Training an Artificial Intelligence system to detect and classify boulders on images of planetary surfaces and asteroids

Yaseen Lahmami
AC40001 Honours Project
BSc (Hons) Computing Science
University of Dundee, 2021
Supervisor: Dr Ian Martin

May 2021

Abstract

Rapid advancements in computer vision [5] and deep learning has given birth to new ways we can use the technology to enhance the overall user experience. Artificial lunar surface models are used by researchers and engineers to help simulate space craft landing [1] and to collect valuable information to help further developments. Using computer vision to detect boulders can help save the user time when creating artificial lunar surfaces from real images, as the user would have to place each boulder individually by hand. The data used to train the deep learning model are images taken from simulated terrains in PANGU (Planet And Asteroid Natural Scene Generation Utility) [2]. This project is also intended to prove that a deep learning model can perform well on real lunar surfaces taken by the LRO (Lunar Reconnaissance Orbiter) [3] being that it was trained otherwise. Overall the results were very good, the size and the positions of most boulders were predicted correctly and the speed at which the models were created increased significantly. Although there were a few anomalies, the user is able to adjust the parameters of each boulder, and add or remove boulders where the network has failed to generalise.

1 Introduction

The Lunar Reconnaissance Orbiter (LRO) [3] is a space probe launched by NASA with an objective to collect data and take images of specific regions on the moon. One use case of the LRO is to locate regions where boulders are present. Boulders on planetary surfaces are rock fragments that have most likely originated from asteroid impacts. These rock fragments are scattered across the surface of many celestial bodies in the solar system, and they are huge obstacles when it comes to landing space craft.

For this reason, the importance of landing space craft on a relatively flat surface becomes a priority. Accurate simulations are necessary to test landing sites as we can detect faults before deploying the real object in outer space [1]. Creating these simulations takes time; The user will first have to generate a 3D surface by defining the size of the terrain, overlaying an image to trace each individual boulder and crater - defining its exact diameter and positions and finally setting the parameters to run the test flight. Evidently, this takes time and effort, improving the experience of one of these steps makes a huge difference on the time saved. The goal of this project is to use deep learning to detect and identify boulders on planetary surfaces, output the size and positions and generate a 3D model based on the original image, saving the user time and effort in creating accurate models of the lunar surface.

To generate these models I will be using PANGU (Planet And Asteroid Natural Scene Generation Utility) [2] which is developed by STAR-Dundee. This tool is able to generate synthetic terrains of different planetary surfaces and flight simulations of space craft (*Figure 1*). I will be using this tool to create training images and use real life lunar images produced by the LRO [3] as test cases. Furthermore, this project is also intended to prove that a deep learning model can be trained using images taken from PANGU and tested using images taken from the LRO with acceptable accuracy.

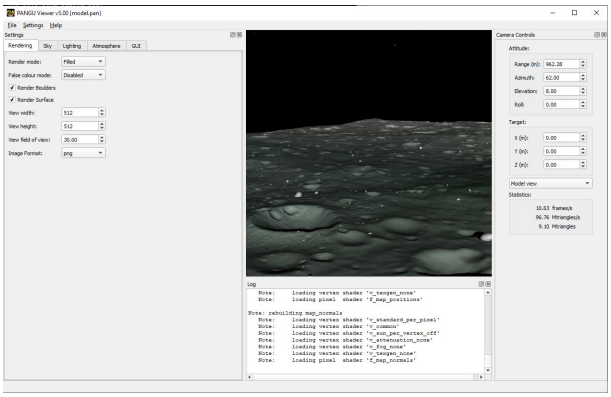


Figure 1: PANGU viewer with a lunar surface being displayed. On the right is a dashboard containing the camera angles and positions and on the left are parameters which control the rendering, sky, lighting, atmosphere and the GUI.

1.1 Project Scope

This project is a combination of both research and software development. Because of this, the time was split evenly between exploring computer vision [5], deep learning and neural networks and full stack development. The project is divided into 4 different steps:

- **Background research**: Studying the basics of neural networks, computer vision pipeline, deep learning and investigating different deep learning architectures.
- **Data collection**: Collecting training, testing and validation images and applying image prepossessing techniques to label and increase efficiency.
- **Training and evaluation**: Building the network, train and tweak hyper-parameters
- **Linking a user interface**: Creating an MVP (Minimal Viable Product) by giving the user an easy interface to the boulder detector

2 Background

2.1 Motivation

Computer vision [5] is a constantly evolving field from simple classification to real time object detection. The main use cases are increasing accuracy in applications such as in medical imagery and automation in self driving cars. Given the advantages of computer vision, this project would be a very good application of computer vision as we can automate the creation of boulders in artificial lunar surface models.

PANGU [2], is a utility tool used to model various planetary objects and surfaces to carry out simulations. The general flow of creating these 3D artificial planetary models is by utilising the PANGU GUI (*Figure 2*). The PANGU user interface is able to generate a flat terrain based on different procedural algorithms such as perlin noise. Depending on the surface the user is simulating, whether that may be a martian surface, asteroid surface or a lunar surface the user is able to add craters and boulders on various regions on the model. This is easily done by using a separate editor, overlaying the real image over an axis which spans the size of the terrain and simply drag over the area at which the target is present. The target can either be a boulder or a crater depending on the mode selected. If the user made a mistake on the position or size of the target, the user can easily adjust the parameters. Once all the targets are highlighted PANGU generates a file which contains a list of positions, sizes and elevation of the target. Depending on the target, the generated list is either called boulder_list.txt or crater_list.txt. PANGU will use these files, and any other defined parameters to create a scene of boulders and craters.

For this project I will be focusing on automating the process of generating a boulder list file, which will be an input to PANGU[2]. Because there are many scattered boulders with various sizes, highlighting each boulder can be very tedious compared craters. This project will utilise the accuracy and automation that computer vision provides. Automating part of the creation of lunar surface models, the user is able to save time that they would

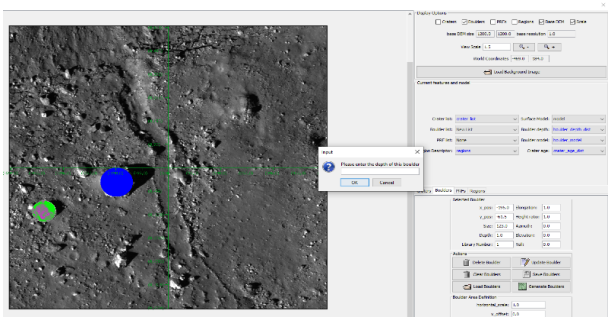


Figure 2: PANGUs [2] feature list editor. Able to highlight boulders and craters over an image. The circle drawn in blue is saved to a boulder_list.txt with the position and area of the circle defined. On the left is a dashboard where the boulder positions and other parameters can be adjusted when needed.

rather be spending carrying out research and and deploying various simulations.

Automating this process would be very beneficial for the user however, the accuracy of the object detection model is more important. It would be near impossible for the model to be fully accurate, but consistently producing acceptable results is necessary to create some value in the tool. Can we train a deep learning model on images taken of generated terrain with acceptable accuracy? And if this is the case, how well can the trained model perform on real life unseen data? The answers to these questions would be very valuable for image collection as the need for simulated images to train an artificial neural network could be favoured over real life images when the data is scarce.

