Exploring Automated Lunar Crater Detection Using Gravity and Optical Data Modalities with Lightweight Segmentation Models

Thesis

Presented in Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By
Hector J. Cotto Ortiz, B.S.
Graduate Program in Geodetic Sciences

The Ohio State University
2025

Thesis Committee:
Demian Gomez, Adviser
Alper Yilmaz, Co-Adviser
Jun-Yi Guo

ABSTRACT

In geodetic and planetary sciences, the study and characterization of impact craters provide invaluable information on the formation, evolution and aging of solid bodies in the solar system. High-resolution optical data is often used in this study, and gravity data remains largely unexplored. This work explores the potential of utilizing gravity data in crater segmentation methods.

In this work we use Datasets containing two data modalities: (1) A GRAIL derived Gravity Disturbance (degree and order 660), and (2) LRO derived Lunar Orthorectified Mosaic. We create a third dataset by combining both modalities using early fusion methods.

To address computational and data limitations we use lightweight models SegFormer and MobileUNETR. SegFormer is pretrained on the ADE20K dataset, while MobileUNETR leverages the International Skin Imaging Collaboration (ISIC) datasets. A Pytorch Lightning framework is implemented, while fine-tuning the models across 50 to 100 different experiments using a random search hyperparameter optimization method.

Validation Losses for each dataset reveal that Mobile UNETR achieved .2562 (Gravity Disturbance), .3755 (Optical) and .2517 (Data Fusion). SegFormer achieved .2820 (Gravity Disturbance), .3614 (Optical) and .3206 (Data Fusion). Best Overall Pixel Accuracy was 80.91%, achieved by MobileUNETR (Data Fusion). Results indicate that it's possible to leverage Gravity Disturbance data in crater semantic segmentation. Results from Data Fusion show marginal improvement, indicating the need for further study to optimize its potential.

GitHub Repository: Guachurro/MobileUNETR-SegFormer-Crater-Detection-Project-with-Gravity-Disturbance-and-Optical-Data

ACKNOWLEDGEMENTS

I'd like to start by thanking the National Geospatial-Intelligence Agency for providing me with the opportunity of a higher education, and the financial means to achieve it. My journey throughout this process has been aided, not just by the agency, but the multiple colleagues I've made along the way, and made this process worthwhile. Of note, I'd like to thank Dr. Trevor Garner and Dr. Nikki Markiel for listening to my idea, providing valuable feedback and offering me their support despite their busy schedules. Thanks also to Angela Anderson for so kindly offering her linguistic skills in revising my work. In addition, I'd like to thank my friend and colleague, Zachary Nixon, for offering his assistance in understanding several concepts of Machine Learning, and his invaluable assistance in several code-related debugging instances. Seriously.

I am grateful to Dr. C.K. Shum, who operated as my advisor for the better part of this 4-year journey. His willingness to assist me in dire needs with several, sometimes last-minute, administrative processes helped me start each semester with boots on the ground. In addition, his early guidance in helping me refine my ideas. His support was crucial to my development as a student, and without it I admit I would have felt very lost.

My special thanks to Dr. Jun-Yi Guo and Dr. Demian Gomez, my now main-advisor, who taught almost every course I ever took in OSU. It's thanks to their efforts that I was able to find a place to learn and develop myself as a student. I really appreciated their willingness to sympathize with my situation, being an online-only student, and without their interest in my education I wouldn't have made it this far. In addition, I'd like to thank Dr. Alper Yilmaz, who co-advised with Dr. Gomez. Both of you stepping up when I needed it was invaluable, and I appreciate that you've been understanding with me throughout this process.

I am especially thankful to all the friends and colleagues I made along the way. Corry Schaffer, John Maureais, Matt Hollitzer, Erin Knese, Max Wooten, Brian Bollin and Scott Ripperda were my fellow online-students class partners with whom I had the privilege of working with throughout my classes. I'd also like to thank Patrick Smith and Ying Zuo, who at one point or another, operated as the Teaching Assistants in my classes and never refused to help me when I reached out to them. While my interactions with other students was short

lived, I am thankful to Melody Pan, Kevin Wang, Yu Zhao, and Thunendran Periyandi, my experiences with you all were short-lived, but I hope we continue interacting in the future. I'd also like to thank Angeletha Rogers and Brooke Felts, who offered their assistance (And charming personalities) in times of administrative need.

Finally, I'd like to thank all my friends and family. Their words of encouragement in my several times of need really filled me with the willpower I required to finish this work. To my partner, Jennifer Aponte Rivera, for reminding me that 'I can do it' and being somebody, I could fall back on without question. To my childhood friends Julian, Michael and Victor for being there for me whenever I needed them. Finally, to my mother Juanita Ortiz Vega and late father Hector R. Cotto Cotto for raising me with the tools to get this far. This work is dedicated to you both,

VITA

2012 ..... Escuela Superior Miguel Melendez Muñoz

2017 ..... B.S. Mathematics & Science, University of Puerto Rico, Cayey

2018 to Present..... Photogrammetrist, National Geospatial-Intelligence Agency

Publications

Fields of Study:

Major Field: Geodetic Sciences

Table of Contents

List of Figures

List of Tables

Prelude: Thesis Introduction and Overview

Introduction and Objectives

Impact craters are a very common structure found in solid bodies that form because of a high-energy kinetic impact (CLRN, n.d.). While Impact craters are often associated with circular structures, it's well-documented that their shape is only approximately spherical (Robbins, 2019). There is a variety of features that can identify an impact crater to the point where two similarly sized impact craters can differ substantially, some of these are: The impactor's morphology, the crater's age, the underlying structure of the interior crust and the level of exposure to space hazards (Chandnani & Herrick, 2022; Kaku et al., 2017; Pieters et al., 2000). Due to Earth's highly dynamic nature and existence of life, this process has an added complexity but is not invalidated and retains relevancy [98]. On bodies like the Moon, however, impact craters can reveal the geological and evolutionary planetary history through crater morphology studies and count analysis, and even aid in autonomous and feature-based navigation (Wu et al., 2020; M. Xie et al., 2017; Yu et al., 2014; Zhao et al., 2023).

The actual process of characterizing craters has historically been a very dense, and challenging task. It generally relies on data modalities that facilitate crater feature extraction such as Optical (Robbins, 2019), Elevation (M. Chen et al., 2018; Zhao et al., 2023) and Radar based data (Greeley et al., 1987). The manual study of craters can involve several challenging aspects related to the evolution of crater formations, their degradation and eventual disappearance due to additional impacts. In addition to these, the increasing amount of high-resolution imagery available for multiple planetary bodies greatly complicates this task. This is something that is documented well by Robbins, who undertook such a manual labor in the creation of his lunar impact crater database(Robbins, 2019). Automatic crater detection algorithms have existed for a long time, and for a long time they depended on hard coded, rule-based algorithms like Hough transforms and template matching (Galloway et al., 2014; Saheba et al., 2016). These have been added upon by the advent of Machine Learning algorithms such as Convolutional Neural Networks (CNNs) and

Transformer algorithms, which have demonstrated the potential to accurately classify and detect crater formations, though this an actively researched topic (Tewari et al., 2023).

AI/ML is generally associated with large quantities of data and computational resources to derive appreciable results. In many cases this limitation can lead to the inhibition of would-be researchers from leveraging their benefits. Fortunately, this is a recognized limitation, and studies have strived to deliver lightweight architectures, intended to require less computational resources, while still delivering comparable results. This is an important aspect of our work, which aims to leverage gravity data, something that is not often used in the automatic detection of craters, even though its use in gravity field analysis and detection of subsurface features is common (Huang et al., 2014). Part of the reason this is not so is simply because ethe availability of gravity data on extraterrestrial bodies is extremely limited, and this conflicts with the requirements for ML.

To this end, this work intends to leverage two types of Deep Learning (DL) algorithms in the semantic segmentation (Pixel-wise classification) of lunar impact craters. The models MobileUNETR and SegFormer are both Transformer-based lightweight models that promise computationally efficient segmentation(Perera et al., 2024; E. Xie et al., 2021). Each model will feature fine-tuned on International Skin Imaging Collaboration (ISIC), and ADE20K datasets respectively. Regarding data, we use a lunar gravity disturbance product obtained from NASA, and an orthorectified lunar optical mosaic of the lunar surface. By themselves, each data modalities form the basis of two datasets, and a third dataset combines them into a single, multi-modal feature. This work's main objectives are enumerated below:

(1) Understand the formation and influence of crater formations on planetary bodies.
(2) Understand the processes associated with Machine Learning feature extraction.
(3) Describe the process involved in establishing a framework capable of leveraging Machine Learning algorithms its application of Geophysical data.
(4) Determine whether gravity data, or some fusion of it, would lead to appreciable results in crater segmentation.

Thesis Structure

This work is organized in 4 main parts, each of which correlates with the thesis objectives:

Part I provides an overview on crater formation and evolution mechanisms driven by space weathering, subsequent cratering and seismic activity. It additionally provides a small introduction to gravity theory and its relationship to impact craters. It finishes by trying together the importance of both concepts in planetary sciences.

Part II provides an overview of Machine Learning concepts and relates their documented use in crater detection studies. It provides a surface level understanding of how CNNs and Transformer-based models extract information from imagery, which is important to understand inherent limitations to models and techniques. It follows with an introduction of SegFormer and MobileUNETR, having given enough of a background to understand the relevance of their innovations. It finishes by introducing the concept of Data fusion.

Part III Details the process of acquiring and processing of our data in creating labelled datasets for semantic segmentation. It additional provides an explanation of the framework utilized in establishing the models for training, validation and testing.

Part IV focuses on analyzing and visualizing several aspects of our results. In contrasts each of the models' performance with respect to each dataset, and additionally compares their overall performance across all datasets. It details several limitations and problems encountered in the progress of this work and provides suggestions to mitigate them in future work.

Part 1 Impact Craters, Gravity Data and Their Relationship in Planetary Sciences.

1. Background: Lunar Crater Formations

1.1. Types of Crater Formations

Craters are the most common feature of the lunar surface. It was in 1610 when Galileo first recognized these features, describing them only as “circular spots”. This was despite recognizing their raised rims and central peaks, for at the time, the description was sufficient. In the centuries that followed this opinion on how to describe these craters, and even what craters were, evolved. Advances in planetary science and observation technology have allowed us to glimpse ever further into the lunar surface, giving us orbital imagery with resolutions as high as half-meter pixel (Robinson et al., 2010). With this level of resolution, and even before, we’ve figured out that not all crater formations on the moon are created equal. Given a study with a particular focus on studying Impact Craters, it becomes equally important to study what is not a crater. Ahead we briefly discuss on the three types of crater formations existing on the lunar surface: Impact Craters, Lunar Pits and Collapse Pits.

1.2. Impact Craters

Impact crater formation is, in fact, the most prevalent origin of crater formations on the lunar surface. These for formed when there is a collision between the lunar surface and meteoroids, comets, asteroids, or some other such object at relatively high speeds. These high-energy impacts release large amounts of energy, creating circular depressions with varying morphologies. The nature of the depth of these depressions and their subsequent morphology largely depends on the speed, size, and shape of the object. One such example is one of the moon’s more recent impact craters, formed when the HAKUTO-R Lunar Lander crashed on the lunar surface in 2023, an event that verifiably altered the landing surface to some degree (Spreyerer, 2023).

Currently, there is no standard on the classification of impact craters, but there are loosely defined rules based on the relationship between crater features and size. For the purposes of this section, we can choose to define impact craters in a basic way as the following: **Simple craters** and complex craters. Simple craters tend to be of the smaller variety, up to 15km in diameter. They have the standard raised rim and have very well-defined bowl-shaped depressions. On the other hand, **Complex craters** tend to be formed by larger impacts, which generally means a larger impactor. Like simple craters you will find ejecta in their surroundings and have raised rims. In contrast, these craters are characterized by a feature known as a central uplift peak, a raised feature found at the center of these craters. Their final characteristics lie in a relatively flat floor, and terraced cater walls. These identifying features are a direct result of the rebound and collapse of the lunar crust post-impact (Collins, 2014). The following figure presents a transect elevation profile of a simple crater (~ 13km diameter) and a complex crater (~55k m diameter), showcasing the complex craters' central uplift peak.

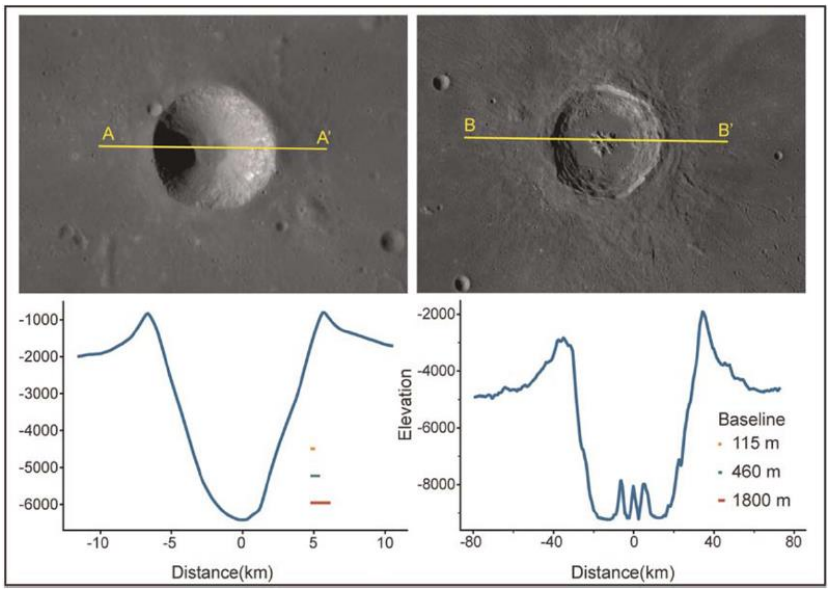

Figure 1. Cross-Sectional elevation profile of a Simple Crater (Left) and a Complex Crater (Right)

Image sourced from (Juntao Wang et al., 2020).

The formation of an impact crater, regardless of size, can be described mechanically. It is well known that the impact creation process generates enormous amounts of kinetic energy and operates similarly at all scales. The velocities of meteoroids, asteroids or comets that impact the surface tend to have velocities of at least 2.4km/s, which is faster than the lunar speed of sound (Wilhem, 1987). This gives the reader an idea of the amount of energy involved in the creation of an impact crater, which is important as we consider their three main stages of impact crater development. **Contact and compression** is the initial energy transfer between the impactor and the surface (Chandnani & Herrick, 2022). This results in shockwaves that propagate through the material, often resulting in a crater's ejecta (Chandnani & Herrick, 2022). Following this is what's known as the **excavation stage**, where the shockwaves create pressure gradients that lead the formation of a transient; and generally, parabolic cavity as material flows out and becomes compressed. Finally, in the **modification stage** the material and impactor begin to settle into a stable crater (CLRN, n.d.). This involves the reshaping of the previously formed cavity as the rims settle. This can potentially lead to ejecta material flowing back into the crater as the rims collapse, something that is more common in larger craters (Collins, 2014). This stage is also where central uplifts can form as the compressed material rebounds, which is more likely in larger craters (Ullrich, 1976).

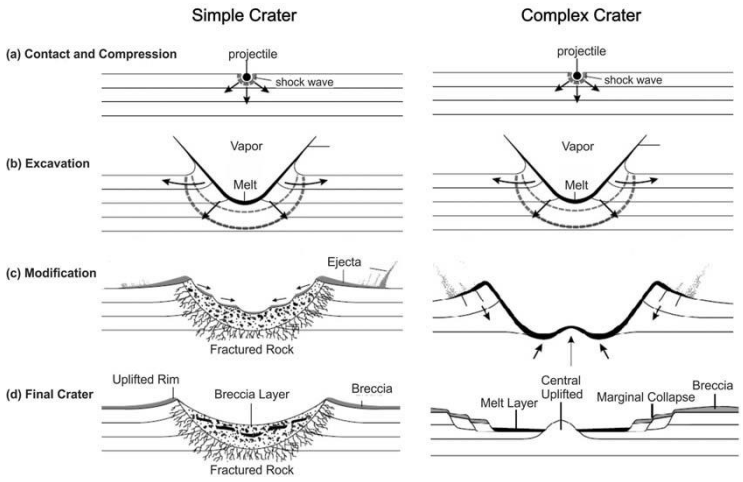

Figure 2. Formation Process of Simple and Complex Impact Craters.

Image sourced from (CLRN, n.d.).

1.3. Lunar Pits and Collapse Pits

"The exogenic-endogenic controversy about the origin of craters probably occupies more early literature than did any other lunar topic." Is a direct quote from Wilhelm's 1987 'The Geologic History of the Moon'(Wilhem, 1987). It alludes to a long-standing debate about the origin of lunar craters, eventually settling the matter on most being formed by the exogenic process described in the previous subsection. That said, crater formations of endogenic origin are prevalent in the lunar environment. Their introduction in the context of a study on impact craters is relevant and necessary. To understand the formation of lunar pits and collapse pits (Skylights, a particular type of lunar pit) , we briefly introduce the concept of lava tubes (Greeley, 1971; Haruyama et al., 2009; R. V. Wagner & Robinson, 2022).

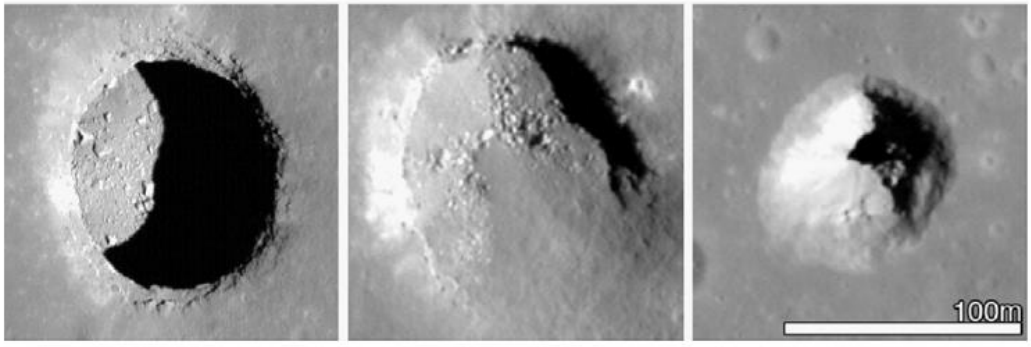

Figure 3. Equal-scale images of lunar pits.
From left to right: Mare Tranquillitatis pit (image M126710873R), Central Mare Fecunditatis pit (image M1105602888R), Southwest Mare Fecunditatis pit (imageM1105645870L). Image sourced from (Robert V. Wagner & Robinson, 2014).

Lunar volcanic activity has been a subject of much study in the effort to understand lunar geology and formation mechanisms. When lava that flows beneath a solid crust eventually drains away, the hollow cavity left behind is referred to as a lava tube [11](Robert V. Wagner & Robinson, 2014). This is a lunar feature that was first theorized in the 1960s, contrasting it with the similar Earth-bound phenomena (Greeley, 1971; Robert V. Wagner & Robinson, 2014). Lunar pits are

small depressions (~10-300m diameter), sometimes characterized by vertical walls and overhangs. They can be found atop small hills, or often connect to lava tubes and similar geological structures. A lunar pit can be referred to as a lava pit when it's believed to connect into magma chambers, or lava tubes. They can also be an opening into large voids caused by geological crustal movement. On the other hand, collapse pits are commonly characterized for being an opening, much like a lunar pit, but specifically into a subsurface like a lava tube, or magma chamber (R. V. Wagner & Robinson, 2022). Overall, there is a good amount of overlap between lava pits and collapse pits, and differentiating the two is not necessarily an easy task. In contrast with impact craters, lava pits and collapse pits tend to be small, and shallow. In addition, they lack the clear indicators of an impactor such as a crater rim, and ejecta blanket. (Horvath et al., 2022; Kaku et al., 2017).

1.4. Sub-Section Summary: Crater Formations

In summary, there are millions of crater formations on the lunar surface that can either be an impact crater or appear to be one. Naturally, these differences in classification can become more evident on the smaller of crater formations. As it was discussed, collapse pits and lava pits tend to be small, on the order of tens or hundreds of meters. This is important when considering how an impact crater should be characterized. In short, impact craters tend to be recognized by their ejecta blanket, raised rim and, for larger craters, central peak or collapsed walls. On the other hand, lava pits and collapse pits lack the distinctive ejecta and rim patterns. Even with these distinctions, amongst expert researchers there can be as much as a 40% variation in the identification of what is a crater when it involves human interaction (Robbins, 2019). As Robbins put it, studies are in general agreement on the quantity of craters larger than 30km, but below that size there are significant differences in quantities.

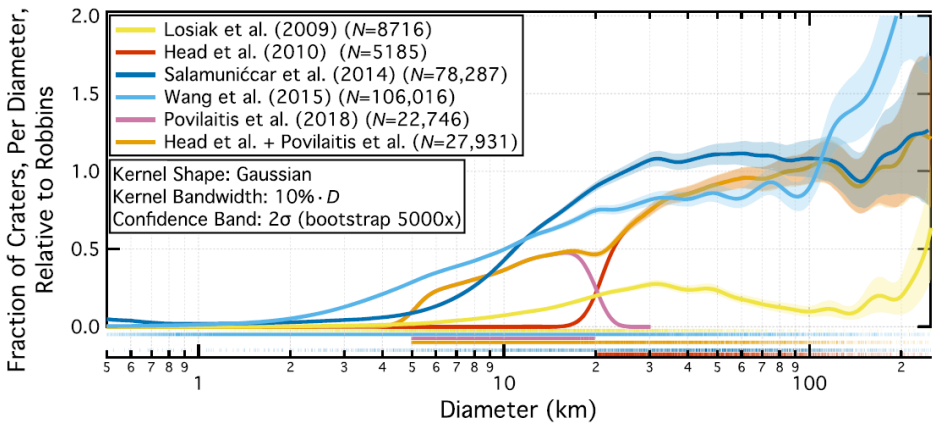

Figure 4. Figure 4. Robbins' comparison of fractional crater distribution by size. Values >1 indicate a higher crater count than Robbins, values <1 indicate a lower crater count than Robbins. Image sourced from (Robbins, 2019).

1.5. Impact Crater Morphologies and their difficulties in research

The study of impact craters on the lunar surface, and most other planetary bodies, can provide information about planetary age and formation (Wilhem, 1987). It can also help study the occurrence of geological processes, the geology of a given area, or even simpler aspects like topographic roughness or temperature changes (Nakamura, 1982). The chosen aspects to study a crater can fit the narrative of a study, but across all studies the one thing there is in common is the difficulty in characterizing craters in a standardized way. The reason for this is that craters will change and deteriorate, the time at which this happens depends on the mechanism involved. The main forces driving said changes are space weathering, micro-cratering (Or subsequent impacts) and lunar crustal contractions (Pieters et al., 2000; M. Xie et al., 2017; Zhao et al., 2023).

2. On the importance of crater characterization

Since the discovery of craters on the lunar surface, the need to characterize and classify them has been crucial. For the eventual study of any planetary body, the initial task of characterizing its surface and features is a very important first step. For the Moon this was of particular importance leading to the Apollo-era of lunar exploration. The identification of crater formations here was not just beneficial from a research standpoint, but also from the perspective of the safety of navigation of would-be Moon-landers. History is now repeating itself as NASA continues preparations for a new era of lunar exploration with the Artemis Program.

There have been many studies that have improved on the study of lunar craters by proposing ways of characterizing them (M. Chen et al., 2017, 2018; Juntao Wang et al., 2020). The decision to characterize lunar craters often boils down to what the intention to characterize them is. In the 1>km lunar crater impact database by Robbins, et al. this characterization was intended to create a comprehensive database for the further study of the lunar surface (Robbins, 2019). Like him, many others did similarly with varying focus on crater sizes such as 5>km (Barlow et al., 2012) or even 20>km (Head et al., 2010). For these purposes, characterizing crater based on diameter, approximate age, type (Simple or complex), depth, approximate elliptical and circular shape, and approximate center often make sense. Their intention is to provide a source of information for additional researchers to take advantage of. From this perspective, this type of crater morphology studies can be very informative and based on the 40% difference in the determination of craters amongst researchers, very thought-provoking (Robbins, 2019).

To provide a different, we can look at Wang's study on the degradation of crater morphology (Juntao Wang et al., 2020). It's a study that focuses on studying the morphologies with craters, but in a much different light. This study goal focused on the study of crater 'roughness', abundance of rocks and the soil temperatures of craters. They achieve this by studying the different characteristics of impact craters separately, rather than grouping them together to study the individual changes of a crater's morphology overtime, establishing patterns. One such finding includes that younger craters tend to have a larger amount of roughness and abundance of rock. Though they note that "the properties studied are not exact proxies of age".

Another study by Chen, et al. takes predetermined crater characterizations together with a Crater Detection Algorithm (M. Chen et al., 2018). This study differs in that it creates a characterization method for the purposes of the automatic detection of a computer algorithm. The study applied a Crater Detection Algorithm (CDA) to mathematically detect potential craters by defining a set of morphologies. This study saw good results with an average detection accuracy of 86.55%, it recognized that the described morphologies are not enough to replace the unique characteristics of each crater. Though it's very important to learn how to properly structure crater morphologies, this last point brings a very important reminder. The lunar surface is subject to environmental changes, which can lead to difficulties in recognizing or classifying craters in accordance with some established nomenclature. It could even impact the detection or classification of impact craters, or their age (M. Xie et al., 2017). Failing to take these considerations into account can negatively impact future studies. Any study, particularly studies with automated detection or characterizations of craters, would benefit from considering the driving forces behind crater degradation and how that may affect their chosen methodology (Zhao et al., 2023).

2.1. Sub-Section Summary: Crater Characterization and its limitations.

In summary, the characterization of impact craters ties nicely with the study of how impact craters degrade overtime. We can see the significance of crater characterization in the study done by Wang, et al. which hopes to better identify and characterize craters based on various aspects of their morphology and how they've degraded over time (Juntao Wang et al., 2020). We similarly have studies that recognize the effects of crater degradation on the task of crater segmentation. On the other hand, we have studies that hope to apply crater degradation models to improve their detection rate [25]. All of this to say that the task of crater detection is a difficult one, faced with many different challenges, particularly depending on the type of detection and classification that you wish to achieve. Clearly, the cut and clear delineation of craters has its challenges, though much less so than that of identifying the causes, or ages of craters. It's the hope that at this point, the reader has a thorough understanding of the various mechanisms involved in the formation, and eventual evolution of impact craters on the lunar surface. Together, these aspects of the evolution of the lunar surface can create a unique challenge for anybody wishing to embark in this task. The

Moon is, by no means, an unchanging landscape, as it's filled with many complexities in its inner mechanisms.

2.2. Limitations: Space Weathering, micro cratering and the lunar crust contraction.

The concept of crater degradation is not new. Since early studies, the irregularities of the surface on the moon were well noted (Heiken et al., 1991; Pieters et al., 2000). Take, for example, when Galileo discovered craters and mountains on the moon, instead of finding an expected spherical structure. The lunar surface goes through many changes other than the obvious impact cratering that dominates its landscape. The purpose of this subsection is to discuss in some detail some of those forces and how they could impact studies. The phenomenon driving change in the lunar surface can be termed as 'Space Weathering'. This refers to the type of weathering that occurs to objects in the space environment (Pieters et al., 2000). This is particularly true for objects that do not have an atmosphere. This includes the collision of cosmic rays, implantation of solar wind particles, and naturally, the bombardment of meteorites, a portion of which are appropriately dubbed *micrometeorites* (M. Xie et al., 2017). These effects are all exogenic in nature, and each one can affect the Moon in a variety of ways.

If we consider solar wind particles, we can take reference to a study by Pieters, et al [16]. They studied the process by which nanophase reduced iron(Fe^0) became available on the surface of asteroids by studying this process thoroughly with lunar samples. The study found that in larger concentrations, Fe^0 reduced the reflectance of a silicate matrix. Notably, the lunar regolith (Loose soil, rock and other minerals) has a high composition of agglutinates, which consist of soil grains bonded by glass (Heiken et al., 1991). Said glass is created by the micro-cratering of the surface of the regolith through a process known as impact vapor fractionation-deposition (Pieters et al., 2000). This presence of Fe^0 in this compound can darken and modify the signal from the soil, creating difficulties in the accurate mapping through remote sensing. In addition to participating in the darkening of lunar soil, micro-cratering; or subsequent impacts of a greater magnitude, can alter previous impact craters by rounding their rims, shifting the regolith, altering the ejecta

patterns, or even replacing them entirely(Li, 2013) . This is a prevalent source of degradation due to the Moon's lack of atmosphere, exposing it to outside impactors and solar particles alike.

Another notable source of lunar surface change comes from an endogenic source. This refers to the lunar thermal-induced crustal deformation, a process where the solid crust contracts and expands due to shifts in temperature. This phenomenon introduces stress on the lunar crust and has been attributed to the formation of lunar geological structures such as lunar graben and rilles (Dawei, 2017). These are structures formed on the lunar surface that have a tectonic-like origin in appearance, and their formation can directly influence the morphology of existing lunar craters. A second source for the lunar's tectonic activity has been attributed to the tidal forces due to the Earth, leading to seismic activity deep within the lunar core (Khan et al., 2013; Nakamura, 1982). This activity can similarly stress the lunar crust leading to further deformation of its surface.

3. Gravity Theory and its relationship with Crater Detection

In previous sections we have discussed several things in detail. We established what crater formations were, and the different aspects related to impact and non-impact craters on the lunar surface. We've established the conditions of their formation, and how they can be further characterized. In addition, we have covered the importance of the accuracy of said characterizations, and how they can be erroneous due to the degradation of impact craters, sometimes beyond the point of recognition. With this information it becomes clear that the optical recognition of craters cannot be the only source. As a matter of fact, there are several studies that examinate craters through widely different perspectives such as through a spectroscopical analysis of craters (Wöhler et al., 2024). In this section we will discuss one such source, lunar-derived gravity data. We will further discuss how this data can be used to supplement the detection of impact craters, particularly those that can no longer be visually identified.

3.1. The Reference Surface

Prior to becoming familiar with the definition of what Gravity is, and how it can be visualized, first we need an object in which to define it. In geodesy, whenever there are observations on gravity or magnetic field applications you often need a way to relate one to the other. Take Earth, for example, and say that you wish to talk about the highest and lowest points on its surface. You set out to acquire this data, but you encounter your first problem, how do they relate to one another? When taking any observations, it's important to have a surface to reference them to, a common ground where these measurements are comparable (Altamimi et al., 2002). If one were to ask about the higher point on Earth one might expect to hear Mt. Everest. From a mean sea level perspective, Mt. Everest is the highest point on Earth. However, if we consider Earth's origin at its center, then Mt. Chimborazo is furthest from the Earth's center (Sharma & Schultz, 2018). There are geophysical reasons for this, as its well known that Earth is not a perfect sphere, but an oblate spheroid, which means it flattens at the poles. This means that if we were to view the Earth's imaginary axes, the polar axes are smaller than the equatorial axes by about 21,000 meters (J. Guo, 2023).

In other words, a reference surface is a mathematically simple shape that is used to reference at object, Earth in our case, and is used to reference subsequent observations. This is very important in geodesy, as having a reference surface supports more precise calculations, and even supports the creation of subsequent coordinate systems. Earth's reference surface is widely supported, but for other planetary bodies, such as the moon, it is not often the case (Iz et al., 2011). Any one reference sphere is described through a collection of parameters that, for Earth, are known as the Earth Orientation Parameters (EOP) (J. Guo, 2023). Once this surface is described It can be applied to an object, Earth in this case, and become what's known as a Coordinate Reference System (CRS). With these definitions in place, it's possible to perform observations and calculation about the world around us while also having a common surface tying everything together. It's worth noting that there are a multitude of CRS, some using different underlying reference surfaces. This amplifies the importance of unbiased and accurate observations, as when you transform between one system and another, errors could be introduced. This is an unavoidable consequence of utilizing different underlying reference systems.

3.2. Definition of Gravity

It's a well-known fact that any object on Earth is being pulled by its gravitational attraction, and some may have also heard that Earth's gravity is $9.8\frac{m}{s^2}$ on the surface. At the simplest of terms, with some caveats, these statements are true, but let's dive deeper into what they mean. First let's consider Earth a point mass (m_e) and put an object of considerably less mass $(m_s = 62kg)$ on the surface of its equator, ~6371km from its center. At this point the force that m_s would feel due to Earth, assuming $m_e = 5972x10^{24}kg$ is given by (J. Guo, 2023; Lowrie, 1997):

$$|F_g| = G\frac{5972x10^{24}kg}{(6371km)^2} = \left(6.674310^{-11}\frac{N\,m^2}{kg^2}\right)\frac{5{,}972x10^{24}kg * 62kg}{(6{,}371{,}000m)^2} = 608.8392\ N$$

Recall that $F = m * a$, and so the force experienced by object m_s while standing on Earth's approximate equatorial distance is also expressed as:

$$608.8392\ N = 62kg * acceleration \frac{m}{s^2}$$

$$Acceleration\ in \frac{m}{s^2} = \frac{608.8392N}{62kg} = 9.8199 \frac{m}{s^2}$$

This is the acceleration of an object due to gravitational attraction, but it isn't what we commonly recognize as Gravity. In geodesy, what we know as Gravity is the combination of two different forces. One of them we're already familiar with, that is the acceleration due to gravitational attraction. The second is what we know as the acceleration due to centrifugal acceleration. It's understood that centrifugal acceleration is a consequence of what's known as centrifugal force, which is a characteristic of particles in a rotating reference frame. As such, gravity is strictly defined as (J. Guo, 2023):

$$\vec{g} = \vec{F_g} + \vec{F_c}$$

Where F_g is the gravitational attraction due to Earth, and F_c is the centrifugal acceleration of an object. Of course, in non-idealized situations the gravitational attraction of all other objects is present, this is important due to the existence of the Moon, our closest planetary body, the Sun, which the Earth orbits, and all other planets in the solar system. These undoubtedly affect the resulting gravity acting on an object, and need to be accounted for, especially when dealing with orbiting objects (J. Guo, 2023).

3.3. Gravity Anomalies and Gravity Disturbance

We have previously discussed that to describe other planetary bodies a simplified reference ellipsoid is used. This is because it offers (1) a simple way to unify all other measurements through one reference, and (2) it offers simpler calculations (J. Guo, 2023). On Earth, the reference ellipsoid is often used to approximate the geoid, itself a theoretical equipotential surface of gravity that would mimic the mean sea level if the effect of tides were removed (Lowrie, 1997). This is a surface where the gravity potential is equal, and the technical definition of the Geoid is that equipotential surface W_0 that closely agrees with the mean sea level (Jekeli, 2000). The direction

of gravity across all equipotential surfaces (Of which there is an infinite amount) is always perpendicular (J. Guo, 2023). Naturally, if you have a point mass exerting some gravitational attraction, there is also an infinite number of equipotential surfaces of gravity surrounding it. For a single point mass (That is invariant) these surfaces may be uniform, and parallel to one another, but on Earth this Is not the case.

A good example as to why a reference surface is chosen is the fact that many things on earth, gravity in particular, experience considerable variation from established models. We may define Earth as having a given mass, and thus give it a number for the gravitational attraction it generates at some distance. This is mostly true the further away you are, but if you are as close as the surface, there are a few things that can affect the gravitational attraction experienced at these points. When you are on the surface, the gravitational potential of two points of equal height can be very different, as this is affected by what's referred to as gravity anomalies (J. Guo, 2023; Jekeli, 2000). These can be anything from higher underground density to the existence of a void underground (like cave systems) or even nearby mountains, as their mass is considerable. In geodesy and geophysics, understanding gravity anomalies can reveal information about the interior surface of planets and their mechanisms (Zuber et al., 2013).

