Investigating the use of a Regression CNN Model for Hazard Detection during Autonomous Landing on Mars

Ryan Gascoigne-Jones

Abstract

The current systems used for autonomous landing onboard autonomous spacecraft have accuracy and adaptability limitations. A deep learning approach to producing a hazard detection system could improve the precision and accuracy of these automated terrain analysis systems, enabling the spacecraft to land in a safer landing site. This report details the production of a camera-based hazard avoidance system using a convolutional neural network model trained for regression to predict the smoothness of a given area of terrain. This model was investigated to determine how well this approach performs, and demonstrate the capabilities that a hazard detection system produced by this method could have. This project necessitated the creation of a simulated dataset that closely mimics Mars terrain in order to train the model on a large enough labelled dataset. The training and optimisation of the model was evaluated and iterated on to ensure the investigation studied the extent to which a model could be trained. The produced model was tested and evaluated to determine its strengths, weaknesses, overall prediction effectiveness as well as its ability to generalise to unseen authentic images of Mars. This model was then used to produce a system emulating how the hazard detection system would work. This used the smoothness values that the model predicted to determine the smoothest and therefore safest prospective landing site within an image of terrain.

Keywords: Deep Learning, Computer Vision, Autonomous Landing, Hazard Detection System, Aerospace

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction

Autonomous extraterrestrial exploration is at the forefront of modern space research and is heavily invested in by organisations like NASA. The autonomous missions require bespoke systems to control and make decisions onboard the autonomous spacecrafts. This is the case in the recent Mars 2020 Perseverance mission, whereby a rover was landed in Jezero crater to study the rocks and other sediments in Martian soil in order to gain a better understanding of the planets history [1]. This report details an investigation into producing a camera-based hazard avoidance system (HDA) that could be used alongside or as part of a Terrain Relative Navigation (TRN) system within an autonomous spacecraft. HDA systems are used to help guide a spacecraft away from hazards during it's landing sequence when human control is not possible. This is often needed because autonomous spacecraft are sent to planets and moons far away from earth, meaning there is a significant communication time delay with Earth, on Mars this can range from 4 to 20 minutes [2]. This necessitates autonomous control of spacecraft during time constrained events such as landing, where split-second decisions have to be made.

Research-based missions prioritise certain areas as more scientifically-viable than others, these tend to be in areas where the terrain is more dangerous and rugged. Landing in these areas can save months of travelling for a Mars rover but it means that the landing sites are often cluttered with hazards such as boulders, craters and slopes. For this reason HDA systems are needed onboard to mitigate the risk of landing in these areas to justify the decision to land in such precarious terrain, where a crash would be almost certainly fatal to the mission. There are many existing HDA systems that have been used on NASA (National Aeronautics and Space Administration - US Government) missions in the past. Many of these systems were produced by NASA's ALHAT (Autonomous precision Landing Hazard Avoidance Technology) project [3] which aimed to improve the accuracy and safety of autonomous landings on other planets. These systems used a variety of methods such as pattern matching and image correlation to measure the craft's position and velocity for TRN objectives [4]. HDA systems were also developed by ALHAT for Hazard mapping, done by analysing terrain topography, Piloting, the decision making system for selecting a landing site (LS), and Guidance, the system in charge of diverting the craft [5].

This project aims to produce a HDA system capable of detecting the smoothness of the surface of terrain from above. This will be used to calculate the safety of a prospective LS during the spacecraft's landing sequence. It will do this with the use of a computer vision (CV) model trained by a convolutional neural network (CNN). The CNN will be a regression based model, as it will attempt to estimate the surface smoothness of an image. Previously developed HDA systems mostly used an algorithmic approach to determining safety such as pixel-based hazard mapping, automatic thresholding and grey level standard deviation [5]. Whereas, this project aims to investigate whether using deep learning can produce a more accurate, and therefore safer system. This system could also be more helpful than existing HDA systems in areas which have not been thoroughly mapped out by satellite imaging, such as other planetary bodies that have not yet been explored or studied closely. The HDA system will aim to emulate a pilot looking at the ground and could be used as a backup to currently in-use primary HDA systems or an added safety validation feature to decrease the chance of errors leading to costly accidents. There is also the added possibility that this model is able to discern surface smoothness and terrain features to a higher degree than humans can with the naked eye and could act as a guide to aid human pilots.

The project will base it's operating requirements on the latest autonomous mission to Mars, the Mars 2020 Perseverance Rover mission [1]. The project will be constrained by the hardware on this mission (cameras) [6], and the points at which the HDA system could be used (landing sequence timings and procedures). This meant that the model was trained as if it was to be used on the Mars 2020 mission. For example, the spacecraft could only start taking photos at 2.5km above the surface when its heat shield was jettisoned [7]. The photos were taken by the lander-vision camera system, which took 1024x1024 pixel top-down images of the terrain below, as this was mounted on the underside of the Perseverance

rover. This aimed to produce a model as close to the possible specifications of a future Mars mission led by NASA as is feasible without unjustifiable speculations.

CNNs are a type of artificial neural network (ANN) that tries to emulate a digital brain in order to process images and recognise patterns within them [8]. They require large datasets to be able to create an accurate model capable of generalising well to unseen data. Each image used for training must also have a label, so that during the training process, the network can evaluate how well it is predicting something from the image; this is known as supervised learning [8]. This project will make use of a regression model, with the label being a measurement of some kind to do with the surface topography. A major problem arose when considering the dataset to be used in the training process; there was no labelled dataset, matching these constraints, that was anywhere near large enough to properly train the model.

This meant that a simulated dataset needed to be created for the model to be trained on. It needed to be realistic enough for the model produced to be able to generalise well to real-world imagery, so the dataset created had to simulate all aspects of Martian terrain, including topography, rocks, and craters. This also meant that the labelling could be very accurate as the system used to render the dataset made use of a terrain map, which topographical measurements could be easily extracted from [9].

To summarise, the aims of this project are:

- Investigate whether a regression-based CNN is capable of creating a HDA model which can safely select a LS on Mars.
- Generate a simulated dataset of Mars which can be used to train the aforementioned model.

2 Project Specification

The hypothesis of this project is to investigate whether a CNN model can be used to create a HDA system that can select a safe enough LS. The dataset used will be computer generated and must aim to simulate Mars as realistic as possible. In order to do this, each image must have:

- Mars-like terrain topography.
- A variation of rocks and boulders.
- A chance of some craters that imitate those found on Mars.
- A reasonable level of sunlight illuminating the terrain, as it would on Mars.

Each image must also mimic how the images were taken in the Mars 2020 mission [1]. This means that the images must seem as though they have been taken:

- From an altitude of around 2.5km.
- Using similar hardware to that used on the mission.
- From a spacecraft that swings slightly, as if it were wobbling around, but staying relatively vertical.

The features within each image must be varied and have a small degree of randomness so that each image is somewhat unique. This is to prevent the model from overfitting the simulated dataset, thereby reducing its ability to generalise onto unseen and possible genuine Martian imagery. Each image must then be broken up into smaller patches, as they provide superior training yields for CNNs. Each patch must also be assigned a label, for use during supervised training. This label must represent the terrain topography in some form of measurement that the model will then have to learn to predict. The dataset should then be used to train the model by being broken up into training, testing, and validation datasets and used appropriately. This is to prevent the model from overfitting the data by learning the dataset too well,

meaning it wouldn't be able to generalise to unseen data as it has learnt the patterns of the training dataset too closely. This issue is resolved by separating the datasets, with a large proportion still being used to train the model, but another smaller subset of data, called the validation dataset, being used to check how well the model is predicting unseen data at the end of each training pass. The results from this check are then used to adjust the weights of nodes within the CNN, in an effort to make the model better for predicting unseen data instead of learning the unique patterns of the training dataset. As well as this, a smaller subset called the test dataset will be used to test how the model generalises after the model has been trained. The dataset split will be:

- 70% Training
- 15% Validation
- 15% Testing

The model itself should aim to follow a similar process as in Giusti et al. 2020 [10], which was reviewed in the literature review conducted as the research for this project [11]. This used a CNN to predict the roughness of image patches of materials [11]. The difference with this application of the system being that Giusti et al. was predicting roughness in terms of micrometers, whereas this model will be trying to predict smoothness in metres, 6 orders of magnitude larger. The CNN used in Giusti et al. can be used as a starting point, but it must be adapted to use the different sized image patches as inputs, and then each layer must then be modified accordingly. This must then be evolved and optimised once the dataset has been finalised for it to be more appropriate for this purpose.

2.1 Plan

The first step will be to produce a number of different test datasets; these should be small enough so that model training can be carried out quickly, but also not too small, as they still need to be representative of the sizeable final dataset. This is so preliminary investigations can be carried out to find which variant of dataset is the most optimal to train the model. These investigations must be carried out on the exact same set of images to ensure an unbiased review. Firstly, the number of patches per image will be investigated, with small datasets of variations of patches being compared. The variant with the best performing loss function however, may not be chosen, as there are other factors, like time taken to generate each patch, which may be used to distinguish between two datasets with relatively similar performances. The measurements of the terrain topography used as labels must then be investigated. These could include different forms of measurements such as range and standard deviation of the terrain in its y-axis. The terrain itself could also be investigated, to see if the model is biased towards the terrain topography (hills, cliffs), craters or rocks and boulders.

Whatever dataset is found to be optimal from these preliminary investigations should then be preprocessed, with different kinds of preprocessing also being studied. This will involve finding different kinds of outliers in the dataset and examining what the results are when they are excluded from training. This should be tested using datasets that do and do not contain the aforementioned outliers. The distribution of the dataset should then be analysed, with normalisation and standardisation investigated for how well a prospective model using these would perform.

