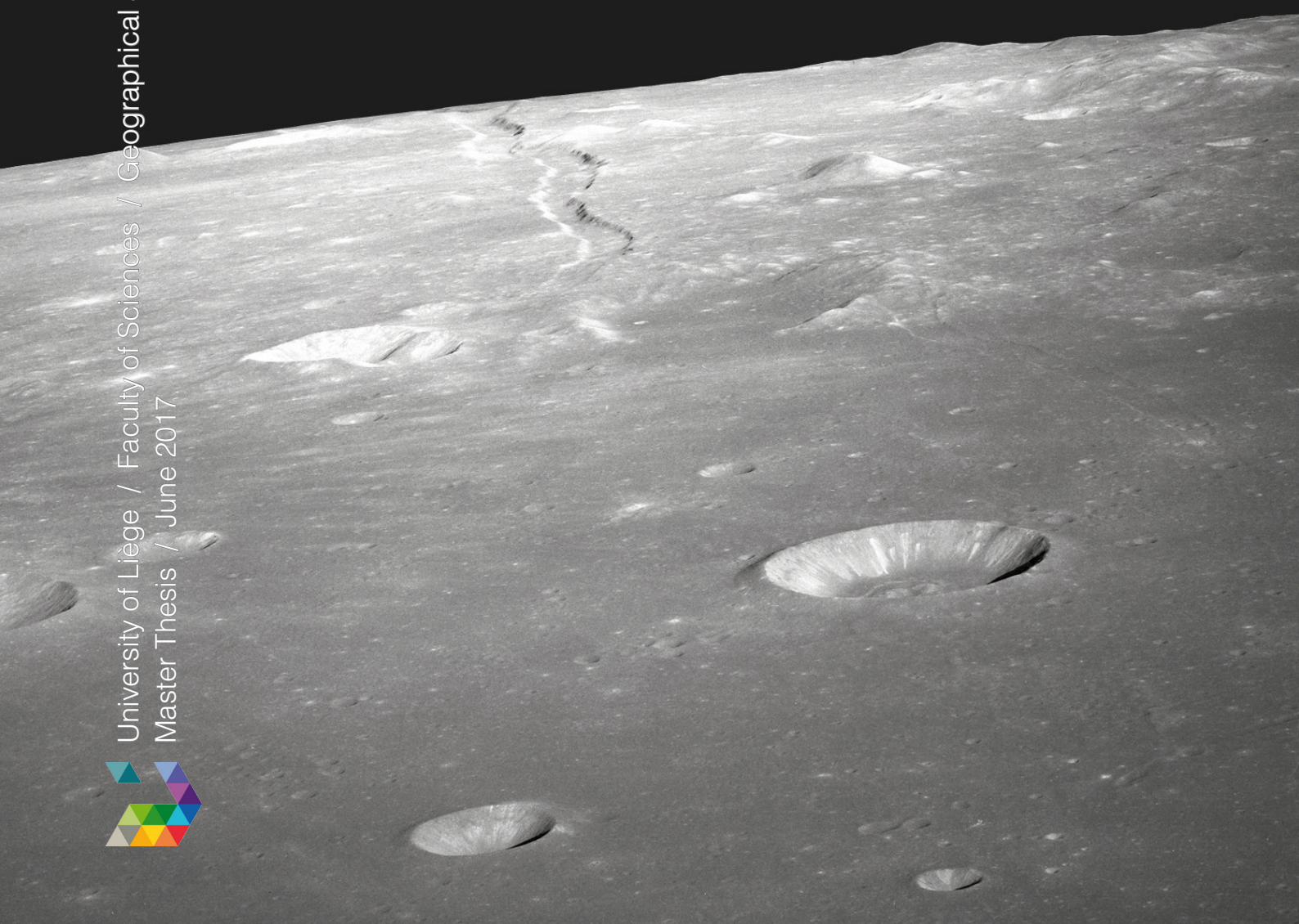
CraterNet

A Fully Convolutional Neural Network for Lunar Crater Detection Based on Remotely Sensed Data

Quentin Glaude

supervised by Mr. Yves Cornet



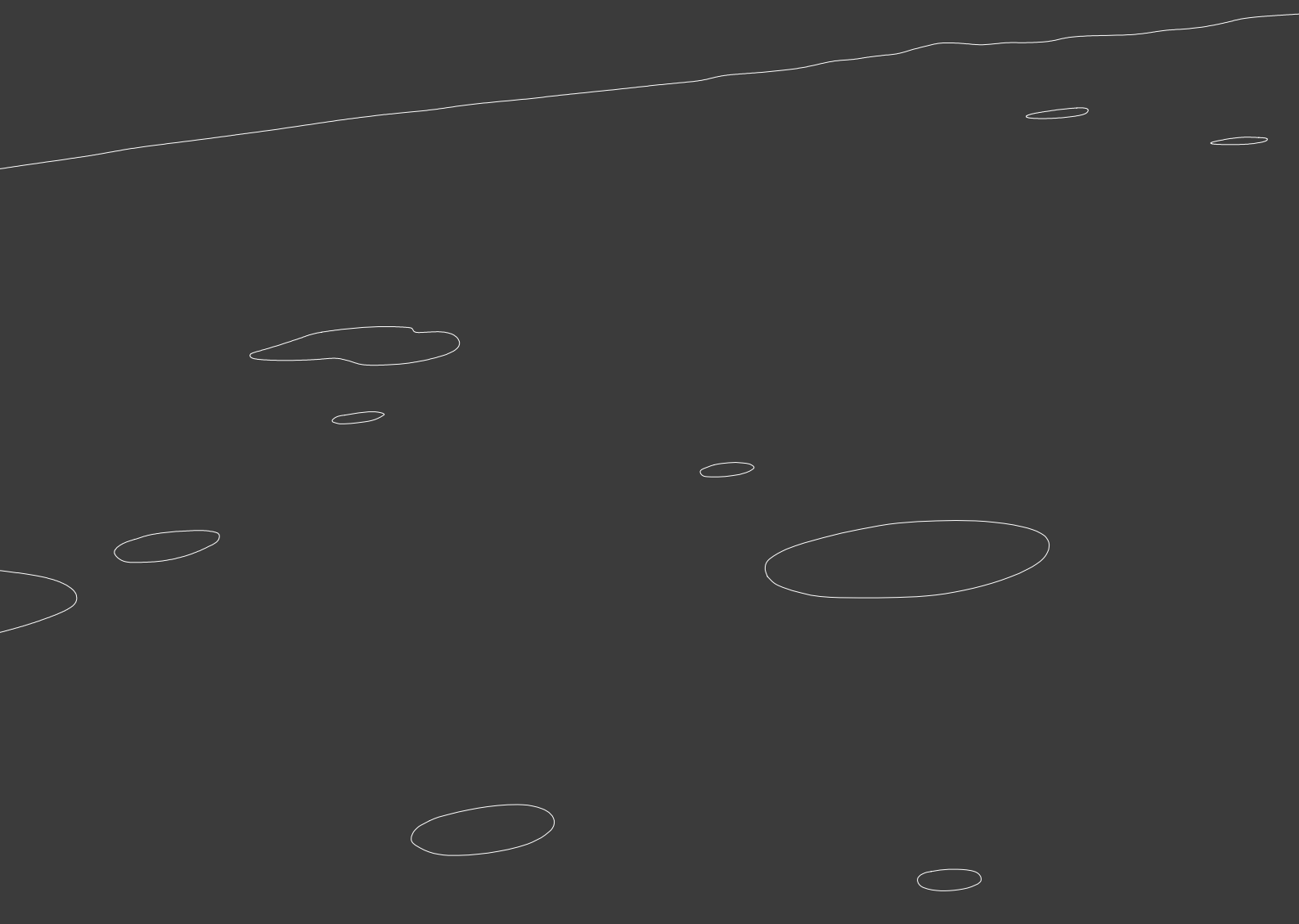
University of Liège / Faculty of Sciences / Geographical Sciences
Master Thesis / June 2017

CraterNet

A Fully Convolutional Neural Network for Lunar Crater Detection Based on Remotely Sensed Data

Quentin Glaude

supervised by Mr. Yves Cornet



As students cross the threshold from outside to insider, they also cross the threshold from superficial learning motivated by grades to deep learning motivated by engagement with questions.

John C. Bean

Firstly, I wanted to thank Mr. Yves Cornet, my promoter, for his involvement in my thesis. I also thank him for his infinite availability, his remarks and advice.

Then I wanted to thank Marc Braham for his huge help in the field of deep learning applied to image processing and without whom my project could not have been developed so far.

I also wanted to thank Sébastien Piérard for his advice and experience in machine learning and his interest in my work.

I wanted to thank Raphaël Marée and Romain Mormont for the integration of my project on the Cytomine platform and for the help they gave me.

I would also like to thank Adrien Deliège and Thomas Kleyntssens for their interest in my master thesis.

I also wanted to thank Philippe Latour for the expensive equipment that he allowed me to use at the beginning of the academic year.

I wanted to thank Antoine Lejeune for the time spent developing a stand-alone software dedicated to the creation of a dataset.

Then I would like to thank Marc Van Droogenbroeck for the valuable contacts he sent me among the staff of the Montefiore Institute.

A special thank to Bletana Tahiraj for her help in creating cover pages for my master thesis.

In closing, I would like to thank my family and friends for the moral support they have given during this year.

Contents

List of Acronyms and Abbreviations

API - Application Programming Interface

CDA - Crater Detection Algorithm

CDR - Calibrated Data Record

CPU - Central Processing Unit

CSV - Comma Separated Values

CUDA - Compute Unified Device Architecture

CuDNN - CUDA Deep Neural Network

CNN - Convolutional Neural Network

DEM - Digital Elevation Model

ESA - European Space Agency

FCN - Fully Connected Network

GPU - Graphics Processing Unit

GUI - Graphical User Interface

GMT - Greenwich Mean Time

IoU - Intersection over Union

LRO - Lunar Reconnaissance Orbiter

LROC - Lunar Reconnaissance Orbiter Camera

NAC - Narrow Angle Camera

NASA - National Aeronautics and Space Administration

ReLU - Rectifier Linear Unit

ROC - Receiver Operating Characteristic

SGD - Stochastic Gradient Descent

SSH - Secure Shell

URL - Uniform Resource Locator

WAC - Wide Angle Camera

1 | Introduction

The craters are important topographic objects on the surface of planets. They give useful information to researchers about their geology and the history of geomorphological process. Moreover they are witnesses of space particles density and inform about presence of allochthonous materials.

In addition, these topographic elements are crucial in the case of exploratory space missions. Some probes are not designed to safely land on these objects. Moreover, craters are structuring elements on the surface of planets that guide local illuminations conditions by producing shadows which can reach several kilometers. Indeed, energy produced by the sun is extremely important for the survival of a mission. Finally, craters constitute landmarks for space navigation.

All these scientific applications require crater detection methods. The problem of crater detection is not new. First, it was solved by manually locating craters on images. With finer remotely sensed data produced, that task became deprecated. Then, many scientists tried to develop supervised or unsupervised algorithms in order to automatically perform the task. Nevertheless, though much effort has been made, we still don't have an unanimously accepted solution.

With recent advances in deep learning associated to image processing, we wanted to try a new attempt considering the case of the Moon.

The rest of this master thesis is structured as follow. Chapter 2 focuses on the state of the art, by explaining how craters are formed and how they are described, the different methods to detect craters present in the scientific literature and the recent advances in machine learning that we will be used in our project. All these information lead to chapter 3 where we build the hypothesis we will try to validate. In chapter 4, we describe the data sources and how they are integrated in our work. Chapter 5 is the wider part, talking about the methodology, the difficulties encountered and their solution, the architecture of our model, the different post-process and so on. The results and the validation are part of the chapter 6 where we produced quantitative and visual experiments on our data with our model. Finally, in chapter 7 we summarize our work and look at the different possible

perspectives. The annexes gather information that are too exhaustive to be placed inside the main part of the document. It includes the list of images that have been used in our project, some visual/quantitative results and the source code.

The contribution of this master thesis is to develop a novel approach to detect crater on a lunar scene. Moreover, we make available a substantial dataset of craters and crater-free areas with more than 10 000 manually selected annotations. This dataset responds to a lack of lunar training data when trying to implement some supervised models and will be useful to researchers focusing on crater detection.

2 | State of the Art

Before going into the details of the developed methodology, it seems important to understand what we are talking about i.e. the concepts behind the notion of crater object.

Then we have to argue the motivation of detecting craters and why it has been a hot topic in space geomorphology for decades.

After that, we can go deep into the work of authors who implements some crater detection techniques to situate what has been done and what are their perspectives in the future.

To conclude, we need to look at the recent advances in machine learning to develop the key ideas of our novel approach to detect craters.

Note that this chapter is intended to be exhaustive enough while not being too technical but also tries to combine the two research areas : crater detection and deep learning. The readers that are very familiar to one of the topic can pass one of the different sections without any impact on the rest of the lecture.

2.1 Crater Description

Craters are one of the different geomorphological topographic elements on the surface of planets. There are different types of craters depending on the way they are formed :

- Impact crater
- Volcanic crater
- Subsidence crater

On the Moon, the large majority of craters are formed by the impact of meteors. The first step of the crater formation is the nearly instantly collision between the planetary

surface and the allochthonous object. A huge amount of kinetic energy is transferred from the meteor to the surface causing an important vertical compression (fig 2.1 top). The compressed soil partially relaxes ejecting a mass in the surrounding (sometimes causing other secondary craters). The depression consists in vertical deformations and horizontal relaxation (fig 2.1 bottom).

Then, the crater enters in a transition phase where it tends to stabilize its slopes up to a balanced profile. Finally, the crater progressively changes its shape through a very slow degradation process.

Note that the planet determines the erosion process. The Moon has no atmosphere nor water. On the Earth, geophysical process are regulated through common erosion/sedimentary process. Moreover, the crustal renewal of the land slowly removes the crater impacts. On the Moon, the transition phase exists even if the gravitational effects are less important. Thus, the slope equilibrium may be sharper than on the Earth. Nevertheless, the lack of atmosphere and water makes geomorphological process nearly nonexistent. Crater shape can still vary through the interaction of other meteor impacts causing seismic disturbances though these effects are infrequent. Finally, the lack of crustal renewal on the Moon avoids the craters to totally vanish.

The size, the speed and of less importance the trajectory of the meteor influence the visual aspect of craters. The shape is almost circle-like. The texture[1] can be however quite different. We can distinguish (see fig 2.2):

- Simple craters
- Complex craters
- Giant craters

The simplest craters are bowl-shaped with a pair of crescent-like highlight and shadow areas (see fig 2.2 left). The rims are formed of ejecta particles. The topographic profile has more or less a quadratic shape. The greater the crater the flatter the bottom of the depression. When the mass of the meteor is important or when its speed is high, we can observe a bounce effect after the compression phase, creating a complex crater (fig 2.2 center). This results in a central-peak in the crater. Finally, when the crater is very large, the seismic effects and the horizontal relaxation of the soil can form multiple rims (fig 2.2 right). That is the case in the Rachmaninoff crater (290 kilometers of diameter).

It has been shown that the transition diameter between simple to complex craters is about 15-30 kilometers on the Moon. Below this the crater can be considered as bowl-shaped (Renson P. & Y., 2014).

[1]Stockman and Shapiro describe the texture as something that *gives us information about the spatial arrangement of the colors intensities in an image.*

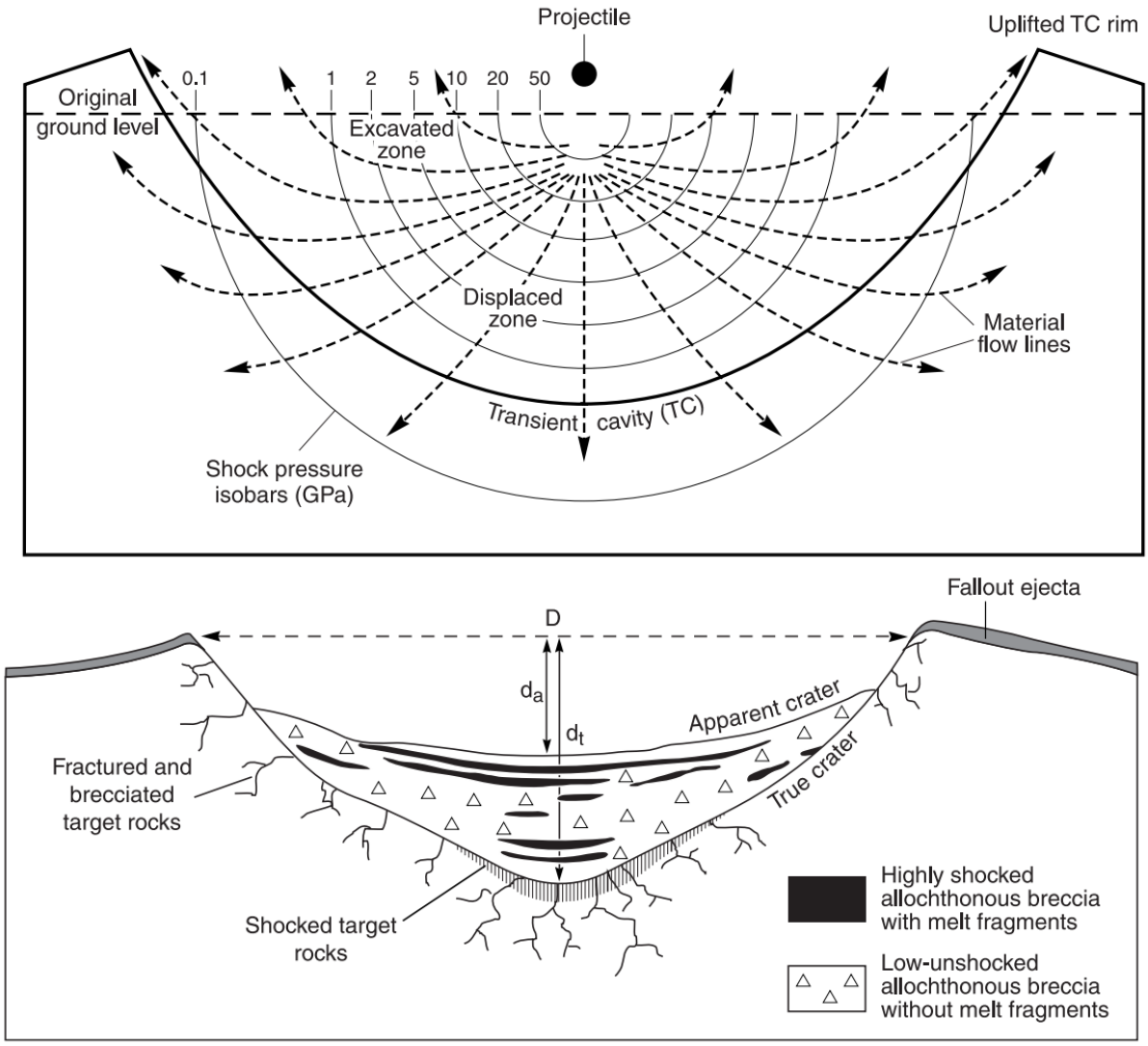


Figure 2.1: Evolution profile of the crater formation. Up : impact and excavation phases. Bottom : stabilization and filling in with allochthonous materials (not real-scaled) (French, 1998).

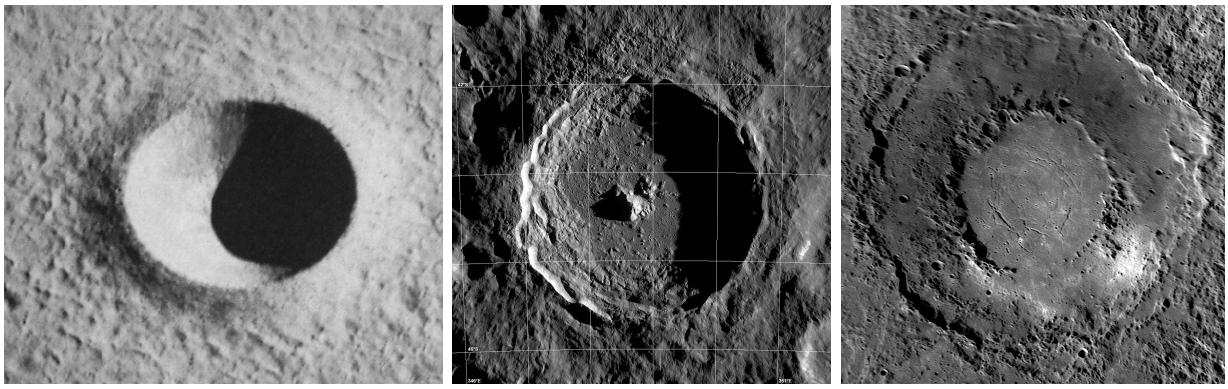


Figure 2.2: Different crater types. From left to right : simple crater, complex crater (Tycho) and giant crater (Rachmaninoff). Source : lunarpedia.org, 2013, NASA/GSFC/Arizona State University, 2011 and impact-structures.com, consulted in November 2016.

The last morphometric parameter that is used in crater description is the depth/diameter ratio. This ratio gives information about the length of the shadow and the sharpness of the crater. This ratio is correlated to different parameters such as the gravitational field, the presence of erosive agents and the size of the crater. The larger the crater, the smaller the depth/diameter ratio. It has an influence of the visual aspect of the crater. Indeed, a smaller ratio means a shorter shadow. For a restricted range of crater size, it is possible to make the assumption that the variability in this ratio is negligible (with other parameters fixed).

To conclude, the craters are the combination of several parts. We already discussed about the rims and the shadow, whose appearance is sensitive to the formation, the collision of the meteor, the illumination conditions and so on. There are also the highlight area and more finer elements. All these components can be analyzed in order to produce models able to recognize the crater object.

2.2 Crater Detection and Motivation

Craters are common objects on the surface of atmospheric-free planets. They are the result of the impact of a solid material with their surface. The study of craters has always been one of the hottest topics in space geomorphology. The planets are indeed covered by billions of craters witness of the space geology. Looking at size/frequency crater distribution, it brings information about the age of exposition surfaces of geological formations for example. This is the only way to estimate this dating by means of remote sensing. Geophysical process, local density variations and the nature of the degradation of craters can also be studied.

In addition, craters can be used for automatic navigation and landing of space modules. In this context, the objective is either to avoid craters and localize safe spots or recognize referenced craters to navigate to a well-known relative position. Thus the crater detection is part of avoiding hazards in a space mission.

However, a crater is a complex object. Its diameter size ranges from a few meters to hundreds kilometers, multiple craters can overlap, the nature of the ground can be different, its rims can have many levels of degradation and the illumination conditions vary their appearance. Moreover, the shape of these objects differs (boil-shaped, flat, central-peak and so on). In conclusion, it is difficult to find discriminant features able to correctly classify the crater object in a scene.

In the beginning, the task was solved by visual interpretation on passive or active remotely sensed images. It consisted in a colossal task for researchers. There are several catalogues of Martian and Lunar craters with dozens of thousands of craters with a diameter of a few kilometers. However, the lack of sub-kilometric dataset is still embarrassing in some studies.

Nevertheless, with the increase of remote sensing in space sciences and the improvement of the space probes sensors, we begin to see finer objects on a still growing number of images. The frequency distribution of crater diameter follows a power-law with a few craters with high diameter-size and a tremendous number of craters with a low diameter-size. Thus the manual task become impracticable and the need of a *crater detection algorithm* (CDA) was born. The expansion of these data resulted in the demand of databases for scientists. Then, it would be possible for them to perform data-mining or extract relevant information.

There have been a lot of papers about the detection of craters : 77 papers in 2012 (Salamunićcara & Lončarićb, 2008). Unfortunately, there is still no standard solution or a concrete answer to this problem. Actually, the methods do not achieve sufficient accuracy to be used in general contexts. One of the main concerns is to find the features with which we can semantically segment a scene. In computer vision, these kinds of detection algorithms can be classified as unsupervised or supervised algorithms. Some CDAs combine both.

Many unsupervised works are related to pattern recognition and are based on the construction of common filters. We can cite Gabor, Canny, or Sobel filters to enhance the rims on the image. Then, with the help of the Hough transform the circular shape of the crater can be reconstructed (Meng, Yunfeng, & Qingxian, 2009; Renson P. & Y., 2014; Honda & Azuma, 2000). In this particular methodology, the intervention of an expert is needed to choose the best filters to recognize the objects.

One of the most cited papers in supervised techniques uses mathematical morphology to extract crater candidates (Urbach & Stepinski, 2009). The basic idea is that a crater is a combination of two crescent-like shapes, one of which is bright (corresponding to the

illuminated area) and the other is dark (corresponding to the shadow). This method is said to be useful because rotation and scale invariant. Nevertheless, this assumption is correct only if every crater characteristic changes proportionally to the size. Based on these candidates and a set of examples, many authors tried different machine learning models to classify what is a crater and what is not (L. Bandeira, Ding, & Stepinski, 2012; Ding et al., 2010; Wang, Ding, Yu, Wang, & Wu, 2011; Cohen, Lo, Lu, & Ding, 2016). In these papers, the method to find candidates alone is not sufficient to separate craters from other objects. Similarly, the machine learning model alone is not able to find the crater but rather to classify the candidates.

L. Bandeira et al. said in 2012 : *A good CDA requires a well-known set of discriminating features, an efficient method for finding all possible crater candidates and a set of criteria that can accurately separate craters from non-craters in the set of candidates.*

The unsupervised techniques have the advantage of being fully autonomous (no need for learning samples) but have lower accuracy than supervised methods. However, machine learning-based algorithms need a set of labelled data to be functional. In this context, a large sub-kilometric dataset has been built on a restricted area on Mars with thousands of referenced craters (Ding et al., 2010).

On the Moon and on Mars, the number of craters is really high compared to the Earth. Actually, the erosion/degradation process is very slow. The absence of water is the main reason (No particles transport by runoff and no frost weathering). Moreover, there is no atmosphere on the Moon thus no wind. Also, the gravitational attraction is lower than on the Earth. Finally, the tectonic effects and therefore the renewal of the surface soil is missing.

2.3 Crater Detection Algorithms (CDAs)

2.3.1 Unsupervised Methods

The goal of unsupervised learning methods is to infer data structures out of unlabelled data. The developed methods often include a rims enhancement process and then reconstruct typical geometries using the Hough Transform.

Finding the rims of the craters can be seen as an edge detector problem. Different methods exist. We can cite : Robert cross (Honda & Azuma, 2000) and Canny filter (Renson P. & Y., 2014). Other edge detectors also exist (Sobel filter and laplacian filter for example).

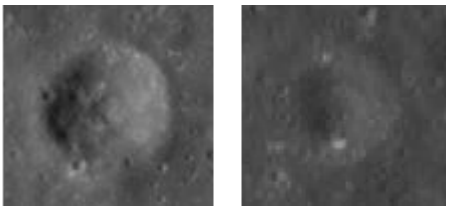


Figure 2.3: Difference between a well-delimited crater and an eroded crater.

$$\text{Robert cross} : G_1 = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} and\ G_2 = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} then\ \nabla I_r = \sqrt{G_1^2 + G_2^2}$$

$$\text{Canny filter} : G_1 = \begin{bmatrix} +1 & 0 & +1 \end{bmatrix} and\ G_2 = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} then\ \nabla I_c = \sqrt{G_1^2 + G_2^2}$$

In (Meng et al., 2009), the authors claimed that the variation in intensity is too smooth to detect eroded craters with these common filters (see fig 2.3). Instead, they use an algorithm developed by L. P. C. Bandeira, Saraiva, and Pina.

After that, the Hough transform is applied. The objective is to find the best parameters which fit a selected analytic formulation of the geometry (most of the time a circle).

The Hough transform is a way to go from the image space into the parameters space, called Hough space. The Hough space has as many dimensions as the number of searched parameters. For the case of a circle, we have three dimensions (x position, y position and r radius). For an ellipse, the number of parameters is 5. The complexity of fitting the rims of the craters to a typical shape increases drastically with the number of parameters.

In the studied case (circle), we can see the problem as finding for each pixel coordinate the best possible radius if one exists (see fig 2.4).

Note that the results applying the Canny edge detector (or other edge detection filters) are not perfectly continuous and/or displayed in a perfect circle shape. Thus the mask of the rims is dilated to increase the probability of finding an unseen crater due to noise (Renson P. & Y., 2014).

In the work of Renson P. and Y., this process only gives a set of candidates. Some descriptors are inferred from these candidates and are compared to a set of descriptors built with real case craters. Thus, the method needs a training set to pre-tune these features

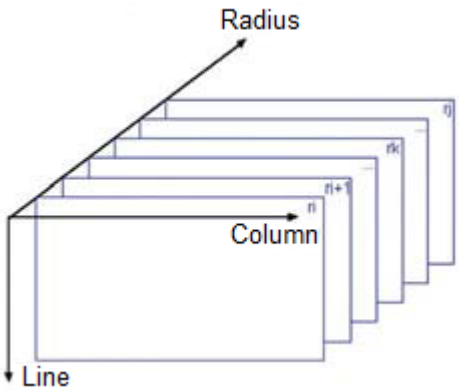


Figure 2.4: Hough space and its 3 parameters (line, column and radius).

2.3.2 Supervised Methods

Crater Candidates

Supervised methods are techniques using different machine learning algorithms to detect craters in a scene. Many of these are built to recognize objects apart from any spatiality, meaning setting a label, and not really to detect the craters in a scene. Thus a candidates generator algorithm has to be expressed before the classification task.

A lot of supervised methods use the work of Urbach and Stepinski to generate candidates (their candidates are available in .csv format containing position and radius corresponding to a specific scene on Mars). Considering these candidates, many authors developed different machine learning methods.

The method developed by Urbach and Stepinski is based on the observation that a crater is a combination of a two crescent-like shapes (one for the shadow and the other for the highlight area). The scheme of the technique can be seen in figure 2.5.

Step by step, the method eliminates elements which are not prone to be a crater. As the goal is to find some crescent-like highlight and dark areas, the technique is developed only to determine the highlight areas then the shadow area is found by inverting the input image.

The background removal consists in a very large low frequency filter subtraction. In this part, the method uses a median filter of tremendous kernel size to eliminate the light alterations associated to natural topography such as big valleys and depressions.

The power filter is a type of filter developed by Young and Evans (2003) to remove indiscernible elements using morphological image cleaning. Traditionally, this filter is used for noise reduction to improve image compression systems. Without going deep in the details, the power filter is a kind of very high-frequency denoising using attribute-based open–close filters.

The area filter eliminates too small elements. In their papers, the authors defined an area threshold of 30 pixels, below which it is not possible to recognize a crater.

The shape filters discard elements which are not crescent-like. The shape is defined by a combination of Hu's seven moment invariants (M.-K. Hu, 1962). The moments are specific weighted averaging of the image, whose objectives are to be representative of different characteristics that can be interpret with ease. The use of these moments gives to each pre-candidate a score measuring how far the element is from a crescent-like shape.

The last step consists in assembling the pairs of highlight/shadow areas. In order to do this they examine each possible pair of highlight and shadow area. A candidate is formed if :

- The distance between the two is small
- The two areas have similar size[2]
- The combination makes a circle
- The regions are aligned with the solar azimuth (a priori known through image metadata).

When the candidates are finally found, the small gaps within the selected area are filled using morphological closing operation.

Note that this detection method is invariant to any scale factor, rotation or translation. In other words, the method is insensitive to any Helmert transformation. The scale factor invariant is related to the assumption that with the range of considered craters, the variation depth/diameter ratio does not impact the method.

Decision Tree

Originally, the authors of the candidates selection method (Urbach & Stepinski, 2009) used the C4.5 algorithm (Quinlan, 1986) to build a decision tree classifier to recognize the

[2]Actually, it depends on the illuminations conditions and the depth/diameter. In their article, Urbach and Stepinski use the following condition : $0.25 \leq \frac{HighlightRegion}{ShadowRegion} \leq 4$. If the sun is at high or low elevation or if the crater is large, the condition must be relaxed.

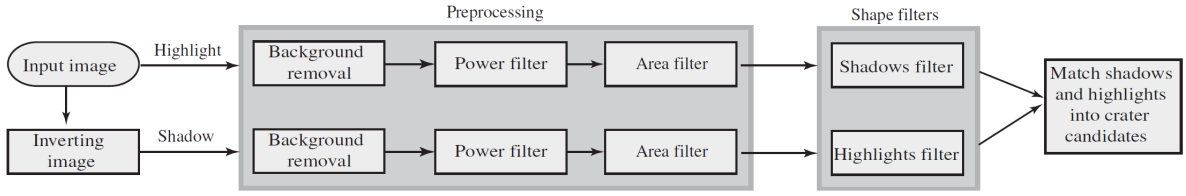


Figure 2.5: Identification of candidates (Urbach & Stepinski, 2009).

craters among all these candidates. In their paper, a vector of attributes is assigned to each region. The features used are the Hu's seven moment invariants (M.-K. Hu, 1962).

A C4.5 decision tree defines a set of rules based on a training sample. The result is a tree whose splits are formed by maximizing a reduction of an impurity measure (often the Shannon's entropy). This reduction of entropy is also defined *information gain*.

Feature Selection and Boosting

We saw that one major problem in crater detection was to find the relevant features able to clearly separate what is a crater and what is not. In some papers written by L. Bandeira et al. (2012), the solution consists in computing hundreds of features and using an algorithm to sort and rank them. Then, they are used in a modified version of the Adaboost algorithm[3]. The algorithm is applied on the bounding boxes of the candidates described in the beginning of section 2.3.2.

The features used in the papers are a combination of Haar-like image texture features, based

[3] Adaboost is a boosting method developed by Freund and Schapire, whose objective is to combine the outputs of a large amount of weak models in order to produce a strong model with low bias. The prediction is made by setting the output value to the majority of the all outputs predicted by the weak models, each weighted according to their accuracy.

In addition, the importance of misclassified elements is increased each time a weak model is created, in order to force the model to concentrate on errors. If we have N observations with variables x and output y and T weak models are built, the algorithm computes the final model using the following algorithm (notations are from Geurts and Wehenkel (2016)):

The weight of each element i is firstly set to $w_i = \frac{1}{N}$.

Then, for each model built, compute the weighted error $err_t = \frac{\sum_i w_i I(y_i \neq \hat{y}_t(x_i))}{\sum_i w_i}$.

Next, the weight associated to the model is computed according to its accuracy by $\beta_t = \log \frac{1-err_t}{err_t}$ and the weights associated to classified elements are increased by $w_i \leftarrow w_i \exp(\beta_t I(y_i \neq \hat{y}_t(x_i)))$ then normalized, so that the sum of all w_i is equal to 1.

Finally, the label of the element is predicted by $\hat{y}(x) = \sum_{t=1}^{T} \beta_t \hat{y}_t(x)$ where x is the inputs of any element.

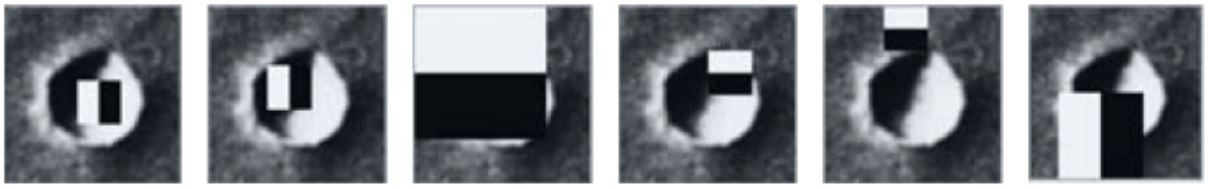


Figure 2.6: Each Haar-like feature is computed using a binary mask. The binary mask is a rectangular sub-region of the image with a white part and a black part in equal area. The Haar-like feature is computed by subtracting the sum of all pixel values contained in the black part to the sum of all pixels values in the white part. In the example above, 6 Haar-like features are produced (L. Bandeira et al., 2012).

on the theory developed by Haar (1910). These variables are widely used in computer vision in the context of face detection algorithm, largely promoted by Viola and Jones (2001). They consist in a variety of binary filters defining regions in a particular kernel. These features are easy to compute with the use of integral images. By combining the 9 different Haar-like features types, their scale and their position, L. Bandeira et al. computed 1089 input features.

Then the candidates are classified using the boosting-like algorithm developed by the authors. In the paper, the weak models consist in single node trees.

LASSO

The LASSO algorithm is a way of selecting features within the machine learning algorithm. In the method developed by Wang et al., a Bayesian Network Classifier is used with a L1 penalty. The penalty forces the model to select only the most relevant features, meaning the variables which, by themselves, explained the vast majority of the problem.

In a linear model with L1 penalization, the model is constrained by the sum of absolute weights. With a least squares approach, the optimization algorithm consists in finding

$$min_\beta\big(\sum_{i=1}^{N}(y_i - (\beta_0 + \sum_j \beta_j x_j))^2 + \lambda \sum_j |\beta_j|\big) \tag{2.1}$$

Where β_i are the parameters of the linear model. From a graphical point of view (see fig 2.7), we can represent (in the case of a two variables optimization) $\hat{\beta}$ the optimal pair of variables, meaning that $\hat{\beta}$ minimizes the expression above without the penalty term. As

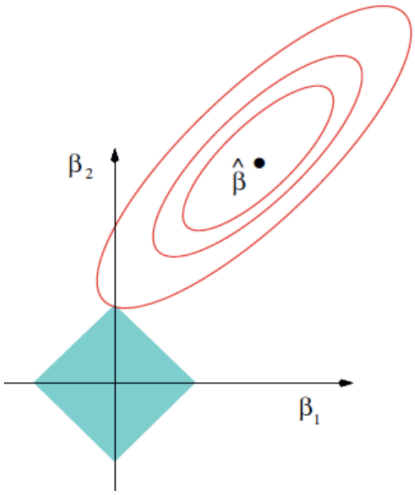


Figure 2.7: Representation of the penalization term effect on the optimal set of β. The elliptical shape represents the increase of the first term of the equation when the parameters move away from $\hat{\beta}$ whereas the diamond shape centered on the origin of the axis corresponds to the second term. The optimal set of β is found by the intersection of the two terms (Geurts & Wehenkel, 2016)

.

one moves away from $\hat{\beta}$, the score increases. With a penalization on the sum of absolute weights, we find another couple of β by the intersection of the two terms of the equation. One of the variable is now set to zero (β_1).

Convolutional Neural Networks

The convolutional Neural Networks (CNN) are new techniques derived from deep learning, vastly used in computer vision with promising results.

As many deep learning approaches, the model learns the convincing features without any human intervention. The goal is to present the model enough labelled images to let it train the features useful for the classification task. As in previous methods, the CNN cannot detect craters on a scene by itself but only classifies an image in a whole. In their article, Cohen et al. also uses the previously defined candidates that they classified.

The goal of a CNN is about finding a set of filters that computes features maps taking the context of the image into account. On these features maps, it is possible to find other filters to compute other features maps. Hierarchical features are built step by step, assembling previous features. Note that the CNN architecture will be discussed in more details in a

dedicated section (2.4.2). I invite the readers to come back to this section later.

The CNN is here defined with a set of 20 filters to train constituting the first convolutional layer. Then, 20 another filters are applied on these 20 first features. Finally, they obtain the fully connected layer consisting in 500 neurons and the labelling output.

2.3.3 Conclusion

Only a glimpse of all CDAs has been described in current section. Salamunićcara and Lončarićb list much more articles on the subject from 1999 to 2007 in a synthetic way. I only detailed the more promising and well-known articles in the literature.

The unsupervised methods are based on the same principle: extract rims and reconstruct typical crater shapes. They differs by using the knowledge of an expert to adequately use contextual information. However, the accuracy does not enable their use in the previously described applications. The article written by Renson P. and Y. is interesting but use a set of training examples. However, he developed a novel way to detect candidates.

Also, other machine learning techniques exist. We can cite for example the use of support vector machine as a crater classifier. Nevertheless, we notice that these models lack in their ability to detect craters in a scene by themselves. In all the discussed techniques, the authors used the crater candidates from Urbach and Stepinski (2009) to apply their machine learning technique.

An alternative option would be to detect shadows (Rufenacht, Fredembach, & Susstrunk, 2014; Kadhim, Mourshed, & Bray, 2015) in order to attenuate the restrictive illumination conditions using topographic normalization. However, many techniques to detect craters use the particular shape of the shadows. Moreover, the topographic normalization needs near-infrared data. We cannot ensure that all lunar space probes are equipped with this type of sensors.

The accuracy of machine learning methods is sometimes really good (in particular with the use of CNN) but are restricted to the ability of the candidates selection algorithm not to miss some craters. In other words, the number of false negative is restricted by this previous work. Some of these authors claim that their algorithm may have better results with an exhaustive pixel-based search but this solution is not feasible in practice.

With all the methods developed in the scientific literature, Salamunićcara and Lončarićb propose different ways to display results/evaluate CDAs in common ways to ease the comparison between articles.

2.4 Machine Learning Advances

2.4.1 Machine Learning and Deep Learning

Machine learning is a field of artificial intelligence which got an important gratitude these last few years. Artificial intelligence has been a field of interest of computer scientists for about 50 years ago with Turing as pioneer (1950). At the beginning, researchers in this domain where very enthusiastic and much effort has been made. Famous authors from that period are Newell and Simon for their work in problem solving domain and Shank and Tesler for their work in natural language understanding. Some researchers thought that they could rapidly recreate human-like skills in many applications. Marvin Minsky, from the MIT, said in 1970 : *In from three to eight years we will have a machine with the general intelligence of an average human being* (Walter, 1994). Nevertheless, though they produced really interesting results, we are still far from the conception of the human behaviour's complexity. Follows a quiet period of disillusion (called *cold winter*) when noticing that some problems were underestimated.

In the nineties, there was a rebirth of the artificial intelligence through the concept of machine learning. Some important discoveries are the random forests (Ho, 1995) and the support vector machines (Cortes & Vapnik, 1995). The key insight is the use of a set of examples to learn. The goal is actually to define an algorithm able to find a function/relation which best approximates the output based on a training dataset. These data are called observations. They are collected either by the user or via available dataset. Using the notations of Geurts and Wehenkel (2016), we can summarize machine learning problems as

From a learning sample $\{(x_j, y_j)|j \in \{1, ..., k\}\}$ *with* $x_j \in X$ *and* $y_j \in Y$*, find a function* $f : X \to Y$ *that minimizes the expectation of some loss function* $\ell : X \times Y \to \mathbb{R}$ *over the joint distribution of the input/output pairs.*

	x_1	...	x_i	...	x_n	Y
element 1	x_{11}	...	x_{1i}	...	x_{1n}	y_1
element 2	x_{21}	...	x_{2i}	...	x_{2n}	y_2
...						...
element j	x_{j1}	...	x_{ji}	...	x_{jn}	y_j
...						...
element k	x_{k1}	...	x_{ki}	...	x_{kn}	y_k

$\Rightarrow$ Find f(X) so that $E_{X,Y}\{\ell(f(X), Y)\}$ is minimum.

The output is either symbolic (classification) or numerical (regression). A well-trained

model is able to predict the output of unseen elements. The prediction is indeed a guess of the most likely value/class according to the ability of the model to generalize the problem. Also, the model helps the user understands the relationship between the variables.