As such, gravity anomalies provide valuable information. By understanding how this deviates from our idealized reference sphere we can construct a more accurate visualization of Earth. We have mentioned that the reference ellipsoid is defined to closely align with the Geoid, and one way we 'tie the knot' between these two surfaces is by defining what's called 'geoid undulation" (J. Guo, 2023; Jekeli, 2000). The Geoid Undulation is, in simple terms, the height between the ellipsoid and the Equipotential surface of gravity W_0 that closely aligns with the mean sea level (On Earth), another name is also geoid height. This is a metric that can be used to better understand the surface of the Earth and provides a plane of reference as it ties the gravitational value of any one point and its corresponding height (Lowrie, 1997). This plays directly into the importance of gravity when it comes to understanding the internal structure of a planet, and how to model it.

Gravity disturbance is a concept that operated like that of the geoid, with the exception that it is the difference between normal and real gravity on the surface (Jekeli, 2000). While the computations of the geoid undulation occur from the same point of reference (The surface of the ellipsoid), gravity disturbance occurs at the point of observation, in this case the surface. Being the difference between normal and observed gravity at the surface, it's useful in revealing localized gravity anomalies. Smaller, fine-detail anomalies are often associated with higher degree harmonic expansions, as noise has more of an effect and causes higher variability than lower degree terms. Because of this, taking their difference results in highlighting smaller detail anomalies due to the higher variability, which means the effect of local topography can be more easily observed (Vajda et al., 2004). In the context of crater detection this can have very useful features as it can reveal subsurface, or completely buried craters. Naturally, this is limited by the spatial resolution of the data, as well as the strength of the gravity signal (Flechtner et al., 2021).

4. Relevance of Gravity in Crater Detection and Planetary Sciences

At this point, it's the expectation that the reader has a good understanding of some of the varying roles of gravity in understanding a planetary body. Obviously, using gravity potential surfaces defining the geoid to then obtain a better understanding of Earth's shape is directly related to geodetic sciences. This begins to divert into Geophysics when other things are taken into consideration, such as crustal density and deformation, plate tectonics movements, or the thermal structure of planets. There is still a good deal of overlap, and one will find that often solutions to one problem (Such as understanding the gravity field) has implications in things like the thermal structure of a planet. This study is concerned with how lunar gravity data overlaps in the study of impact craters. With this in mind, we can briefly discuss some of the overlapping. Despite this gravity data has a large range of uses in geophysics, some of which we mention briefly in this sub-section.

Let's take, for example, the Gravity Recovery and Laboratory Mission (GRAIL) that launched in September of 2011 (Zuber et al., 2013). It consisted of a pair of satellites in the same orbit, equipped with a Lunar Gravity Ranging System (LGRS) to measure their relative distances and speeds. This was an application of what's known as satellite gravimetry, where you take carefully measures satellite observations to infer information regarding the gravity field they are located in. This gives information about the gravity field of a planetary surface by measuring the disturbances to the vehicles as they move through them and inferring the surface gravity (Flechtner et al., 2021). Because we know that gravity is inversely proportional to distance, it's necessary for said vehicles to be as close to the surface as possible. This was, in fact, the case with the twin vehicles Ebb and Flow, who reached an average operational altitude as low as 22km before deliberately impacting the lunar surface in December of 2012.

This mission had two primary objectives, and they were (1) to determine the structure of the lunar interior from crust to core, and (2) to advance the understanding of the thermal evolution of the Moon (Zuber et al., 2013). This groundbreaking mission produced the best gravity field measurements of the Moon to date (Bertone et al., 2021). The implications of these results in the study of the moons structure, and thus establishing the importance of gravity in planetary sciences,

is obvious. This is applied to the analysis of gravity field data to reveal subsurface craters and similar structures (Evans et al., 2016; Huang et al., 2014). This helps show the importance of gravity in the detection of craters, but the application of gravity field data is not often applied to this automated task. To knowledge, the only study focusing on automated crater detection with gravity data was performed with an UNET Convolutional Network in 2022 (Z. Chen & Chen, 2022). In this study it was shown that the application of gravity data in a Convolutional Neural Network (CNN) showed a success rate of about 80%, despite the clear limitations in data availability and resolution. It must be noted that it's not this works intention to have ignored any other similar studies implementing gravity data. The fact remains that it's not a very common application, and given the requirements, perhaps the main limitation is current data quantities. While the data we currently have is very good from a scientific and analytical standpoint, its use in image processing tasks may prove limited to most.

Part 2: Machine Learning Concepts

5. Types of Machine Learning Models and their role in Crater Detection

5.1. Introduction to Machine Learning

Historically the idea of crater detection has been proliferous and well documented since the early stages of planetary science. Being able to detect and categorize the variety of craters on a planetary surface has proven invaluable in understanding its formation and history. However, prior to the advent of Machine Learning algorithms, crater detection through manual labor has proven time consuming. With the ability to acquire data becoming more evident, this task has only grown in difficulty. In addition, Crater detection Algorithms (CDAs) have shown to have challenges in their performance, aspects such as data modality, noise and the large variety of craters shapes and sizes (Tewari et al., 2023). This is an area where Machine Learning Algorithms could improve upon with their ability to learn and adapt to patterns in data. But to understand this further, it's necessary to expand on what a Machine Learning model is.

The concept of Machine Learning refers to the enabling of computers to learn patterns from data based on previously observed examples. It's not very different from what humans attempt when they collect observations and attempt to build a model with variables that would correlate to these observations. If these two things are accomplished what we've obtained is a model capable of predicting the behavior of some phenomenon. In Machine Learning these observations are called inputs, and the model with prediction capabilities is out output. This is automatically assembled by a ML algorithm by optimizing some metric (e.g. loss and accuracy) (Yousef & Allmer, 2014). There are a few basic steps that need to be taken to use ML algorithms. These can be broadly defined as (1) Data collection and feature selection (2) model selection and evaluation (Ma et al., 2018; Pengxiang Wang et al., 2024).

For human constructed models, it's clear how more observations (inputs) can lead to the construction of a better model. The same phenomenon is true for ML algorithms, but there are two

caveats here, (1) ML requires considerably more data and (2) is far more sensitive to the presence of outliers, or bad data (Yousef & Allmer, 2014). The **Data collection and feature selection** steps are of outmost importance for this reason. Though it can be time consuming, the process of gathering good quality data and selecting the features for the ML algorithm to learn from and predict can determine the models' success or hinder it by introducing bad information. This is very important because when constructing models, humans are capable of detecting outliers and bad data, but a ML depends largely on the information fed to it and doesn't have the expert knowledge in the field that humans have.

The process for **model selection and evaluation** can likewise determine your success, and unfortunately there is no easy way to choose a model. A model is generally selected based on your desired application, and this is sometimes done experimentally(Pengxiang Wang et al., 2024). Once a model is selected, and you've made sure your input data is in a format that this model will accept, the evaluation of the model mostly consists of three parts: **Training, Validation and Testing** (Ma et al., 2018). These three steps can give you an idea of how well the model is able to take your input data and learn to generate predictions. There are other considerations such as the possibility of model overfitting, this occurs when a model tries 'too hard' to fit a pattern, this causes the model's prediction to be very poor outside of the data you've collected, this is often called an inability to generalize. In the model selection and evaluation step it's not uncommon to test several models before choosing a best fit (Ma et al., 2018; Yousef & Allmer, 2014).

In relation to training, validation and testing, different models will have varying techniques. Supposing that you've obtained a dataset with the desired, highlighted features (A label), then your next task is to input this data into the model so it can start to extract information and generate predictions. It's imperative that this data is split into a set for training, validation and testing. Each must occur on unseen data otherwise the user wouldn't be able to detect issues with model generalization (e.g. Overfitting or Underfitting) until they attempt to use a completely different dataset(Pin Wang et al., 2021). In the Training step, given an input and a label, a ML model will use the label to prioritize the desired features from your given input. In this process it begins to construct the predictive model. The validation step is a pseudo-test, where it will take an input without the label, and generate a prediction based on the data it has received so far. This prediction

is then compared against the previously ignored label to determine metrics that give the user an idea of how well the model's predictions fit the input data. A 'training session' often refers to several iterations of the same training and validation process, until it's determined that the ML model has 'learned'. Finally, a testing step is akin to a validation process, but the ML model will not be making any changes to its predictive model's configuration.

The previously described process in the training, validation and testing of a ML model with a labelled dataset is called **Supervised** training (Ma et al., 2018). This means that the input data highlighted the features that a model should prioritize. In this context quantifying the success of a model tends to be a direct comparison between the model's prediction and the true value (label). From the context of this study, crater detection, this makes sense as there are several other 'patterns' to be learned from the lunar surface. **Unsupervised** learning methods don't require feature identification at all, they typically work by way of clustering algorithms to reveal patterns in data, rather than try to fit a known pattern to given data (Jain et al., 1999; Ma et al., 2018). Because there is no associated label to unsupervised methods, the success is generally measured by analyzing the patterns found by the ML model's predictions.

5.2. Convolutional Neural Networks

Up until now, I have been using the term Machine Learning very broadly but here are some important definitions. Machine Learning is often used to describe three things: Machine Learning (ML), neural networks and deep learning (DL). All of these, which I will expand upon shortly, are sub-sets of Artificial Intelligence (AI). This work's intention is It's not this works' intention to delve into AI, so the discussion here will be limited to the brief discussion of these three subsets, with particular interest in Dl models.

Firstly, we touch on **Machine Learning**, stating that its broad application is focusing on using algorithms that can enable a computer to mimic the way in which humans learn. The work done by Kotsiantis, et al. does a very good job at putting forward several of the techniques associated with ML algorithms (Kotsiantis et al., 2006). In this work he established 4 main categories of supervised techniques, those are: Logic, Perceptron, Statistical, and Support Vector Machines

(SVM). For the purposes of this work the focus is on Perceptron-based ML techniques which includes **Neural Networks**. These consist of a joined set of **units** that simulate neurons in biological nervous systems (Like the human brain) (O'Shea & Nash, 2015). Much like ML, which requires input, processing and outputs, Neural Networks are composed similarly, except each layer is composed of the interconnected units that make up the algorithm. A very common type of Neural network is known as **Artificial Neural Network** (ANN)

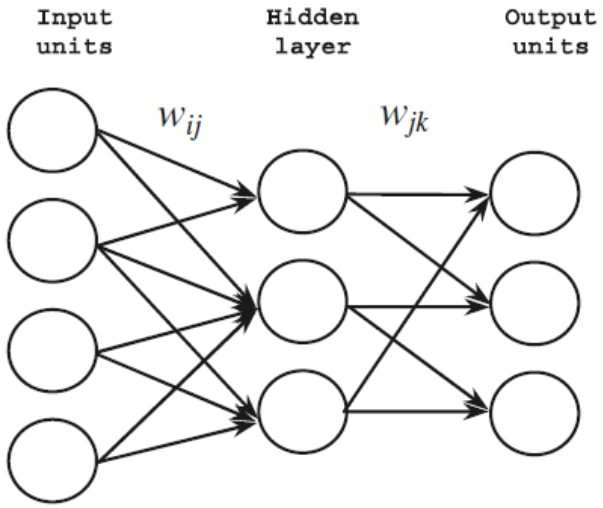

Figure 5. Feed-forward ANN

Output of one layer is only able to propagate forward. Image sourced from (O'Shea & Nash, 2015).

In the types of models described in Figure 5 each node has three main characteristics. First is the input and the activation functions of a unit (neuron), network architecture and the weight associated to each connection. The first two aspects of this are pre-determined by the model architecture. A unit's activation value is an operation performed on the input and is tied to the representation of a feature in the data. This Is propagated forward, and each unit performs its unique activation function. The model can measure each unit's contribution, which they define as the weight between the connected units multiplied by the activation value of the receiving unit (Kotsiantis et al., 2006). One problem associated with architectures like those in Figure 5 is that larger network architectures can be increasingly complex and is what's tied into the large

computational requirements of these models, in addition to the effect of high resolution images and their quantities. A subset of ANN's is **Convolutional Neural Networks** (CNN), which are considered a type of **Deep Learning** algorithm. They are mainly characterized by having multiple hidden layers stacked on each other. CNNs are primarily used for image processing functions, which is the largest difference between them and ANNs (O'Shea & Nash, 2015).

Because CNNs are primarily intended for image processing, this allows researchers to construct architectures in a way that it maximizes this type of input. As such, another difference between the units of an ANN and a CNN is that the latter are comprised of height, width and depth, together this makes up the **activation volume**. CNNs are followed by a set of differently names layers: **Convolutional Layers**, hence the name, **Pooling Layers** and **Fully Connected Layers** (Kotsiantis et al., 2006). In a Convolutional layer the image is taken through an operation designed to extract features from an image. These layers are composed of a **kernel filter**, **stride length** and **padding**. An image is generally seen as a matrix of numbers, and a convolutional layer employs a kernel (or filter), provided its smaller than the input image (Purwono et al., 2023). The kernels are of a given size, and may contain multiple filters, all of which are learnable and used to 'extract' features by multiplying input values with a corresponding weight. These learnable kernels 'glide' through the image in predetermined steps, adding a padding as needed to maintain their size (O'Shea & Nash, 2015; Purwono et al., 2023). The activation maps are determined by the filter values according to:

$$\Sigma_{y=0}^{Columns}\left(\Sigma_{x=0}^{Rows}\ Input(x-p,y-q)Filter(x,y)\right)$$

Where each element (x, y) in the input image is multiplied by its filter value, this is visually shown in *Figure 6*. Each pixel in the activation map is considered a unit (neuron) and is connected to the number of weights in the kernel that filtered them. The activation map is subjected to an activation function, think of it as a function to interpret activation values, and stacked (if multiple filters are applied), which gives us the previously described activation volume (Kotsiantis et al., 2006; Purwono et al., 2023). This activation function can be linear or nonlinear. A linear activation function is of the form

$$F(x) = cy$$

where c is the unit's (y) weight. It results in an output proportional to the input (e.g. 'Yes' or 'No' answers.). A non-linear activation function is necessary for a model to learn more complex patterns and allows it to produce complex mappings between the input and outputs (Purwono et al., 2023). An example of a non-linear function is the Rectified Linear Unit (ReLu) activation function. It's a common function in CNNs and has the form:

$$f(x)ReLu = \max(0, x)$$

Where for positive inputs the function behaves as a linear function

$$f(x) = x$$

But is forced to an output of 0 for any non-positive values. The feature being described here where fixed kernels filter localized patches is what gives CNNs their explicit ability to find local dependencies on data, conversely, it also means that CNN architecture is not set up to recognize long-range dependencies.

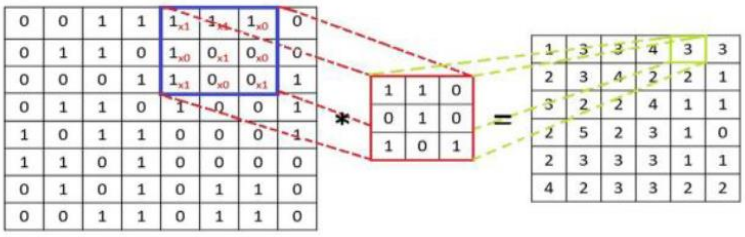

Figure 6. Visualization of a Convolutional Layer.
Input image(left) is subjected to a 3x3 kernel (Filter values in red) to perform elementwise multiplication. To the right is the resulting activation map. Image sourced from (Purwono et al., 2023).

The **Pooling layer** of a CNN oversees down sampling the activation maps of two convolutional layers. This can simplify the computational loads and help with overfitting. This operation further reduces the dimensionality of the previous output, and if it's the final convolution/pooling layer in the structure it's commonly further reduced (flattened) into a 1-Dimensional vector (O'Shea & Nash, 2015; Purwono et al., 2023).

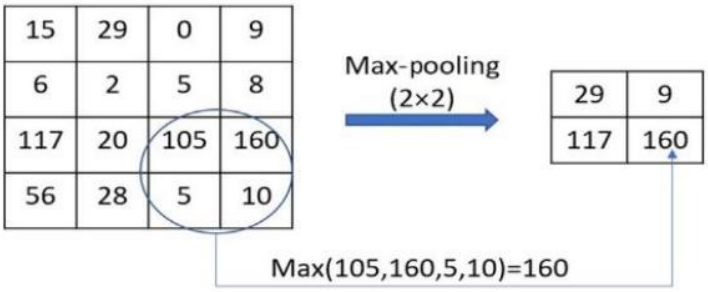

Figure 7. Visual interpretation of a Pooling Layer.
On the left is a Feature Map from the Convolution Layer, and on the left is the down-sampled Feature Map after the pooling process. Image sourced from (Purwono et al., 2023).

As a flattened 1-D Vector the **Fully Connected Layer** or convolutional output layer receives the final input and oversees producing class scores and classifications. This is primarily done by passing the output through connected units (neuron) layers to produce a vector of N size, where N is the number of neurons in the architecture. This process can include activation functions for each neuron as described for *Figure 5*, meaning it functions much like the previously described ANNs. *Figure 8* shows the process being described here. The Input volume is an image input into the CNN. This produces 3 28x28 activation Maps, presumably because 3 kernel filters ae applied These which are further reduced by pooling as described in *Figure 7*. Layers are flattened into a 1-D Vector of size 14x14x3=588 and subjected to the fully convolutional layer. The final output produces classes, of which there are 5.

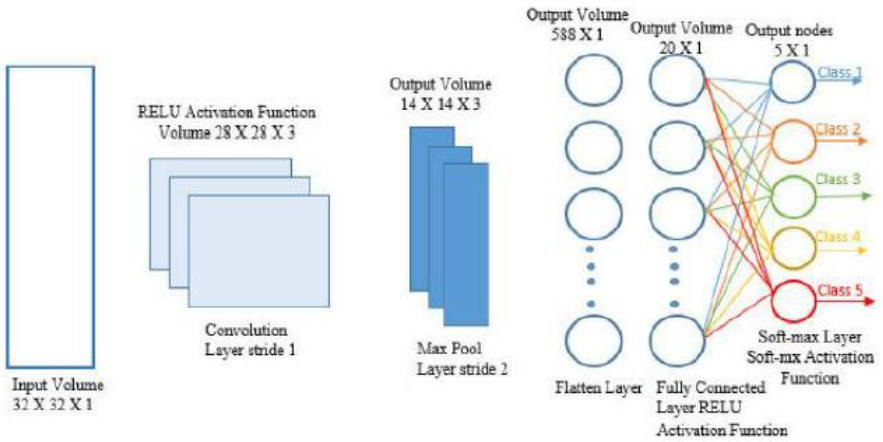

Figure 8. Complete structure of a CNN.
Image sourced from (O'Shea & Nash, 2015)

5.3. Vision Transformer

Another common type of DL algorithm that this work takes interest in are **Transformer** based algorithms. Of interest are those designed for computer vision tasks such as Dosovitskiy, et al. Vision Transformer (ViT) (Dosovitskiy et al., 2021). In a time where CNNs were the de-facto architecture for image processing tasks, ViT demonstrated that it was possible to obtain comparable results on image-processing tasks while requiring fewer computational resources. ViT was modelled after a self-attention exclusive architecture for language modelling developed by Vaswani, et al. (Vaswani et al., 2017)

.

At the time, most models implementing attention mechanisms were utilized in conjunction with a recurrent network (RNNs) such as the long-short term memory network. RNNs are characterized by their ability to 'remember' past states by calculating and storing them sequentially in what's known as hidden states (Hochreiter & Schmidhuber, 1997). This creates two main issues; (1) memory problems given the long 'past states' being stored, and (2) because each state depended on the previous one it made parallel processing impossible. Together with self-attention layers, which main characteristic is its ability to learn long-range dependencies(Vaswani et al., 2017). The

architecture for ViT can be states as follows: **Input Image Splitting**, **Vector Flattening & Embedding**, **Transformer Encoder Layer**, **Classification Layer** (Output Layer).

The Transformer developed by Vaswani, et al. accepted strings of words, and each became a token within the architecture. For images to be treated the same way, Dosovitskiy, et al. introduced an image splitting step, where each input image would be divided into patches of size PxP, given by:

$$Number\ of\ Patches = \frac{Height}{Patch\ Size} x \frac{Width}{Patch\ Size}$$

At this point each patch can be treated as a single word, though not right out of the box. A second necessary step follows a flattening of each patch into 1-D Vectors of size V that can be mapped with a linear projection of dimension D, this works similarly to Vaswani, et al.'s work. The reader will note this is like a CNN's use of multiple kernels in the convolutional layer. The resulting vector with learnable weights is known as patch embeddings (Dosovitskiy et al., 2021; Vaswani et al., 2017).

$$Patch\ Embeddings = (V_1 + V_2 + \cdots + V_i) \begin{pmatrix} W_1 & \cdots & W_i \\ \vdots & \ddots & \vdots \\ W_1 & \cdots & W_i \end{pmatrix} + (b_1 + \cdots b_d)^T$$

Each Patch Embedding is given positional embeddings to maintain positional awareness. Without this additional embedding each patch would be processed, and the architecture wouldn't have a way to determine their previous order. At this point the Patch Embeddings with positional the input is ready to be processed by the Transformer layers, however there is an additional aspect to consider. Transformer based models are known for their ability to capture global dependencies, this is due to a classifier token that is prepended in front of the embedded patches. This specialized token oversees aggregating information by communicating with each patch. The work by Dosovitskiy, et al. deviates once more by adding an additional of these classification tokens. Both classifier tokens are updated through the self-attention mechanism and used to summarize global information (Dosovitskiy et al., 2021; Vaswani et al., 2017).

In the **Transformer Layer** the patch embeddings, with their added positional embeddings, are processed in alternating layers of self-attention and Multi-Layer Perceptron (MLP) blocks. Self-attention blocks are the main mechanism by which the embedded patches communicate with one another, and similarly how the classifier tokens prepended in front of the patches gather information. This is followed by a Multi-Layer Perceptron block, which is a set of units (neurons) with activation functions, as described in ANNs (O'Shea & Nash, 2015; Vaswani et al., 2017).

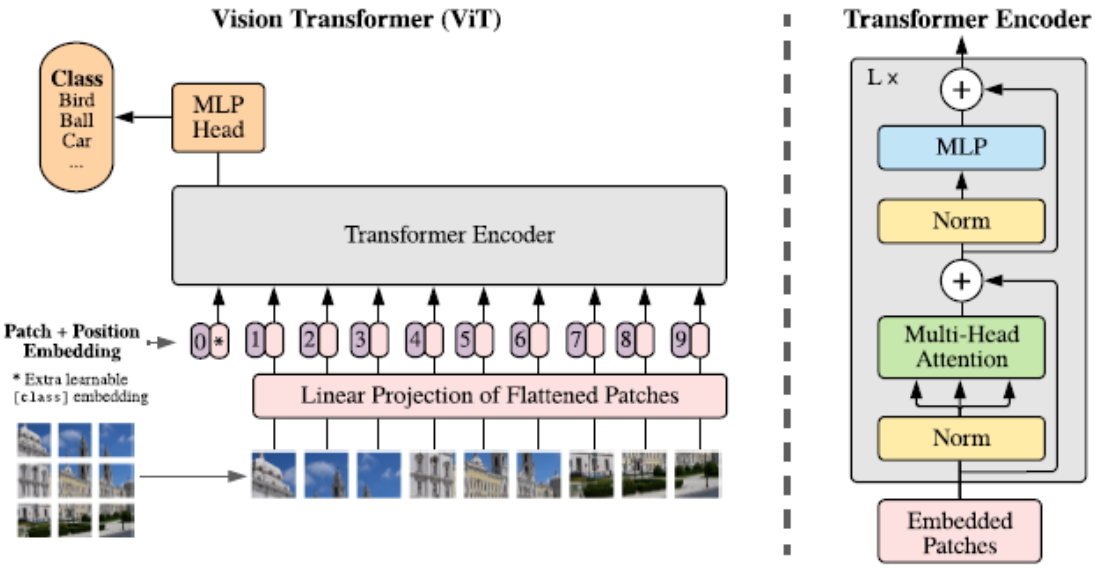

Figure 9. Vision Transformer Overview.

Image sourced from (Dosovitskiy et al., 2021).

6. Usage of Machine Learning in Crater Detection Tasks

It's well known that crater detection has been a very important aspect of planetary sciences, and this is evidenced by the variety of studies in the field(Finkelstein et al., n.d.; Tewari et al., 2023). As it stands the increasing amount of data continually increases the challenge of crater detection. Historically, Crater detection Algorithms (CDA) have been employed with techniques such as shadow patterns, Hough transform and edge detection (Galloway et al., 2014; He et al., 2010; Zuo et al., 2019). ML algorithms work much in the same way, with the primary difference that they're capable of learning by utilizing learnable weights as described in previous sections. Since then, there has been an influx of ML-based CDAs to tackle the task of automated crater detection for planetary sciences with a variety of results.

To my knowledge, there are two main studies that focus on the challenges of producing CDAs via Machine Learning algorithms to date. These studies by DeLatte in 2019 (DeLatte et al., 2019), and later improved by Tewari in 2023 (Tewari et al., 2023), are very important in helping new researchers understand the current state of research in crater detection. These studies take DL architectures into consideration, and as such they are not a full overview. For the purposes of this work, whose interest lies in DL, they are sufficient in introducing their variety and exposing their challenges. Some of these challenges include data quality, repeatability, and visualization (DeLatte et al., 2019). This last point proved very important, as we already know from Robbins' work that when determining impact crater formations, researchers can have a difference of up to 45% (Robbins, 2019). This is even considering that these researchers are well versed in the identification of craters and are looking for a multitude of crater features that extend beyond basic circular patterns (Tewari et al., 2023). These circumstances can shed some light on how even ML algorithms, and CDAs in general can fail to adequately describe crater formations. This is something that is continuously being explored by attempting to leverage continuously evolving algorithms, which has even led to the construction of models for this explicit purpose (Yue Guo et al., 2023).

Another important aspect of the unique challenges in crater detection can be due to their data types. There are several imaging methods to observe the lunar surface, you can use LRO's

Wide Angle Camera to capture optical imagery, and you can likewise utilize its Low Altitude Laser Altimeter (LOLA) to capture information about the terrain's elevation (Robinson et al., 2010; Smith et al., 2017). In this example its clear to see that something like the sun's zenith at the time of collection would affect the optical image, but not the altimeter readings. Conversely, the altimeter readings will likely fail to capture the overall texture and complexity of the terrain. For instance, in a Digital Elevation MAP (DEM) constructed from altimeter readings will not have information about a crater's ejecta pattern, nor will it care about the crater's obscuration due to degradation as would be the case with optical imagery (Degirmenci & Ashyralyev, 2010).

Aspects like these, together with Machine Learning's ability to be very good at learning very specific patterns can cause difficulties in applying a similar model or technique to different: crater shapes, surfaces and lighting conditions (Tewari et al., 2023). This is another aspect of Machine Learning referred to as **Transfer Learning** where you take a ML architecture that has already 'learned' from some source and attempt to apply it to a different task (Dosovitskiy et al., 2021). This Is generally done for specialized tasks and can be met with success even for tasks that appear fundamentally different, as evidenced by Chen & Chen's work, when they successfully applied a pretrained UNET architecture to gravity-based crater segmentation (Z. Chen & Chen, 2022). This has led researchers to attempt to build machine learning algorithms with the specific intention of detecting craters. One recent example of this is called Crater-DETR, a Transformer aiming to improve on the detection of small craters. It was reported to perform well against other CNNs by implementing techniques specifically designed to solve this problem with a reported precision of about 87% (Yue Guo et al., 2023). This clearly indicates that there's still avenues to explore when it comes to applying DL models as CDAs.

Prior to transformer models making an entry into image processing, most image processing algorithms were CNN based (O'Shea & Nash, 2015; Raghu et al., 2021). Part of this success was because the convolutional layers inherent to CNNs introduce a local bias that proved very good at recognizing local patterns, as described in section 5.2 (DeLatte et al., 2019; O'Shea & Nash, 2015; Purwono et al., 2023). This what leads to a reduced performance in the task of exploiting long-range dependencies, given the architecture is not designed for it (Ashtari et al., 2023; Raghu et al., 2021). From the context of this work, automated crater detection, global pattern recognition is very

important given the wide range of impact crater sizes and their distribution. When Vision Transformer (ViT) was popularized, it was understood that its main mechanism, self-attention, was effective at exploiting information available globally, in the context of what's contained in an image (Dosovitskiy et al., 2021; Paul & Chen, 2022; Vaswani et al., 2017).

Naturally, the expanding use of transformer-based algorithms for image processing over the years resulted in a large variety, as per a review by Han, et al. This study attempts to provide a comprehensive understanding of the different aspects in the evolution of transformer models, many of which date between 2019 and 2021(Han et al., 2023). In this review one of the proposed ideas is a Transformer with Convolution, both methods have been explored in sections 1,3 and 1.2 respectively. This idea is not novel, as a variant of it is used by LeCun, et al. (LeCun et al., 1989), where they implement a convolutional layer to extract a feature map, instead of directly splitting an input image as we see in ViT. At the time, it was also noted that hybrid models slightly outperformed ViT, but this was not relevant for larger model sizes (Dosovitskiy et al., 2021).

7. SegFormer and MobileUNETR Architectures

This work adopts the use of two lightweight transformer-based models. A 'lightweight' model in this context means that they are small models designed to perform on low computational resources. This choice was determined by the user's computational limitations, as well as the need to build foundational knowledge in the area. The models chosen were created with image semantic segmentation in mind, this refers to the pixel-by-pixel classification of imagery, rather than clustering pixels together to 'detect' an object (Yanming Guo et al., 2018). Applying classifications to pixels in an image is generally a harder task than grouping them together, as such they generally rely on large numbers of labeled images. When considering SegFormer and MobileUNETR, however, they are described as lightweight and efficient segmentation models, which means they require fewer examples to achieve comparable results, but this depends on the task they are performing (Perera et al., 2024; E. Xie et al., 2021).

7.1. SegFormer Architecture Overview

SegFormer is designed as a simpler and effective semantic segmentation transformer-based model for general purpose use (E. Xie et al., 2021). The work is partly inspired by ViT. It differs in that It splits images into overlapping patches and doesn't apply positional embeddings. SegFormer has two main features in its structure, (1) an efficient self-attention layer designed for the reduction of computational complexity, and (2) A Mix-Feed Forward Network that utilizes 3x3 convolutions to leverage its implicit positional relationships. SegFormer implements a sequence reduction ratio (R) to reshape and aggregate information in patches, reducing the computational complexity. For example, a 16x16 patch with dimension C would have a total of N=256 tokens. Applying the reduction ratio (e.g. R=4) to reduce and reshape each patch would lead to patches of size $\frac{N}{R} x \frac{N}{R} = 4x4$, meaning 4 times as fewer tokens while maintaining dimensionality C. This is processed by the self-attention mechanism, resulting in a much lower computational complexity. The **Mix-Feed Forward Network** utilizes a 3x3 convolution (As seen in CNNs) and utilizes the zero-padding (Adding 0s to a filtered chunk to maintain size) to 'detect' edge features of an image or patch. In other words, recalling Convolution kernels from section 5.2, when the kernel 'window'

is applied, and 0s are added to the 'cropping' to maintain size, this implies information about the current kernel 'position' (O'Shea & Nash, 2015; Purwono et al., 2023). In lieu of positional embeddings (e.g. ViT), this is the method used to maintain positional awareness across patches. The output of this transformer block is the merged feature map of overlapping patches, which is taken as an input to the next block. Each subsequent block processes the Feature Map further, lowering the resolution. The multi-resolution Feature Maps are known as multi-layer features, all of which are input into the MLP decoder. The idea is that each transformer block further refines the previous feature map and this information is stacked, to provide global 'coarse' information and pixel-wise 'fine grain' information. This process of processing feature maps further and stacking them as multi-feature layers is known as hierarchical architecture. As such, SegFormer is not a true Transformer-CNN hybrid model, but it implements techniques seen in CNN models. Note it does not include an actual Convolution layer in the same sense (Convolution+Pooling), only the convolution process itself to learn localized features within patches (E. Xie et al., 2021).

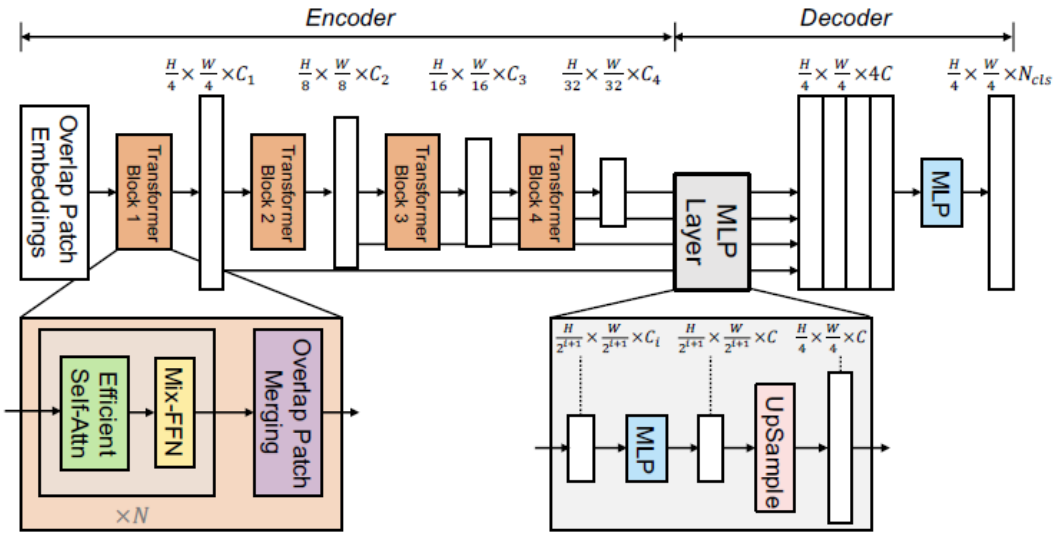

Figure 10. SegFormer Architecture Overview.

Image sourced from (E. Xie et al., 2021).

7.2.MobileUNETR Architecture Overview

MobileUNETR is a Transformer-CNN lightweight model designed for medical image segmentation, unlike SegFormer's general use (Perera et al., 2024). It employs a hierarchical encoder/decoder structure, leveraging CNN and Transfomer mechanisms to maximize efficiency and feature extraction. The **hybrid hierarchical encoder** aims to maximize the feature extraction of CNNs (Local) and Transformers (Global) by a two-step process (Dosovitskiy et al., 2021; O'Shea & Nash, 2015). The first step is to utilize a CNN-style convolution for local feature extraction and down sampling. A second step involves the use of a Transformer-CNN, known as MobileViT, that applies a convolution (Down sampling further and capturing local features) before flattening the resulting non-overlapping patches and passing them through self-attention layers (Capturing global dependencies). The **hybrid hierarchical decoder** applies transpose convolutions for upscaling and combines them with skip connections, red arrows in *Figure 11*, to transfer information lost from down sampling features. The final step in the decoding stage is allowing the Transformer-CNN layers to upscale itself dynamically based on global dependencies (Perera et al., 2024).

