row': num_row, 'col': num_col, 'count': count, '
    source': filename}

  for i in range(num_row):
    for j in range(num_col):
      imgs_crop.append(img[input_dim*(i):input_dim*(i+1),input_dim*j:
    input_dim*(j+1)])

  return imgs_crop, patches, num_row*num_col+count # new count
```

Saving:

```
def save_crops(img, imgs_crop, input_dim, filenum, newfolder):
  h,w,c = img.shape
  for row in range(int(h/input_dim)):
    for column in range(int(w/input_dim)):
      filename= "{}.jpg".format(str(filenum).zfill(4) + '_' + str(row).
    zfill(4) + '_' + str(column).zfill(4) + '_' + 'val')
      PIL.Image.fromarray(imgs_crop[row*int(w/input_dim)+column]).save(
    os.path.join(newfolder, filename), 'JPEG')
```

After cropping data, a mandatory step is the precomputation of some KNN information; as already explained, those are useful for future comparisons and similarities and needed some changes in the code. In particular the DataLoader had to be modified due to Colab GPU constraints: the batch size was then set to 128, while at the beginning it was equal to 256.

Once we reached the training phase we got to interface with Hydra, a framework born to simplify the management of hyperparameters that takes advantage of configuration files. In this way it is possible to change and tune hyperparameters without modifying the structure or the core of our model. Moreover, the LightningModule from PyTorch Lightning with its characteristic structure, is adopted as the parent class to define our network and is then integrated in the Trainer, from PyTorch Lightning too, to automate our code once more. Within the Lightning Module is indeed present a defined structure which allows the organization of the code in six sections: initialization and set up, training step, validation step, test step, predict step and finally optimizers configuration. However in our model structure the test step and predict step are not integrated in this module but they will be, later separately.

Images pre-processing

Before being fed to the network each of the images, already cropped in 512x512 px patches passes through some more transformations, in particular it is resized to 224x224 px during the training or to 320x320 px if it is during the validation as defined in the configuration file, then it is converted from PIL Image to a torch Tensor and finally it is normalized with mean and standard deviation.

After that we also apply two types of image augmentation, right before loading the dataset, that is actually a ContrastiveSegDataset object, a class whose parent is Dataset from PyTorch, into the DataLoader. The first exploits some geometric transformations and the second one uses photometrics transformations. These transformations are all randomic, this is why we need to save a coefficient, the 'seed', to replicate that exact action in case of need. When we applied geometric transformations we applied random horizontal flips with a 0.5 probability, then a randomic crop of an image with a resize immediately following to come back to the wanted shape. On the other hand, for photometric augmentation we applied some random jitter to the colors, i.e. we randomly changed contrast, saturation, brightness and hue with a probability of 0.3, 0.3, 0.3, 0.1 respectively. Then some images are randomly converted into grayscale images, preserving the initial number of channels, with a probability of 0.2. Finally some Gaussian Blur is applied to some of the images too. All these processing transformations can be resumed in the following scheme.

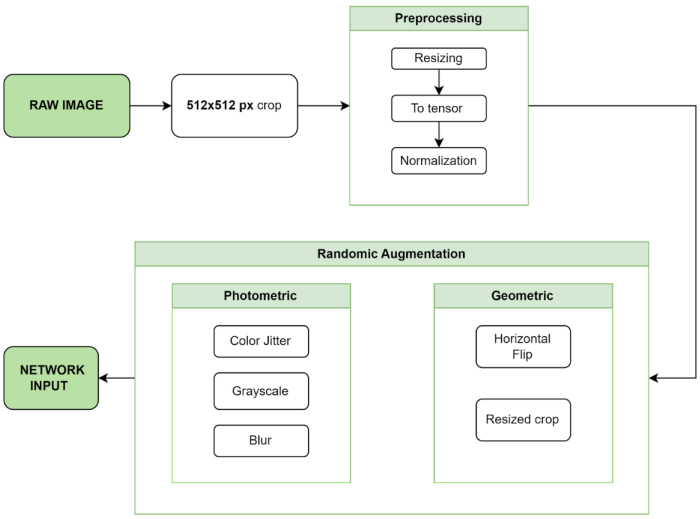


Figure 4.4: Images pre-processing

Training parameters

Once data has been prepared, both for validation and training dataset, the network is finally trained.