Once these investigations into the dataset have been carried out, a larger dataset of the most optimum dataset found should then be compiled. This should then be used to optimise the CNN model structure. Features that should be investigated include input shape and the size of the image used as the CNNs input; an image of a set size can be downsized or upsized using different image sampling techniques. The filter size and number of filters of each convolution layer should also be investigated. The loss functions used to change the weights of neurons should also be considered. The number of epochs and early stopping of the training can also be reviewed to determine whether this would lessen overfitting.

Once the architecture of the model has been finalised, the full dataset should be used to fully train the model. This should then be used as the backend part of a small-scale GUI (graphics-based) system where the user can import an image and have the model predict the safest (smoothest) LS from within it, showing the result graphically. The model should aim to be able to predict the smoothness of the surface terrain in a patch within a moderately small error range. The level 0 requirements of the ALHAT project had hazard detection requirements of classifying a feature that is 30cm above or below the surrounding terrain and any slope 5° or greater [3]. For this reason, various tiers of accuracy of the model will be quantified within an error range using the estimated gradient of a slope. As the ALHAT project used a gradient of 5° to qualify a slope as a hazard, it will be designated as the base tier (1) of model accuracy. It will also employ 2 other lower tiers and a higher tier called tier 0, this tier represents an even greater degree of accuracy to gain a better understanding of the models success. These will each decrease in the predicted safety margin for every downward step in the hierarchy. The criteria for these tiers is as shown in the following table, where the accuracy classification is the minimum accuracy expected of the model in the corresponding error range:

Tier	Slope gradient error range	Accuracy classification
0	2.5°	40%
1	5°	80%
2	10°	90%
3	15°	98%

These values were chosen because slopes above 5° are classified as hazards by ALHAT and those above 20° are classified as very dangerous [3]. Therefore tier 3 should be near perfect accuracy as an error this large would be very dangerous, with the same reasoning being applied to tier 2, with slightly less strictness. Tier 1 would ideally have an accuracy of 95% but acknowledging that this project is a smaller scale to the ALHAT project leads to some leniency. Tier 0 will have a lower accuracy as it will be used to gauge how good the model is at distinguishing small differences that the human eye may not be able to discern.

Once the final model has been produced and installed into the backend of the GUI system, this should then be used to test different varieties of terrain. This would include features like adverse lighting and overly-rugged terrain, as well as comparing these to more standard terrain with lots of even ground. Also, authentic images of Mars should be tested on the model to gauge how well the model would work in its intended operating conditions. This may be difficult to quantify as there aren't any publicly-available images of Mars of a high enough quality and also in the form that the simulated dataset used to train the model was in. Even fewer data sources will have images in which terrain data of Mars can be easily associated with, without using TRN systems that aren't publicly available to determine the location the image was taken. For the Mars 2020 mission, NASA only publicly released 7 images from the lander-vision camera taken during the craft's landing [12]. There is also no easy way to correlate terrain information with this accurately enough, even though there is a large amount of data due to the site being very heavily studied. This means it will be very difficult to quantify how well the model generalises to authentic images, but it can still be judged how well the model predicts these images compared to the human eye.

3 Dataset Generation

3.1 Design

3.1.1 Overall Design

The dataset needed to be generated had to be at a high enough quality and realistic enough so that the model it would be used to train could generalise well to possible real images of Martian terrain. The

dataset also had to accurately represent the conditions in how real images taken from a lander would be taken. In addition, the dataset had to include the majority of typical terrain features, such as boulders, rocks, craters and an overall uneven terrain and surface. To produce a large and varied enough dataset with these characteristics specialist scenery generation software would need to be used. This was then researched and found that a company called Planetside Software have a piece of software called Terragen 4, which is often used to produce scenes for film, TV, games and VR [9]. With Terragen a scene can be created, rendered into an image, and a 3D model can also be produced from it.

The ability to generate a 3D model alongside it's corresponding image would be especially helpful for this project as each image in the dataset had to be labelled so it could be used for supervised learning. The 3D model of the terrain could be used to generate labels by parsing each vertex in an object file to calculate different measurements of the terrain. The images generated could also be outputted in grey scale which was ideal as the model would then be able to generalise more easily to other terrain that doesn't match the exact colour chosen to illustrate Mars.

Terragen utilises a GUI to interact with the user, in this the user can adjust and fine tune essentially anything to do with the scene. Since each image in the dataset had to be a unique image with random and varied features, this would require a lot of repetitive work to create a new scene and then render it. However, Planetside also offered a python [13] module, called Terragen-rpc [14], which could be used to change certain parts of the scene from within a python script, as long as the file was open on the system. This could then be ran using a batch file running a loop with this script and a render command (which was not possible to do from within the script) to automatically render hundreds of images without any manual input necessary. This script could randomise and vary features such as number, size, and distribution of rocks and craters. Light intensity and direction could also be varied, simulating the differing positions of the sun. The position and angle of the camera could also be changed to simulate the swinging of the spacecraft and slight height variations in relation to the ground. This random variation is done so that the model will not as easily overfit the data and will be able to generalise better to unseen images.

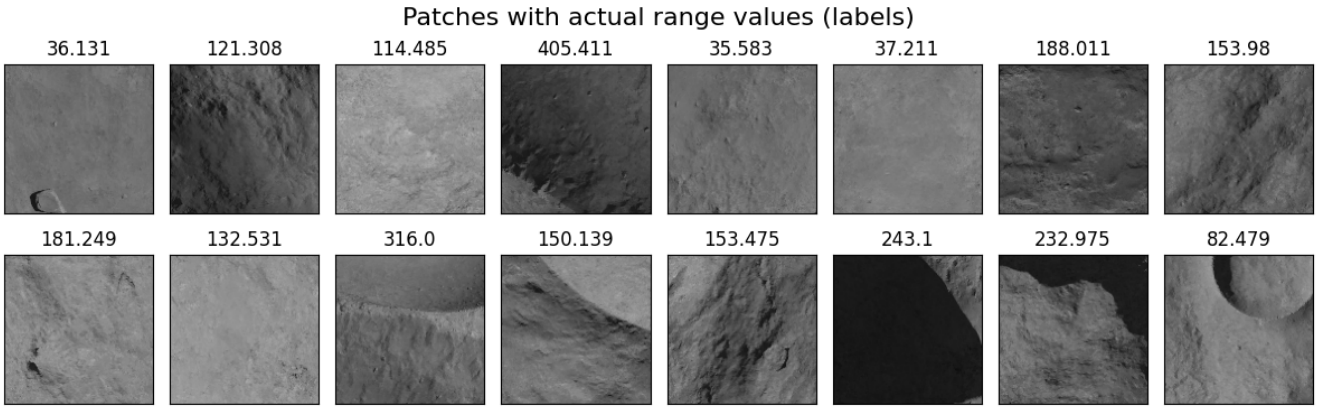

Figure 1: Example patches generated.

3.1.2 Feature Design

The default terrain on Terragen was too uneven and rugged to accurately depict Martian terrain, so the noise in the terrain was reduced to smoothen it. This would replicate the effects of the loose sands [15] and high winds [16] of Mars producing a smoother, flatter surface than that of Earth. The terrain would also be limited to around 700m to reduce the number of large hills and mountains which are less likely to be at a potential LS. The colour of the terrain would also be set to better represent the surface of Mars. Within the python script the seed of the terrain would be randomised each time to produce a new surface shape for every image.

Rocks and boulders are very common on Mars, especially in areas where a prospective LS is likely to be [16]. To better represent the various size of these, three different sizes of rocks were to be implemented. Each different size would be implemented in a different layer, with the smaller rocks being on average 0.5m in diameter, medium being 10m, and large rocks being on average 50m in diameter. Each size has a differing number and density, with there being many more small rocks and in higher densities than large rocks with medium being in between the two. The densities of rocks would be set so that rocks should appear more in clumps to simulate boulder fields. Within the python script, for every scene, the seeds of every rock is randomised along with the density pattern, and the colour of the rocks is slightly varied from the colour of the surface of Mars to represent different rock types [15].

Diameter Dependent Impact Crater Parameters and Relationships

Parameter		Simple crater relationship	Complex crater relationship	Large crater relationship	Comments
depth	d	$d=0.21D^{0.81}$	$d=0.36D^{0.49}$		Fit to profile data for simple craters, DEM data for complex and large craters.
Rim height	H	$h=0.04D^{0.31}$	$h=0.02D^{0.84}$	$h=0.12D^{0.35}$	Complex 7-100 km, large >100km. Used DEM and Profile data.
Central Peak Height	h_{cp}	—	$h_{cp}=0.04D^{0.51}$	—	Not done for large or simple craters. Complex range 7-100 km. Used DEM and Profile data.
Central Peak Diameter	D_{cp}	—	$D_{cp}=0.25D^{1.05}$	—	Not done for large or simple craters. Complex range 7-Used DEM and Profile data.100 km.
Cavity Shape	z	$z=0.204x^{1.76}$	$z=0.008x^{2.65}$	—	These are for profile data, for the function $z=kx^n$ as discussed in the text. Preliminary simple crater DEM data fit: $z=0.21x^{1.68}$, and complex crater DEM data fit $z=0.014x^{2.34}$
Cavity shape fit coefficient	k	$k=0.76D^{-1.17}$	$k=5.62D^{-2.51}$	—	$z=kx^n$, for profile data "good" fits
Cavity shape fit exponent	n	$n=1.04D^{0.366}$	$n=1.62x^{0.14}$	—	$z=kx^n$, for profile data "good" fits. For "good+fair" (includes central peak and polar filled craters) and "good+fair+poor" fits, the simple crater relationship does not significantly change, while the complex fit changes to $n=1.88x^{0.10}$ and $n=2.10x^{0.07}$, respectively.
Inner Cavity Wall Slope	s	$s=28.40D^{-0.18}$	$s=23.82D^{-0.28}$	—	Complex craters 7-100 km. Simple fit to profile data. Complex fit is to DEM data. Note offset of fits at transition.