2.2 Research

Although I had some previous experience with machine learning, deep learning and computer vision [5] was an entirely new subject to me. The knowledge of my research are outlined throughout the rest of this section.

Artificial neurons are the basic building blocks of an artificial neural network [4]. As seen in Figure 3 an artificial neuron shares similarities from the neurons in the human brain. Both have an input and an output and both have a way to calculate the signal strength. With an artificial neuron the signal strength is calculated by passing the weighted sum into a function known as an activation function. The activation function outputs the signal strength from the previous signals with the values $0 < O(Z) < 1$ where the weighted sum is $Z = \sum x_n.w_{nk}$ and O being the activation function.

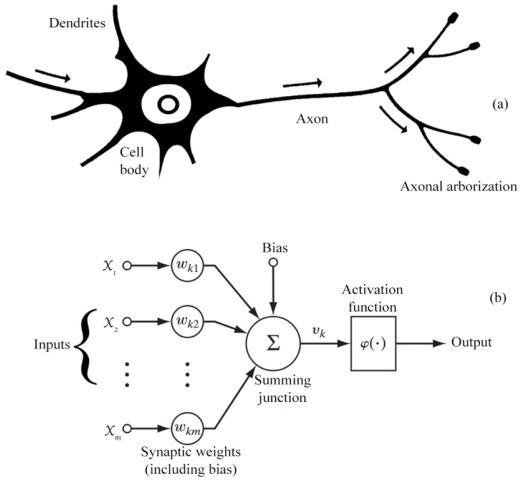


Figure 3: Similarities between an artificial neuron and a neuron in the human brain [32].

Weights are connections between neurons in each layer. Initially, weights are randomised but when trained, every weight is adjusted to fit the input data. *Figure 4* is an example of a fully connected network (FCN) where each neuron is connected to every other neuron. Neurons and their respective weights are organised in layers. Layers are separated into three categories. The input, hidden and output layers. The more layers you have the deeper the network becomes.

Biases are an additional neuron that is attached to each layer to allow for the results to shift freely without being attached to one bias. The weighted sum $Z = \sum x_n.w_{nk}$ forms a linear graph that crosses the origin. Adding a bias to the weighted sum as such $Z = \sum x_n.w_{nk} + b$ will allow for more flexibility, allowing the neuron to access a wider range of values along the y axis as shown in

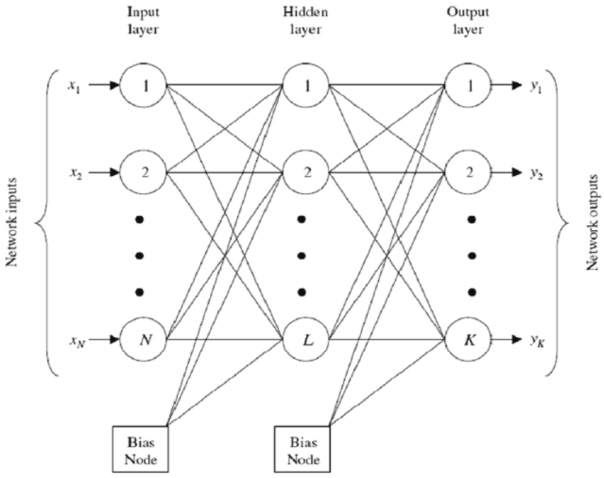


Figure 4: An example of a fully connected network showing the connections between each layer and the flow of data from the input to the output. [33]

Figure 5 .

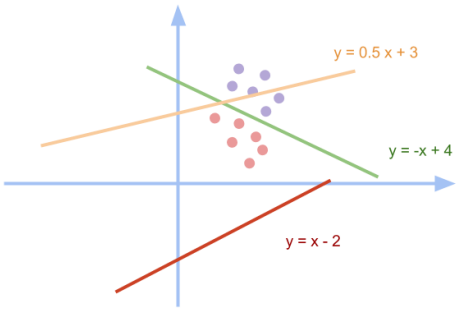


Figure 5: An example showing why biases are effective to prevent over-fitting and under-fitting when classifying two classes of data. [34]

Training a fully connected network [4] consists of two steps. First we **feed forward** the data into the network in the input layer. The weighted sum of each neuron is calculated, the results from the previous layer is fed forward to the hidden layers until the output layer. So far the network has done a linear combination to produce an output. To learn the network uses a process called **back-propagation**. A loss function such as the Mean Squared Error (MSE) outputs a value that indicates the strength of the network against the ground truth. The further away the loss is from zero, the further away the loss is from its minimum. **Gradient descent** is an optimisation algorithm to find the minimum loss against the weights. By finding the minimum loss, the network is able to find the minimum values of the weights. As shown in *Figure 6*, the weights are updated by subtracting the current weight by the change in the weight, which is represented by the learning step. Many learning steps occur over many iterations which in turn creates an epoch.

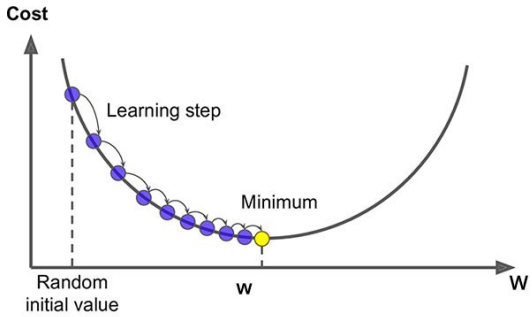


Figure 6: Similarities between an artificial neuron and a neuron in the human brain. [35]

Epoch is a term used to describe a cycle. When the entire data-set has been passed through the network from the input to the output, it's known one epoch. After each epoch the losses are averaged out ready for the next cycle. The more epochs the more the network is able to notice patterns within the data and the closer network can reach its optimum state.

Computer vision [5] is a field in computing where the goal is to enable machines to distinguish between different objects in a scene. One of the main objectives in computer vision is to not only classify the object, but label where the object lies spatially. To achieve this, the computer vision pipeline is a protocol used to structure machine vision problems.

1. *Input data*: The data used as an input. In computer vision the input data is typically images and videos accompanying a csv file containing ground truths.

2. *Preprocessing*: Image prepossessing techniques are used to reduce the image properties only if these properties are not needed to solve the problem. This includes changing the image from color to black and white, reducing the number of channels from three to one. Removing the background to reduce noise. Resizing the images so that they are all the same size, as the input to the neural network is a fixed length. By removing any unnecessary properties, the number of parameters needed for the network are greatly reduced, meaning a decrease in training times, losses and increased efficiency.
3. *Feature extraction*: Images are then sent to a feature extractor. In deep learning the feature extraction is typically done in a **Convolutional Neural Network** (CNN) [6]. Feature extractors extract regions of interest by estimating foreground objects based on the ground truths.
4. *Deep learning model*: This consists of a CNN [6] and a Fully Connected Neural Network (FCN) [4]. The FCN is used to classify the foreground objects found and predict where the objects are located spatially.