In the previous section, we mentioned several crater classification techniques using machine learning (C4.5 decision tree for example).

The *deep learning* is simply another way to use machine learning. The figure 2.8 gives a visual interpretation of the relationship between deep learning and the concepts previously discussed.

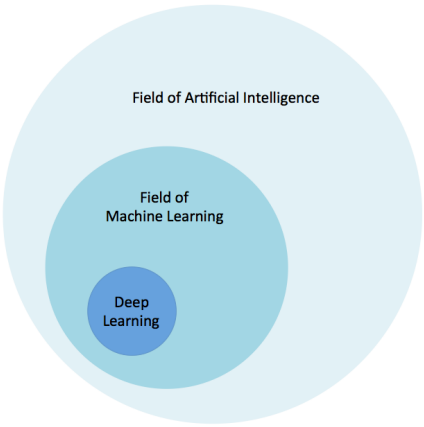


Figure 2.8: Set representation of Artificial Intelligence, Machine Learning and Deep Learning (Algorithmia, 2016)

Deep learning techniques are often related to *artificial neural networks*. This term is chosen to make the analogy with human neurons. In machine learning, a neuron is a structure that receives different inputs and computes one single output with a weighted summation of the different inputs (and bias) (Rojas, 1996). Then an activation function is used (see fig 2.9). The main purpose of the latter is to force the neural network's decision boundary to be non-linear (Raschka, 2015).

$$y = f(\sum_{j} w_j * x_j + b) \tag{2.2}$$

Where x_j is the j^th input, w_j its associated weight and b is a bias term.

However, multiple neurons are often used in deep learning models. Then, an architecture of several layers each containing a certain number of neurons can be built (fig 2.10), called

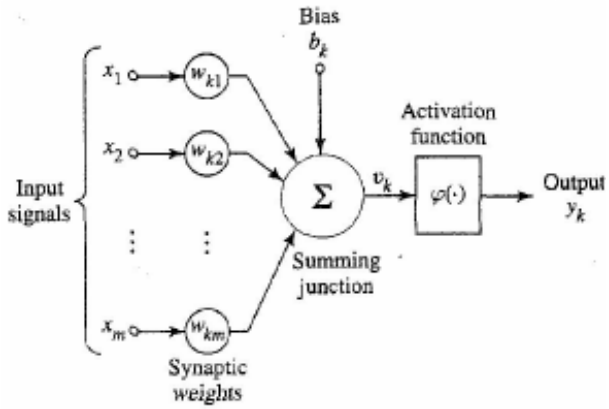


Figure 2.9: Single neuron representation (Edgar Amaya, 2017).

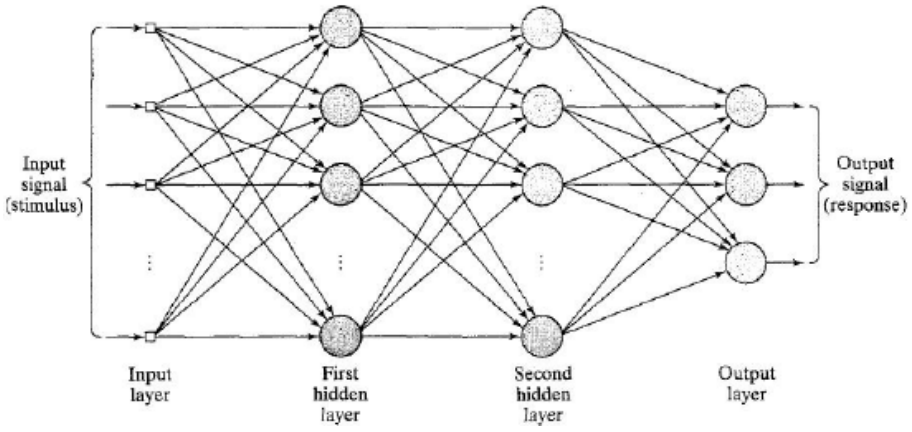


Figure 2.10: Architecture with multiple neurons (multi-layer perceptron) (Edgar Amaya, 2017).

a multi-layer perceptron.

When assembling many neurons, a neural network is formed. This network can rapidly have thousands of parameters (weights and bias) and can approximate any kind of function, which makes it multipurpose.

Even if the model can potentially be very complex (far more than "traditional" machine learning models), they are trained in the same way: given a set of observations with associated output, the algorithm tries to find the weights of the model that minimizes a loss function.

Similarly to our brain which reinforces the synaptic strength of certain synapses when we learn new things in our life, a neural network adjusts its weights during the training

process.

As before-mentioned, a neural network has an important number of parameters (up to millions). The number of possible combinations of the different weights is infinite, what makes it difficult to train. That's the reason why a lot of networks originally have one single hidden layer (i.e. a layer which is a between the input and the output). Moreover, when multiplying the number of hidden layers, the model is affected by the *vanishing gradient effect*. This effect occurs because the gradient computed by back-propagation follows a chain rule from the last layer to the first. Computing that gradient layer by layer means that it decreases exponentially from the last layer to the first, making the first layers difficult to train.

In computer vision problems, we work with images. The number of input variables in an image is equal to the number of pixels[4], which can be important. Moreover, an image associated to a label can be highly different. If a developer wants to produce a model able to recognize a car, he has to show the algorithm a sufficient amount of examples such that the model can predict the label of an unseen image of car. But with this example, many possibilities exist (color, type, shape and so on), complicating the problem.

In image recognition, the one hidden layer model is often not sufficient but the multi-hidden layer is still too difficult to train and use. The idea is thus to create a set of characteristics of the image and give them as input of the one hidden layer model. This step of extracting features is a very important stage as already discussed in the description of CDAs. It requires the knowledge of an expert who determines what are the best characteristics to extract in a specific task. Indeed, the quality of the model is thus related to the expertise of the developer.

In a deep learning approach, no set of features is given to the algorithm. A deep neural network is simply built and, in the last layers, the algorithm creates a set of hierarchical characteristics hidden in the neurons. Hence, we do not need an expert : the model learns the features by itself.

2.4.2 Convolutional Neural Networks

Machine learning in computer vision has always been a highly particular task due to the great number of input variables and the importance of spatial auto-correlation between the pixels. Deep learning takes this last assertion in consideration to create a new kind of architecture specific to images. That architecture is called a *convolutional neural network* (CNN), with LeCun, Bottou, Bengio, and Haffner as pioneers but Krizhevsky, Sutskever,

[4]Note that the pixel is here considered as a variable (or a vector of variables for multi-channel images), contrary to geo-spatial analysis where the pixel is above all a segmentation of the space according to a specific ground sample distance (GSD).

and Hinton are famous names too in the deep learning field.

CNN, as other deep learning architectures, is composed of a set of layers. Yahn Lecun described the functioning of its network saying that *the first layer detects small elementary contours, the second layer assembles these contours into patterns, then the patterns are assembled into parts of objects, parts of objects into objects and so on.*

The word *convolutional* is related to the convolution operation in image filtering. Actually, the goal of a CNN is to produce a set of auto-trained filters.

The architecture of such a network is composed with three types of hidden layers: a convolutional layer, a pooling layer and a fully-connected layer (see fig 2.11).

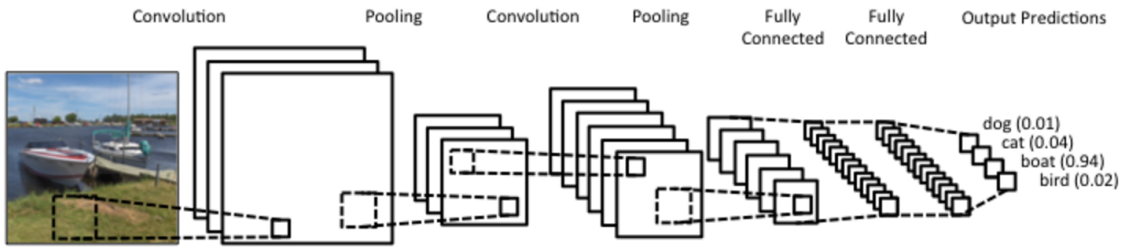


Figure 2.11: Example of CNN model (WILDML, 2015).

In previously described neural networks, the network was described as a sequence of layers from input to output with a vector representation of each layer (see fig 2.10). In CNNs, each layer is described as a 3D tensor to make the representation more convenient. The input layer has indeed three dimensions : two for the geometry and one for the number of channels (3 channels in RGB images for example). This visual representation can also be used for the other types of layer.

Convolutional layer

In machine learning, developers use common filters/kernels in order to extract some contextual quantifiers. For example, the Sobel filter is used in edge detection (fig 2.12).

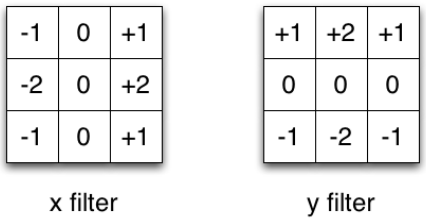


Figure 2.12: Approximation of horizontal and vertical derivatives by using the Sobel filter.

A filter is a matrix with weights. Image filtering consists in sliding the kernel over the image doing a convolution operation (fig 2.13) with a given row/column step. The greater the dimensions of the matrix, the further contextual information is taken into account.

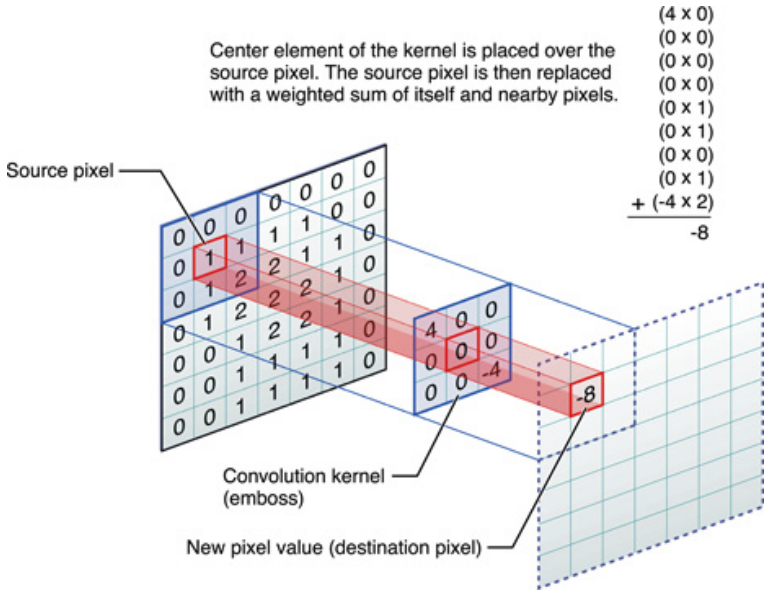


Figure 2.13: Filtering operation (developer, 2016).

In the convolutional layer, several filters produce a certain amount of outputs. The initial weights of the filters follow a probabilistic distribution law[5]. In the training process, the algorithm adjusts its weights to produce relevant features. The convolutional layer is sometimes used as a features extraction technique. This approach is useful because it doesn't need human intervention.

[5]Note that we cannot simply set the weights all equal to a specific value (1 for example) because of symmetry issues in the training process.

As convolutional operators are used, the network is more locally connected than classical multi-neurons networks (importance of contextual information). Note that the sliding operation does not have to be exhaustive thanks to the auto-correlation. The parameter defining the sliding step is called the stride. A stride greater than one makes the geometry of the output smaller. As layers are connected by filters, the number of parameters is highly reduced (in multi-neurons models, two successive layers consists in a complete bipartite graph).

After the weighted summation, the result is then post-processed to break the linearity using an activation function. That function was originally a sigmoid or a hyperbolic tangent. However, the *Rectified Linear Unit* (ReLU) is nowadays commonly used because it shows satisfying results and is highly more efficient (Maas, Hannun, & Ng, 2013). The softplus function is another activation function that approximates the ReLU. Figure 2.14 shows different activation functions.

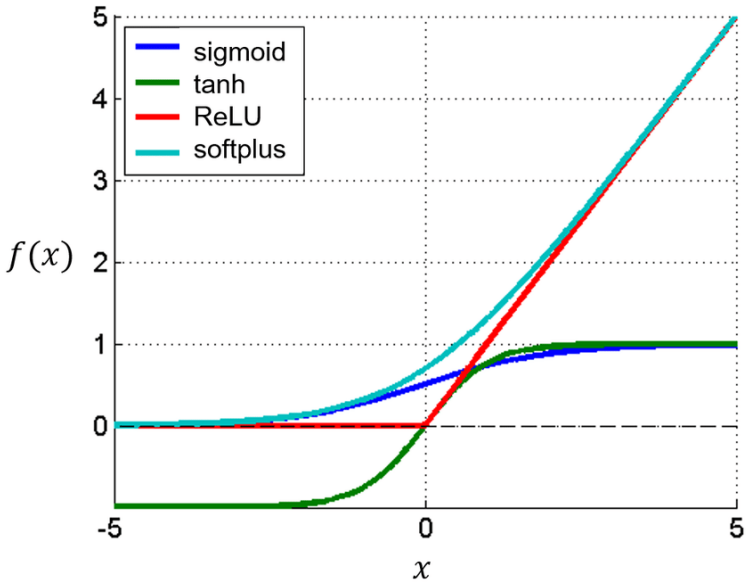


Figure 2.14: Examples of classical activation functions.
$sigmoid = \frac{1}{1+e^{-x}}$, $tanh = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, $ReLU = max(0,x)$, $softplus = ln(1+e^{x})$

Pooling layer

The pooling layer consists in a sub-sampling operation decreasing the size of the image. It limits the computation cost of the training process but also generalizes the layers, which decreases potential over-fitting effects (i.e. a model that is restricted to produce satisfying results only on the observations used to train the model).

The pooling process is defined by two parameters : the kernel size and the stride. These parameters have strictly the same role as in the convolutional layer. The pooling layer is also a filtering operation. Originally, the operation itself was an average over the kernel. However, experiments shows that a max operation (called max-pool) gives better results. Nowadays, some deep CNNs use both : GoogLeNet is one example (Szegedy et al., 2014).

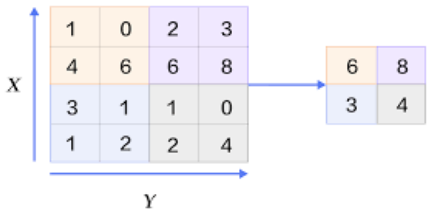


Figure 2.15: Max-pooling operation (kernel 2x2, stride 2x2) (Li et al., n.d.).

Fully-connected layer

At the end of the multiple convolutional/pooling layers, the features of the input images are sufficiently well-encoded to be all mixed together, meaning that a combination of these hierarchical characteristics enables the model to recognize the object. This stage corresponds to an one hidden layer model with a set of features as input. The difference between traditional approaches using one hidden layer models and CNNs is that this set of relevant features is automatically computed without any human intervention.

It is possible to have multiple fully-connected layers. However, the more hidden layers, the more difficult is the model to be trained.

Now that the different parts of the model have been described, I suggest the reader to go back to the end of section 2.3.2 to clearly understand how Cohen et al. created a CDA based on CNN.

2.4.3 Fully Convolutional Networks

As described in section 2.3.2, the machine learning models (a fortiori deep learning models) are not built for crater detection but rather for crater recognition. The CDA developed are based on a set of crater candidates which are post-classified using these supervised algorithms. The output is only semantic. The CNN is not an exception to the rule. The CNN developed by Cohen et al. (2016) uses the same process to produce a CDA. One solution to semantically segment a scene is to perform an exhaustive search on the image,

by extracting pixel per pixel the surrounding window and *in fine* labelize the whole image. This technique is however not feasible with remotely sensed data, considering the amount of time needed to analyze such a large scene.

Fortunately, in 2016 came important articles related to *fully convolutional networks for semantic segmentation* (FCN) (Zhao, Shi, Qi, Wang, & Jia, 2016; Shelhamer, Long, & Darrell, 2016; Badrinarayanan, Kendall, & Cipolla, 2015). These articles use some concepts of the CNN (convolutional and pooling layers) but instead of computing the probability of belonging to a class as output, the model gives the probability of belonging to a class at pixel level (see fig 2.16) by removing the fully connected layers and using an efficient upsampling approach.

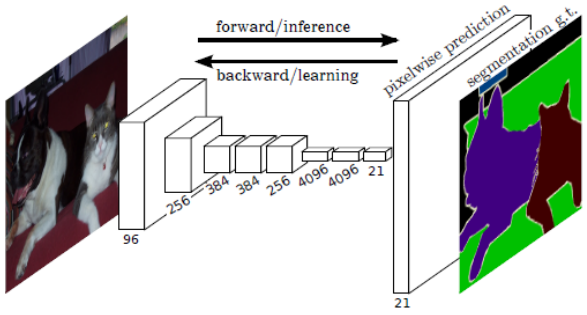


Figure 2.16: Result of a semantic segmentation using a FCN (Shelhamer et al., 2016).

The theory developed manages to link the semantic to the location, the *what* and the *where*. The deep characteristics encode the location and semantic into a hierarchy from local to global. Because of the fully-connected layer, the output in CNN is non-spatial. In these articles, the network computes local to general characteristics by removing the fully connected layers and adding deconvolution layers. Thanks to the upscaling in the deconvolution process, the output is at the same dimension as the input.

In image processing, deconvolution is used as a deblurring process when the quality of the image is limited by the optical properties of the sensors, in particular the *point spread function*.

If the *real/ideal* image is noted *f(x)* and the received image *g(x)*, an alteration between the two can be seen as the application of a deformation filter *h(x)*, representing this point spread function. Convolution is the formalization of the effect of a filter. Thus, it can be written

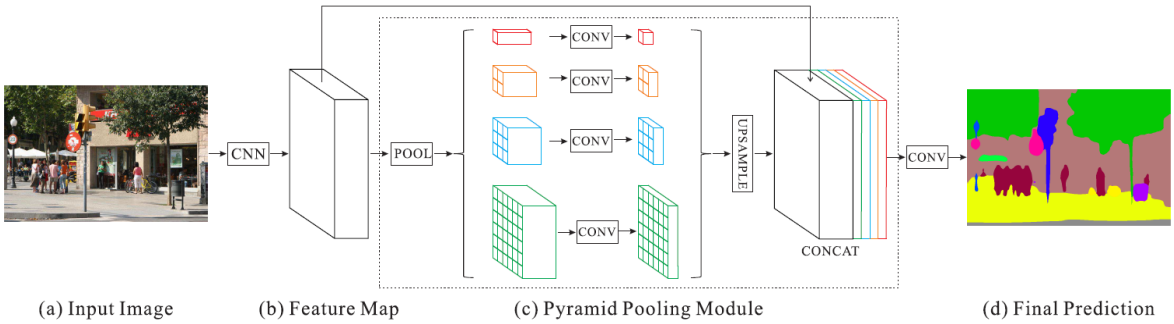


Figure 2.17: Pyramid scene parsing network (Zhao et al., 2016).

$$g(x) = f(x) \otimes h(x) = \int_{-\infty}^{+\infty} f(t)\ h(x-t)\,\mathrm{d}t \tag{2.3}$$

To obtain *f(x)*, the deconvolution product on *g(x)* has to be applied.

$$f(x) = g(x)\ /\ h(x) \tag{2.4}$$

Looking at a CNN model, it can be noticed that the different pooling layers reduce the geometrical dimensions of the image. Hence, an upscaling process has to be defined. In the different before-mentioned articles (Zhao et al., 2016; Shelhamer et al., 2016; Badrinarayanan et al., 2015), each team of researchers uses its own method.

Zhao et al. (2016) developed a very atypical network (fig 2.17). The first layer is a classical convolutional layer producing some feature maps. Then, this result is downsampled by average-pooling at different resolutions (1x1, 2x2, 3x3, and 6x6 in this case). Each resolution is treated in a parallel network where a single convolution with 1x1 filter size over all the features is performed. These results are upsampled by bilinear interpolation to the input image size and added to the first features. The predication map is computed in a final convolution layer.

Shelhamer et al. (2016) first trained the classical Alexnet CNN model (Krizhevsky et al., 2012). Then they discarded the two fully-connected layer and upsampled the previous layer to original image size (fig 2.18). The upsampling process is performed using a non-integer stride deconvolution. If the upscaling is 2 for example, the stride of the convolution process is 1 over 2.

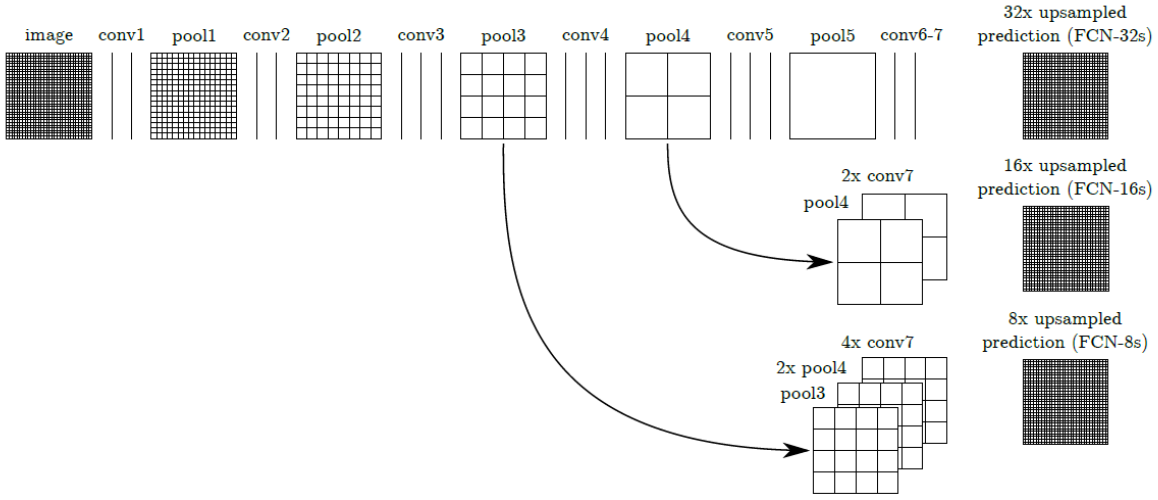


Figure 2.18: Fully convolutional networks for semantic segmentation (Shelhamer et al., 2016). Depending on where the model is cut, we focus more on location or more on semantic. In the first row, the prediction is made by upsampling the last convolutional layer by a scale factor of 32 (since 5 pooling operations have been performed). In the second row, the upsampling is is made one stage sooner, limiting location artifacts but decreasing semantic complexity.

Using Alexnet and only removing the last fully-connected layer, the last convolutional layer has a poor geometry. Even if the upsampled prediction is pixel-wise, the artifacts are present. Indeed, at the end of Alexnet model, several pooling operations drastically decrease the geometrical dimensions of the layer. With high scale upsampling, the quality of location is altered. An idea to improve the geometry is to discard one more stage of layer (comprising a convolutional and a pooling layer). The sooner you cut the network, the smaller the geometrical artifacts are. However, the features learned with more shallow networks are also less complex. There is a trade-off between semantic and location.

Badrinarayanan et al. (2015) used the VGG16-Net (Simonyan & Zisserman, 2014). The network consists in a CNN using max-pooling operations at different stages. In a max-pool operation, the maximum value is chosen in a specific filter size. The novel approach proposed by (Badrinarayanan et al., 2015) is to memorize the location of the maximum to directly use it in the upsampling (fig 2.19). Hence, the upsampled features map is a sparse image (i.e. containing many zeros) which is then processed by a convolutional decoder (fig 2.20).

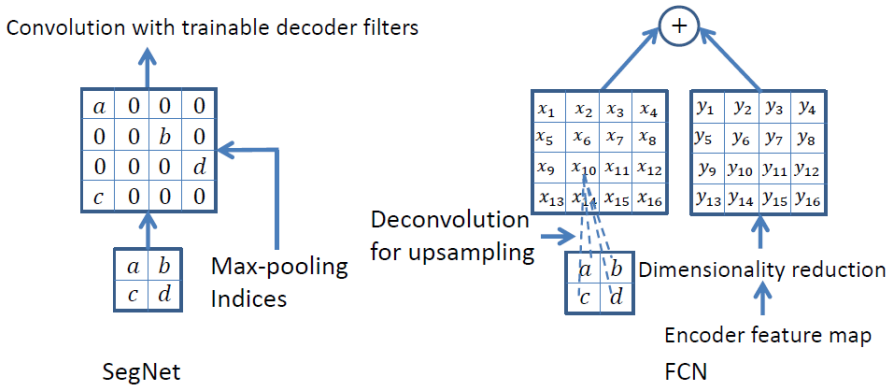


Figure 2.19: Comparison between SegNet upsampling and classical FCN upsampling (Badrinarayanan et al., 2015).

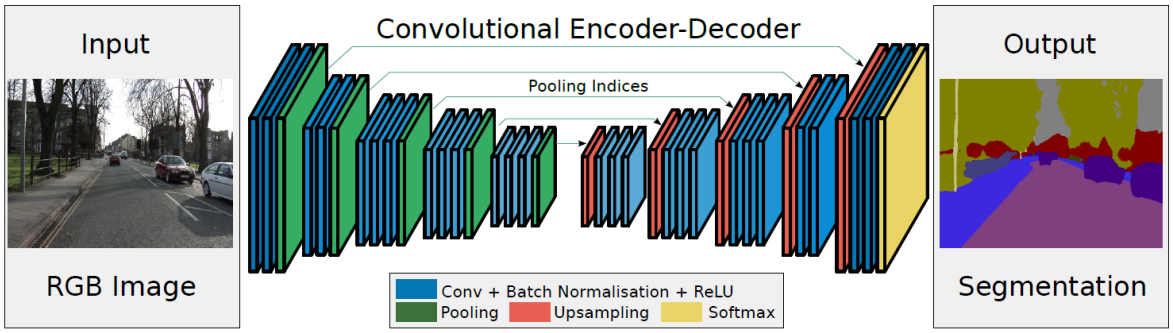


Figure 2.20: Deep convolutional encoder-decoder architecture for image segmentation (Badrinarayanan et al., 2015).

3 | Hypothesis

Deep learning advances proved to be highly efficient in various classical tasks such as image classification. More recently, deep learning techniques are used in fields where no clear answer exists to see if recent advances in artificial intelligence can solve the issue.

In the crater detection task, many proposals tried to solve the problem with a certain exhaustivity and precision. It appeared that unsupervised techniques are not suitable as a solution in many applications. The supervised techniques are more efficient, especially with convolutional neural networks. However, they lack in their ability to extract crater candidates. It results in a mix of unsupervised (to extract candidates) and supervised (to post-classify) CDAs.

The hypothesis developed in this brief is based on the idea that the benefits of deep learning techniques can be applied to the task of crater detection. In this master thesis, we propose a semantic segmentation using deep learning techniques to produce a quick and efficient CDA. The working hypothesis can thus be formulated as :

A fully convolutional neural network is an innovative and convincing model for the detection of lunar craters based on remotely sensed data.

This hypothesis has to be worked and verified as a response to the lack of clear answer in the field of CDA.

4 | Data Description

In 2009, the NASA launched a space probe called *Lunar Reconnaissance Orbiter (LRO)* for the Constellation mission, producing high quality remotely sensed products about the Moon (*LRO Mission Overview*, 2015).

In the year 2011, The *European Space Agency (ESA)* initiated a project called *LandSAfe (Landing Site Risk Analysis Software Framework)* with the collaboration of the geomatics unit from the University of Liège and the institute of photogrammetry and geoinformation from the University of Hanovre, coordinated by *Spacebel* (*LandSAfe: 10/10 pour SPACEBEL et ses partenaires académiques*, 2014). The goal of the project was to create a software able to select, analyse and validate landing sites based on planetary products from the NASA.

4.1 Lunar Reconnaissance Orbiter Camera

Similar to the *LandSAfe* project, we used the data produced by the space probe LRO. This probe is equipped with seven acquisition instruments. In our case, we used the LROC images (*Lunar Reconnaissance Orbital Cameras).* The LROC is composed of a set of three cameras: one with a large field of view (*WAC - Wide Angle Camera*) and two with a little field of view placed in parallel with a small overlap region (*NAC L/R - Narrow Angle Camera Left/Right*) ("LROC EDR/CDR data product software interface specification", 2010). The resolution of the WAC images is 100 meters per pixel and cover regions of 60 kilometers wide whereas the NAC products have a resolution of 0.5 meter per pixel but cover regions of only 5 kilometers (*2*2.5km*). The second major difference is the channels used by the two cameras; the WAC is sensitive to 7 different wavelengths (from 320 to 690 nanometers) when the NAC sensor covers the whole visible spectrum into one channel, called panchromatic. Finally, the satellite is also equipped with a compression system allowing it to send its data with a bandwidth of 100 Mo/sec. The probe and its equipment can be seen in figure 4.1.

The NASA has an open archive of the products of the Constellation mission and allows

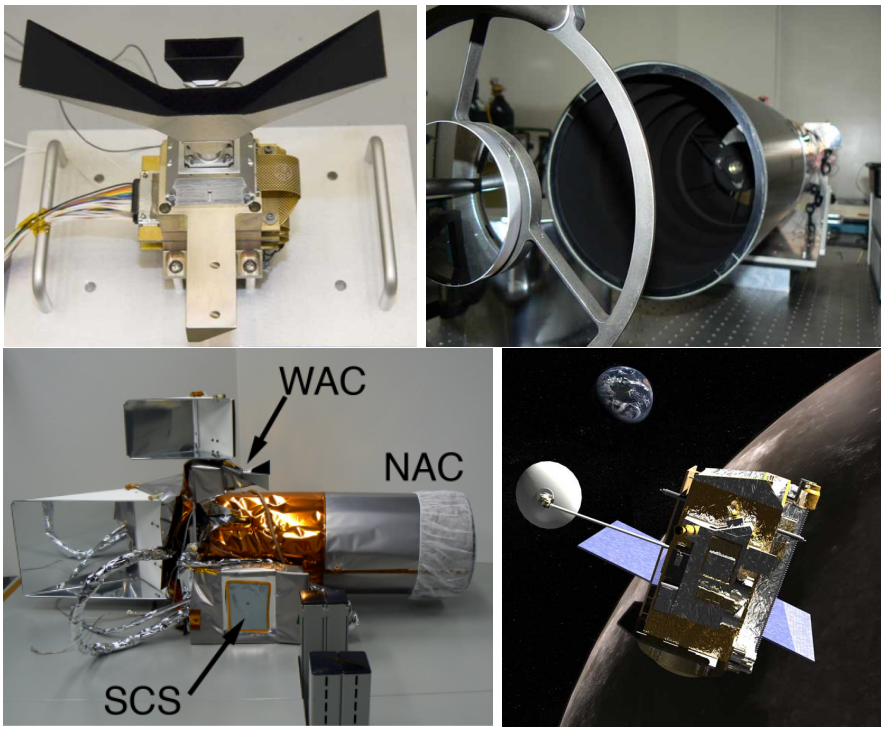


Figure 4.1: Upper left: WAC. Upper right: NAC. Lower left: LROC instrument. Lower right: the LRO space probe ("LROC EDR/CDR data product software interface specification", 2010).

authorized users to download data. The website has a query tool to choose those data according to some parameters. We can cite : lunar time, GMT, planetometric coordinates, emission angle, incidence angle and so on.

Nevertheless, the archive also has an index file with the metadata of all images in a CSV-like format. This file is easily convertible into a MySQL database for example. To each image corresponds a tuple with 83 fields. It enables the user to create more complex SQL queries if necessary.

To reduce the panoramic disturbance and to work at finer resolution, we choose to use the calibrated NAC images[1]. In this context, the calibration is concerned by the radiance of the object. It means that for different images of different scenes, a same digit number indicates an equivalent radiance. Said in other words, the linear relationship between the digit number and the radiance is the same for all the scenes. Finally, the NAC images covers all the surface of the Moon.

[1]The images are not ortho-rectified but as the field of view (FOV) is small (1.21°), the problem of an altered visual aspect of the craters due to the panoramic disturbance can be neglected.

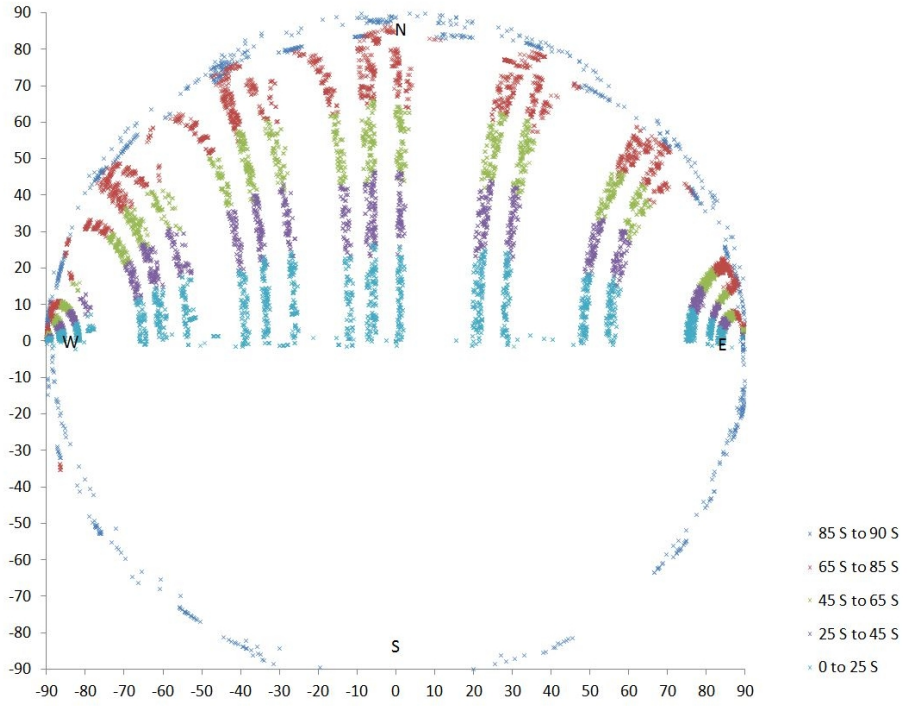


Figure 4.2: Sun azimuth and sun elevation by range of latitude (NAC images -10° to +10°longitude).

For reasons of convenience, we use the images in *.tif* format.

Note that the orbit of the satellite and the solar latitude do not allow all possible illumination configurations everywhere on the Moon. We can however simulate other illuminations conditions by rotating our images.

With a sun at very low elevation, the images are not usable. In fact, the shadows hide a large part of the image. Similarly, when the incidence angle is near 0, the shadows are almost non-existent. The images used here are supposed to have a good overall quality, judged by our expertise. Whatever the chosen images, we will always have on a same image a set of sharp craters and a set of eroded craters, with other possible topographic elements such as lunar rocks for example.

It is possible to build a *skyplot* (4.2). This way, we have a visual representation of the different illumination conditions encountered.

Parallel to these NAC images, the Arizona State University produced lunar digital elevation models (DEM) based on NAC products using stereophotogrammetry, with a resolution of 2 to 5 meters depending on the image quality. Photogrammetric processing needs a human

intervention thus these DEM cannot cover the whole Moon surface. There are currently 388 treated sites. The production of DEM enables the creation of ortho-rectified images.

4.2 Description of the Dataset

The goal of a supervised learning algorithm is to adjust its parameters using a dataset of observations with associated ground truth to let the model learn the input-output relation by an optimization process.

The dataset would consist in a set of lunar craters (positives) and crater-free zones (negatives). The first question to be asked is to know if such a dataset exists about the Moon. This dataset would be useful to learn and evaluate a model.

Some lunar craters databases exist (Salamunićcar, Lončarić, & Mazarico, 2012). We can cite the still growing LU60645GT lunar database, containing 14 923 lunar craters. It currently constitutes the most complete lunar craters database. Unfortunately, it contains only craters with a diameter above 8 kilometers. For the time being, there is still no sub-kilometric lunar database available.

If we look the authors who implement CDAs, some used a restricted test zone with an available dataset to recognize craters (especially in unsupervised methods). The method is questionable in the sense that nothing can argue that the method would work on different scenes (with another illumination condition for example).

On supervised methods, most recent papers focus on a mars craters dataset related to a small region of Mars. This dataset consists in a set of *instances* which are labelled as a crater or not a crater.

This dataset was built by Urbach and Stepinski (2009) using the *High Resolution Stereo Camera panchromatic* image h0905_0000 taken by the *Mars Express spacecraft*. The scene contains many different topographic conditions. It is thus an advantageous place to create an efficient CDA.

The scene is decomposed into six tiles (see fig.4.3). Each one is related to two .csv files (one for the positives and one for the negatives). The file is structured as *x, y and r* where

- **x** : center of crater measured from the left side of image
- **y** : center of crater measured from the top of image
- **r** : radius of crater

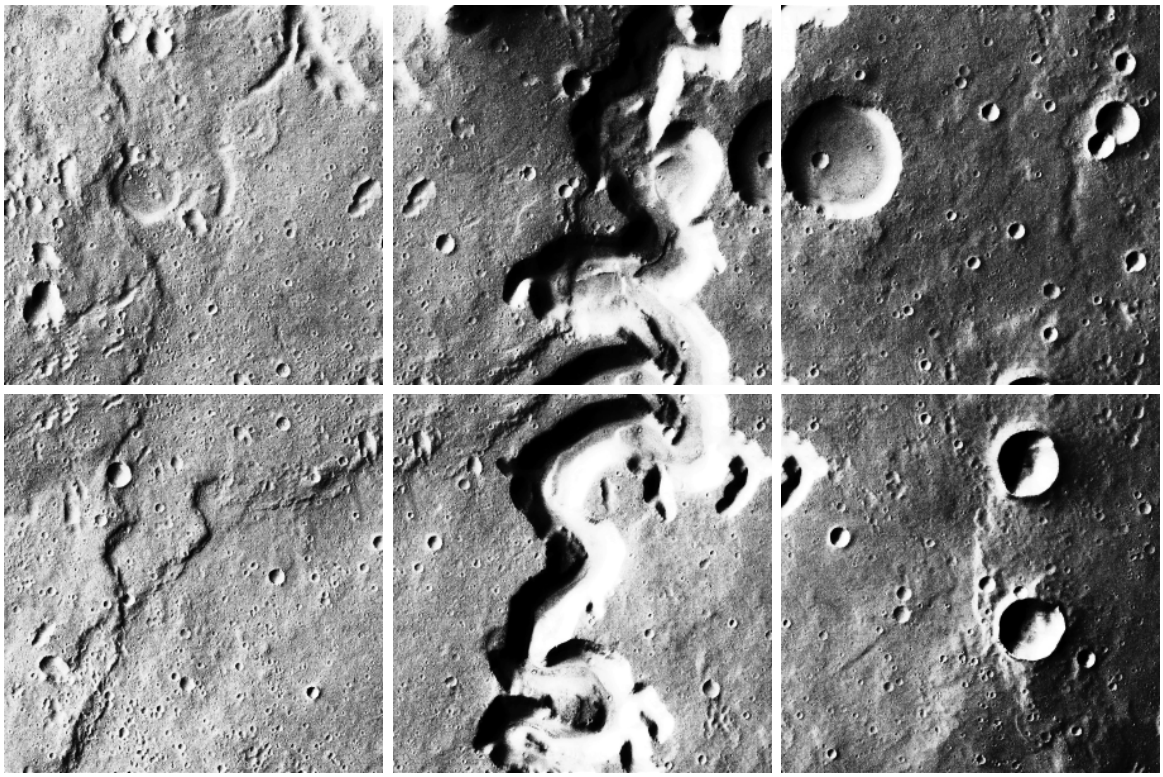


Figure 4.3: The different tiles of the Martian dataset.

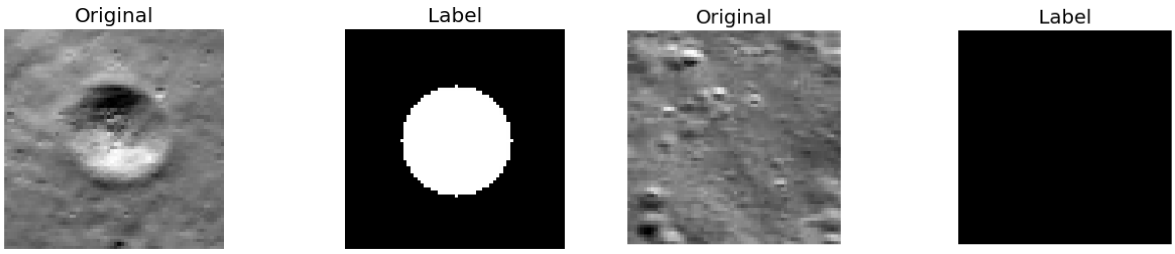


Figure 4.4: Labelling of patches (left = positive, right = negative).

Though we appreciate the efforts made on building such a useful dataset and it them available, these samples suffer from the same imperfections from before, that is to say that only one illumination condition is treaten (only one image is chosen). Even if the scene is decomposed into several tiles, we can assume that the developed models are too much related to the metadata of the scene and the direct application on another image with different illumination conditions may lead to bad results.

Coming back to the Moon, as no dataset exists, we need to manually create these resources. Our efforts would result in an available dataset for any lunar researcher interested in the case of the Moon.

Based on the NAC images, we extract thousands of annotations of different sizes representing either a crater or a crater-free zone. The dataset will serve as examples to let the model learn what is a crater.

Note that craters with a diameter below 8 meters (16 pixels) are hardly distinguishable. Thus, they are neglected in our database and in our quality assessment protocol. In the same way, craters with a diameter larger than one kilometer (2000 pixels) are frequently cut into multiple images (a NAC image is about 5000 pixels large, equivalent to 2500 meters). There is thus a range of craters that are in our interests, between $\oslash_{min}$ and $\oslash_{max}$ (8 meters to 1 kilometer).

In our case, the crater detection must be generalized to the different illumination conditions. It implies to have a large variety of NAC images. To increase the variability of the dataset, we can use some tricks (for example rotating annotations). Also, we need to vary the degree of wear to be able to recognize fresh to eroded craters.

In a CNN, the output to be predicted is only symbolic, with a single value (0 for crater-free and 1 for crater for example). In fully convolutional neural networks, we need to produce an output image whose pixels are the labels to be predicted, thereby mixing the semantic and the location (fig 4.4).