Figure 2: Typical relationship between diameter and other features of impact craters found on Mars [17].

Craters on the surface of Mars are much more common and larger than on Earth because the atmosphere is much thinner, this results in meteors and space debris not being slowed down and burnt away as much as on Earth [18]. To represent this, a minimum of zero and maximum of five craters could be generated in each image. The craters were separated into the more common small craters, with diameters from 125m to 1250m and then a slimmer probability of larger craters, which are between 1250m and 2500m. The depth, rim and overall shape of the craters would be generated in a way that reflected that of typical Martian craters [17] (as shown in Figure 2) with slight random variations. The python script will randomise the number of craters and then each crater's location, diameter, and depth to give some variation among craters so the model will have greater generalisability.

The camera used to render the image of the scene would be set at 2.5km above the ground, to mirror the point in the landing sequence where photos would first be able to be taken [7] (as shown in Figure 3). Using images from this altitude would give the best opportunity to analyse the surroundings and make a decision on where to land, therefore the model should aim to reflect this. This height is varied by up to 250m to ensure the model will be able to give accurate predictions while the height at which the images are taken are not exactly 2.5km, to account for altimeter errors or tall terrain features. To simulate the spacecraft swinging from it's parachute during its descent, the angle of the rendering camera would be varied by under 5° for the majority of images, but with a slim probability (0.25) of up to 10°. The field of view and focal length of the render camera would be set identically to that of the lander-vision system camera [1]. The resolution of the camera used on that spacecraft was 1024x1024 pixels, but as more modern cameras will be used for future missions (an assumption was made that newer cameras would be used) the images were rendered in 2K resolution (2160x2160 pixels).

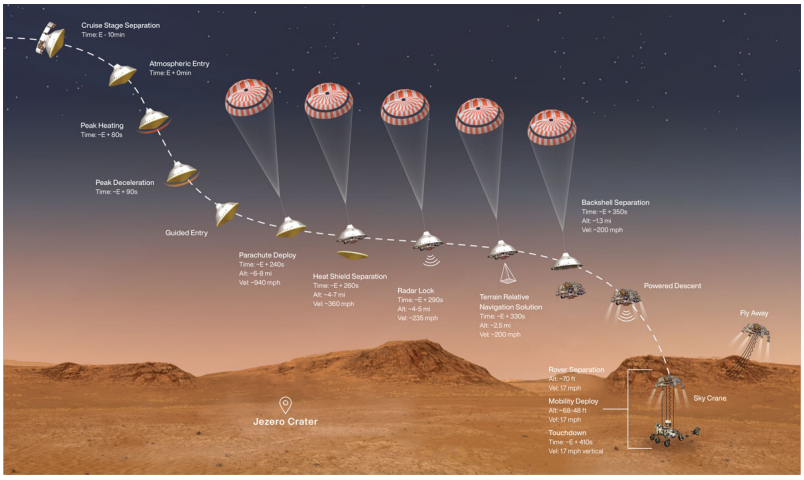

Figure 3: Perseverance's Entry, Descent, and Landing Profile [7].

The main source of light on Mars is from the Sun, and due to the planet having a thinner atmosphere, which is also composed of different gases, the light from the Sun appears more blue than on earth [19]. The sun light source within the scene will be adjusted to reflect this. The heading of the sun is also randomised so that the model will not be biased towards expecting a shadow to be facing a certain direction. Also, the elevation of the sun, which dictates how long shadows appear, is varied, with lower probabilities for circumstances where the sun is directly overhead, as this is much less frequent than at a slight angle. Also, there is a slimmer probability of the sun being at a very acute angle as this can put a large part of the image into shadow, which is unhelpful as training data, and the model will not be expected to work in lower light conditions anyway. The light intensity from the sun was adjusted from the default value. As Mars is further away from the Sun than Earth, slightly less light reaches the surface [19], therefore the light intensity of the sun within the scene must be adjusted to illustrate this.

3.1.3 Design Conclusion

This project aimed to produce around 30,000 image patches, with associated labels, that could each be used for training and testing. Each patch would be a smaller section of a larger image which has been cropped. Each image could be broken up into either 4, 16, 64 or 256 sections; this could be broken down further, but after this the images become slightly too small to be useful for training. This made it easier to generate lots of different patches quickly as, if there were 64 patches in a full image, generating that full image will be 64 times faster than if done so individually. The object files could also then be broken down in the same way that the patches were being cropped. These sub sections of object files could then be used to generate the labels for the corresponding image patch.

3.2 Methods

The full images, of which many patches are a part of, first need to be generated. The images were generated at 2160x2160 pixel resolution in grey scale, with a 5 million vertices object file produced alongside it. All this was carried out by a batch file running a combination of scripts and render commands, for automated generation of the entire dataset.

The batch file itself first ran a command to render the image from the currently generated scene on Terragen. This created an image file and associated object file and were placed in separate directories. This main image was then cropped into a number of different patches using OpenCV [20] (a third-party python library), which were then saved as separate JPEG files. Each image patch was named so that it could be identified from which main image it had come from and from where within it.

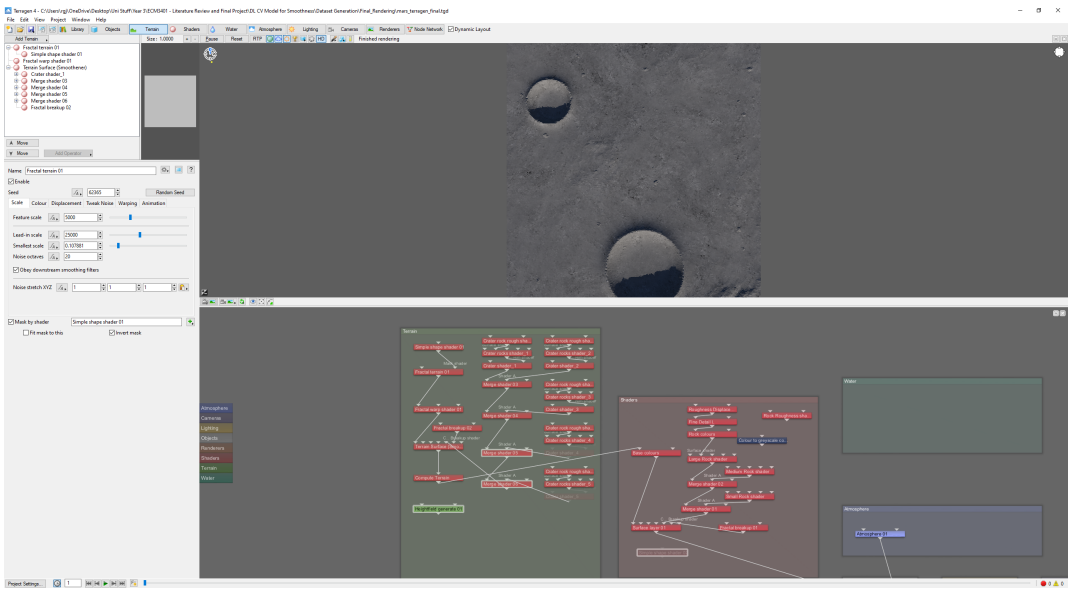

Figure 4: Screenshot of Terragen GUI after a script has generated a scene [9].

The labels for each patch were then to be calculated. Each object file was split up into a grid just as the images had been cropped. The range of y values in each patch was then calculated by parsing all the vertices in the object file. The calculated range measurements for each patch were then compiled into a text file for each main image, with each row having the value for a separate image patch's label. While investigating different ways of measuring smoothness, standard deviation was also calculated for a test dataset. This was done in a similar way while using NumPy's standard deviation function [21]. Once the labels for all patches within a main image were fully calculated and the text file had been compiled, the object file was deleted as it was no longer needed.

Next, the script in charge of generating a new scene within the Terragen software was ran. This used the python module offered by Planetside, terragen-rpc [14]. The script itself randomised seeds, distributions, and densities of rocks, craters and terrain as well as varying the the number of craters in each scene. It varied the size and shape of rocks to ensure each image had unique features to prevent overfitting. Also, it randomised different attributes to do with the camera and lighting, as outlined above in the model design section (3.1). The script then saved this scene to the project file, overwriting the previous scene, as this had already been rendered. Once the batch file had executed this script, the image number was iterated, and the process was repeated on this newly generated scene.

3.3 Rendering

For the rendering of the dataset a windows computer was used. The rendering process is heavily reliant on the processor and barely uses the GPU, while using only around 4GB of RAM. The computer used ran Windows 11, which enabled Terragen to be ran and the batch file to be executed easily. This could not be done on a remote resource from the command line as the GUI needed to be open in order for the script to automatically modify it. The CPU on the system was a i7-13700H, which had 14 physical cores and up to 20 threads, running at 4.2GHz each. A powerful CPU was necessary to carry out complex rendering tasks such as ray tracing and also to reduce the time each image would take to render. With this system it was possible to generate on average, 45 full images, with accompanying object files, per hour. This was ran for a total of around 40 hours generating a variety of test datasets for investigations as well as the full dataset used to train the final model. This produced a total of around 1500 full images, 460 of which were used for the final model. The final dataset, with each image being broken down, totalled nearly 30,000 images (before preprocessing).