Convolutional neural network (CNN) [6] is a network compromised of convolutional layers where the next layer is derived from the previous layer. Unlike a fully connected network, a CNN is not fully connected and it is not layered with artificial neurons, rather sliding windows. The sliding windows in a CNN are known as receptive fields. These receptive fields are matrices of sizes $2x2$, $3x3$ or $5x5$. Receptive fields behave like image filters. Just like weights, these filters (also known as kernels) are randomly initialised and their parameters are adjusted depending on the loss. Filters scan the image and apply the dot product of the pixel values currently being scanned with the filter matrix. This operation produces an augmented version of the image allowing certain features to be extracted, which is shown in *Figure 7* . The window slides until the whole image is scanned. This operation produces an image in the next layer, with reduced dimensions while capturing finer details. The details captured depends on the type of filter. Some filters can capture fine lines while others capture darker regions in the image. Multiple filters are used in each layer to take advantage of the variety. Filters are learnable parameters and are adjusted per epoch with gradient descent.

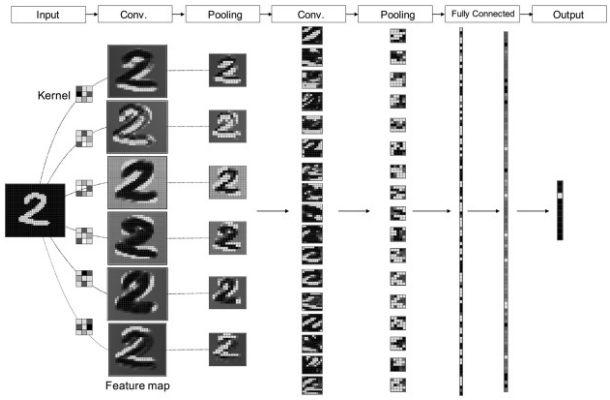


Figure 7: The extracted features layer by layer become almost unrecognisable to the human eye. This shows the evolution of details being extracted so that a fully connected network can classify the features. [36]

CNN architectures [7] typically consist of a Convolutional layer (CONV), activation function and a pooling layer (POOL). This structure is repeated many times - the more layers the finer the details being extracted. The convolutional layer extracts features, an activation function such as ReLU is applied to the convolutional layer to introduce non-linearity and a pooling layer is applied to reduce the dimensions of the filtered image. As we can see, convolutions also help the efficiency of the network by reducing the size of the images without compromising accuracy. Of course too much optimisation can cause overfitting and hardware being a limiting factor, which is why CNN architectures exist. Examples of CNN architectures include ResNet-50 [8], VGG-16 [9] (*Figure 8*) and GoogLeNet [10]. These architectures help improve the speed and maintain accuracy of the network while decreasing the size of the CNN.

Object detection requires more advanced architectures than image classification. The general object detection framework consists of the following steps:

1. *Region proposal*: Algorithms that propose regions of interest which are mainly foreground objects.

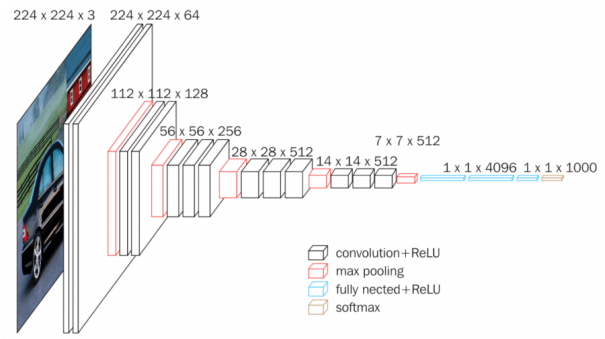


Figure 8: VGG-16 architecture developed by Visual Geometry Group from Oxford. The architecture consists of 16 convolutional layers and 5 pooling layers. The final layers is a FCN with three layers for classification. [37]

2. *Feature extraction and network predictions*: CNN analyses the regions of interest (ROI) and extracts features. The output of a CNN is flattened and passed through to the classification part (FCN) [4]. The FCN outputs the object type and bounding box coordinates.
3. *Non-maximum suppression (NMS) [15]*: The same object may contain multiple bounding boxes. NMS tries to avoid repeated detection.
4. *Evaluation metrics*: Metrics such as FPS (Frames Per Second) and mAP (Mean Average Precision) give better details on speed and accuracy.

Object detection architectures follow this framework. An example of such architecture is R-CNN [11] which stands for Region-based Convolutional Neural Network. There are different variants of R-CNN's which are R-CNN [11], Fast R-CNN [12], Faster R-CNN [13] and Mask R-CNN [14]. Each iteration is an improvement on the previous architectures. Faster R-CNN and Mask R-CNN are the best architectures in the R-CNN family based no their speed and accuracy. Depending on the use case, faster R-CNN and Mask R-CNN is used for object detection, but where the difference lies is Mask R-CNN can also perform image segmentation. For this paper the focus will be on faster R-CNN's.

Faster R-CNN [13] is the third iteration of the R-CNN family. The feature extractor is typically a ResNet-50 [8] CNN architecture followed by a Region Proposal Network (RPN) and a fully connected network for classification and predictions as shown in *Figure 9* . RPN [13] is a FCNN (Fully Convolutional Neural Network) that consists of two layers. The first layer acts as an input, the second layer is two convolutional neural networks working in parallel. The first convolutional network in the second layer acts as a classification network which is used to predict the objectness score. The objectness score is a prediction on whether the region contains a foreground or background object. The second convolutional network in the second layer acts as a region regressor which is used to predict the (x,y,w,h) of the bounding box.

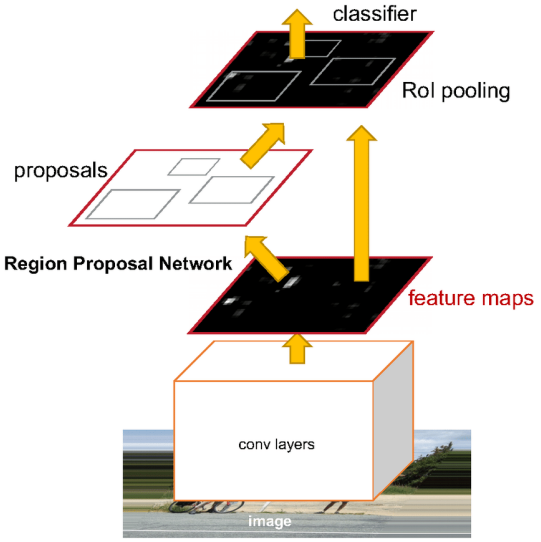


Figure 9: Simple overview of a Faster RCNN architecture. Part of the feature extraction goes to the RPN and the rest to the FC layers for classification. [38]

Anchors [13] are a collection of points that are evenly placed on the original input image. These points are used as references to draw reference boxes or anchor boxes. Anchor boxes stem from each anchor point, many can be drawn at different aspect ratios. Anchor boxes are used to compare to the ground truth bounding boxes. For each anchor there are limited possible anchor boxes as each anchor will have n aspect ratios and n scales (*Figure 10*).

CNN filters will slide across the feature map and calculate the objectness score and the bounding box coordinates. If the objectness score is > 0.7 then the region is classified as a foreground object and labelled positive. If the anchor box is < 0.3 then the region is classified as a background object and labelled negative. The negative regions are discarded and the output of the RPN moves to the fully connected layers for further classification [4].

Figure 10: An example of the number of anchor boxes produced by the RPN, which is why some are discarded if the object is a background object. [39]

3 Specification

The overall vision for this project is to create a deep learning tool to detect boulders on the lunar surface and output their size and positions. The user would need to run a prediction script and input certain parameters such as the image path, image size and the PANGU surface model size. Once the prediction has finished, a boulder list is generated containing every detected boulder with parameters such as size and positions. The user would then use the file to visualise the boulders using PANGU.