It can be noticed that, on the annotations, small craters are not labelled as crater (they

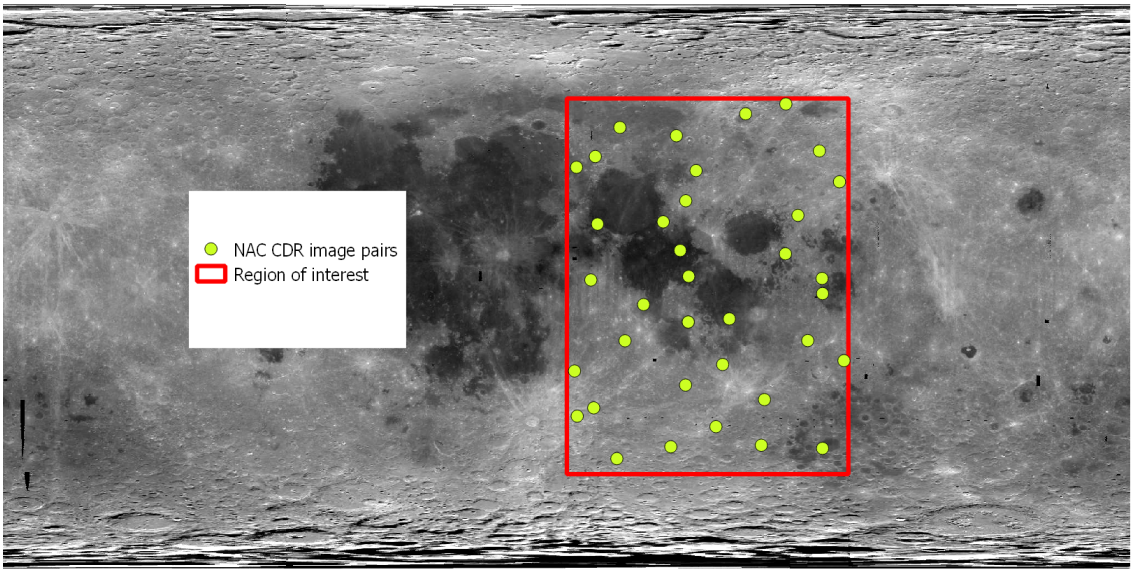


Figure 4.5: Mapping of the pairs of **NAC CDR images**. Note that the images are encoded in *geotiff*. They are already georeferenced in equirectangular cartographic system.

remained black). The reason is that the annotations focus on labelling craters with a diameter of half the annotation size. The rest is neglected. Indeed, an exhaustive annotation would not be feasible, considering the time available to produce the dataset.

Given the orbit of the space probe, we only consider NAC CDR images whose center is comprised between a latitude of -60° and +60° to avoid undesired illumination conditions that affect overall image quality. The longitude does not influence the images in terms of spectral variations thus we only considered longitudes ranging from 0°and 90°[2].

From this geographical area we extracted NAC images by systematic random sampling. The area has been divided in rectangle of 30 degrees in longitude and 10 degrees of latitude. In each sub-area we randomly pick 1 pair of NAC images (left/right). If the expert judges that the overall quality of the images is not sufficient, another random pair of NAC images is chosen. From this process we selected 72 images (fig 4.5).

The range of lunar coordinates and the variability of the acquisition date insure that we meet most of the different cases, adding more complexity to the dataset. The full list of retained NAC images can be seen in the annexes.

From the ortho-rectified images created by the Arizona State University, we only considered

[2]Indeed we were planning to select images from 0° to 360° but the task proved to be more time-consuming that expected.

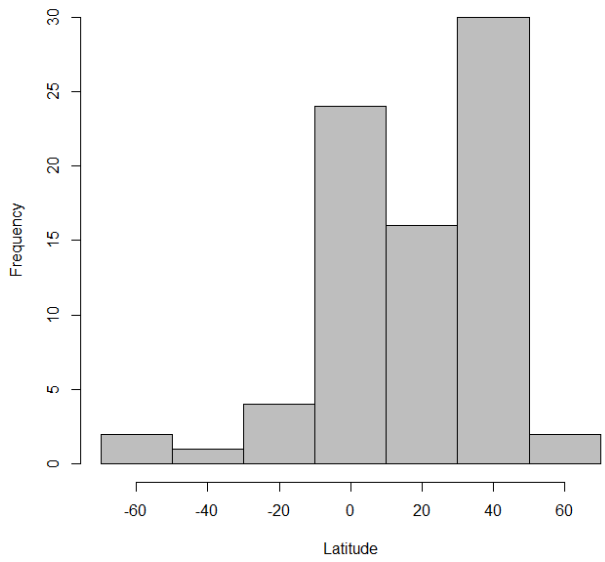


Figure 4.6: Spatial distribution of the **ortho-images** according to their latitude.

the images with a *ground-sampling-distance* (GSD) of 0.5 meter. In addition, some images are not appropriate (very particular topographic structures or incoherent mosaic process). In the end, we choose 79 sites (also listed in the annexes). The location of these sites is mainly between -50°and +50°of latitude (fig 4.6).

Our approach is thus defined with two types of data:

- Case 1 : we have the NAC CDR images.
- Case 2 : we have the ortho-images from the Arizona State University.

For the question of the amount of data, the larger the dataset, the more likely it is representative of all lunar conditions encountered in a scene. However, the overall accuracy of the model increases logarithmically with the number of samples up to a moment where collecting more data becomes almost useless.

As we have to create our own dataset, we need to be economical in time when collecting data. To know if the number of collected samples is representative enough, we can check the stability of the some metrics when evaluating the developed model.

When building a dataset of crater annotations based on remotely sensed data, a sampling protocol is to be described. In the case of a spatial sampling, we have several possibilities (fig 4.7).

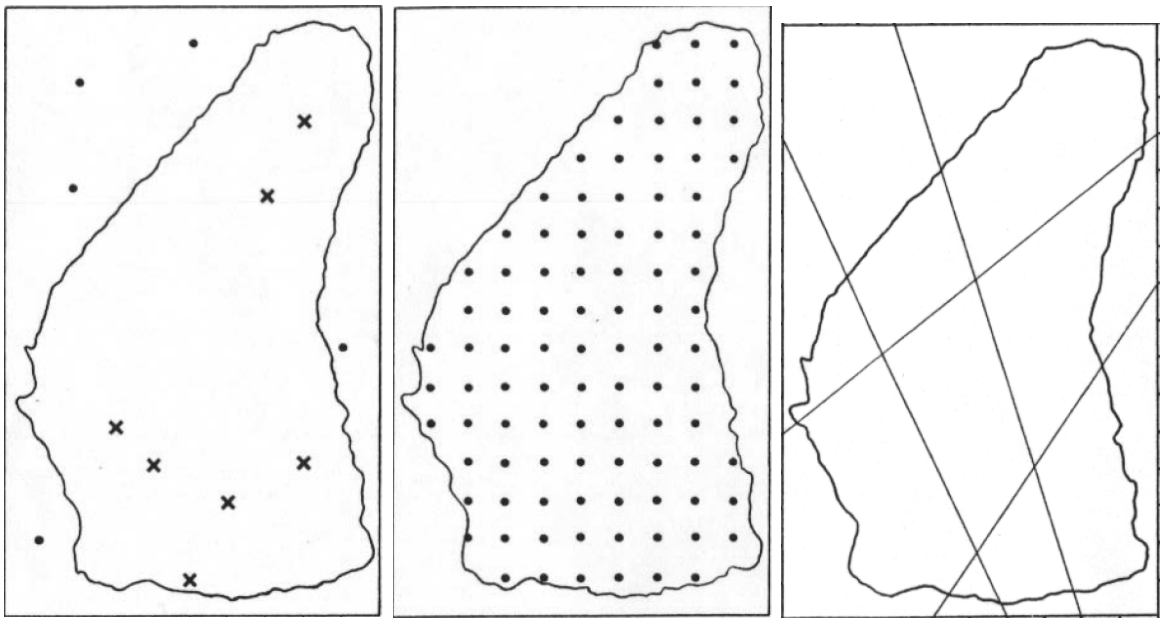


Figure 4.7: Different sampling methods. From left to right : random, systematic and transect sampling (*CATMOG concepts and techniques in modern geography*, n.d.).

The first way to sample a region is the simple random sampling. In that case each point on the region of interest has the same probability to be chosen. The second way to proceed is the systematic sampling. This sampling protocol is realised by superimposing a regular grid on the region. A variant is possible by incorporating a small amount of randomness in the systematic sampling : each vertex of the grid being moved using a small random shift. The last sampling method is the transect sampling. The latter consists in sampling by following random lines crossing the region.

In our case, we had to look at the balance between performance and time-budget dedicated to collect data. For each image, we define 10 equally distributed sub-regions. In each sub-region, we annotate 10 positives (crater) and 10 negatives (crater-free) to preserve priors of 50%. We thus extracted 200 annotations per image. Due to lack of time, not every selected image could be annotated. Nevertheless, we collected in total 11 239 different annotations.

5 | Methods and Developments

The problem to be assessed can be formalized as follow. As input, we have a panchromatic image in tiff format, encoded in 8 bits. The ground sampling distance of the image is 0.5 meter. The output is a set of binary masks whose combination gives the precise segmentation of the craters on the lunar scene.

5.1 Implementation and Computation

Building a deep learning application without any previous work in the field can be an ambitious task. Furthermore, no sub-km lunar crater dataset exists. We thus need an efficient way to produce the needed samples to train and evaluate our model. To build the dataset according to the sampling method developed in previous chapter, we used the Cytomine open-source web-oriented database.

The mathematical formulation of the problem has already been discussed. We can thus write our own code designed to build and train our deep learning model. Nevertheless, some deep learning libraries emerged the last few years to help us creatie our application.

Finally, we would like to benefit from the recent advances in Graphics Processing Unit (GPU) computation. In this context, we used the Compute Unified Device Architecture (CUDA) Deep Neural Network (CuDNN) library to parallelize calculations.

5.1.1 Cytomine

Cytomine is an open-source web-oriented database with a Graphical User Interface (GUI) and a communication interface to deal with very large images. One of the objectives of Cytomine is to encapsulate what is independent of the problem by setting tools for users. In many projects, there is a need of creating a software able to collect and manage datasets. Not only the implementation of these tools can be long and difficult (especially for people

that are not from computer sciences) but also often meets the needs of other researchers. Thus, Cytomine is a solution to get rid of these issues. In the beginning, Cytomine was designed to bio-medical research. Nevertheless, the features of Cytomine can be adapted to other applications. Ours is one example.

The main features of Cytomine are (Marée et al., 2016) :

- Upload : we can upload our large images in the online Cytomine database.
- Organize : we can organize our images in different projects/folders with their own ontology, restriction and so on.
- Explore : Cytomine has a dynamic interactive tool to navigate within the images. It is possible to associate pixels with geographical coordinates.
- Annotate : the main concept of Cytomine is the ontology[1]. It is possible to select a sub-area in an image and assign a label (a *crater* for example). A shape with an associated label is called an annotation. All the annotations are stored and administrated in the online database.
- Share : a database on Cytomine can be shared with other contributors. Each one has his specific permission rights.
- Analyze : given a set of annotations, Cytomine is able to build a model of each label to recognize some objects, using segmentation algorithms already pre-implemented (Marée et al., 2016).
- Extend : we can integrate our own algorithms into Cytomine.

We can extract the cropped images corresponding to the annotations. These operations can be carried out in three ways :

- With GUI in a web-browser.
- Using queries directly in the web-browser's URL.
- Using an application programming interface (API).

Note that all the operations cannot be done only with a GUI. To automate the procedures, the best solution is to opt for an API. Cytomine has a Python-Client-API available on

[1] An ontology is a formal naming and definition of the types, properties and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse (The International Association for Ontology and its Applications, 2017).

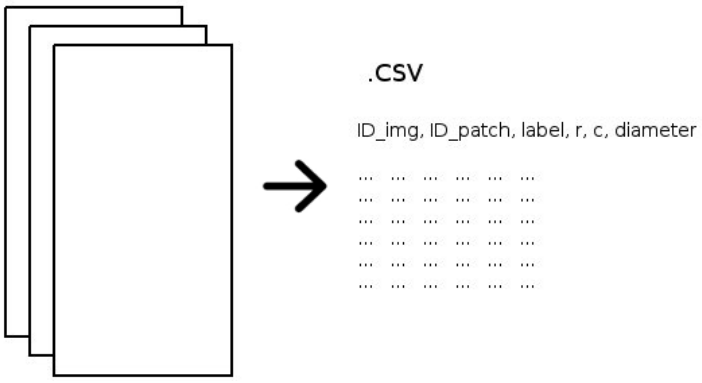


Figure 5.1: Extraction of the entire database in .csv format corresponding to all the annotations on all images.

GitHub[2]. This API enables the user to interact with Cytomine using pre-built functions such as *get_annotation, dump_annotation, add_user* and so on.

All these annotations are reusable in our own codes on a personal computer having a network connection. The connection is established by setting the different parameters related to our Cytomine account.

One advantage of using Cytomine is the reliability of the dataset ; it is easy to check the annotations, remove/correct some objects, select specific images and so on. Moreover, it is possible to add contributors to a project to assess our dataset and even improve it by selecting new annotations on our image set (according to their rights).

By selecting the needed image, it is easy to work with stratified datasets. For example, it is possible to focus on annotations associated to images with a same illumination orientation.

It is also possible to make requests to the database. We can extract the whole database in CSV format (fig 5.1). For each annotation, we have different information such as the Id, the area, the label (called *term* in Cytomine), the centroid position, the original image filename and so on.

However, if any update of the images metadata is done, it makes some operations more difficult. In a late stage of our project, I decided to update the resolution of my images to one unit per pixel to express our coordinate system in terms of pixels instead of geographical units. Nevertheless, the update of images metadata does not affect the previously defined annotations, meaning that the area of the annotations are still expressed using the previous image resolution. The Python client has an update functionality designed to recompute these kind of features. Unfortunately, this new function still has some bugs which does not

[2] `https://github.com/cytomine/Cytomine-python-client`

allow me to use it properly.

To counteract this issue, we need to implement a script that builds the CSV file. The first step consists in retrieving the annotations and images information. These are gathered in JSON data format (see examples in annexes). It contains many attributes and spatial data. In the examples present in annexes, we can see that the resolution of the image is set to *1.0* but the area of the annotation is still enormous.

Then, we write the header of our CSV file. The considered variables are : id, X, Y, radius, label and id_image. Note that the X and Y coordinates are expressed according to the image coordinate system in pixels.

By retrieving the geometry of the polygon constituting the annotation, it is possible to re-compute the area. Then, for each treaten annotation we can write the needed information, separated by a comma.

In total, 11 223 annotations have been created manually, with a prior of 50% for each class (5612 negatives and 5611 positives).

Offline Solution

There exists another way to efficiently produce annotations on images. Antoine Lejeune, a researcher from the Montefiore's Institute, built for our application a dedicated software written in C++ designed to annotate large images.

At current state, the software is able to

- Load remotely sensed products (TIFF/GEOTIFF and pyramid TIFF/GEOTIFF).
- Define directories to store annotation (one per class).
- Zoom in button.
- Zoom out button.
- Original size button.
- Define patch dimensions.
- Browse on the image with arrow keys.
- Define key shortcuts to save annotations to specific directories (for example, when the key button associated to the first term is pressed, the current selection is saved into the dedicated directory).

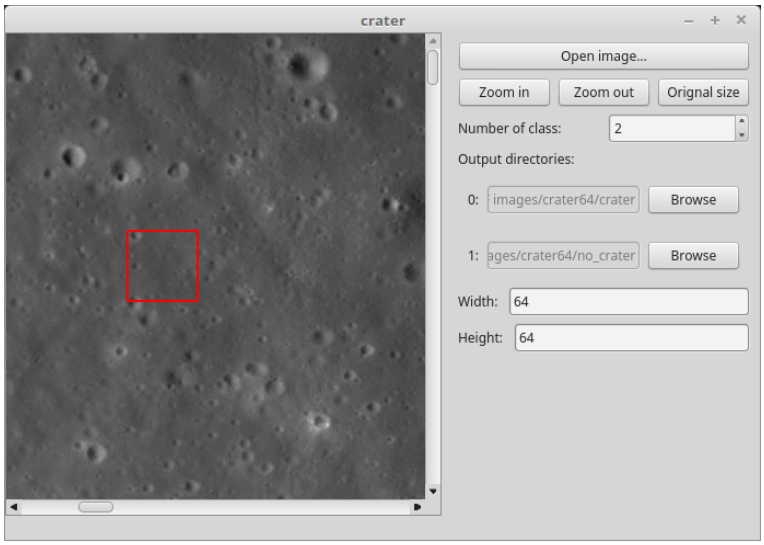


Figure 5.2: Stand-alone software to collect annotations.

- Left click to center the patch to the click location.

When working, the software writes a logfile in a CSV format where each tuple records the same attributes than the ones from Cytomine (namely id, X, Y, radius, label and id_image).

Using the logfile, it is then possible to add the annotations to Cytomine via the Python client.

Both Antoine Lejeune's software and Cytomine are able to produce a huge amount of annotations in a reasonable amount of time. However, when possible, we prefer to use Cytomine thanks to its pleasant GUI.

5.1.2 GPU Computating

As seen in 2.4, the training process and the use of neural networks can be very computationally expensive when building a deep learning model. The fully connected layers implies to have a connection between each neuron of the current layer and each neuron of the next layer.

These operations can easily be rewritten in terms of inner-product between the weights vector and the input vector, with the addition of the vector of biases. These operations can be replaced by a simple matrix multiplication (Oh & Jung, 2004).

$$W = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1N} \\ w_{21} & w_{22} & ... & w_{2N} \\ ... & ... & ... & ... \\ w_{M1} & w_{M2} & ... & w_{MN} \end{bmatrix}$$

$$X = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1L} \\ x_{21} & x_{22} & ... & x_{2L} \\ ... & ... & ... & ... \\ x_{N1} & x_{N2} & ... & x_{NL} \end{bmatrix}$$

$$B = \underbrace{\begin{bmatrix} b_1 & b_1 & b_1 & b_1 \\ b_2 & b_2 & b_2 & b_2 \\ ... & ... & ... & ... \\ b_M & b_M & b_M & b_M \end{bmatrix}}_{L}$$

$$M = W \times X + B \tag{5.1}$$

with :

w_{ij}, the weight between the j^{th} input node and the i^{th} output node

N, the number of input nodes

M, the number of output nodes

x_{ij}, the value of the j^{th} input vector for the i^{th} feature

L, the number of input vectors

b_i, the bias of the i^{th} output node

These kinds of matrix operation is very common in neural networks. We saw the mathematical formulation of a fully-connected layer but it also works with other matrix operations such as activation functions, pooling, convolutional layers and deconvolution layers. If the network consists in more than one layer, we repeat the operations following a chain rule.

GPU was originally designed to rendering. However, pixel shader allows a very effective parallelism compared to a classical central processins unit (CPU). Matrix multiplication is

indeed performed by rendering. For this reason, GPU becomes a powerful tool for parallel computation.

NVIDIA® developed a library dedicated to GPU-accelerated computing : CuDNN. This library allows the user to use its graphic card for computation. CuDNN also implements standard deep learning operations such as pooling, activation functions and so on.

In my project, I had the opportunity to use dedicated GPU to train and run my models. According to the circumstances, several GPU can be used.

- Using my personal computer, our GPU is a NVIDIA Geforce GTX 970 overclocked (+8.5% on based frequencies).
- Using secure shell (SSH) network protocol, we have access to more powerfull GPUs (NVIDIA GeForce GTX TITAN X, Pascal architecture) located in Montefiore Institute.
- Integrated in Cytomine, the model would be able to use the NVIDIA DGX-1, containing 8 NVIDIA Tesla P100.

Depending on the GPU, some memory restrictions are made. With the first mentioned GPU, we have 4Go VRAM. The second one has 12 Go and finally the DGX-1 has 128 Go VRAM. The VRAM is highly used with CuDNN and limits some operations. We need to pay attention to it.

5.1.3 TensorFlow

As described in 2.4, a neural network is a complex architecture. The mathematical formulation has been written and the way the model is trained using well-known techniques.

Though it is possible to recreate an algorithm to create, train and evaluate a neural network, it represents a huge task in implementation. The objective of this master-thesis is not to create an extensive implementation of a deep network. Moreover, our expertise is not sufficient for an optimal implementation.

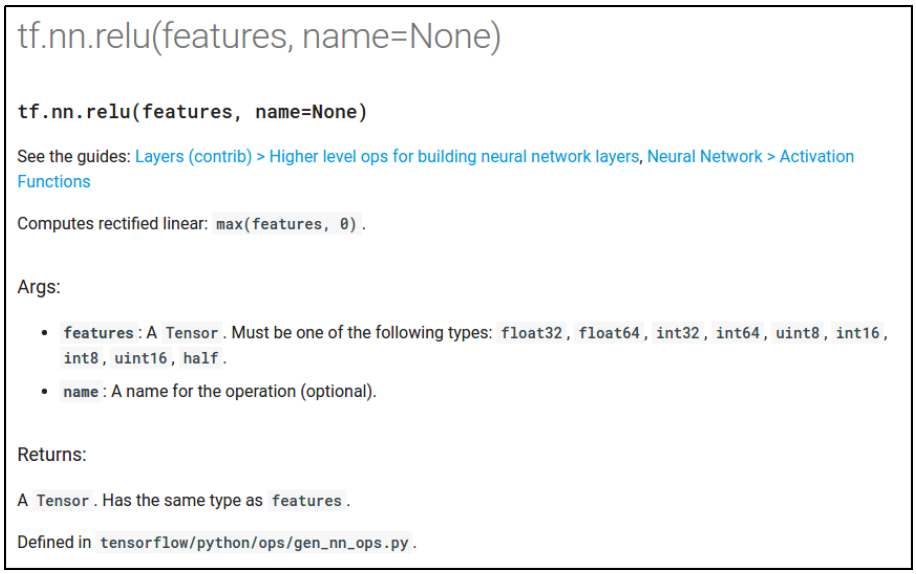
In deep learning coding, there are several open-source libraries that can be used. The most popular are Theano (Python), Caffe (C++), DeepLearning4j (Java), Torch (LuaJIT) and more recently, TensorFlow (Python). Thanks for the rapid growth of TensorFlow (in terms of community, performance, documentation and features), our choice fell on this one.

TensorFlow is a open source library originally developed by the Google Brain research team. Its name comes from the architecture of the data, represented as tensors. Tensor representation enables parallelism programming and thus efficient computation.

TensorFlow is a very efficient library using CuDNN for GPU computation. It has many classes, functions and methods that are imported in the beginning of the code. The convolution functions, the cost functions, the activation functions, the minimization functions and more are already implemented.

TensorFlow works in two times. Firstly, we build the computation graph. We set all the variables. These variables are stored in placeholders. These placeholders are tensors with user-defined shape and type. Then we can define the operation between the variables, namely the different layers, the filters, the loss function and so forth.

After that we run the computation graph and the training process. In this step, the program will run the user-defined operations and training algorithm.


tf.nn.relu(features, name=None)

tf.nn.relu(features, name=None)

See the guides: Layers (contrib) > Higher level ops for building neural network layers, Neural Network > Activation Functions

Computes rectified linear: max(features, 0).

Args:

- features: A Tensor. Must be one of the following types: float32, float64, int32, int64, uint8, int16, int8, uint16, half.
- name: A name for the operation (optional).

Returns:

A Tensor. Has the same type as features.

Defined in tensorflow/python/ops/gen_nn_ops.py.

Figure 5.3: Example of TensorFlow's API's documentation.

5.2 Data Importation

5.2.1 Annotations Importation

The first step of the process consists in downloading the crops of the annotations taken on Cytomine. That task was performed using the Python client. The source code produces two directories : one with the positives (craters) and the second with the negatives (crater-free) (fig 5.4). It is possible to specify what are the images to be chosen (for example, we can separate NAC CDR images from the ortho-images).

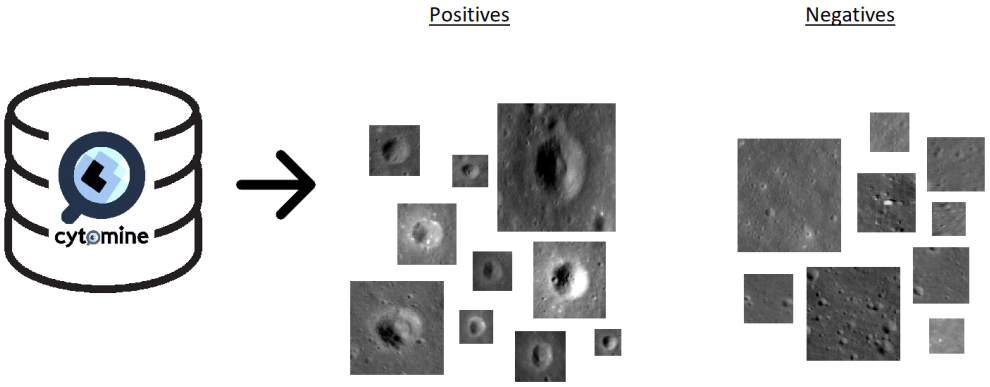


Figure 5.4: Extraction of the dataset into positive and negative folders.

Unfortunately, the Cytomine Python client is designed only to download the smallest bounding box of the annotation. In our algorithm and in geographical problems in a more general sense, we need to extract the surrounding context to create efficient models (for example a bounding box equivalent to twice the size of the annotation). This functionality is not available at current state and we had to modify the core of the Python client to adjust the *dump_annotation* function by adding a parameter *increaseArea* in the function. Also, when using the Python client to download the annotations, some may be corrupted (reasons remain unknown) that we need to remove.

When the annotations are correctly downloaded, the function searches for all *.png* files in specified directories (*.png* and *.jpg* files are still the only decodable format for images in TensorFlow). Each image is resized to a specified dimensions (in our case, the dimensions are 64x64) and added to an images list data structure. The resampling is processed using bicubic interpolation. In parallel we compute to each annotation a ground truth label image (fig 5.5). If the annotation is a crater (a positive), the image label is a centered circle shape filled with positives with a diameter of size half the annotation size. The rest of the image is negative. If the annotation is negative, the image label is entirely filled with negatives

The goal is indeed to produce a model able to recognize a crater with a specific diameter size (about half the annotation size). The assumption used here is that the shape of all sub-kilometric craters are similar enough to be mixed all together. We may wonder why we do not produce a model for all diameter sizes in order to use it to detect all craters on the scene. The main problem is the lack of a huge exhaustive dataset. As we have to build our own dataset, we have to be economical about the time spent to collect data. The trick here is to be able to find a crater of specific dimensions and then applying the model on a pyramid of the scene. By applying the model on each level of the pyramid, all crater sizes can be found (more details on section 5.5.2).

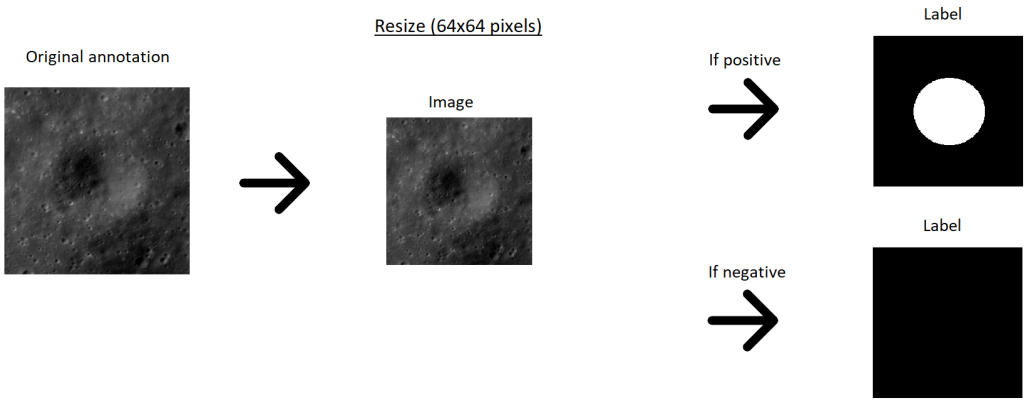


Figure 5.5: Normalization of the annotations size and construction of the image label.

5.2.2 Patches Creation

Given a dataset of crater images and non-crater images, we have to train the model. The model has to be reliable, meaning that for each possible situation encountered on a scene it has to find the best possible label. The golden rule is that the model is able to detect only things that it learned.

This statement is problematic at different levels. The main problem is that if the goal is to detect all craters with diameters between $\oslash_{min}$ and $\oslash_{max}$, we have to show examples of all these diameters. This implies to have patches of very different sizes (from 16x16 to 2048x2048 pixels). This is not feasible in practice and I think the model may have troubles to find the relevant features to understand what is a crater.

Our solution to this is to train the model to a given diameter size. Then the scene is re-sampled to produce a pyramid of resolutions. From each new image, we extract the craters corresponding to the given size.

To simulate different situations encountered on lunar scenes we have to extract from our image dataset uncentered patches to vary the way the model learns features. In fact, centered patterns are a priori knowledge that we do not want the model to know. To do so, we import the different crops of the craters with geographical context. From this image, we randomly draw points from which we extract patches with its associated label (fig 5.6).

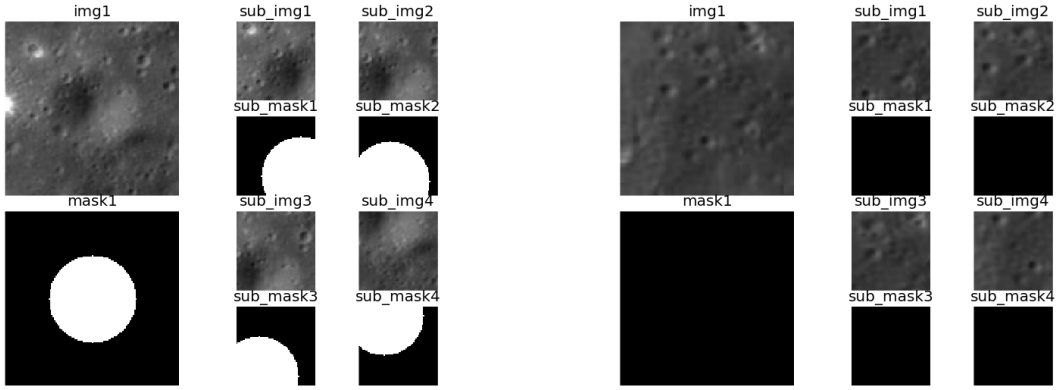


Figure 5.6: Example of the extraction of 4 patches per annotation (white = crater).

To multiply the number of patches and to generate more illumination conditions, we also rotate the patches by angles of +90, +180 and +270°. The use of other angles would imply truncation of the patches to eliminate regions out of the patch. Note that this trick only enables to simulate other sub-solar azimuth angles but not the sun elevation.

5.2.3 Learning Set / Test Set / Validation Set

In any supervised learning algorithm, it is always important to ensure that we have an independent set of data to validate our model. 10% of the entire dataset is dedicated to the validation set (11 133 annotations). However, we also need a test set different from the train set to determine the hyper parameters of the model's architecture. Doing so, we slightly overfit the model because the hyper-parameters are related to the test set.

To counteract this issue, we decided to randomly separate the learning set from the test set with a proportion of 70%/30%. If the number of patches per annotation is 10, that is 100 200 patches that are used, 70 138 to train the model and 30 059 to to test the model.

With these high numbers, it can be statistically shown that priors (meaning the proportion of positives/negatives in the train and test sets) are respected. In 99% of the cases the priors are comprised between 49.5% and 50.5% for the train set and comprised between 49.3% and 50.7% for the test set.

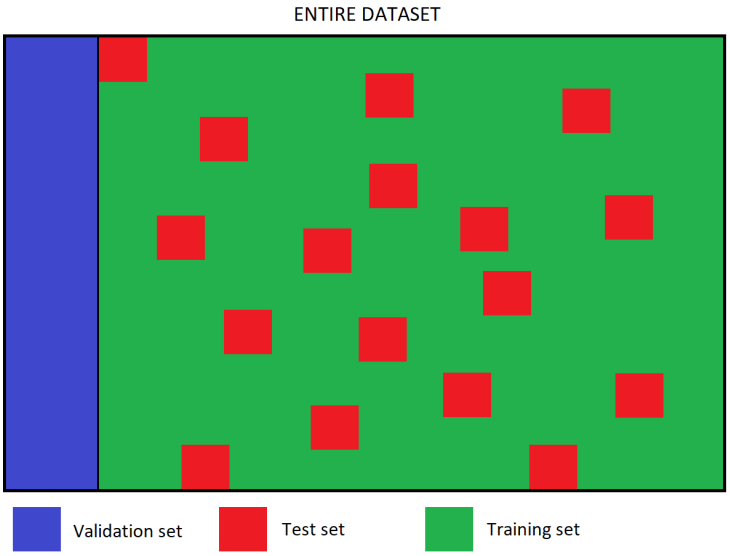


Figure 5.7: Distribution of the dataset in the different sets.

5.3 Description of the Model Architecture

5.3.1 Input Layer

The first step is to import the data. To do so, we create two placeholders designed to receive the input data and the ground truth image labels. These tensors are fed with the previously defined patches that have been stored in the images list with its associated ground truth.

5.3.2 Convolutional Layer

In each convolution layer, we compute a simple linear model with specific weights stored in a filter. By sliding the filter, we produce a feature map.

The parameters of each convolutional layer to be defined are the number of input layers, the dimensions of the filter and the number of feature maps to compute (i.e. the number of filters to train). The parameters are initially set using a truncated Gaussian distribution, with a mean of 0 and a standard deviation of 0.1. Values greater than 2 and lower than -2 are discarded and recomputed. These parameters are stored in a 4D-tensor.

The stride is defined here by a step of 1 for the columns and 1 for the rows (equivalent to an exhaustive search). Also, we use zero-padding, meaning that the outside of an analyzed patch is filled with zeros to ensure that the output of the convolution layer has the same shape as the input layers (fig 5.8).

The fact that the geometry is unchanged throughout convolution operation does not mean that the process is not insensitive to boundary effect. Actually, the greater the filter size, the greater is boundary effect. However, a greater filter size enables the model to make more use of the contextual information. As this geographical context is crucial, in particular in geographic domains, we need to make a good trade-off between boundary effect and contextual information.

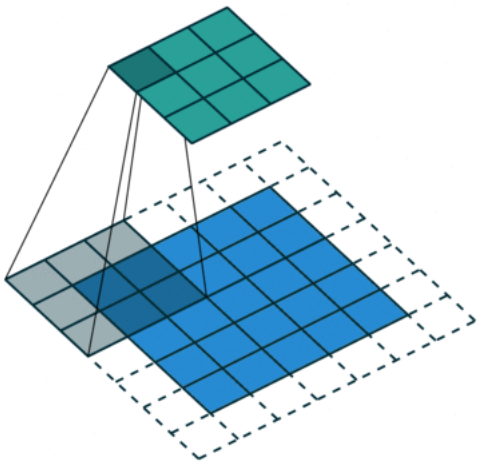


Figure 5.8: In this example, the filter is represented in the upper part (green) and the patch is situated below (blue). To ensure that the output image has the same dimensions that the blue image, we add zeros out of the patch (Hien, 2017).

5.3.3 Activation Layer

After the convolutional layer, we use an activation function. The activation function is a non-linear function whose role is to introduce non-linearity in the network.

In our work we used the ReLU within the hidden layers and a sigmoid in the output layer to produce probabilities.

5.3.4 Pooling Layer

The pooling layer reduces the geometry of the scene by down-sampling the previous layer. This aggregation also diminish the potential over-fitting.

In our case, we uses a max-pool filter (a filter that searches the maximum value) of size 2x2 without overlap (i.e. a stride equivalent to the filter size). Each pool operation diminishes the size of the layers by a factor 2x2.

5.3.5 Deconvolutional Layer

In a FCN, the deconvolution is a way of reconstructing a scene by efficiently upsampling the features maps, contrary to classical CNN that produces a non-spatial response to a given input[3].

With a simple linear or bicubic upsampling, the blur effect seriously affects the quality of the map. With deconvolution, we apply a contrast restoration by computing a inverse convolution operation. The parameters of the deconvolution procedure are weights that have to be determined by our optimization process.

5.3.6 Output

After the last deconvolution layer, we have one features map with the same resolution as the input. This features map is then processed using a sigmoid as an activation function on each pixel. One property of the sigmoid function is to transform any real value into a probability map (see fig 5.9).

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

[3]To semantically segment the images using a CNN, we have to make the model slides through the entire scene (i.e. by the means of exhaustive search). The process is quite long for results similar to FCN models (judged by our expertise). Personal experiments shows that the computation time was about 2500 times longer.

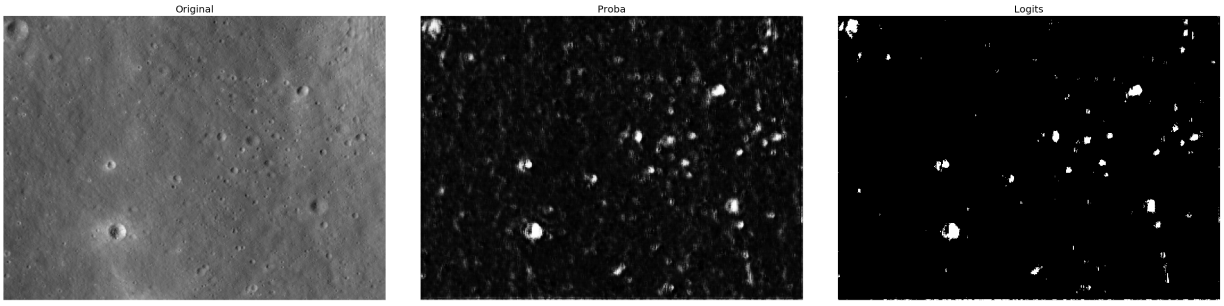


Figure 5.10: Computing a probability map and thresholding the result at a probability level of 0.5.

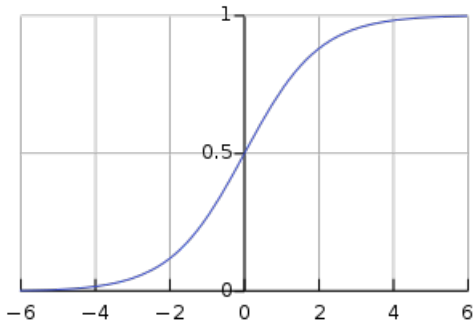


Figure 5.9: The sigmoid activation function.

From this probability map, we can predict the label by using a threshold beyond which the pixel is considered as a crater. Depending on the application, we can use different threshold values. If the goal is to be sure that a crater is indeed a crater (i.e. minimizing false positives), we set a high threshold. On the contrary, if we need to focus on detecting all craters (i.e. minimizing false negatives), we set a low threshold. Fig 5.10 represents a scene whose threshold was chosen to be 0.5.

5.4 Training the Model

In neural networks, the most common way to train a model (i.e. adjust the weights such as we minimize a cost/loss function) is to use a gradient descend algorithm. However, there are different variants of that technique.

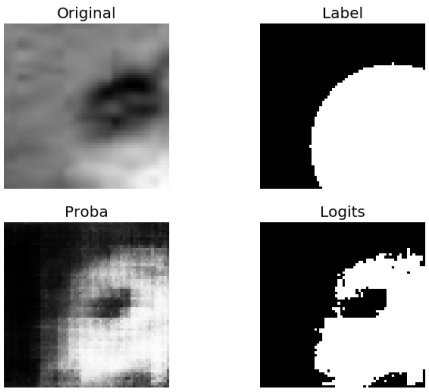


Figure 5.11: Prediction of a patch during the training phase (upper right = ground truth, lower right = prediction).

5.4.1 Loss Function

The loss function is a measure of how far the predictions are from the actual labels.

The goal of a machine learning algorithm is to minimize that loss (most of the time with gradient descent-based algorithm in deep learning). In the end, if convergence is acquired, the best weights are found.

In the case of our work, the model tries to predict the probability of belonging to a label for all pixels for each training patch (see fig 5.11). Based on these predicted probabilities, the algorithm computes the pixel-wise loss and adjusts its weights.

The value of the loss depends on the function used to calculate that metric. One common loss function used when computing probabilities comes from information theory, called the cross-entropy function.

$$H_y(\hat{y}) = -\sum_i \left(y_i * \log \hat{y}_i + (1 - y_i) * \log\left(1 - \hat{y}_i\right)\right) \tag{5.3}$$

Where $\hat{y}_i$ is the predicted probability of the i^{th} pixel and y_i is the true label of the i^{th} pixel (1 for crater and 0 when not a crater). When looking at the equation, we are glad to notice that true detections do not increase the loss. If n_p represents the number of pixels which are likely to be correctly labelled as crater (true positives) and n_n the number of pixels that are likely to be correctly labelled as non-crater, we have

$$\text{True positives :} \qquad n_p * \big(1 * \log(\approx 1) + 0 * \log(\approx 0)\big) \approx 0 \tag{5.4}$$

$$\text{True negatives :} \qquad n_n * \big(0 * \log(\approx 0) + 1 * \log(\approx 1)\big) \approx 0 \tag{5.5}$$

Using a sigmoid as an output estimator of probabilities, the problematic case where the predicted probability is zero does not exist ($\lim_{x \to -\infty} sigmoid(x) = 0$).

5.4.2 Gradient Descent Algorithms

The principle of *gradient descent* is to minimize a specific loss function (cross-entropy or least squares for example) computed on a model with parameters θ with a specific training set. To do so, we update the parameters to the opposite direction of the gradient of the loss function with a particular *learning rate*.

The *learning rate* gives an indication of the convergence speed. A high *learning rate* enables the model to train faster. However, a high learning rate can miss some optimum in the N-dimensional loss function curve (where N is the number of parameters to fit). The lower the *learning rate*, the higher the chances of finding an optimum, but the slower the train process.

Problems arise when the slope of each parameter is highly different. In those cases the rate of change is slow due to the rapid change in slope direction with regards to some parameters. To counteract this problem, we allow the model to converge faster in some directions, introducing a *momentum*.

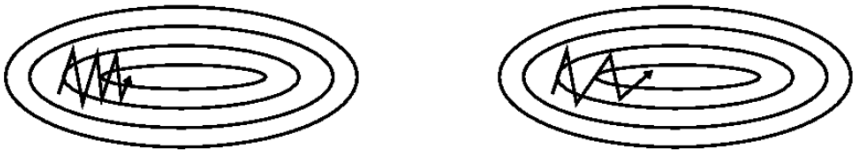


Figure 5.12: Example of the training process without momentum (left) and with momentum (right).

Adagrad and *Adadelta* are algorithms that adapt the gradient descent to the frequencies of the parameters (in cases of some input variables are missing). Frequent parameters are adjusted with smaller updates since they are supposed to be updated more frequently.

Finally, the *Adaptive Moment Estimation (Adam)* performs all of previous considerations by computing the estimated first two moments of the model parameters. Empiric results

show that *Adam* is very powerful optimizer that can adapt to most circumstances. That's the reason why we decided to use it in our master thesis.

5.4.3 Variants of Gradient Descent

As previously said, the *gradient descent* computes the gradient of a typical loss function according to all the parameters. There are three ways to compute the gradient of the loss function :

- Batch gradient descent.
- Online/Stochastic Gradient Descent (SGD).
- Mini-Batch gradient descent.

In *batch gradient descent*, the cost function is computed for the whole training dataset.

To take the notations of S. Ruder (2016), if θ is the vector of parameters, $\bigtriangledown$ the gradient notation, J the loss function and η the learning rate, the new parameters are computed by

$$\theta = \theta - \eta . \bigtriangledown_{\eta} J(\theta) \tag{5.6}$$

That operation is performed several times, either with a fixed number of iterations or up to a specific amount of loss.

Computing the cost function on the entire dataset can lead to very slow training. Moreover, some memory limitations do not allow the use of very large training dataset in one pass.

The *SGD* is performed using only one sample of the training dataset at once. It enables fast training, but often requires a significant larger number of iterations to achieve satisfying results. The variance of the loss is high, enabling the models not to be stuck in local minima. However, the high variation of loss at each step can lead to miss some optimum weights.