4 Model

4.1 CNN Basics

Artificial neural networks are machine learning models designed to mimic the structure and function of a human brain, using a large network of interconnected nodes called neurons [8]. These neurons are assigned weights and can be combined to produce a result, if these weights are trained properly the model can predict an output from a given input with relatively high accuracy. ANNs often have hidden layers, these use backpropogation, an algorithm designed to calculate the gradient of an error, to optimise the weight of the neurons based on the results of the previous layer [8]. Multiple hidden layers stacked onto each other is what is known as deep learning [8]. Convolutional neural networks are a variant of these that focus on image-based pattern recognition, this is done by converting the image into a matrix with a single space representing each pixel. CNNs are used for two different tasks, one being image classification, where the model tries to classify the image into discrete categories. The other is image regression, where the model tries to predict a value of some kind representing that image, this is the kind of CNN that will be used in this project. CNNs are feed-forward networks, meaning data only travels in one direction and isn't iterated on in a single pass.

The basics to training an algorithm involves feeding in an entire dataset into the model in batches, this is known as an epoch of gradient descent [22], where the weights of the neurons are attempted to be optimised. The accuracy of the model is then tested with a loss function on unseen validation data that will help the model generalise to new unseen data in the future. The full training is usually carried out for many epochs, usually until the validation loss function starts to plateau or increase whilst the loss function for the training data continues to decrease. A loss function calculates the difference between predicted outputs and the actual expected outputs, the lower this is the better the model should be.

The core element of a CNN is the convolution layers, these take the input image as a matrix and apply a number of filters on it. These act as a window, where only pixels inside the filter are taken into account, this is where the model essentially focuses on a small square section of the image, typically only a few pixels wide [8]. These filters are moved around the image at a rate defined by the stride value, with neutral padding values added to the edges so the filters can properly analyse the corners. With the results of these filters, a feature map of the image is created, an activation function is then used to decide whether a neuron for a particular filter should be activated or not [8]. ReLU is often used as the activation function as it solves the vanishing gradient problem, whereby lots of neurons are completely deactivated and ignored [23], leading to far less learning. Another major feature of CNNs are pooling layers, these are used to downsample images while maintaining the meaningful characteristics of the data. This is carried out by aggregating groups of pixels and summarising them using pooling windows that act similar to filters. This can be done by either taking the maximum value of a group (max-pooling) or taking the average value of a group (average-pooling). The final defining feature in any deep learning CNN is a fully-connected layer (a.k.a. dense layer), consisting of hidden layers of neurons that will be optimised during the training.

4.2 Original Design

The neural network itself will take as an input a grey scale image patch of 270x270 pixels. The rest of the model was adapted from Giusti et al. 2020, which used four convolution layers with a max-pooling layer after each one, with a dense layer at the end [10]. This model was used as the starting point as during the research conducted in the literature review, it was found that this was used for a similar purpose as this project. It was used to predict the roughness of the surface of materials in terms of micrometers from top-down images of 752x480 pixels [11], whereas this project will predict roughness in terms of metres and an image size of 2160x2160 pixels. Despite the difference in measurements being large, the overall strategy should remain relatively similar.

The original model adaption of this used 3 convolution layers, with a max-pooling layer after each one, as well as a dense layer at the end and the output being a single scalar value. Similar to the model used in Giusti et al., the first convolution layer had the least number of filters with 64 filters each of a size of 10 by 10. The next convolution layer had 128 filters each of a size of 6 by 6, and the final one had 256 filters each of a size of 4 by 4. In between each convolution layer was a max-pooling layer, with a pooling window size of 2 by 2, that sized down the shape by a factor of 2 each layer. The final layer was a dense layer of 270 hidden neurons, this was the part to be trained by optimising the weights. The model took an input shape of a 270x270 pixel patch and output a single scalar value predicting the smoothness of the input patch. This model used the ADAM optimiser with a learning rate of 0.001 and a loss function of mean squared error. It used a stride of 1 and equal padding, and was ran for 30 epochs of gradient descent with a batch size of 256. The original compilation of this resulted in a validation loss function of 0.015 (mean squared error).

4.3 Methods

The design, training and testing of the CNN model was all completed within Jupyter notebooks [24] using Python. The model was implemented in Python using the Keras deep learning library, with Tensorflow used as its backend. As training CNNs is a very computationally expensive task, it is important to have a strong GPU. For this reason the hardware accelerators offered by Google Colab [25] were used. For the investigation of the dataset an L4 GPU was used, this had 22.5GB of GPU RAM and was sufficiently powerful to train models on the smaller dataset. However, when training the CNN with the final dataset, which was around six to ten times larger than most training datasets, an A100 GPU was used. This is the most powerful hardware accelerator offered by Google Colab, with 40GB of GPU RAM, which would be fully utilised to train the models as swiftly as possible. Using these powerful GPUs hosted by Google enabled a more efficient investigation and optimisation of the model to be carried out, resulting in an overall better model, as more varieties of it had been explored in the same amount of time. The results of these investigations were analysed using tools from SciPy [26], Seaborn [27] and Matplotlib [28] Python libraries.

4.4 Investigations

Before any investigations could take place, some preliminary preprocessing had to be done on the test datasets. This involved removing any patches which had erroneous or wildly anomalous label values. There was a small minority of patches with null or 0 values, these were usually from the edges and more specifically corners of images. These values were created like this due to the terrain being very sloped in that area, resulting in too few total vertices in that specific patch to generate a representative label value. This was also the case for the anomalous data, with nearly all being from corner patches. These along with the null and 0 values were all removed from the training and validation datasets to ensure the training was not contaminated, with only the test dataset retaining some less anomalous corner patches for testing purposes.

Firstly, the type of measurement of smoothness used for the labels was investigated. Using the range of y values within a patch area was compared with using the standard deviation of y values of vertices within a patch area. This was tested using the same set of 100 images each being broken down into 64 patches. Each dataset was ran through the same original CNN for around 50 epochs. The range test had a best validation loss of 0.016 and the standard deviation test had a best validation loss of 0.02. This told us that the CNN was better at being able to predict the largest height difference within a patch rather than the standard deviation of the entire patch's surface in the y axis.

Then the number of patches was tested using the same set of 50 main full images, with 3 different datasets using 16, 64 and 256 patches respectively. The results of these tests showed that using 16 patches was considerably the worst. It was found that using 256 patches was the best at 0.0115, closely followed

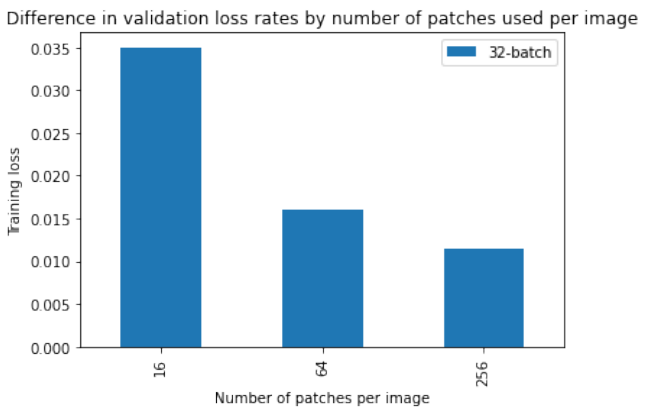

Figure 5: Graph showing different validation loss rates when using a different number of patches.

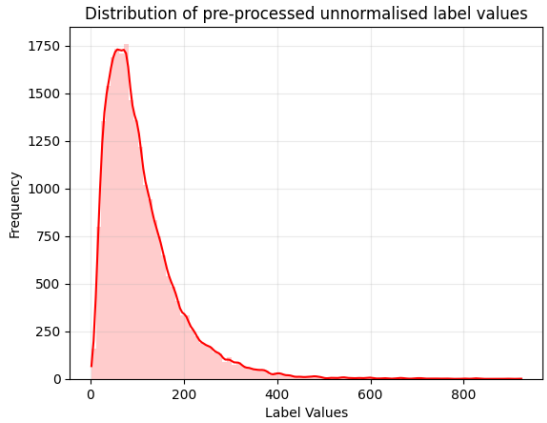

Figure 6: Graph showing distribution of label values after preliminary preprocessing.

by the 64 patch test at 0.016 (as shown in Figure 5). Even though the 256 patch dataset trained the model slightly better, it was decided that further datasets should use 64 patches. This was because even though this would give a slightly weaker validation score, it would be alot quicker to generate (4 times quicker). It was also found that there were slight inconsistencies between the cropping of the images into patches and the breaking down of the object file when the terrain was highly irregular, resulting in small number vertices being counted in a neighbouring patches measurement. This was much more prevalent and obvious in the 256 patch test (and would be even more so in larger datasets), as while the absolute error stayed the same, the smaller patches meant the percentage error quadrupled.

After each of these investigations had concluded, the best model design so far was used to predict the smoothest patch in a full image, simulating what the model was being produced to do. While carrying out these tests, it was found that images with a more normal distribution and smaller range of label values, indicating a large area of even terrain, were being more inaccurately predicted. This meant that the model was finding it easy to predict smooth terrain but was having great difficulty predicting rough terrain. This is shown in Figure 7 in the graph representing the predictions for the image 0 (bottom-left), which has a positive skew of label values (as seen in Figure 7 top-left graph showing distribution of labels within image 0). In this graph the grouping is very close for the patches with low label values and they are being predicted reasonably accurately, whereas the patches with higher values are much more inaccurately predicted. When this is compared to the same graph for image 4, which has a more normal distribution of label values, there was far less grouping, and the patches were being predicted significantly less accurately. It was hypothesised that this was because the model was overfitting to patches with low label values, and was therefore not able to accurately predict areas of rough terrain. This was because there was a large positive skew in the distribution of the label values (as shown in Figure 6), due to most label values being quite low compared to the more rough terrain which had very wildly varying labels. The consequence of these findings led to the realisation that an area of rough terrain could be inaccurately predicted as the smoothest patch and therefore, the model would select an unsafe location as the LS.