Using this tool as a command line application can seem quite cumbersome hence as an additional extra, the user would be able to interact with a front end instead. This gives an enhanced user experience as there is easy access to interact with the tool. More details of the full system is discussed in section 4 Design.

Main requirements gathered:

1. The system should be reasonably accurate, allowing some deviation in the detection.
2. The system shall generate a boulder_list.txt file .
3. The boulder_list.txt file must contain the position, siz, elevation and library number for each detected boulder.
4. The boulder_list.txt file shall be compatible with PANGU [2] to produce a surface model containing the boulders defined.

Considering just the command line application, the object detection is projected to complete by the 22nd of March, giving the remaining time for any extras, the report and testing. I have planned out specific project goals in a Gantts-chart which can be found in the appendices.

[clear image of timeline]

4 Design

The project design is separated into two parts, the object detection system and a front end for the user to interact with the trained model. Adding a front end became additional material for this project as the tool will not be user friendly when interacting with a command line. For the most part, this section will largely be devoted to covering the object detection system as it is critical for the overall system to function.

4.1 Object detection system

Object detection is an important field within computer vision hence it is paramount that the right methods are used for problem itself. Object detection methods include bounding box object detection and image segmentation.

Image segmentation [16] involves predicting masks which are highlighted regions of interest. The output is the area of the highlighted region and its position (*Figure 11*). Image segmentation would be very useful for

the design of this project as one of the use cases of this method is spotting regions in geospacial images such as the lunar surface. My reasoning for considering image segmentation was inspired by a similar paper that implemented image segmentation on craters of the lunar surface. Titled "Lunar Crater Identification via Deep Learning" [17] the team utilised the UNET [18] architecture which is an FCNN and achieved 83% accuracy with craters with a radius of < 15 pixels. Although this seemed compelling, using image segmentation for this problem would be unnecessary. Firstly, bounding box object detection would be able to locate the boulders, surrounding each detected boulder with a box like the example shown in *Figure 11* . By doing some simple geometry, the system will be able to find the position and size of the boulder, from the midpoint of the bounding box. Lastly, automatically labelling the data for image segmentation problems is more difficult, as implementing complex image processing techniques would be unnecessary as there is already a more feasible method with bounding box object detection. For the reasons outlined, bounding box object detection seemed achievable for the scope of this project.

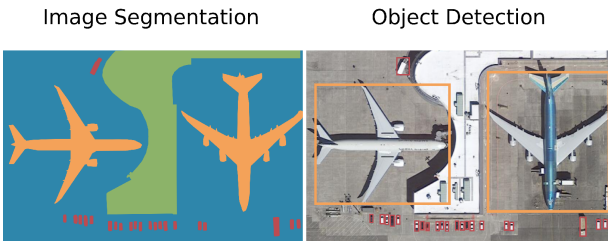


Figure 11: The image shows detection on geospacial data of an airport. The left is an example of image segmentation and the right is an example of object detection. [40]

The object detection system will follow the computer vision pipeline hence, it will be split into four parts. Part one will be data collection. PANGU [2] will be used to collect images of boulders from generated models as there is not enough data online to support training. The images are sent for prepossessing, where the data is labelled, storing the ground truths in a csv file and augmenting the images. Finally once the data is ready the system is split into training and prediction modules. First the network is trained, validated and saved in a file with the values of all the parameters after training. This file is loaded into the prediction module, that takes input of a lunar surface and outputs a boulder list for PANGU to interpret. Refer to *Figure 12* for an overview of the project architecture.

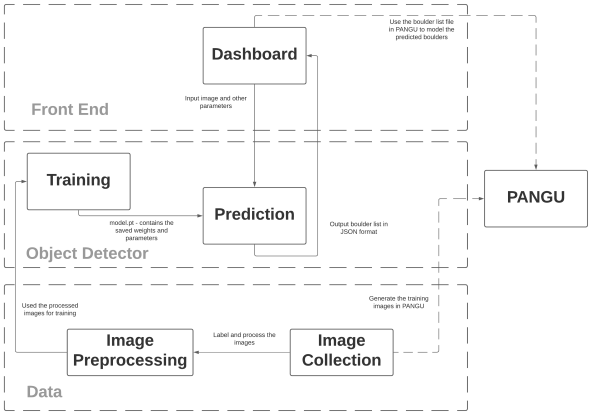


Figure 12: The full system architecture which is organised into three layers. First the data layer, then the object detection layer (which will act as a backend) and a front end. A detailed diagram is available in appendix x

The CNN architecture of choice will be a Faster R-CNN [13]. The architecture is part of the R-CNN [11] family and it is especially good for images that contain smaller objects such as boulders. Architectures such as YOLO [19] (You Only Look Once) and SSD [20] (Single Shot Detection) are considered to be more efficient architectures as their FPS is considerably faster than R-CNNs. However, because images are the only input, it would be unnecessary to choose an architecture that performs better on FPS rather than accuracy. The Faster R-CNN will use a pretrained feature extractor CNN with the ResNet-50 [8] architecture. Although this feature extractor contains more layers, the CNN architecture is designed to create smaller feature maps which is necessary to detect objects such as boulders. The rest of the network will contain an RPN for region proposals and a classifier in the final layers. The output will be the bounding box coordinates in an (x, y, w, h) tuple for each boulder. Finally the output bounding box pixel coordinates are converted

to world coordinates to scale with PANGU [2].

Adding a front end and creating an end to end deep learning system was not the main objectives of this project. However an additional extra such as this would give the project a finished product feel. The object detection system will act as a backend and the front end will be a simple interface to run the prediction model. For testing purposes the backend will run on localhost and will listen to 'GET' requests through RESTful API [21]. A 'GET' request will run the prediction model locally on the machine and output a predicted boulder list, which is used as an input to PANGU.

5 Implementation and Testing

5.1 Tools and frameworks

Python [22] - "is an interpreted, high-level and general-purpose programming language". [22] The language is capable of functional and object oriented approaches. Predominantly, the language is used for machine learning, as the language is easy to use and efficient enough for complex algorithms. Other use cases include automation, data preparation and web development.

Pytorch [23] - "An open source machine learning framework that accelerates the path from research prototyping to production deployment." [23]. Fundamentally, Pytorch allows developers to create neural networks and deploy them in applications. Pytorch is capable of classical machine learning, to solving deep learning problems such as computer vision. **Torchvision** [24] is a library built with pytorch with the goal of providing industry standard pretrained models and image preprocessing functions for computer vision. By using pretrained models (known as transfer learning), developers are able to build prototypes quickly and at scale which is the reasons for using Pytorch over other libraries such as **Tensorflow** [25]. In addition, Pytorch is "Python friendly" meaning that it follows Pythonic practices, allowing for better project structure and ease of use.

Notable libraries or frameworks used throughout this project include:

- **OpenCV** [26]: "OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library." [26] Also useful for labelling bounding boxes and image processing.
- **Pandas** [27]: "Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool." [27] Allows the creation of dataframes which are helpful in generating csv files for ground truths.
- **Matplotlib** [28]: "Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python." [28] Useful for evaluating the data visually, so that conclusions could be made on whether the network is overfitting or underfitting.
- **Flutter** [29]: "Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase." [29] Utilises a language known as Dart [30], which enforces object oriented principles to build scalable and reusable user interfaces. The SDK is used to build the desktop application so that the user can easily interface with the trained network.