The *mini-batch gradient descent* is a trade-off between the two. At each iteration, the cost function is applied to a subset (also called batch size) of the entire dataset. Mini-batch gradient descent is most common way to train a neural network.

In our work, memory limitations forces to neglect batch gradient descent. Also, the sudden jumps at each iterations do not allow us to use SGD. We thus use the mini-batch gradient

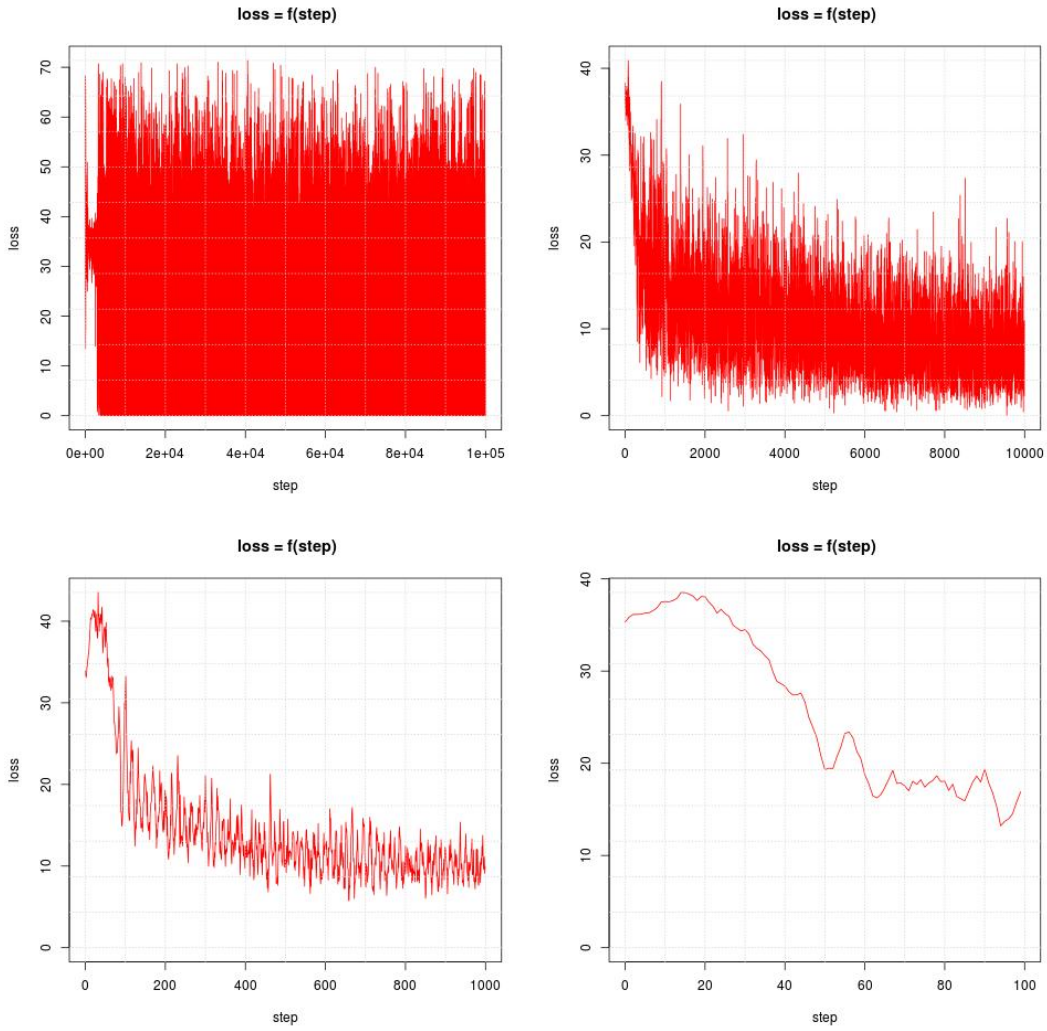


Figure 5.13: Loss as a function of the iteration step.
Upper-left : Batch-size = 1 (equivalent to SGD), number of steps = 1 000 000.
Upper-right : Batch-size = 10, number of steps = 100 000.
Bottom-left : Batch-size = 100, number of steps = 10 000.
Bottom-right : Batch-size = 1000, number of steps = 1 000.

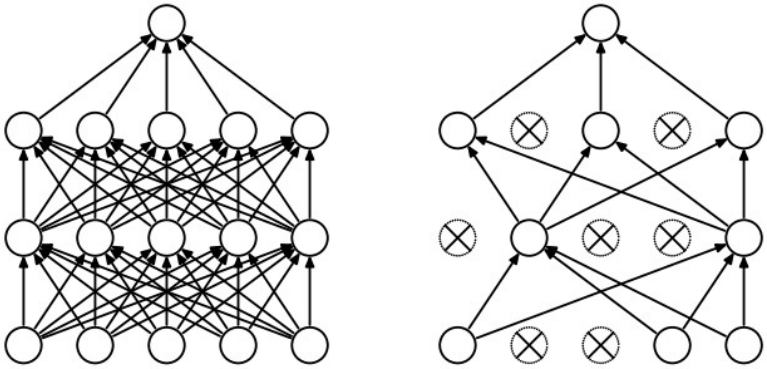


Figure 5.14: Standard neural net (left) and after applying dropout (Srivastava et al., 2014).

descent. The batch size varies the variance of the loss and the number of steps to obtain convergence (see fig 5.13).

The smaller the batch size, the smaller the computation time of each step. Nevertheless, the number of steps is bigger. We can notice that an optimum batch size is comprised between 50 and 200 patches in terms of total computation time.

5.4.4 Dropout

The dropout is a way to prevent overfitting. In large neural networks, there can be millions of neurons. To avoid some neurons to co-adapt too strongly on the train set, we can randomly drop some neurons in the training process (fig 5.14).

Working with CNNs, we traditionally use the dropout on the fully connected layers. However, Srivastava et al. (2014) claim that dropout can be applied on any layer, including convolution and deconvolution layers.

Nevertheless, when we apply dropout on our networks, we notice a decrease of overall accuracy, whatever the degree of dropout (the more present is the dropout, the bigger the loss). This problem is often met by researchers in deep learning but we still let the possibility to use it, even if we did not employ it in our project.

5.4.5 Penalization

The penalization is a way to penalize too large weights in a supervised learning algorithm. Here we used L1 penalization.

The idea of L1 penalization is to add a constraint in the loss function. This constraint is equal to the sum of all absolute weights, multiplied by a λ factor. The greater the λ, the greater the importance of the penalization. This technique is used in many machine learning optimization algorithms to prevent overfitting.

$$\text{Penalization term} = \lambda \sum_{k} |w_k| \tag{5.7}$$

Where k covers all the weights of the model.

However, when we apply L1 penalization, the accuracy of the model decreases. The greater the λ, the smaller the accuracy. The best way to train the model is simply to set λ to zero.

5.4.6 TensorBoard

TensorFlow has an event listener functionality that enables us to visualize the network, to interact with it, to display the history of some metrics (for example the loss at each training step), the weights of the filters trained by the model and so on. This set of features is called the TensorBoard.

In the code, it is possible to serialize the data into a logfile that TensorFlow modifies to build an html file. That html file can be opened in any web browser.

A deep learning model can be quite complex. TensorBoard is useful to better analyze and debug the code. Each part of the code can be named, it is possible to group some layers in order to make the model visually clearer.

5.4.7 Saving Parameters

A neural networks-based model can be quite complex and saving its parameters in order to load them for any type of use is not an easy task for a programmer. Fortunately, TensorFlow developed some functions to save the parameters as checkpoints and restore them when desired. These checkpoints are stored in binary files in proprietary format.

5.4.8 Considered Models

There may have different models to be considered : many hyper-parameters are indeed possible.

The number of layers and their arrangement are important. The more the number of layers, the more the model is able to encode deep characteristics. Also, it makes the model more complex and introduce the problem of vanishing gradient[4]. In addition, we can insert several convolution layers between each pooling operation. The deconvolution can be processed in one or in multiple steps. Moreover, the network can be symmetrical with as many convolution layers than deconvolution layers. Many possibilities exists.

In our work, we built 24 models, each having different possible hyper-parameters (number of layers, number of features per layer, the filter size per layer and so forth). Each model has a different amount of parameters to determine.

The main hyper-parameters that we considered are :

- The number of stages : 1, 2 or 3.
- The number of convolution layers at each stage : 1 or 2.
- Presence of a pooling operator before the first deconvolution layer : 1 or 0 (resp. yes or no).
- Symmetry of the network : 1 or 0 (resp. yes or no).

The number of stages will influence the depth of the networks. In each stage, we add new parameters to train. The pooling layer between each stage reduces drastically the number of parameters by geometrical aggregating the layers. Moreover, this reduction of too located information enables to reduce overfitting effects.

Having two convolutional layers in one stage enables the model to extract more complex features. However, more complexity means more potential overfitting and an increase of the number of parameters.

The presence of a pooling operator before the first deconvolution is subject to debate. This spatial aggregation enables to decrease by a factor of 4 the number of parameters but also decreases the location precision without adding any semantic aspect.

The secondary hyper-parameters are :

- The number of features computed at each convolution layer
- The kernel size at each convolution layer.

[4]When applying gradient descent algorithm, we update the parameters according to the opposite direction of the gradient of the loss function. On the last layers, this update is quite fast. On the first layers, the convergence can be much longer. The deeper the network, the longer the training time.

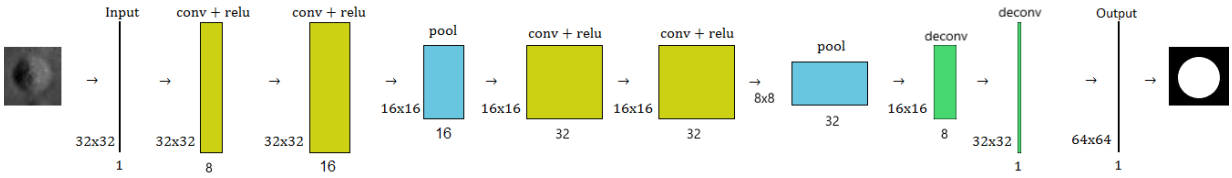


Figure 5.15: Example of one developed model.

Unfortunately, TensorFlow does not allow to change the major hyper-parameters. We need to build each of these models in separate python codes. For the minor hyper-parameters, these are just variables that can be easily changed.

To represent a model, I will use the following figure (5.15). In that example, the input is the patch to be analyzed. To make the visualization more convenient, we can omit one geometrical dimension. The input only contains 1 panchromatic layer and its dimensions are 32x32 pixels. In the first convolutional layer, we decided to extract 8 features meaning that we have 8 filters to train. We also use a ReLU as activation function. In the second convolutional layer, we produce 16 features based on the 8 previous featuresmaps and we use a ReLU as an activation function. After that, we perform a first max-pooling operation, reducing the dimensions of our 16 features maps from 32x32 to 16x16 pixels. Then, another stage of two convolution layer (+ activation function) and a pooling operation is applied. At this stage we now have 32 features maps of size 8x8. The features represent the deep characteristics to classify the pixels. To up-sample the features maps, we use 2 deconvolution layers in a row with each a scale factor of 2. Finally, we use the sigmoid function to compute pixel-wise probabilities and we round the result to associate at each pixel a label. The representation of the other models are available in the annexes.

Note the the set of hyper-parameters (major and minor) influences the numbers of parameters to determine. In each layer, we train several filters from a certain amount of inputs to a certain amount of outputs. The number of parameters in each layer is equal to the the size of the kernel multiplied by the cartesian product of the number of inputs and the number of outputs. Table 5.1 summaries the number of parameters of model *3_2_1_1*.

5.5 Analysing a Scene

Once the model is calibrated and able to correctly recognize a crater, we can take a larger scene containing an important number of craters and determine their positions.

Table 5.1: Number of parameters of the model *3_2_1_1*.

Layer Name	Number of parameters	
conv. 1. 1.	(11*11+1)*1*8	= 976
conv. 1. 2.	(9*9+1)*8*8	= 5 248
conv. 2. 1.	(7*7+1)*8*16	= 6 400
conv. 2. 2.	(5*5+1)*16*16	= 6 656
conv. 3. 1.	(3*3+1)*16*32	= 5 120
conv. 3. 2.	(3*3+1)*32*32	= 10 240
deconv. 3. 2.	(3*3+1)*32*32	= 10 240
deconv. 3. 1.	(3*3+1)*32*16	= 5 120
deconv. 2. 2.	(5*5+1)*16*16	= 6 656
deconv. 2. 1.	(7*7+1)*16*8	= 6 400
deconv. 1. 2.	(9*9+1)*8*8	= 5 248
deconv. 1. 1.	(11*11+1)*8*1	= 976
Total		66 280 parameters

5.5.1 Restoring Parameters

Before running the computation graph, we need to restore the parameters we trained in the training process. We only need to be coherent between the graph used in the training process and the graph used to analyze a scene (i.e. using the same hyper-parameters).

5.5.2 Application of the Model

The crater detection cannot be done in one operation. After importing any scene, different steps are executed, including pyramid scene processing, image tiling, scene reconstruction and several post-processing including mathematical morphology.

Images Pyramid

The model is calibrated to a small range of crater sizes. In our case, it is able to recognize craters of about 32x32 pixels. Empirical results show us that craters dimensions ranging from 16x16 to 64x64 pixels are also recognized.

In order to find smaller craters, we need to up-sample the scene. Similarly, we down-sample the scene to find bigger craters. We decided to create a pyramid of resolution with bicubic interpolation with a scale factor of 2 between each level. The idea is thus to build a pyramid of images, each treated independently. Considering our observations, the scale factor of 2 seems to be reasonable.

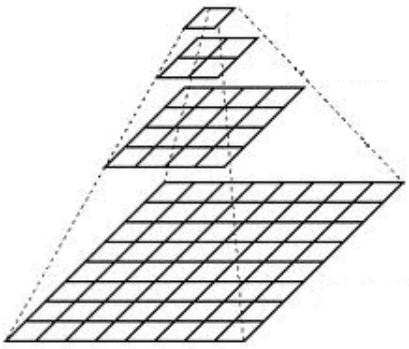


Figure 5.16: Hierarchical representation of a pyramid in image processing, with a scale factor of 2 between each level (Bradski, 2000).

However, if we try to up-sample too small elements on a scene, the blur effect becomes too important and we may not be able to recognize if whether of not the pixel belongs to a crater (even with expert eyes). Thus we have to set a minimum diameter size below which we do not consider the crater detection. Empirical pre-results show that below a diameter of 4 meters (8 pixels), we cannot ensure the reliability of the process.

Similarly we have to set a maximum diameter size. This parameter is not related to any image processing artifact but to the acquisition scene. Each NAC CDR image is about 5000x50000 pixels, meaning that diameters bigger than half the width of the image is likely to be cut on the edge of the scene. We decided to set the maximum diameter size to 2048 pixels (about one kilometer).

To summarize, we have the following layers to analyse :

Scale Factor	Diameter (in meters)
0.25	4x4
0.5	8x8
1	16x16
2	32x32
4	64x64
8	128x128
16	256x256
32	512x512
64	1024x1024

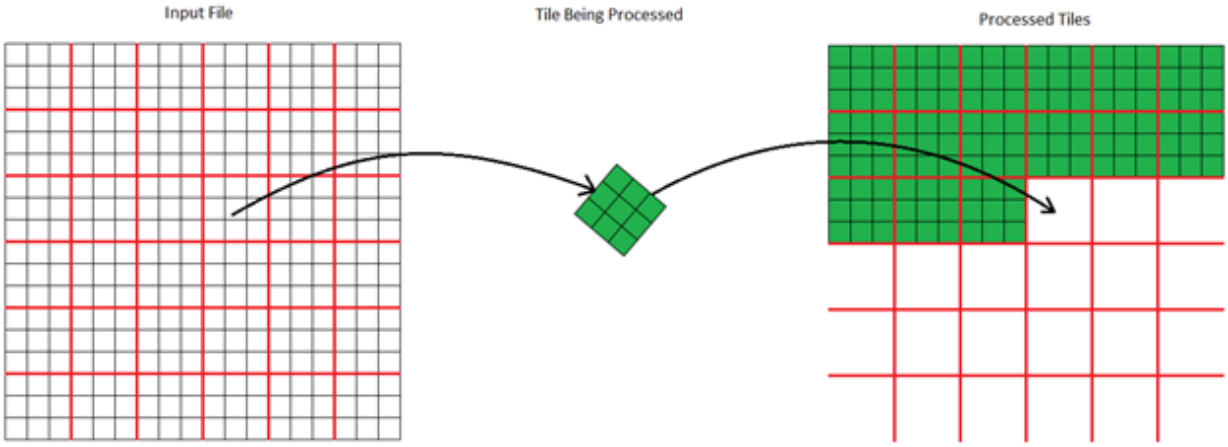


Figure 5.17: Image processing using tiles (Geomatics, 2016).

Tiling Images

The hardware limitation forces us to use image tiling to apply our model. In tile processing, we extract from a large scene some sub-scenes that are processed one after the other. Note that to avoid boundary effects inside the scene when we reconstruct the image, we apply the model on each tile with a small overlap between them. In the project, we defined tiles of size 1000x1000 pixels (at pyramid level) and an overlap of 16 pixels (i.e. half the path size). Note that if the patch size was greater, the overlap should be more important too.

Computing Probabilities

The application of the model on the image produces a probability map. This probability is transformed into a label by setting a threshold beyond which the pixel is considered as belonging to a crater (see section 5.3.6).

5.5.3 Mathematical Morphology

The classified image contains isolated predictions creating some noise. A crater consists in an ensemble of joined pixels labeled as crater whose shape is circular with a diameter of about 32 pixels. Thus, isolated pixels labeled as craters may be false detections An *opening* operation eliminate this artifact.

The opening results from cascading an erosion and a dilation with the same structuring element (i.e. a kernel in computer vision) (Van Droogenbroeck P., 2016):

$$X \circ B = (X \ominus B) \oplus B \tag{5.8}$$

The opening of a set X by structuring element B is the set of all the elements of X that are covered by a translated copy of B when it moves inside of X.

Similarly, a crater-free pixel inside a set of crater pixels forming a circle can be considered as a false negative. The idea is to use a *closing* operation to fill the gaps in crater objects.

A closing is obtained by cascading a dilation and an erosion with a unique structuring element (i.e. a kernel in computer vision) (Van Droogenbroeck P., 2016):

$$X \bullet B = (X \oplus B) \ominus B \tag{5.9}$$

These morphological operators are already implemented into the OPENCV library (Bradski, 2000).

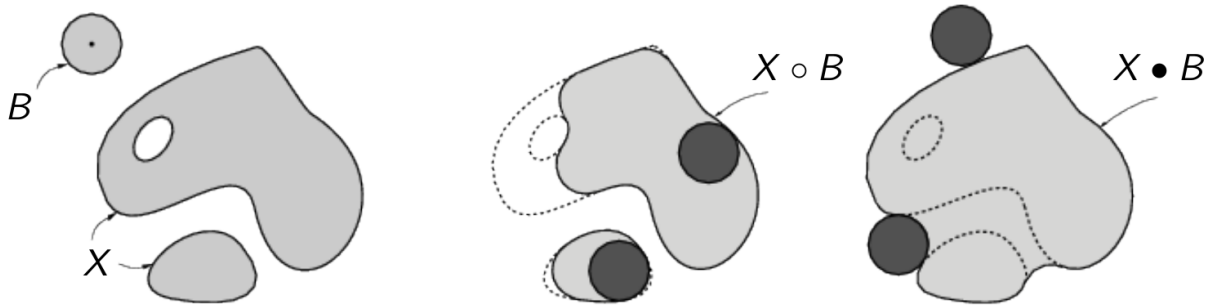


Figure 5.18: Application of the opening of the set X by structuring element B (center). Application of the closing of the set X by structuring element B (right) (Van Droogenbroeck P., 2016).

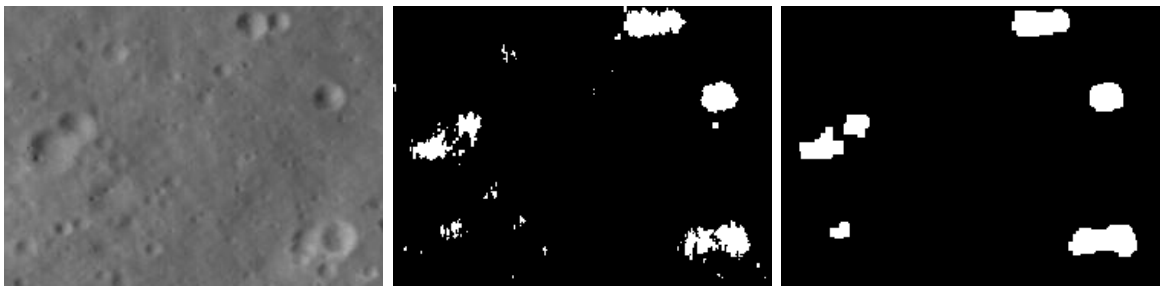


Figure 5.19: Application of mathematical morphology to clean a processed image.

5.5.4 Object Extraction

To extract objects on a image, we need to visualize our binary mask as a graph image where each crater pixel is a node and an edge between two nodes exists only if two crater pixels are neighbors (8-connectivity).

Based on this graph, we use a connected-component labeling algorithm to create our objects (Dillencourt M., Samet H. and Tamminen M., 1992). This task was performed using *scipy* Python library which has a package specialized in multi-dimensional image processing.

Then we compute the total area of each component and we remove those that are not relevant. The concept of relevance used here is related to the fact that our model is supposed to be able to detect craters with a diameter ranging from 16 pixels to 64 pixels (in each single pyramid level). Therefore, the conditions to keep a crater are

$$\pi 8^2 \leq Area \leq \pi 32^2 \tag{5.10}$$

With the area expressed in square pixels.

Note that these conditions can be a little relaxed for two reasons :

- The smallest craters present in the range of valid diameters may not be entirely recognized. The area of a correctly recognized crater might be below the lower bound of the condition.
- If some valid craters are arranged one next to the other, they are likely to merge with our mathematical morphology.

The objects can be qualified in terms of image position and radius. The position is related to the centroid of each connected-component and the radius is an aggregation of the total area covered by the object, supposed to be a circle. Given the area, the radius is computed using

$$Radius = \sqrt{\frac{Area}{\pi}} \tag{5.11}$$

These craters are not directly geolocated but associated with its NAC CDR or ortho rectified image metadata, their geographical position can be retrieved. The crater object is thus defined by a tuple whose fields are : *id_crater, id_image, X, Y, Radius.*

5.5.5 Reconstruction of the Scene

At each stage of the pyramid, craters with a specific diameter are detected. We thus have a list of masks with different resolutions. Each image can be resampled to produce geometrically coherent maps of craters. Note that with classified images, the re-sample is executed with nearest neighbor interpolation.

With these images, it is possible to produce a map of all crater types. The result is nothing but a *OR* operation between the different masks (see figure 5.20).

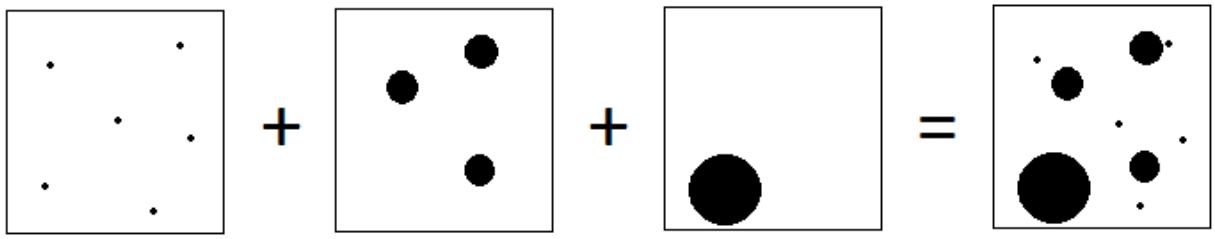


Figure 5.20: Combination of the different crater masks.

Note that the reconstruction may be subject to errors. Firstly, if the down-sampling leads to rounding some dimensions, the up-sampling in the reconstruction may be incoherent, meaning that the superposition of the different layers is inconsistent. Secondly, the overlap is different at each stage of the pyramid. If the overlap between each tile is 16 pixels, We have an overlap region out of the whole set of tiles which is different when re-sampled to the original resolution.

Thus we need to be careful when reconstructing the mask at original resolution to be perfectly coherent.

Since some large craters can hide smaller craters, we can let the different crater masks separated.

5.6 Integration to Cytomine

Cytomine was mostly used to handle our large remotely sensed data online and to efficiently generate a dataset. However, it is also possible to add some python code to Cytomine in the form of *jobs*.

The job can be integrated into Cytomine. That way, the computations are done on servers (meaning using the DGX-1 of the Montefiore's Institute). It has the advantages that anybody can apply the methods developed in our project. However, the integration process can be long and tedious. It is recommended to integrate the source code when the job is done (or in advanced stage).

The job can also be integrate on the client side. That way, the server asks the user to produce results that are integrated into Cytomine. This method is more suitable when frequent changes are made (test/adapt coding). Nevertheless, the power of personal desktop is much lower.

However, since it is not the main goal of our project, we preferred to focus on the core of the crater detection problem instead of adding our CDA into Cytomine.

6 | Results, Validation and Discussion

6.1 Framework for Objective CDA Performance Evaluation

Salamunićcara and Lončarićb (2008) designed a framework to evaluate the efficiency of CDAs.

From 1998 to 2007, 73 CDAs have been developed. When a new method to detect craters is built, there is a need to compare the novel CDA with the best previously developed CDAs throughout coherent comparison protocols and criteria. It implies either to re-implement the *best* methods or compare on similar datasets with the same metrics. In the first way, the re-implementation can be a laborious task and some articles do not provide enough information (especially for hyper-parameters). Though the second way looks easier, there is no common lunar crater dataset used for CDAs as it is the case on Mars.

In our work, we built a substantial dataset of lunar craters on various scenes on NAC CDR LROC products and on ortho-rectified NAC images. We vary the illumination conditions, the latitude, the lunar hour, the overall image quality and the geological lunar context. Any researcher can use the dataset to make its own experiments.

Moreover, we use in our work common pattern recognition metrics/protocols put forward by CDA developers. The objective is to use a coherent way to evaluate the semantic precision of the detected craters. On the one hand, we produced masks of craters. The evaluation can be based on these maps in a pixel-based approach. On the other hand we extracted from these masks the crater objects in CSV files, containing radii and positions. using these files, we can evaluate our CDA according to an object-based approach.

6.1.1 Metrics

Error/Confusion Matrix

The error matrix (or confusion matrix) is one way to assess the performance of a classification process. In this matrix, the predicted results are compared to the true labels. Dealing with binary classification, the error matrix only contains four cells.

		Predicted	
		0	1
Ground Truth	0	TN	FP
	1	FN	TP

- TN (true negative) : predicted negatives that are actually negatives.
- FP (false positive) : predicted positives that are actually negatives.
- FN (false negative) : predicted negatives that are actually positives.
- TP (true positive) : predicted positives that are actually positives.

These values are either absolute (count) or relative (proportion of total).

Note that the confusion matrix can be built either pixel-wise or object-wise.

Metrics

From the before-mentioned values, we can compute useful metrics :

- Recall (or sensitivity) $= \frac{TP}{TP+FN}$: corresponds to the proportion of positives detected.
- Specificity $= \frac{TN}{TN+FP}$: corresponds to the proportion of negatives detected.

The objective of any CDA is to maximize these two metrics.

Other metrics also exists :

- Precision $= \frac{TP}{TP+FP}$: reliability of the predicted positives.
- Accuracy $= \frac{TP+TN}{TP+FN+FP+TN}$: proportion of well-classified elements amongst all elements.
- f1-score $= 2 * \frac{Precision * Recall}{Precision + Recall}$: harmonic mean of the precision and recall. Considered as an ideal compromise between both.
- Intersection over Union (IoU) $= \frac{Positive_{pred} \cap Positive_{true}}{Positive_{pred} \cup Positive_{true}}$: represent the concordance between the targeted elements we predict and their true positions (fig 6.1).

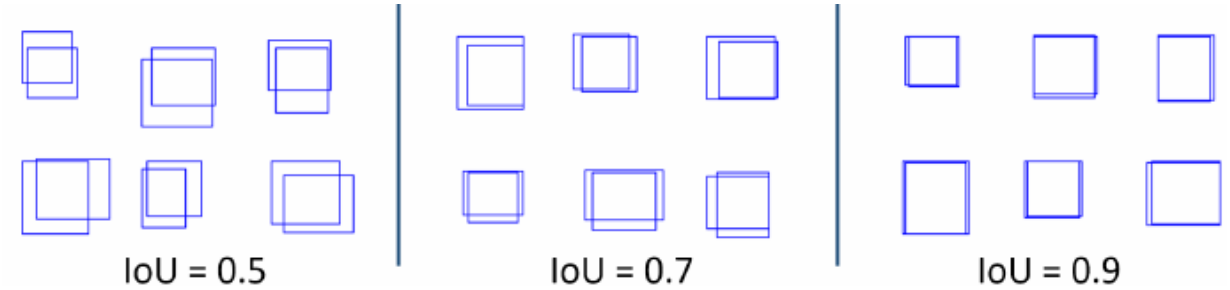


Figure 6.1: Graphical representation of intersection over union metric on detected objects.

6.1.2 ROC Curve

With the sensitivity and specificity, it is possible to represent a model in a bidimensional space called receiver operating characteristic (ROC) space (see fig 6.2). In this space, an ideal classifier is situated in the upper-left corner. The upper-right corner consists in a classifier that always predicts *positive*. Similarly, the lower-left corner corresponds to a classifier that always predicts *negative*. The center of the space is a random classifier.

A classifier often computes the probability of an object to belong to a class. The choice to associate the object to the label is finally made by setting a threshold above which the class is predicted.

By varying this threshold, we change the sensitivity and specificity, making the model describe a curve on the ROC space, called the ROC curve.

It is possible to plot different ROC curves to compare different models. If one is always upper than the other, the model is better. However, if the two curves cross, we choose the model according to the metric we want to maximize.

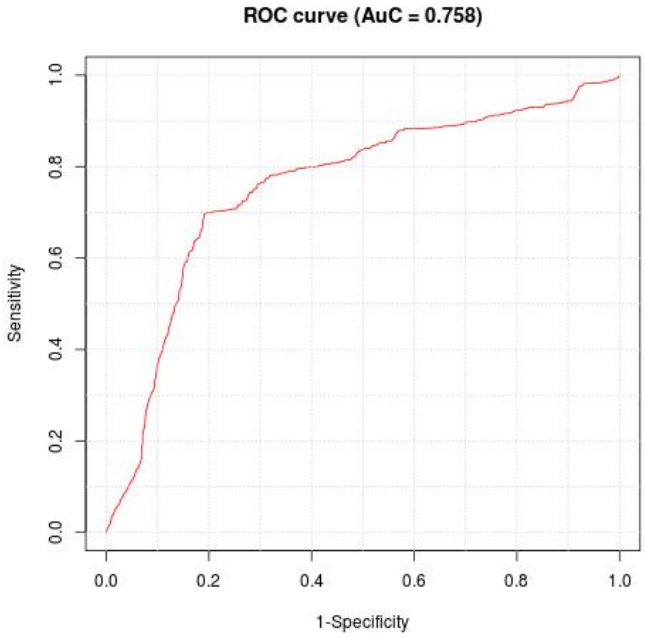


Figure 6.2: Example of a ROC curve produced in our work.

In the case of safety landing for example, the priority is that all craters are detected, i.e. maximizing sensitivity. In the case of crater landmarks-based navigation, we need a high reliability when a crater is detected meaning maximizing the false negative rate (1-specificity).

Finally, it is possible to summarize the graph as the area under the curve (AuC).

The Python code created on this purpose saves the results by building a dedicated R-script that plots and saves the ROC curve.

6.1.3 Object-Based Evaluation

The pixel-based evaluation is an easy task to implement. To each pixels in the image, we look at true label and we compute the confusion matrix (and its related metrics). Since the evaluation is simple, the interpretation is sometimes confusing.

The object-based evaluation is more complex to implement. It is impossible to detect craters that are perfectly equivalent to the true craters. There is always a shift in center positions or a difference in radius and shape. We have to define a certain tolerance. To do so, Salamunićcara and Lončarićb encourage researchers to compute a *measure of differences*

in position and size between two craters. This measure is normalized by the crater size, meaning that this score is not biased by the size of the crater (with the assumption that all crater characteristics evolves proportionally to the crater size).

The measure of crater difference is defined by

$$f = max\left(\frac{r_1}{r_2} - 1, \frac{d}{r_2}\right) \tag{6.1}$$

Where *1* and *2* represent the predicted crater and the true crater. r_1 is the bigger radius between the two and d is the euclidean distance between the craters.

The first term of the equation is based on the difference in radius. The second is based on the difference in center position. The two terms are normalized by the radius of the bigger crater. The score does not simply averages the two terms but only takes the worst. A crater is considered as *well-predicted* if the score is below a specific threshold. Fig 6.3 shows different examples of computed measures of crater difference.

To produce that metric, we need to spatially pair the craters. For each true crater, we compute all possible measures of crater difference with all predicted craters. The f is chosen taking the minimum value encountered, i.e. the crater that is the most likely associated with the true crater.

Though that metric is visually interesting, it has two important disadvantages. First, it only concerns the positives, meaning that false negatives are not taken into account. Secondly, the metric cannot be used alone. If not, the CDA could simply create a huge amount of craters to maximize the score. It must at least be paired with false negative rate for example.

Note that this metric was already used by Wetzler et al. (2005).

6.2 Results and Discussion

To evaluate metrics that represents the CDA, we can use the test sample. It contains patches that need to be predicted. The evaluation is thus processed by comparing the prediction with the ground truth.

The train and test set are randomly divided at each execution of the code. To compute the metrics, we can execute several times the code and aggregate the results. With a random

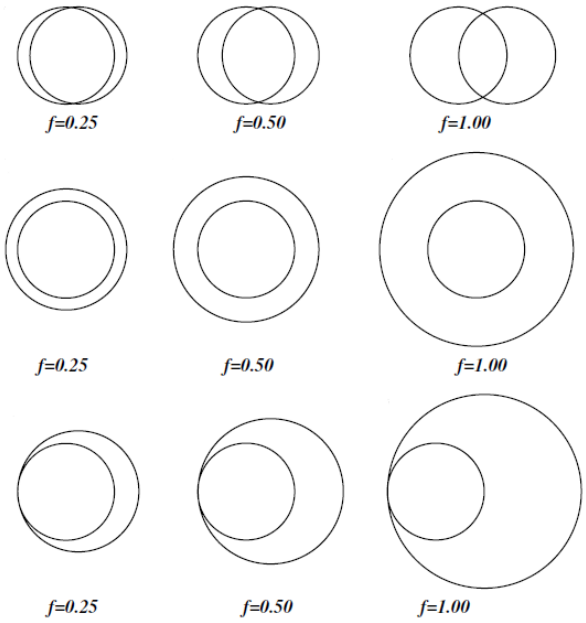


Figure 6.3: Examples of different comparisons between ground truth and predicted craters and its associated measure of difference (Salamunićcara & Lončarićb, 2008).

separation of the train/test set, we do not strictly need a validation set[1]. However, to be consistent with traditional machine learning approaches, we defined a validation set which represents 10% of the whole dataset (fig 5.7 visually summaries the different sets).

Note that, similarly to the train step, memory limitations do not allow us to predict the whole test set in once. Thus we have to evaluate the model using mini-batches.

6.2.1 Evaluation Procedure

In deep learning, each training operation can be quite long. Producing a sufficient amount of data needs an evaluation procedure to limit the unreasonable time needed to produce an exhaustive analysis on all considered models. The idea developed here is to compute at the beginning a limited amount of trials to eliminate models whose metrics are well below the others (see annexes for an exhaustive description of analyzed models). Then we repeat the process with the remaining models by producing a bigger amount of runs. After that, we

[1]Traditionally, evaluation methods use a train set, a test set and a validation set. The reason why using 3 sets is simple. If your sets are fixed and you optimize the hyper-parameters of the model using a metric computed on the test set, you overfit the test set. Even if the training of the model only uses the train set, the choice of hyper-parameters are made using the test set, what makes it not independent. The validation set is thus a totally independent set which gives an approximation of the overall accuracy of the model.

can remove the worst remaining models. We iterate up to find the different useful models considering some applications. Some may have a good ability to minimize false positive rate. In contrast, some might be better in recognizing craters with a high reliability (low false negative rate). Finally, the last ones can split the difference.

In our work, the evaluation procedure of the training stage is separated in 3 steps. First, we rapidly eliminate models that do not suit our requirements. The eliminated models are called *third order models*[2]. Then, from the remaining ones, we produce more results. From these we separate the *second order models* (that are discarded) from the *first order models* (retained). Note that the first order models are chosen not only in decreasing order according to the scores. Actually, some good models are quite redundant. There are thus put into the *second order models*. Also, even among models that are not at the top of the rankings, some have characteristics that may be useful in certain circumstances (a better recall for example).

Then, to measure the ability of the model to generalize the problem to all considered latitudes (from -60° to +60°), we divide our dataset into two subsets. The first one is concerned with the latitudes closest to the lunar equator (from -30° to +30°). The second one contains the annotations of the images situated beyond these latitudes (below -30° and above +30°). As the images have been chosen according to a systematic random sampling, the subsets are balanced in terms of amount of annotations and in terms of priors.

In the same way, we can separate the different data sources (namely the annotations from the NAC CDR images and those from ortho-rectified images). In section 4.2, we were wandering if the ortho-rectification brings something useful when trying to detect craters. That must be verified in this section.

6.2.2 Results

The results displayed below are aggregated results from the different models on the 3 different datasets (namely train set, test set and validation set). The metrics are computed using pixel-wise comparison between ground truth and prediction with a threshold probability of belonging to a crater set to 50%.

Note that we discarded the object-based evaluation method in our evaluation. Though we were confident in our approach, we notice that craters close to each other tend to merge, leading to very poor results when trying to extract crater objects from the connected component search algorithm, even if visual interpretation leads us to conclude that the

[2]Note that the models 1_1_0_0 and 1_1_0_1 are redundant, just as 1_1_0_0 and 1_1_0_0. The notion of symmetricality (a symmetric model is a model that has the same number of convolutional layers than deconvolutional layers) is pointless for such simple models. In our framework, we only consider 1_1_0_0 and 1_1_1_0.

detection performs well.

Exhaustive quantitative results, including ROC curves, are available in the annexes. Annexes also comprises visual results on crater prediction on the different sets used but also detections of craters on parts of the scenes (with a distinction between NAC CDR images and Ortho images).

The first series of tests allows us eliminate all models with only one stage (namely *1_1_0_0, 1_1_1_0, 1_2_0_0, 1_2_0_1, 1_2_1_0* and *1_2_1_1*). These models are not able to encode the deep enough characteristics representative of the crater object. We also eliminate some of the models with two stages with only one convolutional layer in each stage, suffering from the same problem (*2_1_0_1* and *2_1_1_1*). Finally, we remove the two models with 3 stages that have very asymmetrical deconvolution network. The rapid growth in their dimension (from one eighth of the patch size to the original patch size in one deconvolutional layer) does not seems relevant (*3_1_0_0* and *3_1_1_0*).

In the second series of tests we eliminate the remaining two-stages models with one convolutional layer in each (*2_1_0_0* and *2_1_1_0*). We also remove *2_2_0_0* because of its redundancy with model *2_2_0_1*. Moreover, we remove the three-stages models with one convolutional layer in each. We can here observe the benefits of multiple non-linearities within the stages. Finally, we withdraw the models *3_2_0_0* and *3_2_0_1* because of their redundancies with models *3_2_1_0* and *3_2_1_1*. Since the latter gives better results, we only consider them.

We can also visualize the correlation between the different sets (table 6.4). We notice that the validation set is highly dependant of the train set. That observation is reassuring. It shows that our model does not overfit any set and that the application of the model to unseen data is not a risky venture.

In addition, we can analyze the dependences between the different metrics (table 6.5). For example, the F1-score and IoU are very redundant metrics. Moreover, we notice that the precision is a very important metric since no other score explains it correctly.

All these models can be represented in a diagram with prediction and recall as axis (fig 6.4).

The retained models are finally *2_2_0_1*, *2_2_1_0*, *2_2_1_1*, *3_2_1_0* and *3_2_1_1*. Their representation can be seen below. The yellow layers represent the convolutional layers (including activation function), the blue ones the pooling layers and the green ones the deconvolutional layers. The thickness of each layer is representative of the number of features maps produced by the layer. More details are available in the annexes.

Considered Model		1_1_0_0	1_1_1_0	1_2_0_0	1_2_0_1	1_2_1_0
Precision	Train	0.724	0.500	0.822	0.802	0.500
	Test	0.726	0.769	0.819	0.801	0.574
	Valid	0.724	0.500	0.823	0.792	0.500
Recall	Train	0.423	0.351	0.149	0.519	0.276
	Test	0.421	0.005	0.151	0.520	0.000
	Valid	0.418	0.355	0.143	0.493	0.267
F1-Score	Train	0.534	0.412	0.253	0.631	0.355
	Test	0.533	0.011	0.255	0.630	0.001
	Valid	0.530	0.415	0.244	0.608	0.348
Accuracy	Train	0.823	0.566	0.789	0.854	0.630
	Test	0.823	0.761	0.787	0.854	0.760
	Valid	0.821	0.561	0.788	0.847	0.637
IoU	Train	0.364	0.260	0.145	0.460	0.216
	Test	0.363	0.005	0.146	0.460	0.000
	Valid	0.361	0.262	0.139	0.436	0.210
Considered Model		1_2_1_1	2_1_0_1	2_1_1_1	3_1_0_0	3_1_1_0
Precision	Train	0.802	0.723	0.803	0.826	0.787
	Test	0.804	0.724	0.810	0.832	0.812
	Valid	0.796	0.714	0.779	0.845	0.824
Recall	Train	0.448	0.241	0.581	0.425	0.109
	Test	0.441	0.239	0.584	0.426	0.104
	Valid	0.427	0.252	0.540	0.407	0.100
F1-Score	Train	0.575	0.362	0.674	0.562	0.192
	Test	0.570	0.360	0.679	0.564	0.185
	Valid	0.556	0.373	0.638	0.550	0.179
Accuracy	Train	0.841	0.796	0.864	0.839	0.777
	Test	0.840	0.795	0.867	0.842	0.782
	Valid	0.836	0.796	0.847	0.840	0.778
IoU	Train	0.404	0.221	0.509	0.390	0.106
	Test	0.398	0.219	0.514	0.392	0.102
	Valid	0.385	0.229	0.468	0.379	0.098

Table 6.1: Third order models - immediately eliminated.