Even in this case we sewed the code on our needs, we indeed changed some of the training parameters already defined. First of all we ascertained that 5000 iterations were too few to be setted as the stopping parameter for the training, taking into account the dataset we were using. In fact, the network was completing less than 10 epochs following that criterion. In order to overtake this limit we substituted that constraint with one that set a max number of epochs equal to 60. We also changed the logic behind the saving of the model in the so called checkpoints. From checking each 400 steps and saving the top 2 checkpoints in terms of mIoU, we switched to checking each epoch and saving only the last one. This decision is due to the fact that without having labels no metric can be computed and hence the model has no metric to monitor to decide which is the best checkpoint to save. The point that we do not have any kind of label is also the reason that led us to exclude the validation dataset from the training process at a certain time, even if it is usually included, and that ended in no metric, neither linear and cluster, mIoU or accuracy, computed in the trainer.fit process.

Image Reconstruction

At the end of the training, we evaluate the model feeding it with the validation dataset already prepared, to have at least some visual results. The predictions are by patches, as the dataset we fed to the network. The need was then to reconstruct the initial image. It was possible to do this thanks to the strategy we adopted in renaming the patches once cropped and to a dictionary we created during the initial 512x512 cropping and in which we saved information about the number of columns and rows, i.e. how many patches each image has and how they are located inside it, the total number of cuts done in the whole dataset and the name of the initial source image. Using this information and applying some post processing such as reshaping and permuting, we finally obtained back the original reconstructed image and its respective prediction mask. The code developed to do this is here reported.

```
1 img_num = 137 # image whose mask prediction I want to visualize
2 outputs = {k: torch.cat(v, dim=0)[patches_info[img_num]["count"]:
    patches_info[img_num]["count"] + patches_info[img_num]['row'] *
    patches_info[img_num]['col']] for k, v in outputs.items()}
```

```
3
4  full_image = outputs['img'].reshape(patches_info[img_num]['row'],
       patches_info[img_num]['col'], 3, cfg.val_res, cfg.val_res) \
5      .permute(2, 0, 3, 1, 4) \
6      .reshape(3, cfg.val_res * patches_info[img_num]['row'], cfg.val_res
       * patches_info[img_num]['col'])
7
8  full_cluster_prob = outputs['cluster_prob'].reshape(patches_info[img_num
       ]['row'], patches_info[img_num]['col'], cfg.dir_dataset_n_classes,
       cfg.val_res, cfg.val_res) \
9      .permute(2, 0, 3, 1, 4) \
10     .reshape(cfg.dir_dataset_n_classes, cfg.val_res * patches_info[
       img_num]['row'], cfg.val_res * patches_info[img_num]['col'])
11
12 crf_probs = full_cluster_prob.numpy()
13
14 reshaped_label = outputs['label'].reshape(patches_info[img_num]['row'],
       patches_info[img_num]['col'], 320, 320) \
15     .permute(0, 2, 1, 3) \
16     .reshape(cfg.val_res * patches_info[img_num]['row'], cfg.val_res *
       patches_info[img_num]['col'])
17 reshaped_img = unnorm(full_image).permute(1, 2, 0)
18 reshaped_preds = model.test_cluster_metrics.map_clusters(np.expand_dims(
       crf_probs.argmax(0), 0))
19
20 reshaped_preds=reshaped_preds.type(torch.int64) # float32)
21 reshaped_preds=(reshaped_preds)*255/3 # /2 # /3 #
22 reshaped_preds=reshaped_preds.type(torch.int32)
23
24 reshaped_img=reshaped_img.permute(2, 0, 1)
```

Saving

To save the prediction to our previously chosen directory we take advantage of some existing functions such as PIL.image.save and imageio.imwrite. We use the former to save the image itself while the latter to save the predictions. The choice fell on imageio.imwrite for predictions because it allowed us to keep a one channel, grayscale format image, without adding any other unique values, as was unlikely happening with PIL, that could lead to some misunderstanding in case of reuse of the mask.

End-to-end

Finally an interface for a possible external user is created. It is actually the end-to-end of our study and puts together the preprocessing steps needed on the image to prepare the input, the loading of the best model obtained from a checkpoint, the evaluation of the results and finally the saving and the visualization of them.