5.2 Implementation

To start, data collection and preparation was a priority. Images needed to contain boulders of planetary surfaces, and for this project the target was the lunar surface. Using the lunar surface means that the artificial network could be stress tested as there are many factors to consider such as craters and shadows.

Mainly, the difficulties were finding and labelling the data. Real life images of the lunar surface were hard to find. Even though there were plenty of images taken by the LRO [3], the images did not fit the specification for training. First the boulders needed to be visible. The camera distance would need to be as close to the lunar

surface as possible and the surface should be illuminated by sunlight. Finally the image taken needs to show the surface at a 90 degree angle. Different angles would cause complications when calculating the size and positions of the boulders. With that being said, only a handful of images were found, which will be used as test cases against the training and validation data.

Due to needing thousands of images for training, the only way was to generate them using PANGU [2]. PANGU enables users to generate Digital Elevation Models (DEM) for simulations or for collecting data. Using PANGUs camera, the software was able represent the camera as an LRO [3] and capture batches of images at different locations of the generated terrain. The camera was angled at 90 degrees and at a height of 350 meters which was enough to fit the specification for training (*Figure 13*).

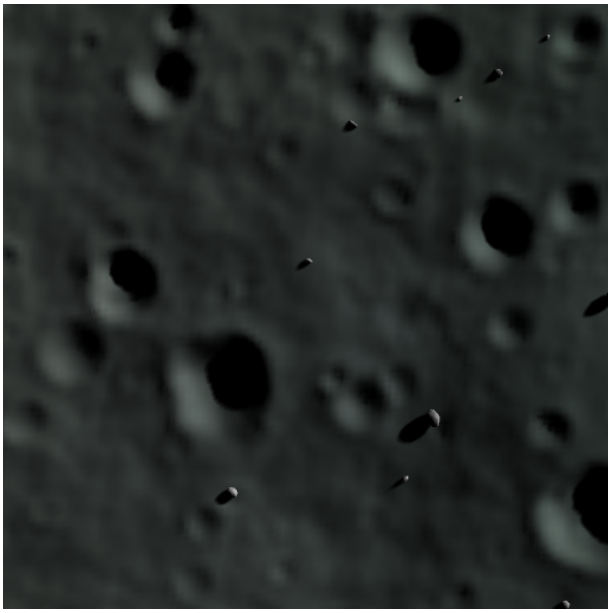


Figure 13: Image taken by the camera in PANGU [2]. The height from the surface to the camera is 350 metres with the camera angled at 90 degrees.

In total around 3,200 images were collected for training and validation and 11 images for testing. For each image, the network needs to compare the ground truths to the prediction hence the data needs to be labelled. The task was not trivial as the network would need the bounding box coordinates of each boulder and a binary label for classification. Labelling the images required finding the point in pixel coordinates from PANGUs [2] coordinate system. This was one of the main hurdles to overcome as the full system specifications would not be achieved if the data is not fully labelled.

Using normalisation techniques was the original solution however that proved not to be as accurate in locating the boulders. Finding ways to label the data, my advisor lead me to a solution to use PANGUs [2] camera. Due to the camera being at a 90 degree angle, trigonometry was used to convert the boulders into pixel coordinates. PANGU has a camera with an FOV (Field Of View) angle and by default the cameras FOV is 30 degrees. From the camera to the surface creates a height in meters. With this, a ratio is returned which gave the number of pixels per meter in PANGU. Multiplying the ratio and adding on the image center point returned the boulder point in pixel coordinates.

Bounding boxes are a tuple in the form $(min_x, min_y, max_x, max_y)$. Encapsulating the boulders accurately with bounding boxes was difficult with just a single pixel point which lead to another approach in labelling the data. The proposed idea was to remove the background and isolate the boulders only (*Figure 14*). By using contours which are outlines around shapes, OpenCV [26] was able to draw bounding boxes around these contours and return the exact bounding box coordinates for each boulder. This approach was much more accurate in labelling the boulders as the width and height of the boulders was also captured. However, erasing the background may cause some of the boulders to be removed and some artifacts such as shadows and craters to be outlined. With that being said, this approach was preferred than finding the ratio. The previous method will not be discarded as the ratio is needed to produce the boulder list output.

The bounding box coordinates are paired with the image file name using a CSV file shown in *Figure 15* . The CSV file is loaded into the network by using a data loader. Data loaders are a necessary component in Pytorch [23]

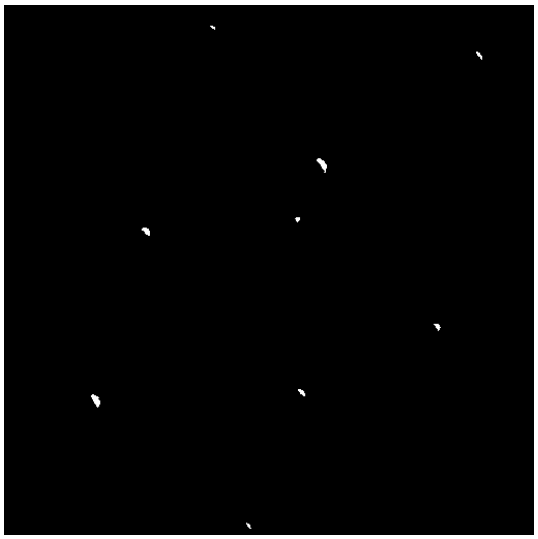


Figure 14: Background removed which isolated the boulders (in white) so that the image can be labelled automatically using contours.

	B	C	D
1	image_id	bbox	
2	bframe_00000.png	[140, 510, 1, 1]	
3	bframe_00000.png	[142, 479, 23, 33]	
4	bframe_00000.png	[198, 474, 43, 33]	
5	bframe_00000.png	[0, 457, 41, 27]	
6	bframe_00000.png	[83, 448, 2, 1]	
7	bframe_00000.png	[72, 437, 1, 1]	
8	bframe_00000.png	[71, 435, 1, 1]	
9	bframe_00000.png	[353, 410, 24, 27]	
10	bframe_00000.png	[68, 403, 53, 49]	
11	bframe_00000.png	[422, 396, 66, 58]	
12	bframe_00000.png	[487, 377, 17, 25]	
13	bframe_00000.png	[472, 357, 4, 3]	
14	bframe_00000.png	[459, 330, 45, 49]	
15	bframe_00000.png	[247, 318, 43, 43]	
16	bframe_00000.png	[507, 315, 5, 7]	
17	bframe_00000.png	[74, 295, 21, 22]	
18	bframe_00000.png	[257, 288, 25, 28]	
19	bframe_00000.png	[424, 269, 18, 27]	
20	bframe_00000.png	[133, 266, 1, 1]	
21	bframe_00000.png	[404, 249, 21, 26]	
22	bframe_00000.png	[460, 220, 18, 16]	
23	bframe_00000.png	[203, 219, 19, 30]	

Figure 15: The CSV file that is loaded into the data loader. The file contains the image name paired with the bounding box. Multiple bounding boxes are labelled for each image hence there are multiple occurrences of the same image file name.

to map data to its ground truths. For computer vision the data are images and each image is mapped to its labels. Since data loaders are iterable, images with their labels are indexed. The images within the data loader are loaded with their pixel values into tensors of three channels of RGB. Once loaded, the data is randomly sampled to reduce linearity and improve training results based on the randomisation of the data. The images are then prepossessed to grey-scale reducing the number of channels from three to one. Further preprocessing includes normalising the tensors between 0 and 1 from 0 to 255. This allows for a smaller range of values which increases the speed of gradient descent, as there are a smaller range of values to work with. Finally the data is split into training and validation sets and loaded into batches. The split is 80% training and 20% validation. Validation will be used along side training to determine whether the network is overfitting, underfitting or generalising with the data as it is important to know how to tweak the network for optimum results.