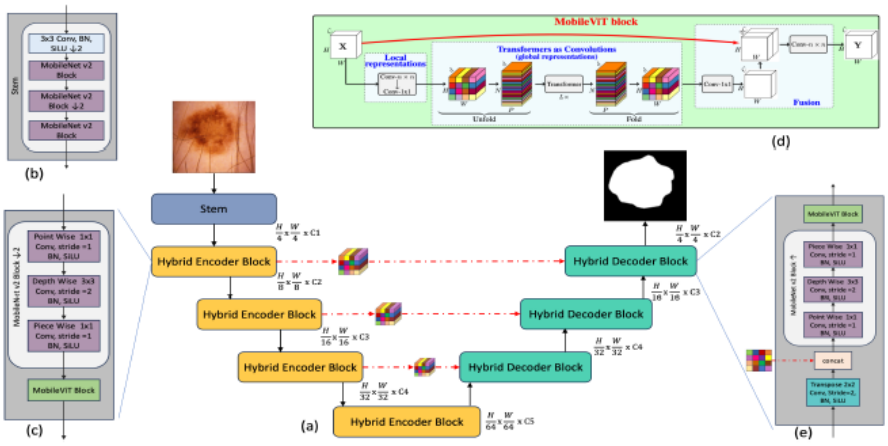

Figure 11. Model overview for MobileUNETR.

In yellow, the hierarchical encoder layers. In green, the hierarchical decoder layers, where the red connections represent skip connections. Image sourced from (Perera et al., 2024).

8. Multi-Modal Data Fusion

In Machine Learning multi-modal data refers to the combining of different data sources to help solve a particular problem. Sometimes this takes the form of leveraging different data sources to ensure more robust predictions, or as a workaround for limitations in certain data types [46]. According to [74] there are three main categories for data fusion, (1) **Early Fusion, Late fusion** and **Intermediate Fusion.** Early fusion refers to the concatenation of multiple data types. That is to say, the data is considered together, each data type describing a different feature of a same phenomenon. This type of data fusion can have problems with data that is sampled at different rates, like combining two data sources that describe a signal. Late fusion refers to the aggregation of two predictions based on different data types to make a final decision. In this instance, two models, or at least two predictions (One for each data type) are needed. There are rules that can be applied that determine the result of this aggregation. Intermediate Fusion works by taking the features extracted from multiple modalities and combining them at one or several layers. This differs from Early and Late fusion in that the data is (1) not combined prior to inputting it in a model, and (2) the data is not necessarily aggregated at the end for a final decision.

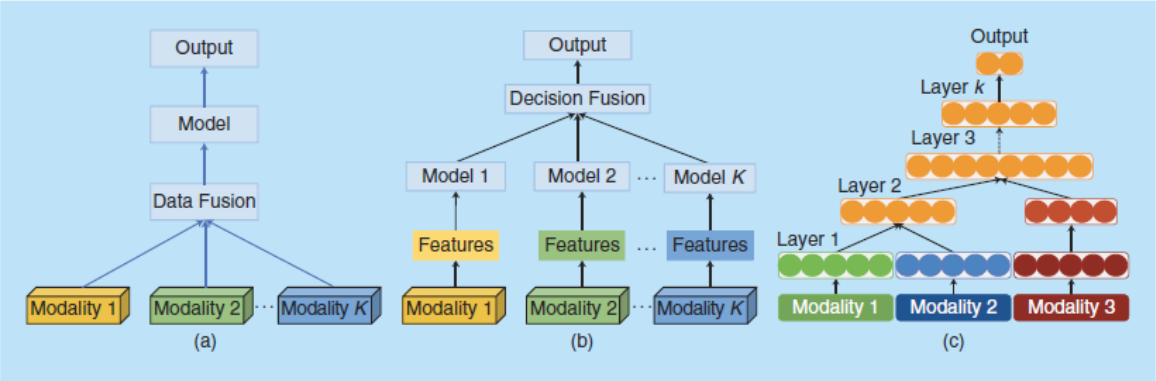

Figure 12. Oversee of data fusion types.

Each modality refers to a data type, such as an optical RGB image (Modality 1) and elevation data (Modality 2). Image sourced from (Ramachandram & Taylor, 2017).

Part 3: Data Overview, Methodology and Experiment Description

9. Challenges and Limitations

To maintain objectivity, I briefly mention the challenges and limitations faces in the preparation, or execution of this work. Either due to unforeseen, or unavoidable factors.

(1) The most important challenge in creating this work was the fact that previous ML experience was limited, and this necessitated time to properly learn and cultivate. This limited the ability to perform tasks such as: Create custom architectures or manually create labelled datasets.
(2) A notable limitation regarding ML models is the computational intensity, which we believe to have addressed appropriately, given our limited computational resources, by choosing lightweight models.
(3) This study was conducted on a CPU due to not having a CUDA capable GPU available. While this was a time-related limitation, it was not severely prohibitive due to choosing lighter models.
(4) Data Quantity was limited given the source of the Gravity Disturbance product is the GRAIL mission. This Is considered in the analysis of the results.
(5) The Gravity Disturbance data is the product with the most limiting pixel size, and because of this we downsample all other products (e.g. Optical Mosaic) to have equal pixel size. ML models obtain better performance on higher resolution data, as such this may limit our results.
(6) This work is being prepared as a requirement to the completion of a M.S. degree. As such there were time constraints related to deadlines. There is a section on future work and idea that, after data analysis, would help provide better results.

10. Data Overview and Storage

The following subsections will carefully explain the steps taken to perform the experiment. First it will cover the tools used, giving as much technical details as are necessary. Likewise, the descriptions of the data and how they were acquired and processed prior to being ingested by the models. Lastly, a detailed pseudocode explanation of the pipeline built to leverage the models is given. Note that this will not include information on the models themselves, as this work did not produce any DL models.

10.1. Tools for the experiment

We used three main tools in the preparation and execution of this work. First and foremost, for data visualization purposes we employed **QGIS** (QGIS Dev Team, 2024). This is a free open-source geographic information system that supports windows, MAX and Linux. It was released in 2002. The primary use for this took was the visualization of our data and their respective labels. It additionally allowed us to 'crop' geographical regions as needed, which was particularly useful in separating the maps and labels into Testing, Validation and Testing, as we didn't have to rely on code to do so.

Our second tool was **Python-based software**, specifically **Python 3.12.5** (Python Software Foundation, 2024). A list of the Python modules that was used is available in the Appendix section together with the code. These python modules are packages that contain pre-written code to carry We used an **anaconda distribution** (Anaconda, Inc, 2024), which already has pre-installed Python modules. This simplified matters significantly, because the built-in function to install new packages, which is named 'conda', is a package and environment manager. It automatically resolves package dependencies within a Python environment. A Python environment can be the base directory where Python runs off as a default, known as the 'Base' environment. Additional environments can be created to work with a clean slate and prevent potential issues from preventing other codes to run. A document named 'requirements.txt' has a list of all the modules

used in this work. It's possible that some of the modules are not used explicitly but were installed as part of requirements for other modules.

The third tool is a module called '**TensorBoard**'(TensorFlow Team, 2024). It's a visualization took kit developed specifically for Machine Learning. In Machine Learning, often the success is denoted by the Model's accuracy. It has the capability to view the several metrics computed by a ML model in steps, or similar time intervals, that can be very useful in understanding a model's progress and help debug if necessary. This module was used primarily for the visualization of our metrics, and initially to debug and ensure proper calculations were being performed.

The fourth, and final tool used is called **PytorchLightning** (Lightning AI, 2024). This is a framework designed to define, train and deploy other ML models, such as SegFormer and MobileUNETR. It follows very basic steps, one of which includes defining a module which includes: Model Architecture, Training Logic and Optimizers. In addition, it allows the customization of your model's training and makes implementations like EarlyStopping, which halts a training session if no improvement is measured, very straightforward. We leverage this framework to streamline the model training process, leaving only overhead model concern, rather than internal logic structures. Within a PytorchLightning Module it's possible to access a given model's Training, Validation and Testing stages, which proved very useful in debugging and visualizing the data as it progressed. It must be noted that the models themselves are Architectures with an algorithm to extract information. In general, to use a model you want to design a framework (Or pipeline) in which they can operate. The PyTorchLightning framework we implemented is described further in section 12.2

As an additional note, the author has decided to use a publicly available GitHub page where they will be storing:

(1) Python codes associated with the handling and processing of imagery data
(2) Regional data with their respective datasets. These will not be split into smaller pieces and is left for users to do with the codes provided.

(3) Python codes that implement Pytorch Lightning frameworks for MobileUNETR and SegFormer, each with their specific requirements met.
(4) All the checkpoints (Model states) and TensorBoard logged metrics related to all the Experiments performed in the process for objectivity and reproducibility.

10.2. Data Descriptions

All the data used in this work is obtained from publicly available sources. Primarily NASA's PDS (NASA, 2024) was used to find the 660-degree spherical harmonics expansion of the Global Gravity Disturbance, also referred to simply as gravity disturbance map in this work. The USGS' Astrogeology Cloud Processing (USGS, 2024) was used to obtain a down sampled version of a LROC-derived, Single-band Orthorectified Optical Global Lunar Mosaic, which we refer to simply as 'Lunar Mosaic'. The acquisition and storage for both is described in more detail. Another source of data is regarding the DL models. The Lunar Database used to create the labels was from the work by Robbins, et al. (Robbins, 2019). SegFormer is imported directly from HuggingFace, a ML open-source community, this was achieved via a module. MobileUNETR can be obtained in the following GitHub Repository (Perera et al., 2024; E. Xie et al., 2021).

10.2.1. Area of Interest

The process of choosing an Area of Interest (AoI) in our work was motivated by two main factors:

(1) A desire to minimize the projection distortions associated with the Simple Cylindrical projections. That means choosing data closer to the lunar equator (0 degrees North) results in data that is minimally distorted, visually speaking.
(2) Because the work by Chen & Chen was the only other found to apply DL models in gravity-based crater detection, choose a similar lunar region, and the same strategy of physically splitting the datasets into Training/Validation/Testing (Z. Chen & Chen, 2022).

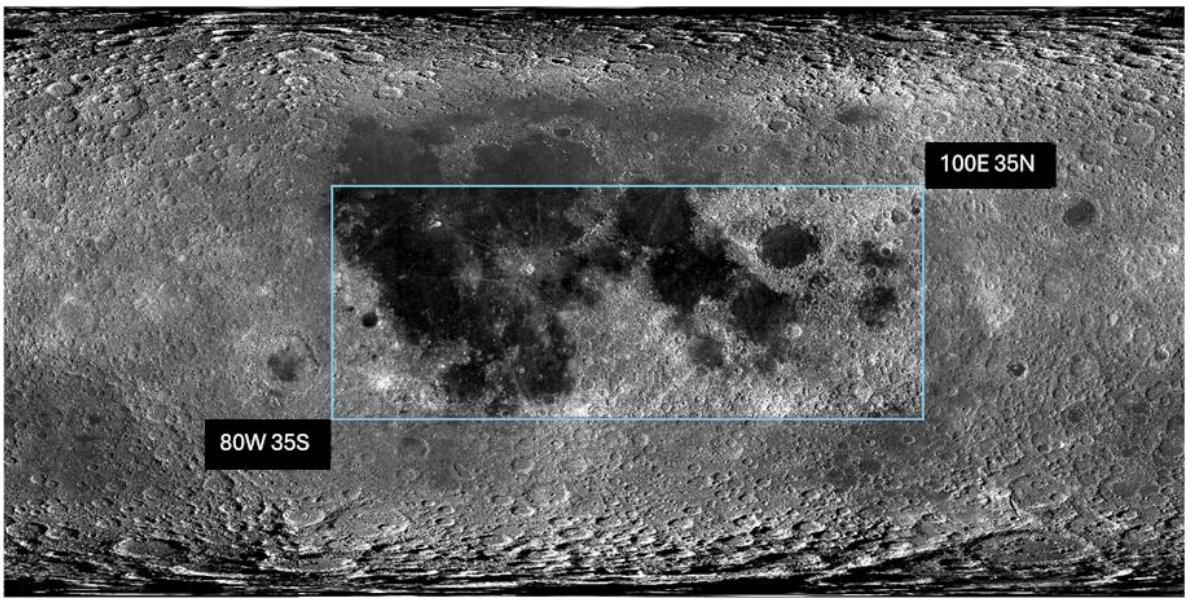

Figure 13. Down sampled Global Orthorectified Mosaic with Region of Interest
The image is visualized through the QGIS software.

Our AoI contains the lunar region centered around coordinates $10^0\,E, 0^0N$ in Mean Earth/Polar Axis coordinate system. The bounding box spans a total of 180^0 in longitude, and 70^0 degrees in latitude. This bounded region is what this work considers in the preparation of the three Datasets. This region is further divided into three smaller sub-regions which correspond to our Training, Validation and Testing regions. This is done to maintain a separation between each of our datasets and ensure that there is no overlap between them. These regions are visualized with the corresponding label in

Figure 14. The Training section is the largest portion, and is centered around coordinates 15E 0N, covering a total of 130 degrees in longitude and 70 degrees in latitude. The Validation section is the second largest portion, centered around coordinates 65W 0N, and covering a total of 30 degrees in longitude and 70 degrees in latitude. The smallest region is the Testing area, centered around 90E 0N, covering a total of 20 degrees in longitude and 70 degrees in latitude. The purpose of this section is simply to familiarize the reader with the AoI and its respective sub-sets, as they are briefly referenced in future sections. While this section only includes the visualization with the Lunar Mosaic data, the creation of a Training, Validation and Testing region is true for each dataset presented in this work. This is detailed in section11.4.

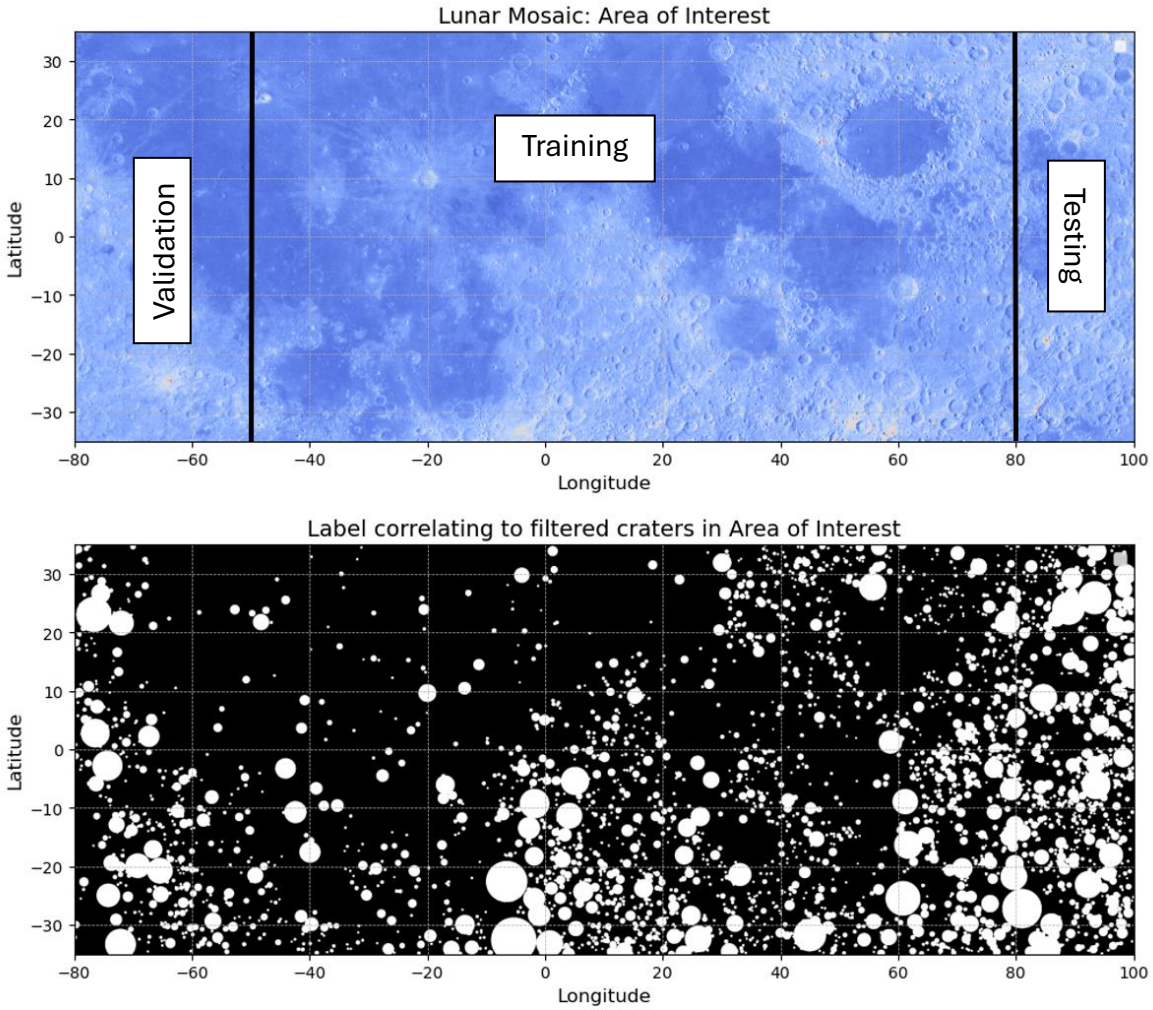

Figure 14. AoI Image and Label pairs with delineated Training, Validation and Testing areas. Python Visualization

10.2.2. Gravity Disturbance Map

The gravity disturbance map is a direct result of the GRAIL mission (Zuber et al., 2013), which lasted about a year and three months. The extended mission where the vehicles orbited at low altitudes of 23km orbits made it possible to obtain significantly better results. Amongst the resulting products is a Gravity Disturbance Map of the entire lunar surface, computed with a

reference radius of 1737.4km. It was previously discussed that the Gravity anomaly is distinguished as the difference between measured gravity, and normal gravity at an observation point. Normal gravity is what's referred to as the value one obtains by performing calculations on a reference sphere. This gravity disturbance is obtained as a spherical harmonics expansion across the lunar surface at degrees 50, 180, 420, 660, 990 and 1200. We follow the associated metadata which defines its pixel size at $1.895\,\frac{km}{pix}$. The Well-Known Text associated with the data is given below. All the provided information is extracted directly from the associated labels and metadata information (Goossens et al., 2016; Lemoine et al., 2014).

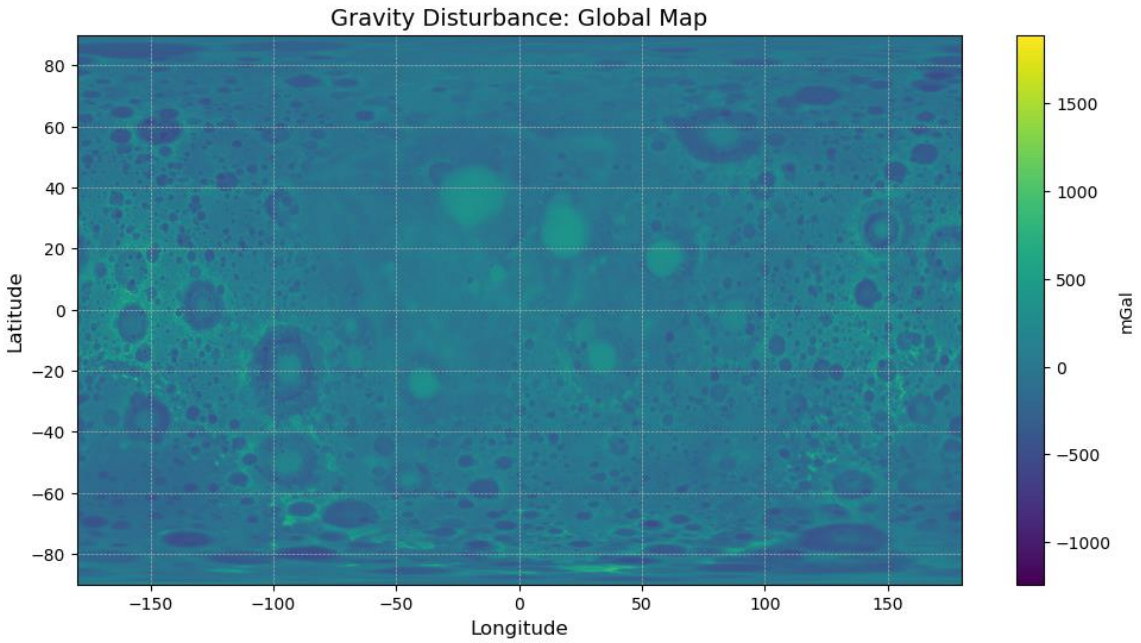

Figure 15 . Degree 660 Global Gravity Disturbance Lunar Map. Python visualization.

WKT: GEOGCS["GCS_Moon_2000",
DATUM["Moon_2000",
SPHEROID["Moon_2000_IAU_IAG",1737400,0]],
PRIMEM["Reference_Meridian",0],
UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],

AXIS[“Latitude”,NORTH],
AXIS[“Longitude”,EAST]]

To aid in the analysis of this data another two aspects are used to describe this data. Of note are the Anomaly Errors and Degree Strengths of the lunar surface. The Anomaly Errors refers to range of error in gravity signal as computer by the GRAIL team for the lunar surface. It’s worth noting that increasing harmonic degree also increases the anomaly error. Degree Strength, on the other hand, is a concept that characterizes the usefulness of an expansion of a given degree by measuring the point beyond which the signal of the observation is overcome by uncertainty. This means that a given observation on the gravity field of the moon, at some geographical region, can only be interpreted via a spherical harmonics expansion up to a given degree before the noise overcomes the signal (Lemoine et al., 2013). Both metrics are useful in informing the data users of potential sources of errors either to be mitigated or be considered in potential analysis. We concern ourselves with the Degree Strength to select an appropriate harmonics expansion of the lunar surface to avoid utilizing data that is dominated by uncertainty. Regarding the Anomaly Error source, we don’t believe it’s something that can easily be mitigated.

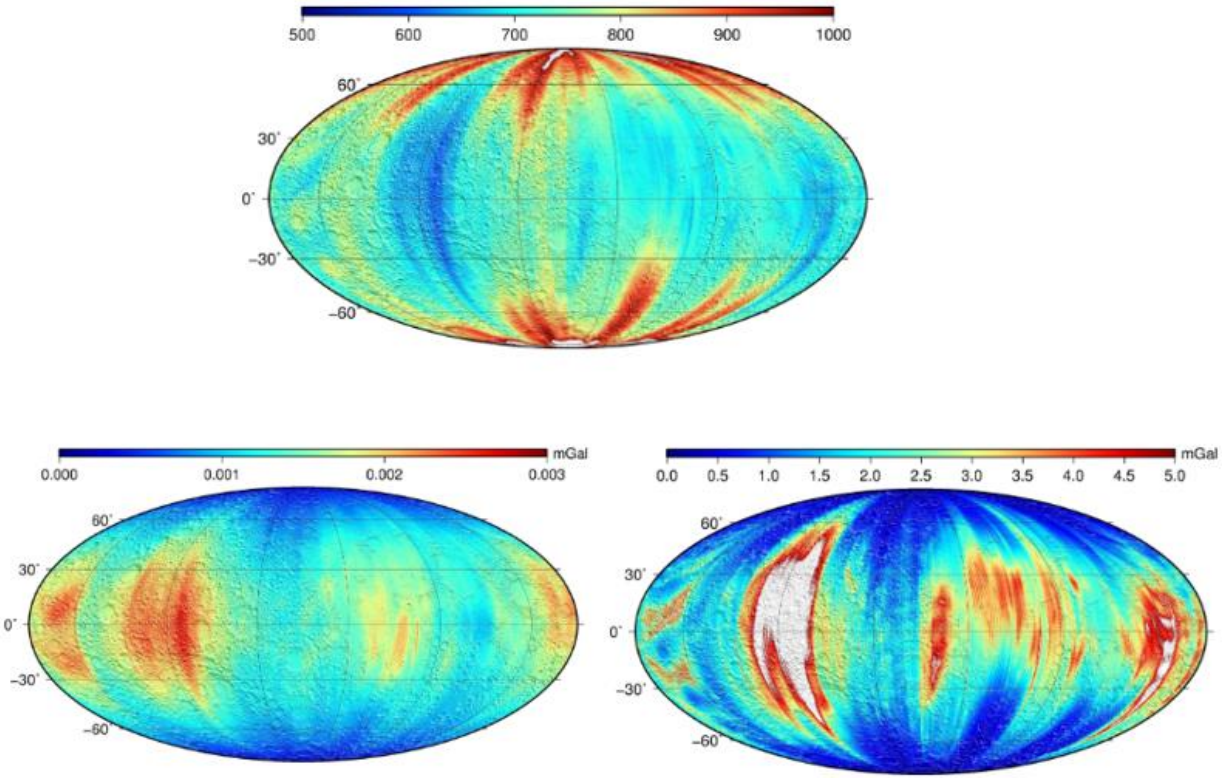

Figure 16. Degree Strength (Top) and Gravity Anomaly Error (Bottom). Gravity Anomaly Error in spherical harmonics expansion of degree 180 (Bottom Left) and 660 (Bottom right). Images sourced from (NASA, 2024).

A numerical exploration of the Degree Strength data helped us in choosing an appropriate degree of expansion for our data. Within the database where this is obtainable there are a total of 6 easily downloadable products. These are degrees 50, 180, 420, 660, 900 and 1200 (Lemoine et al., 2013, 2014). We can note from looking at the data that, in general, the polar regions have a much lower Gravity Anomaly, and correlate with a higher Gravity Strength. Because the GRAIL Vehicles, Ebb and Flow, were in a polar orbit around the Moon, this can be attributed to that (Zuber et al., 2013). During this orbit they would've had a longer time of exposure in those regions. The equatorial region, on the other hand, seems to be associated with the lower of Degree Strength values and high Gravity Anomalies. This information can be good to make informed decisions, although it our case the most useful information we can get out of this is choosing the minimum degree of expansion to balance the Gravity Anomaly with Degree Strength. This is because it's ideal for us to utilize data closer to the equator, where the deformation is minimal. We utilize

Python to numerically observe the values associated with the Degree Strength in our AoI, centered around 10E 0N, to find the minimum degree of expansion required to visualize the Gravity Disturbance data. Visualizing this (*Figure 17*) reveals that 50% of the values fall under degree 711, that means at least 50% of the data will reach a maximum signal-to-noise ratio where noise will begin to dominate the signal. As we're looking to avoid noise overcoming our signal, we choose to truncate the degree expansion at 660 degrees. We recognize that while this means we are eliminating some of the higher degree features, we note that the noise introduction to the data if we were to choose a 900-degree expansion would result in every value in the region being dominated by noise.

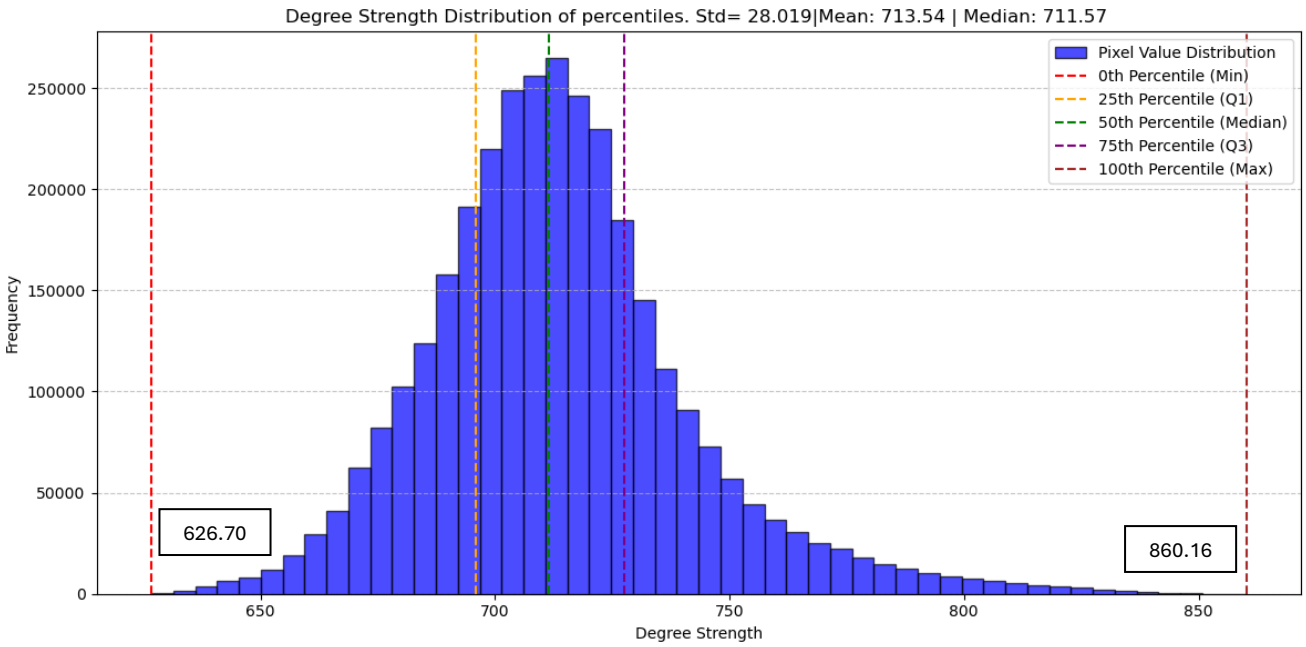

Figure 17. Statistical distribution of Degree Strength values in AoI.

10.2.3. LRO Wide Angle Camera (WAC) Orthorectified Mosaic

The Lunar Reconnaissance Orbiter (LRO) is part of an ongoing process in lunar and scientific exploration. Originally being launched in June 18, 2009, it had two main objectives which were carried out In mission phases (Robinson et al., 2010). These objectives included the Exploration and Acquisition of science data to allow researchers to (1) discover future landing sites, (2) discover lunar resources, (3) measure space radiation environment, to name a few. To support this, NASA equipped the LRO with a Lunar Reconnaissance Orbiter Camera (LROC) which contained two separate instruments. A High-Resolution Narrow Angle Camera able to capture images with a pixel size of 50 $\frac{cm}{pix}$. It has a spectral resolution consistent of a $\pm$ 150 band centered at 550nm. The second object was a Multispectral Wide-Angle Camera with the capability to capture images at 100 $\frac{m}{pix}$ (400 $\frac{m}{pix}$ at Ultraviolet Ranges), consisting of 7 spectral filters between 315nm to 680nm (Robinson et al., 2010).

From images captured by this instrument we obtained a global lunar mosaic at a pixel size of 1.895 $\frac{km}{pix}$, which matches that for the Gravity Disturbance Map. This was deliberately downsampled from the original 100 $\frac{m}{pix}$ pixel size to match the extent of the Gravity Disturbance data. This is a single band greyscale Image that has been Orthorectified, which means that the pixels correspond to brightness detected by the sensor and has been 'stitched' together from a series of images processed to appear as if each pixel was captured from a perfectly vertical position. This is of particular importance if we want to overlay these optical images with a data-derived map like the Gravity Disturbance product. This was obtained directly from USGS' Astrogeology Cloud Processing (USGS, 2024), the details of which are in a document called '*USGS Job Information.log'*, found in our GitHub Repository. The resulting image was Cylindrical Projection of the lunar images, utilizing a lunar radius of 1737.4 km. Its coordinate reference system is the Mean Earth/Polar Axis, and the Well-Known Text associated with the data is given below.

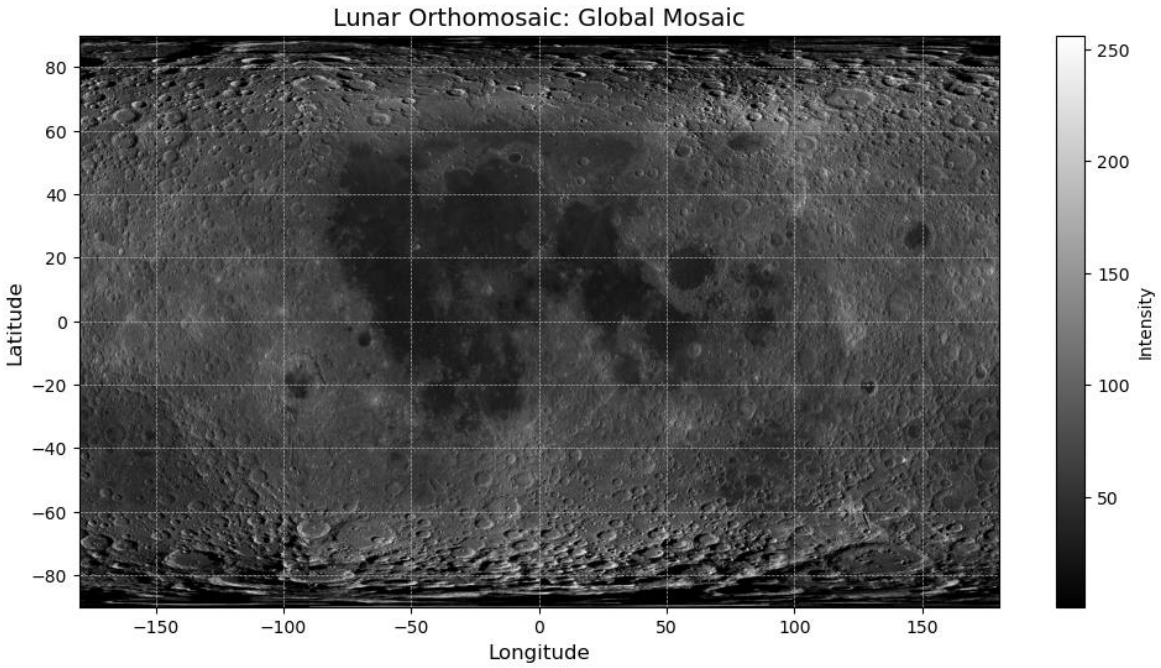

Figure 18. Simple Cylindrical Projection of LROC-derived Lunar Mosaic. Python Visualization.