It is clear that at the beginning some parameters are required to be added by the user. In particular we are asking for the path of three folders, the first where we can find the image to segment, the second where to save the results and the third which represents where to create the directory required in this form: */datasetname/imgs/val/*. Then we ask for the name of the dataset they are feeding to the network, for the number of classes the network should look for and for the number of the image in their dataset for which they want to visualize the prediction.

This is actually how the process will be implemented inside the generic pipeline of the SINAV project.

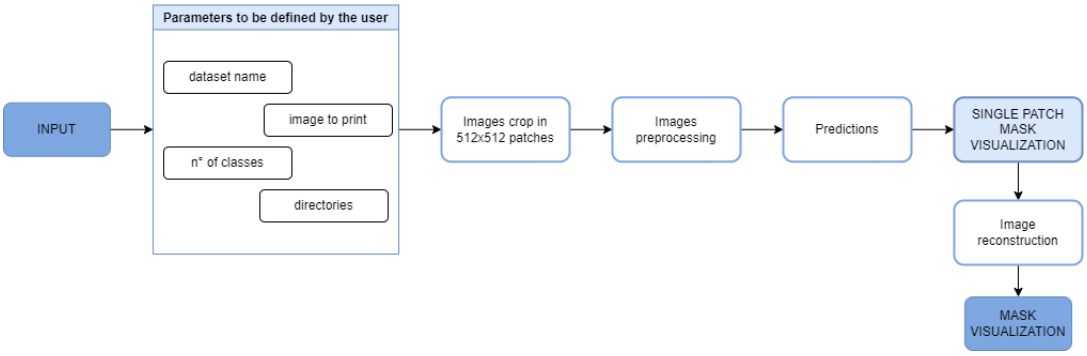


Figure 4.5: "End-to-end schematic process"

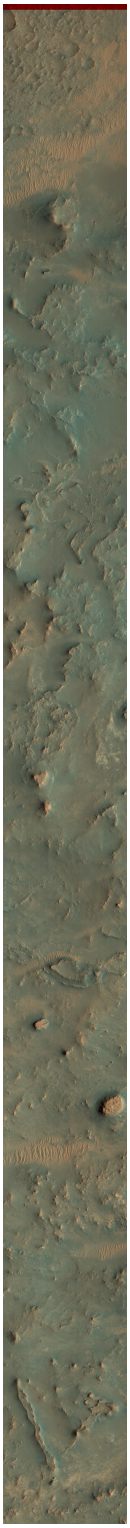


Figure 4.3: “HiRISE RGB - NOMAP product“

Chapter 5

Results

The results we obtained from STEGO predictions on HiRISE images are critical to comment mostly because metrics above predictions from unsupervised semantic segmentation models are pretty difficult to evaluate. This depends on the fact that nearly all the already existing metrics are based on the presence of a Ground Truth to compare with and that the few metrics that follow different procedures are specifically related to clustering models such as K-means.

From a visual examination our results clearly match the purpose of this study, but to find some quantitative measures we needed a step back. First of all we would like to recall the good results presented together with the STEGO code, which we are reporting here, and that was the point that brought us to choose this network, to get then to test and verify this application with a different and unusual set of data. In fact this network shows improvement as well as the Coco and Cityscape dataset, also for Potsdam images, reaching a +12% with respect to the previous state-of-the-art. Secondly we took advantage of a synthetic dataset of a virtual Martian environment obtained by simulation in Unity of a drone camera generated within another Master thesis student project whose usage we are now going to deepen.

5.1 STEGO validation with Syntethic dataset

In this Syntethic Dataset there are 1008 images and they have a resolution of 1000x1000px, in addition they are taken in six different altitudes (5m, 7m, 9m, 11m, 13m, 15m) and four different brightening conditions:

- Zenith
- Low West, 20 degrees over the horizon
- Low East, 20 degrees over the horizon
- High East, 70 degrees over the horizon

Moreover, masks are available and will be used in order to plot some metrics after having trained the network in an unsupervised way with this synthetic dataset. In the GT four distinct classes are present: Soil, BigRock, BedRock and Sand where we can identify BigRocks as the most dangerous for the Rover between all the others.