The deep neural network was built using Pytorch [23] Torchvision [24] library. Torchvision has pre-built Faster-RCNNs ready for training. The CNN model contained ResNet-50 [8] CNN feature extractor with an RPN for region proposals and fully connected layers for the classification. The network was pretrained against a large dataset of images so that the weights are optimised before training. This is known as transfer learning and it is a useful technique to save time and resources when training. The deep learning model can be adjusted to fit the data set. The adjustments were the input and the number of classes. The input can vary in size depending on the image dimensions. The dimensions are 512x512 for the training and validation sets. The network is adjusted for 2 classes, which are labelled 0 for ”no boulder” and 1 for ”boulder”. Faster-RCNN [13] is an object that accepts an input which is the data in the data loader. Each instance in the data loader is iterated and is used as input into the Faster-RCNN model. Model outputs the losses which is used to update the gradients in gradient descent. Finally the losses are averaged out after each epoch and the cycle continues until the number of epochs are satisfied. While training, the network is also validating against unseen data which is important to understand if the network is generalising.

Once trained the model is evaluated. By setting the model to evaluation mode, the network is able to output the predicted bounding boxes and the classes. Further evaluation includes validation loss which will be discussed in section 6 Evaluation. The next step is to save the trained Faster-RCNN model [13] in a .pt file. Parameters such as weights, biases, nodes and other information are saved in this file. This allows the model to be reused without training which saves time and improves efficiency. Next, the saved model is loaded again into a Faster-RCNN this time not pretrained. At this stage, the model is ready to predict the boulders from the 11 test images. The output is a dictionary containing a list of detected bounding boxes and the predicted classes.

The output bounding box coordinates is in the form $(min_x, min_y, max_x, max_y)$ and it is written to a .txt file known as a boulder list. A boulder list defines attributes for each boulder such as the boulder position, size, elevation and other attributes used by PANGU [2] to place objects in the form of boulders. Before the bounding boxes are written into this file, the boxes need to be converted into PANGUs coordinate system. The conversion is as simple as finding the midpoint of the bounding boxes if the DEM size is the same size as the image. If not then the meters per pixel calculation mentioned previously is used instead. In most cases the image will be the same size as the DEM (Digital Elevation Model) hence the calculation is one to one. For the size of the boulders, the length of the bounding box will act as the diameter as PANGU uses this parameter to calculate the area of boulders. Finally the elevation is normalised in the range $0 < x < 0.5$ depending on the boulder size.

Adding a front end was not part of the original specification, however creating an easy interface for the user would only help better the user experience. The front end was implemented using Flutter [29] which uses a language known as Dart [30]. For the object detection system to connect to the front end, a server running on local host was implemented using a library called flask [31] which utilises the RESTful API [21]. Parameters such as the image file location, image size, surface model size and the boulder list text file output location are sent to the server using a GET request. The request calls a function which runs the object detection with the parameters defined. The output from the GET request is a JSON object which contains the boulder list and the output image location. Finally the output image and boulder list gets displayed back to the user after a successful request.

5.3 Testing & Debugging

Apart from labelling the data, one of the main difficulties was representing the boulders accurately in PANGUs [2] feature list editor which is shown in *Figure 16*. There were troubles accurately representing the size and sometimes the positions of each boulder. The problem was a scaling issue where larger boulders were over scaled compared to the smaller ones. Previously, the area was calculated directly by representing the bounding box with an inner circle. The result massively scaled the boulders as PANGU as shown in *Figure 16*. The fix was to use length of the bounding box which acted as a diameter in which PANGU uses to calculate the area of the boulders.

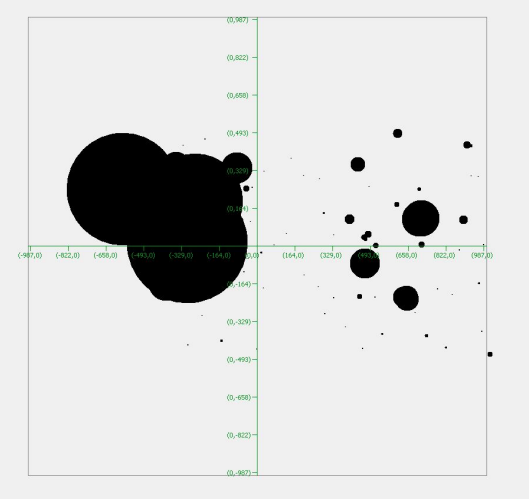


Figure 16: PANGU feature list editor displaying the oversized boulders as a result of a miscalculation. *Figure 18 (b)* shows an accurate version where all the boulders are scaled and positioned correctly.

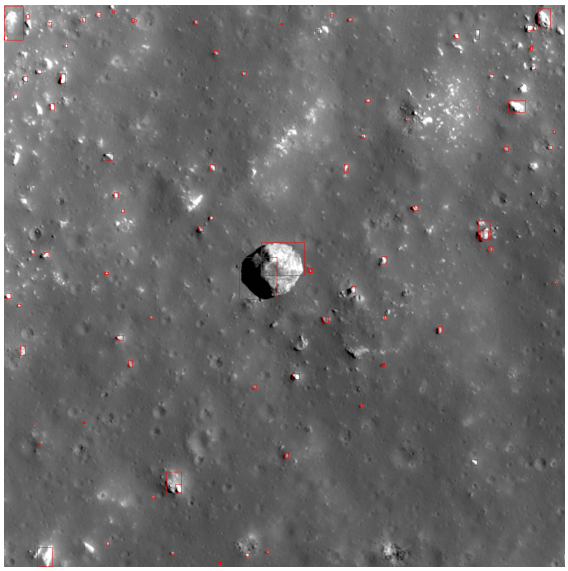
Training and tuning the network was also quite difficult.

Training requires an efficient and fast GPU at which that was not available on the local machines. For this to work, an external GPU was used that was available for a limited time only. Using the external GPUs decreased the training time significantly but it became inconvenient when it came to tuning hyper-parameters and evaluating results.

At the beginning, the network performed well on a small set of images of around 700. The loss after 10 epochs was oscillating between 0.3 and 0.4 with a starting loss of around 0.8. Due to the losses being close to 0, the results were not bad however, increasing the epochs only improved the results by a small amount indicating that the model may be overfitting. More images were needed for training, after gathering a sample size of 1600 images the network performed approximately the same, however the losses decreased gradually with less oscillation.

When testing the networks accuracy, it became clear that smaller boulders were captured more than its larger counterparts. This was due to the dataset containing predominantly smaller boulders. To solve this, the dataset was adjusted and increased to contain larger boulders which was all generated in PANGU [2]. Expanding the variation of the data set proved to be effective as shown in *Figure 17*. The dataset increased to approximately 3200 images with a variation of larger and smaller boulders. After 35 epochs the average loss decreased to 0.27 which showed that overall the network performed better with an increased sample size and increased variation of the data.