WKT: PROJCS[“SimpleCylindrical Moon”,
GEOGCS[“GCS_Moon”,
DATUM[“D_Moon”,
SPHEROID[“Moon”,1737400,0]],
PRIMEM[“Reference_Meridian”,0],
UNIT[“degree”,0.0174532925199433,
AUTHORITY[“EPSG”,”9122”]]],
PROJECTION[“Equirectangular”],
PARAMETER[“standard_parallel_1”,0],
PARAMETER[“central_meridian”,0],
PARAMETER[“false_easting”,0],
PARAMETER[“false_northing”,0],
UNIT[“metre”,1,AUTHORITY[“EPSG”,”9001”]],
AXIS[“Easting”,EAST],
AXIS[“Northing”,NORTH]]

10.3. Storage Structure

The python code provided in our GitHub repository makes necessary assumptions regarding the data storage for each associated Dataset. Figure 19 contains an overview of how our data folders were structured. Inside the '*Root'* folder we have Python codes and related utilities. Within the *'Datasets'* folder we have a shapefile containing a polygon of every crater in out Area of Interest (AoI), described in section 11.2, with diameters between 12km to 200km. It also contains the segmentation label corresponding to the region. Each of the *'Dataset #'* folders, of which there are three contain

(1) MobileUNETR and SegFormer folders to store model training Checkpoints and Tensorboard logs.
(2) A folder where the Training, Validation and Testing Image/Label pairs are stored.
(3) The corresponding regional map for each Dataset.
(4) Inside each Training, Validation and Testing folder we have:
    a. Corresponding Training/Validation/Testing portion of the AoI and their label.
    b. Image and Label folders where the pairs are stored. These are created by processing the Training/Validation/Testing portions with '*Overlap Image Splitter.py'*

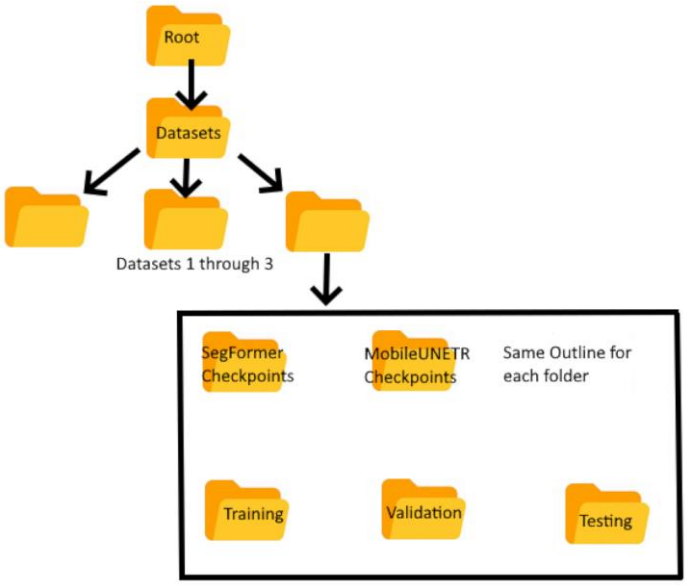

Figure 19. General Folder Structure.

11. Dataset Creation

11.1. Data Processing and Early Fusion

Because the Lunar Mosaic and the Gravity Disturbance Data do not have matching projections, we solve this issue by using a Python code named ‘Reprojection Code’. This permitted us to visualize both data in the same CRS. It works by defining a Target CRS, calculating a transform and reprojecting the original data points. It’s necessary to change the necessary aspects of the Metadata (CRS, transform, width and height). Below we define the CRS we used to apply the transform (MapTiler team, 2024).

Target WKT: 'GEOGCS["GCS_Moon_2000",
DATUM["D_Moon_2000",
SPHEROID["Moon_2000_IAU_IAG",1737400,0,AUTHORITY["ESRI","107903"]],
AUTHORITY["ESRI","106903"]],
PRIMEM["Reference_Meridian",0,
AUTHORITY["ESRI","108900"]],
UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],
AUTHORITY["ESRI","104903"]]')

After reprojecting the Lunar Mosaic to match the Gravity Disturbance data we use a separate Python code called ‘Stack Bands’ to extract, normalize and rearrange the bands into three different resulting products as described below:

(1) Gravity Disturbance: 3 Bands composed of the normalized gravity disturbance data, normalized to a [0,1] range.

(2) Optical Mosaic: 3 Bands composed of the normalized optical greyscale data, normalized to a [0,1] range.

(3) Gravity/Optical Data Fusion: Bands 1 & 2 represent the normalized ([0,1] range) Gravity Disturbance and LRO Global Mosaic. A third zero-valued-band was added.

It made sense to make all three bands equal in the first two examples, as it was the only data source. Because SegFormer and MobileUNETR expect RGB images, 3 bands are necessary unless we alter the architecture itself, which is not in the scope of this work. For the Fusion data it made more sense to add a 'dummy band', or a band of zeros to make up the last band. This decision came from not knowing whether duplicating the Optical of Gravity Disturbance band would've biased results towards one, or the other. As is the case with the reprojection we also need to update the "count" and "nodata" values in the metadata we'll use to save the new data. These values were '3' and '–32767' respectively, the latter added to maintain standards.

$$Normalized\ Band = \frac{band - band.\min}{band.max - band.min}$$

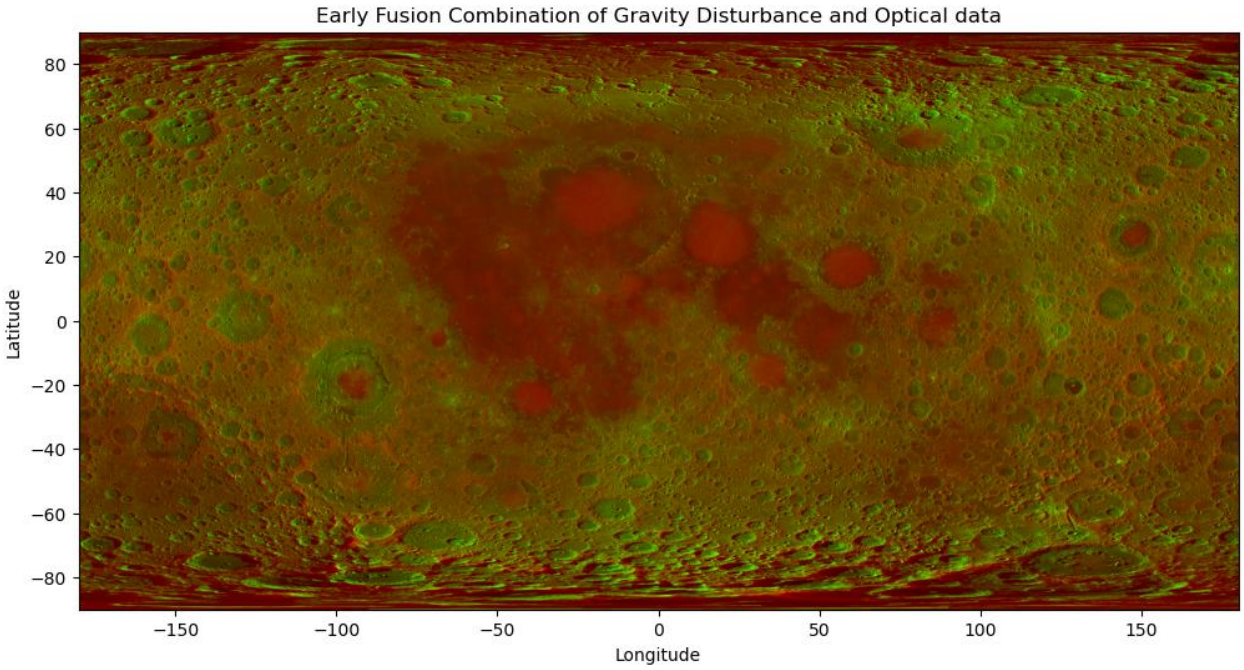

Figure 20. Normalized Global Gravity/Optical Data Fusion.

11.2. Crater Filtering Process and Distribution Analysis for AoI

To create a label image to go with our chosen data, we realize that we must observe the crater distribution in the area to know our expectations. We have a 180by70 degree region of the moon, and at a pixel size of ~1.895 km/pix, our data contains a total of 2880 (width) by 1120 (height) pixel area. Contrasting this with LROC-derived maps that can reach a pixel size of up to $\frac{50cm}{pix}$ (Robinson et al., 2010), which was used in the creation of Robbins, et al.'s database, it's clear we must take certain measures to ensure the data we are going to represent has visible, and identifiable features. We do this by reading and filtering the identified craters in Robbins' database, which is given as an ESRI shapefile. This filtering process goes in two steps. We (1) filter craters whose center is not located in out chosen region, and (2) we filter craters by establishing minimum and maximum sizes. This will be followed by a statistical analysis of the crater distributions of the craters in our region before and after filtering.

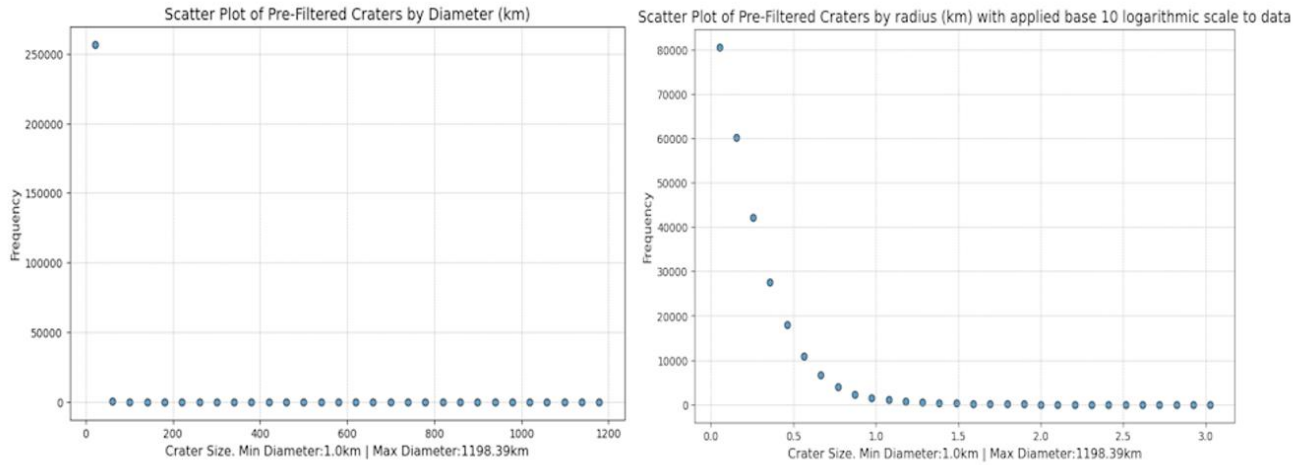

Figure 21. Distribution of Craters within the AoI.
Left: Unaltered data. Right: Base 10 Logarithmic scaling applied to data. This distribution totals 257,141 craters.

Looking at *Figure 21*, the data for which we obtained after filtering by region, appears to be heavily dominated by the smaller craters when viewed at first glance. According to a study done by Wang, et al. (Jiao Wang et al., 2015) cumulative crater frequency decreased exponentially with increasing crater diameter. If we recall Chapter 1, the secondary formation of craters occurs when the debris and ejecta of a freshly former crater results in the formation of secondary craters (Collins, 2014; Robbins, 2019). This study did not mention whether it excluded secondary crates or not, but it's behind one of Robbins' explanations as to why his database included more impact craters than previous studies. A second reasoning was the inclusion of subdued or degraded craters, which is very relevant in this discussion considering that smaller craters are far more likely to be (1) hidden or (2) erased due to subsequent impacting, space weathering and internal lunar processes (Pieters et al., 2000; M. Xie et al., 2017; Zhao et al., 2023).

The rest of the craters in our region were further filtered by size. The reasoning behind this filtering process was twofold. In accordance with Robbins, et al., it requires at least 10 pixels to appropriately interpret crater formations(Robbins et al., 2014). This was in reference to optical imagery, Chen, et al. who worked with gravity data, instead claimed that features less than 6 pixels in diameter were difficult to make out (Z. Chen & Chen, 2022). Given that our data is limited at ~1.895 km/pix, six pixels gives ~11.37km, and so we select a minimum crater diameter value of 12km. We mentioned that this limitation gives us a data with width and height of 2880 by 1120. This entire data is going to be split into three regions for training, validation and testing. Then, each of those regions is further divided into smaller patches representing the samples to perform training and inference. When splitting our data regions into training units we selected a size of 100 by 100 pixels. This means that one training unit could fit a crater/anomaly with an approximate diameter of 189.5km, and so placing an upper boundary of filtering out craters larger than 200km seemed appropriate. We opted to avoid larger craters to prevent potential situations where a single crater semantic label dominated the entire patch, which could introduce errors during training. On larger datasets this may not have as much of an impact, but in our much smaller dataset we need to consider these situations. Generally, a good dataset will have data points that a model can learn valuable information from. This means avoiding situations where an image is largely dominated by one class, or another. While this can't always be avoided, depending on the nature of your data, we determine that too large of a crater label would work against our interests.

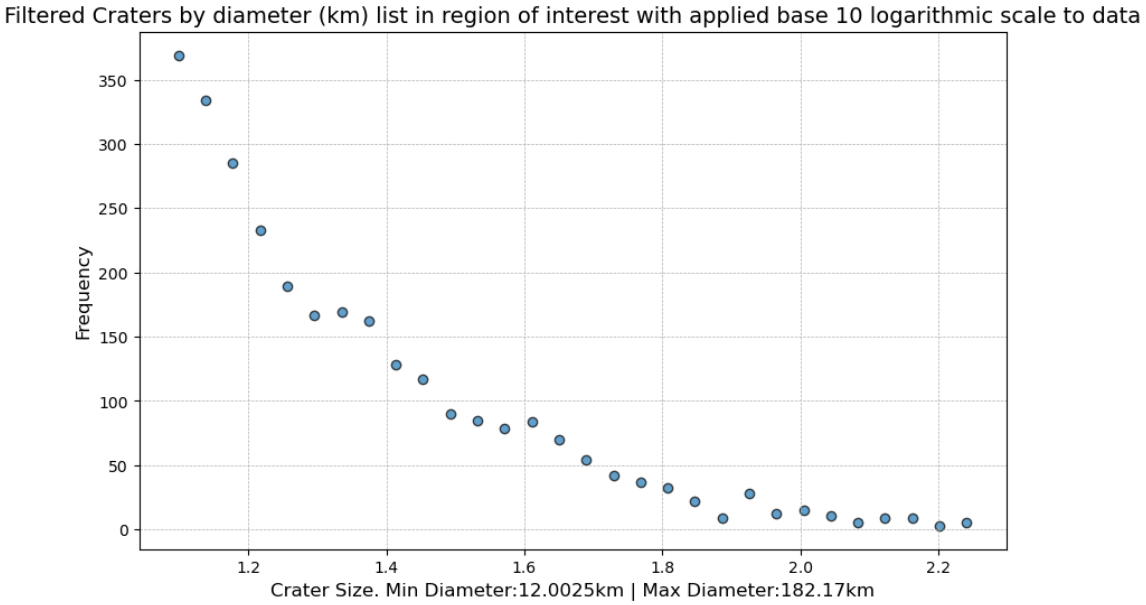

Figure 22. Distribution of filtered craters in AoI.

This distribution represents craters with diameters between 12km to 200km in the area. They total 2,854 craters.

The total number of craters originally in our unfiltered AoI is 257,141. Of this, a total number of 254,273 made up craters under 12km, of which an astonishing 180,621 were smaller than 2km in diameter. The sub 12km craters alone represent a 98.88% of the total craters in the region. Of the remaining 2,868 (1.11%), 2,854 craters are between 12 to 200km, and only 14 make up craters larger than 200km. The center locations of these craters are visualized in *Figure 23* in various data sources to understand how much of this could inhibit our models. Notably, many of these craters are clearly identifiable in both: Optical and Gravity Disturbance data, except for a few that are relatively smaller. While this means that some of the data will be mislabeled, we choose to accept this. A more prudent approach would have been to label the data after it has been split, this would've prevented much larger craters (With no visible rims in the image) to overshadow the smaller craters, which was our main concern. This may have required manual labelling, a task for which we did not have time. For the moment this can be accepted as training the model for smaller-sized crater anomalies, recognizing that much larger ones will likely go unnoticed. Another prudent approach, which is observed by DeLatte, et al. (DeLatte et al., 2018) could have been to label the edges of the craters instead of as an entire solid feature. According to

their results, which used a U-Net architecture on Thermal IR images, best results were obtained with labelled rims, rather than solid features. This is an approach that could not be considered in this work due to the nature impact craters gravity anomalies, which are better detectable as whole spheres. This is on contrast with the Lunar Mosaic, in which only the crater rim is required to achieve identification. The author notes that it may be possible to leverage these approaches by using different types of data fusion techniques.

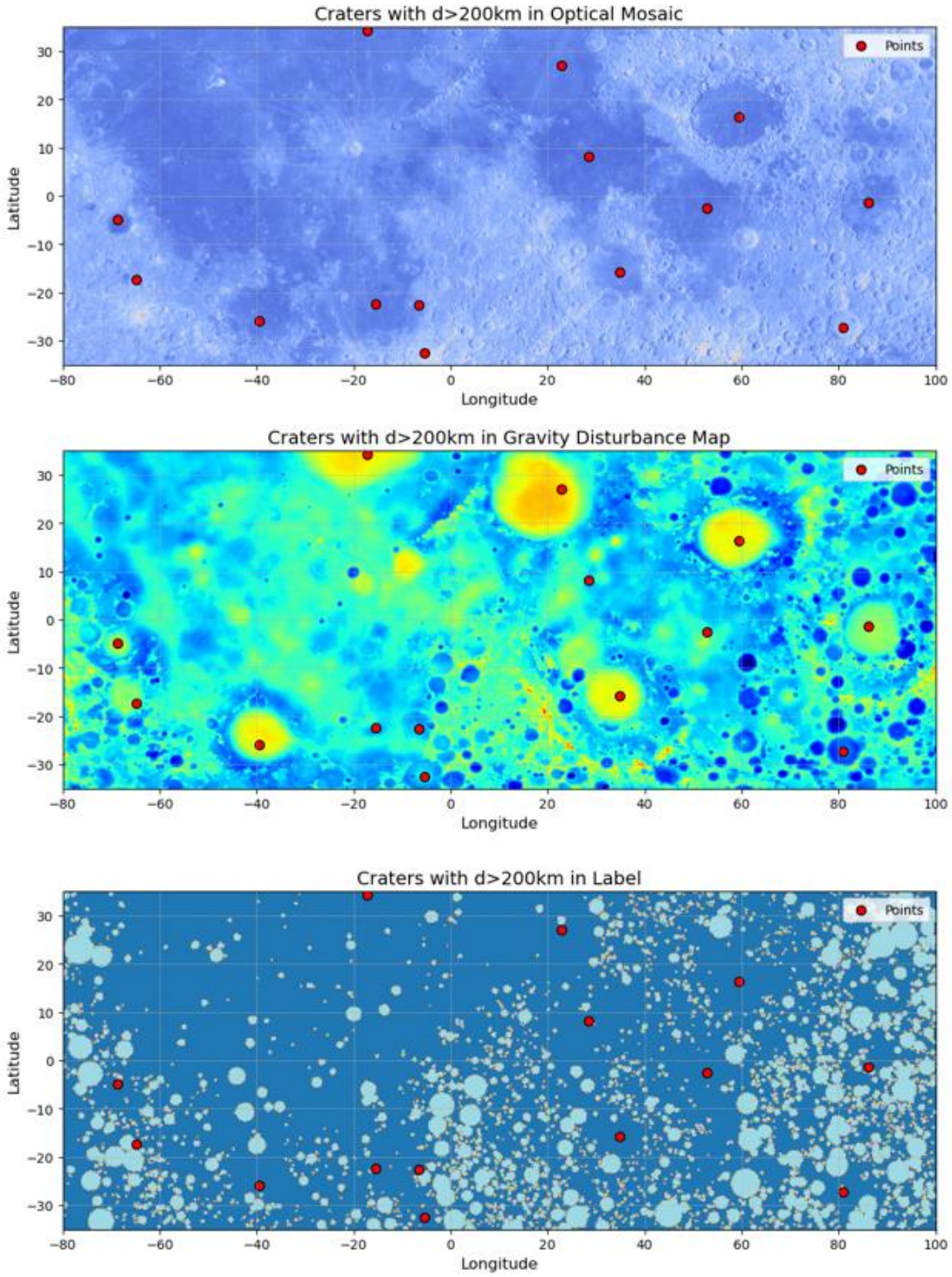

Figure 23. Locations of filtered craters with d>200km in AoI, marked in red. Optical (Top), Gravity Disturbance (Middle) and Label (Bottom). In the label image, dark colors represent background, bright colors represent identified crater.

Having understood the potential limitations introduced in the dataset by choosing to filter the craters by size, we can consider these appropriately during the evaluation of our results. The final consideration to be taken in having filtered the data in this way is in how it affects the data distribution. As it stands, we have chosen to ignore all craters under 12km and those over 200km. It could be said that this means our dataset consists largely of complex craters, which are often considered to occur at diameters close to 15km (Collins, 2014). Observing the trends for *Figure 21* and *Figure 22*, one could infer that the distribution of data is, in fact, affected. To verify, a Kolmogorov-Smirnov (KS), from the *'numpy'* module, is performed on logarithmically scaled Pre-Filtered and Filtered data (Lopes et al., 2007). This type of test assumes that given two distributions, one is a subset of another or follows a similar trend. This is represented by the Kosmogorov-Smirnov statistic, where a number close to 1 means the distributions are very different, while values closer to 0 indicate they're similar. The given p-value gives the probability of observing the KS statistic under the null-hypothesis assumption, that both distributions are the same. Usually, a threshold of 0.05 is applied to the p-value, where it can support the acceptance or rejection of the underlying assumption, depending on the value of your KS statistic. The results gave us the following: A Kolmogorov-Smirnoff (KS) statistic of 0.9888, and a p-value of 0. This says that the two distributions are more than significantly different, as the maximum possible value is 1. The p-value is interpreted as, assuming both distributions are the same, there is a 0% chance of obtaining a KS statistic of .9888. This must mean that these distributions are **not** the same, leading us to reject the null-hypothesis (Lopes et al., 2007). Although we scaled the data logarithmically, which doesn't inherently change the distribution and would allow for a more objective comparison, we did remove all the sub 12km craters, which undoubtedly changed the distribution of the data, as these made up a considerable amount of the total.

11.3. Label Creation

At this stage, almost all the information regarding the thought process, data gathering, and preparation for this experiment has been discussed. The reader is well acquainted with the types of data modalities at hand and has an idea of the size of the data that's being handled. Unsurprisingly it isn't very large when contrasted with other semantic segmentation datasets, which number in the upper 20,000s. By now, it's evident that the selection of lightweight semantic

segmentation models was primarily driven by this anticipated limitation. With this properly addressed, in this subsection we further detail how we use the data processed so far to finalize it into three datasets for crater semantic segmentation. These datasets are as described in section 11.1

In the creation of the labelled data, we only required the post-filtered shapefile, named '*12to200_10E_0N_180by70Area_BoundariesOnly.shp*'. There is an additional step that was not previously discussed, as it was saved for this section. Among the information contained within the shapefile, it included the best approximation of their radius if a circular distribution was assumed. It also included the approximate ellipsoidal axes of the craters' radius, though we opted not to use this statistic. Prior to saving our desired shapefile, we add an additional column of information. This relied on taking each center coordinate for each crater, and computing a Euclidean polygon based on its approximate radius. Each of these polygons were placed in a column named 'Boundaries' and were later projected into the appropriate CRS, which we could take directly from either of our data, as we took measures to ensure compatibility. For storage memory, and efficient computations we eliminate all other columns at this point, as they're no longer needed. This simplified shapefile is then used to create a 'global' mask of our data. This required us to use a global image, rather than the cropped image, because the crater polygons were not completely contained within our AoI, and this created issues with the code. This produced a slightly larger map of our AoI where any pixel not contained within a crater polygon has a 'nodata' value, and all other pixels are given a value of 1. The 'nodata' values are then given a value of 0 before using the QGIS software for (1) visualizing the mask against our data to validate the locations of several craters to ensure correctness, and (2) to crop the larger image down to match our original data size. Since the label is the same for all images, copies of this were distributed across the different 'Dataset' folders described in section 10.3.

11.4. Dataset Splitting and Augmentation

The creation of three datasets consisted of splitting each of the three image types described in section 11.1. This is handled by a python code by the name *'Overlap Image Splitter.py'* and is designed to split a given image with the desired overlap. This same code also handles data augmentation as a method of combating our limited data size and is described in more detail further

ahead. However, before this happens, we use QGIS to manually separate each regionalized image into physically separate Training, Validation and Testing areas as shown in *Figure 14*.

(1) Training Area comprised of a 130 Longitude by 70 Latitude degree region centered on 15E and 0N. It had a pixel size of 2080 Width by 1120 Height pixels.

(2) Validation Area comprised of a 30 Longitude by 70 Latitude degree region centered on 65W and 0N. It had a pixel size of 480 width by 1120 Height pixels.

(3) Test Area comprised of a 20 Longitude by 70 Latitude degree region centered on 90E and 0N. It had a pixel size of 320 width by 1120 Height pixels.

Because the pixel area covered by the resulting split regions is already small, we compensate this limitation by utilizing a technique called 'Dataset Inflation'. This relies on performing operations or transformations to data to alter, or very the features contained within them. Common ways of doing this involve image rotation, pixel erasure, gaussian noise addition and image flipping. The effectiveness of these methods depends on the nature of your data, if you are training a DL model to segment houses, then presenting it with an image of an inverted house, while they do exist, could be considered unrealistic and unnecessary. Similarly, in our case, we're using data that is limited in resolution, as well as be known to contain a certain amount of noise, at least for the Gravity Disturbance data (Lemoine et al., 2013). Because of this, our image augmentation can't depend too much on altering the data itself by introducing additional noise, for example. What we do is select different augmentation methods for each Dataset, where we more aggressively apply transformations on the training dataset. According to Shorten, et al [91], the introduction of variability in the training data can reduce the risk of overfitting and result in better generalization. The Validation and Test data are similarly augmented, though less aggressively. Their purpose is to verify the model's current understanding, rather than be used to learn their patterns.

With this said, the *'Overlap Image Splitter.py'* code contained two main features. The first was to sequentially split each image and label, save each cropping, and then perform image augmentation operations to the cropping before saving it again. This is true for Validation and Testing, the Training Dataset included an additional Augmentation and Saving round, hence the previous descriptor, aggressive augmentation, at least in contrast with the other two datasets. For

each Dataset we describe the (1) total number of original cropping, (2) Types of Augmentations performed and (3) the total resulting images. It's worth noting that for each Augmentation pipeline described, each operation was performed at random based on a threshold. In addition, all operations were performed equally on Image and Label. Data altering augmentations (e.g. Gaussian Noise) were avoided. For reference, in this work we have three datasets, each composed of Images of different modalities as described in section 11.1, and each is further separated into the following three groups:

Training Data was created by cropping the 2080by1120-pixel Image, allowing for a 5% overlap between each patch. This resulted in a total of 231 unique crops. The transformation pipeline performed on the image and label were: Vertical Flipping, Horizontal Flipping, Rotations of up to 45 degrees, and removal of up to 4 'patches' of maximum 10by10-pixels. These operations were performed twice, but not consecutively. It generated a total of 693 Training image pairs.

Validation Data was created by cropping the 480by1120-pixel region, allowing for a 30% overlap between each patch. This resulted in 90 unique crops. The transformation pipeline performed on the image and label cropping were: Vertical Flipping Horizontal Flipping and Rotations of up to 30 degrees. It generated a total of 180 Validation image pairs.

Test Data was created by cropping the 320by1120-pixel region, allowing for a 30% overlap between each patch. This resulted in 60 unique crops. The transformation pipeline performed on the image and label cropping were: Vertical Flipping Horizontal Flipping and Rotations of up to 30 degrees. It generated a total of 120 Validation image pairs

The storage of the now processed image patches and respective labels, which were named to ensure uniqueness and ease of identification, are stored as described in *Figure 19*. The naming conventions for the images are as follows:

patch#{count}_{row}_{col}
{augmentation}_patch#{count}_{row}_{col}_Transform

The first naming structure is applied to the unaltered images, where the patch count, and subsequent row/column of the data identify their position in the overall image. Their geospatial information is updated to be able to visualize them in QGIS. The second naming structure refers to any augmented images, where *{augmentation}* refer to the Augmentation Pipeline. Each Augmentation Pipeline is used to determine the types of data augmentations, and the number of times it's performed, for each There was the pipeline associated with the Training Data, identified by 3, and the one used on Validation and Testing Data, which was identified by 2. In addition, the word Transform is appended to identify them as augmentations of the original data. Because the Training data was put through the same pipeline twice, the second round of generated images had an additional 1 appended to the end. In **all** cases, each image was accompanied by a similarly named label, which was properly identified by appending the text '*_Label'* to the end. In this manner, each image has a unique name, and a unique label pair, while still referencing something about its origin and where it was originally obtained from, at least this is true within each respective Dataset.

12. Experiment Description

The previous subsections in this Chapter have defined how (1) The data was acquired, and what their sources are, (2) What their respective projections were, (3) How the data was processed to ensure it could operate together, (4) How the data was handled prior to 'fusing' it, and how (5) it was split into the Datasets used in this experiment. The next logical step is to explain how the experiment was setup to be conducted. The culmination of this setup is described in the codes named '*MobileUNETR_Framework.py*' and '*MobileUNETR_Framework.py*'. The reason for having two codes for each model is that the inputs and requirements for each model differ slightly. For the most part, both contain very similar structure and class definitions, differing only in the small differences in requirements. For example, MobileUNETR expects an Image as an input, and the mask is used separately to calculate loss and accuracy. SegFormer, on the other hand, expects both to be given as input. It's possible other 'neatly packaged' versions of these models exist and handle these matters internally.

PytorchLightning acts by working together with the model architecture, and handles the training logic, allowing for modularity and flexibility during the training progress. By training logic what is meant is the several small steps that are taken between the Training, Validation and Testing instances. This is done in 'Classes' or 'Modules' that are predefined and can give the model additional instructions on top of the normally taken actions, like calculating loss, saving metrics, or even plotting data. We implement PytorchLightning using (1) tensorboard to store metrics, (2) EarlyStopping to measure model performance and halt if necessary, and (3) A Random Search hyperparameter technique by defining the training within a loop that selects, and stores, new hyperparameters at the start of each new training session.

12.1. Hardware

It was previously mentioned that this model was trained in a CPU due to the lack of access to a CUDA capable GPU. While it's said that this didn't severely limit the experiment, as it still allowed for the time to process it, it still took at least (Average) minutes per training session, each of which

consisted of up to 100 epochs. The CPU leveraged for this was an 13th Generation Intel 13600K Processor. Further, the training was performed on single cores because the author could not figure out how to get the process to work with multiprocessing. This last point reflects the lack of expertise in leveraging computing resources, which the completion of this work partly helps address.

12.2. PyTorchLightning Class Descriptions

The process to use PytorchLightning in conjunction with a model depends on predefining classes that will manage data input, outputs, and conversely, model behavior at each stage of training. This approach is flexible, and further includes a separate 'Trainer' function with its own parameters. These parameters can include Number of max epochs, data logging intervals, stopgap/early stopping measures. In this work's Pytorch Implementation there are three main classes: a DataLoader, a Pytorch Lightning Module, and a Trainer

(1) DataLoader: This class has the responsibility of reading the prepared Datasets described in previous sections. This is not part of PyTorchLightning, but for any model, a proper DataLoader in the pipeline is crucial to ensure that the model is being given properly formatted data.

(2) LightningModule: This is the core Pytorch Lightning class, it's defined with several sub-classes that contain the logic to handle model input and output processing. This is in addition to the definition of several model hyperparameters such as Optimizers and Loss Functions, as described in Part 2, starting on page 21.

(3) Initialization Class (__init__): The main LightningModule subclass. It pre-defines the hyperparameters, models and pretrained weights for transfer learning, if applicable. In this study we use them for both models. This class prepares the components to begin the training process.

(4) Forward Class: This represents the input of data into the model architectures to extract information from images, which outputs a set of final probabilities known as logits. This

information extraction depends on the model architecture, as described in Part 2, starting on page 21.

(5) Training/Validation/Testing Classes: These three are steps associated to a model's training process. The forward step is called during each step to extract information out of the images. In this step we define a 'DiceLoss' loss function and apply a 'Sigmoid' classification function to process the output Logits into class probabilities. The class probabilities are used to calculate model loss, an important metric in performance. As described in Chapter 2, the Training/Validation/Testing steps use the class predictions for different purposes during the 'Model Training' process.

(6) Training/Validation/Testing Epoch-End: This triggers at the end of each step in a respective epoch. It calculates Mean and Per Category IoU and Overall and Class-wise Accuracy.

(7) Optimizers: This step is generally initialized at the beginning of a Training Session for a given model. It defines one of the following Optimizer Functions: Adam, AdamW and SGD with Momentum. One of their main purposes is to dictate how model parameters are updated with the goal of minimizing loss.

(8) Pytorch Lightning Trainer: This is a PytorchLightning function that is not explicitly part of the LightningModule definition. It's capable of handling internal Training/Validation/Testing loops, and abstract metric logging, checkpoint saving, etc. It additionally simplifies a code, ensuring proper reproducibility and modularity. Our Trainer is defined with the following parameters:

1- Using CPU to execute training process.
2- Training has a limit of 100 epochs.
3- Perform Validation once at the end of each epoch.
4- Establish Early Stop logic by measuring Validation, with a patience of 10 epochs.
5- Establish model checkpoint saving logic. It saves the last epoch, and the top 5 in addition based on validation loss.

6- We define a TensorBoard to log the metrics, as well as how often they are logged. This is also a PytorchLightning function.

12.3. Hyperparameter Selection

There are two main types of parameters in a ML model. You can have model parameters, referring to internal parameters that are iteratively adjusted and represent a model's inner mechanisms. These parameters are generally randomly initialized at model start, or sometimes defined by a pre-trained set of parameters. From this point forward the model updates them as training progresses (Monica & Agrawal, 2024). Hyperparameters are fixed throughout the model. They directly influence several model behaviors such as overfitting, and speed of convergence. Choosing the right set of hyperparameters is a crucial step in obtaining promising results, and reporting these is equally important for objectivity and reproducibility (Arnold et al., 2024).

The act of selecting an appropriate set of hyperparameters to optimize model performance is called hyperparameter tuning. In many cases this is done in a case-by-case basis, and finding an appropriate set to correctly influence a model is usually done by (1) extensive search and experimentation and (2) experience. Often, when manually searching for different hyperparameters, prior researcher experience with a given model can be used to troubleshoot. This type of approach can work for specific problems, but it reduces the ability to apply the same fine-tuned model to other tasks (Arnold et al., 2024). Non-manual methods are also employed to find proper hyperparameters, which can include a methodical search through a set of predefined hyperparameters (Grid Search) or employ algorithms iteratively update hyperparameters based on previous results (Bayesian Optimization).

In this work we employ a method called Random Search, where a list of hyperparameters, also called a hyperparameter space, is defined, as in a Grid Search. Where this differs is that several experiment iterations are chosen, with a total of 50 experiments for single-band data and 100 for multi-modal data. The hyperparameter space is exposed in Table # for the reader's convenience. In addition, a random seed is selected, which is logged together with the Hyperparameters to ensure reproducibility. According to Arnold, et al., only 20.31% of studies publish their hyperparameters

together with their results (Arnold et al., 2024). In addition to employing a randomized hyperparameter search, we also ensure to shuffle the order of images in each Dataset prior to a training session.

Part 4: Experiment Results and Discussion

Throughout this work we've been explaining the processing of the obtained data, an optical LRO WAC derived lunar orthorectified mosaic, and a lunar surface gravity disturbance map (Lemoine et al., 2014; USGS, 2024). Parts 2 (Page 21) & 3 (Page 38) further established how the data was processed, and how it will be used by the architectures to obtain information, learn, and generate predictions. In this chapter we discuss the culmination of that process. We have described a total of three Datasets in this work, and the steps to create each one. The first is a gravity disturbance map, derived from a spherical harmonics expansion of degree 660. The second dataset comes from the LRO WAC's orthorectified lunar mosaic, which was down sampled to match the gravity disturbance map. The third dataset is an early fusion data combining the gravity disturbance and optical imagery into a single source. In section 12 wed escribe the process and the conditions for the training sessions of each Dataset, all of which were trained on MobileUNETR and SegFormer architectures. This was all achieved by defining a framework in which to utilize each architecture using PytorchLightning (Lightning AI, 2024; Perera et al., 2024; E. Xie et al., 2021).