To test our unsupervised network with this dataset some adjustment were needed both for the data and for the network itself. In particular we did:

1. cropping of the images in 500x500 patches, 2 rows and 2 columns
2. inclusion of the validation dataset during the training
3. addition of the best two checkpoints saving based on mIoU, maximizing it
4. inclusion and application of the Hungarian algorithm to each prediction and before the computation of the metrics.

We tried to find the best configuration for the training, thus we tested different batch sizes from 10 to 32 stopping at this value because of computation constraints. It also happened not to be able to end the training for GPUs limits but thanks to the fact that the checkpoints are saved monitoring a metric, and they were always saved after a few epochs, we considered those results as valuable too. As a matter of fact we ended up choosing as the best model the one who trained with a batch size of 16.

We needed also to check the correlation between the label and the GT itself which had been realized with some post processing on the predicted mask before the computation of the metrics. This step is needed to ensure the correct correspondence between the assigned class inside the prediction and the one present in the ground truth which otherwise was not guaranteed. What we actually were looking for was a pattern in predicted and true label correspondence that could possibly be applied with other dataset too.

We initially started, just to find some confirmations, checking by our eye for some correspondence, but soon we moved on for a more scientific approach. We then decide to verify if each specific class was associated to a characteristic mean value of pixel. To do

that we firstly convert the source images to Grayscale pattern and then we computed the mean of all the pixels in an image, allocated to each class.

What we found is reported in figure 5.1: it can be noted that as expected the curves fluctuates depending on whether the image was captured with the Sun high on the horizon or low over it, in addition means for each of the classes are mostly superposed and this happened due to the fact that the color gradation is well and truly the same or at least very close pixel to pixel. This means unfortunately that it was almost impossible to isolate one single curve into a specific interval of mean values, with that interval different from the other curves' one and to get then to find a characteristic relation.

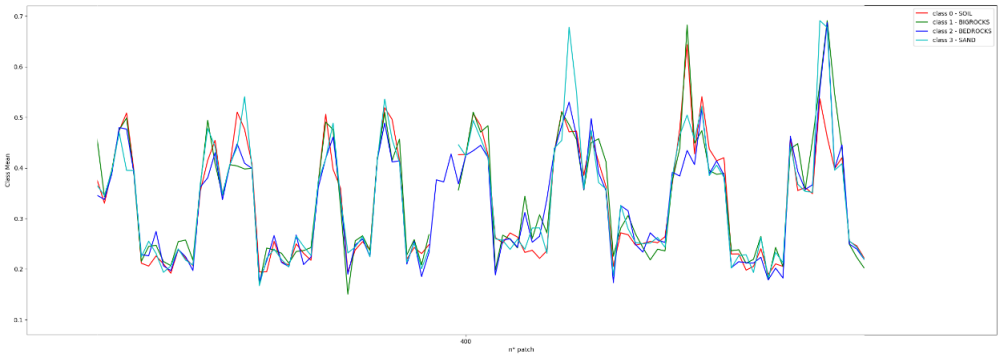


Figure 5.1: Classes mean pixel value

The other try that was made, looking for pattern, was about implementing a form of Hungarian algorithm with which we reassigned a class in the prediction mask based on the one with the highest correspondence in the Ground Truth, to finally check whether or not the network already worked well and then compute again mIoU and accuracy.

Generally, with the presence of the labels it was possible to better valuate the total unsupervised predictions of the STEGO network, working for example with precision, recall, F1, in addition to accuracy and Jaccard index, both mean value and single class. We are reporting here in tables some of the results obtained, even if they are not fulfilling they are for sure the starting point for real dataset evaluation and future steps in the project.

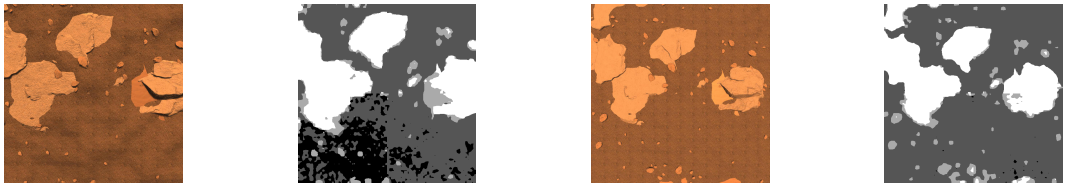
We reached indeed with one of our configurations an accuracy for the BigRocks class, the one we are interested the most in, of 0.45 which was in line and acceptable to come back to test and to evaluate HiRISE images with a higher degree of reliability than before.