[a]



[b]

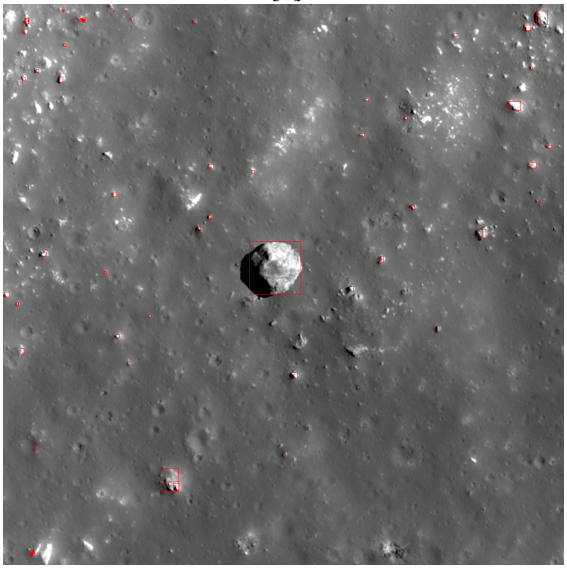


Figure 17: Output from the network using one of the test images taken by the LRO [3] with the code M175212953RE.(a) First output without adjusting the data set to account for larger boulders. The center boulder contains 2 bounding boxes showing that the network struggled to perform on larger boulders. (b) After introducing more variance in the data, the center boulder has been fully detected with the bounding box encapsulating the entire object

6 Evaluation

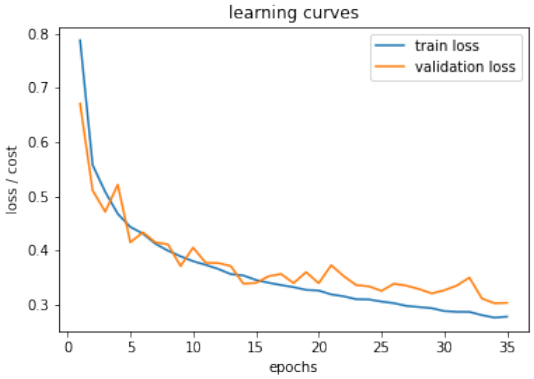


Figure 18: Training loss and validation loss plotted for version 3 of the network. The x axis represents the epoch and the y axis represents the loss. The training loss and the validation loss are descending downwards together which shows that the network is generalising on unseen data. The network is neither overfitting or underfitting.

For the final results, the network was run with the following configuration:

Learning rate: 0.01
Batch size: 4
Epochs: 35

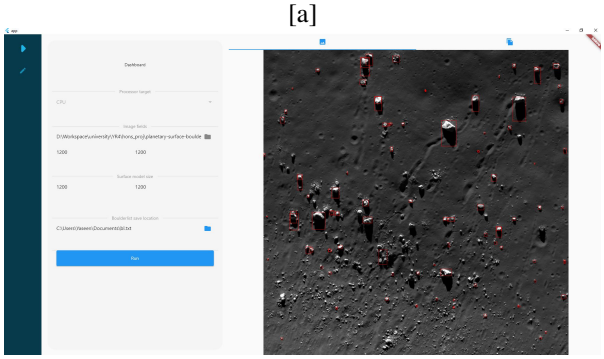
After training with this following configuration the network achieved a loss of 0.27. The table below shows the performance of the network over three iterations.

Network Iterations				
Version	Epochs	Dataset size	Starting loss	Final loss
1	10	400	0.68	0.30
2	25	1600	0.86	0.37
3	35	3206	0.77	0.27

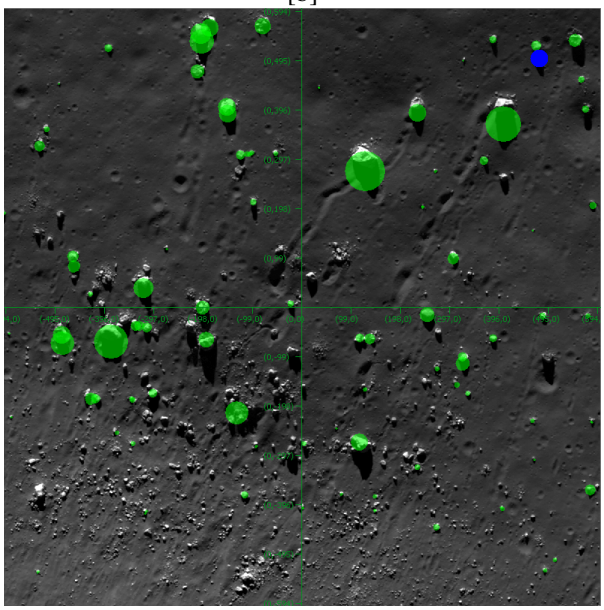
This shows that more epochs along with an increased dataset size improves the loss of the network. The closer the loss is to 0 the better the network is generalising on the trained data. To visually evaluate the performance of the network against unseen data *Figure 18* shows the training loss and the validation loss plotted against each other.

Overall, the final specification has been satisfied (Refer to Section 3 Specification). The system is able to detect and classify boulders on the lunar surface *Figure 18 (a)*, output a boulder_list.txt file *Figure 18 (b)* that is compatible with PANGU [2] to generate a surface model with the defined boulders *Figure 18 (c)*. Finally as an additional extra, the user is able to interface with the system using a GUI, allowing the user to produce boulder list files at ease without using the command line.

[a]



[b]



[c]

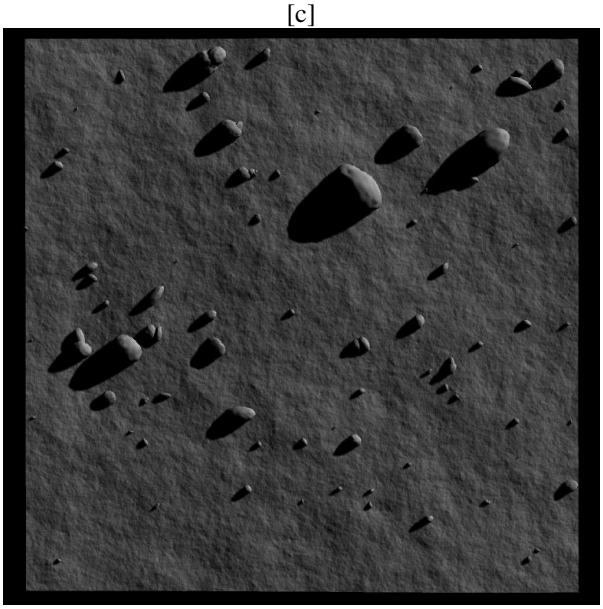


Figure 18: Evidence of the final specification. This shows the output from the a test image taken by the LRO [3] labelled M185903952RE. Detailed images are available in the appendix (a) Caption of the front end. On the left is a dashboard for the user to input specific parameters that the object detection system requires. Once the user runs the program an output image of the detection is revealed along with the boulder list. (b) Boulder list visualised - the image shows a good distribution of large and small boulders that are detected and positioned. Although not fully accurate, the user is able to modify the parameters of each boulder and add new boulders where the model failed to generalise. (c) A surface model generated in PANGU using boulders predicted by the object detection system.