Each dataset will be analyzed using the results of both models. It was mentioned that we employed a Random Search method to explore the hyperparameter space, described in section 12.3. The hyperparameter space we defined also included the random choosing of the three optimizers mentioned: Adam, AdamW and SGD. The complete tables of hyperparameters used for each training session can be found in Appendix A (Page 120). The results exploration will depend on observing model behavior across steps. This means to observe their Training/Validation losses in conjunction with Mean IoU and mean accuracies, as well as single-class scores. It was during this analysis that the Datasets were deemed to have a class imbalance, because the background class (Class 0) outnumbers the foreground class (Class 1) by about 4.3 times as much. The total background class pixels consisted of 8,063,043 pixels, while the total foreground class consisted of 1,866,957 pixels. The author believes that this negatively affected the training performance metrics and is considered in the results discussion. Because this wasn't addressed prior to training, no further attempts to address this are made and is identified as potential future work.

Finally, for each model we will select an appropriate set of hyperparameters that represent the best results for each model, and a final testing session will be performed to measure segmentation accuracy on test data. This represents the final step in a model's training, which is what we've been referring to as the testing stage. This methodology is carried out for each Dataset, and the results summarized at each sub-chapter. Because the work's intention is to contrast these three data approaches, we will finish by looking at the results of each Dataset together to determine whether there was something gained by introducing Gravity/Optical data as a pair, instead of considering them independently. One final subsection will detail the intended future work.

13. Dataset Results and Analysis Discussion

13.1. Dataset 1: Gravity Disturbance

Dataset # 1 was decided to be the Gravity Disturbance dataset. The initial intention of this work was simply to use the Gravity Disturbance dataset by itself. However, initial results prompted the author to seek other ways of exploring its application, which led to Datasets # 2 and # 3. This dataset was trained with SegFormer and MobileUNETR for a total of 50 iterations, where a random configuration was chosen. A full list of the hyperparameters used in these iterations is fully referenced in Appendix A (Page 120). Some of these results will be referenced in the discussion ahead.

13.1.1. Performance Metrics Analysis.

This initial inspection of the models' performance can be very useful, especially in the context of what we already know. Out dataset is small, and this is a limitation we already tried to compensate by augmenting the dataset and choosing lightweight models. A second consideration is the knowledge of class imbalance, which was not known prior to model training and thus, has no countermeasure. It's bad enough to consider the models' performance being affected by its size, but the results were very clear in showcasing the effect that class imbalance has on the dataset.

To understand model behavior across all 50 iterations we start by observing the Training/Validation loss behaviors, presented in Figure 24. This overview can be useful in establishing global patterns across both models. As previously discussed, training and validation loss as metrics refer to the quantity of correctly, or incorrectly classified pixels, in the context of image segmentation. Training loss, specifically, is an indicator of how well a model can learn from training data. On the other hand, validation loss refers to how well a model can generalize on unseen data. Together these can give us a clue of the models' overall performance. Ideally, a stable, decreasing Training and Validation losses mean the model is learning and applying its knowledge well.

The first thing to note about these learning curves is that the Training loss appears to be smoother than Validation for both models. Although SegFormer appears to have more difficulty in learning from data, as evidenced by the several plateaus around the .75 loss mark. This behavior is seen in MobileUNETR, but is less relevant, with Training loss taking a sharper drop compared to SegFormer. This says that both models can learn from training data, and that MobileUNETR appears to learn the most, as its lowest Training loss is about .2103, whereas the lowest SegFormer Training loss is at .3685, even if SegFormer appears to show more stability.

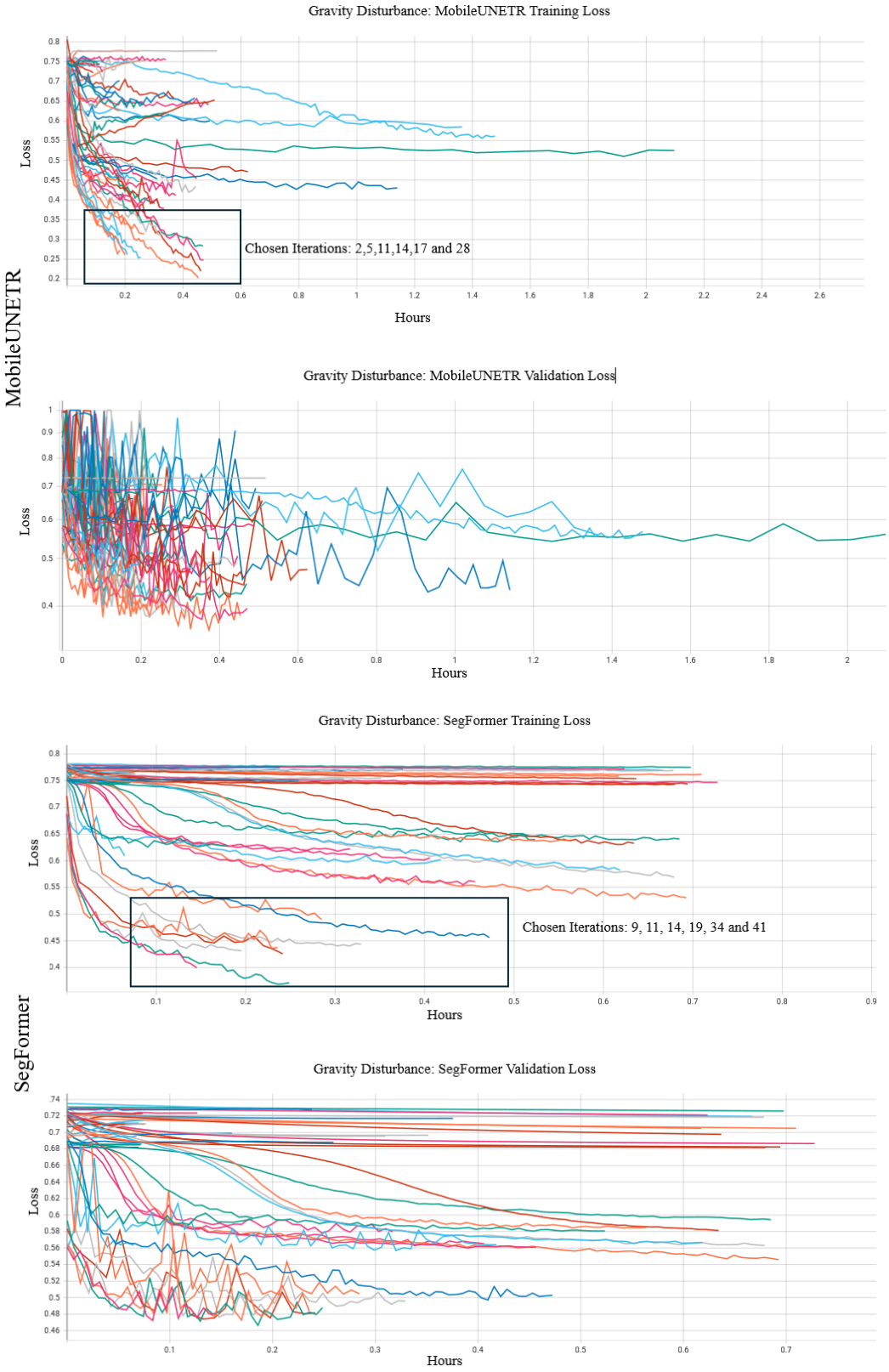

Figure 24. Dataset 1 Training and Validation Loss Curves.

Contrasting these curves with the Validation loss is crucial, as it tells us how well the models are using the learned patterns. Similarly to the Training loss curves, SegFormer appears to be more stable, and leads to long-term plateaus often. MobileUNETR, on the other hand, show a large amount of variability, showing very few examples of plateaus. This issue can arise in part due to the small dataset size but is likely exacerbated by previously discovered class imbalance inherent in our chosen region. To navigate this better we've identified several iterations belonging to MobileUNETR and SegFormer's best Training loss curves. For MobileUNETR we've isolated iterations 2, 5, 11,14, 17 and 28. For SegFormer we've isolated 9, 11, 14, 19, 34 and 41.

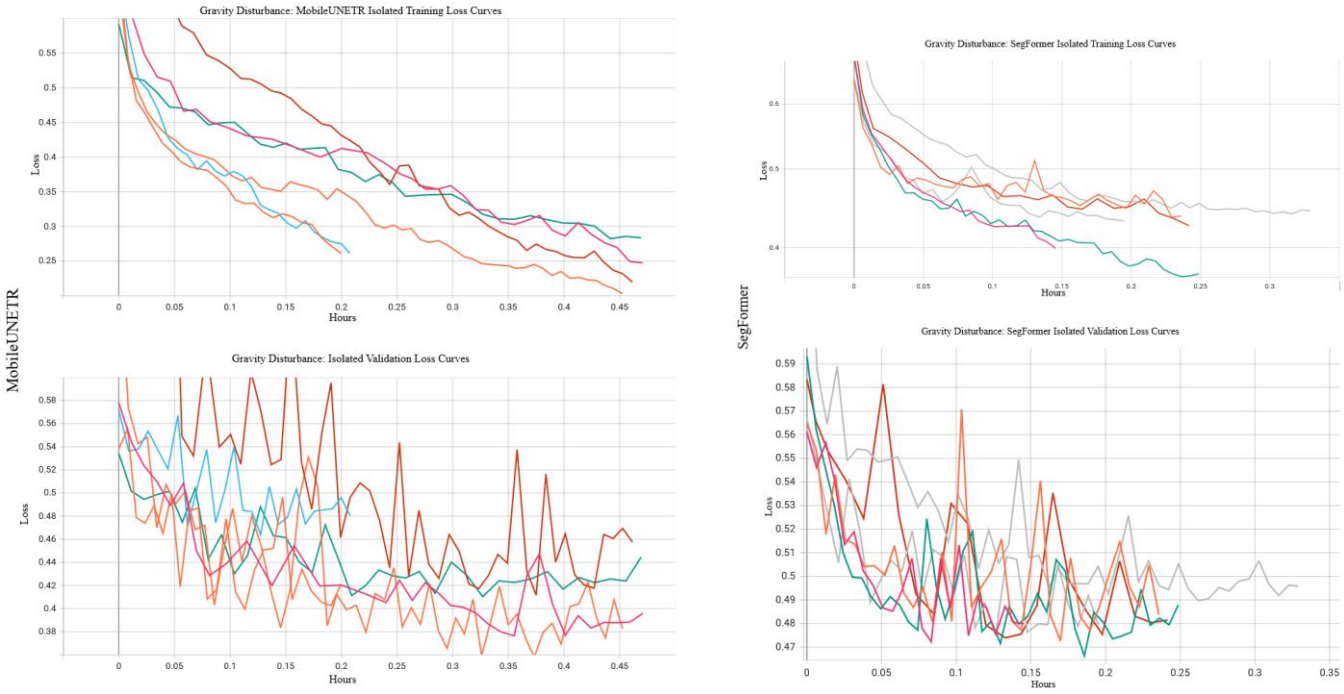

Figure 25. Dataset 1 Isolated iterations for MobileUNETR (Left) and SegFormer (Right).

Table 1. Hyperparameter Selections for MobileUNETR and SegFormer iterations representing lowest Training loss.

MobileUNETR							
Optimizer	Iteration	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?	Lowest Val Loss
AdamW	2	4e-4	32	5e-3	Betas: (.9,.999) AMSGrad: False	False	.4116
AdamW	5	5e-3	8	3e-2	Betas: (.9,.999) AMSGrad: True	True	.4104
AdamW	11	5e-4	8	5e-3	Betas: (.9,.999) AMSGrad: True	True	.3798
AdamW	14	1e-3	32	.2	Betas: (.9,.999) AMSGrad: True	True	.3995
Adam	17	4e-4	24	3e-3	Betas: (.9,.999)	True	.4639
AdamW	**28**	**5e-3**	**40**	**.3**	**Betas: (.9,.999) AMSGrad: True**	**True**	**.3581**
SegFormer							
Optimizer	Iteration	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?	Lowest Val Loss
Adam	9	2e-2	16	1e-2	Betas: (.9,.999)	True	.474
AdamW	11	5e-4	8	5e-3	Betas: (.9,.999)	True	.4723

					AMSGrad: True		
AdamW	14	1e-3	32	.2	Betas: (.9,.999) AMSGrad: True	True	.4726
AdamW	**19**	**4e-3**	**32**	**.5**	**Betas: (.9,.999) AMSGrad: True**	**False**	**.4663**
Adam	34	3e-5	8	3e-3	Betas: (.9,.999)	True	.4897
SGD	41	4e-3	16	5e-3	Momentum:3e-2 Nesterov: False	True	.4754

Figure 25 and Table 1, gives a visualization of the 6 best iterations according to the Training loss. One observation is that both models suffer from noisy validation, which is not unexpected for small datasets. However, MobileUNETR appears to have larger changes, whereas SegFormer's changes appear to be to a lesser extent. Despite this, it's clear that none of these iterations finished their maximum of 100 epochs, which means that the training was halted due to an inability to obtain a better validation in at least 10 epochs. This means that in all 12 cases, the validation loss didn't improve past it's best score at the time. Looking at the behavior of these learning curves, it's easy to see this is true. This behavior, paired with a positive Training loss metric, can indicate that the models' generalization is poor, and unable to adapt to unseen data. Though it must be noted that MobileUNETR achieved the lowest validation loss of .3581, while SegFormer's was .4663.

Because we're aware of a dataset imbalance, its necessary to clarify that simply observing Training and validation loss is not enough in this case. Because we're aware of class imbalance, it's possible that the model is biased towards the background class. To obtain a clearer picture of

this we can observe the class-wise Intersection over Union (IoU) and overall accuracy for validation, which focuses on generalization of unseen data. IoU for a given class is calculated as:

$$IoU = \frac{True\ Positives}{True\ Positives + False\ Positives + False\ Negatives}$$

The IoU can be calculated in two ways. It could be obtained by considering only the True Positives for a given class (e.g. 5 classes = 5 different IoU calculations), and their respective aggregation as what's known as Mean IoU. When considering it as a class specific metric it can be thought of as an approximation of how precise the model segmentation is for a given class. This is like a Class-specific accuracy, except that this metric does not consider False Positives or Negatives, only whether you correctly classified a set number of pixels as their respective classes. For this reason, we give class-wise IoU higher regards, especially considering our models could potentially bias towards our foreground class. Therefore, by observing the IoU for both classes we can obtain a numerical idea of how precise our models are. To exemplify this, if a model's prediction claims that every single pixel in an image corresponds to Class 1, the Class 1 Accuracy will be 100%, but unless the model is correct, it misclassified many other pixels in the process. Depending on the actual class distribution in the given image, this will result in a lower IoU, indicating problems that would otherwise have gone unnoticed. While IoU will act as the precision with which a model can classify pixels of a given class, Overall Accuracy is the mean value of the Pixel-specific Accuracies of both classes. This means that the value is higher when the models are performing well in both of our classes, and is given by the following formula:

$$Overall\ Accuracy = \frac{Total\ of\ Correctly\ Classified\ Pixels}{Total\ Number\ of\ Pixels}$$

We have two metrics (1) IoU to measure how well a given class is performing, and (2) Overall Accuracy to act as a global metric. To give an example, if we find that the IoU for our Foreground is 20%, but 70% for background, we can determine there is a bias towards the background class. Similarly, a low class-wise IoU but high overall accuracy tells us that one class is being favored over another. The class-wise results for Classes 0 and Class 1 for both models are

shown on Figure 26. We first show the global behavior to understand the range of results possible for these models, and to be able to contrast both models easily.

One of the first things to notice from Figure 26, other than the wide range in IoU calculations, is that SegFormer appears to offer more stable values across epochs than MobileUNETR. This is comparable to the global behavior of Training and Validation losses, where it appears that SegFormer's stability is more dependent. We also note the range of values, where the top IoU values for Class 0 appear to be .827 for SegFormer and .8636 for MobileUNETR, these results are from iterations 14 and 28, respectively. On the other hand, the achieved IoU for Class 1 is .4726 and .3652 for MobileUNETR and SegFormer, which corresponds to iterations 28 and 19. It should be noted that MobileUNETR Iteration 28, and SegFormer Iteration 19 were both identified as having obtained the lowest validation loss at some epoch, while being amongst the best iterations for training loss. To contrast this appropriately, we record the class-wise IoU at the same steps for which the lowest validation occurred. This can tell us whether these values occurred simultaneously, or whether there was a trade-off between class-wise IoU and validation loss. This information is in *Table 2*, and we consider all iterations described in *Table 1* for completeness.

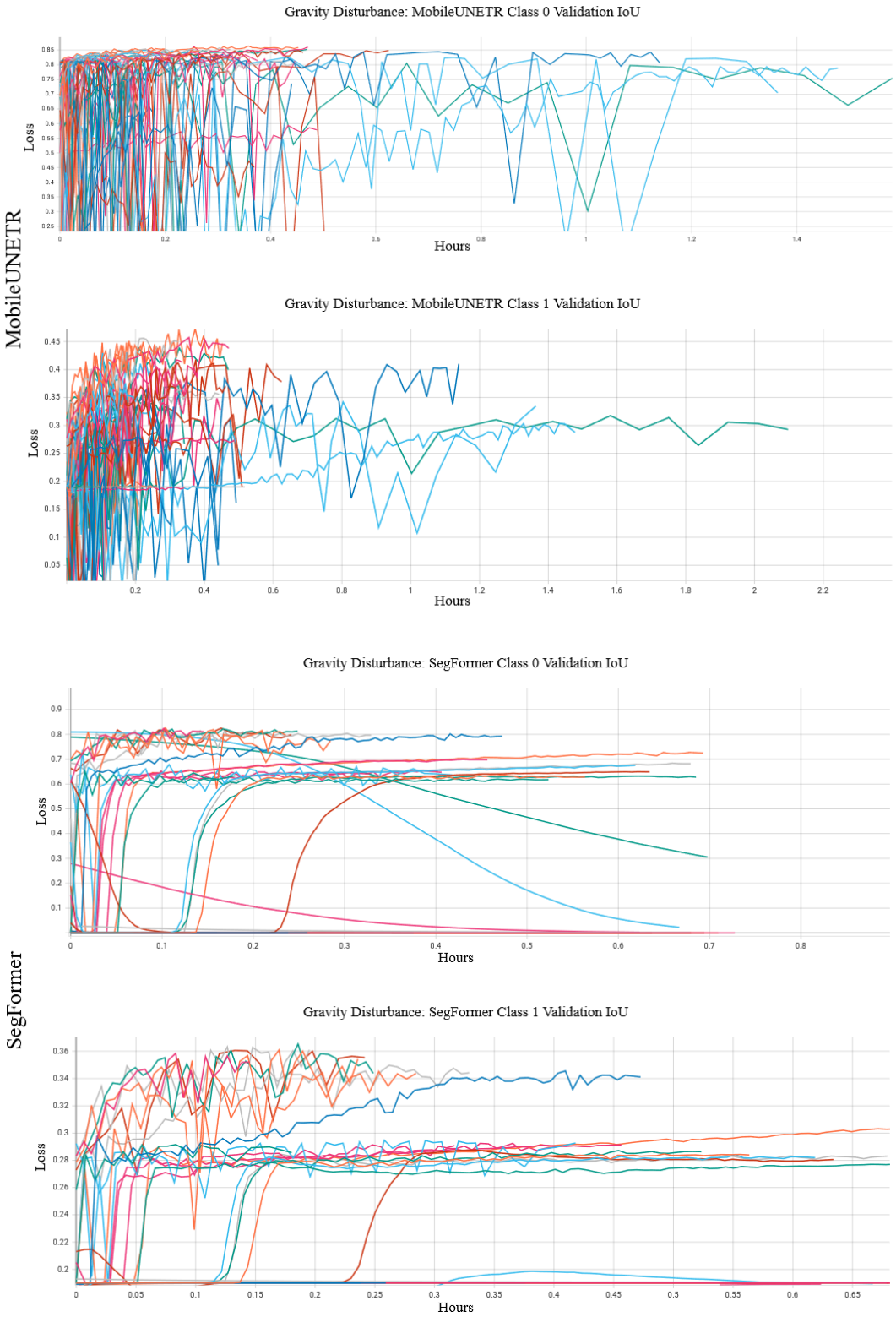

Figure 26. Dataset 1: Class-Wise IoU results for MobileUNETR and SegFormer

Table 2. Report of class-wise IoU at same step where lowest validation loss was registered.

MobileUNETR					SegFormer				
Iteration	Step for lowest validation loss	Lowest Validation	Class 0 Val IoU at same step	Class 1 Val IoU at same step	Iteration	Step for lowest validation loss	Lowest Validation	Class 0 Val IoU at same step	Class 1 Val IoU at same step
2	901	.4116	.8259	.4179	9	233	.474	.7557	.3606
5	2.435k	.4104	.8393	.4387	11	251	.4723	.7698	.3584
11	2.261k	.3764	.8495	.4574	14	782	.4726	.7733	.3609
14	329	.3995	.826	.4432	**19**	**557**	**.4663**	**.794**	**.3652**
17	463	.4639	.8213	.3706	34	1.159k	.4897	.7867	.3497
28	**827**	**.3581**	**.8575**	**.4726**	41	550	.4754	.786	.3632

Interestingly, for each of the chosen iterations, the highest Class 1 IoU was achieved while the lowest validation loss was also achieved. The highest value for Class 0 IoU did not correlate with any of the best validation loss scores. Considering that the top Class 0 IoU scores were .8636 for MobileUNETR and .827 for SegFormer, we can conclude that MobileUNETR's Iteration 28 and SegFormer's Iteration 19 still appear to give the best result, as the models, at that specific instance in training, maximized the class-wise IoU while minimizing loss during validation. Observing the learning curves for Overall Pixelwise Accuracy reveals that for MobileUNETR, the score for Iteration 28 at step 827 was .8736, where the best score across all iterations was achieved by Iteration 28 at step 719, with a score of .876. For SegFormer, the score achieved by Iteration 19 at step 557 was .8159, while the best score was achieved by Iteration 9, with a score of .8381.

Finally, observing the overall values for what we could consider to be the optimal Iterations for MobileUNETR and SegFormer, even at their best performance, the IoU for Class 0 is double that of Class 1. This can be rationalized in that there are 4.3 times more Class 0 pixels than Class 1, and this bias is a natural occurrence of this dataset imbalance. Had this been detected prior to

training the models, it would've been possible to assign class weights, or weighted loss handling functions to mitigate this. This is considered in the analysis of the Test performance, which will be handled by choosing the hyperparameters that were determined to achieve optimal performance for each model.

Table 3 Dataset 1Hyperparameters for MobileUNETR Iteration 28, and SegFormer Iteration 19.

MobileUNETR						
Optimizer	Seed	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?
AdamW	1638565519	5e-3	40	.3	Betas: (.9,.999) AMSGrad: True	True
SegFormer						
Optimizer	Seed	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?
AdamW	103299523	4e-3	32	.5	Betas: (.9,.999) AMSGrad: True	False

13.1.2. Testing Results with Optimal Hyperparameters

The previous section covered the models' overall behavior, and hyperparameters that gave us the best results. Out of that we pick one group for MobileUNETR and SegFormer. The process to 'Test' a model is not different from the Validation step in a normal training schedule. We will 'load' the checkpoints associated with the optimal performance into the model, which means to restore the state of the weights, and all other hyperparameters, at that point in its training. Then a third, unseen, dataset will be seen by it, and after extracting information, it will generate its predictions. This is the equivalent of taking the model at the point where it was most successful, according to our analysis, and using that heightened state to perform predictions. This previous process of

training and identifying optimal model states in relation to hyperparameters can be thought of as fine-tuning.

Because we are testing the model, there is no need to run multiple iterations. A single iteration on the target test dataset suffices. The results of this test on both models are summarized in Table 4. Interestingly, as opposed to the metrics obtained in Training and Validation, in Testing there isn't the previously apparent bias, and the validation loss obtained is substantially smaller than expected for both models. This was a tradeoff with a reduced accuracy in Class 0, as evidenced by the lower Class 0 IoU, even though it's compensated with an increased detection of Class 1 pixels. This prompted us to take a closer look at the previously discovered class imbalance. We discovered that while the overall class imbalance is representative of the class distributions in the Training and Validation groups, this was not the case for Testing. The Test group of images contained a total of 692126 Background and 507874 Foreground pixels, which correspond to ~57.6% and ~42.3%, respectively, which the author believes no longer constitutes a class imbalance. To contrast, in the Training and Validation groups the percentages are (85%|14%) and (80%|19%) respectively.

*Table 4. Test Metrics from optimal MobileUNETR (Iteration 28) and SegFormer (Iteration 19) models. We mark the best metrics in **bold**.*

	MobileUNETR	SegFormer
Validation Loss	**.2562**	.2820
Class 0 IoU	**.7289**	.6531
Class 1 IoU	**.5940**	.5627
Overall Accuracy	**.8059**	.7601
Mean Accuracy	**.7879**	.7560

While the fortuitous occurrence of having a better distributed test dataset resulted in better results, this still needs to be accepted with care. In both cases the models performed Training and Validation with heavily imbalanced image/label pairs. It can be argued that this resulted in less robust models than they otherwise would've been, had the imbalance been corrected. This could mean that the model didn't necessarily perform better because it was better prepared, but because pixels were more evenly distributed across the images. That is to say, the model could still have trouble generalizing but has correctly classified more Class 1 pixels simply because their relative numbers more than doubled. After all, it's important to remember that the Training/Validation and Testing images should all be equally representative of the entire distribution. In our case, this is no longer true, as we've discovered.

On the other hand, these results could imply that if the dataset limitations are addressed (e.g. Class Imbalance) then the resulting models would be more robust, even if we maintain a small dataset size. For added context, we visualize two of the segmentation examples for MobileUNETR and SegFormer on Figure 27, where the first example is a segmentation of the same image. What becomes immediately apparent is that MobileUNETR's segmentation is smoother, and more contained, which adds to the appeal. SegFormer segmentation appears to be sparser, where nearby pixels can sometimes be classified as well, as though the segmentation were 'leaking' into its surroundings. This may be a result of SegFormer having been designed for more classes than MobileUNETR.

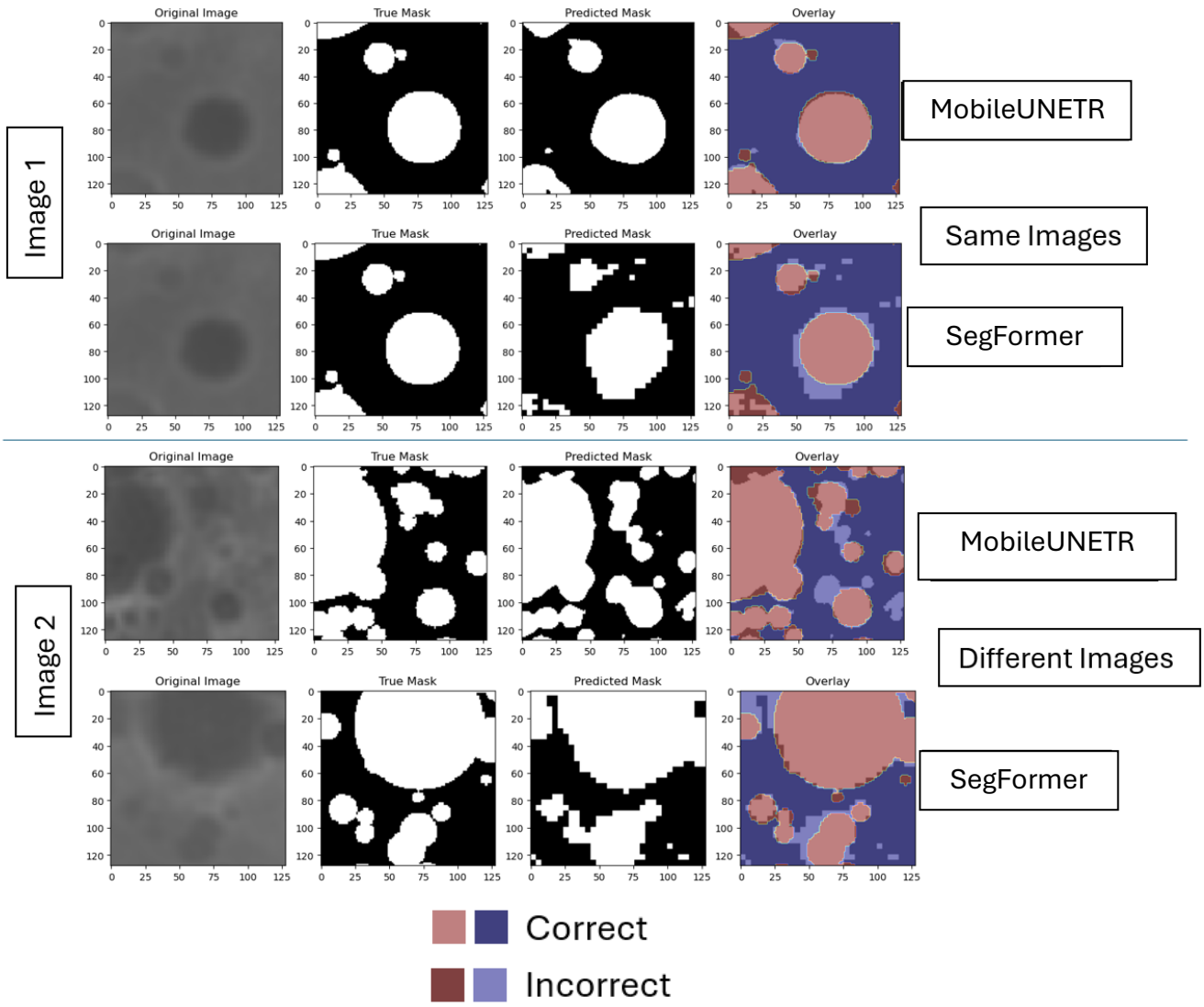

Figure 27. Dataset 1 MobileUNETR and SegFormer prediction pairs with overlap.

13.2. Dataset 2: Optical Mosaic

Dataset #2 was created out of a need to have a benchmark to determine whether Gravity Disturbance data alone, or a fusion of it, would provide better, or worse results in segmentation. This meant obtaining orthorectified optical imagery of the moon at the same segmentation, and within the same reference system. The processing of this data is described in Part 3 (Page 38). As with Dataset #1, this dataset is used to train MobileUNETR and SegFormer, using a Random Search hyperparameter fine-tuning method, and a total of 50 iterations. The analysis of these results follows the same logic as they did with Dataset #1, and so this discussion should be more to the point. A full list of the hyperparameters used in these iterations is fully references in Appendix A (Page 120), and in this section we will only reference some of them.

13.2.1. Performance Metrics Analysis

We started by observing the global aspects of our Training and Validation loss learning curves, and selected iterations corresponding to the lowest recorded Training loss. The selected models are visualized Figure 28, it reveals a problem with the corresponding Validation loss curves. The curves present for these iterations reveal an initial drop in validation loss and are followed by a very noisy behavior which doesn't appear to trend to lower values at all. This is concerning and means that the hyperparameters associated would not result in promising results. Because of this, the analysis cannot follow the same route as Dataset #1, where choosing lowest Training loss curves corresponded to strong metrics. This is even though the validation loss shown is amongst the lowest across all iterations.

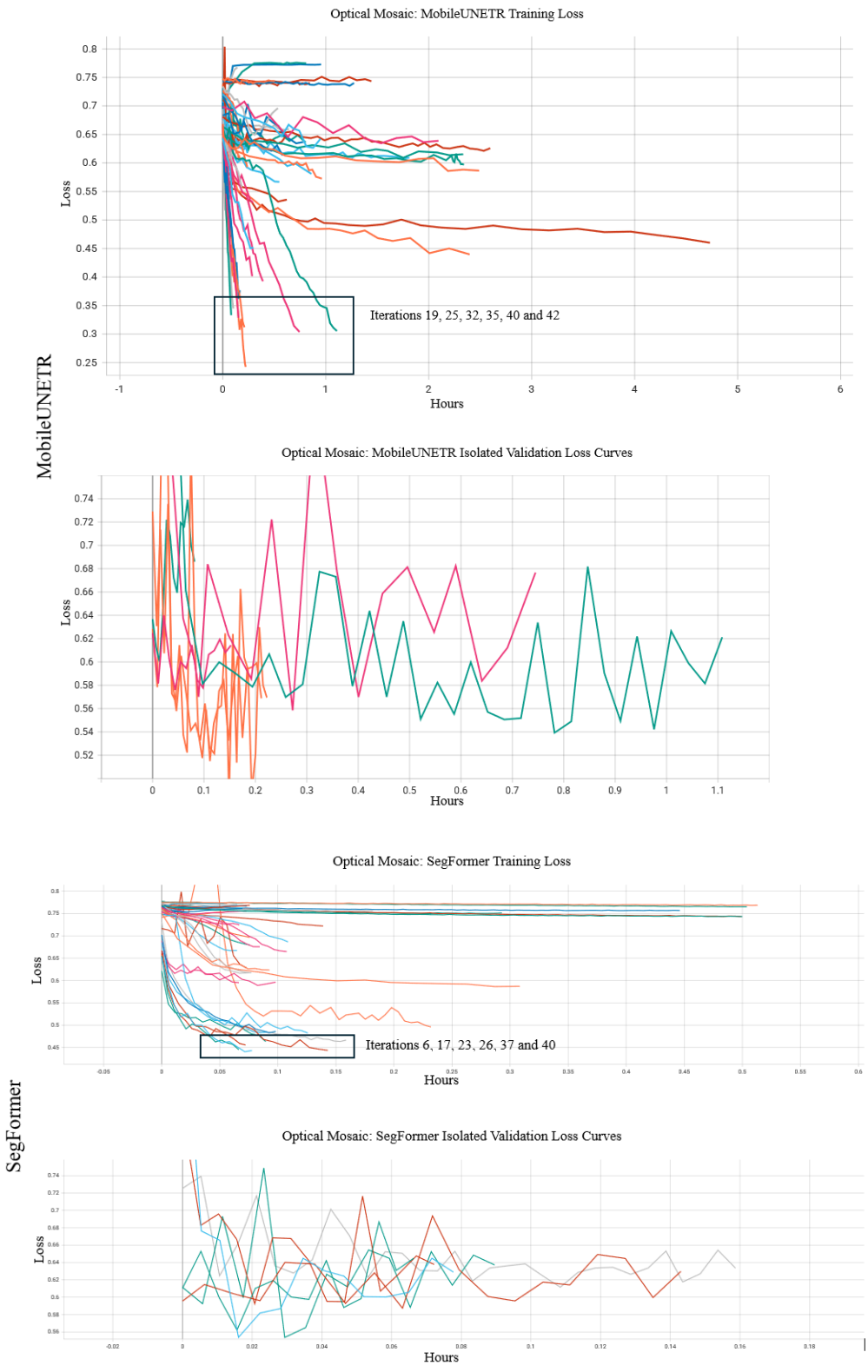

Figure 28. Dataset 2: Training and Isolated Validation Loss Curves.