This led to the investigation of whether changing the distribution of the training dataset itself would help to train a less biased and more balanced model that could predict rough patches more accurately. In order to compare the two models, a different form of measurement had to be used as validation training loss will not be as representative because the dataset was transformed. For two of the benchmarks, accuracy within an error range of 0-25m and also 0-50m was chosen, these shall be referred to as A-25m and A-50m respectively. As well as this was the model's prediction capability, quantified by the percentage of full images predicted correctly within the first quartile (lowest 25%) of actual smoothness within that image. In other words, the model would predict the smoothest patch correctly within an error range of 25% of the smoothest patches within an image, this will henceforth be referenced as the Q1 prediction percentage

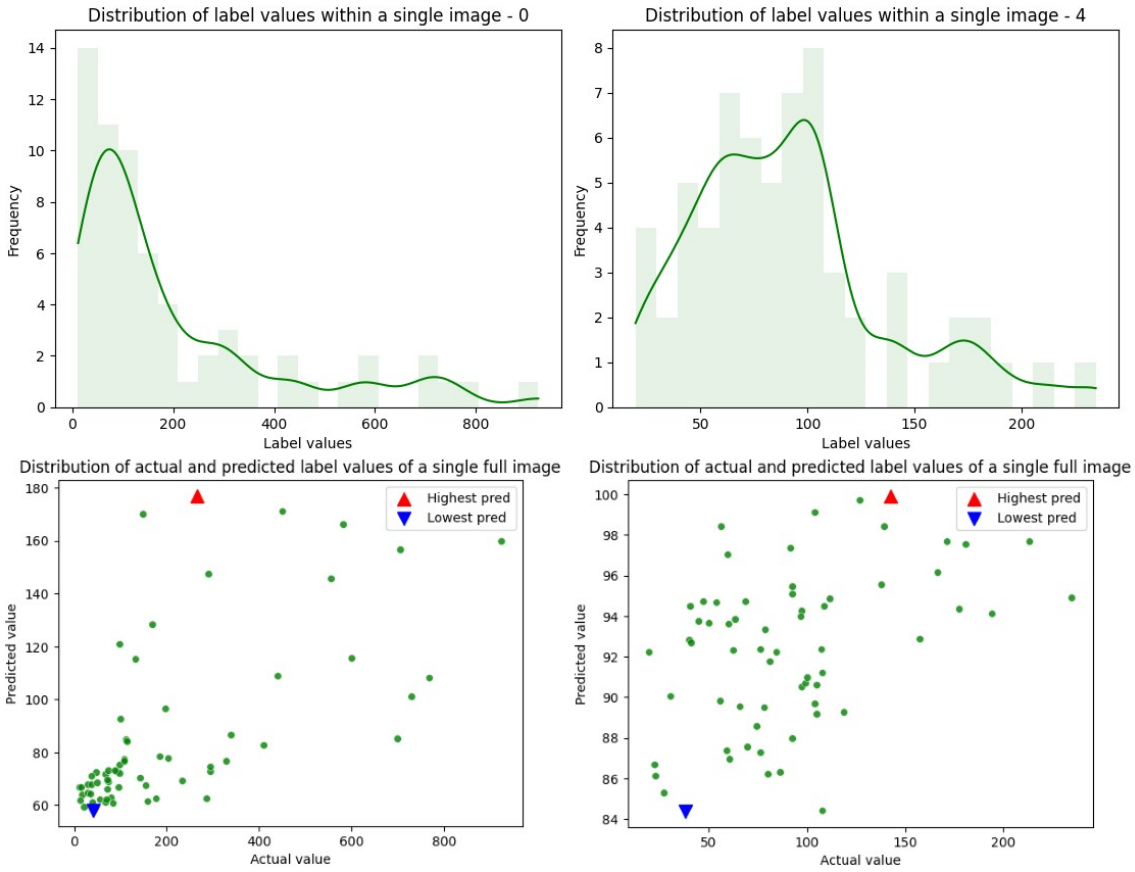

Figure 7: Collection of graphs showing the correlation between the distribution of labels and the predicted values.

(QPP). Other variations of QPP were also taken in future investigations using the 5th and 40th percentile for test control purposes, these will be called QPP-5 and QPP-40. To clarify, the percentiles being used are based off the value of the range measurement or prediction, and not the ranking. For reference, the model produced so far (before this investigation), had an A-25m of 25%, an A-50m of 50.7%, and a QPP of 21.2%.

The box-cox transformation was chosen as the method used to transform the dataset to resemble a normal distribution. This was because it revealed many nuances about the images previously very hard to discern from the dataset in it's original form [29]. This could be used to train the model to recognise rough terrain better, whilst still remaining relatively simple and easy to reverse when it came to getting predictions out of the model [29]. To summarise, the box-cox transformation aims to equalise the variance of a variable by applying a transformation on the original data, x, using a transformation parameter, λ, to calculate the transformed data, y [29]. This is determined using the following equation:

$$y(\lambda) = \begin{cases} \frac{x^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0 \\ \ln(x), & \text{if } \lambda = 0 \end{cases} \quad [29]$$

The same dataset tested on the current model design was then transformed using SciPy's [26] statistical box-cox tool. This optimised the lambda value for the maximum normality, and stored it in a variable so it could be used to inverse the operation, which would be helpful later, for converting the output of the model back into its native domain. The dataset was then normalised between 0 and 1 for better understanding of the loss function values during training. This was then used to train the model for 20 epochs, achieving a validation training loss of 0.0135. The model was tested on validation data, and it was found to have an A-25m of 37.3% and an A-50m of 69.3%, an average increase of 43% in model accuracy (these results are also shown in Figure 8). The QPP value of this was found to be 60.5%, an

increase of 185.4% in image prediction capability. This showed that using a dataset with a transformed normal distribution resulted in a much better and well-balanced model, and therefore would be used in every model and investigation from this point.

4.5 Model Optimisation and Testing

At this point, the loss function being used for adjusting weights within the training process was re-evaluated. The current loss measure being used was mean squared error (MSE), it was theorised that this could be improved by using mean absolute error (MAE). This was discovered to be not the case however, because while MAE provides a value more representative to how good the model is [30], this would be relatively useless, due to the fact the data has been transformed. As well as this, MSE is more sensitive to larger errors, due to the squaring process [30]. This is especially important for this project, because the model's prediction capability could be greatly impaired if a patch with rough terrain was to be highly inaccurately predicted, resulting in a rough area of terrain being selected as the LS.

All of the following investigations and tests were carried out on the full dataset of 460 full images with the model being trained for 20 epochs and only the median of 3 training runs being recorded to reduce the impact of outliers. All data discussed is shown in the graphs in Figure 8. The optimisation of the model as a whole began with first testing the number of filters in each convolution layer. The number of filters within a convolution layer determines how many feature maps are created within that specific layer, with this number increasing for each deeper layer as this enables the model to determine simple patterns first, and then learn increasingly complex features for each layer traversed. For the investigation the number of filters was halved for each convolution layer, this brought the number of filters from 64-128-256 on each layer respectively (first to last layer), down to 32-64-128. It was found that the CNN with the fewer filters per convolution layer produced a better model, with increases in every benchmark metric. It is theorised this could be because the first layer used too many filters and was therefore unable to learn the more simple patterns as well.

The next test was carried out on the size of the filters in each layer, while using the 32-64-128 filter count layout. The original model used a 10-6-4 layout for the size of filters for each layer respectively, the investigation involved testing different combinations, including those with a decrease in size the further the model is traversed, as well as each layer having the same size filters. It was found that although the current layout yielded the best accuracy by a slim margin, the models with the best prediction capability (QPP) were those with the same number of filters in each layer. The model using a 4-4-4 layout proved to be relatively accurate, as well as by far the best QPP-5 of 24.6%, indicating it could more accurately predict an image within a smaller error range; for this reason it was chosen to be used in future models. The stride of each of these filters was also explored, with the original model's stride of 1 being compared to a stride of 2, with the former performing much higher in all benchmarks.

The dense layer was the next to be reviewed, with the number of hidden neurons being investigated. The original model used 270 hidden neurons in the dense layer. This was compared to 135 and 540 neurons, but it was found that using 270 neurons was still the best model. It was decided that although it was slightly worse in terms of accuracy, it had a QPP-5 far surpassing that of the other two.

The final test aimed to investigate whether decreasing the size of the input patches being fed into the CNN. It was hypothesised that this could emulate the increased accuracy that a 256 patch dataset would provide but without the downsides of the increased anomalous data. This was possible by performing a resizing of the image before it was fed into the model. The study found that the aforementioned hypothesis was in fact correct, with a model using an input shape of 135x135 pixels considerably outperforming the former model using the native 270x270 pixels in every model benchmark, with an increase of at least 10% in every metric but QPP-5. Even though some data is known to be lost when downsizing images, these results justified sizing down images in further models as it resulted in a far better model.