6.1 Usability

Usability should be evaluated with a description of the user-centred design methods employed to produce a usable product, including rapid prototyping, usability methods, results and re-designs as appropriate.

7 Description of the Final Product

The final product is a full stack deep learning system with the backend acting as the boulder detector and the front being an easy interface for the user define parameters. The system allows the user to easily input a single image that contains boulders and any other parameters that help generate the boulder list. The output is a boulder list file that contains the diameter and positions of each boulder that is converted to PANGU coordinate system. In addition, an image is displayed showing bounding boxes around the predicted regions to give the user an idea of the accuracy of the system. Finally, by using a visual tool in PANGU [2], the user is able to edit the predicted boulder list simply by editing existing points or creating new points where the model has failed to generalise.

8 Conclusion

Summarise the main points and ensure that you have described the final product. Include a critical appraisal of the project indicating the rationale for design and implementation decisions, lessons learnt during the course of the project and an evaluation (with the benefit of hindsight) of the final product and the process of its production (including a review of the plan and any deviations from it). Make recommendations for future work.

In conclusion, although the object detection system could always be improved, all of the system specifications have been satisfied. To build on the usability of the product, the addition of a front end gave the user easy access to the object detection system and gave the project a finished feel. In addition, the performance of the network was better than expected. Using PANGU [2] to the generate training images on artificial terrain, and using the network to test against real lunar surface images, proved that a deep learning system can generalise reasonably well, being that it was trained in different conditions. The object detection system was built using Python along with the Pytorch [23] library. Using the Faster R-CNN architecture, meant that the system is

able to detect boulders of a range of sizes. Furthermore, the input data are not video frames, which means that choosing an algorithm based on FPS (Frames Per Second) such as YOLO [19] would be unnecessary. The system output is a list of bounding box coordinates which is converted to a boulder list text file, which is formatted to be compatible with PANGU. The output file is then used to visually represent the detected boulders on a 3D surface model in PANGU. Finally the speed at which the surface models are generated in PANGU has significantly increased. Although not fully accurate, the user is able to modify the placements, add and remove boulders when appropriate using the feature list editor in PANGU.

Reflecting back, preparing the data before starting the project would have saved time that could have been used to further adjust the network. My inexperience in deep learning and computer vision should have been the trigger to practice some of the frameworks before hand rather than focusing most of the time on research. With that being said, the confidence gained from this project will give me a better understanding on how to approach deep learning problems in the future. For further improvements, implementing a learning rate scheduler would help further gradient descent as the learning rate is set algorithmically based on the loss. On occasions, the network would fail to generalise of images containing boulders in clusters. Further gathering a set of images that contained these clusters would help the network perform better on these edge cases. Finally, the addition of other evaluation metrics such as precision, recall and mean average precision (mAP) would give more insight into the network performance, to make better adjustments to the network if necessary.

Acknowledgements

The author would like to thank her wonderful supervisor, and her mum, dad, dog, cat and budgie for all their support.

References

[1] Ian Martin (Lead / Corresponding author), Stephen (Steve) Parkes, Martin Dunstan, Nicholas (Nick) Rowell, Sohrab Salehi (Contributing member), Daniele Gherardi. *"Validation Of Mission Critical Vision-Based Navigation Systems For Planetary Landers, Rovers And In-Orbit Rendezvous"* - 6^{th}*June*2014

[2] PANGU (Planet And Asteroid Natural Scene Generation Utility) website: **`https://pangu.software/`**

[3] LRO (Lunar Reconnaissance Orbiter) overview: **`https://www.nasa.gov/mission_pages/LRO/overview/index.html`**

[4] Enzo Grossi and Massimo Buscema. *Introduction to artificial neural networks* - January 2008

[5] Victor Wiley and Thomas Lucas. *Computer Vision and Image Processing: A Paper Review* - February 2018

[6] Keiron Teilo O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks* - November 2015

[7] Asifullah Khan, Anabia Sohail, Umme Zahoora and Aqsa Saeed Qureshi *A Survey of the Recent Architectures of Deep Convolutional Neural Networks* - 17 January 2019

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition* - 10^{th}*December*2015

[9] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition* - 4^{th}*September*2014

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich. *Going Deeper with Convolutions* - 17^{th}*September*2014

[11] Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik *Rich feature hierarchies for accurate object detection and semantic segmentation* - $11^{th}November2013$

[12] Ross Girshick *Fast R-CNN* - $30^{th}April2015$

[13] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* - $4^{th}June2015$

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár and Ross Girshick *Mask R-CNN* - $20^{th}March2017$

[15] Jan Hosang, Rodrigo Benenson and Bernt Schiele *Learning non-maximum suppression* - $8^{th}May2017$

[16] Song Yuheng and Yan Hao *Image Segmentation Algorithms Overview* - $7^{th}July2017$

[17] Ari Silburt, Mohamad Ali-Dib, Chenchong Zhu, Alan Jackson, Diana Valencia, Yevgeni Kissin, Daniel Tamayo and Kristen Menou *Lunar Crater Identification via Deep Learning* - $6^{th}March2018$

[18] Olaf Ronneberger, Philipp Fischer and Thomas Brox *U-Net: Convolutional Networks for Biomedical Image Segmentation* - $18^{th}May2015$

[19] Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi *You Only Look Once: Unified, Real-Time Object Detection* - $8^{th}June2015$

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu and Alexander C. Berg *SSD: Single Shot MultiBox Detector* - $8^{th}December2015$

[21] *RESTful API Wikipedia article*

[22] *Python website*

[23] *Pytorch website*

[24] *Torchvision documentation*

[25] *Tensorflow website*

[26] *OpenCV website*

[27] *Pandas website*

[28] *Matplotlib website*

[29] *Flutter website*

[30] *Dart website*

[31] *Flask website*

[32] Artificial Neuron [Image]: **https://www.researchgate.net/figure/Similarity-between-biological-and-artificial-neural-networks-Arbib-2003a-Haykin_fig2_326417061**

[33] Fully Connected Network [Image]: **https://www.researchgate.net/figure/A-typical-Artificial-Neural-Network_fig1_346428006**

[34] Bias example [Image]: **https://makeyourownneuralnetwork.blogspot.com/2016/**

[35] Gradient Descent graph [Image]: **https://medium.com/analytics-vidhya/understanding-gradient-descent-8dd88a4c60e6**

[36] Extracted features example [Image]: **https://www.researchgate.net/publication/337804593_Deep_learning-based_image_recognition_for_autonomous_driving/fulltext/5e614a0592851c7d6f2586f3/Deep-learning-based-image-recognition-for-autonomous-driving.pdf**

[37] VGG-16 architecture example [Image]: **https://cs.colby.edu/courses/F20/cs343/lectures/lecture14/Lecture14Slides.pdf**

[38] Faster R-CNN architecture example [Image]: **https://www.researchgate.net/figure/Architecture-of-F-RCNN-12-Demonstrates-the-architecture-of-Faster-RCNN-There-is-a_fig1_344775420**

[39] Anchor boxes example [Image]:
https://link.springer.com/chapter/10.1007/978-1-4842-6543-7_10

[40] Image segmentation example [Image]:
https://www.mdpi.com/remotesensing/remotesensing-12-01667/article_deploy/html/images/remotesensing-12-01667-g003-550.jpg

Appendices

The report should read as a self-contained document. In addition, there should be a number of appendices submitted electronically; see the project webpages for details.