The exact reason why these do not correspond with good validation loss curves is not immediately apparent but could be due to some of the already mentioned limitations. Another reason is that, in contrast to Dataset #1, the training loss drops much more sharply, which could indicate that under those conditions, the model is not able to generalize adequately. While a low loss metric is good, if the drop occurs too sharply, it could be a sign of model imbalance as evidenced by the corresponding validation loss. While it does not necessarily mean we cannot use the hyperparameters associated with these curves, it's a better idea to find an alternative way to select as these would indicate an inability to generalize, which will not give the best results. This led to observe the epochs completed for MobileUNETR and SegFormer. We found that MobileUNETR's highest epoch achieved was 40 (Iteration 23), followed by 34 (Iterations 19 and 28) with the minimum number of epochs achieved was 10 (Iterations 20, 39, 41, 46), which is the absolute minimum given that the EarlyStopping method will wait a maximum of 10 epochs to see an improvement before stopping a training session. SegFormer's iterations had 9 examples that reached 100 epochs (Iterations 1, 2, 5, 7, 8, 12, 14, 19, 30) and 2 that made it past 50 epochs (Iterations 33, 36). Of the rest, 7 complete between 21 to 27 epochs (Iterations 9, 10, 38, 39), and the rest are between 19 to 10 epochs. The distribution of completed epochs across models is very disparate, MobileUNETR's results says that the model's performance breaks down during the first half of the allotted training time, and in SegFormer, more than half (26 Iterations total) don't make it past the first 20 epochs, while a select few achieve maximum epochs. This says that in SegFormer, there are some hyperparameter configurations that could potentially obtain better results if given more time or are adjusted slightly, which is something to consider in future work. However, when looking at the class-wise IoU metrics for these iterations we found that the values for Class 0 trended towards 0, while those of Class 1 trended towards a value of .20. This could simply mean that the model's prediction under these circumstances is simply classifying most, if not all, pixels as Class 1.

Overall, there does not appear to be a very clear way of choosing hyperparameter configurations like we did with Dataset #1. Attempting to choose based on minimized training loss leads to unstable validation loss that doesn't appear to generalize well. Choosing based on learning curve stability, which is only present in SegFormer, appears to mean the model constantly attempts to classify everything as Class 1, as evidenced by the Class 0 IoU downward trend. This leaves

attempting to choose the best models based on a maximization of class-wise IoU and Overall Pixel Accuracy, as these are an indicator of how well a model is predicting pixels to be of their correct class. We begin by selecting the models with a top score in Overall Pixel Accuracy and compare them.

Table 5. Dataset 2: MobileUNETR and SegFormer results for best Overall Pixel Accuracy. Green values indicate those scores correspond with the best score across that iteration. Best metrics for each model are set in bold.

MobileUNETR				SegFormer			
Iteration	Step for top Overall Pixel Accuracy	Val Class 0 IoU at same step	Val Class 1 IoU at same step	Iteration	Step for Top Overall Pixel Accuracy	Val Class 0 IoU at same step	Val Class 1 IoU At same step
35	**557**	**.8358**	**.2761**	10	835	.8196	.1635
25	251	.8269	.1846	**26***	**215**	**.818**	**.1823**
18	241	.8238	.1874	49	608	.8187	.1533
29	179	.8245	.1372	23	179	.8158	.1815
41	241	.8249	.1253	40	434	.8171	.1374
*Originally, step 215 in Iteration 26 was chosen, but no Checkpoint for this step was found. Instead, we use step 197. This has an associated Class 1 IoU of .25 and Class 0 IoU of .7985							

Considering what we have already learned from looking at learning curve behaviors, we can see that the model has difficulty to generalize, as shown by the relatively low Class 1 IoU. A high Overall Pixel Accuracy correlates often with high Class 0 IoU, but not a high Class 1 IoU. This reinforces the idea that both models are having difficulty generalizing. We already have been made aware of our small dataset, and class imbalance, which is likely playing a role here. However, compared with the results from Dataset #1, these results look poor, which means that there is a

factor that was not present in the previous Dataset. It's the author's opinion that this factor is the learnable patterns in the data. In Dataset #1 we had gravity anomalies associated to craters, whereas in Dataset #2 we have optical imagery with down-sampled resolution, and the variety of indicators that can describe a crater visually has more variability. This increased variability in crater optical indicators could be expected to work in the model's favor, but this doesn't appear to be the case in practice. In the face of lightweight model usage, this could be a potential reason for the poorer results, as more optical variability might only be useful in models with an increased number of parameters. Notably, MobileUNETR obtained better metrics than SegFormer, which further reinforces the idea that MobileUNETR was already designed with 2 classes in mind, whereas SegFormer has 150.

Table 6. Dataset 2: Hyperparameters for MobileUNETR Iteration 35, and SegFormer Iteration 26.

MobileUNETR						
Optimizer	Seed	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?
AdamW	451007880	5e-4	40	.3	Betas= (.9,.999) AMSGrad: True	False
SegFormer						
Optimizer	Seed	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?
AdamW	9553094	3e-2	16	5e-3	Betas= (.9,.999) AMSGrad: True	False

13.2.2. Testing Results with Optimal Parameters

Because the results were difficult to observe and break down, it must be noted that these are the 'best' hyperparameters according to the author's best attempt at analyzing the results. Ideally, under these circumstances, it might be better to either (1) perform additional training with different hyperparameters, or (2) perform training for additional time. However, time constraints force our hand at having to choose from amongst the available results. A secondary reason is that the idea is to test all datasets with both models using similar training conditions, this second intention has been maintained, as no additional processing outside of what is handled by the code is performed.

*Table 7. Dataset 2: Test Metrics from optimal MobileUNETR (Run 35) and SegFormer (Run 26) models. We mark the best metrics in **bold**.*

	MobileUNETR	SegFormer
Validation Loss	.3755	**.3614**
Class 0 IoU	**.6667**	.5546
Class 1 IoU	.4556	**.4725**
Overall Accuracy	**.7394**	.6816
Mean Accuracy	**.7094**	.6808

The test results for both Hyperparameter sets are seen in Table 7, whereas the visualization of some of the images and their predictions can be found in Figure 28. Initial observations reveal a similar characteristic in SegFormer, that being that its labeling appears choppy, and is not contained very well. In addition, many nearby Class 1 features are grouped together. This is also

seen in MobileUNETR, but to a lesser extent, possibly because its labelling is smoother in comparison. Again, as it happened with Dataset #1, the overall metrics improved, but not all. Both models saw an improvement in Class 1 IoU and loss, but lost value in Class 0 IoU, Overall Accuracy, Mean Accuracy, and mean IoU. Because our Class 0 IoU decreased, this clearly meant a decrease in the rest of the metrics. We likely see an increase in Class 1 IoU because the ratio of pixels increased, and the class imbalance is less evident. But this could be a good thing, as it could be a bad thing. The same reasoning follows from Dataset #1, in that the models were trained on an imbalanced Dataset, and the Test data is not representative of the Training and Validation sets. This could inhibit a model's ability to generalize. This appears to be the case for SegFormer, where its segmentation boundaries reveals that it doesn't appear to be following some established pattern, as we can see with MobileUNETR.

Despite a visual similarity in MobileUNETR and SegFormer test metrics, the visual examination of the actual segmentation, visualized in **Error! Reference source not found.** says d ifferently. Looking at Example #2, we can observe that MobileUNETR's apparent difficulty is in the reduced visibility of some craters. While we can visually see the telltale circular patterns, these are not very strongly defined. Naturally, this is not an ideal situation, but contrasting this with SegFormer's segmentation appears to imply that it's following the light/dark differences in terrain, which is certainly a pattern in many craters, but they are circular in nature. In cases like this, SegFormer performs better than MobileUNETR in terms of raw numbers, but when considering how the segmentation is performed, it must be noted that MobileUNETR does better, despite missing more pixels.

Looking at Example #1 reveals craters that have better definition. This appears to correlate with a higher segmentation count from MobileUNETR, which correctly identifies 9 craters, most of which are the better-defined craters in the Image. Of note is the larger overall crater (Example 1, Upper Left Corner), which is clearly degraded, and its identification likely depends on a larger scale than what's available here. Although this represents many pixels in the area, the fact that it was missed by MobileUNETR is consistent with other results. Despite this, even some of the identified craters; like the one close to the center of the image, also shows some degree of degradation, and was still identified, showing that it's not out of the model's capability to account

for this. Contrasting this with SegFormer' segmentation reveals a very stark difference. Its segmentation is not as precise as MobileUNETR's, which so far, is consistent with its overall behavior.

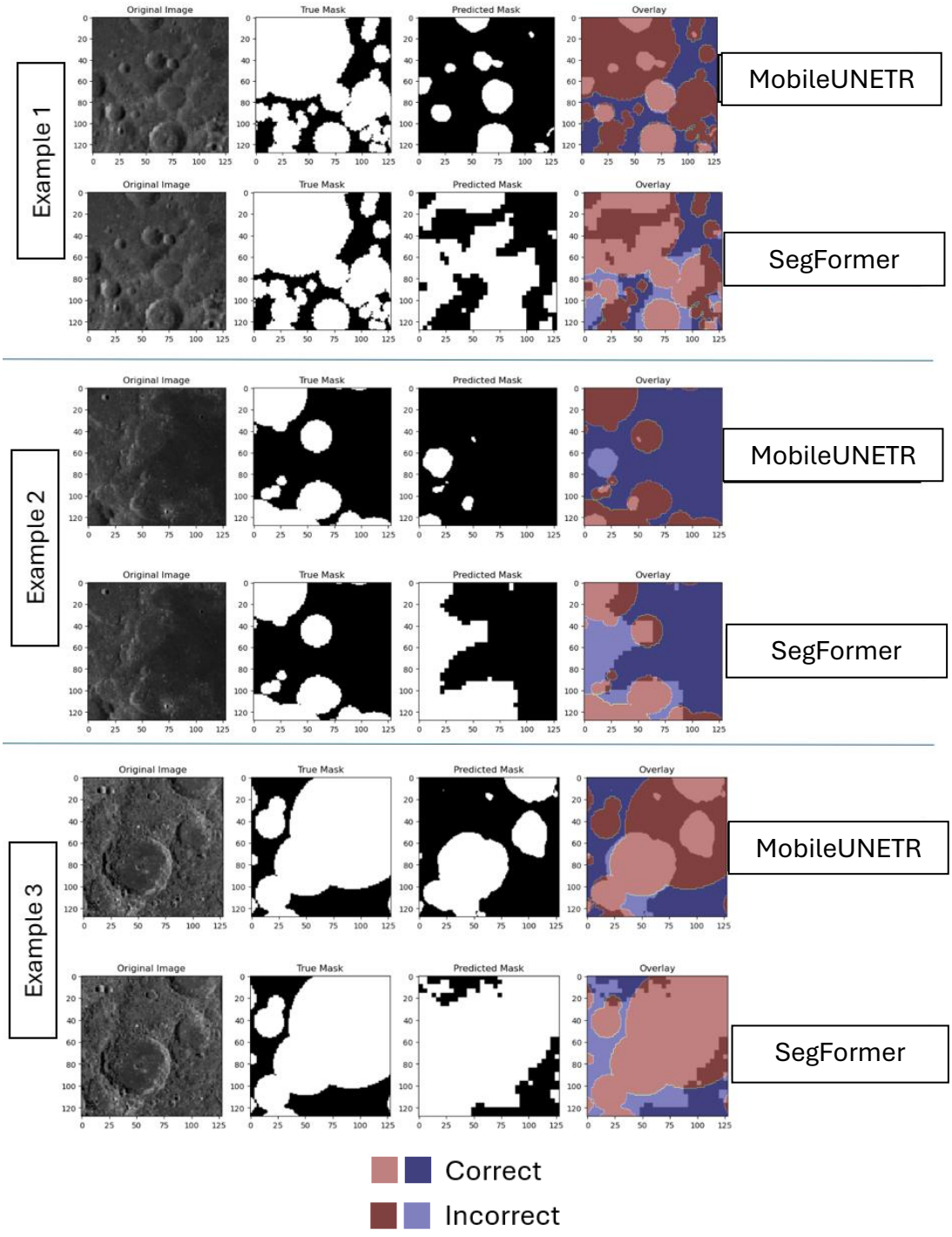

Figure 29. Dataset 2: Segmentation results comparison across three images.

13.3. Dataset 3: Gravity Optical

Dataset #3 is the early fusion of Gravity Disturbance and Optical Mosaic data and is the reason the latter was down sampled to match. It represents the final test in the work being presented here, after having discussed the results for Datasets #1 and #2. It was similarly used with a Random Search hyperparameter fine-tuning method, with the difference that this instance had a total of 100 training sessions. This is not true for MobileUNETR, who only has a total of 78 training sessions recorded. What happened to the final 32 is not certain, but one explanation would be that the computer performing the iterations was unable to finish all training sessions and this was not noticed by the researcher. That said, the results we obtained should still be representative of the model's overall performance. Everything else, including the hyperparameter space, the total range of options to choose from, remains the same. A full list of the hyperparameters used in these iterations is fully references in Appendix A (Page 120), and in this section we will only reference some of them.

13.3.1. Performance Metrics Analysis

Keeping in line with the type of analysis being performed with the previous two Datasets, the first metrics we look at are Training and Validation loss for each model. The overall Training and Validation behavior is seen in *Figure 30*. We can observe stable Training curves across both, telling us the model is learning from the Training data, but this by itself is not enough. Observations surrounding the correlated Validation curves reveal that they are very different for both models. While MobileUNETR has an overall improving but very noisy behavior, SegFormer's validation loss curves can be broken down into two overall groups. Group 1 reaches the lowest validation loss and is associated with having a lot of noise, which may indicate problems with generalization. This is not dissimilar from what we saw in Dataset #2 results, and it's important because Dataset #3 combines both data modalities. Group 2 reveals stabler curves with a less pronounced incline, and stabilizations at higher validation losses. Overall, comparing MobileUNETR with SegFormer's group 1, the curves associated with the lowest validation loss, we can see that while MobileUNETR has associated noise, there appears to be an overall improvement trend, while SegFormer does not show this. For MobileUNETR, Group 1 is composed of iterations 10, 14, 28,

34 and 28. For SegFormer, Group 1 is composed of iterations 13, 27, 33, 42, 90 and 95. SegFormer Group 2 is composed of iterations 5, 14, 35, 56, 59 and 69.

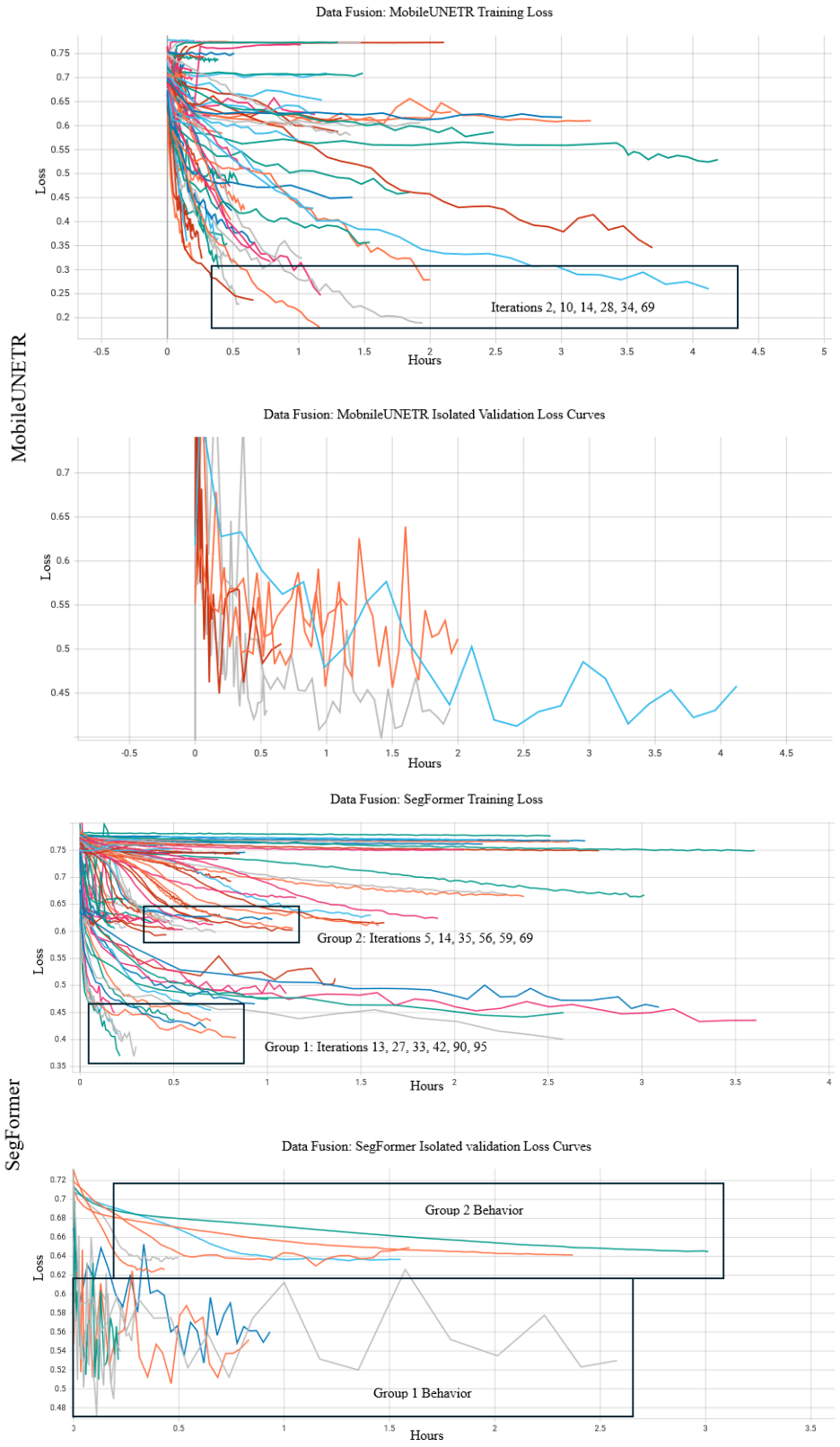

Figure 30. Dataset 3: Training and Isolated Validation Curves with Annotations

As established, it's always helpful to look past Training and Validation loss, visualized in *Figure 30*. At first glance, it appears that the better candidates for SegFormer would be from amongst Group 2, who have stabler training loss curves, and longer training times, indicating stability. When Training loss curves like SegFormer's Group 2 were observed in Dataset #2, it was revealed that those had decreasing IoU values. In *Figure 31* we visualize the Class 0 and Class 1 validation IoU for both models and their selected groups. In SegFormer's case Group 1 reveals an unsurprisingly noisy IoU metric that appears to maintain overall values of about .81 for Class 0 and .28 for Class 1, though the latter is much noisier, with few iterations appearing to near convergence. Group 2 on the other hand, reveals a very stable trend with values of .7 and .24 for Classes 0 and 1, respectively. This is in line with higher Validation loss values associated to Group 2. For MobileUNETR, we can see that Classes 0 and 1 have long term values of about .8 and .3 for Classes 0 and 1, with some of the iterations appearing to converge.

The argument made for both models is very different. Since both models have the same disadvantages of class imbalance and small datasets, we can expect the effects of these to show differently. For instance, choosing iterations with lowest Validation loss curves is an appropriate method for MobileUNETR, given that the correlating Validation loss doesn't appear to struggle beyond what is expected. The same decision leads to stagnated validation loss curves with SegFormer. Choosing loss curved with higher loss, but better stability expectedly results in lower class-wise IoU metrics. However, those curves are stabler, which is better for generalization purposes. Because of this, we choose MobileUNETR's iteration based on the validation loss, and SegFormer's based on Class 1 and Class 0 IoU stability. Choosing the model that achieves a high overall value, and maintains it, earlier in the training session is preferable. For SegFormer, this is iteration 14, which is the first to reach a stable Class 0 IoU at a value of ~.7. This correlates with a period of stability in Class 1 IoU, with a value of ~.24. There are other solutions at this same instance that have a higher Class 1 IoU, such as Iteration 35, with a value of ~.29, but this correlated with a Class 0 IoU of ~.53. Together with the fact that not soon after this peak value of ~.29 there is trade between class-wise IoU. This behavior is common across many iterations, and most

stabilize at values around ~.23. It's likely that this initial period of stability for Iteration 14 is a good solution, as it reaches the convergence values of other iterations several epochs earlier.

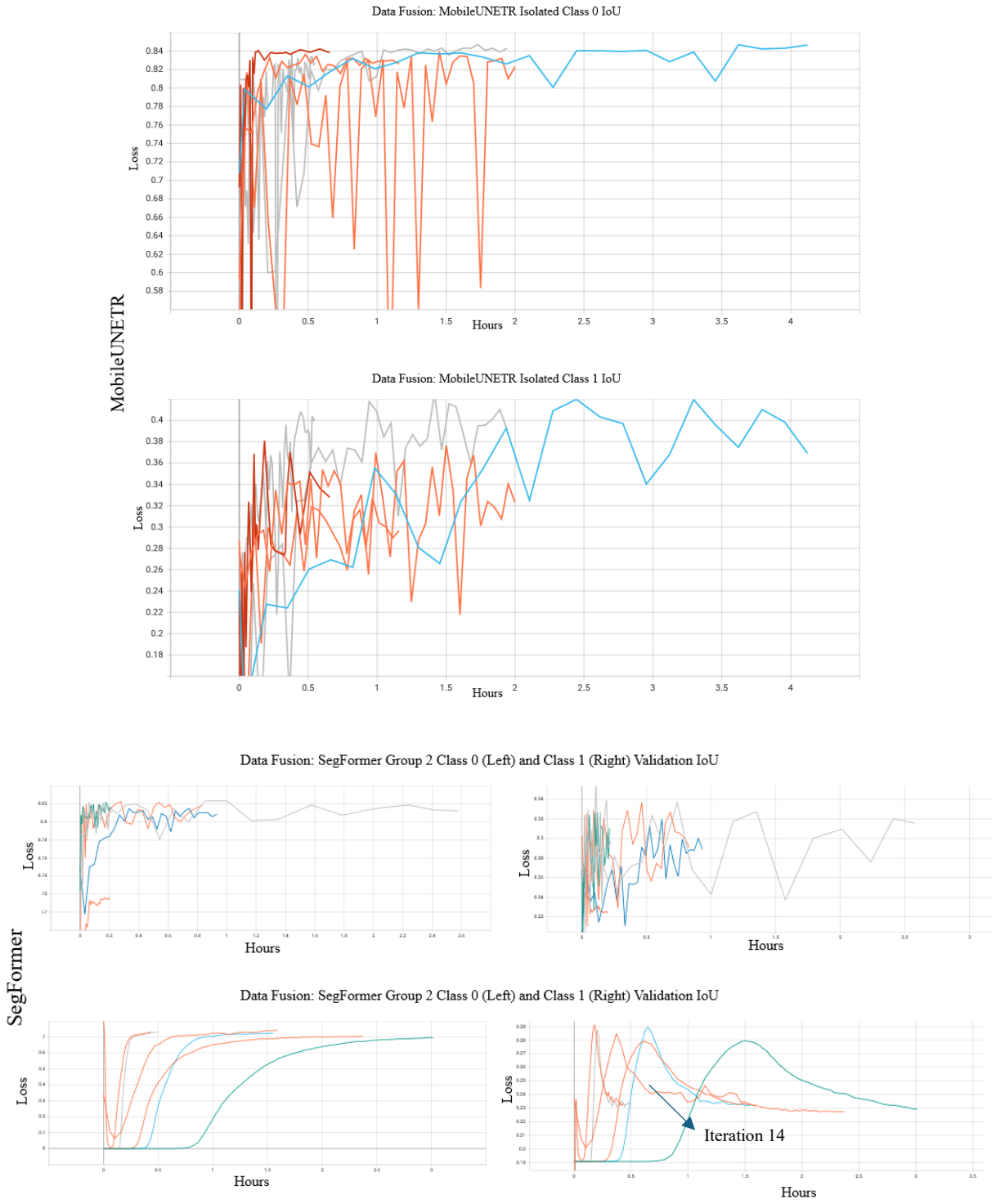

Figure 31. Dataset 3: Class-wise IoU Cross-model Comparison.

Table 8. Dataset 3: Hyperparameters for Iterations displayed in Figure 30. We mark the chosen Iterations for each model testing in bold.

MobileUNETR							
Optimizer	Iteration	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?	Seed
AdamW	2	2e-4	24	.3	Betas: (.9,.999) AMSGrad: True	False	2813218568
AdamW	10	1e-4	8	2e-3	Betas: (.9,.999) AMSGrad: True	False	50991077
AdamW	**14**	**1e-3**	**32**	**4e-2**	**Betas: (.9,.999) AMSGrad: False**	**True**	**2545580629**
AdamW	28	4e-2	16	4e-2	Betas: (.9,.999) AMSGrad: True	True	1947833320
AdamW	34	4e-4	40	.2	Betas: (.9,.999) AMSGrad: False	False	2771647456
AdamW	69	2e-2	16	5e-2	Betas: (.9,.999)	False	2547639868

						AMSGrad: True		
SegFormer								
Optimizer		Iteration	LR	Batch Size	Decay	Optimizer Parameters	Encoder Frozen?	Seed
Group 1	AdamW	13	4e-3	8	2e-2	Betas: (.9,.999) AMSGrad: True	True	2117194391
	AdamW	27	1e-2	32	2e-2	Betas: (.9,.999) AMSGrad: False	True	3768529250
	AdamW	33	5e-3	16	3e-3	Betas: (.9,.999) AMSGrad: False	True	446637793
	AdamW	42	3e-3	16	.3	Betas: (.9,.999) AMSGrad: False	True	496184434
	AdamW	90	4e-4	8	.2	Betas: (.9,.999) AMSGrad: False	True	2363435643
	AdamW	95	4e-3	8	4e-3	Betas: (.9,.999) AMSGrad: True	True	2136008925

Group 2	SGD	5	5e-3	24	3e-3	Momentum Factor: .01 Nesterov: False	True	4061407919
	AdamW	**14**	**2e-5**	**8**	**.4**	**Betas: (.9,.999) AMSGrad: True**	**True**	**1186657557**
	AdamW	35	1e-5	16	4e-3	Betas: (.9,.999) AMSGrad: True	True	3156726416
	SGD	56	3e-3	16	1e-2	Momentum Factor: .6 Nesterov: False	True	159123353
	AdamW	59	1e-5	16	.2	Betas: (.9,.999) AMSGrad: True	True	3134590206
	AdamW	69	1e-5	8	5e-2	Betas: (.9,.999) AMSGrad: True	True	1999384030

13.3.2. Testing Results with Optimal Parameters

Having discussed the overall model behavior and made the arguments for specific hyperparameters and weight conditions, we use this section to visualize the test result metrics and the segmentation maps for both models. We showcase the segmentation maps for both models under three different images, which are the same in both cases. This would provide the best objective view of how these models handle similar situations, where we attempt to have a balance between visible, degraded, large, and small craters. Figure 32 has the direct comparisons under the hyperparameters chosen in the previous section.

*Table 9. Dataset 3: Optimal test Metrics for MobileUNETR (Run 34) and SegFormer (Run 14) models. We mark the best metrics in **bold**.*

	MobileUNETR	SegFormer
Validation Loss	**.2517**	.4040
Class 0 IoU	**.7328**	.1962
Class 1 IoU	**.5994**	.4379
Overall Accuracy	**.8091**	.5057
Mean Accuracy	**.7912**	.5596

As opposed to Datasets #1 and #2, what we found was an almost complete inability by SegFormer to adapt and generalize well, despite our attempts to utilize conditions that would not result in this behavior. The reasoning behind this is unknown, but we know that SegFormer appeared to struggle with appropriate generalization and pattern adaptation in Dataset #2. This was not so much the case with Dataset #1, but there was still a visible struggle. It's possible that,

because of this, the combination of data modalities simply exacerbated the problem beyond the model's ability to learn the necessary patterns. Another thing that must be considered is that this time we deliberately avoided the learning curves with the lower Validation loss due to generalization concerns and model instability.

In the interest of being objective regarding SegFormer's performance, we returned to the originally chosen Group 1 iterations, and identified several instances that could provide better results, and hopefully, better segmentation performance. These instances were chosen on the minimization of validation loss, as this is the metric used to store iterations in the first place. We also don't observe any of the other metrics as we did in selecting the hyperparameters associated to Iteration 14. The overall results of this can be found in *Table 10*. It contains the test metrics of the instances with the best validation loss for each of the 6 iterations in SegFormer group 1. All but one of the metrics were shown to be 'best' at Iteration 42, but the differences between one and the other were around 1%-2%, which is not very significant. This could indicate that all these instances were in a similar state of chaos, as these iterations correspond to very noisy validation loss curves.

Table 10. Dataset 3: SegFormer best validation loss in Group 1.

	Iteration 13	Iteration 27	Iteration 33	Iteration 42	Iteration 90	Iteration 95
Validation Loss	.3269	.3252	**.3198**	.3206	.3362	.3246
Class 0 IoU	.5853	.5933	.5858	**.6077**	.5655	.6071
Class 1 IoU	.5085	.5105	.5164	**.5207**	.5028	.5121
Overall Accuracy	.7098	.7144	.7128	**.7249**	.6981	.7218
Mean Accuracy	.7098	.7130	.7144	**.7224**	.7012	.7176

Figure 33 offers a view into how the differences between the SegFormer Group 1 iterations differs, at least when considering their respective states where validation loss was optimized. Visually, it doesn't appear that there is a significant difference. Though not shown here, a similar comparison of different images reveals a similar pattern, where an overall segmentation mask is visually similar, with small differences that don't appear to follow any specific pattern. Despite the best attempts at achieving a strong, and visually coherent, segmentation with SegFormer, it doesn't appear that it's able to perform at its best under these conditions. Because of this, it's noted that MobileUNETR appears to be better at adapting to this dataset than SegFormer.

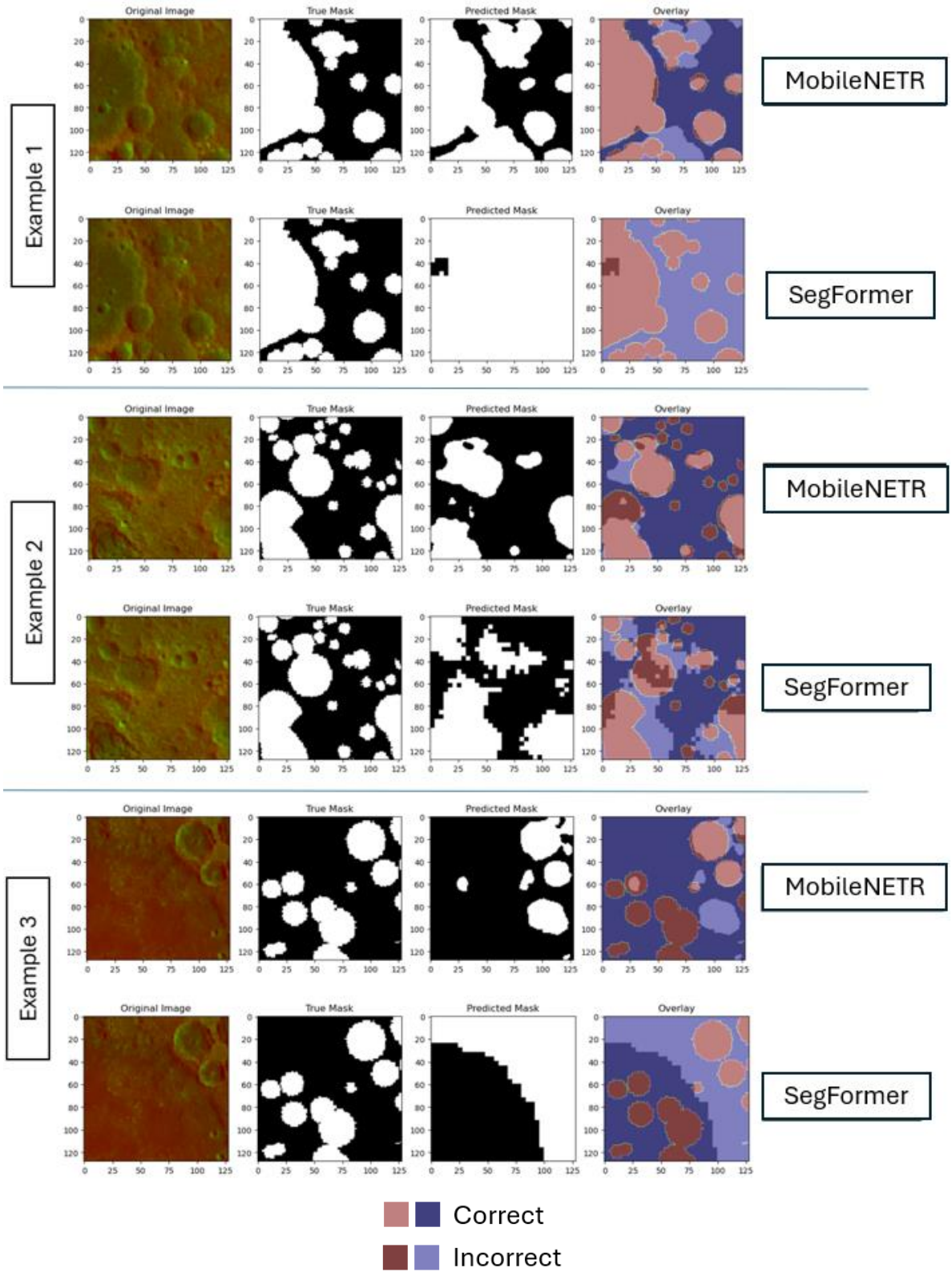

Figure 32. Dataset 3: Comparisons in segmentation predictions between MobileUNETR and SegFormer (Group 2).

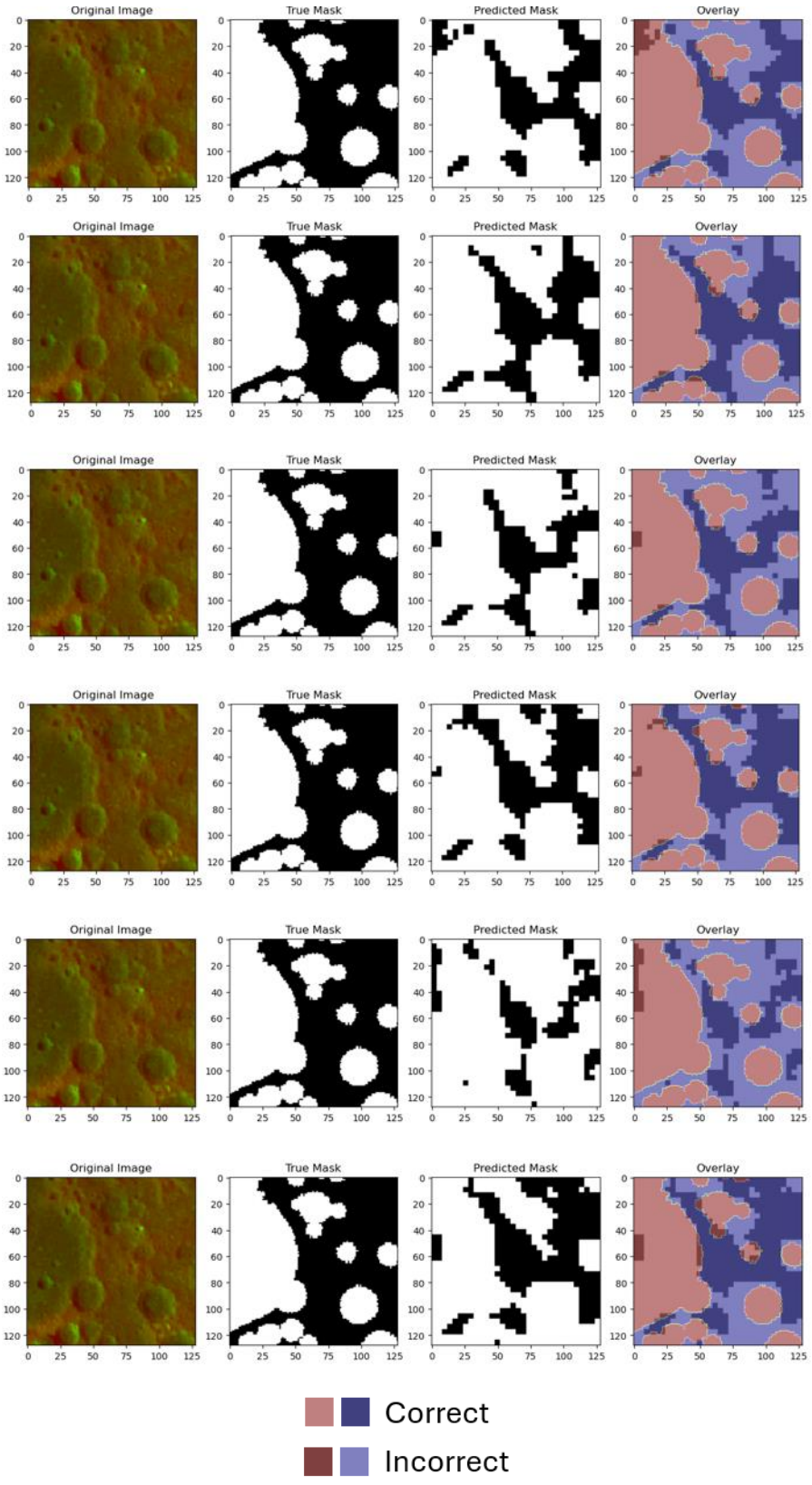

Figure 33. Dataset 3: SegFormer Group 1 Segmentation Contrast.