Looking at the results on the synthetic dataset we can say it clearly recognized each

METRICS FOR SYNTHETIC DATASET		bat=16,hungin,ep=50		bat=16,hungin,ep=300	
		best clus/mIoU	last epoch	best clus/mIoU	last epoch
Jaccard(IoU)		0.09728	0.09810	0.11936	0.11949
Accuracy	mean	0.24819	0.24790	0.29396	0.29590
	SOIL	0.14720	0.19030	0.23470	0.22910
	BIGROCKS	**0.44970**	0.11850	0.11050	0.11910
	BEDROCKS	0.21790	0.57830	0.62340	0.61100
	SAND	0.17790	0.10450	0.20720	0.22440
Precision	mean	0.25045	0.24839	0.28369	0.28439
	SOIL	0.67890	0.68480	0.76080	0.76120
	BIGROCKS	0.13740	0.12200	0.16490	0.16450
	BEDROCKS	0.12320	0.12160	0.16780	0.17120
	SAND	0.06230	0.06520	0.04130	0.04060
Recall	mean	0.24819	0.24790	0.29396	0.29590
	SOIL	0.14720	0.19030	0.23470	0.22910
	BIGROCKS	0.44970	0.11850	0.11050	0.11910
	BEDROCKS	0.21790	0.57830	0.62340	0.61100
	SAND	0.17790	0.10450	0.20720	0.22440
F1	mean	0.17555	0.17481	0.20609	0.20667
	SOIL	0.24200	0.29780	0.35870	0.35220
	BIGROCKS	0.21050	0.12020	0.13240	0.13820
	BEDROCKS	0.15740	0.20100	0.26450	0.26750
	SAND	0.09230	0.08030	0.06880	0.06880

Table 5.1: Synthetic Dataset metrics and configurations

relevant object in the scene but has some trouble with the distinction between bed rocks and big rocks which usually are assigned both to the big rock category, as we can see in the following pictures.



(a) Source image 119 (b) Predicted mask 119 (c) Source image 120 (d) Predicted mask 120

Figure 5.2: Respective prediction for validation images n° 119 and n° 120

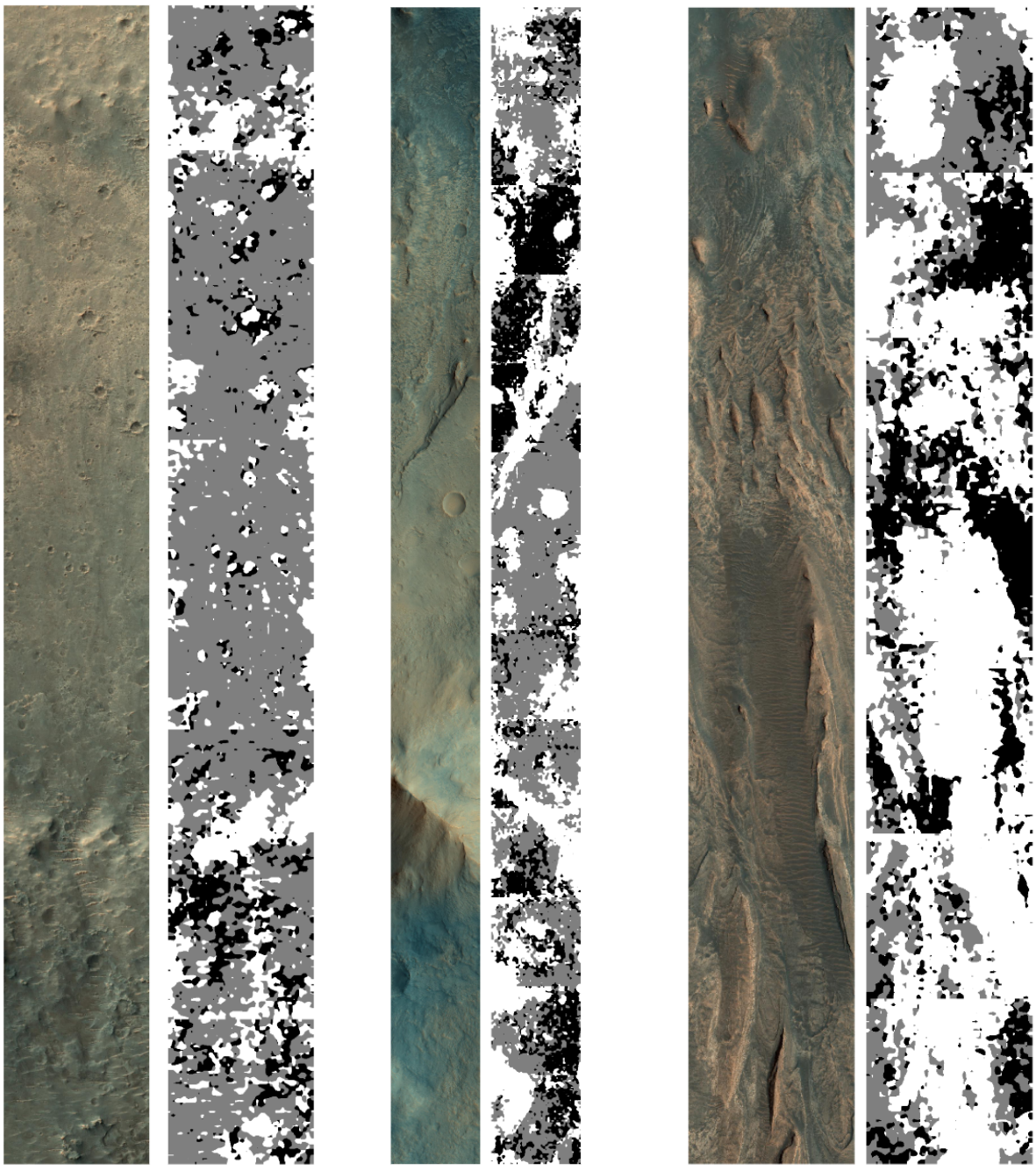
Looking at those images it is also to notice how sometimes the brightening conditions affect the prediction, and particularly they do it when the Sun is low over the horizon. This happens usually when objects inside the scene projected their shadow or when the low light incurs in the pattern of the ground. At the same time bounders and contours are well recognized, as proof of the goodness of the model.

5.2 Results and prediction on HiRISE

We then examined and looked at the real HiRISE dataset predictions, which actually can only be evaluated in a visual way, but at this point having established the validity of the STEGO model. The training parameters used are the one reported in the previous chapter.