Model		2_1_0_0	2_1_1_0	2_2_0_0	3_1_0_1	3_1_1_1	3_2_0_0	3_2_0_1
Precision	Train	0.788	0.773	0.860	0.823	0.746	0.832	0.808
	Test	0.786	0.771	0.863	0.842	0.763	0.849	0.804
	Valid	0.793	0.776	0.854	0.834	0.737	0.818	0.801
Recall	Train	0.627	0.543	0.543	0.741	0.643	0.763	0.760
	Test	0.625	0.543	0.760	0.750	0.648	0.775	0.754
	Valid	0.600	0.520	0.733	0.720	0.603	0.730	0.725
F1-Score	Train	0.698	0.628	0.807	0.779	0.689	0.796	0.783
	Test	0.696	0.628	0.808	0.792	0.698	0.810	0.778
	Valid	0.683	0.615	0.789	0.772	0.662	0.772	0.760
Accuracy	Train	0.870	0.853	0.912	0.896	0.858	0.903	0.899
	Test	0.870	0.853	0.913	0.906	0.867	0.912	0.896
	Valid	0.866	0.850	0.904	0.896	0.848	0.889	0.891
IoU	Train	0.536	0.467	0.676	0.641	0.530	0.662	0.644
	Test	0.534	0.466	0.678	0.657	0.532	0.681	0.637
	Valid	0.519	0.452	0.652	0.630	0.507	0.628	0.614

Table 6.2: Second order models - eliminated after deeper computations.

Considered Model		2_2_0_1	2_2_1_0	2_2_1_1	3_2_1_0	3_2_1_1
Precision	Train	0.699	0.844	0.728	0.834	0.849
	Test	0.698	0.856	0.725	0.843	0.847
	Valid	0.704	0.826	0.733	0.830	0.836
Recall	Train	0.964	0.776	0.960	0.796	0.787
	Test	0.963	0.782	0.958	0.803	0.781
	Valid	0.954	0.755	0.949	0.770	0.754
F1-Score	Train	0.809	0.808	0.828	0.814	0.817
	Test	0.809	0.817	0.825	0.822	0.813
	Valid	0.815	0.772	0.827	0.798	0.792
Accuracy	Train	0.891	0.909	0.904	0.911	0.914
	Test	0.890	0.916	0.902	0.917	0.914
	Valid	0.892	0.895	0.904	0.904	0.903
IoU	Train	0.681	0.678	0.707	0.687	0.691
	Test	0.679	0.691	0.703	0.699	0.685
	Valid	0.681	0.644	0.705	0.666	0.657

Table 6.3: First order models - retained and validated.

R	Train	Test	Valid
Train		0.947	0.948
Test			0.913
Valid			

Table 6.4: Correlation between the different dataset (train set, test set and validation set).

R	Precision	Recall	F1-Score	Accuracy	IoU
Precision		0.313	0.397	0.788	0.411
Recall			0.903	0.695	0.920
F1-Score				0.747	0.990
Accuracy					0.776
IoU					

Table 6.5: Correlation between the different metrics.

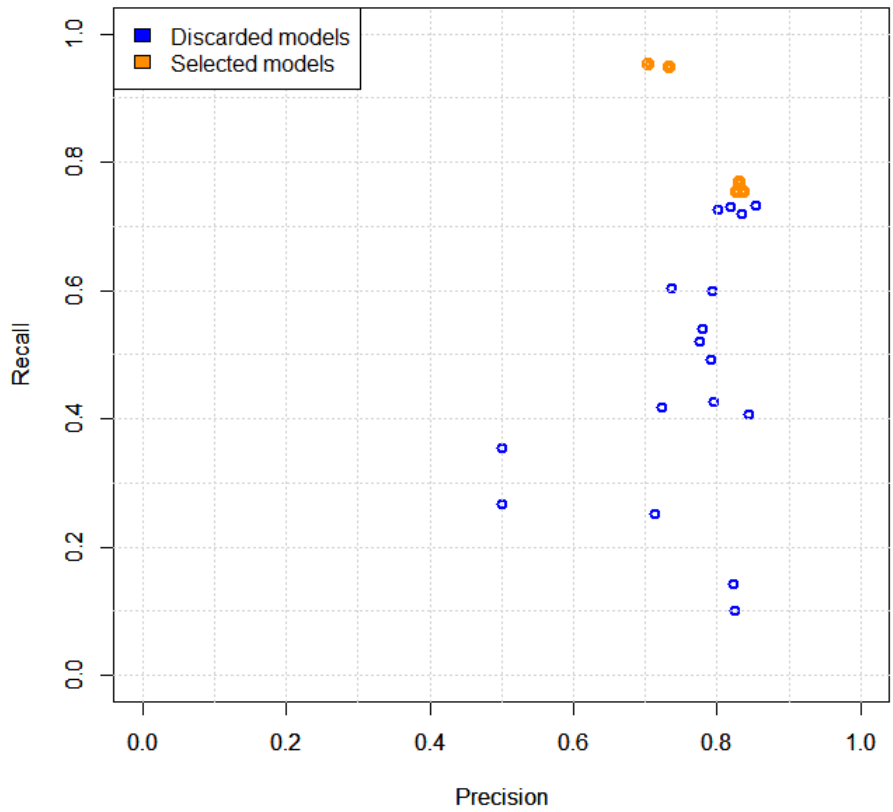


Figure 6.4: Representation of the models in the precision-recall space.

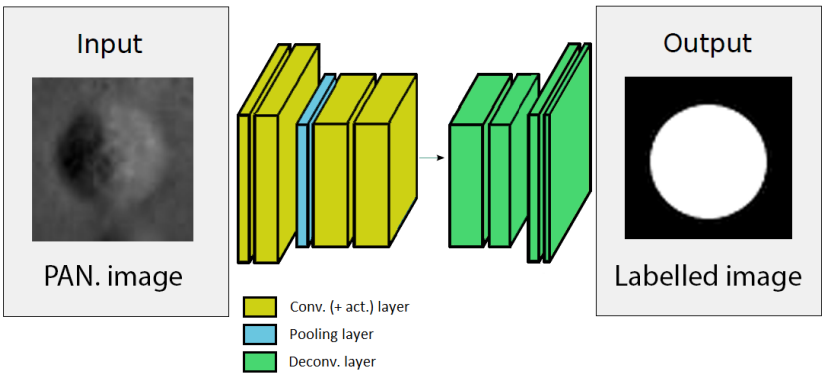


Figure 6.5: CraterNet 2_2_0_1.

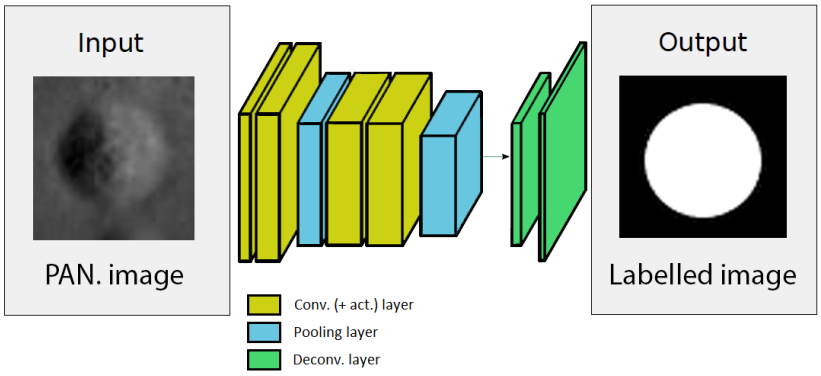


Figure 6.6: CraterNet 2_2_1_0.

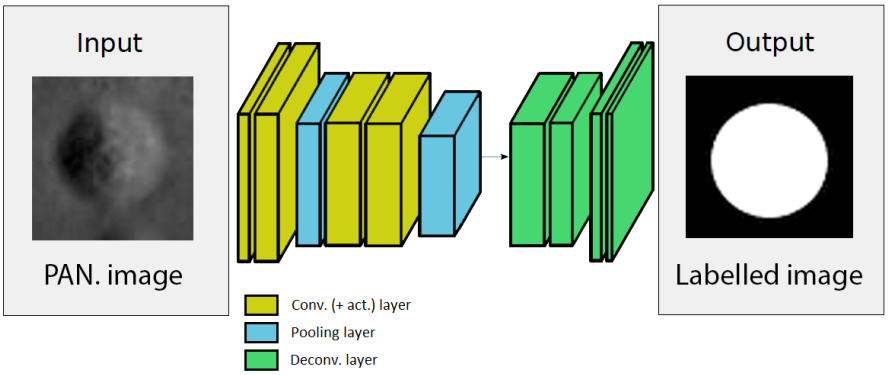


Figure 6.7: CraterNet 2_2_1_1.

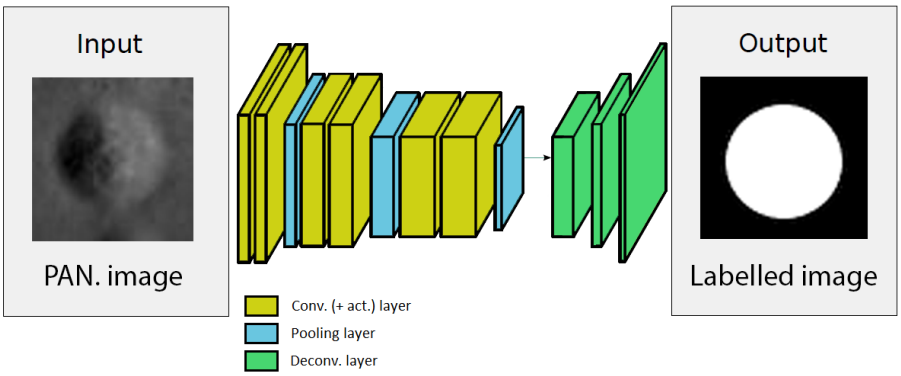


Figure 6.8: CraterNet 3_2_1_0.

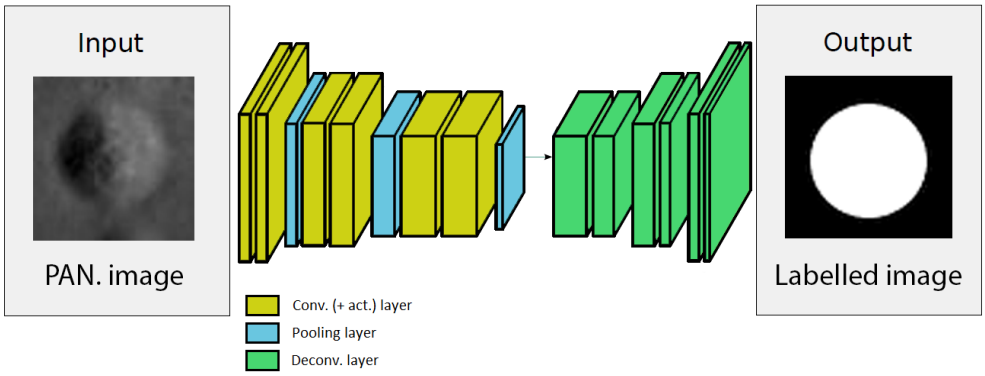


Figure 6.9: CraterNet 3_2_1_1.

After computing the results on the whole dataset, we can focus on the stratified dataset according to the latitude (table 6.6). Here, we split the dataset in central latitudes (-30° to +30°) and in external latitudes (-60° to -30°and 30° to 60°) where we train and test each model on its same range of latitudes. We only considered the models *2_2_1_1* and *3_2_1_0*.

We clearly notice the improvement when stratifying the dataset. With the model *2_2_1_1*, the intersection over union increases from about 0.71 to 0.80 (low latitudes) and 0.75 (high latitudes). The F1-Score also increases from 0.83 to 0.89 (low latitudes) and 0.86 (high latitudes). The same phenomenon occurs with model *3_2_1_0*, with slightly lower results.

We can also observe that the two subsets do not produce similar results. In both models, the annotations situations in latitudes ranging from -30° to +30° perform better than latitudes beyond.

Considered Model		2_2_1_1 Low lat.	2_2_1_1 High lat.	3_2_1_0 Low lat.	3_2_1_0 High lat.
Precision	Train	0.821	0.779	0.857	0.856
	Test	0.820	0.774	0.899	0.849
Recall	Train	0.967	0.955	0.817	0.816
	Test	0.963	0.949	0.848	0.805
F1-Score	Train	0.888	0.857	0.836	0.836
	Test	0.885	0.852	0.872	0.826
Accuracy	Train	0.941	0.924	0.916	0.922
	Test	0.941	0.921	0.941	0.919
IoU	Train	0.799	0.751	0.720	0.718
	Test	0.795	0.744	0.775	0.705

Table 6.6: Results when separating the datasets according to the latitude.

Considered Model	2_2_1_1 Low lat.	2_2_1_1 High lat.	3_2_1_0 Low at.	3_2_1_0 High lat.
precision	0.764	0.716	0.845	0.838
recall	0.957	0.951	0.744	0.745
F1	0.849	0.817	0.791	0.789
Accuracy	0.918	0.897	0.902	0.904
IoU	0.738	0.690	0.655	0.651

Table 6.7: Results on stratified validation sets according to the latitude.

Also, we can train our model with the entire dataset and validating on different ranges latitudes (table 6.7). With the model *2_2_1_1*, we notice an improvement at low latitudes and a degradation on high latitudes. This is coherent with previous results. Also, we observe that model *3_2_1_0* is insensitive to latitudes variations, though the model performs less well on most metrics.

After that, we can evaluate the model trained with the entire dataset on the different data source (namely NAC CDR images and ortho-rectified images) (table 6.8). Surprisingly, the results are better on the NAC images. The reasons may be that the ortho-rectification, though it corrects some alterations due to panoramic disturbances, brings some linear noises through lines and columns in some images. Those linear artifacts may be decreased using Fourier denoising pre-process. However we did not test it in our work.

Considered Model	2_2_1_1 NAC	2_2_1_1 Ortho	3_2_1_0 NAC	3_2_1_0 Ortho
precision	0.751	0.713	0.827	0.772
recall	0.958	0.949	0.773	0.741
F1	0.841	0.814	0.799	0.756
Accuracy	0.913	0.896	0.903	0.876
IoU	0.727	0.687	0.665	0.608

Table 6.8: Results on stratified validation sets according to the type of data.

Then, we can try to train our two datasets separately and measure the performances (table 6.9). We notice a slight improvement when separating the dataset, with again better results on the NAC CDR images.

However when training the models on one specific data source and trying to predict the other one (i.e. performing somehow *transfer learning*), the results are drastically decreased (table 6.10).

Considered Model	2_2_1_1 NAC	2_2_1_1 Ortho	3_2_1_0 NAC	3_2_1_0 Ortho
precision	0.771	0.769	0.843	0.825
recall	0.962	0.949	0.792	0.792
F1	0.856	0.849	0.816	0.808
Accuracy	0.922	0.918	0.910	0.906
IoU	0.749	0.738	0.690	0.679

Table 6.9: Results when separating the datasets according to the data source.

Considered Model	2_2_1_1 NAC	2_2_1_1 Ortho	3_2_1_0 NAC	3_2_1_0 Ortho
precision	0.710	0.686	0.799	0.719
recall	0.910	0.843	0.685	0.647
F1	0.797	0.756	0.737	0.681
Accuracy	0.889	0.868	0.880	0.837
IoU	0.663	0.607	0.584	0.517

Table 6.10: Results when trying to predict the other data source.

Analyzing those results, we notice that in exchange of a slight loss of overall accuracy, the model can combine different data sources in order to produce a model that is satisfying in different circumstances.

6.2.3 Discussion

The results shows that depending on the application, some models are more suitable than others. We notice that models *2_2_0_1* and *2_2_1_1* distinguish in their ability not to miss craters (more than 0.95 in recall). If the goal is to ensure that a probe does not land in a crater, these models must be considered. In return, the reliability of these craters are sometimes low, with about 30% of pixels considered as craters that are actually not. We notice that model *2_2_1_1* is slightly better in general, meaning that the central pooling layer brings some benefits to crater recognition.

If the goal is to maximize the reliability of each crater found, the 3 other models are more likely to be picked. They all produce similar results, with *3_2_1_0* that performs a bit better. It is moreover the quicker model to train and use. That model is thus to be privileged.

Looking at visual results, we notice that the asymmetrical models are more flexible in terms of decision making (fig 6.10). The probability distribution of belonging to a crater is more relaxed than symmetrical models, making possible to slightly move the threshold above which the pixel is classified as crater to produce significantly different results. Depending on the application, this option is welcome.

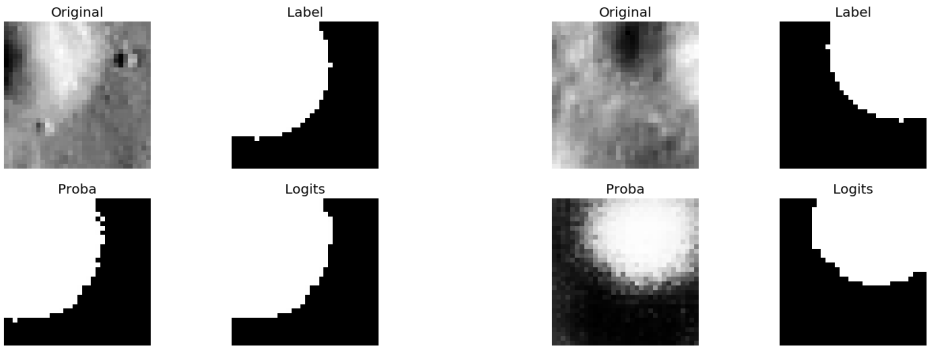


Figure 6.10: Difference between a symmetrical model (left) and a asymmetrical model (right) in terms of computed probabilities (bottom-left patches).

In visual results, we are disappointed that some lunar rocks or other topographic elements are classified as craters (fig 6.11). The a priori knowledge of solar illumination conditions would have enable the model to understand that the highlight/shadow areas should have been inverted. Since the model was designed to work autonomously, we did not let it know that information. We thought that showing a set of lunar rocks classified as negatives would have enable the model to learn the slight difference between the two (in terms of

sharpness and shape for example).

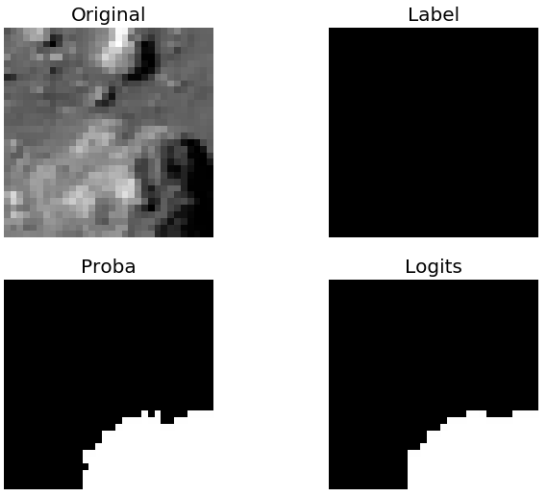


Figure 6.11: Misclassified crater. Ground truth = label, prediction = logits.

Though the results show that the method is viable, we are a bit disappointed not to see spectacular enhancement of the metrics compared to state of the art methods. The main problems are encountered with the smaller craters when applying the model on a scene. The noisy background of the lunar ground is often interpreted as small craters (especially on ortho-rectified images). Some parts of the topographic lunar surface are rugged and are also misinterpreted as craters.

Nevertheless, when working on stratified datasets, we observe a net improvement of our models. Indeed, the stratification of the dataset also decreases the complexity of the problem. That observation indicates that our fully convolutional neural networks have difficulties to generalize the problem. In addition, we notice that the model is more able to detect craters that are closer to the lunar equator. Looking back at the skyplot (fig 4.2), that observation is understandable since high latitudes leads to more varied illumination conditions.

When separating the two data sources, we notice that the ortho-rectification of the images has a negative impact in all metrics. Also, we observe that transfer learning is not viable with our models. However, we are glad to see that combining two different data sources is possible with minor decreases in overall accuracy.

6.2.4 Trials on Mars Dataset

The CDA developed in our project is focused on the Moon but we can try the application directly on the Mars dataset introduced in section 4.2.

In order to rapidly produce results, we use the freely available dataset[3] from Urbach and Stepinski (2009) to train our model. The kit is composed of 6 tiles of a single scene with a set of candidates in CSV files. This set of candidates comes along with its ground truth : for each candidate an expert put a label.

However, the proper use of these data is not designed to semantically segment the scene but rather to non-spatially classify the set of candidates. Traditionally, articles using this dataset developed a machine learning model able to classify the candidates. Nevertheless, those methods are not able to determine craters that are not part of the candidate dataset.

This point is important because our CDA is indeed designed to classify an entire scene. In our case we need annotations of craters but also crater-free zones. In the Mars dataset, the only available data are the candidates i.e. a set of craters or zones that look like craters (since they have been classified as *candidates*). In this dataset we are thus missing annotations of various simple crater-free zones, making our CDA unable to classify scenes that should be easily classified as crater-free areas.

Moreover, though positive annotations (craters) are quite relevant to our CDA, the negative annotations are highly redundant (fig 6.12). We sometimes meet more than 10 annotations of the same object. With some tiles having less than 300 annotations, the amount of examples to train our model is quite small. In addition, some of the negative martian annotations are misleading according to our CDA and lead it to unlearn the concepts describing the crater object because what Urbach and Stepinski consider as negative candidates contains pixels that are actually positives (fig 6.13).

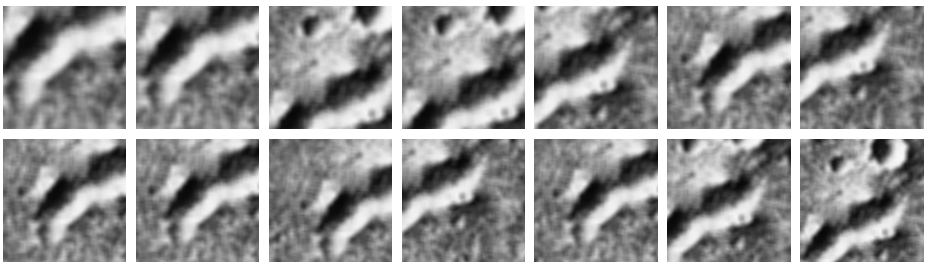


Figure 6.12: These 14 negative martian annotations represent the same object with a slight translation (Urbach & Stepinski, 2009).

[3]https://github.com/ieee8023/CraterDataset

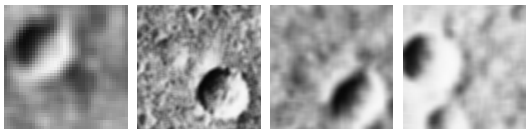


Figure 6.13: These negative martian annotations contain craters with diameter size that should have been classified as positive, according to our methodology (Urbach & Stepinski, 2009).

At finer resolution, the model is even less able to detect crater-free areas because of the speckle noise of this particular Mars scene. It results in large craters that may be correctly classified and smaller craters which tend to merge, forming tentacular connected components at fine scale (fig 6.14).

Figure 6.14: Attempts to retrieve craters on a Mars scene. Left : craters of about 64 pixels. Right : craters of about 16 pixels.

6.2.5 Comparison with Other Models

The use of the framework developed by Salamunićcara and Lončarićb (2008) helps the comparison with other CDAs developers. Nevertheless, the most renowned authors in CDAs were working on Mars, which makes our comparison difficult. Moreover, the geophysical process and the geological history of the planet may influence the results. Also, CDA developers who worked on the Mars dataset developed by Urbach and Stepinski (2009) only evaluate their results based on the ground truth related to the candidates and not on the entire scene. We can argue that the CDA can miss craters/find false craters because the candidates pre-selection may not be exhaustive. Nevertheless, our evaluation method suffers from the same problem.

For any researcher who would want to reuse our dataset, we recommend to take benefits from the framework for CDA performance evaluation to make comparisons easier. We are aware of the issues met by authors who wants to situate their results among the literature. The confusing variety of ways to evaluate the efficiency of the methods is one major problem

but the use of a different dataset is another one when trying to make comparisons.

The CDA we developed has several pro and cons compared to the CDAs developed by previous researches.

The first advantage is that our models are independent from any previous work. In the case of CDA developers that work on the Mars dataset, their results are dependent on the craters candidates estimator developed by Urbach and Stepinski.

Also, our models are quite efficient in terms of computation time. Once trained, each model is able to map an entire region of 5000x50000 pixels in less that one minute. By comparison of the candidates estimator takes a couple of hours to produce its results[4].

Our CDA, as many deep learning models, is quite flexible. Choosing a different set of hyper-parameters may lead to completely different results. It thus enables researchers to use one specific model for one particular application.

Finally, once the CDA is employed, the model may look like a black-box of parameters difficult to interpret for humans. We can look at the different trained filters (see annexes) and try to extract general conclusions about the way our models define the crater object. In practical terms, the number of filters and the spatial arrangement of the filter weights make the interpretation almost impossible.

[4]Note that with recent advances in terms of hardware and image processing, the time needed to compute these candidates is probably much slower, but certainly still slower than ours by several order of magnitude.

7 | Conclusion and Perspectives

7.1 Conclusion

In this project, we tried to use the recent advances in deep learning into the unsolved problem of crater detection. To do so, we had to study the formation of craters and look at scientific literature in the crater detection problem. We also needed to investigate in progress in deep learning and try to implement some classical models such as CNN or FCN.

After that, we were looking for data to train our model. Unfortunately, no dataset met our needs. Then came the question of how to create our learning observations. in this context, we created a Cytomine project and we integrated some lunar images from the LRO space probe. Cytomine enabled us to create a fairly large number of annotations necessary to our CDA.

Doing so, we contributed somehow to the research in crater detection by making an entire crater dataset available to the scientific community. This dataset consists in more than 11 000 annotations of craters and crater-free area in two types of data sources. The first one is the NAC CDR image available in the NASA's LROC archive. The second one is the ortho-rectified images produced by the Arizona State University. This dataset is available directly on Cytomine with a authorized account or by the use of a CSV file containing all the needed information to easily extract the annotations on the images.

Based on these annotations, we created patches that help our model learn the relevant filters allowing to separate the crater object from the background. As our model is only functional for a fixed crater size, we produced a pyramid of the scene, to recognize craters of different sizes.

Then, we evaluate our models using a pixel-based approach. Though we are aware that our CDA did not clearly overpass previously described CDAs, some advantages make it a viable technique that could inspire geomorphological researcher, considering some applications.

7.2 Perspectives

In this master thesis, we trained a model based on remotely sensed data. Firstly The idea was to use only the NAC CDR images, which are radiometrically calibrated images from the panchromatic sensor of the LRO probe that has the smallest field of view. However we discovered later in March the ortho-rectified images produced by the Arizona State University. One idea was to use these images and look at the advantages of the ortho-rectification in the results. However we could also use the DEM that maps the terrain elevation across the area. These topographic information could have been beneficial in our CDA. Nevertheless, due to lack of time and since this was not the primary objective of the project, we discarded this option.

In addition, one idea that maybe could have been implemented was to introduce in the annotations a rank according to the level of degradation. Doing so, we could have evaluated the model according to the freshness of the crater. However, the limits between the different classes of degradation is difficult to establish and criteria could have been subject to confusion.

Another possible stratification is the geological context of the model. If we go back to figure 4.5, we notice different types of ground. These information could have been taken into account when creating annotations.

Moreover, the image quality is generally fairly good. One question to be answered is to know what happens when a certain amount of noise is added on the images. This artifact could have been added by simulating salt-and-pepper noise for example.

In terms of implementation, we would have liked to use TensorBoard in a more efficient way. TensorBoard recently reached a high level of interaction with developers and the possibilities it offers can be useful.

In conclusion, each time a question is answered, new questions emerge, giving the researcher the opportunity to deepen his research. However, we arrive at a point where it is time to stop computing results and developing ideas. These new perspectives are all opportunities for young researchers to get involved in the exciting adventure of crater detection and in image processing in remote sensing in a more general sense.

Annexes

Database

NAC images

Exhaustive NAC-CDR images list (bold product IDs are annotated images), from the free planetary data system database :

- M1105680456LC
- **M1105680456RC**
- M1113630818LC
- **M1113630818RC**
- M1113929391LC
- **M1113929391RC**
- **M1118438810LC**
- M1118438810RC
- **M1120982294LC**
- M1120982294RC
- **M1128984946LC**
- M1128984946RC
- M1129170030LC
- **M1129170030RC**
- **M1131721504LC**
- M1131721504RC
- M1134111735LC
- **M1134111735RC**
- M1136506064LC
- **M1136506064RC**
- **M1144451737LC**
- M1144451737RC
- M1146592844LC
- **M1146592844RC**
- M1146911198LC
- **M1146911198RC**
- M1147117663LC
- **M1147117663RC**
- **M1149076545LC**
- M1149076545RC
- M1151417501LC
- **M1151417501RC**
- M1151437972LC
- **M1151437972RC**
- M1151617197LC
- **M1151617197RC**
- M1164519486LC
- **M1164519486RC**
- M121681384LC
- **M121681384RC**
- M121729272LC
- **M121729272RC**

- M129221791LC
- **M129221791RC**
- **M137299729LC**
- M137299729RC
- **M139374320LC**
- M139374320RC
- **M139376089LC**
- M139376089RC
- M139666180LC
- **M139666180RC**
- M144490338LC
- **M144490338RC**
- M154868397LC
- **M154868397RC**
- M159365173LC
- **M159365173RC**
- M159698003LC
- **M159698003RC**
- M168048543LC
- **M168048543RC**
- **M168191400LC**
- M168191400RC
- M175218641LC
- **M175218641RC**
- **M183403129LC**
- M183403129RC
- **M190536605LC**
- M190536605RC
- M190580987LC
- **M190580987RC**

Ortho-images

Table 7.1: List of the ortho-images we take some annotations

Image name	Number of annotations
NAC_DTM_APOLLO16_2	200
NAC_DTM_LICHTENBER2	200
NAC_DTM_ARISTARCHU5	200
NAC_DTM_LINNECRATER	200
NAC_DTM_GRTHSENNW	200
NAC_DTM_REINER4	200
NAC_DTM_HARDINGH	200
NAC_DTM_BRISBANEZ	200
NAC_DTM_APOLLO12	200
NAC_DTM_KUGLERRIDGE	200
NAC_DTM_MNDLSHTMLS	200
NAC_DTM_GRUITHUISE8	200
NAC_DTM_ENDYMION	200
NAC_DTM_LICHTENBER8	200
NAC_DTM_LICHTENBER7	200
NAC_DTM_LICHTENBER10	200
NAC_DTM_RUMKERDOME2	200
NAC_DTM_LUNA16	200
NAC_DTM_HORTENSIUS5	200

Exhaustive ortho-images list, from the free Arizona State University database :

- NAC_DTM_KUGLERRIDGE
- NAC_DTM_REINER4
- NAC_DTM_ARISTARCHU5
- NAC_DTM_LINNECRATER
- NAC_DTM_GRTHSENNW
- NAC_DTM_APOLLO12
- NAC_DTM_HARDINGH
- NAC_DTM_APOLLO16_2
- NAC_DTM_LICHTENBER2
- NAC_DTM_BRISBANEZ
- NAC_DTM_APOLLO16_1
- NAC_DTM_RUMKERDOME3
- NAC_DTM_LUNA24
- NAC_DTM_MARIUS4
- NAC_DTM_APOLLO11
- NAC_DTM_FECNDITATS3
- NAC_DTM_MRECRISIUM1
- NAC_DTM_APOLLO16_6

- NAC_DTM_MRECRISIUM2
- NAC_DTM_APOLLO16_5
- NAC_DTM_RUMKERDOME5
- NAC_DTM_RUMKERDOME4
- NAC_DTM_GRUITHUISE3
- NAC_DTM_HORTENSIUS3
- NAC_DTM_RUMKERDOME2
- NAC_DTM_GRUITHUIS12
- NAC_DTM_GRUITHUISE8
- NAC_DTM_ORIENTALE1
- NAC_DTM_ARISTARCHU4
- NAC_DTM_ARISTARCHU2
- NAC_DTM_APOLLO16_3
- NAC_DTM_HORTENSIUS2
- NAC_DTM_HORTENSIUS5
- NAC_DTM_LICHTENBER10
- NAC_DTM_GRUITHUISE2
- NAC_DTM_MRECRISIUM3
- NAC_DTM_LICHTENBER13
- NAC_DTM_RANGER
- NAC_DTM_LICHTENBER8
- NAC_DTM_MARIUS6
- NAC_DTM_LICHTENBER11
- NAC_DTM_COMPTONBELK
- NAC_DTM_FRSHCRATER4
- NAC_DTM_HANSTEENAL1
- NAC_DTM_A13SIVB
- NAC_DTM_LICHTENBER12
- NAC_DTM_LICHTENBER7
- NAC_DTM_LICHTENBER1
- NAC_DTM_GRUITHUISE7
- NAC_DTM_LICHTENBER3
- NAC_DTM_INACALDERA3
- NAC_DTM_REINER3
- NAC_DTM_GRUITHUISE9
- NAC_DTM_ENDYMION
- NAC_DTM_MARIUS2
- NAC_DTM_HORTENSIUS7
- NAC_DTM_GRUITHUISE4
- NAC_DTM_APOLLO16_4
- NAC_DTM_TRANQPIT1
- NAC_DTM_FRSHCRATER10
- NAC_DTM_GRUITHUISE6
- NAC_DTM_LUNA16
- NAC_DTM_HORTENSIUS6
- NAC_DTM_CRISIUMVOLC
- NAC_DTM_REINER2
- NAC_DTM_LICHTENBER9
- NAC_DTM_GRUITHUIS11
- NAC_DTM_LICHTENBER6
- NAC_DTM_ARISTARCHU3
- NAC_DTM_MNDLSHTMLS
- NAC_DTM_GRUITHUIS10

- NAC_DTM_APOLLO14
- NAC_DTM_LASELMASIF1
- NAC_DTM_SURVEYOR7
- NAC_DTM_LICHTENBER5
- NAC_DTM_RANGER9
- NAC_DTM_MARIUS3
- NAC_DTM_RUMKERDOME1

JSON examples

Example of a JSON file associated to a specific annotation :


```
{
"class": "be.cytomine.ontology.UserAnnotation",
"id": 21583989,
"created": "1490299000045",
"updated": null,
"deleted": null,
"location": "POLYGON ((4271 4909, 4277.776996252509 4915.1167393786245, 4281.480188354018
4923.461111559433, 4281.469260747733 4932.590298786123, 4277.746102916053 4940.925781651136,
4270.954482671905 4947.0262794416285, 4262.268732855433 4949.836960668341, 4253.1906985174655
4948.871833012938, 4245.290054266391 4944.297775814947, 4239.932893268318 4936.905685117907,
4238.045517328095 4927.9737205583715, 4239.954270892059 4919.046299953035, 4245.329113128517
4911.66705542763, 4253.240684985877 4907.11192539893, 4262.3210038156285 4906.168533230957,
4271 4909))",
"image": 21556903,
"geometryCompression": 0.0,
"project": 19653296,
"container": 19653296,
"user": 19652953,
"nbComments": 0,
"area": 1.470352965E9,
"perimeterUnit": "mm",
"areaUnit": "micron",
"perimeter": 137.0,
"centroid": { "x": 4260.000000000002, "y": 4928.0 },
"term": [19653341],
"similarity": null,
"rate": null,
"idTerm": null,
"idExpectedTerm": null,
"cropURL": "http://demo.cytomine.be/api/userannotation/21583989/crop.jpg",
```

```
"smallCropURL": "http://demo.cytomine.be/api/userannotation/21583989/crop.png?maxSize=256",
"url": "http://demo.cytomine.be/api/userannotation/21583989/crop.jpg",
"imageURL": "http://demo.cytomine.be/#tabs-image-19653296-21556903-21583989",
"reviewed": false
}
```


Example of a JSON file associated to a specific NAC image :


```
{
"class": "be.cytomine.image.ImageInstance",
"id": 21556903,
"created": "1490290632354",
"updated": null,
"deleted": null,
"baseImage": 21556891,
"project": 19653296,
"user": 19652953,
"filename": "/1490290559594/NAC_DTM_FECNDITATS3_M150178619_50CM.TIF",
"extension": "tif",
"originalFilename": "NAC_DTM_FECNDITATS3_M150178619_50CM.TIF",
"instanceFilename": "NAC_DTM_FECNDITATS3_M150178619_50CM.TIF",
"sample": 21556890,
"path": "/1490290559594/da15a872-9cca-4226-913a-04841d259ff5.tif",
"mime": "openslide/ventana",
"width": 9318,
"height": 57465,
"resolution": 1.0,
"magnification": null,
"depth": 8,
"preview": "http://demo.cytomine.be/api/abstractimage/21556891/thumb.png?maxSize=1024",
"thumb": "http://demo.cytomine.be/api/abstractimage/21556891/thumb.png?maxSize=512",
"macroURL": "http://demo.cytomine.be/api/abstractimage/21556891/associated/macro.png",
"fullPath": "/data/images/19652953//1490290559594/da15a872-9cca-4226-913a-04841d259ff5.tif",
"numberOfAnnotations": 201,
"numberOfJobAnnotations": 0,
"numberOfReviewedAnnotations": 0,
"reviewStart": null,
"reviewStop": null,
"reviewUser": null,
"reviewed": false,
"inReview": false
}
```

Sample of the Database in CSV Format

Here is a sample of the crater annotations table (the full table has 11 239 annotations at current state).

```
id;x;y;radius;label;id_image
23695500;5270.0;42647.5;15.8890897914;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583989;4260.0;4928.0;21.633952132;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583975;3819.0;4843.0;14.7809927636;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583952;4029.0;4679.0;19.1329977494;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583934;3760.0;4271.0;11.0171015314;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583920;3851.0;4623.0;33.9784272264;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583904;4318.0;4401.0;14.7809927636;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583883;4968.0;4381.0;18.5139760683;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583827;5298.0;4626.0;23.9960463426;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583724;5156.0;4783.0;34.2771756884;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583686;4607.0;4773.0;39.2431569046;0;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583654;3932.0;4678.0;10.0491427376;1;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583634;4488.0;4888.0;16.2516251775;1;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583602;5278.0;4265.0;11.2352831384;1;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583584;4601.0;4279.0;10.2878729972;1;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583549;4234.0;4802.0;16.5183059299;1;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
21583529;3487.0;4615.0;14.3476335455;1;NAC_DTM_FECNDITATS3_M150178619_50CM.TIF
...  ...  ...  ...  ...  ...
...  ...  ...  ...  ...  ...
```

Visual Results

Considered Models

Here is compiled the scheme of all the considered models in our project. The terminology comes as follow :

- The first term corresponds to the number of stages in the model. Each stage is composed of one or 2 convolutional layers (+ activation function) followed by a pooling layer. We considered models with 1, 2 or 3 stages.
- The second term informs on the number of convolutional layers per stage (1 or 2).

- The third term is a boolean that indicates if whether or not a pooling operation is employed just before the deconvolution process.
- The last term is also a boolean inquiring about the symmetricality of the model. A symmetrical model is a model that has as many convolutional layers than deconvolutional layers.

Figure 7.1: Craternet 1_1_0_0.

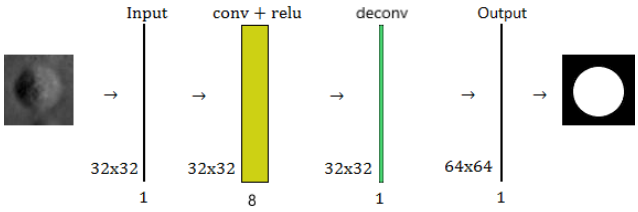


Figure 7.2: Craternet 1_1_0_1.

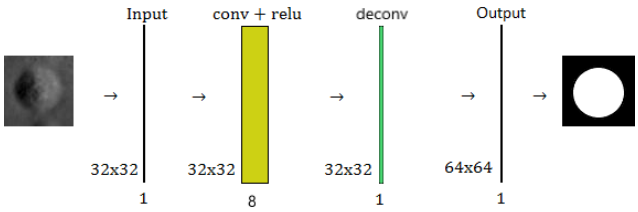


Figure 7.3: Craternet 1_1_1_0.

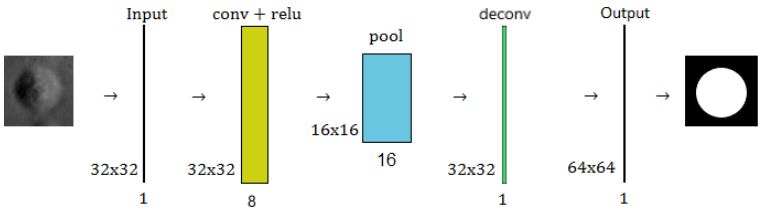


Figure 7.4: Craternet 1_1_1_1.

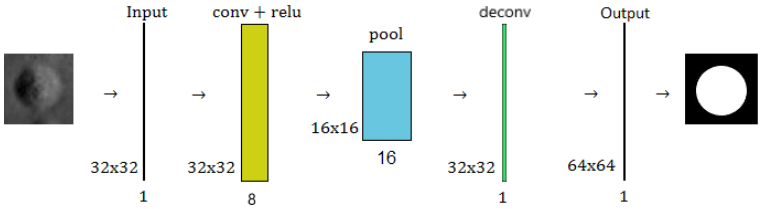


Figure 7.5: Craternet 1_2_0_0.

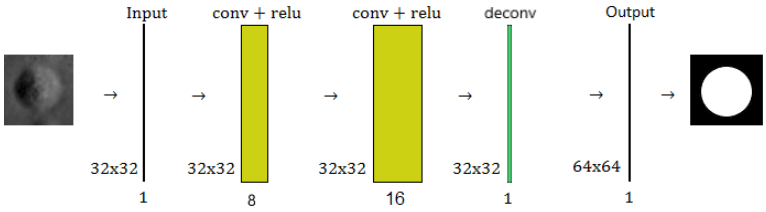


Figure 7.6: Craternet 1_2_0_1.

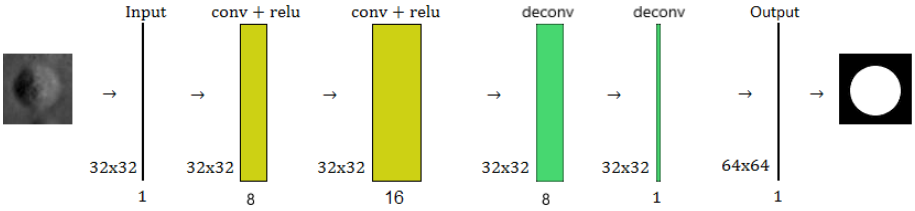


Figure 7.7: Craternet 1_2_1_0.

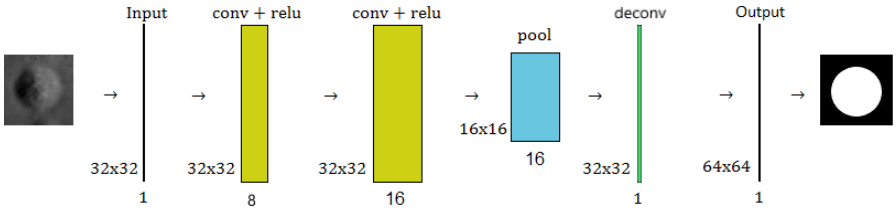


Figure 7.8: Craternet 1_2_1_1.