14. Overall Results and Conclusion

At this point we have discussed both model performances at length under different datasets, and subjected to similar conditions, except for the data modalities. Apart from the Challenges and Limitations we described, in the performance of this work several other problems were encountered, some of which arose due to author's lack of oversight. These are enumerated here:

(1) The Datasets acquired are very small.
(2) The label annotations were performed automatically and guided by Robbins' previous work, which was created based solely on optical data. Future work would necessitate careful visual inspection of labels, especially in relation to gravity data.
(3) The chosen lunar region resulted in a class imbalance, where Background pixels dominated Foreground ~4.3:1. This imbalance exists in the Training and Validation portions, but not in the Test portion of the datasets.
(4) Both models use pretrained weights based on training with images that are 512by512 pixels in size. Our images are 128by128 pixels, this was necessary due to the limited nature of the data.
(5) An oversight error led to SegFormer's encoder always being frozen, as opposed to being randomized. This means that the training conditions for both models were not 100% similar. This was only discovered after putting together the full list of hyperparameters.

These identified problems have been documented throughout the work, some of which were not known until after training had been completed. It's certainly true that these problems have limited the models' performance throughout this work. It's the author's belief that these problems don't invalidate comparisons drawn between the models' performances in the context of the data used, and the best attempt was made at drawing appropriate conclusions.

The way in which the combination of the enumerated factors affected the work varied between the models, and the datasets that were being used. Consider, for example, Datasets #1 and #2. On one hand you have a Gravity Disturbance map with identified gravity anomalies associated

to craters, and these patterns are very simple to see. In terms of visible features to look out for, there isn't much standing out other than gradual changes in the gravity field. It's possible that factors like limited training data did not impact training as severely as it could have affected Dataset #2. You have fewer features that the respective models need to recognize when considering gravity data. Optically, however, there are a larger number of issues associated with being able to observe a crater, and many of these were discussed in Part 1. Similar sized craters might be completely different depending on their age, the region they're located in, how dark or bright they appear, or whether they even have a central peak uplift, which is associated with larger craters but not always present (CLRN, n.d.; Collins, 2014; Wilhem, 1987). Considering that MobileUNETR and SegFormer are not as complex as other segmentation models, it must be recognized that when there are far more features to necessitate identification, a limited dataset would have a much greater impact. This reason is likely why the results for Dataset #1 were much better than those of Dataset #2, and even Dataset #3, to an extent.

Another way in which these issues changed the approach in analyzing model performance was the selection of hyperparameters. Dataset #1, for example, often the optimal metrics correlated with one another, where the lowest validation loss was obtained at the same instance that other metrics were being optimized across that iteration. Dataset #2 offered more difficulty, in that relying on the optimization of loss metrics proved difficult due to their instability, which led us to attempt to choose optimal states for Overall Accuracy, and individual IoU. In Dataset #3, this problem appeared to persist, but only for SegFormer. Choosing the state of lowest validation loss with MobileUNETR, did coincide with optimal states for class-wise IoU, and naturally, Overall Accuracy. For SegFormer, this resulted in very noisy validation loss curves. While we initially opted for hyperparameters associated with more stable metrics, we discovered that these resulted in worse results than anticipated. This led to an acceptance of the initially disregarded hyperparameters, although it's debatable to say whether this was an actual improvement, given the state of the visual segmentation.

Table 11. Summary of Test Metrics of each model for every Dataset.
*In bold we have the best metrics per Dataset. In **green** we highlight the best metrics across both models and all 3 Datasets, and in red we highlight the worst metrics.*

	Dataset 1		Dataset 2		Dataset 3	
	MobileUNETR	SegFormer	MobileUNETR	SegFormer	MobileUNETR	SegFormer
Validation Loss	**.2562**	.2820	.3755	**.3614**	**.2517**	.3206
Class 0 IoU	**.7289**	.6531	**.6667**	.5546	**.7328**	.6077
Class 1 IoU	**.5940**	.5627	.4556	**.4725**	**.5994**	.5207
Overall Accuracy	**.8059**	.7601	**.7394**	.6816	**.8091**	.7249
Mean Accuracy	**.7879**	.7560	**.7094**	.6808	**.7912**	.7224

Table 11 summarizes the results of what are considered the best results across Datasets with both models. One immediate thing to notice about this is that, according to the Test metrics, the best results often came from MobileUNETR. The only exception to this is Dataset #2, where SegFormer obtained better Validation loss and Class 1 IoU. This appears to indicate that in this experiment, MobileUNETR appears to be better in generalizing and adapting to the patterns in the data. It's been previously mentioned that MobileUNETR was created with medical imaging in mind, particularly skin lessons as those see in in ISIC datasets. SegFormer was created as a general use dataset, and its weights are derived from the ADE20K dataset. On one hand, skin lessons represent a very specific application, and when contrasting this to craters one can see a slight resemblance. This is especially true for Dataset #1, which consists of gravity data, aside from the shapes not always being circular. On the other hand, SegFormer pretrained with ADE20K was fine-tunes to look for objects that were very different from what was given to it. The expectation going into this work was that SegFormer would perform just as well as MobileUNETR due to the simple patterns associated with craters. While it was true with Dataset #1, which did contain very

simple shapes, patterns and features, the optical variability of craters likely spearheaded the results seen in the other Datasets, though this could be studied more in-depth.

One reason for applying multimodal data to the segmentation of craters was driven primarily because (1) Optically, craters are easier to identify than through gravity, and (2) In some cases, a crater may not have a detectable gravity anomaly, and still be clearly visible in an optical modality. This is something that was discussed as an error, or limitation in how the labelled datasets were created, because Robbins based his observations from high-resolution optical data. The author simply took these observations and used them to label the appropriate gravity disturbance product without verifying whether each crater had an identifiable anomaly. Because of this fact, it was the expectation that when considered together, the optical data would help the model identify craters that would've been otherwise missed due to having a nigh detectable gravity anomaly. A second expectation was that their robustness would increase when considering both data modalities together, as the models would learn to associate the gravity anomaly with the optical features that identify an impact crater.

The analysis of Dataset #3 results revealed two different behaviors between MobileUNETR and SegFormer. While MobileUNETR, in accordance with our expectations, did perform slightly better than did it compared to Dataset #1, SegFormer did not, although it didn't see worse performance than Dataset #2. We didn't investigate the improvement of MobileUNETR's Dataset #3 performance well enough to say with any definitive answer if this improvement is represented by the mutual interaction expected of the two datasets, where Optical aided gravity observations and vice-versa. This result could simply be a matter of coincidence, where it didn't necessarily do better or worse, and only appears to have done better. This was the opposite behavior when looking at SegFormers' Dataset #3 results, where it still underperformed when compared to Dataset #1, but not Dataset #2. This leads us to say that, under these conditions, MobileUNETR proved to have an overall better performance than SegFormer, and while some reasons for this have been theorized, it's difficult to say so with absolute certainty. Furthermore, it appears the combination of Gravity and Optical data can result in a positive trend, although perhaps our method of Data Fusion was not the most appropriate in this instance. As this work was intended to be mainly exploratory, it must be noted that this aspect requires further study.

15. Potential for Future Work

In the process of this work there were several issues identified, most of which were discussed throughout the preceding chapters. Some of these problems are related to the type and quantity of data used, which merit alternative methods to solve appropriately. Additionally, some of the limitations encountered, such as the class imbalance that was not addressed, arose due to author oversight. While lessons have been learned, we recognize the importance of reflecting on these and maintaining note of how they may be better resolved in future work. We enumerate a small list of actions that could be taken in the interest of addressing these problems below:

(1) Explore and apply different Machine Learning methods designed to address class imbalances in data.
(2) Create a manually labelled dataset specifically for Gravity data.
(3) Utilize additional gravity data sources to expand the dataset with additional examples, such as using NASA's Mars Gravity Model
(4) Pretrain a model on an existing crater detection dataset, even if this happens to be an optical-based dataset.
(5) Explore different methods of data fusion.
(6) Explore the creation of a Deep Learning architecture to improve segmentation of crater formations. This could include taking any of the previously used models and modify it.

It must be noted that this list is not all-inclusive, and it's the author's intention to continue exploring approaches to improve the methodology associated with this work.

References

Altamimi, Z., Sillard, P., & Boucher, C. (2002). ITRF2000: A new release of the International Terrestrial Reference Frame for earth science applications. *Journal of Geophysical Research: Solid Earth*, *107*(B10). https://doi.org/10.1029/2001JB000561

Anaconda, Inc. (2024). Anaconda | The Operating System for AI. Retrieved November 22, 2024, from https://www.anaconda.com/

Arnold, C., Biedebach, L., Küpfer, A., & Neunhoeffer, M. (2024). The role of hyperparameters in machine learning models and how to tune them. *Political Science Research and Methods*, *12*(4), 841–848. https://doi.org/10.1017/psrm.2023.61

Ashtari, P., Sima, D. M., Lathauwer, L. D., Sappey-Marinier, D., Maes, F., & Huffel, S. V. (2023). Factorizer: A Scalable Interpretable Approach to Context Modeling for Medical Image Segmentation. *Medical Image Analysis*, *84*, 102706. https://doi.org/10.1016/j.media.2022.102706

Barlow, N. G., Mest, S. C., Gibbs, V. B., Kinser, R. M., & Gundy, A. (2012). COMPILATION OF A GLOBAL GIS CRATER DATABASE FOR THE MOON.

Bertone, S., Arnold, D., Girardin, V., Lasser, M., Meyer, U., & Jäggi, A. (2021). Bertone 2021 Assessing Reduced-Dynamic Parametrizations for GRAIL Orbit Determination and the.pdf. American Geophysical Union.

Chandnani, M., & Herrick, R. R. (2022, May 11). Influence of Target Properties on Wall Slumping in Lunar Impact Craters within the Simple-to-Complex Transition. https://doi.org/10.1002/essoar.10511317.1

Chen, M., Lei, M., Liu, D., Zhou, Y., Zhao, H., & Qian, K. (2017). Morphological Features-Based Descriptive Index System for Lunar Impact Craters. *ISPRS International Journal of Geo-Information*, *7*(1), 5. https://doi.org/10.3390/ijgi7010005

Chen, M., Liu, D., Qian, K., Li, J., Lei, M., & Zhou, Y. (2018). Lunar Crater Detection Based on Terrain Analysis and Mathematical Morphology Methods Using Digital Elevation Models. *IEEE Transactions on Geoscience and Remote Sensing*, *56*(7), 3681–3692. https://doi.org/10.1109/TGRS.2018.2806371

Chen, Z., & Chen, Z. (2022). The Identification of Impact Craters from GRAIL-Acquired Gravity Data by U-Net Architecture. *Remote Sensing*, *14*(12), 2783. https://doi.org/10.3390/rs14122783

CLRN. (n.d.). Impact Crater Formation and Morphology | Canadian Lunar Research Network. Retrieved November 14, 2024, from https://clrn.uwo.ca/clrn-research/impact-crater-formation-and-morphology/

Collins, G. S. (2014). Numerical simulations of impact crater formation with dilatancy. *Journal of Geophysical Research: Planets*, *119*(12), 2600–2619. https://doi.org/10.1002/2014JE004708

Dawei, L. (2017). Lunar Landscape: Grabens. In B. Cudnik (Ed.), *Encyclopedia of Lunar Science* (pp. 1–3). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-05546-6_50-1

Degirmenci, M., & Ashyralyev, S. (2010). Impact Crater Detection on Mars Digital Elevation and Image Model.

DeLatte, D. M., Crites, S. T., Guttenberg, N., Tasker, E. J., & Yairi, T. (2018). EXPERIMENTS IN SEGMENTING MARS CRATERS USING CONVOLUTIONAL NEURAL NETWORKS.

DeLatte, D. M., Crites, S. T., Guttenberg, N., & Yairi, T. (2019). Automated crater detection algorithms from a machine learning perspective in the convolutional neural network era. *Advances in Space Research*, *64*(8), 1615–1628. https://doi.org/10.1016/j.asr.2019.07.017

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2021, June 3). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv. https://doi.org/10.48550/arXiv.2010.11929

Evans, A. J., Soderblom, J. M., Andrews-Hanna, J. C., Solomon, S. C., & Zuber, M. T. (2016). Identification of buried lunar impact craters from GRAIL data and implications for the nearside maria. *Geophysical Research Letters*, *43*(6), 2445–2455. https://doi.org/10.1002/2015GL067394

Finkelstein, M., Baby, S. M., & Kitano, H. (n.d.). Automatic Lunar Crater Detection from Optical Images and Elevation Maps, 7.

Flechtner, F., Reigber, C., Rummel, R., & Balmino, G. (2021). Satellite Gravimetry: A Review of Its Realization. *Surveys in Geophysics*, *42*(5), 1029–1074. https://doi.org/10.1007/s10712-021-09658-0

Galloway, M. J., Benedix, G. K., Bland, P. A., Paxman, J., Towner, M. C., & Tan, T. (2014). Automated crater detection and counting using the hough transform. In *2014 IEEE International Conference on Image Processing (ICIP)* (pp. 1579–1583). Paris, France: IEEE. https://doi.org/10.1109/ICIP.2014.7025316

Goossens, S., Lemoine, F. G., Sabaka, T. J., Nicholas, J. B., Mazarico, E., Rowlands, D. D., et al. (2016). A Global Degree and Order 1200 Model of the Lunar Gravity Field Using GRAIL Mission Data, 1484. Presented at the 47th Annual Lunar and Planetary Science Conference.

Greeley, R. (1971). Lava tubes and channels in the lunar Marius Hills. *The Moon*, *3*(3), 289–314. https://doi.org/10.1007/BF00561842

Greeley, R., Christensen, P. R., & McHone, J. F. (1987). Radar characteristics of small craters: Implications for Venus. *Earth, Moon, and Planets*, *37*(1), 89–111. https://doi.org/10.1007/BF00054327

Guo, J. (2023). *Physical geodesy: A theoretical introduction* (2023rd ed., Vol. 1). Springer International Publishing.

Guo, Yanming, Liu, Y., Georgiou, T., & Lew, M. S. (2018). A review of semantic segmentation using deep neural networks. *International Journal of Multimedia Information Retrieval*, *7*(2), 87–93. https://doi.org/10.1007/s13735-017-0141-z

Guo, Yue, Wu, H., Yang, S., & Cai, Z. (2023). Crater-DETR: A Novel Transformer Network for Crater Detection Based on Dense Supervision and Multiscale Fusion. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., et al. (2023). A Survey on Vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(1), 87–110. https://doi.org/10.1109/TPAMI.2022.3152247

Haruyama, J., Hioki, K., Shirao, M., Morota, T., Hiesinger, H., Van Der Bogert, C. H., et al. (2009). Possible lunar lava tube skylight observed by SELENE cameras. *Geophysical Research Letters*, *36*(21), 2009GL040635. https://doi.org/10.1029/2009GL040635

He, J., Cui, H., & Feng, J. (2010). Edge information based crater detection and matching for lunar exploration. In *2010 International Conference on Intelligent Control and Information Processing* (pp. 302–307). Dalian, China: IEEE. https://doi.org/10.1109/ICICIP.2010.5565284

Head, J. W., Fasset, C. I., Kadish, S. J., Smith, D. E., Zuber, M. T., Neumann, G. A., & Mazarico, E. (2010). Global distribution of large lunar craters: Implications for resurfacing and impactor populations.

Heiken, G. H., Vaniman, D. T., & French, B. M. (1991). Lunar Sourcebook A users guide to the moon.pdf. Cambridge University Press.

Hochreiter, S., & Schmidhuber, J. (1997). Hochreiter 1997 Long Short Term Memory.pdf.

Horvath, T., Hayne, P. O., & Paige, D. A. (2022). Thermal and Illumination Environments of Lunar Pits and Caves: Models and Observations From the Diviner Lunar Radiometer Experiment. *Geophysical Research Letters*, *49*(14), e2022GL099710. https://doi.org/10.1029/2022GL099710

Huang, Q., Xiao, Z., & Xiao, L. (2014). Subsurface structures of large volcanic complexes on the nearside of the Moon: A view from GRAIL gravity. *Icarus*, *243*, 48–57. https://doi.org/10.1016/j.icarus.2014.09.009

Iz, H., Shum, C., Ding, X., & Dai, C. (2011). Orientation of the Geometrically Best fitting Triaxial Lunar Ellipsoid with Respect to the Mean Earth/Polar Axis Reference Frame. *Journal of Geodetic Science*, *1*(1), 52–58. https://doi.org/10.2478/v10156-010-0007-2

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, *31*(3), 264–323. https://doi.org/10.1145/331499.331504

Jekeli, C. (2000). Heights, the Geopotential and Vertical Datums: Technical Report.

Kaku, T., Haruyama, J., Miyake, W., Kumamoto, A., Ishiyama, K., Nishibori, T., et al. (2017). Detection of Intact Lava Tubes at Marius Hills on the Moon by SELENE (Kaguya)

Lunar Radar Sounder. *Geophysical Research Letters*, *44*(20). https://doi.org/10.1002/2017GL074998

Khan, A., Pommier, A., Neumann, G. A., & Mosegaard, K. (2013). The lunar moho and the internal structure of the Moon: A geophysical perspective. *Tectonophysics*, *609*, 331–352. https://doi.org/10.1016/j.tecto.2013.02.024

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, *26*(3), 159–190. https://doi.org/10.1007/s10462-007-9052-3

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

Lemoine, F. G., Goossens, S., Sabaka, T. J., Nicholas, J. B., Mazarico, E., Rowlands, D. D., et al. (2013). High–degree gravity models from GRAIL primary mission data. *Journal of Geophysical Research: Planets*, *118*(8), 1676–1698. https://doi.org/10.1002/jgre.20118

Lemoine, F. G., Goossens, S., Sabaka, T. J., Nicholas, J. B., Mazarico, E., Rowlands, D. D., et al. (2014). GRGM900C: A degree 900 lunar gravity model from GRAIL primary and extended mission data. *Geophysical Research Letters*, *41*(10), 3382–3389. https://doi.org/10.1002/2014GL060027

Li, K. (2013). Study on Smal-Scale Lunar Craters' Morphology and Degradation. *Wuhan University*.

Lightning AI. (2024). Welcome to ⚡ PyTorch Lightning — PyTorch Lightning 2.4.0 documentation. Retrieved November 22, 2024, from https://lightning.ai/docs/pytorch/stable/

Lopes, R. H. C., Reid, I., & Hobson, P. R. (2007). The two-dimensional Kolmogorov-Smirnov test.

Lowrie, W. (1997). *Fundamentals of Geophysics* (7th ed.). Institute of Geophysics: Press Syndicate of the University of cambridge.

Ma, H., Xu, C., Shen, Z., Yu, C., & Li, Y. (2018). Application of Machine Learning Techniques for Clinical Predictive Modeling: A Cross-Sectional Study on Nonalcoholic Fatty Liver Disease in China. *BioMed Research International*, *2018*, 1–9. https://doi.org/10.1155/2018/4304376

MapTiler team. (2024). EPSG.io: Coordinate Systems Worldwide. Retrieved December 10, 2024, from https://epsg.io

Monica, & Agrawal, P. (2024). A Survey on Hyperparameter Optimization of Machine Learning Models. In *2024 2nd International Conference on Disruptive Technologies (ICDT)* (pp. 11–15). Greater Noida, India: IEEE. https://doi.org/10.1109/ICDT61202.2024.10489732

Nakamura, Y. (1982). Lunar Seismic data Analysis.pdf. NASA.

NASA. (2024). Welcome to the Planetary Data System. Retrieved November 22, 2024, from https://pds.nasa.gov/

O'Shea, K., & Nash, R. (2015, December 2). An Introduction to Convolutional Neural Networks. arXiv. Retrieved from http://arxiv.org/abs/1511.08458

Paul, S., & Chen, P.-Y. (2022). Vision Transformers Are Robust Learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*(2), 2071–2081. https://doi.org/10.1609/aaai.v36i2.20103

Perera, S., Erzurumlu, Y., Gulati, D., & Yilmaz, A. (2024, September 4). MobileUNETR: A Lightweight End-To-End Hybrid Vision Transformer For Efficient Medical Image Segmentation. arXiv. Retrieved from http://arxiv.org/abs/2409.03062

Pieters, C. M., Taylor, L. A., Noble, S. K., Keller, L. P., Hapke, B., Morris, R. V., et al. (2000). Space weathering on airless bodies: Resolving a mystery with lunar samples. *Meteoritics & Planetary Science*, *35*(5), 1101–1107. https://doi.org/10.1111/j.1945-5100.2000.tb01496.x

Purwono, P., Ma'arif, A., Rahmaniar, W., Fathurrahman, H. I. K., Frisky, A. Z. K., & Haq, Q. M. U. (2023). Understanding of Convolutional Neural Network (CNN): A Review. *International Journal of Robotics and Control Systems*, *2*(4), 739–748. https://doi.org/10.31763/ijrcs.v2i4.888

Python Software Foundation. (2024). Python Release Python 3.12.5. Retrieved November 22, 2024, from

QGIS Dev Team. (2024). Spatial without Compromise · QGIS Web Site. Retrieved November 22, 2024, from https://qgis.org/

Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., & Dosovitskiy, A. (2021). Do Vision Transformers See Like Convolutional Neural Networks?

Ramachandram, D., & Taylor, G. W. (2017). Deep Multimodal Learning: A Survey on Recent Advances and Trends. *IEEE Signal Processing Magazine*, *34*(6), 96–108. https://doi.org/10.1109/MSP.2017.2738401

Robbins, S. J. (2019). A New Global Database of Lunar Impact Craters >1–2 km: 1. Crater Locations and Sizes, Comparisons With Published Databases, and Global Analysis. *Journal of Geophysical Research: Planets*, *124*(4), 871–892. https://doi.org/10.1029/2018JE005592

Robbins, S. J., Antonenko, I., Kirchoff, M. R., Chapman, C. R., Fassett, C. I., Herrick, R. R., et al. (2014). The variability of crater identification among expert and community crater analysts. *Icarus*, *234*, 109–131. https://doi.org/10.1016/j.icarus.2014.02.022

Robinson, M. S., Brylow, S. M., Tschimmel, M., Humm, D., Lawrence, S. J., Thomas, P. C., et al. (2010). Lunar Reconnaissance Orbiter Camera (LROC) Instrument Overview. *Space Science Reviews*, *150*(1–4), 81–124. https://doi.org/10.1007/s11214-010-9634-2

Saheba, S. M., Upadhyaya, T. K., & Sharma, R. K. (2016). Lunar surface crater topology generation using adaptive edge detection algorithm. *IET Image Processing*, *10*(9), 657–661. https://doi.org/10.1049/iet-ipr.2015.0232

Sharma, B., & Schultz, K. (2018). How Tall is Mount Everest? For Nepal, It's Touchy Question. *New York Times*. Retrieved from https://link.gale.com/apps/doc/A526346695/AONE?u=anon~ce4ec67a&sid=googleScholar&xid=efb6cdcf

Smith, D. E., Zuber, M. T., Neumann, G. A., Mazarico, E., Lemoine, F. G., Head Iii, J. W., et al. (2017). Summary of the results from the lunar orbiter laser altimeter after seven years in lunar orbit. *Icarus*, *283*, 70–91. https://doi.org/10.1016/j.icarus.2016.06.006

Spreyerer, E. (2023, May 23). Impact Site of the HAKUTO-R Mission 1 Lunar Lander. Retrieved November 14, 2024, from https://www.lroc.asu.edu/images/1302

TensorFlow Team. (2024). TensorBoard. Retrieved November 22, 2024, from https://www.tensorflow.org/tensorboard

Tewari, A., Prateek, K., Singh, A., & Khanna, N. (2023, September 28). Deep Learning based Systems for Crater Detection: A Review. arXiv. Retrieved from http://arxiv.org/abs/2310.07727

Ullrich, G. W. (1976). The Mechanics of Central Peak Formation in Shock Wave Cratering Events, 140.

USGS. (2024). Astrogeology Cloud Processing. English. Retrieved from https://astrocloud.wr.usgs.gov/index.php?view=map2

Vajda, P., Vanı ˇcek, P., & Meurers, B. (2004). On the removal of the effect of topography on gravity disturbance in gravity data inversion or interpretation. *Contributions to Geophysics and Geodesy*, *34*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is All you Need.

Wagner, R. V., & Robinson, M. S. (2022). Lunar Pit Morphology: Implications for Exploration. *Journal of Geophysical Research: Planets*, *127*(8), e2022JE007328. https://doi.org/10.1029/2022JE007328

Wagner, Robert V., & Robinson, M. S. (2014). Distribution, formation mechanisms, and significance of lunar pits. *Icarus*, *237*, 52–60. https://doi.org/10.1016/j.icarus.2014.04.002

Wang, Jiao, Cheng, W., & Zhou, C. (2015). A Chang'E-1 global catalog of lunar impact craters. *Planetary and Space Science*, *112*, 42–45. https://doi.org/10.1016/j.pss.2015.04.012

Wang, Juntao, Kreslavsky, M. A., Liu, J., Head, J. W., Zhang, K., Kolenkina, M. M., & Zhang, L. (2020). Quantitative Characterization of Impact Crater Materials on the Moon: Changes in Topographic Roughness and Thermophysical Properties With Age. *Journal of Geophysical Research: Planets*, *125*(10), e2019JE006091. https://doi.org/10.1029/2019JE006091

Wang, Pengxiang, Xiao, K., & Zhou, L. (2024). Research on feature coding theory and typical application analysis in machine learning algorithms. *Applied and Computational Engineering*, *32*(1), 85–92. https://doi.org/10.54254/2755-2721/32/20230188

Wang, Pin, Fan, E., & Wang, P. (2021). Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. *Pattern Recognition Letters*, *141*, 61–67. https://doi.org/10.1016/j.patrec.2020.07.042

Wilhem, D. E. (1987). The Geologic History of the Moon.pdf. U.S Government Printing Office, Washington.

Wöhler, C., Arnaut, M., & Bhatt, M. (2024). Multiband Spectropolarimetry of Lunar Maria, Pyroclastics, Fresh Craters, and Swirl Materials. *The Astronomical Journal*, *167*(5), 187. https://doi.org/10.3847/1538-3881/ad2f2f

Wu, B., Li, F., Hu, H., Zhao, Y., Wang, Y., Xiao, P., et al. (2020). Topographic and Geomorphological Mapping and Analysis of the Chang'E-4 Landing Site on the Far Side of the Moon. *Photogrammetric Engineering & Remote Sensing*, *86*(4), 247–258. https://doi.org/10.14358/PERS.86.4.247

Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., & Luo, P. (2021, October 28). SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. arXiv. Retrieved from http://arxiv.org/abs/2105.15203

Xie, M., Zhu, M. H., Xiao, Z., Wu, Y., & Xu, A. (2017). Xie 2017 Effect of Topography Degradation on Crater Size-Frequency Distributions.pdf.

Yousef, M., & Allmer, J. (Eds.). (2014). *miRNomics: MicroRNA Biology and Computational Analysis* (Vol. 1107). Totowa, NJ: Humana Press. https://doi.org/10.1007/978-1-62703-748-8

Yu, M., Cui, H., & Tian, Y. (2014). A new approach based on crater detection and matching for visual navigation in planetary landing. *Advances in Space Research*, *53*(12), 1810–1821. https://doi.org/10.1016/j.asr.2013.04.011

Zhao, F., Zuo, W., & Li, C. (2023). Improvement of Lunar Surface Dating Accuracy Utilizing Crater Degradation Model: A Case Study of the Chang'e-5 Sampling Area. *Remote Sensing*, *15*(9), 2463. https://doi.org/10.3390/rs15092463

Zuber, M. T., Smith, D. E., Lehman, D. H., Hoffman, T. L., Asmar, S. W., & Watkins, M. M. (2013). Gravity Recovery and Interior Laboratory (GRAIL): Mapping the Lunar Interior from Crust to Core. *Space Science Reviews*, *178*(1), 3–24. https://doi.org/10.1007/s11214-012-9952-7

Zuo, W., Li, C., Yu, L., Zhang, Z., Wang, R., Zeng, X., et al. (2019). Shadow–highlight feature matching automatic small crater recognition using high-resolution digital orthophoto map from Chang'E Missions. *Acta Geochimica*, *38*(4), 541–554. https://doi.org/10.1007/s11631-019-00356-8

Appendix A: Tables with Hyperparameter selections for each Iteration across all models.

Table 12. Dataset 1: MobileUNETR Hyperparameters ordered by Optimizer selection

Iteration:	Optimizer	LR	Batch Size:	Weight Decay Factor:	Nesterov	Momentum Factor	Freeze Encoder?	Seed
version_3	SGD	0.0004	32	0.1	TRUE	0.9	TRUE	3160330609
version_6	SGD	3.00E-05	24	0.2	TRUE	0.04	FALSE	2590440101
version_15	SGD	0.0004	40	0.3	TRUE	0.02	FALSE	3419115976
version_21	SGD	0.02	8	0.4	TRUE	0.4	TRUE	875899372
version_23	SGD	0.0002	16	0.003	TRUE	0.05	FALSE	2672782426
version_25	SGD	3.00E-05	16	0.04	TRUE	0.07	TRUE	2668122319
version_27	SGD	0.0004	16	0.001	TRUE	0.04	FALSE	3532976774
version_29	SGD	0.0004	24	0.004	FALSE	0.8	FALSE	3144583547
version_30	SGD	0.0002	24	0.5	FALSE	0.2	FALSE	2757725213
version_31	SGD	0.001	8	0.002	TRUE	0.6	FALSE	2207091799
version_41	SGD	0.004	16	0.005	FALSE	0.03	TRUE	511952957
version_42	SGD	0.0002	24	0.1	TRUE	0.09	FALSE	978062988
version_43	SGD	0.003	8	0.4	TRUE	0.03	TRUE	865409610
version_44	SGD	2.00E-05	40	0.004	FALSE	0.9	TRUE	3924237756
version_46	SGD	0.04	24	0.005	TRUE	0.3	FALSE	3534356937
version_48	SGD	0.003	16	0.004	TRUE	0.6	FALSE	1655156371
version_1	Adam	2.00E-05	8	0.3	N/A	(0.9, 0.999)	FALSE	4013674510
version_8	Adam	0.04	32	0.03	N/A	(0.9, 0.999)	TRUE	2271946616
version_9	Adam	0.02	16	0.01	N/A	(0.9, 0.999)	TRUE	618435319
version_10	Adam	0.0002	24	0.05	N/A	(0.9, 0.999)	TRUE	4229589396

version_12	Adam	0.001	32	0.2	N/A	(0.9, 0.999)	FALSE	2167922276
version_13	Adam	0.0004	16	0.01	N/A	(0.9, 0.999)	TRUE	671243131
version_17	Adam	0.0004	24	0.003	N/A	(0.9, 0.999)	TRUE	1657827273
version_20	Adam	0.05	8	0.4	N/A	(0.9, 0.999)	TRUE	182025129
version_22	Adam	0.05	24	0.001	N/A	(0.9, 0.999)	TRUE	3595724627
version_26	Adam	4.00E-05	16	0.5	N/A	(0.9, 0.999)	FALSE	2246152837
version_34	Adam	3.00E-05	8	0.003	N/A	(0.9, 0.999)	TRUE	180387573
version_36	Adam	0.04	32	0.003	N/A	(0.9, 0.999)	TRUE	3784573425
version_37	Adam	0.002	24	0.5	N/A	(0.9, 0.999)	TRUE	692580778
version_39	Adam	1.00E-05	8	0.4	N/A	(0.9, 0.999)	FALSE	2361609063
version_40	Adam	0.005	40	0.1	N/A	(0.9, 0.999)	TRUE	1084308364
version_45	Adam	0.04	16	0.03	N/A	(0.9, 0.999)	TRUE	2304908219
version_47	Adam	0.002	8	0.005	N/A	(0.9, 0.999)	FALSE	1773918367
version_49	Adam	0.003	8	0.1	N/A	(0.9, 0.999)	FALSE	3859828295
version_0	AdamW	0.0001	40	0.3	TRUE	(0.9, 0.999)	FALSE	97973326
version_2	AdamW	0.0004	32	0.005	FALSE	(0.9, 0.999)	FALSE	1490073412
version_4	AdamW	0.003	32	0.004	FALSE	(0.9, 0.999)	FALSE	1621131566
version_5	AdamW	0.005	8	0.03	TRUE	(0.9, 0.999)	TRUE	3928351133
version_7	AdamW	0.001	24	0.5	FALSE	(0.9, 0.999)	TRUE	1142768951
version_11	AdamW	0.0005	8	0.005	TRUE	(0.9, 0.999)	TRUE	4178408584
version_14	AdamW	0.001	32	0.2	TRUE	(0.9, 0.999)	TRUE	2963527865
version_16	AdamW	3.00E-05	16	0.3	TRUE	(0.9, 0.999)	FALSE	1510376342
version_18	AdamW	0.004	8	0.5	FALSE	(0.9, 0.999)	TRUE	2801502967
version_19	AdamW	0.004	32	0.5	TRUE	(0.9, 0.999)	FALSE	103299523
version_24	AdamW	0.04	40	0.005	TRUE	(0.9, 0.999)	TRUE	2519141265
version_28	AdamW	0.005	40	0.3	FALSE	(0.9, 0.999)	TRUE	1638565519

version_32	AdamW	0.05	32	0.2	TRUE	(0.9, 0.999)	TRUE	3963989502
version_33	AdamW	3.00E-05	40	0.4	FALSE	(0.9, 0.999)	FALSE	4065806951
version_35	AdamW	0.05	32	0.5	FALSE	(0.9, 0.999)	TRUE	1365852761
version_38	AdamW	0.003	40	0.05	FALSE	(0.9, 0.999)	TRUE	2024838781

Table 13. Dataset 1: SegFormer Hyperparameter ordered by Optimizer selection.