The identified classes are three: smooth terrain, uneven/irregular terrain and relief, between which we consider as practicable the smooth terrain only. In the following prediction the smooth terrain class is represented by gray pixels, with consistency among all the predicted masks.

Comparing these example figures, what we obtained is an almost total non-practicability for image 21, a good traversability depending on the area for the image 11 and a fully traversability for what concerns image 16.



(a) n° 16 (b) n° 11 (c) n° 21

Figure 5.3: Source images and predicted masks

Chapter 6

Conclusions and further steps

The purpose of this Master Thesis work was to develop and deepen a neural network to perform image semantic segmentation on satellite captures from Mars surface. The semantic segmentation on Mars terrain images paves the way for the path planning the rover will perform, its goal is to give indeed initial recommendations on whether or not an area is traversable. We investigated either supervised and unsupervised models to perform this task and we came up with the development of a modified version of a ConvDeconv and a STEGO network. We then ended up choosing the latter since it best fitted our needs in terms of availability of a ground truth, GPUs availability, and results obtained previously with satellite images. No Mars application existed already and no ground truth was available for our dataset from HiRISE cam, that is why we validate the network with the aid of a synthetic Mars drone dataset.

To use a totally unsupervised approach for sure opened new possibilities and roadmaps for the future thanks to the fact that the effort in manual labeling to create ground truth masks is no more needed, but at the same time requires some more work to refine and improve either the results and the model itself. Predictions were fulfilling if examined in a visual and qualitative way, but the ordinary metrics were not appropriate to assess these results as we are used to, hence is highly necessary as a further improvement to develop a proper unsupervised metric for semantic segmentation.