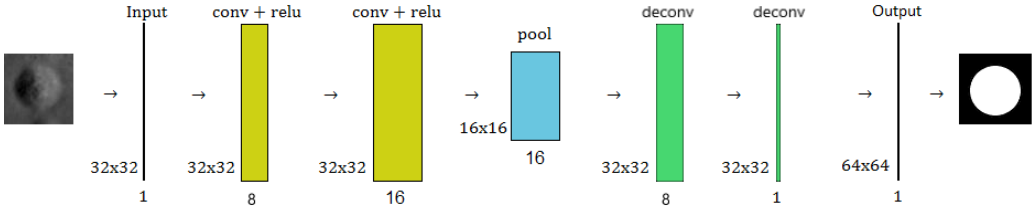


Figure 7.9: Craternet 2_1_0_0.

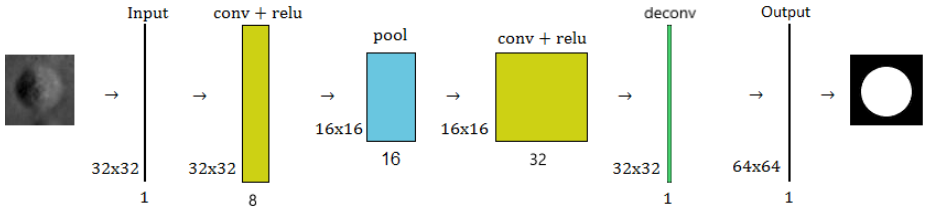


Figure 7.10: Craternet 2_1_0_1.

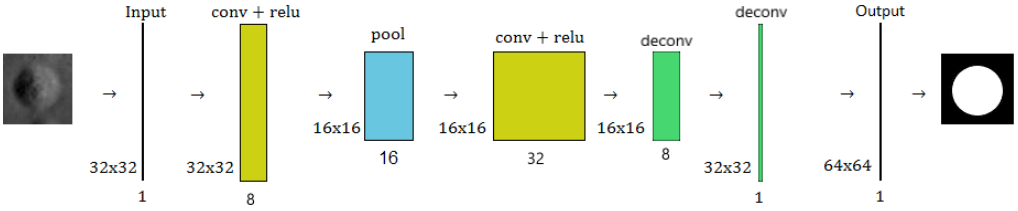


Figure 7.11: Craternet 2_1_1_0.

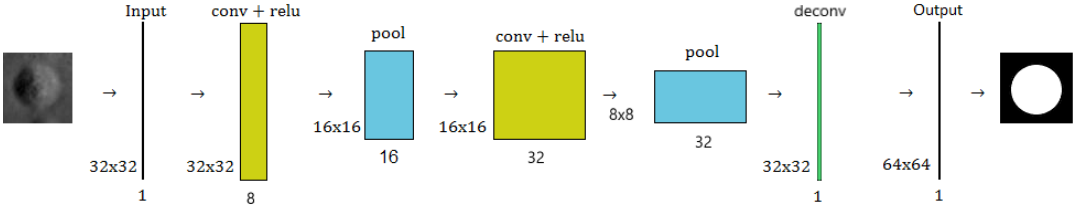


Figure 7.12: Craternet 2_1_1_1.

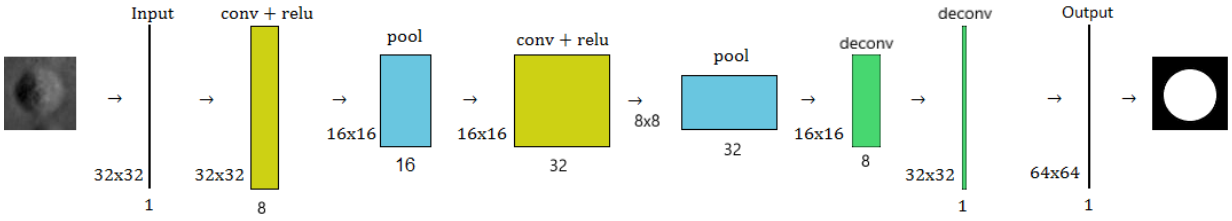


Figure 7.13: Craternet 2_2_0_0.

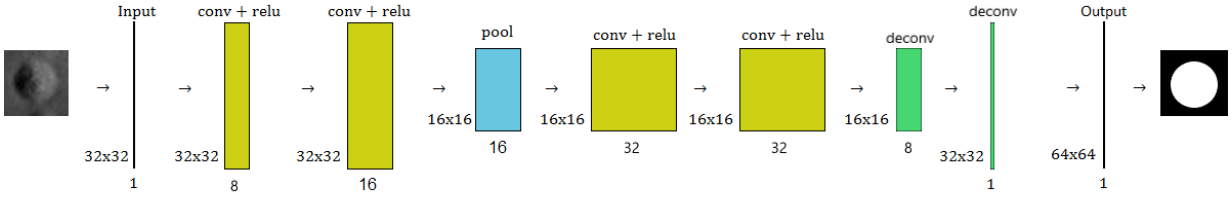


Figure 7.14: Craternet 2_2_0_1.

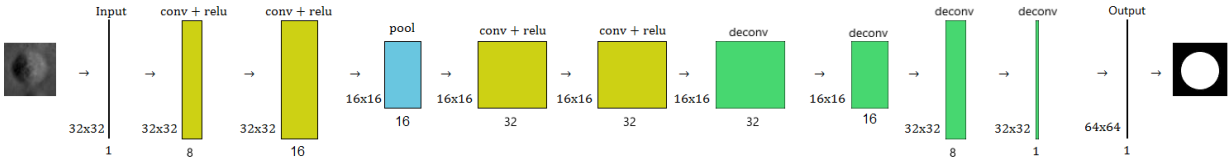


Figure 7.15: Craternet 2_2_1_0.

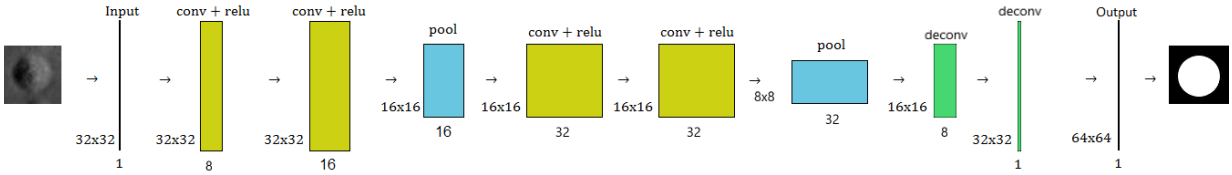


Figure 7.16: Craternet 2_2_1_1.

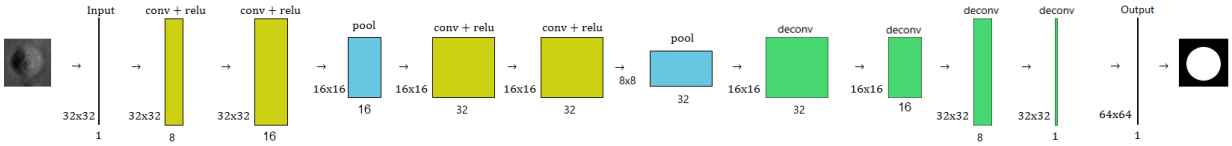


Figure 7.17: Craternet 3_1_0_0.

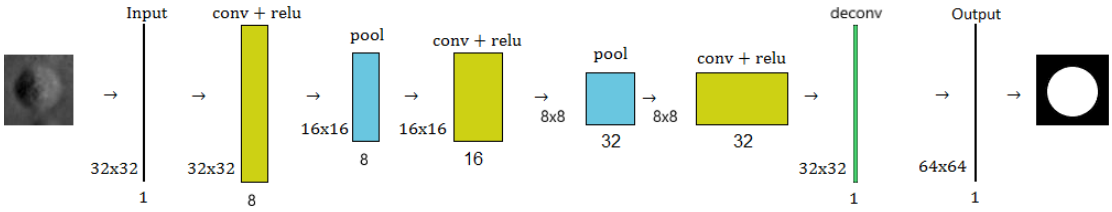


Figure 7.18: Craternet 3_1_0_1.

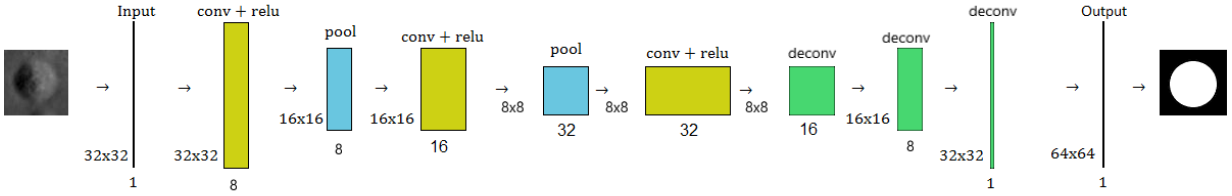


Figure 7.19: Craternet 3_1_1_0.

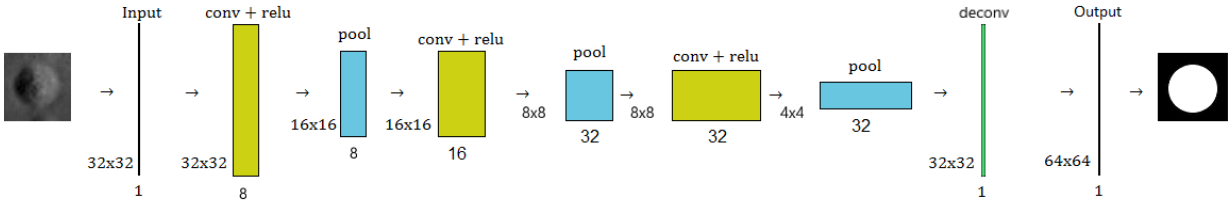


Figure 7.20: Craternet 3_1_1_1.

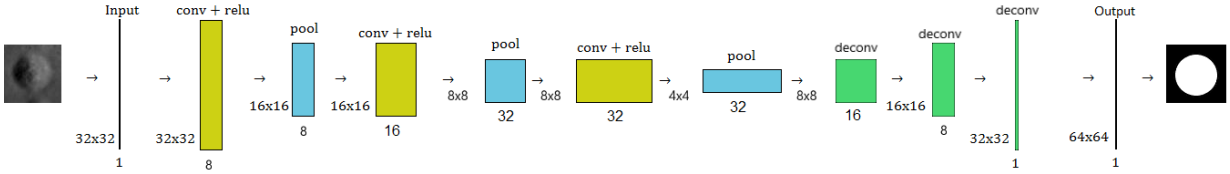


Figure 7.21: Craternet 3_2_0_0.

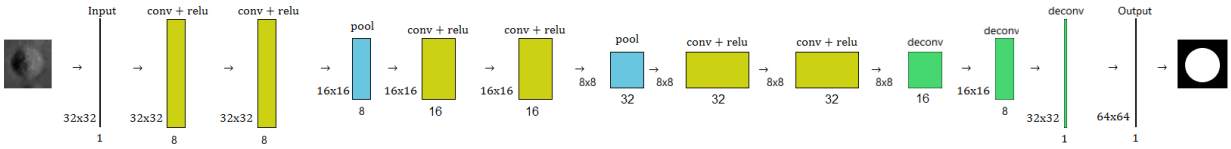


Figure 7.22: Craternet 3_2_0_1.

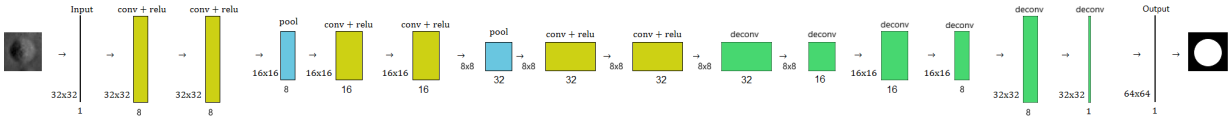


Figure 7.23: Craternet 3_2_1_0.

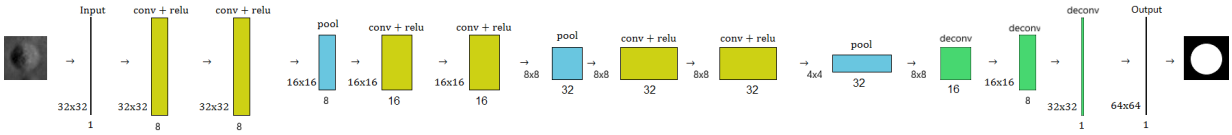
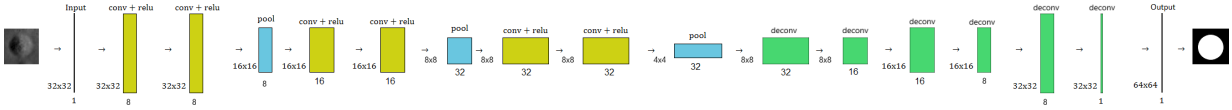


Figure 7.24: Craternet 3_2_1_1.



Filters Trained

Here are displayed some filters from the 3_2_1_1 model, i.e. the network with 3 stages of 2 convolutional layers (+ activation layer) each followed by a pooling layer and a symmetric deconvolution network. Note that the whole network contains 3984 filters.

Figure 7.25: Examples of filters trained from the first convolution layer (patch size = 32x32 pixels).

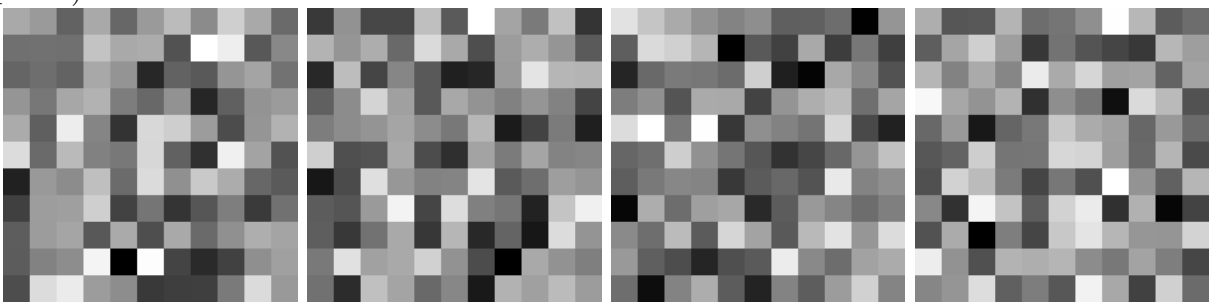


Figure 7.26: Examples of filters trained from the second convolution layer.

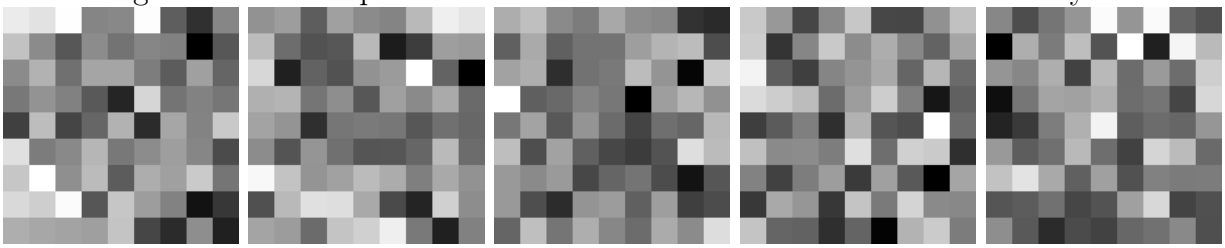


Figure 7.27: Examples of filters trained from the third convolution layer (after one pooling operation, i.e. patch size = 16x16 pixels).

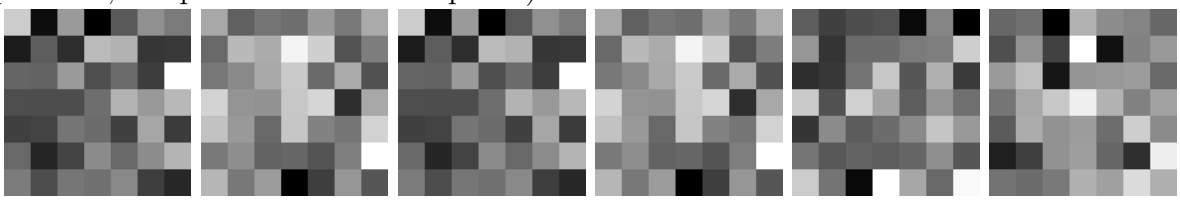


Figure 7.28: Examples of filters trained from the forth convolution layer.

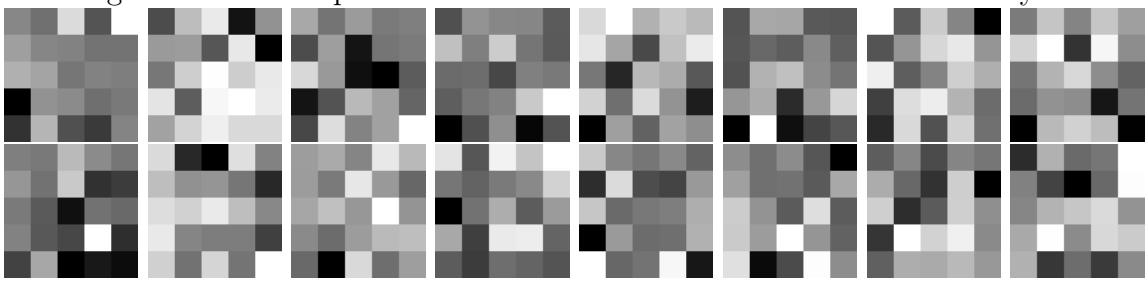


Figure 7.29: Examples of filters trained from the fifth and sixth convolution layers (after two pooling operations, i.e. patch size = 8x8 pixels).

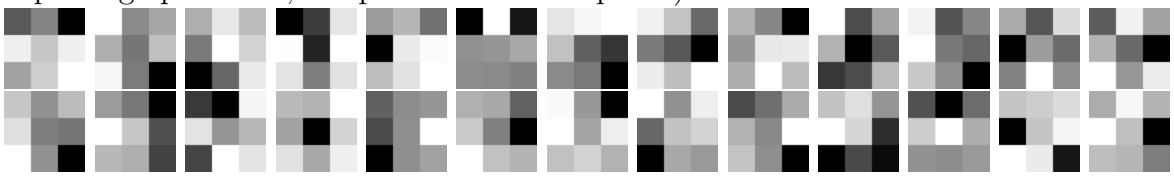


Figure 7.30: Examples of filters trained from the first deconvolution layer (after three pooling operations, i.e. patch size = 4x4 pixels).

Figure 7.31: Examples of filters trained from the second deconvolution layer (after two upsampling operations, i.e. patch size = 8x8 pixels).

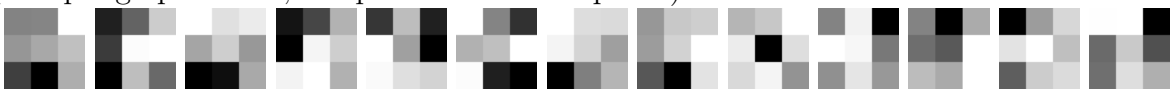


Figure 7.32: Examples of filters trained from the third deconvolution layer.

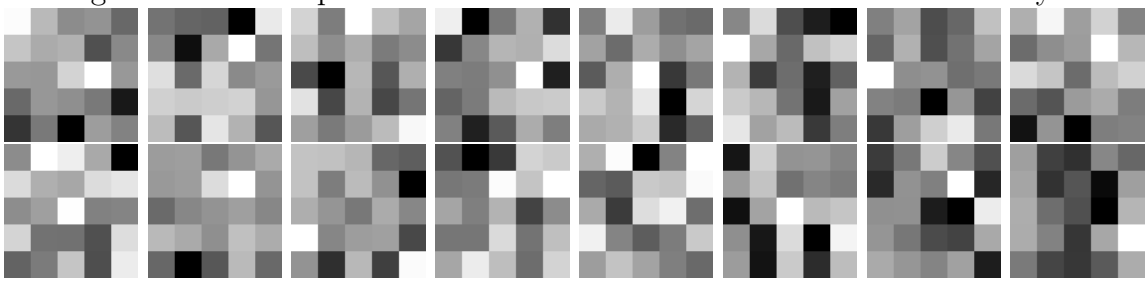


Figure 7.33: Examples of filters trained from the forth deconvolution layer (after two upsampling operations, i.e. patch size = 16x16 pixels).

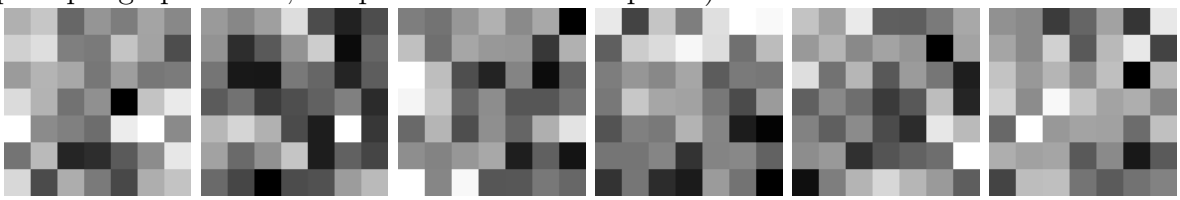


Figure 7.34: Examples of filters trained from the fifth deconvolution layer.

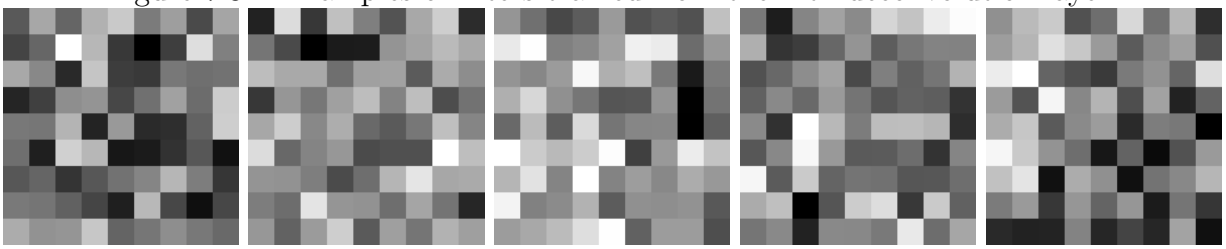
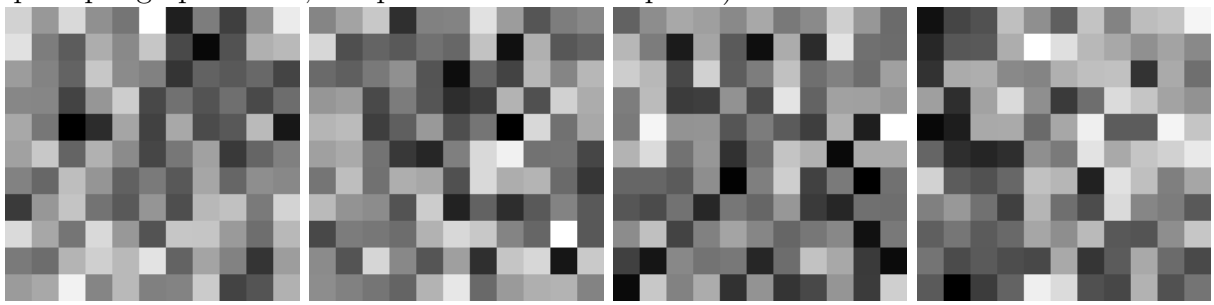


Figure 7.35: Examples of filters trained from the last deconvolution layer (after three upsampling operations, i.e. patch size = 32x32 pixels).



Patches Prediction (test set)

In the following pages, we displayed the semantic segmentation of patches used to test the model (considering only the first order models). In the following examples, *Original* refers to the panchromatic image, *Label* to the ground truth, *Proba* to the probability of the pixel to belong to a crater and *Logits* is finally the predicted class.

CraterNet 2_2_0_1

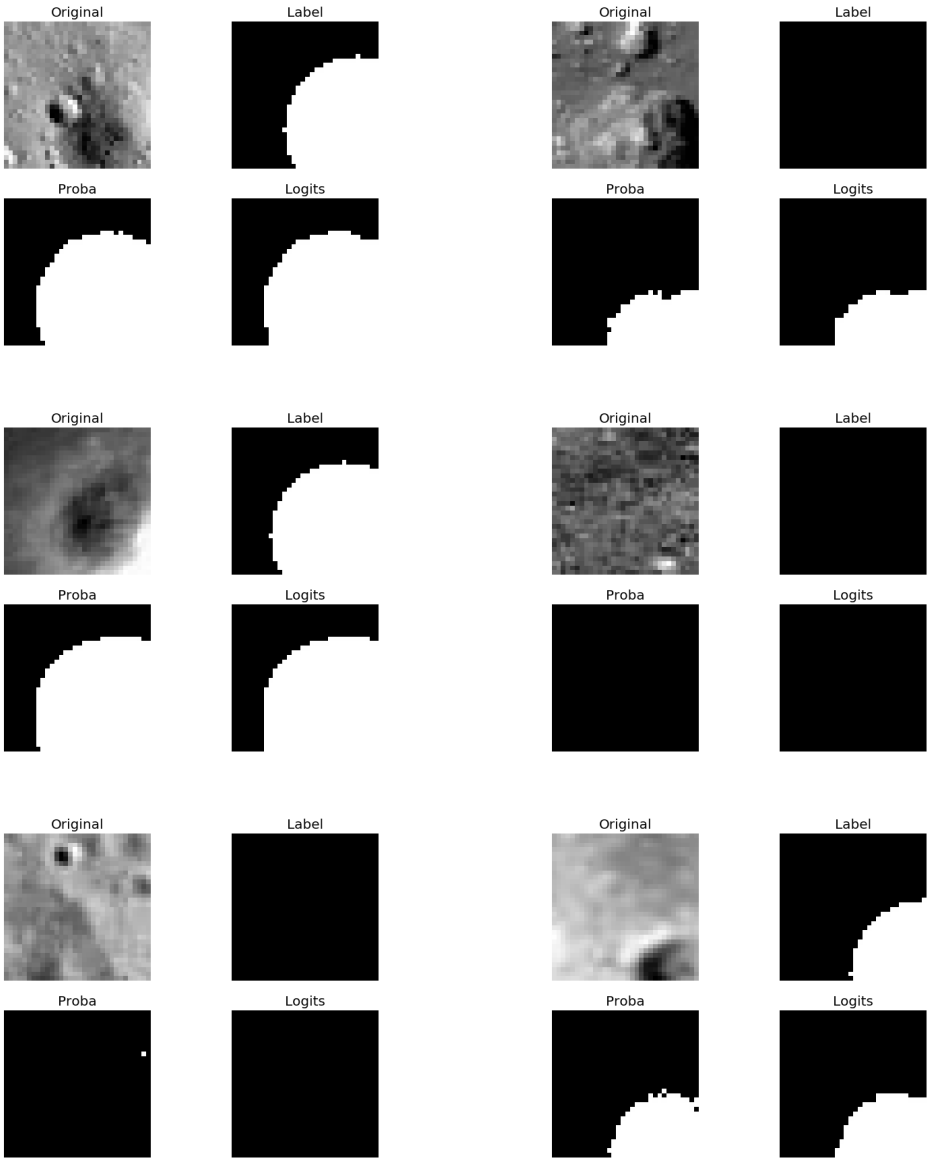


Figure 7.36: Examples of predicted patches. We can notice the ability of the model to recognize craters but also its lack of reliability to detect other topographic elements without *a priori* knowledge (upper right).

CraterNet 2_2_1_0

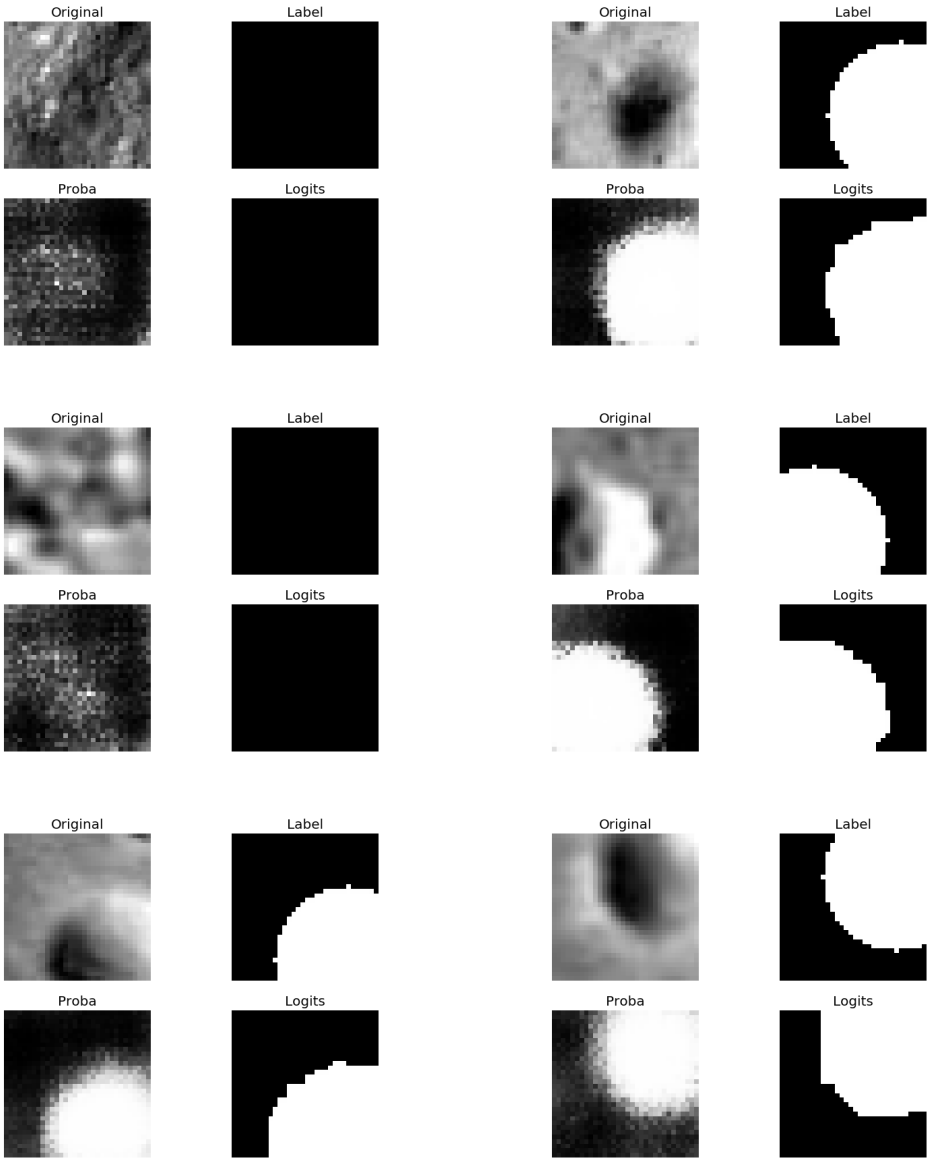


Figure 7.37: Examples of predicted patches. Contrary to the previous one, the model computes here probabilities with a far less degree of certainty.

CraterNet 2_2_1_1

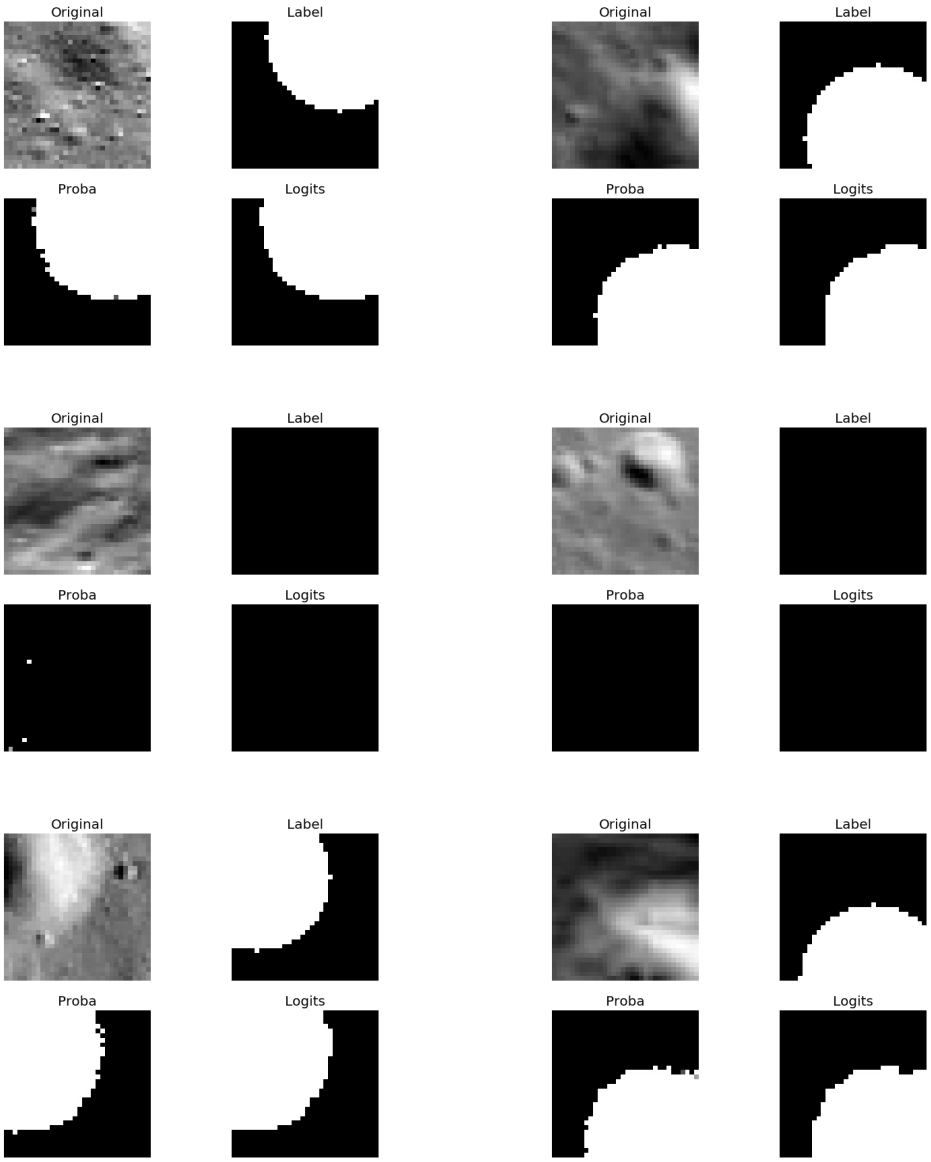


Figure 7.38: Examples of predicted patches. We notice that the model does not get trapped with smaller craters (middle right) or with topographic artifacts (middle left).

CraterNet 3_2_1_0

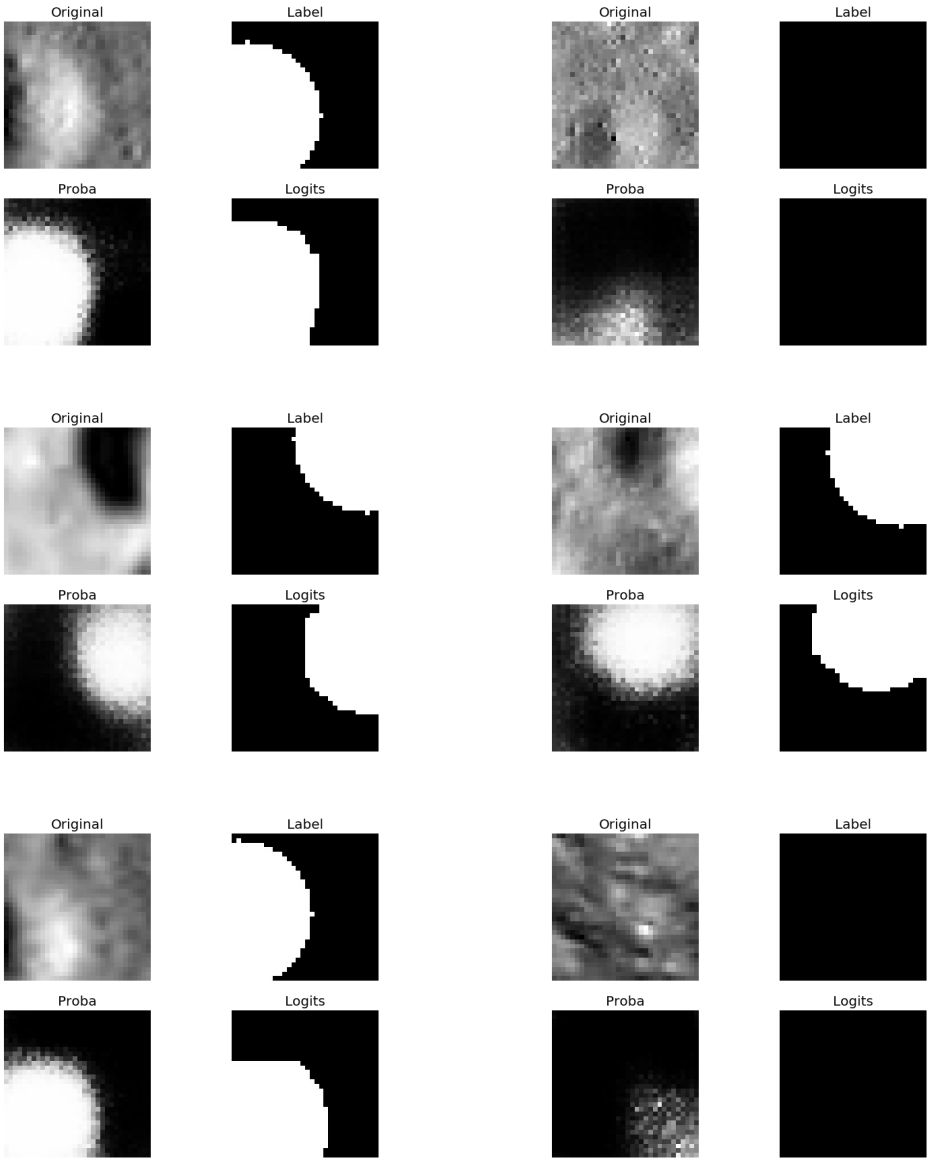


Figure 7.39: Examples of predicted patches. As for CraterNet 2_2_1_0, the non-symmetry of the model prevent the model to compute a sharp probability map.

CraterNet 3_2_1_1

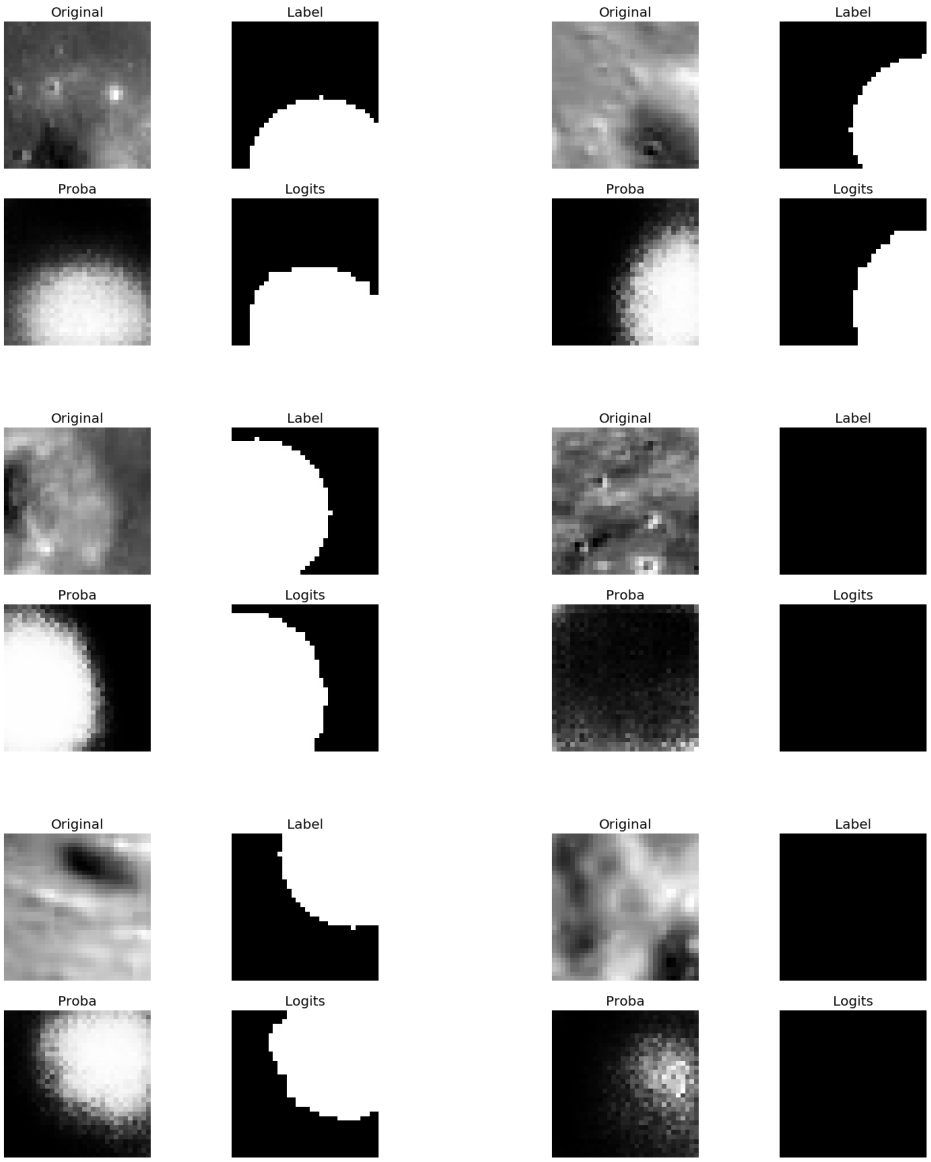


Figure 7.40: Examples of predicted patches.

Scene Parsing

NAC CDR Images

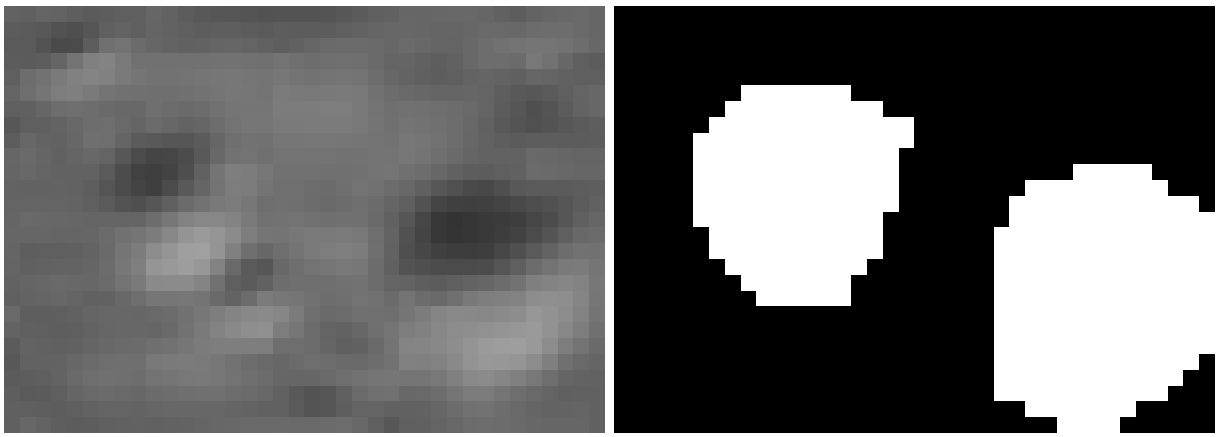


Figure 7.41: Extracting craters of size $\simeq$ 16 pixels.