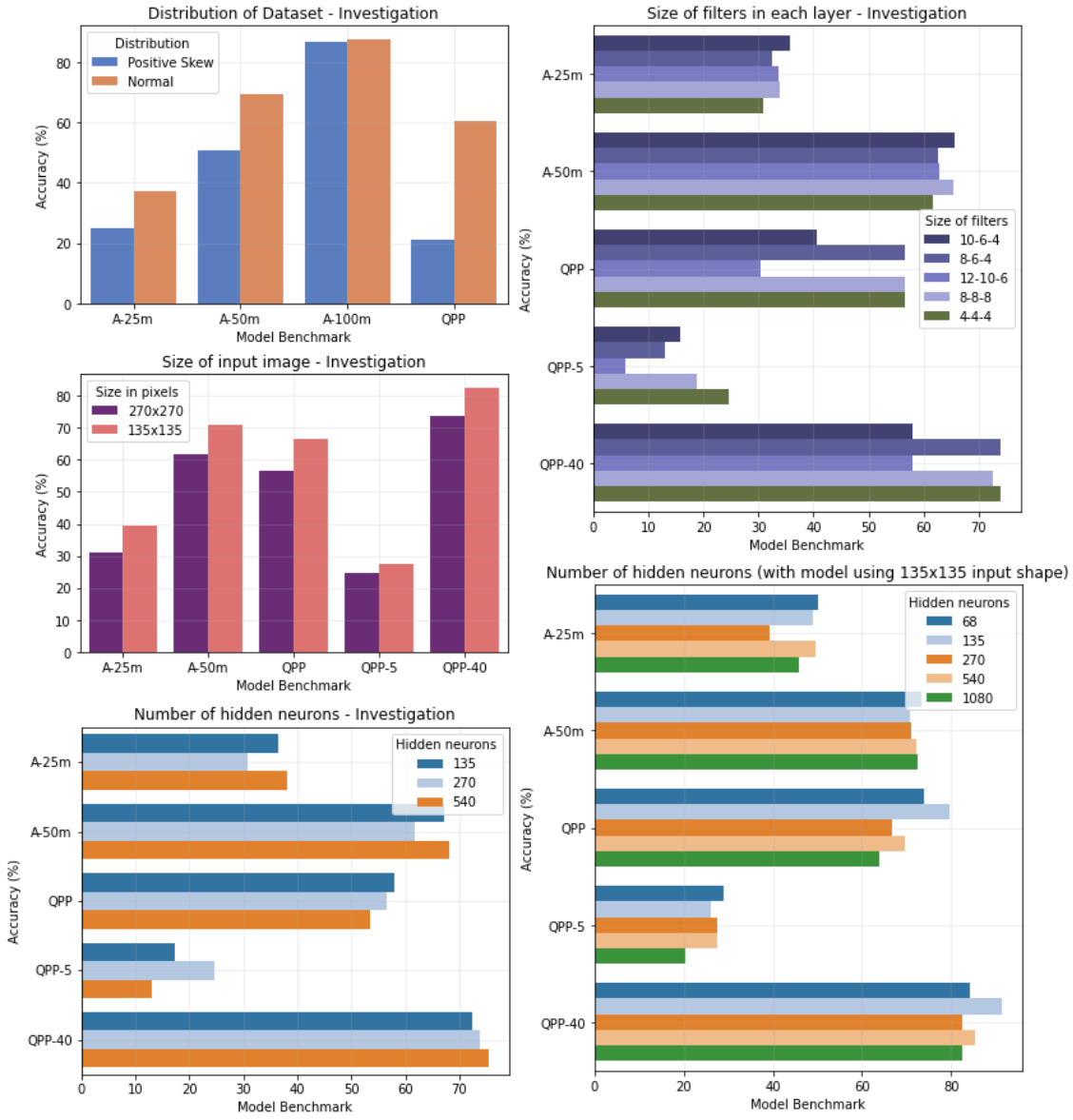

Figure 8: Graphs showing results of some of the optimisation investigations as well as the distribution test.

Due to this massive improvement in model quality, it was decided that the number of hidden neurons would be re-investigated with a model using a 135x135 pixel input shape. For this investigation, along with the 270 hidden neuron model, 68, 135, 540 and 1080 hidden neuron models were tested. This found that using 135 neurons produced a much greater model with a QPP of 79.7% and a QPP-40 of 91.3%. This is most likely because now the model is able to be more accurate, adding more neurons would increase the overfitting and therefore the benchmarks for the testing data using this model would be lower. This is because, while the model would be able to predict training data more accurately, it would not be able to generalise as well to unseen data, as the test and validation datasets are.

5 Final Model

5.1 Final Model Training and Preliminary Testing and Results

After these investigations into model optimisation were concluded, the finalised CNN was used to train the final model. This model consisted of 3 convolution layers, each taking an input of an image with a

size of 135x135 pixels. Each layer had 32, 64 and 128 filters respectively, with each filter being a size of 4x4, using a stride of 1 with equal padding. These three convolution layers used an activation function of ReLU to solve the vanishing gradient problem [23]. Between each convolution layer was a max-pooling layer using a pooling window of size 2x2 pixels with a stride of 2, ensuring no data was missed. The penultimate layer is a dense layer using 135 hidden neurons. This was then all used to predict a single value, representing smoothness for the input patch, with a value of between 0 and 1. This value could then be unnormalised and transformed back into its native distribution by applying an inverse box-cox transformation using the saved value of lambda.

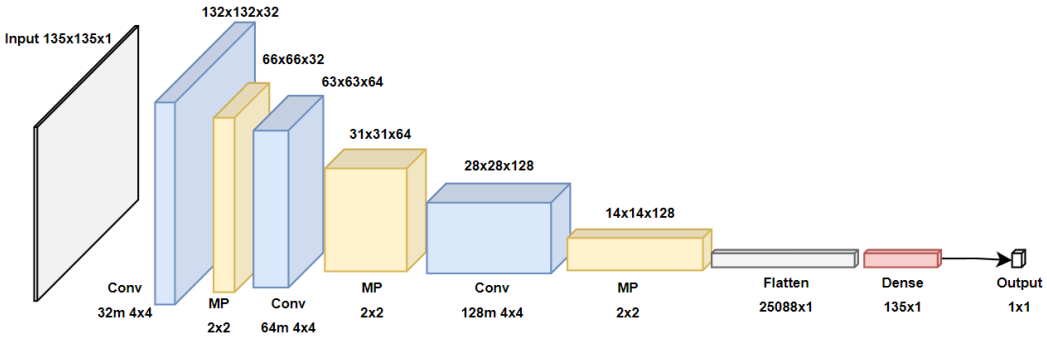

Figure 9: Graph depicting the architecture of the CNN used in this project

This model was then applied to the entire dataset using a batch size of 256, resulting in a total number of 3,552,751 parameters that were capable of being trained. The training was performed using an ADAM optimiser with a learning rate of 0.001 and a loss function of mean squared error. It was ran for 100 epochs of gradient descent, whilst using Keras' early stopping callback to ensure the model did not overfit to the training data. This is done by stopping the training of the model after the validation loss (MSE) begins to increase or plateau whilst the training loss (MSE) keeps decreasing. The approach tracks this and then stops the training once a certain number of epochs, known as the patience value, have elapsed since the validation loss has decreased. In this instance, a patience value of 20 was used. The produced model had a validation loss of 0.0089, the best seen throughout this project. This was tested on a large number of full images from all the datasets, being able to predict a safe LS for the vast majority of images, when judged by how smooth the surface appeared to the human eye. It's model benchmarks are shown below:

Benchmark	A-25m	A-50m	QPP	QPP-5	QPP-40
Accuracy	49.4%	73.8%	78.3%	30.4%	88.4%

5.2 Model Evaluation

The criteria used to evaluate the accuracy, and therefore success of this model, had 4 tiers of different gradients of slopes as previously described in the project specification. The accuracy of each of these was to be calculated using the difference in predicted range of a patch and the actual range used as the patch's label. This could be done as each patch represented an area of terrain that was around 400 by 400 metres in size. The longest distance in a straight line in this patch was easily calculated at around 565 metres. Using simple trigonometry it was found that a slope of 5° across a distance of 565 metres in the x-axis would mean that the distance in the y-axis was around 50 metres. This was repeated for all of the tiers, finding the estimated error range in distance for each (detailed in Table 1).

This allowed the model to use some of its already recorded measures such as A-25m and A-50m and also to easily record more. The tier 0 accuracy of the model (accuracy within 25m) is 49.4%. This is considerably better than the tier classification by almost 10%, meaning the model is almost 25% more

Tier	0	1	2	3
Slope Gradient Error Range	2.5°	5°	10°	15°
Estimated Distance Error Range	25 metres	50 metres	100 metres	150 metres
Classification Accuracy	40%	80%	90%	98%
Determined Accuracy	49.4%	73.8%	90.5%	96%

Table 1: Table showing the evaluation criteria tiers.

accurate in terms of predicting within a small error range. This means that the model is quite good at discerning between smaller differences in smoothness that even the human eye may not be able to recognise. The tier 1 accuracy of the model (accuracy within 50m) is 73.8%. This is just below the 80% classification for tier 1, meaning the model is slightly less accurate than was expected for the amount of training that it has undergone. However, this is not a major problem as it could be overcome with more training if it were to be used practically. The tier 2 accuracy of the model (accuracy within 100m) is 90.5%. This meets the tier 2 classification of 90%, meaning it can very accurately predict a patch's smoothness within an error range of 100m. This should be enough to easily find a suitably smooth and safe area of an image to designate as the LS. The tier 3 accuracy of the model (accuracy within 150m) is 96%. This falls short of the tier 3 classified accuracy of 98% by a small margin that would indicate that this model is more prone to a large error than is ideal, and therefore may need to be trained more if it were to be used as a primary HDA system as errors of this scale could prove dangerous.

6 System

6.1 Design

As previously mentioned in the project specification, a graphics-based system that used the model as a backend needed to be produced to enable a user to interact with the model in a seamless and simple manner. This system would exist more of a prototype and way to easily demonstrate the system as a small-scale prediction app rather than a high-end quality product. In reality if the model were to be practically used, a different, more specialised system would be produced to utilise the model for the exact purpose required. The system produced in this project would use a simple user interface (UI) with a singular window in which the user could click a button to upload an image. This image would then be resized and broken up into the same size patches that were used to train the model (that being 270x270 pixels). These patches would then be input into the model and it should output a predicted smoothness value for each. The system should then identify which patch has the lowest predicted smoothness value and produce an image highlighting which area on the original uploaded image this patch is from along with displaying the patch's predicted smoothness value.