Moreover, as part of the SINAV project, to better integrate the network in its pipeline and to enhance mask predictions and ensure then the creation of an adequate traversability map, we strongly recommend the development of an additional head. The head, together with the inclusion of a manual labeled small dataset, should be supervised so that by

training it, it will be possible to refine the very final output. The training of the unsupervised model should be performed on a big amount of data, in our case images, to better extract low level features and should aim after that at mapping to the class required, that will happen thanks to the addition of these few layers inside the head. For these layers in fact the training will be supervised and the network will be refined thanks to the comparison with the ground truth labels.

It is anyway important to be careful not to forget that in this way the smaller the labeled dataset will be the higher the network is specifically characterized in recognition of that few and particular features.

Furthermore and consequently to the previous point, a significant dataset should be labeled, at least while waiting for the definition of new appropriate metrics for this kind of tasks.

Finally a further study is needed to better define and perfect the strategy behind the choice of the classes to look for, even in their numbers. This study should take place possibly with the support of scientists with expertise in the field of planetary morphology and martian rover engineers. This final sentence remarks once more how this field is able to put together different scientist figure to work together and join their expertise mutually enhancing each other.

Bibliography

Salem Saleh Al-amri, Namdeo V. Kalyankar, and Khamitkar S. D. Image segmentation by using threshold techniques. *CoRR*, abs/1005.4020, 2010. URL http://arxiv.org/abs/1005.4020.

A.S.I. Proposta tecnico programmatica. sinav, soluzioni innovative per la navigazione autonoma veloce. *Bando pubblico ASI Tecnologie Abilitanti Trasversali. Area tematica Tecnologie Spaziali*, 2021.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *CoRR*, abs/2104.14294, 2021. URL https://arxiv.org/abs/2104.14294.

T. Claudet, K. Tomita, and K. Ho. Benchmark analysis of semantic segmentation algorithms for safe planetary landing site selection. *IEEE Access*, 10:41766–41775, 2022.

D.M. DeLatte, S.T. Crites, N. Guttenberg, E. J. Tasker, and T. Yairi. Segmentation convolutional neural networks for automatic crater detection on mars. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(8):2944–2957, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

P. Garg and S. Jain. Convolution-deconvolution-network. 2022. URL https://github.com/pgtgrly/Convolution-Deconvolution-Network-Pytorch.

Edwin Goh, Jingdao Chen, and Brian Wilson. Mars terrain segmentation with less labels. *CoRR*, abs/2202.00791, 2022. URL https://arxiv.org/abs/2202.00791.

Eric Guérin, Killian Oechslin, Christian Wolf, and Benoît Martinez. Satellite image semantic segmentation. *CoRR*, abs/2110.05812, 2021. URL https://arxiv.org/abs/2110.05812.

M. Hamilton, Z. Zhang, B. Hariharan, W. T. Freeman, and N. Snavely. Unsupervised semantic segmentation by distilling feature correspondences. *ICLR*, 2022.

Otsu N. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, man and Cybernetics*, 9:62–66, 1979.

S. J. Robbins and B. M. Hynek. A new global database of mars impact craters 1 km: 1. database creation, properties, and parameters. *J. Geophys. Res.: Planets*, 117(5), 2012a.

S. J. Robbins and B. M. Hynek. A new global database of mars impact craters 1 km: 2. global crater properties and regional variations of the simple-to-complex transition diameter. *J. Geophys. Res.: Planets*, 117(6), 2012b.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

L. Rubanenko, S. Pérez-López, J. Schull, and M. G. A. Lapôtre. Automatic detection and segmentation of barchan dunes on mars and earth using a convolutional neural network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14(8):9364–9371, 2021.

Kristina P. Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. *IEEE Access*, 8:80716–80727, 2020. doi: 10.1109/ACCESS.2020.2988796.

Humera Tariq and S.M.Aqil Burney. K-means cluster analysis for image segmentation. *International Journal of Computer Applications*, 96, 06 2014. doi: 10.5120/16779-6360.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Grace Vincent, Alice Yepremyan, Jingdao Chen, and Edwin Goh. Mixed-domain training improves multi-mission terrain segmentation, 2022.

Wenjing Wang, Lilang Lin, Zejia Fan, and Jiaying Liu. Semi-supervised learning for mars imagery classification and segmentation, 2022.

Song Yuheng and Yan Hao. Image segmentation algorithms overview. *CoRR*, abs/1707.02051, 2017. URL http://arxiv.org/abs/1707.02051.