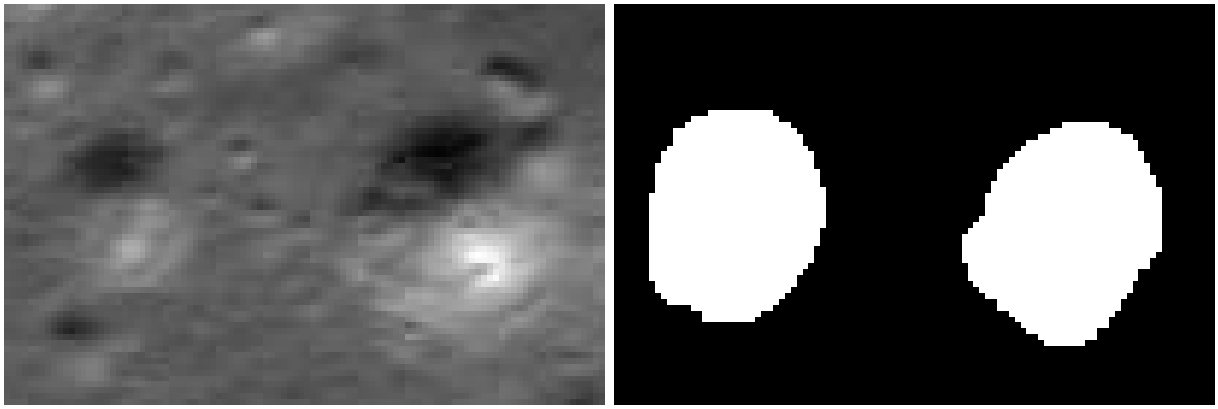


Figure 7.42: Extracting craters of size $\simeq$ 32 pixels.

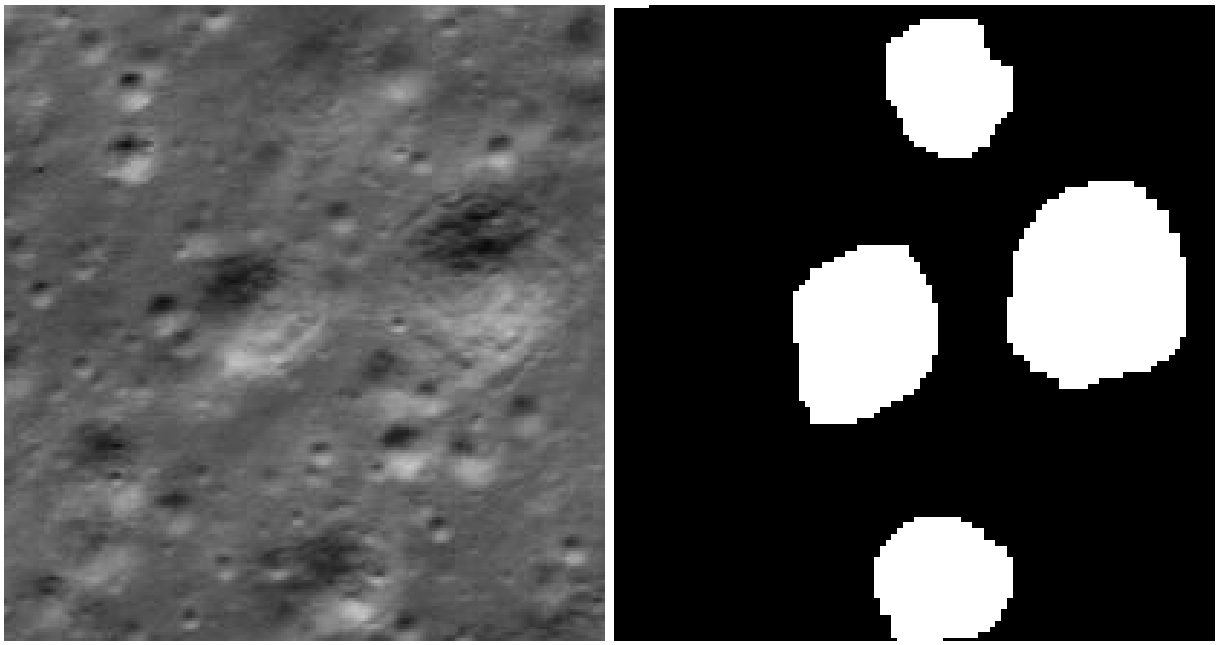


Figure 7.43: Extracting craters of size $\simeq$ 64 pixels.

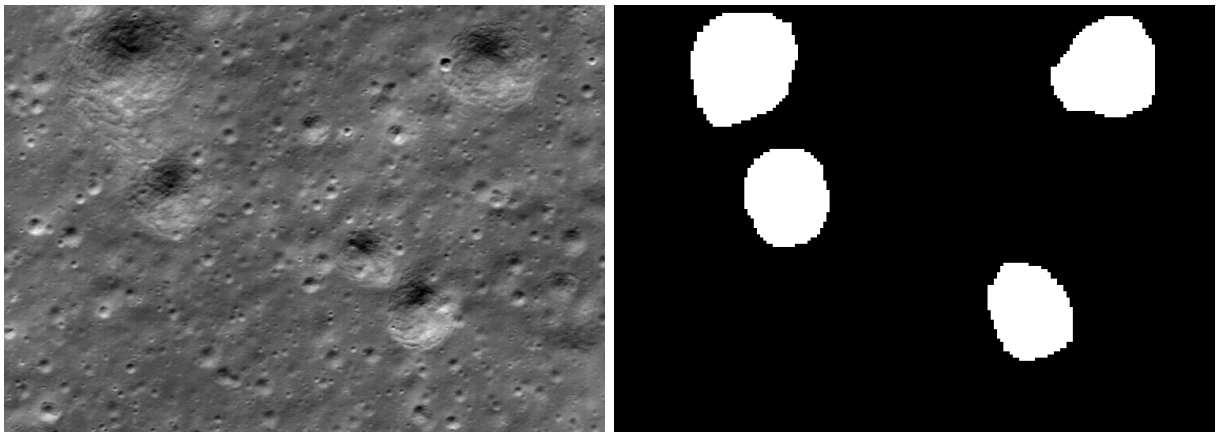


Figure 7.44: Extracting craters of size $\simeq$ 128 pixels.

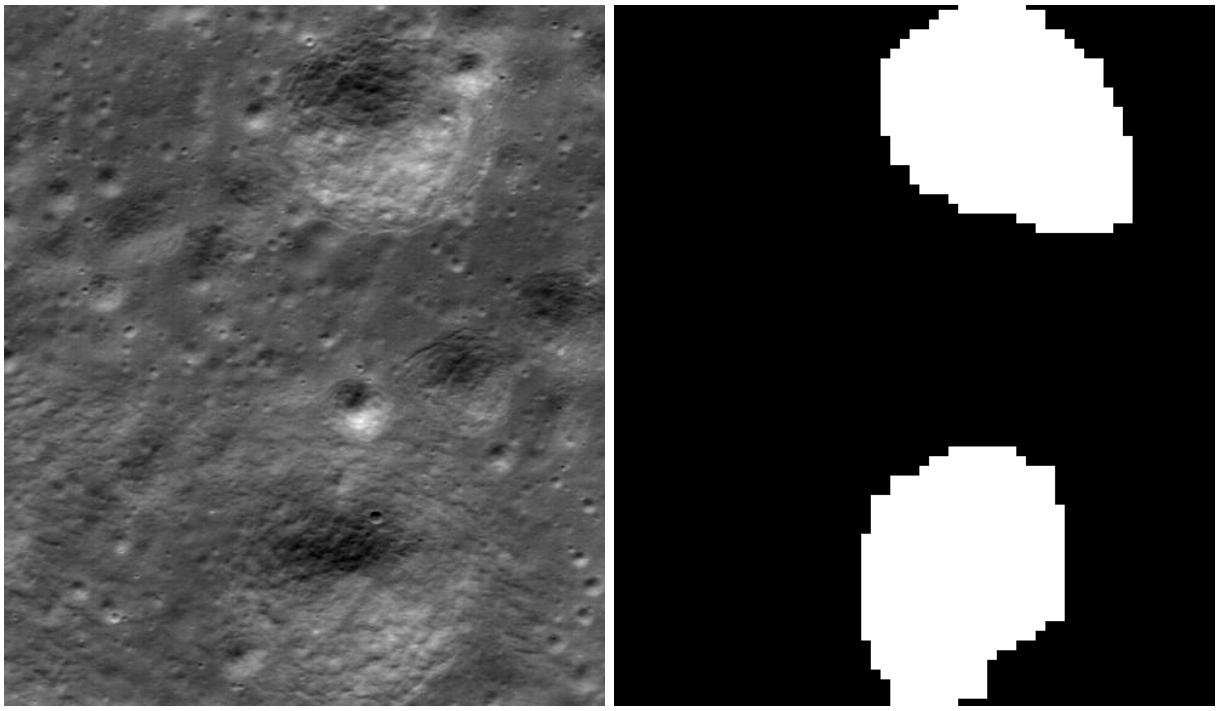


Figure 7.45: Extracting craters of size $\simeq$ 256 pixels.

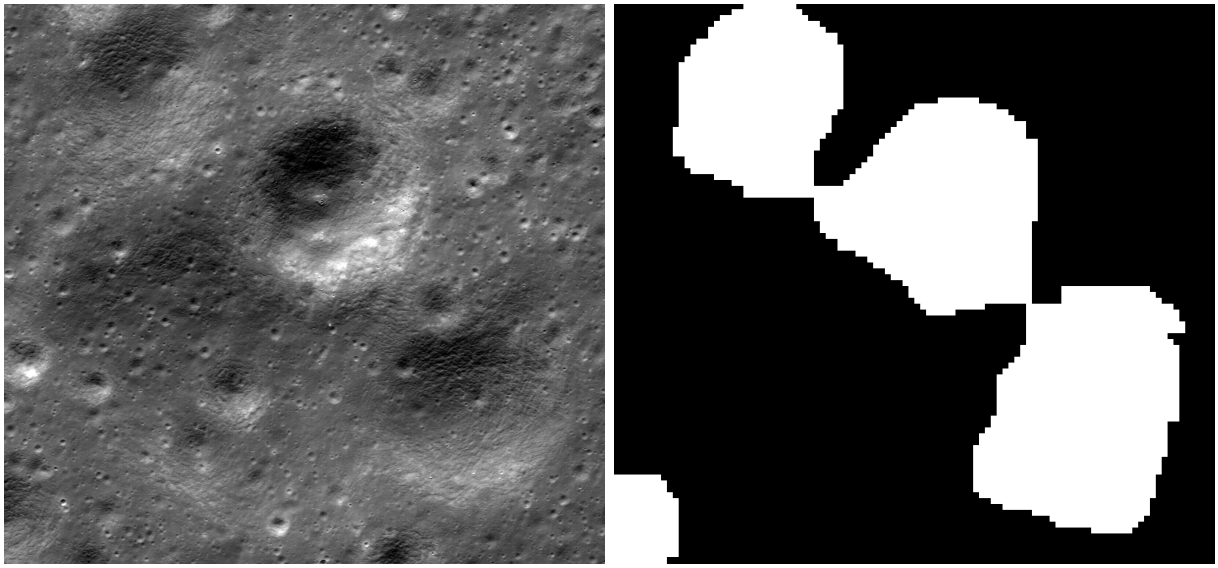


Figure 7.46: Extracting craters of size $\simeq$ 512 pixels.

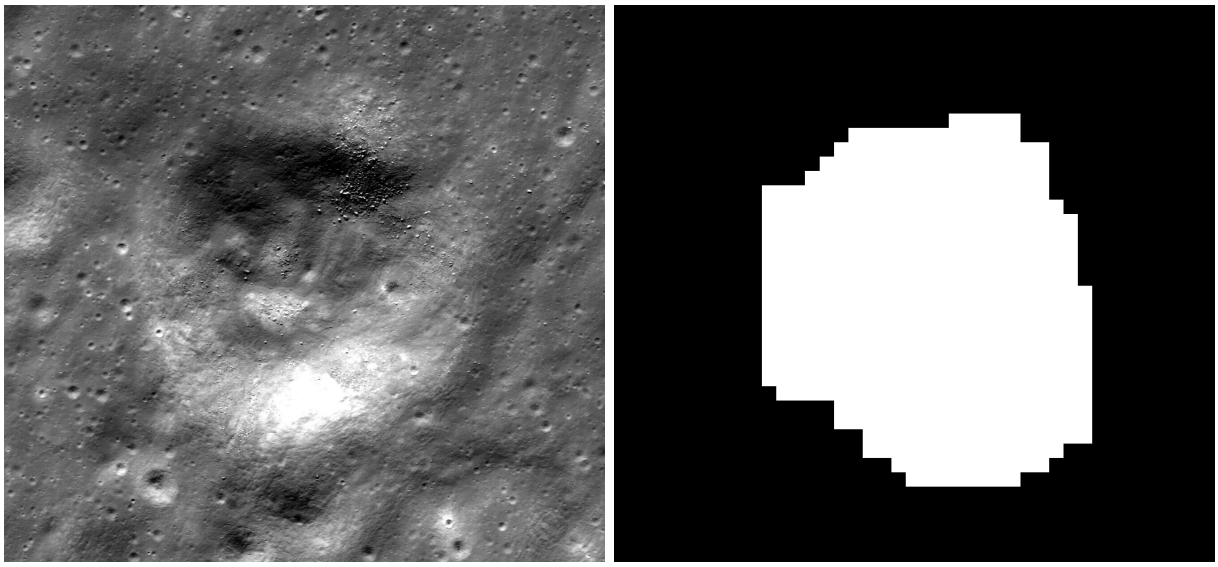


Figure 7.47: Extracting craters of size $\simeq$ 1024 pixels.

Ortho-Images

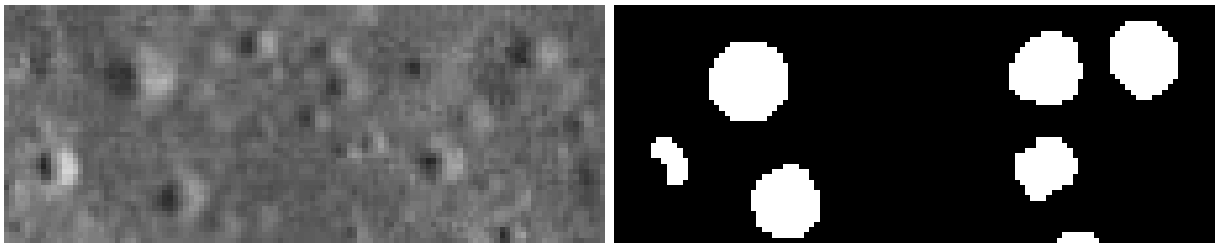


Figure 7.48: Extracting craters of size $\simeq$ 16 pixels.

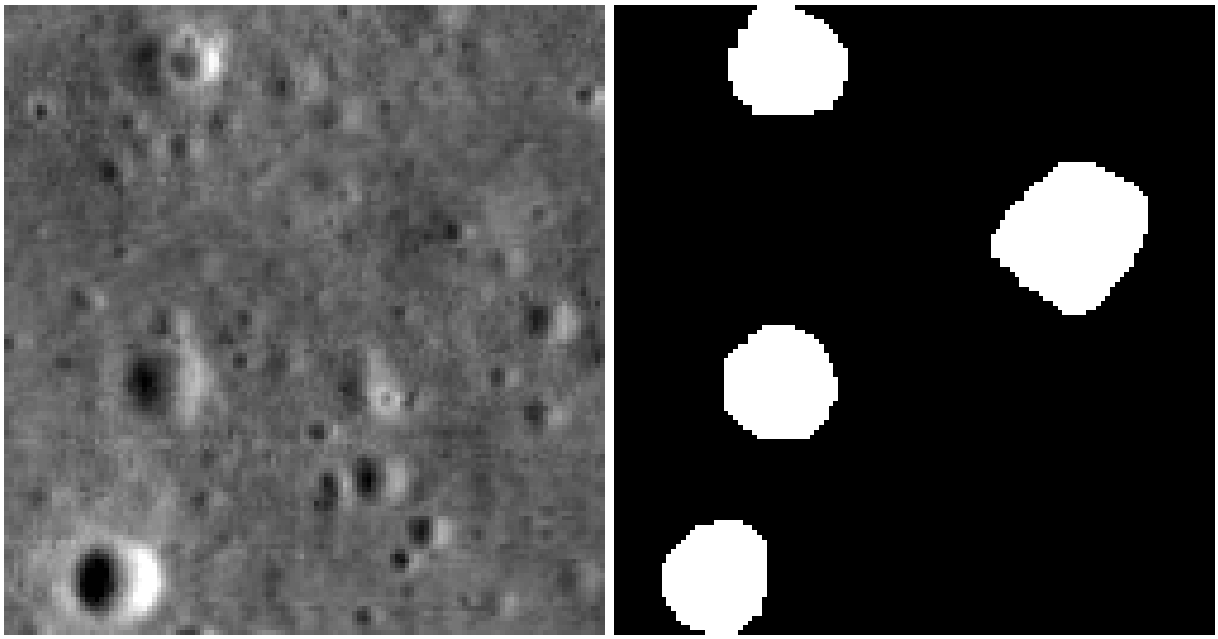


Figure 7.49: Extracting craters of size $\simeq$ 32 pixels.

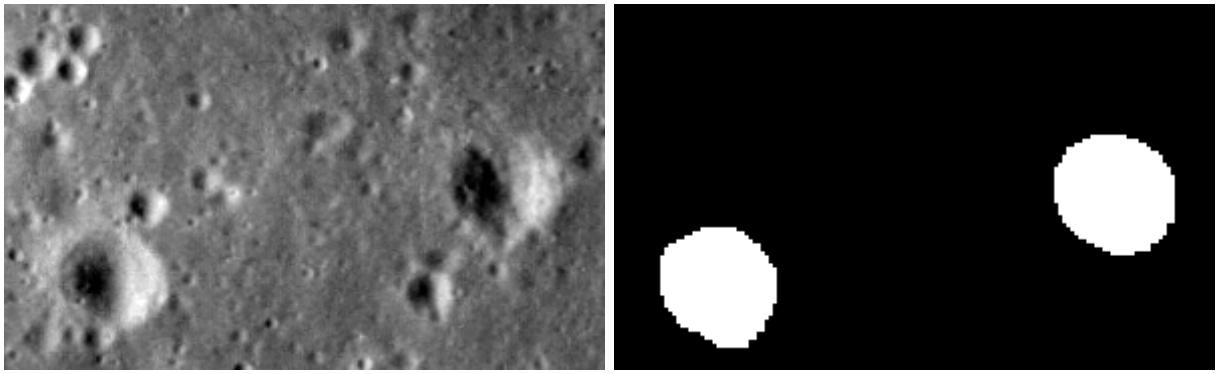


Figure 7.50: Extracting craters of size $\simeq$ 64 pixels.

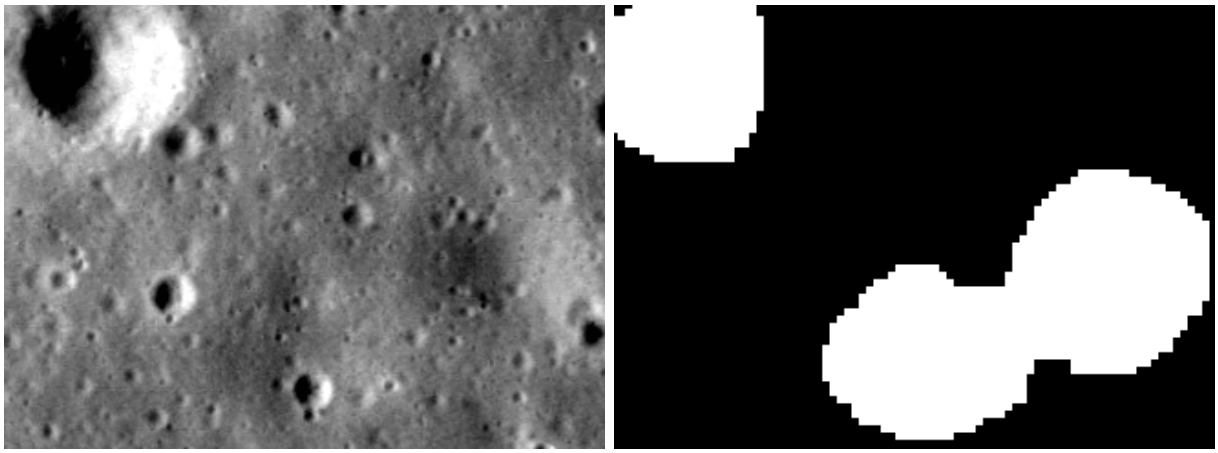


Figure 7.51: Extracting craters of size $\simeq$ 128 pixels.

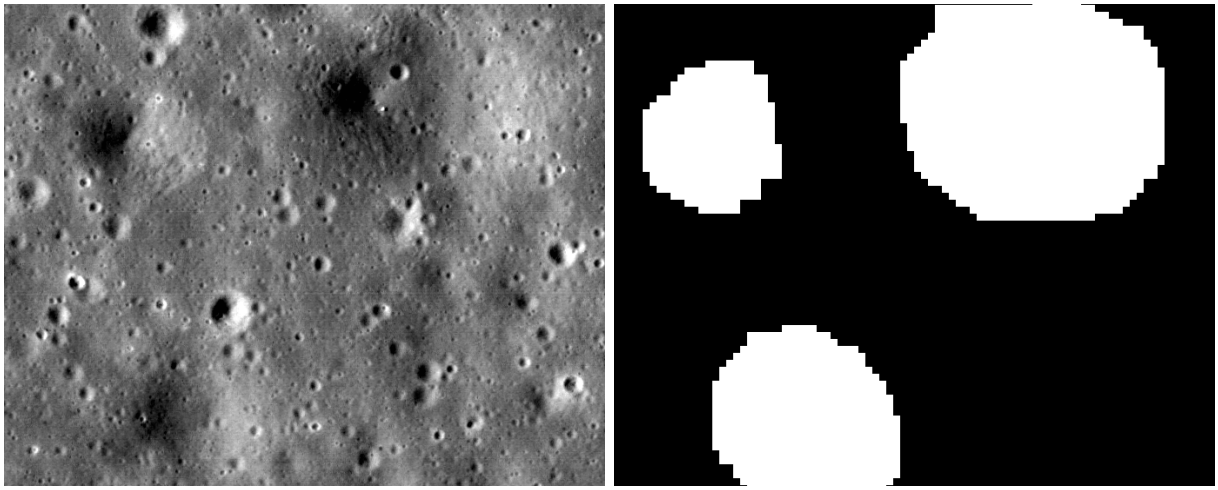


Figure 7.52: Extracting craters of size $\simeq$ 256 pixels.

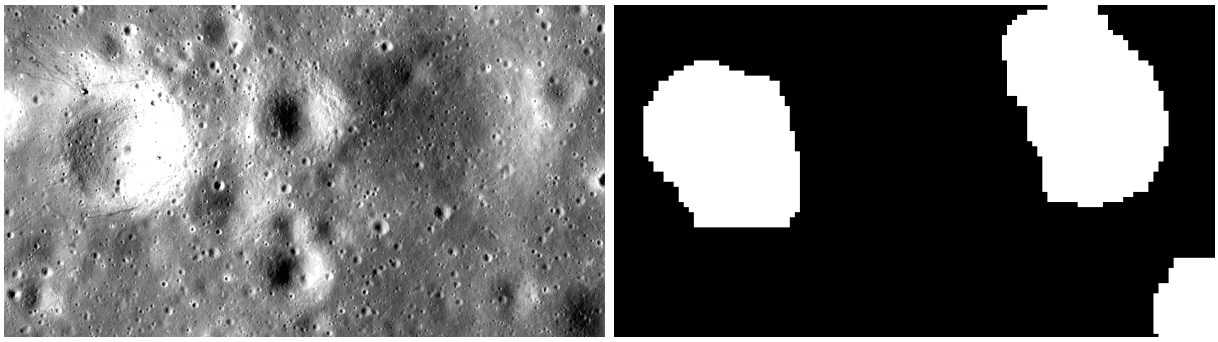


Figure 7.53: Extracting craters of size $\simeq$ 512 pixels.

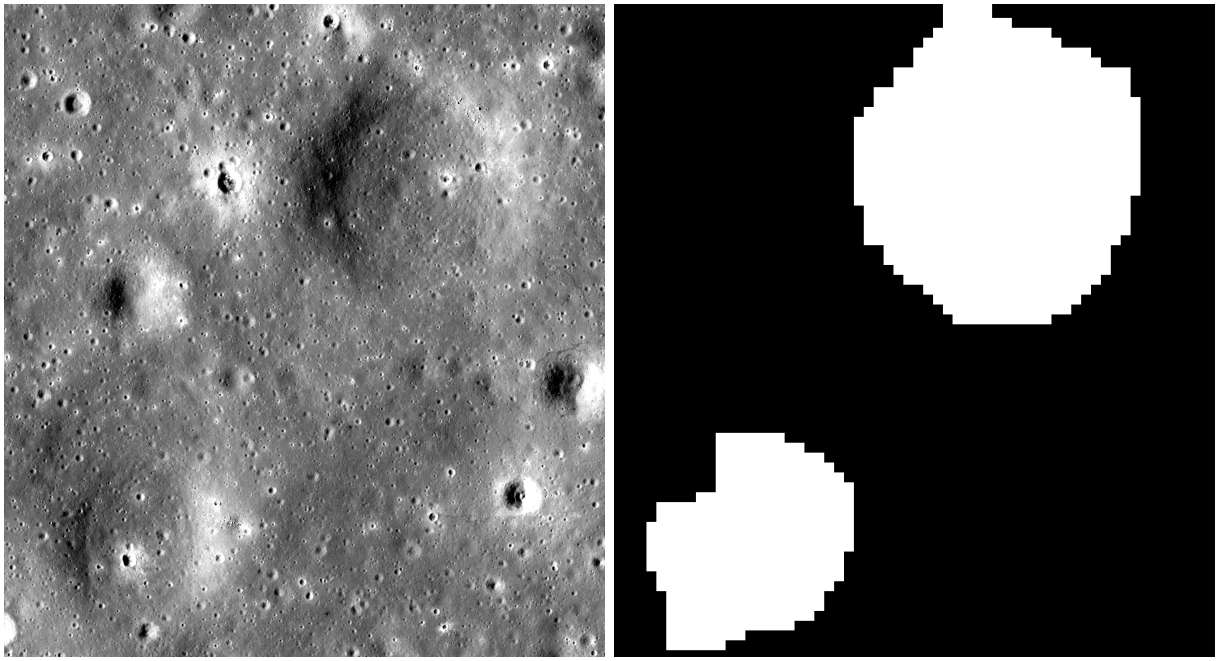


Figure 7.54: Extracting craters of size $\simeq$ 1024 pixels.

Numerical Results

Roc curves

Here are displayed the roc curves of the first order models.

CraterNet 2_2_0_1

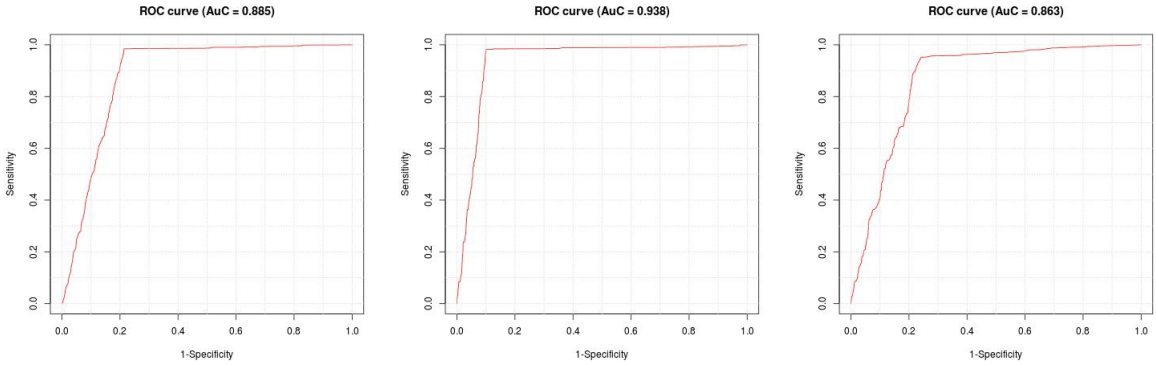


Figure 7.55: From left to right : train set, test set and validation set.

CraterNet 2_2_1_0

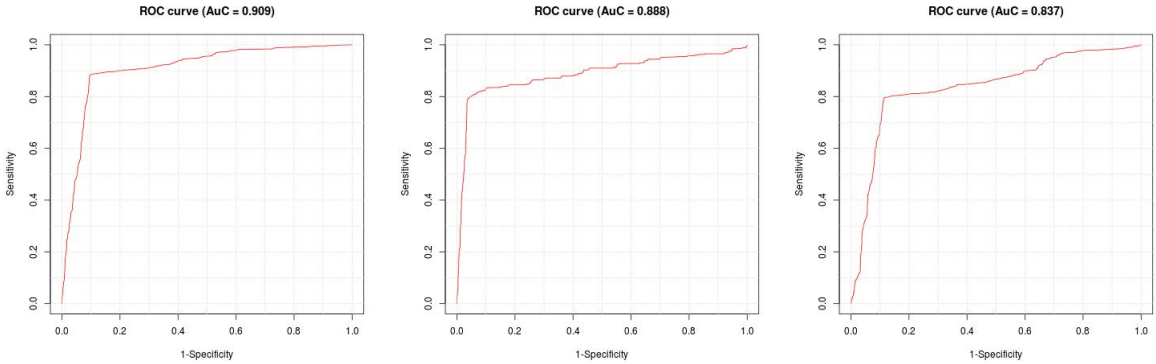


Figure 7.56: From left to right : train set, test set and validation set.

CraterNet 2_2_1_1

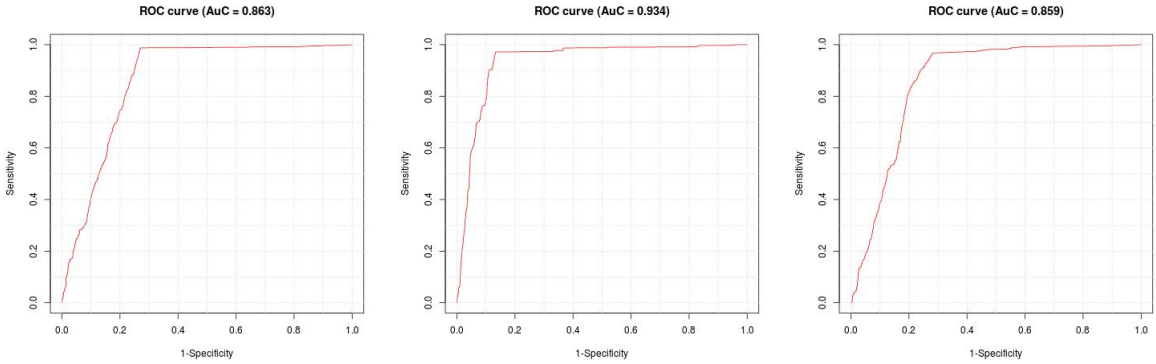


Figure 7.57: From left to right : train set, test set and validation set.

CraterNet 3_2_1_0

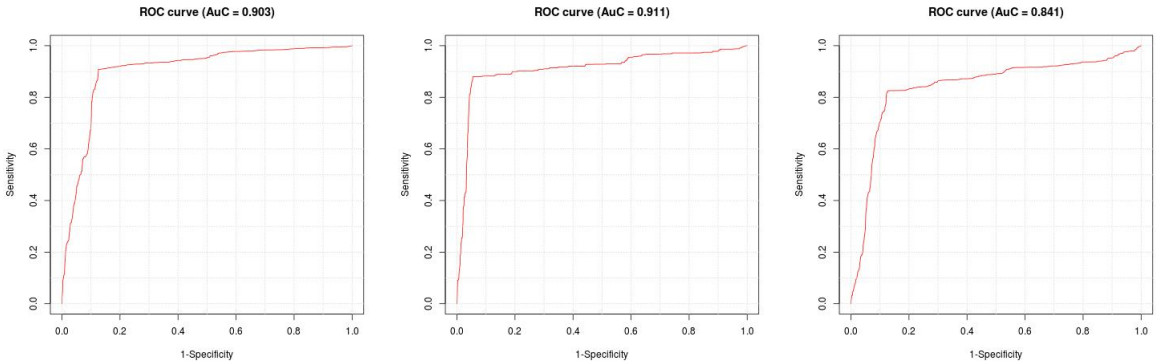


Figure 7.58: From left to right : train set, test set and validation set.

CraterNet 3_2_1_1

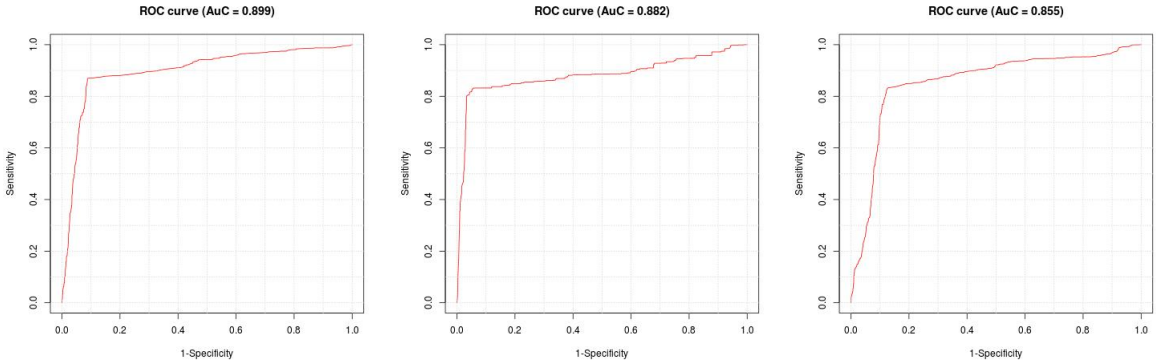


Figure 7.59: From left to right : train set, test set and validation set.

Train Process

Here is displayed the evolution of the loss function value during the train process. The considered models are the first order models.

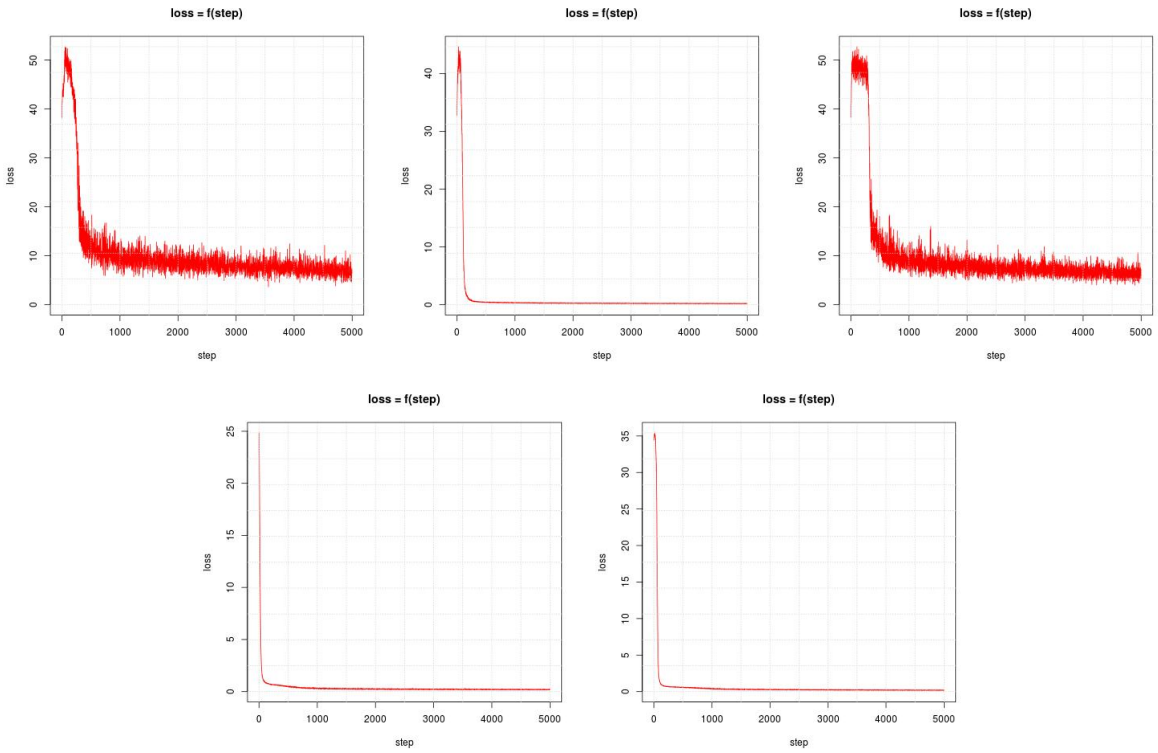


Figure 7.60: From left to right : train set, test set and validation set.