Iteration:	Optimizer	LR	Batch Size:	Weight Decay Factor:	Nesterov	Momentum Factor	Freeze Encoder?	Seed
version_12	SGD	0.03	8	0.003	FALSE	0.02	TRUE	3714546793
version_18	SGD	2.00E-04	8	0.004	TRUE	0.7	TRUE	425845949
version_20	SGD	1.00E-05	16	0.04	FALSE	0.06	TRUE	2723712301
version_24	SGD	0.05	16	0.04	TRUE	0.05	TRUE	2167577817
version_27	SGD	0.001	8	0.05	TRUE	0.01	TRUE	3799370574
version_29	SGD	1.00E-04	8	0.4	TRUE	0.09	TRUE	3653064758
version_30	SGD	0.005	24	0.5	FALSE	0.06	TRUE	2331010606
version_32	SGD	4.00E-05	40	0.002	FALSE	0.4	TRUE	401343198
version_33	SGD	0.004	24	0.001	FALSE	0.8	TRUE	3040358993
version_35	SGD	0.0002	40	0.001	FALSE	0.09	TRUE	2627032494
version_36	SGD	0.005	16	0.05	TRUE	0.1	TRUE	2143498569
version_37	SGD	0.002	16	0.004	FALSE	0.08	TRUE	3156326446
version_44	SGD	0.004	24	0.01	FALSE	0.4	TRUE	957119004
version_45	SGD	1.00E-04	32	0.2	FALSE	0.05	TRUE	3066362343

version_46	SGD	0.04	40	0.002	TRUE	0.8	TRUE	1601834329
version_47	SGD	2.00E-05	24	0.001	FALSE	0.02	TRUE	3060679071
version_49	SGD	5.00E-05	16	0.01	TRUE	0.3	TRUE	2801392326
version_0	Adam	0.0001	24	0.004	N/A	(0.9, 0.999)	TRUE	3362641886
version_1	Adam	0.02	8	0.004	N/A	(0.9, 0.999)	TRUE	3446608861
version_2	Adam	0.002	8	0.02	N/A	(0.9, 0.999)	TRUE	599820371
version_3	Adam	0.004	32	0.01	N/A	(0.9, 0.999)	TRUE	601223770
version_5	Adam	0.003	24	0.3	N/A	(0.9, 0.999)	TRUE	3582987025
version_6	Adam	5.00E-05	32	0.04	N/A	(0.9, 0.999)	TRUE	3157492760
version_7	Adam	0.002	24	0.04	N/A	(0.9, 0.999)	TRUE	1278894072
version_13	Adam	0.0005	24	0.05	N/A	(0.9, 0.999)	TRUE	4118435223
version_15	Adam	2.00E-02	16	0.4	N/A	(0.9, 0.999)	TRUE	538650662
version_17	Adam	4.00E-02	32	0.04	N/A	(0.9, 0.999)	TRUE	934450018
version_21	Adam	0.002	8	0.01	N/A	(0.9, 0.999)	TRUE	3616333974
version_23	Adam	1.00E-05	40	0.02	N/A	(0.9, 0.999)	TRUE	1112660074
version_25	Adam	3.00E-02	40	0.3	N/A	(0.9, 0.999)	TRUE	2733042989
version_26	Adam	0.0004	8	0.002	N/A	(0.9, 0.999)	TRUE	1921641067
version_31	Adam	0.03	32	0.002	N/A	(0.9, 0.999)	TRUE	3704186556
version_38	Adam	0.003	40	0.005	N/A	(0.9, 0.999)	TRUE	395743910
version_39	Adam	0.0003	24	0.003	N/A	(0.9, 0.999)	TRUE	3113148355
version_4	AdamW	4.00E-05	16	0.05	FALSE	(0.9, 0.999)	TRUE	964732658
version_8	AdamW	0.02	8	0.005	TRUE	(0.9, 0.999)	TRUE	432505937

version_9	AdamW	0.02	40	0.05	TRUE	(0.9, 0.999)	TRUE	3774987763
version_10	AdamW	2.00E-05	24	0.1	TRUE	(0.9, 0.999)	TRUE	4278239771
version_11	AdamW	0.005	40	0.001	FALSE	(0.9, 0.999)	TRUE	1571787734
version_14	AdamW	0.02	24	0.4	FALSE	(0.9, 0.999)	TRUE	1540655208
version_16	AdamW	1.00E-05	24	0.04	FALSE	(0.9, 0.999)	TRUE	1753054773
version_19	AdamW	0.005	40	0.001	FALSE	(0.9, 0.999)	TRUE	664393850
version_22	AdamW	2.00E-04	32	0.2	TRUE	(0.9, 0.999)	TRUE	1549816816
version_28	AdamW	0.03	8	0.4	FALSE	(0.9, 0.999)	TRUE	709720899
version_34	AdamW	0.0005	24	0.005	TRUE	(0.9, 0.999)	TRUE	3219677645
version_40	AdamW	0.04	16	0.003	FALSE	(0.9, 0.999)	TRUE	3765516917
version_41	AdamW	0.01	24	0.4	FALSE	(0.9, 0.999)	TRUE	1702668217
version_42	AdamW	4.00E-05	16	0.02	FALSE	(0.9, 0.999)	TRUE	2439074284
version_43	AdamW	4.00E-02	24	0.01	TRUE	(0.9, 0.999)	TRUE	3411715659
version_48	AdamW	3.00E-05	40	0.05	TRUE	(0.9, 0.999)	TRUE	1355291291

Table 14 Dataset 2: MobileUNETR Hyperparameters ordered by Optimizer Selection.

Iteration:	Optimizer	LR	Batch Size:	Weight Decay Factor:	Nesterov	Momentum Factor	Freeze Encoder?	Seed
version_1	SGD	0.0002	40	0.3	FALSE	0.8	FALSE	1777070265
version_2	SGD	5.00E-05	40	0.1	TRUE	0.02	TRUE	3596209335

version_6	SGD	3.00E-05	40	0.003	TRUE	0.07	FALSE	2175016992
version_10	SGD	3.00E-05	32	0.004	FALSE	0.5	FALSE	1751483967
version_11	SGD	0.0002	16	0.04	FALSE	0.05	FALSE	524309863
version_13	SGD	0.002	16	0.5	FALSE	0.5	TRUE	4271059500
version_14	SGD	0.0001	40	0.04	FALSE	0.01	TRUE	1474219938
version_15	SGD	0.0002	40	0.01	TRUE	0.05	TRUE	3042752907
version_16	SGD	1.00E-05	8	0.3	TRUE	0.1	TRUE	4260259948
version_22	SGD	0.004	32	0.05	FALSE	0.03	FALSE	1546283536
version_23	SGD	4.00E-05	32	0.003	TRUE	0.5	FALSE	227155888
version_30	SGD	0.0003	16	0.05	FALSE	0.2	FALSE	3903231904
version_31	SGD	0.004	16	0.04	TRUE	0.06	FALSE	3392348771
version_33	SGD	0.0001	16	0.001	TRUE	0.8	FALSE	1807017945
version_49	SGD	0.005	16	0.2	FALSE	0.02	TRUE	3717993424
version_50	SGD	0.005	40	0.3	TRUE	0.08	FALSE	2362969148
version_0	Adam	0.05	24	0.01	N/A	(0.9, 0.999)	TRUE	1680022827
version_3	Adam	0.001	24	0.004	N/A	(0.9, 0.999)	FALSE	2648527343
version_4	Adam	0.0005	32	0.5	N/A	(0.9, 0.999)	TRUE	972487537
version_5	Adam	0.0002	16	0.5	N/A	(0.9, 0.999)	FALSE	2515534093
version_7	Adam	0.002	8	0.004	N/A	(0.9, 0.999)	TRUE	2730741455
version_8	Adam	0.04	32	0.4	N/A	(0.9, 0.999)	FALSE	3876333178
version_9	Adam	0.005	16	0.005	N/A	(0.9, 0.999)	TRUE	1343450799
version_19	Adam	0.001	24	0.001	N/A	(0.9, 0.999)	FALSE	1852036557
version_21	Adam	0.05	8	0.003	N/A	(0.9, 0.999)	TRUE	1237630745
version_25	Adam	0.0004	40	0.002	N/A	(0.9, 0.999)	FALSE	1115340519
version_28	Adam	5.00E-05	24	0.4	N/A	(0.9, 0.999)	TRUE	236016728

version_29	Adam	0.001	40	0.1	N/A	(0.9, 0.999)	TRUE	3606381444
version_34	Adam	0.01	24	0.3	N/A	(0.9, 0.999)	TRUE	2014246102
version_36	Adam	0.0002	24	0.03	N/A	(0.9, 0.999)	TRUE	1364399902
version_37	Adam	0.0003	8	0.05	N/A	(0.9, 0.999)	FALSE	3261081483
version_39	Adam	0.0001	16	0.003	N/A	(0.9, 0.999)	TRUE	2614211096
version_40	Adam	0.0003	40	0.003	N/A	(0.9, 0.999)	TRUE	1310053805
version_41	Adam	0.005	32	0.002	N/A	(0.9, 0.999)	TRUE	1317789841
version_46	Adam	0.003	40	0.003	N/A	(0.9, 0.999)	TRUE	2047136179
version_47	Adam	0.02	8	0.2	N/A	(0.9, 0.999)	FALSE	1508640104
version_12	AdamW	0.02	8	0.05	FALSE	(0.9, 0.999)	FALSE	464257913
version_17	AdamW	0.005	16	0.05	FALSE	(0.9, 0.999)	TRUE	1137996605
version_18	AdamW	0.005	32	0.03	FALSE	(0.9, 0.999)	TRUE	2399454444
version_20	AdamW	0.0001	32	0.005	FALSE	(0.9, 0.999)	TRUE	288466879
version_24	AdamW	4.00E-05	16	0.004	FALSE	(0.9, 0.999)	TRUE	602399376
version_26	AdamW	0.03	16	0.005	TRUE	(0.9, 0.999)	FALSE	9553094
version_27	AdamW	3.00E-05	40	0.4	FALSE	(0.9, 0.999)	FALSE	3141439327
version_32	AdamW	0.0004	8	0.004	TRUE	(0.9, 0.999)	TRUE	1139574530
version_35	AdamW	0.0005	40	0.3	TRUE	(0.9, 0.999)	FALSE	451007880
version_38	AdamW	0.04	32	0.05	FALSE	(0.9, 0.999)	FALSE	3146342530
version_42	AdamW	0.0001	16	0.02	FALSE	(0.9, 0.999)	FALSE	1621525216
version_43	AdamW	0.0002	8	0.03	FALSE	(0.9, 0.999)	TRUE	3303522495
version_44	AdamW	0.03	8	0.002	TRUE	(0.9, 0.999)	TRUE	2665487820
version_45	AdamW	0.03	24	0.4	FALSE	(0.9, 0.999)	FALSE	3385543524

Table 15. Dataset 2: SegFormer Hyperparameters Ordered by Optimizer Selection.

Iteration:	Optimizer	LR	Batch Size:	Weight Decay Factor:	Nesterov	Momentum Factor	Freeze Encoder?	Seed
version_1	SGD	2.00E-05	32	4.00E-01	FALSE	0.2	TRUE	787794450
version_5	SGD	0.0004	8	0.04	FALSE	0.07	TRUE	1732410814
version_7	SGD	1.00E-05	8	0.05	TRUE	0.4	TRUE	397804987
version_8	SGD	0.0005	16	0.001	FALSE	0.1	TRUE	4154860022
version_9	SGD	0.005	40	0.004	TRUE	0.6	TRUE	1751009286
version_12	SGD	0.0002	40	3.00E-02	FALSE	0.9	TRUE	2214140210
version_14	SGD	4.00E-05	40	2.00E-03	FALSE	0.4	TRUE	2071859585
version_15	SGD	0.02	32	0.1	FALSE	0.05	TRUE	1514753217
version_18	SGD	0.04	24	0.02	FALSE	0.05	TRUE	2469923986
version_19	SGD	5.00E-05	40	0.004	FALSE	0.6	TRUE	1216127117
version_22	SGD	0.005	16	0.2	TRUE	0.09	TRUE	1252123789
version_24	SGD	0.01	32	0.4	TRUE	0.8	TRUE	1530868984
version_28	SGD	0.03	16	0.04	TRUE	0.01	TRUE	355730456
version_30	SGD	0.001	40	0.003	FALSE	0.01	TRUE	2512377065
version_34	SGD	0.02	16	2.00E-01	TRUE	0.05	TRUE	1277187661
version_35	SGD	0.03	40	0.005	FALSE	0.05	TRUE	1035218642
version_43	SGD	0.02	8	0.03	FALSE	0.8	TRUE	475011007
version_45	SGD	0.05	32	0.002	FALSE	0.07	TRUE	647318767
version_47	SGD	0.03	32	0.001	FALSE	0.07	TRUE	90412784
version_48	SGD	0.01	32	0.3	TRUE	0.04	TRUE	515870984

version_0	Adam	0.002	40	2.00E-03	N/A	(0.9, 0.999)	TRUE	754045431
version_2	Adam	2.00E-05	40	5.00E-02	N/A	(0.9, 0.999)	TRUE	3170888410
version_3	Adam	0.0001	16	5.00E-01	N/A	(0.9, 0.999)	TRUE	983859802
version_4	Adam	0.004	8	0.001	N/A	(0.9, 0.999)	TRUE	1549074188
version_11	Adam	0.0001	16	0.005	N/A	(0.9, 0.999)	TRUE	491506924
version_16	Adam	0.0001	8	0.2	N/A	(0.9, 0.999)	TRUE	1476711029
version_21	Adam	0.0001	8	1.00E-03	N/A	(0.9, 0.999)	TRUE	3322584895
version_25	Adam	0.0004	24	0.03	N/A	(0.9, 0.999)	TRUE	2526104304
version_27	Adam	5.00E-05	32	0.2	N/A	(0.9, 0.999)	TRUE	2081454612
version_32	Adam	0.005	24	3.00E-03	N/A	(0.9, 0.999)	TRUE	737713687
version_33	Adam	3.00E-05	16	3.00E-02	N/A	(0.9, 0.999)	TRUE	3540922163
version_36	Adam	1.00E-05	32	4.00E-01	N/A	(0.9, 0.999)	TRUE	1518456478
version_38	Adam	4.00E-05	16	0.002	N/A	(0.9, 0.999)	TRUE	2402859719
version_42	Adam	0.001	16	0.5	N/A	(0.9, 0.999)	TRUE	3280269146
version_44	Adam	0.05	24	0.005	N/A	(0.9, 0.999)	TRUE	1001397974
version_46	Adam	0.003	16	0.02	N/A	(0.9, 0.999)	TRUE	2349337237
version_6	AdamW	0.001	40	0.001	FALSE	(0.9, 0.999)	TRUE	2783125841
version_10	AdamW	0.02	16	0.3	TRUE	(0.9, 0.999)	TRUE	2864822261
version_13	AdamW	0.0001	40	0.005	TRUE	(0.9, 0.999)	TRUE	3135574020
version_17	AdamW	0.01	40	0.003	TRUE	(0.9, 0.999)	TRUE	2319272921
version_20	AdamW	0.03	32	0.01	FALSE	(0.9, 0.999)	TRUE	851508180

version_23	AdamW	0.02	40	0.005	FALSE	(0.9, 0.999)	TRUE	3024383362
version_26	AdamW	0.01	40	0.5	TRUE	(0.9, 0.999)	TRUE	3845877712
version_29	AdamW	0.0004	8	0.03	TRUE	(0.9, 0.999)	TRUE	2798714768
version_31	AdamW	0.0005	8	0.002	TRUE	(0.9, 0.999)	TRUE	1837243233
version_37	AdamW	0.01	32	0.5	FALSE	(0.9, 0.999)	TRUE	1873367680
version_39	AdamW	2.00E-05	16	0.03	FALSE	(0.9, 0.999)	TRUE	2378136884
version_40	AdamW	0.003	8	0.5	FALSE	(0.9, 0.999)	TRUE	1566012393
version_41	AdamW	0.0001	32	0.04	FALSE	(0.9, 0.999)	TRUE	2786452079
version_49	AdamW	0.05	24	0.4	FALSE	(0.9, 0.999)	TRUE	332849507

Table 16. Dataset 3: MobileUNETR Hyperparameters Ordered by Optimizer Selection

Iteration:	Optimizer	LR	Batch Size:	Weight Decay Factor:	Nesterov	Momentum Factor	Freeze Encoder?	Seed
version_0	SGD	4.00E-05	16	0.002	FALSE	0.08	TRUE	2782496948
version_5	SGD	3.00E-05	32	0.4	TRUE	0.08	FALSE	1802987551
version_13	SGD	0.004	40	0.2	TRUE	0.2	TRUE	3173483391
version_15	SGD	0.005	8	0.03	TRUE	0.7	FALSE	628167708
version_17	SGD	3.00E-03	24	0.3	FALSE	0.07	FALSE	2552794548
version_18	SGD	0.0002	32	0.003	TRUE	0.6	FALSE	2952590435
version_20	SGD	0.01	32	0.005	FALSE	0.09	FALSE	1064091531
version_22	SGD	3.00E-05	40	0.5	TRUE	0.01	TRUE	4049240303
version_25	SGD	0.05	40	0.1	TRUE	0.1	FALSE	2129607382
version_26	SGD	0.05	32	0.4	FALSE	0.6	TRUE	1793424636

version_27	SGD	0.04	24	0.3	FALSE	0.7	FALSE	3467204965
version_31	SGD	0.0001	16	0.05	FALSE	0.03	FALSE	1154221992
version_37	SGD	0.0003	8	0.03	TRUE	0.06	FALSE	2249629608
version_39	SGD	4.00E-04	24	0.01	TRUE	0.07	TRUE	2084475041
version_40	SGD	1.00E-04	40	0.01	FALSE	0.05	FALSE	1867221450
version_45	SGD	0.03	8	0.5	TRUE	0.03	FALSE	2797175026
version_48	SGD	0.0004	8	0.1	FALSE	0.2	FALSE	1082542733
version_50	SGD	0.003	32	0.05	TRUE	0.06	FALSE	1453271064
version_53	SGD	2.00E-05	32	0.03	TRUE	0.09	FALSE	1940813984
version_54	SGD	0.0002	40	0.02	FALSE	0.1	TRUE	273298920
version_55	SGD	4.00E-05	24	0.4	TRUE	0.05	TRUE	3058649352
version_57	SGD	3.00E-05	8	0.02	FALSE	0.5	TRUE	307037875
version_59	SGD	5.00E-03	32	0.05	TRUE	0.4	FALSE	1834005098
version_60	SGD	0.0001	16	0.4	TRUE	0.2	TRUE	3474395336
version_64	SGD	2.00E-05	8	0.03	FALSE	0.06	FALSE	1059924335
version_66	SGD	0.0001	8	0.2	FALSE	0.08	TRUE	1042083092
version_70	SGD	0.02	40	0.03	TRUE	0.1	FALSE	3778441935
version_74	SGD	3.00E-02	16	0.5	TRUE	0.3	TRUE	885322843
version_4	Adam	0.0005	24	0.03	N/A	(0.9, 0.999)	TRUE	867516631
version_7	Adam	0.04	32	0.005	N/A	(0.9, 0.999)	FALSE	4141985389
version_9	Adam	0.03	8	0.1	N/A	(0.9, 0.999)	TRUE	784292206

version_12	Adam	1.00E-05	16	0.004	N/A	(0.9, 0.999)	FALSE	4265527806
version_16	Adam	4.00E-05	24	0.5	N/A	(0.9, 0.999)	FALSE	922905827
version_19	Adam	0.005	32	0.004	N/A	(0.9, 0.999)	TRUE	2935930588
version_21	Adam	0.0002	40	0.005	N/A	(0.9, 0.999)	TRUE	163064955
version_24	Adam	0.002	40	0.01	N/A	(0.9, 0.999)	FALSE	1067877200
version_29	Adam	0.01	24	0.004	N/A	(0.9, 0.999)	TRUE	709818403
version_30	Adam	0.0001	16	0.03	N/A	(0.9, 0.999)	FALSE	3604932256
version_35	Adam	0.05	32	0.003	N/A	(0.9, 0.999)	FALSE	4209568390
version_36	Adam	0.02	8	0.004	N/A	(0.9, 0.999)	TRUE	324536145
version_38	Adam	0.0001	16	0.03	N/A	(0.9, 0.999)	TRUE	1550648931
version_42	Adam	0.002	16	0.001	N/A	(0.9, 0.999)	TRUE	1964656455
version_43	Adam	0.0005	16	0.004	N/A	(0.9, 0.999)	FALSE	1029827231
version_44	Adam	0.02	8	0.04	N/A	(0.9, 0.999)	TRUE	242493103
version_46	Adam	0.001	40	0.3	N/A	(0.9, 0.999)	FALSE	4015442301
version_47	Adam	0.0001	40	0.002	N/A	(0.9, 0.999)	FALSE	1448052593
version_49	Adam	0.0004	40	0.05	N/A	(0.9, 0.999)	FALSE	287256953
version_58	Adam	0.0003	16	0.05	N/A	(0.9, 0.999)	TRUE	4030662200
version_63	Adam	2.00E-05	16	0.1	N/A	(0.9, 0.999)	TRUE	2068519396
version_65	Adam	0.01	16	0.1	N/A	(0.9, 0.999)	TRUE	4155682570
version_68	Adam	0.04	8	0.002	N/A	(0.9, 0.999)	TRUE	3848364076
version_71	Adam	3.00E-05	32	0.2	N/A	(0.9, 0.999)	FALSE	1080195582
version_75	Adam	0.001	16	0.003	N/A	(0.9, 0.999)	FALSE	1245240297
version_1	AdamW	3.00E-05	40	0.001	FALSE	(0.9, 0.999)	FALSE	2264656384
version_2	AdamW	0.0002	24	0.3	TRUE	(0.9, 0.999)	FALSE	2813218568
version_3	AdamW	0.0001	8	0.04	FALSE	(0.9, 0.999)	TRUE	4093846572

version_6	AdamW	0.004	32	0.005	FALSE	(0.9, 0.999)	TRUE	3768272870
version_8	AdamW	0.0001	40	0.03	FALSE	(0.9, 0.999)	TRUE	2592434236
version_10	AdamW	0.0001	8	0.002	TRUE	(0.9, 0.999)	FALSE	50991077
version_11	AdamW	0.0003	16	0.5	TRUE	(0.9, 0.999)	TRUE	3463259066
version_14	AdamW	0.001	32	0.04	FALSE	(0.9, 0.999)	TRUE	2545580629
version_23	AdamW	2.00E-05	8	0.02	TRUE	(0.9, 0.999)	TRUE	1071854699
version_28	AdamW	0.04	16	0.04	TRUE	(0.9, 0.999)	TRUE	1947833320
version_32	AdamW	5.00E-05	40	0.5	FALSE	(0.9, 0.999)	TRUE	874838127
version_33	AdamW	0.03	8	0.003	TRUE	(0.9, 0.999)	TRUE	1629626414
version_34	AdamW	0.0004	40	0.2	FALSE	(0.9, 0.999)	FALSE	2771647456
version_41	AdamW	1.00E-05	8	0.03	FALSE	(0.9, 0.999)	TRUE	2866711065
version_51	AdamW	0.01	16	0.3	FALSE	(0.9, 0.999)	TRUE	2647820512
version_52	AdamW	0.05	8	0.002	FALSE	(0.9, 0.999)	TRUE	1165501003
version_56	AdamW	0.004	8	0.1	FALSE	(0.9, 0.999)	FALSE	3172194916
version_61	AdamW	0.001	8	0.3	TRUE	(0.9, 0.999)	FALSE	776316234
version_62	AdamW	5.00E-05	8	0.5	TRUE	(0.9, 0.999)	TRUE	2632750570
version_67	AdamW	0.0001	40	0.01	TRUE	(0.9, 0.999)	TRUE	2938866002
version_69	AdamW	0.002	16	0.05	TRUE	(0.9, 0.999)	FALSE	2547639868
version_72	AdamW	0.0005	16	0.5	FALSE	(0.9, 0.999)	FALSE	4218538598
version_73	AdamW	5.00E-05	32	0.003	FALSE	(0.9, 0.999)	FALSE	2071533160
version_76	AdamW	4.00E-05	24	0.4	TRUE	(0.9, 0.999)	TRUE	2380793403
version_77	AdamW	0.001	16	0.4	FALSE	(0.9, 0.999)	FALSE	3548838933

Table 17. Dataset 3: SegFormer Hyperparameters Ordered by Optimizer Selection

Iteration:	Optimizer	LR	Batch Size:	Weight Decay Factor:	Nesterov	Momentum Factor	Freeze Encoder?	Seed
version_0	SGD	5.00E-04	16	0.2	FALSE	0.01	TRUE	3140189307
version_3	SGD	3.00E-02	16	0.1	TRUE	0.3	TRUE	4239850451
version_5	SGD	0.005	24	0.003	FALSE	0.01	TRUE	4061407919
version_6	SGD	0.01	40	0.01	FALSE	0.2	TRUE	3953725560
version_7	SGD	5.00E-05	32	0.5	FALSE	0.07	TRUE	1897067294
version_8	SGD	4.00E-05	40	0.01	TRUE	0.06	TRUE	2435029235
version_12	SGD	0.0002	8	0.003	FALSE	0.06	TRUE	3620805683
version_16	SGD	5.00E-03	16	0.04	FALSE	0.9	TRUE	2644357332
version_22	SGD	0.05	8	0.3	TRUE	0.2	TRUE	237034694
version_24	SGD	5.00E-05	40	0.005	TRUE	0.3	TRUE	1778613001
version_25	SGD	0.03	32	0.01	TRUE	0.06	TRUE	351795201
version_32	SGD	0.0003	24	0.005	FALSE	0.03	TRUE	1055911872
version_41	SGD	0.0001	24	0.05	FALSE	0.08	TRUE	1541850331
version_50	SGD	5.00E-05	32	0.03	TRUE	0.3	TRUE	955386942
version_54	SGD	2.00E-05	24	0.001	FALSE	0.05	TRUE	2800142668
version_56	SGD	0.003	16	0.01	FALSE	0.6	TRUE	159123353
version_62	SGD	0.0001	40	0.2	FALSE	0.08	TRUE	1560322196
version_64	SGD	0.0002	32	0.3	TRUE	0.2	TRUE	1696226329

version_70	SGD	4.00E-05	32	0.2	FALSE	0.05	TRUE	829914494
version_71	SGD	0.02	32	0.001	TRUE	0.8	TRUE	732471529
version_79	SGD	4.00E-04	16	0.002	TRUE	0.8	TRUE	4129678827
version_80	SGD	5.00E-05	24	0.2	FALSE	0.05	TRUE	4262897179
version_82	SGD	3.00E-03	16	0.4	TRUE	0.3	TRUE	1064332127
version_86	SGD	0.005	16	0.02	TRUE	0.7	TRUE	31607705
version_88	SGD	3.00E-02	16	0.002	FALSE	0.6	TRUE	643357389
version_96	SGD	0.05	40	0.01	TRUE	0.8	TRUE	2054474794
version_98	SGD	0.0005	16	0.003	TRUE	0.07	TRUE	3628515121
version_2	Adam	0.0002	24	0.02	N/A	(0.9, 0.999)	TRUE	4006724516
version_4	Adam	0.003	40	0.004	N/A	(0.9, 0.999)	TRUE	3596820585
version_10	Adam	0.002	16	0.02	N/A	(0.9, 0.999)	TRUE	1493427740
version_17	Adam	2.00E-04	24	0.04	N/A	(0.9, 0.999)	TRUE	638895325
version_26	Adam	5.00E-02	24	0.002	N/A	(0.9, 0.999)	TRUE	4209627283
version_29	Adam	0.0005	8	0.003	N/A	(0.9, 0.999)	TRUE	4091092866
version_30	Adam	4.00E-05	24	0.002	N/A	(0.9, 0.999)	TRUE	3848867657
version_31	Adam	0.003	24	0.05	N/A	(0.9, 0.999)	TRUE	3705645761
version_34	Adam	0.004	32	0.002	N/A	(0.9, 0.999)	TRUE	1443716070
version_36	Adam	0.002	24	0.5	N/A	(0.9, 0.999)	TRUE	135707683
version_37	Adam	0.02	24	0.004	N/A	(0.9, 0.999)	TRUE	4257553677
version_39	Adam	1.00E-05	16	0.003	N/A	(0.9, 0.999)	TRUE	1954031314

version_43	Adam	0.0002	16	0.003	N/A	(0.9, 0.999)	TRUE	3765555370
version_46	Adam	3.00E-05	24	0.05	N/A	(0.9, 0.999)	TRUE	3532680473
version_48	Adam	0.01	40	0.02	N/A	(0.9, 0.999)	TRUE	200462574
version_49	Adam	0.0002	32	0.1	N/A	(0.9, 0.999)	TRUE	2727699753
version_51	Adam	1.00E-05	24	0.04	N/A	(0.9, 0.999)	TRUE	167398425
version_53	Adam	0.002	8	0.002	N/A	(0.9, 0.999)	TRUE	1400211304
version_55	Adam	0.03	40	0.4	N/A	(0.9, 0.999)	TRUE	1068552183
version_57	Adam	0.001	16	0.5	N/A	(0.9, 0.999)	TRUE	3701337400
version_58	Adam	3.00E-03	32	0.003	N/A	(0.9, 0.999)	TRUE	684288195
version_61	Adam	0.003	24	0.01	N/A	(0.9, 0.999)	TRUE	1613857457
version_66	Adam	0.0001	24	0.003	N/A	(0.9, 0.999)	TRUE	1472613902
version_73	Adam	4.00E-05	8	0.03	N/A	(0.9, 0.999)	TRUE	3269252347
version_74	Adam	0.04	16	0.005	N/A	(0.9, 0.999)	TRUE	375055354
version_75	Adam	5.00E-02	8	0.002	N/A	(0.9, 0.999)	TRUE	1224517405
version_77	Adam	0.0001	32	0.002	N/A	(0.9, 0.999)	TRUE	3609968529
version_78	Adam	1.00E-05	8	0.2	N/A	(0.9, 0.999)	TRUE	2079001186
version_83	Adam	0.004	32	0.003	N/A	(0.9, 0.999)	TRUE	2751815540
version_84	Adam	5.00E-05	40	0.3	N/A	(0.9, 0.999)	TRUE	4061842033
version_85	Adam	4.00E-05	32	0.03	N/A	(0.9, 0.999)	TRUE	3093927295
version_87	Adam	0.003	24	0.005	N/A	(0.9, 0.999)	TRUE	4058218124
version_89	Adam	0.0004	40	0.02	N/A	(0.9, 0.999)	TRUE	65131522

version_92	Adam	3.00E-04	24	0.5	N/A	(0.9, 0.999)	TRUE	2494958661
version_97	Adam	0.005	32	0.05	N/A	(0.9, 0.999)	TRUE	3294218344
version_1	AdamW	2.00E-03	32	0.4	FALSE	(0.9, 0.999)	TRUE	1802528677
version_9	AdamW	3.00E-05	32	0.4	FALSE	(0.9, 0.999)	TRUE	1681680122
version_11	AdamW	2.00E-05	40	0.3	FALSE	(0.9, 0.999)	TRUE	1124029776
version_13	AdamW	4.00E-03	8	0.02	TRUE	(0.9, 0.999)	TRUE	2117194391
version_14	AdamW	2.00E-05	8	0.4	TRUE	(0.9, 0.999)	TRUE	1186657557
version_15	AdamW	0.02	16	0.02	FALSE	(0.9, 0.999)	TRUE	1069946188
version_18	AdamW	4.00E-05	24	0.5	TRUE	(0.9, 0.999)	TRUE	295193393
version_19	AdamW	0.01	16	0.001	TRUE	(0.9, 0.999)	TRUE	4109879362
version_20	AdamW	5.00E-05	24	0.5	TRUE	(0.9, 0.999)	TRUE	501640473
version_21	AdamW	4.00E-05	40	0.2	FALSE	(0.9, 0.999)	TRUE	48703242
version_23	AdamW	0.0001	16	0.04	TRUE	(0.9, 0.999)	TRUE	957335145
version_27	AdamW	0.01	32	0.02	FALSE	(0.9, 0.999)	TRUE	3768529250
version_28	AdamW	1.00E-05	40	0.005	FALSE	(0.9, 0.999)	TRUE	1107277637
version_33	AdamW	5.00E-03	16	0.003	FALSE	(0.9, 0.999)	TRUE	446637793
version_35	AdamW	1.00E-05	16	0.004	TRUE	(0.9, 0.999)	TRUE	3156726416
version_38	AdamW	0.0004	16	0.05	FALSE	(0.9, 0.999)	TRUE	3637264974

version_40	AdamW	5.00E-05	32	0.03	FALSE	(0.9, 0.999)	TRUE	3208168833
version_42	AdamW	0.003	16	0.3	FALSE	(0.9, 0.999)	TRUE	496184434
version_44	AdamW	4.00E-05	24	0.1	FALSE	(0.9, 0.999)	TRUE	4159862951
version_45	AdamW	0.001	32	0.04	FALSE	(0.9, 0.999)	TRUE	2396974210
version_47	AdamW	0.0005	32	0.003	FALSE	(0.9, 0.999)	TRUE	4268741770
version_52	AdamW	0.04	40	0.02	FALSE	(0.9, 0.999)	TRUE	3146004101
version_59	AdamW	1.00E-05	16	0.2	TRUE	(0.9, 0.999)	TRUE	3134590206
version_60	AdamW	0.03	16	0.05	TRUE	(0.9, 0.999)	TRUE	1323816906
version_63	AdamW	0.001	24	0.2	FALSE	(0.9, 0.999)	TRUE	3167875963
version_65	AdamW	4.00E-05	40	0.002	TRUE	(0.9, 0.999)	TRUE	2330424409
version_67	AdamW	0.02	16	0.001	TRUE	(0.9, 0.999)	TRUE	2485586391
version_68	AdamW	0.001	32	0.2	FALSE	(0.9, 0.999)	TRUE	4039316507
version_69	AdamW	1.00E-05	8	0.05	TRUE	(0.9, 0.999)	TRUE	1999384030
version_72	AdamW	1.00E-05	32	0.003	TRUE	(0.9, 0.999)	TRUE	1739813791
version_76	AdamW	4.00E-05	24	0.4	TRUE	(0.9, 0.999)	TRUE	932830465
version_81	AdamW	0.04	24	0.004	FALSE	(0.9, 0.999)	TRUE	944419080
version_90	AdamW	0.0004	8	0.2	FALSE	(0.9, 0.999)	TRUE	2363435643
version_91	AdamW	0.02	8	0.004	FALSE	(0.9, 0.999)	TRUE	3931919808
version_93	AdamW	0.03	32	0.03	FALSE	(0.9, 0.999)	TRUE	183737174
version_94	AdamW	0.03	40	0.1	TRUE	(0.9, 0.999)	TRUE	2035037744
version_95	AdamW	0.004	8	0.004	TRUE	(0.9, 0.999)	TRUE	2136008925
version_99	AdamW	0.005	32	0.3	TRUE	(0.9, 0.999)	TRUE	3391257295