The system itself will need to process these images so that differing varieties of images could be used with it, as is neccessary when aiming to test how well the model generalises to authentic Martian images. The system should therefore only accept image files to be uploaded, and these will then need to be handled in a variety of ways. First the images should be in a 1:1 ratio, if they are not, the images should be cropped, while maintaining the same image centre point. Next these images should be converted to grey scale, as the model was trained on images in this format. The model was also trained on image patches of size 270x270 pixels, with these being sized down to 135x135 pixels when being put into the model. As this sizing down is done dynamically, it will be possible to feed in patches already of the size of 135x135 pixels. This means that for any image less than 1080x1080 pixels, it won't need to be scaled up as dramatically, as it could be sized up to 1080x1080 pixels and then easily cropped into 64 patches of 135x135 pixels. At the same time if any images were uploaded into the system larger than this they could be re-scaled to 2160x2160 pixels and cropped accordingly into 270x270 pixel patches with the model's prediction ability remaining largely unaffected.

The model could then predict each patch within an image, calculating a predicted smoothness value for each. However, due to the normalisation and transformation carried out on the training data before the model was trained, this value will not be in it's native form (that being the range in the y-axis in metres). Inverse scaling and transformations will therefore need to be carried out on this value. In order to do this, the parameters and shaping used to scale and transform the training data will need to be used. These were exported after the pre-processing of the training data on Google Colab [25] and imported into this program. These parameters can be used to inverse the box-cox transformation and the min-max scaling, producing the predicted smoothness value in its correct form.

6.2 Methods used and System Implementation

For this simple system, the standard python module Tkinter [31] was used to create a basic single window system. The only features of this window were a button allowing the user to choose an image, and a text box where the predicted value would be displayed to the user. To display the original image with the patch highlighted, the OpenCV python module was used which allowed for an image to be displayed in a separate window. On this image a rectangle could be overlayed to highlight the position of the smoothest patch on the image, as predicted by the model. The produced system was able to be ran from a batch file and was fully launched within 15-20 seconds. The system runs smoothly, with the uploading and processing of the image and the actual predicting of the whole image only taking around 5 seconds. The window popping up showing the highlighted image can be closed and another image can be uploaded straight away without any issues.

7 Final System and Model Evaluation

7.1 Functionality Tests

Once this system was complete, unit tests written in Pytest [32] were executed on it to ensure it's functions and features work as expected in the design. This mainly involved handling any incompatible images and any errors that could occur as well as ensuring that the system was using the results directly from the model. These tests included ensuring the model could handle the standard sized images, that being those in the same format as the training dataset (resolution and bit depth), as well as images larger and smaller than this, to test the image scaling element of the system. Also, the tests assessed how the system handled unexpected varieties of images, including different file types like PNG instead of JPG and images with different bit depths. Other images tested were those that were not in the ratio of 1:1, which the system had to crop into this ratio, and an image from the lander-vision camera onboard the Mars 2020 mission, as this is what the model's conception was based on. Along with this, some testing was carried out that took a small handful of full images and passed them through the system, ensuring these all produced the expected results from the model. Since all these tests ran successfully, the system could next be used to evaluate what kind of images the model can predict better and those that it struggles on, as well as testing the model's ability to generalise to unseen and authentic images.

7.2 Analysing Image Prediction

The model was then used to test different kinds of terrain depicted in the simulated dataset. The final GUI system could also be used in conjunction with the model to provide visual context of what area the model is predicting correctly or incorrectly. This can help to more easily understand the models strengths and limitations. These tests were carried out using images from the testing dataset to ensure that there was no bias where the model had overfit to a certain feature prevalent in the training dataset. These investigations involved analysing how well the model performs on images with different varieties of terrain, prevalence of craters, lighting and camera angles from simulated swinging. Evaluation of the models accuracy used the inverse percentile ranking of where the predicted patch was ranked by it's actual

smoothness value. The lower this is, the closer the model is to predicting the actual smoothest patch within an image.

It was found that the model performs acceptably on images with flat terrain and excelled on images with very rough terrain. This shows that the model is able to distinguish between safe (even) and unsafe (rough) terrain more than distinguishing between very similar areas. This accomplishes one of the main goals of the project, that being ensuring the system recognises that rough terrain is a bigger hazard than smaller differences between patches. However, the model struggled slightly with relatively balanced terrain, the presumed cause of this is because the majority of the terrain is very similar, therefore any small error in predicting a patch's range value would have a large affect. It was also discovered that the number of craters had little impact on the accuracy of the model as long as the rest of the terrain was not completely flat. At this point, it was evident that the model was very good at choosing the smoothest point within a very rough image. This trend was only strengthened by the performance of the model on images with multiple large craters and rough terrain, where the predicted smoothest patch for the image shown in Figure 10 was within the 6th percentile of actual surface smoothness, indicating that the model managed to choose one of the safest areas in the whole image with high precision.

Figure 10: Image showing a generated image with rough terrain and several craters being accurately predicted by the model.

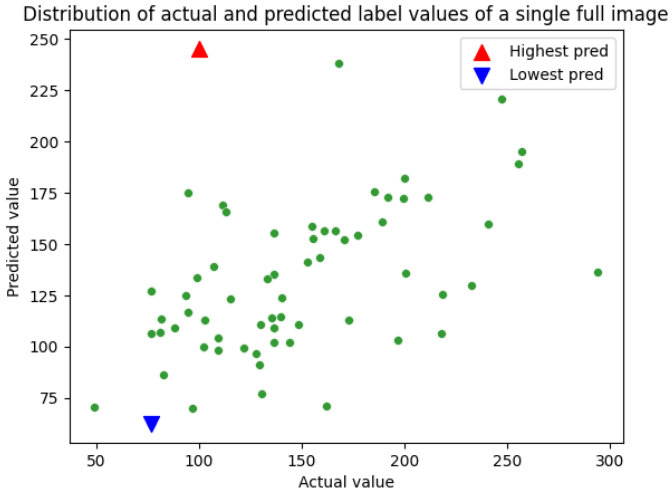

Figure 11: Graph showing distribution of actual and predicted label values for the image in Figure 10

Further analysis found that the model generalised well to images which were at notably different angles than the rest of the dataset. This was where there was considerable simulated swinging from the camera. This is very good because it means the model rather successfully generalises from the camera perspective. This suggests that the model will be able to work generally quite well even in situations that it has not been specifically trained for and should still be able to work to a considerable extent when the camera is not at a right angle to the ground.

Despite this, it was also found that the model does not work well when the sun is directly overhead at solar noon, or low in the sky near sunrise or sunset. This was generally expected to some degree because the shadows cast by the sun in these situations produce unrecognisable gradients that the model fails to be able to discern. This may be because the model relies on the length of shadows too heavily in order to predict smoothness. However, this is less likely as the direction of shadows was randomised during the dataset generation, to prevent the overfitting of images oriented a certain way. It could also be because areas in shadow are more difficult to see and therefore a lot of the information about that terrain is lost, resulting in the model not working as well. This exact situation is mirrored in many other HDA systems,

whereby they cannot operate when the sun is between 5° and 87° in elevation [5].

7.3 Evaluating Model Generalisability

To properly test the ability of the model to generalise, as well as evaluate how well the generated dataset was able to accurately depict Mars, actual images taken during the Mars 2020 mission's landing were used. NASA released a small selection of seven different images taken by the lander-vision camera [12]. These images were taken within a period of 71 seconds, starting from when the heat shield was jettisoned from the craft, to moments before landing [12]. Two of these images were taken within seconds of landing and therefore cannot be accurately used by the model because these were taken from an altitude of around 100-200 metres [12], much closer than the 2.5km used to train the dataset. This leaves only 5 images that can be accurately used to test on the model and with no terrain data able to be utilised in conjunction with these images it will be hard to quantify exactly how accurate the model is at predicting unseen authentic images from the model's intended operating conditions.

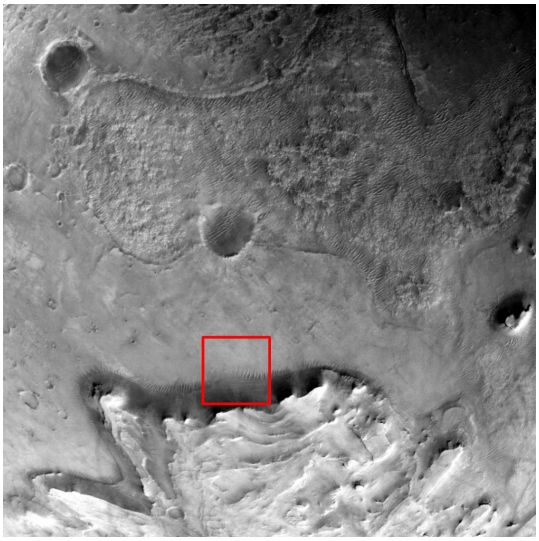

Figure 12: Image showing an authentic image taken by the lander-vision camera on-board the Mars 2020 mission [1] being predicted by the model.

Nonetheless, these 5 images were tested on the model with no preprocessing done outside of the final system itself. In general, the chosen smoothest patch in each image avoided most of the obvious hazards such as craters and cliffs. The model mostly chose a patch in which a large area seemed very flat and generally seemed a safe prospective LS. For three of these images, there was a slight overexposure caused by the sun's reflection, producing a darker shadow over a corner of the image, which the model chose as the safest LS each time. This was most probably because the dataset did not account for this possibility and therefore the model was limited to choosing somewhere that wasn't overexposed, as the model more closely recognised these areas. When the model, for one of these images, wrongly predicted an LS in an obvious rough patch, the model estimated this to be much smoother than the others that seemed to be a more safe LS, with the model outputting a value of 28.36 compared to an average of around 65 for the others. This adds to the evidence that the model cannot generalise to overexposure and bright lighting that distorts shadows from the patterns it expects.