Source Code

Importation Function

```
import glob
import cv2
import random
import numpy as np

def importData(path, imageList, labelList, dim, n_sub, patch_size, label, rotated = True):

      init = len(imageList)

      if label==1:
            labelImage = np.zeros((dim,dim))
            cv2.circle(labelImage,(int(dim/2),int(dim/2)),int(dim/4),1,-1)
      else:
            labelImage = np.zeros((dim,dim))

      path += '*.png'
      for filename in glob.glob(path):
            img = cv2.imread(filename,0)
            img = cv2.resize(img, (dim,dim), interpolation = cv2.INTER_CUBIC)

            for i in range(0,n_sub):
                  rand_x = random.randrange(0, dim-patch_size)
                  rand_y = random.randrange(0, dim-patch_size)

                  sub_img = img[rand_y:rand_y+patch_size,rand_x:rand_x+patch_size]
                  sub_label = labelImage[rand_y:rand_y+patch_size,rand_x:rand_x+patch_size]

                  if rotated:
                        for j in range(4):
                              M = cv2.getRotationMatrix2D((patch_size/2,patch_size/2),j*90,1)
                              img_rotated = cv2.warpAffine(sub_img,M,(patch_size,patch_size))
                              label_rotated = cv2.warpAffine(sub_label,M,(patch_size,patch_size))

                              img_rotated = np.asarray(img_rotated).reshape(-1)
                              label_rotated = np.asarray(label_rotated).reshape(-1)
                              imageList.append(img_rotated)
```

```
                        labelList.append(label_rotated)
          else:
                sub_img = np.asarray(sub_img).reshape(-1)
                sub_label = np.asarray(sub_label).reshape(-1)
                imageList.append(sub_img)
                labelList.append(sub_label)

     print(repr(len(imageList)-init) + ' patches imported')
```

Mini-Batch Creation

```
import random

def miniBatch(imageList, labelList, batchImage, batchLabel, sizeBatch):
     for i in range(0,sizeBatch):
          idx = random.randrange(len(imageTrain))
          batchImage.append(imageList[idx])
          batchLabel.append(labelList[idx])
```

Convolutional Network Functions

```
def weight_variable(shape):
     initial = tf.truncated_normal(shape, stddev=0.1)
     return tf.Variable(initial)

def bias_variable(shape):
     initial = tf.constant(0.1, shape=shape)
     return tf.Variable(initial)

def conv2d(x, W):
     return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
     return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                strides=[1, 2, 2, 1], padding='SAME')
```

R Script Writer

```
def R_script(filename,folder,title,namex,namey,x,y,xmax = 1, ymax = 1):
      f = open('./' + folder + '/'+ filename +'.R', 'w')

      f.write("x = c(")
      for value in x[:-1]:
            f.write(repr(value)+',')
      f.write(repr(x[-1]))
      f.write(")\n")

      f.write("y = c(")
      for value in y[:-1]:
            f.write(repr(value)+',')
      f.write(repr(y[-1]))
      f.write(")\n \n")

      f.write("""plot( x, y, type="l", col="red", xlab=" """+namex+""" ",
            ylab=" """+namey+""" ",main=" """+title+""" ",
            xlim = c(0,"""+repr(xmax)+"""), ylim = c(0,"""+repr(ymax)+"""))""" + '\n')

      doted = np.linspace(0,ymax,11)
      for value in doted:
            f.write("""abline(h = """ + repr(value) + """, col = "lightgray", lty = "dotted",
lwd = par("lwd"))""" + '\n')
      doted = np.linspace(0,xmax,11)
      for value in doted:
            f.write("""abline(v = """ + repr(value) + """, col = "lightgray", lty = "dotted",
lwd = par("lwd"))""" + '\n')

      f.write("dev.copy(jpeg,'" + filename + ".jpg')" + '\n')
      f.write("dev.off()")

      f.close()
```

Error Matrix Computation

```
import numpy as np
```

```
import math
import tensorflow as tf
import sklearn.metrics as skm

def error_matrix(image, label, threshold = 0.5):

      sizeBatch = 100
      step_max = math.floor(len(image)/sizeBatch)
      err_mat = np.zeros((2,2))

      for i in range(0,step_max):
            print('Evaluating the model :  step ' + repr(i+1) + ' of ' + repr(step_max)
            + ' (batch size :  '  + repr(sizeBatch) + ')')

            proba = sess.run(predictions, feed_dict=x_input:  image[i*sizeBatch:(i+1)*sizeBatch])

            proba_flat = proba.flatten()
            y_pred = np.round(proba_flat+0.5-threshold)

            y_true = []
            for sublist in label[i*sizeBatch:(i+1)*sizeBatch]:
            for val in sublist:
            y_true.append(val)

            err_mat = err_mat + skm.confusion_matrix(y_true, y_pred)

      return err_mat
```

Metrics Computation

```
def evaluate(err_mat, printable = True):

      tp, fp, fn, tn = int(err_mat[1,1]), int(err_mat[0,1]), int(err_mat[1,0]), int(err_mat[0,0])

      if printable:
            print('\n\n Confusion Matrix')
            print('GT\P \t 0 \t\t 1')
            print('0 | ' + repr(tn) + '\t\t' + repr(fp))
            print('1 | ' + repr(fn) + '\t\t' + repr(tp))
```

```
    precision = tp / (tp + fp)
    if printable:  print('\n Precision :  '  + repr(precision))

    recall = tp / (tp + fn)
    sensitivity = recall
    specificity = tn / (tn + fp)
    if printable:  print('\n Recall/sensitivity :  '  + repr(recall))

    F1 = 2 * (precision * recall) / (precision + recall)
    if printable:  print('\n F1-score :  '  + repr(F1))

    eps = (tp+tn) / (tp + fn + fp + tn)
    if printable:  print('\n Accuracy :  '  + repr(eps))

    iou = tp / (tp + fn + fp)
    if printable:  print('\n Intersection over Union :  '  + repr(iou) + '\n\n')

    results = [specificity, sensitivity, F1]
    return results
```

ROC Curve

```
import numpy as np
import cv2

def PolyArea(x,y):
    return 0.5*np.abs(np.dot(x,np.roll(y,1))-np.dot(y,np.roll(x,1)))


def roc_curve(images, labels, folder, suffix):

    proba = sess.run(predictions, feed_dict=x_input:  images)
    pred = np.round(proba)

    kernel = np.ones((3,3),np.uint8)
    for i in range(100):
        pred[i] = np.expand_dims(cv2.morphologyEx(pred[i],
                  cv2.MORPH_CLOSE, kernel),axis=2)
```

```
            pred[i] = np.expand_dims(cv2.morphologyEx(pred[i],
                        cv2.MORPH_OPEN, kernel),axis=2)

      y_pred = pred.flatten()

      y_true = np.asarray(labels)
      y_true = y_true.flatten()

      tmp = np.c_[y_pred, y_true]
      tmp = tmp[tmp[:,0].argsort(),]
      tmp = tmp[::-1]

      values = tmp[:,0];
      labels = tmp[:,1];

      TPRs = labels.cumsum(axis=0)
      TPRs = np.concatenate(([0], TPRs))
      TPRs = TPRs / TPRs[-1]

      index_0 = np.where(labels==0)
      index_1 = np.where(labels==1)
      labels[index_0] = 1
      labels[index_1] = 0

      FPRs = labels.cumsum(axis=0)
      FPRs = np.concatenate(([0], FPRs))
      FPRs = FPRs / FPRs[-1]

      AuC = PolyArea(np.append(FPRs,1), np.append(TPRs,0))
      AuC_str = '%.3f' % AuC

      R_script('roc_curve'+suffix[8:], 'model'+suffix[0:8], 'ROC curve (AuC = '+AuC_str+')',
            '1-Specificity','Sensitivity',FPRs[:-1],TPRs[:-1])
```

Visualization of Predicted Patches

```
import cv2
import random
import tensorflow as tf
```

```
from PIL import Image
from matplotlib import pyplot as plt
from matplotlib import gridspec

def plot_predict(image, label, folder, suffix, n = 1):
     kernel = np.ones((3,3),np.uint8)
     for k in range(n):
          i = random.randrange(test_size)
          original, label, proba, logits = sess.run(
                [x_[0,:,:,0 ], y_[0,:,:,0], predictions[0,:,:,0],
                          tf.round(predictions[0,:,:,0])],
               feed_dict=x_input:  imageTest[i:i+1], y_input:  labelTest[i:i+1])

          original_min = original.min()
          original_max = original.max()
          original_image = Image.fromarray((255*(original-original_min))
                          /(original_max-original_min))

          label_min = label.min()
          label_max = label.max()
          label_image = Image.fromarray((255*(label-label_min))/(label_max-label_min))

          proba_min = proba.min()
          proba_max = proba.max()
          proba_image = Image.fromarray((255*(proba-proba_min))/(proba_max-proba_min))

          logits = cv2.morphologyEx(logits, cv2.MORPH_CLOSE, kernel)
          logits = cv2.morphologyEx(logits, cv2.MORPH_OPEN, kernel)
          logits_min = logits.min()
          logits_max = logits.max()
          logits_image = Image.fromarray((255*(logits-logits_min))/(logits_max-logits_min))

          gs = gridspec.GridSpec(2, 2)

          plt.subplot(gs[0, 0]),plt.imshow(original_image,'gray', interpolation='none', vmin=0,
vmax=255),plt.title('Original')
          plt.axis('off')
          plt.subplot(gs[0, 1]),plt.imshow(label_image,'gray', interpolation='none', vmin=0,
vmax=1),plt.title('Label')
          plt.axis('off')

          plt.subplot(gs[1, 0]),plt.imshow(proba_image,'gray', interpolation='none', vmin=0,
vmax=1),plt.title('Proba')
```

```
          plt.axis('off')
          plt.subplot(gs[1, 1]),plt.imshow(logits_image,'gray', interpolation='none', vmin=0,
vmax=1),plt.title('Logits')
          plt.axis('off')

          plt.savefig("./model"+ suffix +'/examples/sample_'+repr(k)+'.png')
```

Extraction of crater objects into CSV file

```
from scipy import ndimage
import cv2
import math
import sys

folder = sys.argv[1]
image = sys.argv[-1]

scale=[]
for i in range(-5,2):  scale.append(pow(2,i))

f = open('./'+folder+ '/'+image[:-4]+'.csv', 'a')
for scaleFactor in scale:
     print('Determining craters of size  ' + repr(int(32/scaleFactor)) + ' meters.')

     mask = cv2.imread('./'+folder+ '/' + image[:-4] +'_label_'+repr(scaleFactor)+'.png',0)

     label_im, nb_labels = ndimage.label(mask)

     sizes = ndimage.sum(mask, label_im, range(nb_labels + 1))

     areaMin = 804 # correspond to pi*Rmin*Rmin with Rmin = 16
     areaMax = 12868 # correspond to pi*Rmax*Rmax with Rmax = 64

     mask_size = sizes < 255*areaMin
     remove_pixel = mask_size[label_im]
     mask[remove_pixel] = 0

     mask_size = sizes > 255*areaMax
     remove_pixel = mask_size[label_im]
```

```
        mask[remove_pixel] = 0

        #cv2.imwrite('./'+folder+ '/'+image+'_'+'filtered_'+repr(scaleFactor)+'.png',mask)

        label_im, nb_labels = ndimage.label(mask)
        print(' -> ' + repr(nb_labels)+' crater(s) found.'  )  # how many craters?

        sizes = ndimage.sum(mask, label_im, range(nb_labels + 1))
        centers = ndimage.measurements.center_of_mass(mask, label_im, range(nb_labels + 1))

        for i in range(1,len(sizes)):
        f.write(repr(int(centers[i][0]/scaleFactor))+','+
        repr(int(centers[i][1]/scaleFactor))+','+
        repr(math.sqrt((sizes[i]/255)/(math.pi*scaleFactor)))+'\n')

        print('\n')

f.close()
```

Data Importation

```
imageTrain = []
labelTrain = []

imageValid = []
labelValid = []

suffix = sys.argv[0][-11:-3]
folder = 'results'

dim_x = 32
dim_y = 32

crop_size = 64
n_crop = 10

path_crater = './NAC_ANNOTATIONS/positive/'
path_no_crater = './NAC_ANNOTATIONS/negative/'

print('\n Import of crater (nac)')
```

```
importData(path_crater,imageTrain,labelTrain,crop_size,n_crop,dim_x,1)
print('\n Import of no crater (nac)')
importData(path_no_crater,imageTrain,labelTrain,crop_size,n_crop,dim_x,0)

path_crater = './ORTHO_ANNOTATIONS/positive/'
path_no_crater = './ORTHO_ANNOTATIONS/negative/'

print('\n Import of crater (ortho)')
importData(path_crater,imageTrain,labelTrain,crop_size,n_crop,dim_x,1)
print('\n Import of no crater (ortho)')
importData(path_no_crater,imageTrain,labelTrain,crop_size,n_crop,dim_x,0)

path_crater = './VALIDATION_SET/positive/'
path_no_crater = './VALIDATION_SET/negative/'

print('\n Import of crater (validation set)')
importData(path_crater,imageValid,labelValid,crop_size,n_crop,dim_x,1)
print('\n Import of no crater (validation set)')
importData(path_no_crater,imageValid,labelValid,crop_size,n_crop,dim_x,0)

# random decomposition of the dataset
imageTest = []
labelTest = []

test_relative = 0.3 # test set = 30% dataset
test_size = int(test_relative*len(imageTrain))

for i in range(0,test_size):
      idx = random.randrange(len(imageTrain))
      imageTest.append(imageTrain[idx])
      labelTest.append(labelTrain[idx])
      del imageTrain[idx]
      del labelTrain[idx]
```

One of The Different Fully Connected Networks

```
# Convolutional Network parameters
kernel_11 = 11
kernel_12 = 9
```

```
kernel_21 = 7
kernel_22 = 5
kernel_31 = 3
kernel_32 = 3

nb_features_11 = 8
nb_features_12 = 8
nb_features_21 = 16
nb_features_22 = 16
nb_features_31 = 32
nb_features_32 = 32

# Input
x_input = tf.placeholder(tf.float32, [None, dim_y*dim_x])
x_ = tf.reshape(x_input, [-1,dim_y,dim_x,1])
y_input = tf.placeholder(tf.float32, [None, dim_y*dim_x])
y_ = tf.reshape(y_input, [-1,dim_y,dim_x,1])

# Convolutional layer 11
W_conv11 = weight_variable([kernel_11, kernel_11, 1, nb_features_11])
b_conv11 = bias_variable([nb_features_11])
h_conv11 = tf.nn.relu(conv2d(x_, W_conv11) + b_conv11)

# Convolutional layer 12
W_conv12 = weight_variable([kernel_12, kernel_12, nb_features_11, nb_features_12])
b_conv12 = bias_variable([nb_features_12])
h_conv12 = tf.nn.relu(conv2d(h_conv11, W_conv12) + b_conv12)

# Pooling layer 1
h_pool1 = max_pool_2x2(h_conv12)

# Convolutional layer 21
W_conv21 = weight_variable([kernel_21, kernel_21, nb_features_12, nb_features_21])
b_conv21 = bias_variable([nb_features_21])
h_conv21 = tf.nn.relu(conv2d(h_pool1, W_conv21) + b_conv21)

# Convolutional layer 22
W_conv22 = weight_variable([kernel_22, kernel_22, nb_features_21, nb_features_22])
b_conv22 = bias_variable([nb_features_22])
h_conv22 = tf.nn.relu(conv2d(h_conv21, W_conv22) + b_conv22)

# Pooling layer 2
h_pool2 = max_pool_2x2(h_conv22)
```

```
# Convolutional layer 31
W_conv31 = weight_variable([kernel_31, kernel_31, nb_features_22, nb_features_31])
b_conv31 = bias_variable([nb_features_31])
h_conv31 = tf.nn.relu(conv2d(h_pool2, W_conv31) + b_conv31)

# Convolutional layer 32
W_conv32 = weight_variable([kernel_32, kernel_32, nb_features_31, nb_features_32])
b_conv32 = bias_variable([nb_features_32])
h_conv32 = tf.nn.relu(conv2d(h_conv31, W_conv32) + b_conv32)

# Pooling layer 3
h_pool3 = max_pool_2x2(h_conv32)

temp_batch_size = tf.shape(x_)[0]

# Deconvolutional layer 32
output_shape32 = tf.stack([temp_batch_size,tf.to_int32(dim_y/8),
                tf.to_int32(dim_x/8),nb_features_31])
W_deconv32 = weight_variable([kernel_32, kernel_32, nb_features_31, nb_features_32])
b_deconv32 = bias_variable([nb_features_31])
h_deconv32 = tf.nn.conv2d_transpose(h_pool3, W_deconv32, output_shape32,
[1,1,1,1], padding='SAME') + b_deconv32

# Deconvolutional layer 31
output_shape31 = tf.stack([temp_batch_size,tf.to_int32(dim_y/4),
                tf.to_int32(dim_x/4),nb_features_22])
W_deconv31 = weight_variable([kernel_31, kernel_31, nb_features_22, nb_features_31])
b_deconv31 = bias_variable([nb_features_22])
h_deconv31 = tf.nn.conv2d_transpose(h_deconv32, W_deconv31, output_shape31,
[1,2,2,1], padding='SAME') + b_deconv31

# Deconvolutional layer 22
output_shape22 = tf.stack([temp_batch_size,tf.to_int32(dim_y/4),
                tf.to_int32(dim_x/4),nb_features_21])
W_deconv22 = weight_variable([kernel_22, kernel_22, nb_features_21, nb_features_22])
b_deconv22 = bias_variable([nb_features_21])
h_deconv22 = tf.nn.conv2d_transpose(h_deconv31, W_deconv22, output_shape22,
[1,1,1,1], padding='SAME') + b_deconv22

# Deconvolutional layer 21
output_shape21 = tf.stack([temp_batch_size,tf.to_int32(dim_y/2),
                tf.to_int32(dim_x/2),nb_features_12])
```

```
W_deconv21 = weight_variable([kernel_21, kernel_21, nb_features_12, nb_features_21])
b_deconv21 = bias_variable([nb_features_12])
h_deconv21 = tf.nn.conv2d_transpose(h_deconv22, W_deconv21, output_shape21,
[1,2,2,1], padding='SAME') + b_deconv21

# Deconvolutional layer 12
output_shape12 = tf.stack([temp_batch_size,tf.to_int32(dim_y/2),
                tf.to_int32(dim_x/2),nb_features_11])
W_deconv12 = weight_variable([kernel_12, kernel_12, nb_features_11, nb_features_12])
b_deconv12 = bias_variable([nb_features_11])
h_deconv12 = tf.nn.conv2d_transpose(h_deconv21, W_deconv12,output_shape12,
[1,1,1,1], padding='SAME') + b_deconv12

# Deconvolutional layer 11
output_shape11 = tf.stack([temp_batch_size,dim_y,dim_x,1])
W_deconv11 = weight_variable([kernel_11, kernel_11, 1, nb_features_11])
b_deconv11 = bias_variable([1])
h_deconv11 = tf.nn.conv2d_transpose(h_deconv12, W_deconv11,output_shape11,
[1,2,2,1], padding='SAME') + b_deconv11

# Probilistic output
predictions = tf.sigmoid(h_deconv11)
```

Loss Function and Optimizer

```
total_loss = -tf.reduce_mean(y_*tf.log(tf.clip_by_value(predictions,1e-31,1.0 - 1e-31))
     +(1-y_)*tf.log(tf.clip_by_value(1-predictions,1e-31,1.0 - 1e-31)))
l1_regularizer = tf.contrib.layers.l1_regularizer(scale=0.000, scope=None)
weights = tf.trainable_variables()
regularization_penalty = tf.contrib.layers.apply_regularization(l1_regularizer, weights)
regularized_loss = total_loss + regularization_penalty
train_step = tf.train.AdamOptimizer(1e-4).minimize(regularized_loss)
```

Training Process

```
sess = tf.InteractiveSession()
```

```
sess.run(tf.global_variables_initializer())

losslist = []
step_max = 5000
sizeBatch = 125
for step in range(0,step_max) :
     batchImage = []
     batchLabel = []
     miniBatch(imageTrain, labelTrain, batchImage, batchLabel, sizeBatch)
     #print('size of batch :  '  + repr(len(batchImage)))
     train_step.run(feed_dict=x_input:  batchImage, y_input:  batchLabel)
     loss = sess.run(regularized_loss, feed_dict=x_input:  batchImage, y_input:  batchLa-
bel)
     losslist.append(loss)
     print('Step '+repr(step+1)+' of '+repr(step_max)+' (batch size :  '  + repr(sizeBatch)+')')

x = np.arange(len(losslist))
R_script('train_evolution','model'+suffix,'loss = f(step)','step','loss',x,losslist,xmax
= len(losslist), ymax = max(losslist))

writer = tf.summary.FileWriter('./'+'model'+suffix+'/log/', sess.graph)
```

Visualization of Trained Filters

```
import cv2
import os

print('\nExporting filters as images.\n')
directory = "./model"+ suffix + "/filters"
if not os.path.exists(directory):
     os.makedirs(directory)

for var, kernel, name in zip([W_conv11,W_conv12,W_conv21,W_conv22,W_conv31,W_conv32,
           W_deconv11,W_deconv12,W_deconv21,W_deconv22,W_deconv31,W_deconv32],
            [kernel_11,kernel_12,kernel_21,kernel_22,kernel_31,kernel_32,
           kernel_11,kernel_12,kernel_21,kernel_22,kernel_31,kernel_32],
            ["conv11","conv12","conv21","conv22","conv31","conv32",
           "deconv11","deconv12","deconv21","deconv22","deconv31","deconv32"]):
filter_tmp = sess.run(var)
```

```
for i in range(len(filter_tmp[0,0,0,:])):
     for j in range(len(filter_tmp[0,0,:,i])):
          x_max = filter_tmp[:,:,j,i].max()
          x_min = filter_tmp[:,:,j,i].min()
          filter_img = 255*(filter_tmp[:,:,j,i]-x_min)/(x_max-x_min)
          filter_img = cv2.resize(filter_img, (20*kernel, 20*kernel),
                         interpolation = cv2.INTER_NEAREST)
          cv2.imwrite(directory+"/"+name+"("+repr(i) + '_' + repr(j) +").png",filter_img)
```

Object-Oriented Evaluation

```
import numpy as np
import math
import sys

folder = sys.argv[1]
image = sys.argv[-1][:-4]

gt = np.genfromtxt('./'+image+'_annotations.csv', delimiter=',')
predict = np.genfromtxt('./'+folder+'/'+image+ '.csv', delimiter=',')

def dist(pos1,pos2):
     d = math.sqrt(math.pow(pos1[0]-pos2[0],2)+math.pow(pos1[1]-pos2[1],2))
     return d

result_raw = open('./'+folder+'/'+image+'_result_raw.csv', 'w')
result_binary = open('./'+folder+'/'+image+'_result_binary.csv', 'w')
ths = 1

for i in range(len(gt)):
     f_min = float("inf")
     for j in range(len(predict)):
          r1 = max([gt[i,2], predict[j,2]])
          r2 = min([gt[i,2], predict[j,2]])
          d = dist([gt[i,0],gt[i,1]],[predict[j,0],predict[j,1]])

          f = max([(r2/r1)-1,d/r2])

          if f < f_min:
```

```
                f_min = f

      result_raw.write(repr(f_min) + '\n')

      if f_min < ths:
            f_min = 1
      else:
            f_min = 0

      result_binary.write(repr(f_min) + '\n')

result_raw.close()
result_binary.close()
```

Saving Parameters

```
saver = tf.train.Saver()
saver = tf.train.Saver()
save_path = saver.save(sess, './'+'model'+suffix+'/model.ckpt')
print("Model saved in file:  %s" % save_path)
```

Restoring Parameters

```
saver = tf.train.Saver()

sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())

saver.restore(sess, './model'+suffix+'/model.ckpt')
print("Model restored.")
```

Analyzing a Scene

```
# limited size of each tile
MAX_DIM = 1000
# overlap (in pixels)
ov = 32
threshold = 0.5
kernel = np.ones((5,5),np.uint8)

studied_region = cv2.imread(image,-1)
orig_h, orig_w = studied_region.shape[:2]

scale=[]
for i in range(-5,2):  scale.append(pow(2,i))

for scaleFactor in scale:

      scene_data = []
      import_scene = cv2.resize(studied_region,None,fx=scaleFactor,
            fy=scaleFactor, interpolation = cv2.INTER_CUBIC)
      h, w = import_scene.shape[:2]
      print(h, w)

      nx, ny = math.floor((w-2*ov)/MAX_DIM), math.floor((h-2*ov)/MAX_DIM)
      print(nx, ny)

      image_label = np.zeros((h, w))
      image_proba = np.zeros((h, w))

      for i in range(0,ny):
            for j in range(0,nx):

                  scene_data =[]
                  sub_img = import_scene[ov+i*MAX_DIM-ov:ov+(i+1)*MAX_DIM+ov,
                        ov+j*MAX_DIM-ov:ov+(j+1)*MAX_DIM+ov]
                  sub_img = np.asarray(sub_img).reshape(-1)
                  scene_data.append(sub_img)

                  proba = sess.run(scene_proba[0,:,:,0],
                        feed_dict=scene_input:  scene_data)
```

```
                logits = np.round(proba+0.5-threshold)

                logits = cv2.morphologyEx(logits, cv2.MORPH_CLOSE, kernel)
                logits = cv2.morphologyEx(logits, cv2.MORPH_OPEN, kernel)

                image_label[ov+i*MAX_DIM:ov+(i+1)*MAX_DIM,
                      ov+j*MAX_DIM:ov+(j+1)*MAX_DIM] = logits[ov:MAX_DIM+ov,ov:MAX_DIM+ov]
                image_proba[ov+i*MAX_DIM:ov+(i+1)*MAX_DIM,
                      ov+j*MAX_DIM:ov+(j+1)*MAX_DIM] = proba[ov:MAX_DIM+ov,ov:MAX_DIM+ov]

      proba_upsampled = cv2.resize(image_proba, (orig_w, orig_h), interpolation = cv2.INTER_NEAREST)
      label_upsampled = cv2.resize(image_label, (orig_w, orig_h), interpolation = cv2.INTER_NEAREST)

      #scipy.misc.toimage(proba_upsampled, cmin=0.0, cmax=1.0).save("./model"+ suffix
      # +'/'+image[:-4]+'_proba_'+repr(scaleFactor)+'.png')
      scipy.misc.toimage(label_upsampled, cmin=0.0, cmax=1.0).save("./model"+ suffix
            +'/'+image[:-4]+'_label_'+repr(scaleFactor)+'.png')
```

Extraction of Cytomine's Annotations in PNG format

```
from cytomine import Cytomine
from cytomine.models import *

#Connection parameters to Cytomine
cytomine_host="demo.cytomine.be"
cytomine_public_key="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
cytomine_private_key=""XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
#Connection to Cytomine Core

conn = Cytomine(cytomine_host, cytomine_public_key, cytomine_private_key, base_path = '/api/',
working_path = '/tmp/', verbose= True)

id_project=19653296 # ULG-GEO-CRATERES
#id_term=19653322 # Positive
#id_term=19653341 # Negative

annotations = conn.get_annotations(
      id_project = id_project,
      #id_image = id_image,
```

```
      #id_term = id_term,
      showWKT=False,
      showMeta=False,
      showGIS=False,
      reviewed_only = False
)
#note that it is possible to specify image or term

print "Number of annotations:  %d" %len(annotations.data())

dump_type=1 #1:  original image, 2:  with alpha mask, 3:  only alpha mask
zoom_level=0 #0 is maximum resolution
output_dir="/the/desired/path/" #local directory where to dump annotation cropped images
if dump_type==1:
      annotation_get_func = Annotation.get_annotation_crop_url
elif dump_type==2:
      annotation_get_func = Annotation.get_annotation_alpha_crop_url
else:
      annotation_get_func = Annotation.get_annotation_mask_url

#Note:  if file already exists locally, they will not be requested again
print "Downloading annotations into %s ..." %output_dir
dump_annotations=conn.dump_annotations(
      annotations = annotations,
      get_image_url_func = annotation_get_func,
      dest_path = output_dir,
      desired_zoom = zoom_level
)
```

Extraction of Cytomine's Annotations in CSV format

```
# Great thank you to Romain Mormont for his help on these lines of code
import os
import sys
from math import pi, sqrt
from cytomine import Cytomine
from shapely.wkt import loads

#Connection parameters to Cytomine
```

```
cytomine_host="demo.cytomine.be"
cytomine_public_key="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
cytomine_private_key=""XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"

#Connection to Cytomine Core
cytomine = Cytomine(cytomine_host, cytomine_public_key, cytomine_private_key, base_path =
'/api/', working_path = '/tmp/', verbose= True)

id_project = 19653296
project = cytomine.get_project(id_project)
instances = cytomine.get_project_image_instances(id_project=id_project)
instances = {inst.id:  inst for inst in instances}
annotations = cytomine.get_annotations(id_project=id_project, showMeta=True, showWKT=True)

print("{} annotations to save..".format(len(annotations)))
with open("project_{}_annotations.csv".format(id_project), "w+") as file:
     file.write(";".join(["id", "x", "y", "radius", "label", "id_image"]) + "\n")
     for annotation in annotations.data():
          polygon = loads(annotation.location)
          if annotation.image not in instances:
               sys.stderr.write("Annotation {} contained in an image ({}) out of project
{}".format(annotation.id, annotation.image, id_project) + os.linesep)
               continue
          instance = instances[annotation.image]

          term =""
          if ",".join(map(str, annotation.term)) == "19653322":
               term = 1
          else:
               term = 0

          data = [
               str(annotation.id),
               str(polygon.centroid.x),
               str(polygon.centroid.y),
               str(sqrt((polygon.area * instance.resolution * instance.resolution)/pi)),
               str(term),
               str(instance.instanceFilename)
          ]
          file.write(";".join(data) + "\n")
```

Bibliography

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `http://tensorflow.org/` (Software available from tensorflow.org)

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, *abs/1511.00561*. Retrieved from `http://arxiv.org/abs/1511.00561`

Bandeira, L., Ding, W., & Stepinski, T. F. (2012). Detection of sub-kilometer craters in high resolution planetary images using shape and texture features. Advances in Space Research.

Bandeira, L. P. C., Saraiva, J., & Pina, P. (2007). Impact crater recognition on mars based on a probability volume created by template matching. *IEEE Trans. Geoscience and Remote Sensing*, *45*(12-1), 4008–4015. Retrieved from `http://dx.doi.org/10.1109/TGRS.2007.904948` doi: 10.1109/TGRS.2007.904948

Basu, S., Ganguly, S., Mukhopadhyay, S., DiBiano, R., Karki, M., & Nemani, R. R. (2015). Deepsat: a learning framework for satellite imagery. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems, bellevue, wa, usa, november 3-6, 2015* (pp. 37:1–37:10). Retrieved from `http://doi.acm.org/10.1145/2820783.2820816` doi: 10.1145/2820783.2820816

Bradski, G. (2000). The opencv reference manual. *Dr. Dobb's Journal of Software Tools*.

Catmog concepts and techniques in modern geography. (n.d.). Geo Books. Retrieved from `https://books.google.be/books?id=1uLCnQAACAAJ`

Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, *abs/1412.7062*. Retrieved from `http://arxiv.org/abs/1412.7062`

Cohen, J. P., Lo, H. Z., Lu, T., & Ding, W. (2016). Crater detection via convolutional neural networks. *CoRR*, *abs/1601.00978*. Retrieved from `http://arxiv.org/abs/1601.00978`

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3),

273–297. Retrieved from https://doi.org/10.1007/BF00994018 doi: 10.1007/BF00994018

Cracknell, A. P. (1998). Synergy in remote sensing-what's in a pixel? *International Journal of Remote Sensing*, *19*(11), 2025-2047. Retrieved from http://dx.doi.org/10.1080/014311698214848 doi: 10.1080/014311698214848

Dillencourt M., Samet H. and Tamminen M. (1992, April). A general approach to connected-component labeling for arbitrary image representations. *J. ACM*, *39*(2), 253–280. Retrieved from http://doi.acm.org/10.1145/128749.128750 doi: 10.1145/128749.128750

Ding, W., Stepinski, T. F., Bandeira, L., Vilalta, R., Wu, Y., Lu, Z., & Cao, T. (2010). Automatic detection of craters in planetary images: An embedded framework using feature selection and boosting. In *Proceedings of the 19th acm international conference on information and knowledge management* (pp. 749–758). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1871437.1871534 doi: 10.1145/1871437.1871534

French, B. M. (1998). *Traces of catastrophe : A handbook of shock-metamorphic effects in terrestrial meteorite impact structures*. Lunar and Planetary Institute: Houston.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119 - 139. Retrieved from http://www.sciencedirect.com/science/article/pii/S002200009791504X doi: http://dx.doi.org/10.1006/jcss.1997.1504

Geomatics, P. (2016, February). *Optimized image processing by tiles.*

Geurts, P., & Wehenkel, L. (2016). *Introduction to machine learning.* University Lecture.

Guru, D., Kumar, Y. S., & Manjunath, S. (2011). Textural features in flower classification. *Mathematical and Computer Modelling*, *54*(3–4), 1030 - 1036. Retrieved from http://www.sciencedirect.com/science/article/pii/S0895717710005236 (Mathematical and Computer Modeling in agriculture (CCTA 2010)) doi: http://dx.doi.org/10.1016/j.mcm.2010.11.032

Haar, A. (1910). Zur theorie der orthogonalen funktionensysteme. (erste mitteilung). *Mathematische Annalen*, *69*, 331-371. Retrieved from http://eudml.org/doc/158469

Hien, D. H. T. (2017). A guide to receptive field arithmetic for convolutional neural networks. Retrieved from https://medium.com/@nikasa1889/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

Ho, T. K. (1995). Random decision forests. In *Third international conference on document analysis and recognition, ICDAR 1995, august 14 - 15, 1995, montreal, canada. volume I* (pp. 278–282). Retrieved from https://doi.org/10.1109/ICDAR.1995.598994 doi: 10.1109/ICDAR.1995.598994

Honda, R., & Azuma, R. (2000). Crater extraction and classification system for lunar images. *Mem. Fac. Sci. Kochi Univ.*, *21*, 13 - 22.

Hu, M.-K. (1962, February). Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, *8*(2), 179-187. doi: 10.1109/TIT.1962.1057692

Hu, W., Huang, Y., Li, W., Zhang, F., & Li, H. (2015). Deep convolutional neural networks for hyperspectral image classification. *J. Sensors*, *2015*, 258619:1–258619:12.

Retrieved from http://dx.doi.org/10.1155/2015/258619 doi: 10.1155/2015/258619

Kadhim, N. M. S. M., Mourshed, M., & Bray, M. T. (2015). Shadow detection from very high resoluton satellite image using grabcut segmentation and ratio-band algorithms. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-3/W2*, 95–101. Retrieved from http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-3-W2/95/2015/ doi: 10.5194/isprsarchives-XL-3-W2-95-2015

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems 25: 26th annual conference on neural information processing systems 2012. proceedings of a meeting held december 3-6, 2012, lake tahoe, nevada, united states.* (pp. 1106–1114). Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

Landsafe: 10/10 pour spacebel et ses partenaires académiques. (2014). http://www.spacebel.be/fr/landsafe-1010-pour-spacebel-et-ses-\\partenaires-academiques/.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the ieee* (Vol. 86, pp. 2278–2324). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665

Li, F.-F., Karpathy, A., & Johnson, J. (n.d.). Cs231n: Convolutional neural networks for visual recognition 2016. Retrieved from http://cs231n.stanford.edu/

Lroc edr/cdr data product software interface specification. (2010).

Lro mission overview. (2015). https://www.nasa.gov/mission_pages/LRO/overview/index.html.

Lv, Q., Dou, Y., Niu, X., Xu, J., Xu, J., & Xia, F. (2015). Urban land use and land cover classification using remotely sensed SAR data through deep belief networks. *J. Sensors*, *2015*, 538063:1–538063:10. Retrieved from http://dx.doi.org/10.1155/2015/538063 doi: 10.1155/2015/538063

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml* (Vol. 30).

Marée, R., Rollus, L., Stévens, B., Hoyoux, R., Louppe, G., Vandaele, R., ... Wehenkel, L. (2016). Collaborative analysis of multi-gigapixel imaging data using cytomine. *Bioinformatics*, *32*(9), 1395. Retrieved from +http://dx.doi.org/10.1093/bioinformatics/btw013 doi: 10.1093/bioinformatics/btw013

Meng, D., Yunfeng, C., & Qingxian, W. (2009). Method of passive image based crater autonomous detection. *Chinese Journal of Aeronautics*, *22*(3), 301-306. Retrieved from http://www.sciencedirect.com/science/article/pii/S100093610860103X doi: 10.1016/S1000-9361(08)60103-X

Mohan, R. (2014). Deep deconvolutional networks for scene parsing. *CoRR*, *abs/1411.4101*. Retrieved from http://arxiv.org/abs/1411.4101

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning*

(icml-10), june 21-24, 2010, haifa, israel (pp. 807–814). Retrieved from http://www.icml2010.org/papers/432.pdf

Newell, A., & Simon, H. A. (1956). The logic theory machine-a complex information processing system. *IRE Trans. Information Theory*, *2*(3), 61–79. Retrieved from https://doi.org/10.1109/TIT.1956.1056797 doi: 10.1109/TIT.1956.1056797

Oh, K.-S., & Jung, K. (2004). Gpu implementation of neural networks. *Pattern Recognition*, *37*(6), 1311 - 1314. Retrieved from http://www.sciencedirect.com/science/article/pii/S0031320304000524 doi: http://doi.org/10.1016/j.patcog.2004.01.013

Ozdemir, B., Aksoy, S., Eckert, S., Pesaresi, M., & Ehrlich, D. (2010). Performance measures for object detection evaluation. *Pattern Recognition Letters*, *31*(10), 1128 - 1137. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167865509002918 (Pattern Recognition in Remote SensingFifth {IAPR} Workshop on Pattern Recognition in Remote Sensing (PRRS 2008)) doi: http://dx.doi.org/10.1016/j.patrec.2009.10.016

Quinlan, J. R. (1986, March). Induction of decision trees. *Mach. Learn.*, *1*(1), 81–106. Retrieved from http://dx.doi.org/10.1023/A:1022643204877 doi: 10.1023/A:1022643204877

Raschka, S. (2015). *Python machine learning*. Packt Publishing.

Renson, P. (2012). *Détection des cratères à la surface de la Lune sur des produits planétaires (Dans le cadre du projet LandSAfe)* (Unpublished master's thesis). Université de Liège, Belgium.

Renson P., V. Y., Poncelet N., & Y., C. (2014). Automatisation de la détection des cratères lunaires sur des images et mnt planétaires. *Société Géographique de Liège*, *61*.

Rojas, R. (1996). *Neural networks - A systematic introduction*. Springer.

Rufenacht, D., Fredembach, C., & Susstrunk, S. (2014, August). Automatic and accurate shadow detection using near-infrared information. *IEEE Trans. Pattern Anal. Mach. Intell.*, *36*(8), 1672–1678. Retrieved from http://dx.doi.org/10.1109/TPAMI.2013.229 doi: 10.1109/TPAMI.2013.229

S. Ruder. (2016). An overview of gradient descent optimization algorithms. *CoRR*, *abs/1609.04747*. Retrieved from http://arxiv.org/abs/1609.04747

Salamunićcar, G., Lončarić, S., & Mazarico, E. (2012). Lu60645gt and ma132843gt catalogues of lunar and martian impact craters developed using a crater shape-based interpolation crater detection algorithm for topography data. *Planetary and Space Science*, *60*(1), 236 - 247. Retrieved from http://www.sciencedirect.com/science/article/pii/S0032063311002789 (Titan Through Time: A Workshop on Titan's Formation, Evolution and Fate) doi: http://dx.doi.org/10.1016/j.pss.2011.09.003

Salamunićcara, G., & Lončarićb, S. (2008). Open framework for objective evaluation of crater detection algorithms with first test-field subsystem based on {MOLA} data. *Advances in Space Research*, *42*(1), 6 - 19. Retrieved from http://www.sciencedirect.com/science/article/pii/S0273117707003675 doi: http://dx.doi.org/10.1016/j.asr.2007.04.028

Shank, R. C., & Tesler, L. (1969). A conceptual dependency parser for natural lan-

guage. In *Third international conference on computational linguistics, COLING 1969, stockholm, sweden, september 1-4, 1969.* Retrieved from http://aclweb.org/anthology/C/C69/C69-0201.pdf

Shelhamer, E., Long, J., & Darrell, T. (2016). Fully convolutional networks for semantic segmentation. *CoRR*, *abs/1605.06211*. Retrieved from http://arxiv.org/abs/1605.06211

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, *abs/1409.1556*. Retrieved from http://arxiv.org/abs/1409.1556

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958. Retrieved from http://dl.acm.org/citation.cfm?id=2670313

Stockman, G., & Shapiro, L. G. (2001). *Computer vision* (1st ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., ... Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, *abs/1409.4842*. Retrieved from http://arxiv.org/abs/1409.4842

T., L. (n.d.). *Géologie de la lune et définition des reliefs lunaires.* http://jeromegrenier.free.fr/atlaspdf/geologie.pdf. (Accessed: 2016-09-30)

Turing, A. M. (1950). Computers & thought. In E. A. Feigenbaum & J. Feldman (Eds.), (pp. 11–35). Cambridge, MA, USA: MIT Press. Retrieved from http://dl.acm.org/citation.cfm?id=216408.216410

Urbach, E. R., & Stepinski, T. F. (2009). Automatic detection of sub-km craters in high resolution planetary images. *Planetary and Space Science*, *57*(7), 880 - 887. Retrieved from http://www.sciencedirect.com/science/article/pii/S0032063309000956 doi: http://dx.doi.org/10.1016/j.pss.2009.03.009

Vaduva, C., Gavat, I., & Datcu, M. (2012). Deep learning in very high resolution remote sensing image information mining communication concept. In *Proceedings of the 20th european signal processing conference, EUSIPCO 2012, bucharest, romania, august 27-31, 2012* (pp. 2506–2510). Retrieved from http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6334194

Van Droogenbroeck P. (2016). *Computer vision.* University Lecture.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 ieee computer society conference on computer vision and pattern recognition. cvpr 2001* (Vol. 1, p. I-511-I-518 vol.1). doi: 10.1109/CVPR.2001.990517

Walter, R. (1994). *The secret guide to computers.* R. Walter. Retrieved from https://books.google.be/books?id=0KEoUPiPncQC

Wang, Y., Ding, W., Yu, K., Wang, H., & Wu, X. (2011). Crater detection using bayesian classifiers and lasso..

Wetzler, P. G., Honda, R., Enke, B. L., Merline, W. J., Chapman, C. R., & Burl, M. C. (2005). Learning to detect small impact craters. In *7th IEEE workshop*

on applications of computer vision / IEEE workshop on motion and video computing (WACV/MOTION 2005), 5-7 january 2005, breckenridge, co, USA (pp. 178–184). Retrieved from https://doi.org/10.1109/ACVMOT.2005.68 doi: 10.1109/ACVMOT.2005.68

Young, N., & Evans, A. N. (2003, Oct). Psychovisually tuned attribute operators for pre-processing digital video. *IEE Proceedings - Vision, Image and Signal Processing*, *150*(5), 277-86-. doi: 10.1049/ip-vis:20030768

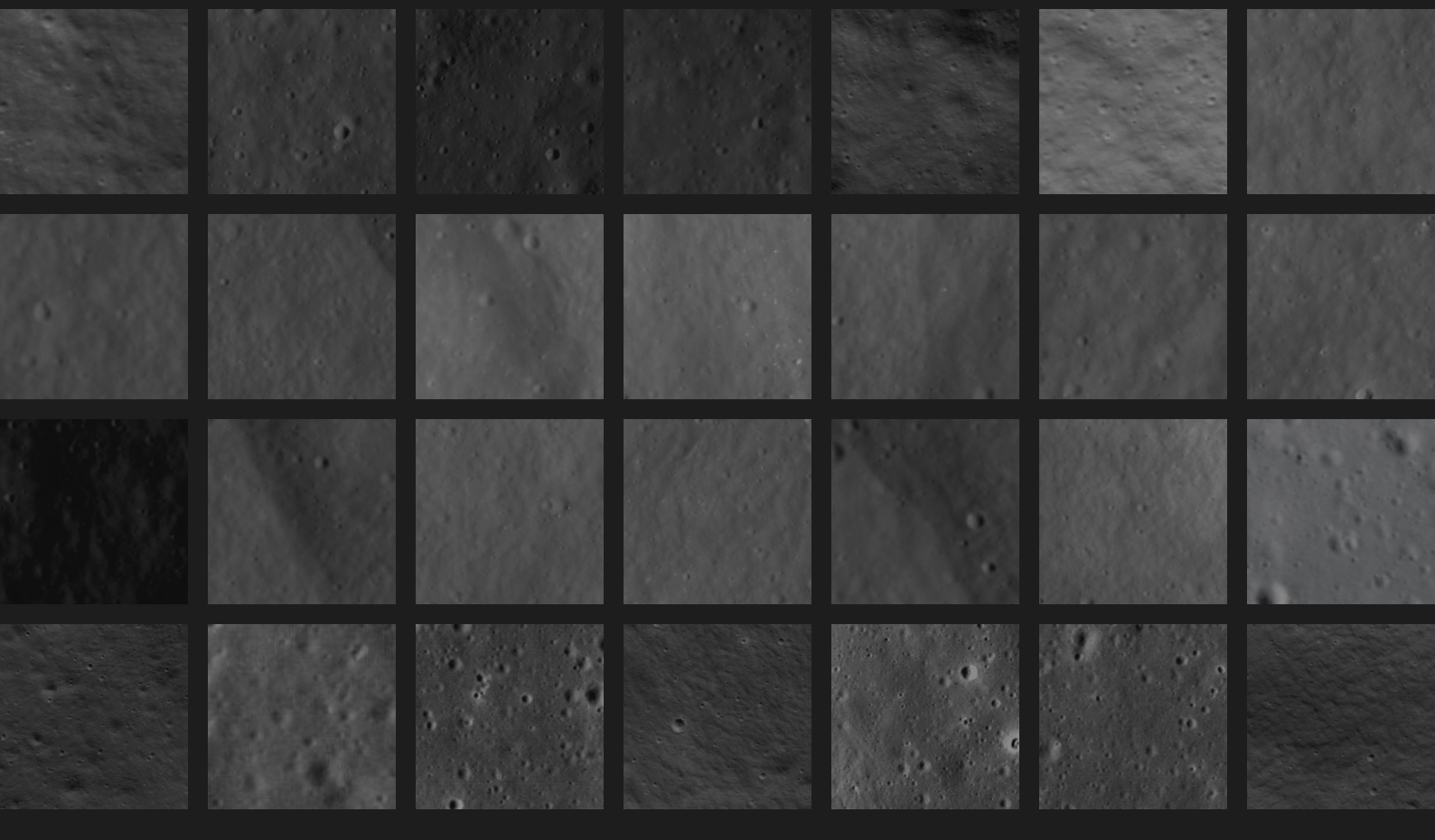
Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer vision - ECCV 2014 - 13th european conference, zurich, switzerland, september 6-12, 2014, proceedings, part I* (pp. 818–833). Retrieved from http://dx.doi.org/10.1007/978-3-319-10590-1_53 doi: 10.1007/978-3-319-10590-1_53

Zeiler, M. D., Krishnan, D., Taylor, G. W., & Fergus, R. (2010). Deconvolutional networks. In *The twenty-third IEEE conference on computer vision and pattern recognition, CVPR 2010, san francisco, ca, usa, 13-18 june 2010* (pp. 2528–2535). Retrieved from http://dx.doi.org/10.1109/CVPR.2010.5539957 doi: 10.1109/CVPR.2010.5539957

Zhang, L., Xia, G., Wu, T., Lin, L., & Tai, X. (2016). Deep learning for remote sensing image understanding. *J. Sensors*, *2016*, 7954154:1–7954154:2. Retrieved from http://dx.doi.org/10.1155/2016/7954154 doi: 10.1155/2016/7954154

Zhang, L., Zhang, L., & Du, B. (2016, June). Deep learning for remote sensing data: A technical tutorial on the state of the art. *IEEE Geoscience and Remote Sensing Magazine*, *4*(2), 22-40. doi: 10.1109/MGRS.2016.2540798

Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2016). Pyramid scene parsing network. *CoRR*, *abs/1612.01105*. Retrieved from http://arxiv.org/abs/1612.01105



In this master thesis, we propose a novel approach to detect lunar craters using new deep learning advances. The architecture of the model employed is a fully convolutional neural network that uses the freely available remotly sensed data from the «Lunar Reconnaissance Orbiter» space probe.

In this brief, we discuss the methodology, the choices made and the evaluation of the model with different visual and quantitative results in addition to the source code.