For two images, the model predicted the safe LS to be in the middle of a small section of dunes. This is most likely another shortcoming of the generated dataset, in which due to a software limitation, dunes could not be generated, meaning the model didn't have any training of dune-like terrain, resulting in it failing to generalise to this image well. Finally, the model did predict a few promising landing sites (like that shown in Figure 12), with the large majority of the patch being located on seemingly very smooth ground. However, it did seem to predict two landing spots in the same location (the LS shown in Figure 12), which was at the bottom of a cliff. This most likely was due to the training datasets not including many sheer cliffs like this, as well as not penalising an areas smoothness value when it was near a large hazard (like a cliff) which could prove problematic for landing. In conclusion, while this small set of images wasn't large or consistent enough to draw any accurate conclusions, the model seemed to be able to generalise moderately well to these unseen authentic images.

8 Conclusion

8.1 Possible Further Improvements

Considering what has been learnt over the course of this project, there are a number of things that were initially not considered, overlooked or decided against due to time and resource limitations. A major part of the project was the dataset, for the most part it was of quite a high quality, yet it did in some respect lack complete similarity with actual Mars terrain and images taken of it. This includes the difference in cliffs, sand dunes and reflections as found when evaluating the model on authentic lander-vision photos. If the project was to be improved, focusing on developing these features would make the dataset more representative of Mars terrain and increase the model's generalisability.

Also, to improve the precision of the produced GUI system, an accompanying model to classify craters within the same image could be developed, increasing safety by ensuring the model does not select an LS near or in hazardous craters. The produced model could have also had a smaller variant, which worked in a similar way to itself, to break down a patch even further after it has been predicted as the safest LS. This would improve the precision of the overall system even more and specify the exact safest spot for a landing, thereby increasing safety. The system could also have highlighted numerous of the safest landing sites instead of just one. This could then be utilised by the craft's decision making system, that could take into account levels of safety and current trajectory, to work out which prospective LS to designate as the most appropriate for any given situation.

8.2 Final Conclusion

In closing, the model produced met alot of the success criteria specified in the project specification, meaning the model is considerably accurate and can consistently predict the smoothness of an input patch within a relatively small error range ($\sim$50% within an error range of 25m). The model is also very good when being used for its intended purpose, calculating the safest LS in a given image. With the model able to very precisely place the prospective LS within the safest 5% of patches for a good 30% of images within the test dataset, with the vast majority of LS being predicted in the first quartile. This accuracy improves even more when the model is being used to predict images containing very rough terrain, where the vast majority of the time the model successfully predicts a very safe LS, with them often being within the safest 10% of patches. This is extraordinarily helpful as the motivation behind creating HDA systems is to enable autonomous crafts to land in rugged and generally unsafe terrain. This model's greatest strength accomplishes this perfectly, whilst also being able to work well-enough in other more safe terrain.

To conclude, the model produced is of a sufficient standard which proves that a deep learning regression-based neural network can be used to predict the smoothness and therefore safety of a prospective LS. If this same model architecture was to be trained using real Mars imagery taken by autonomous landing vehicles, along with high-quality terrain data to properly train the model, it should generally achieve equal or better accuracy measurements for its intended use of real images from autonomous Mars landing craft. However, only large-scale aerospace companies and national space programs have access to the resources and dataset required to properly train this model, and therefore this could not be done within this project. If an organisation with these resources decided to do this however, it is feasible that a variant of this system could potentially be used in a backup HDA capacity onboard a future autonomous mission to Mars.

9 Bibliography

References

[1] mars.nasa.gov, "Mission Overview - NASA." https://mars.nasa.gov/mars2020/mission/overview/. (accessed Nov. 14, 2023).

[2] "Space Communications: 7 Things You Need to Know - NASA." https://www.nasa.gov/missions/tech-demonstration/space-communications-7-things-you-need-to-know/, Oct. 2020. Section: Goddard Space Flight Center. (accessed Nov. 13, 2023).

[3] C. D. Epp and T. B. Smith, "Autonomous Precision Landing and Hazard Detection and Avoidance Technology (ALHAT)," in *2007 IEEE Aerospace Conference*, pp. 1–7, Mar. 2007. ISSN: 1095-323X.

[4] A. E. Johnson and J. F. Montgomery, "Overview of Terrain Relative Navigation Approaches for Precise Lunar Landing," in *2008 IEEE Aerospace Conference*, pp. 1–10, Mar. 2008. ISSN: 1095-323X.

[5] P. Rogata, E. Di Sotto, F. Câmara, A. Caramagno, J. Rebordão, B. Correia, P. Duarte, and S. Mancuso, "Design and performance assessment of hazard avoidance techniques for vision-based landing," *Acta Astronautica*, vol. 61, pp. 63–77, June 2007. Publisher: Pergamon.

[6] mars.nasa.gov, "Rover Cameras - NASA." https://mars.nasa.gov/mars2020/spacecraft/rover/cameras/. (accessed Nov. 14, 2023).

[7] "Entry, Descent and Landing (EDL) - NASA." https://mars.nasa.gov/mars2020/timeline/landing/entry-descent-landing/. (accessed Nov. 14, 2023).

[8] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks." http://arxiv.org/abs/1511.08458, Dec. 2015. arXiv:1511.08458 [cs].

[9] "Planetside Software." https://planetside.co.uk/, Nov. 2023. (accessed Apr. 17, 2024).

[10] A. Giusti, M. Dotta, U. Maradia, M. Boccadoro, L. M. Gambardella, and A. Nasciuti, "Image-based Measurement of Material Roughness using Machine Learning Techniques," *Procedia CIRP*, vol. 95, pp. 377–382, 2020.

[11] R. Gascoigne-Jones, "Literature Review - Evaluating the use of Deep Learning Models for Hazard Detection Systems in Autonomous Spacecraft," Nov. 2023. University of Exeter - ECM3401.

[12] mars.nasa.gov, "Images from the Mars Perseverance Rover - NASA." https://mars.nasa.gov/mars2020/multimedia/raw-images/index.cfm?urlpath=ELM_0000_0666952847_800ECM_N0000023LVS_04000_0000LUJ&mission=mars2020. (accessed Apr. 28, 2024).

[13] "Welcome to Python.org." https://www.python.org/, Apr. 2024. (accessed Apr. 18, 2024).

[14] "High Level Python API — Terragen RPC 0.9.3 documentation." https://planetside.co.uk/docs/terragen-rpc/high-py-api.html. (accessed Apr. 17, 2024).

[15] H. Y. McSween Jr., "The rocks of Mars, from far and near," *Meteoritics & Planetary Science*, vol. 37, no. 1, pp. 7–25, 2002. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1945-5100.2002.tb00793.x.

[16] P. R. Christensen, "The spatial distribution of rocks on mars," *Icarus*, vol. 68, pp. 217–238, Nov. 1986.

[17] J. B. Garvin, S. E. H. Sakimoto, and J. J. Frawley, "CRATERS ON MARS: GLOBAL GEOMETRIC PROPERTIES FROM GRIDDED MOLA," 2003.

[18] NASA, "AT A GLANCE - NASA Mars Exploration." https://mars.nasa.gov/education/modules/GS/GS38-49.pdf. (accessed Apr. 17, 2024).

[19] D. Coffeypublished, "What color is the sunset on other planets?." https://www.livescience.com/what-color-are-other-planets-sunsets.html, July 2020. (accessed Apr. 17, 2024).

[20] "OpenCV." https://opencv.org/. (accessed Apr. 18, 2024).

[21] "numpy.std — NumPy v1.26 Manual." https://numpy.org/doc/stable/reference/generated/numpy.std.html. (accessed Apr. 18, 2024).

[22] Y. Sari, Y. F. Arifin, Novitasari, and M. R. Faisal, "The Effect of Batch Size and Epoch on Performance of ShuffleNet-CNN Architecture for Vegetation Density Classification," in *7th International Conference on Sustainable Information Engineering and Technology 2022*, (Malang Indonesia), pp. 39–46, ACM, Nov. 2022.

[23] Y. Hu, A. Huber, J. Anumula, and S.-C. Liu, "Overcoming the vanishing gradient problem in plain recurrent networks," July 2019. arXiv:1801.06105 [cs].

[24] "Project Jupyter." https://jupyter.org. (accessed Apr. 23 2024).

[25] "colab.google." http://0.0.0.0:8080/. (accessed Apr. 23 2024).

[26] "SciPy -." https://scipy.org/. (accessed Apr. 23 2024).

[27] M. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, p. 3021, Apr. 2021.

[28] "Matplotlib — Visualization with Python." https://matplotlib.org/. (accessed Apr. 23 2024).

[29] A. C. Atkinson, M. Riani, and A. Corbellini, "The Box–Cox Transformation: Review and Extensions," *Statistical Science*, vol. 36, pp. 239–255, May 2021. Publisher: Institute of Mathematical Statistics.

[30] T. O. Hodson, "Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not," *Geoscientific Model Development*, vol. 15, pp. 5481–5487, July 2022. Publisher: Copernicus GmbH.

[31] "tkinter — Python interface to Tcl/Tk." https://docs.python.org/3/library/tkinter.html. (accessed Apr. 28, 2024).

[32] "pytest: helps you write better programs — pytest documentation." https://docs.pytest.org/en/8.2.x/. (accessed Apr. 29 2024).